Dynamic Resource Management Algorithms for Complex Systems and Novel Approaches to Adaptive Kalman Filtering

Lingyi Zhang
University of Connecticut - Storrs, lingyi.zhang@uconn.edu

Follow this and additional works at: https://opencommons.uconn.edu/dissertations

Recommended Citation
https://opencommons.uconn.edu/dissertations/2564
Dynamic Resource Management
Algorithms for Complex Systems and
Novel Approaches to Adaptive Kalman
Filtering

Lingyi Zhang, Ph.D.
University of Connecticut, 2020

ABSTRACT

This thesis considers three combinatorial optimization problems of substantial practical
importance. First, a new approach to efficiently obtain a large number of ranked solu-
tions to a 3-dimensional assignment problem is presented, and is applied to generate
fuel assembly loading patterns. Second, we formulate the problem of dynamically
scheduling maritime surveillance assets, and solve it using branch-and-cut and approxi-
mate dynamic programming (ADP) with rollout, and investigate the tradeoffs between
the two. Third, a multi-objective ship routing problem is also investigated, where
we propose a solution combining approximate dynamic programming techniques and
clustering techniques to contain the computational and storage complexity. Lastly,
this dissertation develops a seminal approach to adaptive Kalman filtering via the use
of post-fit residuals given data samples – an approach not yet discussed prior to this
work.
Dynamic Resource Management
Algorithms for Complex Systems and
Novel Approaches to Adaptive Kalman
Filtering

Lingyi Zhang

M.S., University of Connecticut, 2019
B.S., University of Connecticut, 2014

A Dissertation
Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy
at the
University of Connecticut

2020
Dynamic Resource Management Algorithms for Complex Systems and Novel Approaches to Adaptive Kalman Filtering

Presented by
Lingyi Zhang, B.S., M.S.

Major Advisor
Krishna R. Pattipati

Associate Advisor
Peter B. Luh

Associate Advisor
Yaakov Bar-Shalom

University of Connecticut
2020
Thank you to my major advisor, Dr. Krishna Pattipati, for his guidance and patience in molding me into who I am today. It is my utmost honor to work and learn under his guidance and support. I would like to thank my associate advisor Dr. Yaakov Bar-Shalom, whom I had the pleasure of writing a paper with. He has shaped my presentation and writing style to be consistent and detail oriented. I also thank Dr. Peter Luh for being on my committee, and whom I had the privilege of being a student in his nonlinear optimization course.

I would like to express my appreciation for my friends/colleagues, David Sidoti, Manisha Mishra, Vinod Avvari, Adam Bienkowski and the rest of the Cyberlab members I had the honor to work with over the years in pursuit of this degree. I would like to also particularly thank David Sidoti for being my mentor and aiding my transition to the lab, providing me with valuable support, advice and guidance throughout the years of my graduate school journey. Lastly, I want to thank my family for their unconditional encouragement, support and love. I would not have made it this far without their support.
# Contents

1 Introduction  
  1.1 Background ................................. 1  
  1.2 Outline of the Dissertation ............... 3  
  1.3 Publications ................................ 6  

2 Approaches to Obtain a Large Number of Ranked Solutions to 3-Dimensional Assignment Problems  
  2.1 Introduction .................................. 10  
    2.1.1 Motivation ............................... 10  
    2.1.2 Related Research ....................... 13  
    2.1.3 Chapter Organization ..................... 15  
  2.2 Problem Formulation .......................... 17  
    2.2.1 Problem Formulation .................... 18  
    2.2.2 Special Cases ......................... 19  
  2.3 Solution Approach ............................ 20  
    2.3.1 3-D Assignment Relaxations ............ 21  
    2.3.2 \( m \)-best 3-D Assignment ............ 27  
  2.4 Optimized implementation of Murty’s search space decomposition ... 31  
    2.4.1 Inheriting dual variables and partial solutions during partitioning 31  
    2.4.2 Sorting subproblems via an upper bound .................. 32  
    2.4.3 Partition in an optimized order .......... 34  
  2.5 Pseudocode .................................. 34  
  2.6 Results ....................................... 35  
    2.6.1 Relaxation Method I vs. Relaxation Method II .. 36  
    2.6.2 JVC vs. Auction Algorithm ............... 39  
    2.6.3 Transportation vs. Transauction vs. RELAX-IV Algorithm ... 40  
    2.6.4 Solution quality evaluation for decomposition methods .... 42
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6.5 Runtime comparison for decomposition methods</td>
<td>43</td>
</tr>
<tr>
<td>2.6.6 Scalability with N</td>
<td>44</td>
</tr>
<tr>
<td>2.6.7 Scalability with R</td>
<td>47</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>53</td>
</tr>
<tr>
<td>3.1.1 Motivation</td>
<td>53</td>
</tr>
<tr>
<td>3.1.2 Related Research</td>
<td>56</td>
</tr>
<tr>
<td>3.1.3 Chapter Organization</td>
<td>59</td>
</tr>
<tr>
<td>3.2 Problem Model and Formulation</td>
<td>60</td>
</tr>
<tr>
<td>3.2.1 Problem Definition and Solution Architecture</td>
<td>60</td>
</tr>
<tr>
<td>3.2.2 Problem Formulation</td>
<td>62</td>
</tr>
<tr>
<td>3.3 Solution Approach</td>
<td>70</td>
</tr>
<tr>
<td>3.3.1 Exhaustive Branch-and-Cut</td>
<td>70</td>
</tr>
<tr>
<td>3.3.2 Greedy Branch-and-Cut I</td>
<td>71</td>
</tr>
<tr>
<td>3.3.3 Greedy Branch-and-Cut II</td>
<td>72</td>
</tr>
<tr>
<td>3.3.4 Parallelized Greedy Branch-and-Cut II</td>
<td>73</td>
</tr>
<tr>
<td>3.3.5 Approximate Dynamic Programming (ADP)</td>
<td>74</td>
</tr>
<tr>
<td>3.3.6 Multi-Step Lookahead Approximate Dynamic Programming-I</td>
<td>78</td>
</tr>
<tr>
<td>3.3.7 Multi-Step Lookahead Approximate Dynamic Programming-II</td>
<td>79</td>
</tr>
<tr>
<td>3.4 Simulation and Computational Results</td>
<td>81</td>
</tr>
<tr>
<td>3.4.1 Scenario Description(s)</td>
<td>82</td>
</tr>
<tr>
<td>3.4.2 Solution Quality with Different Objective Function</td>
<td>85</td>
</tr>
<tr>
<td>3.4.3 Scalability: Available Asset Sensitivity</td>
<td>89</td>
</tr>
<tr>
<td>3.4.4 Scalability: Varying the Number of Cases</td>
<td>92</td>
</tr>
<tr>
<td>3.4.5 Robustness: Monte Carlo Evaluation of Asset Allocation Strategies</td>
<td>96</td>
</tr>
<tr>
<td>4. Many-Objective Maritime Path Planning for Dynamic and Uncertain Environments</td>
<td>99</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>99</td>
</tr>
<tr>
<td>4.1.1 Motivation</td>
<td>99</td>
</tr>
<tr>
<td>4.1.2 Related Research</td>
<td>101</td>
</tr>
<tr>
<td>4.1.3 Chapter Organization</td>
<td>103</td>
</tr>
<tr>
<td>4.2 Environmental Setup and Modeling</td>
<td>103</td>
</tr>
<tr>
<td>4.2.1 Environmental Impact</td>
<td>103</td>
</tr>
<tr>
<td>4.2.2 Fuel Cost Calculation</td>
<td>104</td>
</tr>
<tr>
<td>4.2.3 Problem Setup</td>
<td>105</td>
</tr>
</tbody>
</table>
5.10.1 Case 1 ................................................................. 166
5.10.2 Case 2 ................................................................. 171
5.10.3 Case 3 ................................................................. 174
5.10.4 Case 4 ................................................................. 180
5.10.5 Case 5 ................................................................. 185

6 Conclusion ................................................................. 189

Appendix A Kalman Filter Derivations ................................. 194
A.1 Steady-State Updated State Covariance Riccati Equation .... 194
A.2 Proof of Necessary and Sufficient Condition for Identifiability of Un-
known Covariances ........................................................... 195
A.3 Proof of Estimability of $R$ ............................................. 196
A.4 Procedure to Obtain $W$ using minimal polynomial ............ 197
A.5 Objective Function Gradient Computation ...................... 200
A.6 Cholesky Decomposition and Eigen Decomposition .......... 203
A.7 Simultaneous Diagonalization ...................................... 204

Bibliography ................................................................... 206
Chapter 1

Introduction

1.1 Background

This dissertation considers two broad topics, one motivated by the need to comply with scarce resource management requirements as defense and industry look to continuously accomplish more with less, and the other motivated by the previous limitations of a steady-state data-driven Kalman filter. The first topic led to the development of efficient dynamic resource management algorithms with applications to nuclear fuel assembly loading pattern optimization, surveillance asset allocation for counter-drug smuggling and multi-objective ship routing, while the second topic resulted in novel approaches for estimating process and measurement noise covariances in adaptive Kalman filtering.

The goal of automated decision making is to determine and understand the decision context, and to effectively explore the problem space to present to the Decision Maker (DM) ranked courses of action to choose from in a timely manner. For example, what
separates the nuclear fuel assembly loading problem from a traditional 3-dimensional (3-D) assignment problem is the requirement to enumerate a dense set of discrete loading patterns though a dynamically estimated probability distribution (represented by a reward tensor). Evaluation of each loading pattern by reactor-physics-based external code may be very time consuming (≈ 0.1 to 10 minutes, depending on the required accuracy of loading pattern response evaluation). The key here is to evaluate only new (unique) loading patterns (assignments). In application to maritime surveillance and drug interdiction, the dynamic resource management problem under uncertainty may be viewed as a moving horizon stochastic control problem. In the context of a counter-smuggling mission, the key problem is to efficiently allocate a set of heterogeneous sensing and interdiction assets to maximize the probability of smuggler detection and interdiction, subject to mission constraints, by integrating information, such as intelligence, weather, asset availability, asset capabilities (e.g., range, speed), sensor management, and asset assignment (e.g., many sensors may need to be coordinated to obtain a better picture of the situation). This problem is PSPACE-hard. In the application involving ship routing, the salient problem is multi-objective planning in a dynamic and uncertain environment. The ship routing problem is exacerbated by the need to address multiple conflicting objectives (as many as fifteen objectives, such as fuel efficiency, voyage time, distance), spatial and temporal uncertainty associated with the weather and multiple constraints on asset operation (e.g., ship limits, navigator specified deadlines, bathymetry, waypoints, etc.).

Lastly, the second major thrust of this thesis is the identification of noise covariances in a steady-state Kalman filter \[^{[85]}\]. The Kalman filter is the state estimator for linear

\[^{[1]}\) In computational complexity theory, PSPACE-hard means that a polynomial amount of memory is needed to arrive at an optimal solution for a given problem.
dynamic systems driven by Gaussian white noise with measurements corrupted by Gaussian white noise. In the classical design of a Kalman filter, the noise covariance matrices are assumed known and they, along with system dynamics, determine the achievable filter’s accuracy. However, in many practical situations, including noisy feature data in machine learning, the statistics of the noise covariances are often unknown or only partially known. Thus, noise identification is an essential part of adaptive filtering. Although this problem has a long history, reliable algorithms for their estimation are not available, and necessary and sufficient conditions for identifiability of the covariances are in dispute. We address both of these issues in this dissertation.

1.2 Outline of the Dissertation

This dissertation is organized as follows: In Chapter 2, we solve the nuclear fuel assembly loading pattern optimization problem where we obtain a large number of ranked solutions (as many as \(10^4\) ranked solutions) to the 3-dimensional (3-D) assignment problems with a non-unity right-hand side constraint. The solution approach involves two phases, in phase I, we partition the original problem space into a series of subproblems via Murty’s \(m\)-best search space decomposition procedure. Modifications previously proposed in the literature for the 2-dimensional (2-D) assignment problem are applied to optimize the search space decomposition for the 3-D assignment problem. In phase II, we solve each subproblem by using Lagrangian relaxation and solving the 3-D assignment problem as a combination of relaxed 2-D assignment problems and 2-D transportation problems. The 2-D assignment problem is solved by the JVC or
auction algorithms, and the 2-D transportation problem is solved by the simplex-based transportation, Transauction or RELAX-IV algorithms. The sequence of relaxed 2-D problems are interchangeable, while adhering to the relaxed constraints. We validate and compare the performance and utility of the proposed algorithms and search space decomposition optimizations via extensive numerical experiments.

In Chapter 3, we tackle the problem of targeting in uncertainty, where we delve into surveillance operations in counter-drug smuggling. We validate four approximate dynamic programming approaches and three branch-and-cut-based methods on a maritime surveillance problem involving the allocation of multiple heterogeneous assets over a large area of responsibility to detect multiple drug smugglers using heterogeneous types of transportation on the sea with varying contraband weights. The asset allocation is based on a probability of activity surface, which represents spatio-temporal target activity obtained by integrating intelligence data on drug smugglers’ whereabouts/waypoints for contraband transportation, their behavior models, and meteorological and oceanographic information. We validate the proposed algorithmic concepts via realistic mission scenarios. We conduct scalability analyses of the algorithms and conclude that effective asset allocations can be obtained within seconds using rollout-based ADP. The contributions of this work have been transitioned to and are currently being tested by Joint Interagency Task Force–South (JIATF-South), an organization tasked with providing the initial line of defense against drug trafficking in the East Pacific and Caribbean Oceans.

Chapter 4 details an enhancement to TMPLAR, a mixed-initiative tool for multi-objective planning and asset routing in dynamic and uncertain environments. It is built upon multi-objective dynamic programming algorithms to route assets in a timely fashion, while considering objectives, such as fuel efficiency, voyage time, distance, and
adherence to real world constraints (asset vehicle limits, navigator-specified deadlines, etc.). The ship routing problem is exacerbated by the need to address multiple conflicting objectives, spatial and temporal uncertainty associated with the weather and multiple constraints on asset operation. The NAPO algorithm optimizes weather-based objectives in a reasonable amount of time, optimizing arrival and departure times at waypoints, asset speed and bearing. The key algorithmic contribution is a fast approximate method for substantially containing the computational complexity by generating the Pareto-front of the multi-objective shortest path problem for networks with stochastic non-convex edge costs, utilizing approximate dynamic programming and clustering techniques. The proposed algorithm is validated by comparing its performance with the new approach to multi-objective A* (NAMOA*), an existing multi-objective optimization algorithm.

In Chapter 5 we discuss the topic of adaptive Kalman filtering, where we present the new approach to identify the unknown noise covariances. The Kalman filter requires knowledge of the noise statistics; however, the noise covariances are generally unknown. Although this problem has a long history, reliable algorithms for their estimation are scant, and necessary and sufficient conditions for identifiability of the covariances are in dispute. We address both of these issues in this thesis. We first present the necessary and sufficient condition for unknown noise covariance estimation; these conditions are related to the rank of a matrix involving the auto and cross-covariances of a weighted sum of innovations, where the weights are the coefficients of the minimal polynomial of the closed-loop system transition matrix of a stable, but not necessarily optimal, Kalman filter. We present an optimization criterion and a novel six-step approach based on a successive approximation, coupled with a gradient algorithm with adaptive step sizes, to estimate the steady-state Kalman filter.
gain, the unknown noise covariance matrices, as well as the state prediction (and updated) error covariance matrix. Our approach enforces the structural assumptions on unknown noise covariances and ensures symmetry and positive definiteness of the estimated covariance matrices. We provide several approaches to estimate the unknown measurement noise covariance $R$ via post-fit residuals, an approach not yet exploited in the literature. The validation of the proposed method on five different test cases from the literature demonstrates that the proposed method significantly outperforms previous state-of-the-art methods. It also offers a number of novel machine learning motivated approaches, such as sequential (one sample at a time) and mini-batch-based methods, to speed up the computations.

We summarize and discuss the research impact of the proposed approaches in Chapter 6.

1.3 Publications

Journal papers that are accepted and published with primary authorship include [186, 188, 190]:


Conference papers that are accepted and published with primary authorship include [187,189]:


Patents that are accepted and published with primary authorship include [184,185]:


Journal papers that are accepted and published with co-authorship include [158, 161]:


Conference papers that are accepted and published with co-authorship include [8, 23, 72, 112, 119]:


Book chapters that are accepted and published with co-authorship include [118]:


Patent applications that are currently pending with co-authorship include [47]:

Chapter 2

Approaches to Obtain a Large Number of Ranked Solutions to 3-Dimensional Assignment Problems

2.1 Introduction

2.1.1 Motivation

Assignment problems are applicable to a diverse array of real world problems [46,55,140]. This set of problems takes the form of how best to assign a number of items or objects to some (possibly different) number of machines or people during different time periods. Assignment problems are of a combinatorial nature, each requiring some form of an objective function to indicate the value or utility of individual assignments. A sampling of how diverse and widely applicable such assignment problems are can
be seen from the following: multi-target tracking, quadratic assignment problems, traveling salesman problems, or vehicle routing problems. Such problems also occur in academia or the military, where a set of military troops or teachers must be assigned to locations or classrooms that are temporally dependent in value or utility. Assignment problems have even been motivated from a telecommunications standpoint, where a set of satellites must be launched from a set of locations to maximize their coverage.

A 2-dimensional (2-D) assignment problem may be viewed as a weighted bipartite graph matching problem, where arcs must link two sets of nodes together such that an objective function is optimized, while satisfying a set of one-to-one constraints. The 3-dimensional (3-D) extension of this problem has been proven to be NP-hard. In particular, one application that we focus on in this chapter is a nuclear fuel assembly (FA) loading pattern optimization. The core of a nuclear reactor is formed by large sets of elongated, rectangular FAs arranged in a cylindrical fashion, as shown in Fig. 2.1.

The nuclear fuel assembly loading pattern optimization problem involves choosing: 1) the position of the FA in the nuclear reactor core, 2) the type of FA to put in the chosen position, and 3) the rotation/orientation of the chosen FA type in the chosen position. Each dimension of the 3-D assignment corresponds to each of the decision variables above. In general, this problem is treated as a multiple objective combinatorial problem, but what separates it from the traditional 3-D assignment problems is the requirement for a dense set of new discrete loading patterns though a dynamically estimated probability distribution (represented by a reward tensor). This conversion to a 3-D assignment problem is a completely new approach for nuclear fuel loading pattern optimization. The reward tensor is dynamically updated based
Figure 2.1: The core of a nuclear reactor is formed by large sets of fuel assemblies where position, type, and rotation/orientation must be chosen for each one. Illustrated here is a nuclear fuel assembly loading operation at Fangchenggang nuclear power plant in China’s Guangxi province.

on the “best” solutions taken from the multi-objective Pareto front. “Best” in this case may not necessarily refer to the optimal, but one of a large number of solutions (assignments). By “large,” we mean on the order of $10^4$ solutions. Evaluation of each loading pattern by reactor-physics-based external code may be very time consuming ($\approx 0.1$ to 10 minutes, depending on the required accuracy of loading pattern response evaluation), so there exists a need to evaluate only new (unique) loading patterns (assignments).

In such scenarios, an $m$-best 3-D assignment problem is needed, wherein a large set of solutions is generated in a reasonable amount of time (< 10 minutes for $10^4$ solutions), so that the set of assignments may be externally evaluated (each of which, in turn, may take 0.1 to 10 minutes). It may also be a viable approach to obtain a dense set of solutions that are near-optimal and satisfy the decision maker (such as in the case of resource allocation or military troop allocation problems) or customer preferences.
(as in [55], where they attempt to satisfy both student and tutor requirements or requests). Having a large set of solutions offers a range of options that may be of interest to a decision maker attempting to optimize with respect to multiple, possibly conflicting, objectives.

This chapter offers an effective solution approach for finding a large number of \( m \)-best solutions to the 3-D assignment problems with non-unity right-hand side constraints with application to many real world challenges. The problem space may be decomposed into multiple partitions based on the optimal assignment, as detailed in [124]. Through a two-phase approach, we offer a method for rapidly generating large numbers of solutions to the 3-D assignment problems.

### 2.1.2 Related Research

There exist a number of well-known algorithms to obtain the optimal solution to a 2-D assignment problem, including the Hungarian algorithm [96], the Jonker-Volgenant-Castañón (JVC) algorithm [50, 82], the auction algorithm [17], and the signature method [10]. However, the assignment problem becomes NP-hard when a third dimension is added [56, 86, 133]. One of the first approaches for solving the 3-D assignment problem was developed by Pierskalla [140], where he proposed a tri-substitution algorithm based on the simplex method. Hansen [68] proposed a primal-dual implicit enumeration algorithm, while [9, 26] proposed branch-and-bound approaches to obtain the optimal solution to such 3-D assignment problems. However, branch-and-bound methods suffer from exponential computational complexity and are unsuitable for large-scale real-world applications where accurate bounds cannot be obtained.
In order to overcome the 3-D assignment problem’s inherent computational intractability, a wide range of algorithms have been developed to obtain suboptimal solutions, including greedy heuristics, genetic algorithms, simulated annealing, tabu search, neural networks, and Lagrangian relaxation approaches [55, 111, 135, 145, 146]. Mazzola [111] proposed a heuristic branch-and-bound method to reduce the computation time. In contrast, Frieze and Yadegar [55] applied Lagrangian relaxation theory to a more general 3-D assignment problem with application to teaching practice scheduling. The Lagrangian relaxation method of obtaining solutions to 3-D assignment problems has become extremely prevalent in data association applications due to the real-time computation speed and solution quality [46, 135, 146]. Poore [145] combined these two approaches, proposing a hybrid branch-and-bound and Lagrangian relaxation algorithm to the 3-D assignment problem.

In this chapter, we seek to solve the aforementioned 3-D assignment problem, but instead of finding a single solution, we aim to provide a large set of ranked solutions. The process of finding the first best, second best, third best, and so on, solution is known as the \( m \)-best optimization problem. The \( m \)-best optimization problem occurs in a variety of contexts, including the shortest path [5, 45, 78], spanning tree [3, 57, 65], traveling salesman [174], directed network [28], multi-target tracking [16, 39, 40, 147, 148] and many other problems. The general approach to the \( m \)-best optimization problem involves partitioning the solution space into smaller subspaces, which are subproblems of the original problem. Murty’s search space decomposition [124] is the most common and widely used technique, where the best solution is found for each partitioned subproblem, given a modified solution subspace. Lawler [98] applied Murty’s search space decomposition procedure within a more general framework for a discrete optimization problem. Pascoal [134] proposed a variant of Murty’s search
space decomposition to reduce the algorithm’s complexity. This variant involved solving
the partitioned subsets in reverse order. Miller et al. [116] proposed modifications to
optimize Murty’s search space decomposition procedure to the 2-D assignment problem
via: 1) inherited dual variables and partial solutions from the initial subproblems;
2) sorting the subproblems based on lower bounds on the optimal reward before
solving the assignment problem; and 3) partitioning in an order based on lower bounds
on cumulative reward. These modifications substantially reduce the complexity of
Murty’s search space decomposition and are implemented in this chapter.

Another alternative way to solve the $m$-best optimization problem is by Gabow’s
binary heap partition method. Similarly, Hamacher [64] also proposed using a
binary search tree procedure, while also combining an approach developed by Carraresi
and Sodini [32] to rank the paths. Chegireddy and Hamacher [34] extended this work
further and developed an $m$-best perfect matching algorithm based on the binary
partition of the solution space to apply to a bipartite matching problem in $O(kn^3)$
time. Recently, a modified version of the Chegireddy and Hammacher’s algorithm
was developed for large datasets [102]. We suggest comparison of our algorithm with
those in [102] as future research.

2.1.3 Chapter Organization

The primary focus of this chapter is on combining a Lagrangian relaxation method and
$m$-best optimization to obtain a very large number of ranked solutions. Motivated by
an approach developed by Pattipati [135], we apply the Lagrangian relaxation approach
that successively solves a series of 2-D problems, since a key advantage of using the
Lagrangian relaxation method is that it prunes the solution space by computing the
upper and lower bounds. The first 2-D problem is a bipartite graph matching problem (2-D assignment problem), which can be solved using either the auction algorithm or the JVC algorithm \[82\]; the latter is more efficient for dense problem spaces \[50\]. The feasible solution is obtained by solving a 2-D transportation problem (via a simplex algorithm or Transauction algorithm) reconstructed from the relaxed solution of the 2-D assignment problem. The second step corresponds to imposing the originally relaxed constraint on the first subproblem’s solutions. As in \[147\], we generate \(m\)-best solutions by exploiting Murty’s search space decomposition procedure. Additionally, we optimize Murty’s search space decomposition via Miller’s \[116\] proposed modifications. An alternate Lagrangian relaxation method involves first solving a 2-D transportation problem at each iteration of the 3-D assignment algorithm using either a simplex algorithm or the Transauction algorithm, and subsequently reconstructing the feasible solution via a 2-D assignment problem. We will show that the former Lagrangian relaxation method is two orders of magnitude faster than the latter.

This chapter is organized as follows. We begin by introducing the problem formulation in Section 2.2. In Section 2.3, we solve the \(m\)-best 3-D assignment problem via Murty’s search space decomposition and the Lagrangian relaxation method. In Section 2.4, we detail Miller et al.’s \[116\] search space optimizations and extend them to the 3-D assignment problem. We provide the pseudocode of the fully optimized \(m\)-best 3-D assignment solution algorithm in Section 2.5. In Section 2.6, we present the results of the \(m\)-best 3-D assignment algorithm and the performance of each different optimization technique.
## 2.2 Problem Formulation

The notation used in the remainder of this chapter is listed in Table 2.1.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{ijk}$</td>
<td>Reward of allocating resource $i$ to task $j$ at time $k$</td>
</tr>
<tr>
<td>$x_{ijk}$</td>
<td>Binary decision variable for the primal problem</td>
</tr>
<tr>
<td>$y_{ij}$</td>
<td>Binary decision variable for the 2-D assignment problem</td>
</tr>
<tr>
<td>$z_{jk}, z_{ik}$</td>
<td>Binary decision variables for the 2-D transportation problem</td>
</tr>
<tr>
<td>$i$</td>
<td>Resource index</td>
</tr>
<tr>
<td>$j$</td>
<td>Task index</td>
</tr>
<tr>
<td>$k$</td>
<td>Time index</td>
</tr>
<tr>
<td>$m_k$</td>
<td>Maximum number of assignment allowed for each $k$</td>
</tr>
<tr>
<td>$W$</td>
<td>Reward tensor</td>
</tr>
<tr>
<td>$N$</td>
<td>Total number of tasks/resources</td>
</tr>
<tr>
<td>$R$</td>
<td>Total number of time units</td>
</tr>
<tr>
<td>$\mu$</td>
<td>Lagrange multiplier</td>
</tr>
<tr>
<td>$q$</td>
<td>Upper bound found from the relaxed 2-D assignment problem (dual)</td>
</tr>
<tr>
<td>$f$</td>
<td>Lower bound found via simplex-based transportation or Transaucation problem (primal)</td>
</tr>
<tr>
<td>$g$</td>
<td>Gradient vector for the subgradient update</td>
</tr>
<tr>
<td>$\mathcal{P}_0$</td>
<td>Original problem space</td>
</tr>
<tr>
<td>$A$</td>
<td>Solution space</td>
</tr>
<tr>
<td>$S$</td>
<td>Feasible assignment in solution space $A$</td>
</tr>
<tr>
<td>$X$</td>
<td>Solution tensor</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>Column for row solution</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Optimal reward from the 2-D assignment problem</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>Layer for row solution</td>
</tr>
<tr>
<td>$\omega$</td>
<td>Optimal reward from the 2-D transportation problem</td>
</tr>
<tr>
<td>$B$</td>
<td>Slack value for upper bound reward computation</td>
</tr>
<tr>
<td>$C$</td>
<td>Relaxed 2-D reward matrix in the 2-D assignment problem</td>
</tr>
<tr>
<td>$T$</td>
<td>2-D reward matrix for the transportation problem</td>
</tr>
</tbody>
</table>
2.2.1 Problem Formulation

Given a 3-D reward tensor $W = [w_{ijk}]$ of dimension $N \times N \times R$, our problem is the following:

$$\max_{x_{ijk} \in \{0,1\}} \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{R} w_{ijk} x_{ijk}$$  \hspace{1cm} (2.1)

s.t. \hspace{1cm} \sum_{j=1}^{N} \sum_{k=1}^{R} x_{ijk} = 1, i = 1, \ldots, N \hspace{1cm} (2.2)

\sum_{i=1}^{N} \sum_{k=1}^{R} x_{ijk} = 1, j = 1, \ldots, N \hspace{1cm} (2.3)

\sum_{i=1}^{N} \sum_{j=1}^{N} x_{ijk} \leq m_k, k = 1, \ldots, R \hspace{1cm} (2.4)

where $x_{ijk}$ is a binary decision variable such that $x_{ijk} = 1$ if resource (row) $i$ is assigned to task (column) $j$ at time (layer) $k$, and 0 otherwise. Constraints (2.2) and (2.3) ensure that each resource $i$ is allocated to exactly one task $j$ and vice versa. Constraint (2.4) requires that there may be no more than $m_k$ assignments at each time $k$ and makes this assignment problem non-standard.

Figure 2.2 shows the 3-D assignment problem as a network flow problem. Consider the first set, indexed by $i$, and the second set, indexed by $j$, each consisting of $N$ nodes. Also, consider a third set, indexed by $k$, with a total of $R$ nodes. There are a total of $N$ assignments that may be made between sets $i$ and $j$ based on constraints (2.2) and (2.3). We view this as a 2-D assignment problem (indicated by the solid box in Fig. 2.2). Additionally, each node in set $j$ must be assigned to one of the nodes in set $k$ (indicated by the dashed (blue) box in Fig. 2.2). Due to constraint (2.4), for every $k$, there may be no more than $m_k$ assignment pairs of $(i,j)$ mapped to each layer. This
may be viewed as an unbalanced transportation problem, where the nodes in set \((i,j)\) are the sources and the nodes in set \(k\) are the sinks. Note that \(\{m_k : k = 1, 2, \ldots R\}\) should be such that \(\sum_{k=1}^{R} m_k \geq N\) so that each \((i,j)\) can be assigned to a node \(k\).

### 2.2.2 Special Cases

Note that our 3-D assignment problem formulation covers a wide range of problems.

#### 2.2.2.1 Tri-index Assignment problem

The problem in (2.1)–(2.4) may be viewed as a traditional tri-index assignment problem by setting \(m_k = 1\) and \(R = N\).

#### 2.2.2.2 Scheduling problem

By setting \(m_k = m\), the problem in (2.1)–(2.4) is related to some resource-constrained assignment scheduling problems.

#### 2.2.2.3 Transportation problem

Note that our problem formulation is a special case of the transportation problem. The general transportation problem involves altering the unity constraint to some non-unity values.

#### 2.2.2.4 Nuclear Fuel Loading Pattern Optimization

In some nuclear reactor fuel assembly loading pattern optimization problems, \(m_k = N\) on the right hand side of the constraint (2.4). In this case, the problem can be reduced
to the traditional 2-D assignment problem, since constraint (2.4) can be subsumed under constraints (2.2) and (2.3) and is, thus, unnecessary. The 3-D assignment problem posed in (2.1) then devolves to a 2-D assignment problem, detailed later in Section 2.3.1.4. An $m$-best 2-D assignment problem is adequate for this version of the problem.

### 2.3 Solution Approach

In order to solve this NP-hard problem, we propose a two-phase solution approach. In phase I, we utilize Murty’s search space decomposition to partition the original
problem space into a series of subproblems. Each subproblem is then relaxed and solved by a 3-D assignment algorithm in phase II.

2.3.1 3-D Assignment Relaxations

We adopt the solution approach of the 3-D assignment problem in [135] by relaxing one of the three constraints and solving the 3-D assignment problem as a series of 2-D subproblems. Since sets $i$ and $j$ have the unity constraint, a similar solution approach can be applied to the 3-D assignment problem here by relaxing either of the two sets of constraints. We then denote Relaxation Method I and Relaxation Method II as the solution approaches for the 3-D assignment problem when constraints \((2.4)\) or \((2.2)/(2.3)\) are relaxed, respectively.

2.3.1.1 Relaxation Method I

Relaxation Method I is developed by relaxing constraint \((2.4)\) via a set of Lagrange multipliers \(\{\mu_k : k = 1, 2, \ldots, R\}\). The result is the Lagrangian function

\[
L(x, \mu) = \max_{x_{ijk} \in \{0,1\}} \left( \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{R} (w_{ijk} - \mu_k) x_{ijk} \right) + m_k \sum_{k=1}^{R} \mu_k \tag{2.5}
\]
Equation (2.5) is then a relaxed 2-D assignment problem of the form,

\[
\begin{align*}
\max_{y_{ij} \in \{0, 1\}} & \sum_{i=1}^{N} \sum_{j=1}^{N} \max_k (w_{ijk} - \mu_k) y_{ij} \\
\text{s.t.} & \sum_{i=1}^{N} y_{ij} = 1, \quad j = 1, \ldots, N \\
& \sum_{j=1}^{N} y_{ij} = 1, \quad i = 1, \ldots, N
\end{align*}
\] (2.6)

where,

\[
y_{ij} = \sum_{k=1}^{R} x_{ijk}; \quad i, j = 1, \ldots, N.
\] (2.9)

The upper bound \(q\) of the relaxed 2-D assignment problem is easily solvable via a 2-D assignment algorithm. To obtain a feasible solution, we reimpose constraint (2.4) by reconstructing the reward tensor and viewing the asymmetric bipartite graph as a transportation problem based on the solution of the relaxed 2-D assignment problem.

For each \(\langle i^*, j^* \rangle\) of the relaxed 2-D assignment problem at each iteration, the reward matrix is dynamically updated for each layer \(k\). Given a new reward matrix \(\tilde{w}_{(i,j)k}\), the transportation variation of the problem is as follows.

\[
\begin{align*}
\max_{z_{jk} \in \{0, 1\}} & \sum_{j=1}^{N} \sum_{k=1}^{R} \tilde{w}_{(i,j)k} z_{jk} \\
\text{s.t.} & \sum_{j=1}^{N} z_{jk} = 1, \quad k = 1, \ldots, R \\
& \sum_{k=1}^{R} z_{jk} \leq m_k, \quad j = 1, \ldots, N
\end{align*}
\] (2.10)
Through this sequence, we obtain a feasible solution and a lower bound $f$. The upper and lower bounds serve as measures of the solution quality. The distance between these bounds is referred to as the approximate duality gap (because it is overestimated by $(f - f^*)$, where $f^*$ is the optimal solution). For discrete 3-D assignment problems, the duality gap may be nonzero. The relative approximate duality gap is given by

$$\text{gap} = \frac{|q - f|}{f}$$

(2.13)

where $q$ and $f$ are the upper and lower bounds, respectively, obtained by solving the series of 2-D subproblems. The 3-D assignment algorithm terminates for a sufficiently small gap, which implies that a near-optimal solution has been obtained. In scenarios where the duality gap is large, the 3-D assignment algorithm updates its Lagrange multipliers via the method proposed in Pattipati [135]. Let us denote $g$ as an $R$-dimensional subgradient vector with components given by

$$g_k = R - \sum_{i=1}^{N} \sum_{j=1}^{N} X_{ijk} \quad k = 1, \ldots, R,$$

(2.14)

where $X$ is the solution tensor related to the optimal value of the relaxed 2-D assignment variables $\{y_{ij}^*\}$ via

$$X_{ijk} = \begin{cases} y_{ij}^*, & \text{if } k = \arg\min_{\alpha} (w_{ija} - \mu_{\alpha}) \\ 0, & \text{otherwise} \end{cases}$$
We then update the Lagrange multipliers by

\[
\mu_k = \max \left( \mu_k - \frac{(p - f)}{\|g\|^2} g_k, 0 \right).
\]  

(2.15)

After updating the Lagrange multipliers, the algorithm iterates back to the relaxation step. The process continues until either the maximum number of iterations is reached or the duality gap is sufficiently small. The flow diagram of the 3-D assignment algorithm when the constraint in (2.4) is relaxed is shown in Fig. 2.3.

2.3.1.2 Relaxation Method II

Note that a relaxed problem is also obtainable by interchanging the sequence of 2-D subproblems. In other words, we may apply the Lagrangian relaxation on constraints (2.2) or (2.3). When constraint (2.3) is relaxed via Lagrange multipliers \(\mu_j\), the Lagrangian function is:

\[
L(x, \mu) = \max_{x_{ijk} \in \{0,1\}} \left( \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{R} (w_{ijk} - \mu_j) x_{ijk} \right) + \sum_{k=1}^{N} \mu_j \tag{2.16}
\]

The 3-D assignment problem is then relaxed into a 2-D transportation problem of the form

\[
\max_{z_{ik} \in \{0,1\}} \sum_{i=1}^{N} \sum_{k=1}^{R} \max_j (w_{ijk} - \mu_j) z_{ik} \tag{2.17}
\]

s.t. \(\sum_{k=1}^{R} z_{ik} = 1, i = 1, \ldots, N\)  

(2.18)

\[
\sum_{i=1}^{N} z_{ik} \leq m_k, k = 1, \ldots, R, \tag{2.19}
\]
where

$$z_{ik} = \sum_{j=1}^{N} x_{ijk}; \ i = 1, \ldots, N; \ k = 1, \ldots, R \quad (2.20)$$

The upper bound $q$ can be obtained by solving the relaxed 2-D transportation problem. The 2-D assignment problem is obtained by reimposing constraint (2.3) and reconstructing the reward tensor based on the solution of the relaxed 2-D transportation problem. The assignment variation of the problem is as follows.

$$\max_{y_{ij} \in \{0,1\}} \sum_{i=1}^{N} \sum_{j=1}^{N} \tilde{w}_{(i,k)j} y_{ij} \quad (2.21)$$

$$\text{s.t.} \sum_{i=1}^{N} y_{ij} = 1, \ j = 1, \ldots, N \quad (2.22)$$

$$\sum_{j=1}^{N} y_{ij} = 1, \ i = 1, \ldots, N \quad (2.23)$$

A feasible solution and a lower bound $f$ can be obtained through this sequence. The duality gap is then computed and compared for algorithm termination. The subgradient is updated in a similar fashion to the first relaxation method, except that it is with respect to dimension $j$ and uses binary decision variables $\{z_{ik}\}$.

2.3.1.3 Algorithm selection for 2-D subproblems

To optimize the 3-D assignment algorithm, state-of-the-art 2-D assignment and transportation algorithms were selected for comparison purposes. The JVC and auction algorithms were selected for comparison when solving the 2-D assignment problem. We solve the 2-D transportation problem via three approaches. The first algorithm
utilizes the Transauction algorithm developed by Bertsekas and Castaño [20], which solves the transportation problem by mapping it to an assignment problem and obtains a solution via a modified auction algorithm. In the second algorithm, we exploit the findings in [133, 167], where the transportation problem was found to be equivalent to the minimum cost network flow problem, and solve the 2-D transportation problem via a (weakly polynomial) simplex-based method. We refer to this simply as simplex-based transportation. The third algorithm is the RELAX-IV algorithm developed by Bersekas and Tseng [21] and further detailed in [54]. It is one of the most efficient algorithms to solve problems of the network flow type.

2.3.1.4 Solution approach for a variant of the nuclear FA loading pattern optimization problems

for this problem, constraint (2.4) is such that \( m_k = N \). In this case, the summations over sets \( i \) and \( j \) are always less than or equal to \( N \) for each \( k \), and, consequently, constraint (2.4) is always satisfied. This implies that the constraints in (2.4) are inactive and the Lagrange multipliers \( \mu_k = 0 \) for \( k = 1, 2, \ldots, R \). Consequently, the 3-D assignment problem takes the form,

\[
\max_{y_{ij} \in \{0,1\}} \sum_{i=1}^{N} \sum_{j=1}^{N} \max_{k} (w_{ijk}) y_{ij} \tag{2.24}
\]

s.t.
\[
\sum_{i=1}^{N} y_{ij} = 1, j = 1, \ldots, N \tag{2.25}
\]
\[
\sum_{j=1}^{N} y_{ij} = 1, i = 1, \ldots, N \tag{2.26}
\]
Initialization

\[ \mu_k = 0, \]
\[ k = 1, 2, \ldots, R, \]
\[ q = -\infty \]

Relax constraint in (2.4) via Lagrange multiplier vector \( \{\mu_k\} \)

Optimally solve the relaxed 2-D subproblem

Enforce constraint in (2.4)

\[ \text{gap} = \frac{|q - f|}{f} \]

Is the gap sufficiently small?

No

Yes

Terminate

Figure 2.3: Flow diagram of the 3-D assignment algorithm when relaxing constraint (2.4).

This problem can be easily solved by an \( m \)-best 2-D assignment algorithm. Furthermore, the approach is the same for the general case when \( m_k \geq N \).

### 2.3.2 \( m \)-best 3-D Assignment

Let \( \mathcal{P}_0 \) be the original problem in equations (2.1)–(2.4) and let \( A \) be the corresponding assignment solution space. Further, let \( A^*_0 \) be the best feasible assignment found by the 3-D assignment algorithm detailed in Section 2.3.1. In general, to find the
(n + 1)th best solution, we have to partition the (n + 1)th problem space, \( \mathcal{P}_n \), into \( N \) subproblems, denoted by \( \mathcal{P}_{nr} \), \( 1 \leq r \leq N \). Then, the complete solution space corresponding to problem space \( \mathcal{P}_n \) is

\[
A_n = \bigcup_{r=1}^{N} A_{nr} = A - \bigcup_{i=0}^{n-1} A_i^* \quad \text{for } n = 1, 2, \ldots, m \tag{2.27}
\]

\[
A_{nr} \cap A_{ns} = \emptyset \quad \text{for } r, s = 1, 2, \ldots, N \quad r \neq s, \tag{2.28}
\]

where \( A_{nr} \) denotes a set of tuples in which each \( i \) and \( j \) appear exactly once, but \( k \) may be repeated. Equation (2.27) is a formalization of the constraint that the solution space \( A_n \) for the \( (n + 1) \)th best solution will not contain any of the best solutions obtained for the previous \( n \) problems. Here, a complete feasible solution is assumed to be a set of tuples. Hence, some solutions may have a similarity, however, as seen in (2.28), the set of solution tuples as a whole are unique, differing by at least one element for each of the previous \( n \) problems. Let an assignment \( A_{nr} \) consist of multiple tuples (in this chapter, triples), where we index the triples within by \( t \). Let \( \ell_{nr_t} \) be the individual reward of the \( t \)th triple, sub-indexed as \( < i_{nr_t}, j_{nr_t}, k_{nr_t} > \), in the solution space \( A_{nr} \). We can then augment the triple into a 4-tuple and write a feasible assignment in \( A_{nr} \) as

\[
S_{nr} = \{ < i_{nr_t}, j_{nr_t}, k_{nr_t}, \ell_{nr_t} > \} \quad \text{for } t = 1, \ldots, N. \tag{2.29}
\]
The primal value of the corresponding assignment \( S_{nr} \) is denoted by \( f_{nr} \), which can be obtained by summing \( \ell_{nrt} \) over \( t = 1, 2, \ldots, N \).

\[
f_{nr} = \sum_{t=1}^{N} \ell_{nrt}
\] (2.30)

The best assignment \( A_{nr}^* \) with the corresponding primal value \( f_{nr}^* \) in the solution space \( A_{nr} \) is found via the 3-D assignment algorithm described earlier and pertains specifically to partition \( r \). The best assignment \( A_n^* \) is found by iterating over all active partitions and finding the argument \( r^* \) which has the maximum primal value.

\[
A_n^* = A_{nr}^*
\]

(2.31)

\[
r^* = \arg \max_r f_{nr}^*
\]

(2.32)

Given the original problem space and its optimal assignment, denoted by \( P_0 \) and \( A_0^* \), respectively, we partition \( P_0 \) into \( N \) problem subspaces \( P_{11} \) to \( P_{1N} \) in order to find the next best solution. To generate subproblem \( P_{11} \), we remove the first of \( N \) tuples in the assignment \( A_0^* \). We then use the 3-D assignment algorithm to obtain the best possible solution \( A_{11}^* \) to problem \( P_{11} \). To partition the subspace \( P_{1s} \), \( 2 \leq s \leq N \), we remove the \( s^{th} \) tuple in \( A_0^* \) as a feasible assignment in \( P_{1s} \), while fixing the first \((s - 1)\) triples to those in the original assignment \( A_0^* \). Thus, as the solution and problem spaces are reduced at every search space decomposition, the complexity of the problem decreases substantially, since the first \((s - 1)\) triples are reused from the previous assignments. We then only need to find assignments for the remaining \( N - s \) assignments, such that the \( s^{th} \) triple from the original assignment \( A_0^* \) is not contained in the solution, while satisfying the constraints. The enforcement of tuples to be either
in or be removed from the problem spaces $\mathcal{P}_{11}$ to $\mathcal{P}_{1N}$ during partitioning ensures the disjointness of the individual subproblems, as in equation (2.28).

Each of the best solutions $A_{11}^*$ to $A_{1N}^*$ is saved into a heap and accordingly sorted based on the respective primal values, $f_{11}^*$ to $f_{1N}^*$. The best solution within the heap is then removed and saved as the second best solution. The problem corresponding to the second best solution is then partitioned into the subproblems $\mathcal{P}_{21}$ to $\mathcal{P}_{2N}$. The best assignment from the top of the heap is then marked as the third best assignment with respect to the original problem $\mathcal{P}_0$. We continue to apply this process until the $m^{th}$ best solution is found or, alternatively, the heap becomes empty.

Murty’s search space decomposition is an ingenious way of decomposing the search space, and has a number of applications in combinatorial optimization [66,98]. Optimizations of the decomposition technique to improve the computational efficiency are discussed in Section IV.

Remark: For small size problems and large $m$, if we apply the Lagrangian relaxation on constraint (2.4), the transportation problem reconstructed from the best $(i, j)$ pair of the 2-D assignment problem may contain too many removed arcs and, thus, no feasible solution may exist. In this case, by interchanging the sequence of relaxed problems solved (i.e., solve the 2-D transportation problem first, as opposed to the assignment problem (normally solved first)), we can obtain a feasible solution to the 3-D assignment problem. This situation arises in small size problems (e.g., of dimension $3 \times 3 \times 2$). However, since the tensor dimensions used in this chapter are large, the solution space is vast and this anomaly did not arise.
2.4 Optimized implementation of Murty’s search space decomposition

We extend the 2-D optimization modifications in [116] to the 3-D assignment problems. These include: 1) inheriting the dual variables and partial solutions from the subproblems being decomposed; 2) sorting the subproblems by an upper bound on reward before solving; and 3) partitioning the subproblems in an optimized order. All three modifications exploit the primal-dual aspects of the JVC algorithm. The following sections explain each modification in detail for the case when the constraints in (2.4) are relaxed in the $m$-best 3-D assignment algorithm. Similar optimization techniques can be applied for the case when the constraints in (2.3) are relaxed.

2.4.1 Inheriting dual variables and partial solutions during partitioning

Solving the 3-D assignment problem via the JVC algorithm provides dual variables $u$ and $v$, which can be inherited by the partitioned subproblem using Murty’s search space decomposition. The solution tensor $X_n$, for the problem space $\mathcal{P}_n$ and the reward tensor $W$, contains $N$ solution triples $<i^*,j^*,k^*>$. During each step of Murty’s search space decomposition, a new subproblem $\mathcal{P}_{nr}$ is generated, associated with a new reward tensor $W'$. Removing the triple $<i^*,j^*,k^*>$ from the subproblem space $\mathcal{P}_{nr}$ is equivalent to setting $w_{<i^*,j^*,k^*>} = -\infty$. This implies we may skip the initialization step for the JVC algorithm and go directly to the augmentation step with only one arc left to assign in the 2-D assignment problem, following the procedure outlined in Algorithm [1]. In this case, the initialization step is only required for the first feasible solution to the 3-D assignment problem.
Algorithm 1 Upper bound reward calculation when inheriting dual variables

1: for each $<i^*, j^*, k^*> \in A$ do
2: \[ w_{<i^*, j^*, k^*>} = -\infty \]
3: \[ u' = u, \ v' = v, \]
4: \[ X' \leftarrow X - X_{<i^*, j^*, k^*>} \]
5: end for

Note that we cannot inherit the Lagrange multipliers $\mu_k$ from the previous problem $P_n$ in the process of partitioning the subproblems. The Lagrange multipliers from the previous problem $P_n$ may be too large for the subproblems $P_{nr}, r = 1, 2, \ldots, N$. This may cause the duality gap to remain above the threshold value required to terminate. Thus, the algorithm will continue to run until the maximum iteration limit is reached.

2.4.2 Sorting subproblems via an upper bound

The upper bound reward of individual subproblems is easily obtainable and can be used to avoid solving subproblems that are unlikely to produce the next best solution. For an $m$-best assignment problem, the best solution from problem $P_n$ is always better than the best solution obtained from the subproblems obtained by partitioning $P_{nr}, r = 1, 2, \ldots, N$. Therefore, for an $m$-best 2-D assignment problem, the objective function of the solution to $P_n$ can be used as an initial upper bound on the objective function value of the best solution to its corresponding subproblems. Since 3-D assignment problems may have a nonzero duality gap, the computation of the upper bound can be determined using either the dual value (denoted by $\phi$) or the primal value (denoted by $\omega$) as initial upper bounds to the partitioned subproblems.

When a subproblem $P_{nr}$ is created by removing a triple $<i^*, j^*, k^*>$ from a copy of $P$, we can compute the upper bound objective function value by finding the best
slack (i.e., next possible best assignment) of all the alternative assignments for a row \( i \). The upper bound objective function value will be the sum of the initial upper bound and the row slack, denoted by \( B_r \). The calculation of the upper bound is shown in detail in Algorithm 2.

Algorithm 2 Upper bound reward calculation when sorting subproblems

1: for each row \( i \) do
2: \[ w_{i,j,k} = -\infty \]
3: \[ B_r = \max_{j,k} \{w_{ijk} - u(i^*) - v(j) - \mu(k)\} \]
4: \[ f' = f + B_r \]
5: end for

A similar procedure can be followed for column \( j \) to find the column slack, \( B_c \). By combining both the row and the column slack, a tighter upper bound can be obtained. The heap of subproblems can be modified to sort its elements (in descending order) based on each element’s respective upper bound reward. This implies that the problems located at the top of the heap are most likely to have the best solutions.

In this optimization method, the initial problem is partitioned into a series of subproblems when it is solved by the 3-D assignment algorithm. Both the original problem and its corresponding subproblems are saved into a heap. During each iteration of Murty’s search space decomposition, if the top problem \( P_n \) removed from the heap has a feasible solution, then the solution will be saved as the \( m^{th} \) best assignment. If \( P_n \) has not yet been solved (i.e., it has a partial solution), then we find its best solution \( A_n^* \) using the 3-D assignment algorithm and add it back into the heap. A new partitioning process is then invoked on \( P_n \) and its solution \( A_n^* \). The process is repeated until the heap is empty or a total of \( m \) solutions are obtained. This method allows us to eliminate subproblems by focusing on their corresponding upper bounds, thus reducing the number of problems needed to be solved by the 3-D assignment.
algorithm.

2.4.3 Partition in an optimized order

The third optimization method proposed here is to carefully select the order in which the partitioning is performed. This modification maximizes the probability that the subsequent smaller subproblems (with a greater number of fixed arcs) have better solutions. For problem $P_n$ with solution $A^*_n$ that contains $N$ triples, we first compute each upper bound reward that would result from excluding each individual arc. These upper bounds are computed via the method explained in Section 2.4.2. We then select the triple that corresponds to the lowest upper bound reward computed and exclude it from the current subproblem, while fixing the corresponding arc in the next subproblem.

In this modification, the heuristic tends to ensure that the largest problem (maximum number of unassigned arcs) has the lowest upper bound. In other words, the largest problem has the highest probability of containing the worst solution and to be pushed to the bottom of the heap (and in turn, will most likely remain unsolved upon algorithm termination). The next worst problem will tend to be the second largest subproblem, and so on. By doing this, we increase the chance that the smallest problem (that which has the least amount of unassigned arcs) contains the best solution.

2.5 Pseudocode

The following variants were used and/or combined for different optimization methods:
(A) Inheritance of the dual variables and partial solutions during partitioning

(B) Sorting subproblems by an upper bound reward before solving, where the upper bound is calculated via:

\[ \omega + B_r \]
\[ \omega + B_r + B_c \]
\[ \phi + B_r \]
\[ \phi + B_r + B_c \]

(C) Partitioning the problem in an optimized order

These variants are denoted as listed for the remainder of the chapter and may be combined, e.g., when combining variant A with variant B(ii) and variant C, the algorithm variant will be categorized as A+B(ii)+C. The pseudocode for Murty’s modified search space decomposition, optimized via variants A, B(ii), and C, is detailed in Algorithm 3. These variants assume JVC and Transauction to be applied in the \( m \)-best 3-D assignment algorithm.

2.6 Results

The proposed \( m \)-best 3-D assignment algorithm was implemented in the MATLAB 2016b and runs on an Intel Core i7-4712HQ CPU processor @ 2.30 GHz with 16 GB RAM. In all experiments, the top \( 10^4 \) ranked solutions were computed.
Algorithm 3 \( m \)-best 3D assignment algorithm

1: \( H \leftarrow \{\} \) \hspace{1cm} \triangleright \text{Initialize binary heap}
2: \( U \leftarrow [] \) \hspace{1cm} \triangleright \text{Initialize solution list}
3: \( \langle A_0^*, P_0, f_0^* \rangle = \text{3DAssign}(w_{ijk}) \)
4: \( \text{Partition}(H, P_0, A_0^*) \) \hspace{1cm} \triangleright \text{Invoke Partition method}
5: \( H \leftarrow \langle A_0^*, P_0, f_0^* \rangle \) \hspace{1cm} \triangleright \text{Add to the heap}
6: \text{counter} = 0
7: \text{while} counter \leq m - 1 \text{ and } H \neq \emptyset \text{ do}
8: \( \langle A_n^*, P_n, f_n^* \rangle = H.\text{pop} \)
9: \text{if } A_n^* \text{ is feasible then}
10: \hspace{1cm} \text{counter} = \text{counter} + 1
11: \hspace{1cm} U \leftarrow A_n^*, f_n^*
12: \text{else}
13: \hspace{1cm} \langle A_n^*, P_n, f_n^* \rangle = \text{3DAssign}(w_{ijk}, \langle A_n^*, P_n, f_n^* \rangle)
14: \hspace{1cm} \text{if } \exists \text{ solution then}
15: \hspace{2cm} \text{Partition}(H, \langle A_n^*, P_n, f_n^* \rangle)
16: \hspace{2cm} H \leftarrow \langle A_n^*, P_n, f_n^* \rangle
17: \text{end if}
18: \text{end if}
19: \text{end while}

2.6.1 Relaxation Method I vs. Relaxation Method II

We first performed 10 Monte Carlo runs to compare the simulation runtimes of the 3-D assignment algorithm when relaxing either constraint (2.3) or constraint (2.4). The reward tensor elements were uniformly distributed in the interval [0,1] and of dimension 60\( \times \)60\( \times \)8. The JVC and Transauction algorithms were implemented to solve the 2-D assignment and the transportation problems, respectively. As shown in Table 2.2, speedup of as much as 2.28 and an average speedup of 1.63 were observed when comparing the two relaxation methods. In general, solving a 2-D assignment problem is significantly faster than solving a transportation problem. The transportation problem obtained from relaxing constraint (2.3) is complex, and thus takes a longer time to solve compared to the transportation problem reconstructed from the best 2-D
Algorithm 4 Partition pseudocode

1: function Partition(H, A* n, P n, f n*, wijk)
2: for each <i*, j*, k*> ∈ A* n do
3: w[i*, j*, k*] = −∞
4: end for
5: for each <i*, j*, k*> ∈ A* n do
6: for each row i* ∈ A* do
7: Br = max_{j,k} \{wijk − u(i*) − v(j) − µ(k)\}
8: Bc = max_{i,k} \{wijk − u(i) − v(j*) − µ(k)\}
9: Bi* = Br + Bc
10: end for
11: (B, i*) = min(Bi * ≠ −∞)
12: i*, j*, k* = A* (i*)
13: f n* = f n + B
14: A* nr ← A* − <i*, j*, k*>
15: P nr ← P n − <i*, j*, k*>
16: H ← A* nr, P nr, f n*
17: A*_{n(r+1)}, FixList ← <i*, j*, k*>
18: for each j, k do
19: w[i*, j, k] = −∞
20: end for
21: for each row ≠ i*, k do
22: w[row, j*, k] = −∞
23: end for
24: end for
25: end function

assignment solution when constraint (2.4) is relaxed. Relaxing constraint (2.4) also consistently resulted in a smaller duality gap compared to when constraint (2.3) was relaxed due to the fact that |k| = |i| = N = |j|. This implies that when constraint (2.4) is relaxed, a smaller number of elements in the 3-D reward tensor are removed when constructing the 2-D subproblem, i.e., since a smaller number of elements are removed, there is a higher likelihood that a better solution remains. For these reasons, the remaining experiments used the $m$-best 3-D assignment algorithm with the relaxation
Algorithm 5 3D assignment subroutine

1: function 3DAssign($w_{ijk}, (A^*_n, \mathcal{P}, f^*_n))$
2: $f^* = -\infty$; lb = $-\infty$; $q^* = \infty$; maxIter = 20
3: MAX = true, $n_3 = R$
4: $FixList, v, \Phi \leftarrow A^*_n$
5: for curlIter = 1 to maxIter do
6: $C = \max_k (w_{ijk} - \mu_k)$
7: for $<i^*, j^*, k^*> \in A^*_n.FixList$ do
8: $C[i^*, j^*] = w[i^*, j^*, k^*]$
9: end for
10: if $\Phi == \emptyset$ then
11: $(\Phi, u, v, \phi) = JVC(C, MAX)$
12: else
13: $(\Phi, u, v, \phi) = Augment(C, \Phi, v, MAX )$
14: end if
15: $q^* = \min(q, \phi + n_3 \sum_k (\mu_k))$
16: for each row do
17: $T[\text{row}] = w[\text{row}, \Phi[\text{row}], k] \ \forall k$
18: end for
19: $(\Omega, \omega) = \text{Transportation}(T, MAX)$
20: if $\omega \geq \text{lb}$ then
21: lb = $\omega$
22: $f^* = \text{lb}$
23: end if
24: gap = $|q^* - f^*|$
25: if gap $\leq 0.05$ then
26: return
27: end if
28: Update Lagrangian multiplier
29: end for
30: end function

of constraint (2.4) only.
Table 2.2
10 Monte Carlo Runs for different Lagrangian Relaxation methods

<table>
<thead>
<tr>
<th>MC</th>
<th>Gap</th>
<th>Objective Function Values</th>
<th>Runtime (CPU s)</th>
<th>Gap</th>
<th>Objective Function Values</th>
<th>Runtime (CPU s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.015</td>
<td>59.021</td>
<td>0.065</td>
<td>0.003</td>
<td>59.627</td>
<td>0.040</td>
<td>1.628</td>
</tr>
<tr>
<td>2</td>
<td>0.010</td>
<td>59.281</td>
<td>0.064</td>
<td>0.004</td>
<td>59.599</td>
<td>0.045</td>
<td>1.422</td>
</tr>
<tr>
<td>3</td>
<td>0.013</td>
<td>59.100</td>
<td>0.054</td>
<td>0.003</td>
<td>59.618</td>
<td>0.038</td>
<td>1.413</td>
</tr>
<tr>
<td>4</td>
<td>0.011</td>
<td>59.219</td>
<td>0.053</td>
<td>0.002</td>
<td>59.664</td>
<td>0.033</td>
<td>1.613</td>
</tr>
<tr>
<td>5</td>
<td>0.011</td>
<td>59.209</td>
<td>0.053</td>
<td>0.001</td>
<td>59.764</td>
<td>0.031</td>
<td>1.692</td>
</tr>
<tr>
<td>6</td>
<td>0.010</td>
<td>59.249</td>
<td>0.053</td>
<td>0.002</td>
<td>59.699</td>
<td>0.032</td>
<td>1.664</td>
</tr>
<tr>
<td>7</td>
<td>0.009</td>
<td>59.335</td>
<td>0.070</td>
<td>0.002</td>
<td>59.719</td>
<td>0.031</td>
<td>2.279</td>
</tr>
<tr>
<td>8</td>
<td>0.013</td>
<td>59.124</td>
<td>0.057</td>
<td>0.005</td>
<td>59.513</td>
<td>0.034</td>
<td>1.700</td>
</tr>
<tr>
<td>9</td>
<td>0.012</td>
<td>59.184</td>
<td>0.053</td>
<td>0.008</td>
<td>59.347</td>
<td>0.034</td>
<td>1.563</td>
</tr>
<tr>
<td>10</td>
<td>0.010</td>
<td>59.290</td>
<td>0.049</td>
<td>0.003</td>
<td>59.632</td>
<td>0.036</td>
<td>1.344</td>
</tr>
</tbody>
</table>

2.6.2 JVC vs. Auction Algorithm

To measure and quantify which algorithms best solve the 2-D assignment and transportation problems within the 3-D assignment problem, we compared the runtimes of the 3-D assignment algorithm when using the JVC or the auction algorithms for the 2-D assignment problem, and Transauction or simplex-based transportation algorithms for the transportation problem. A tensor was generated with elements sampled from a uniform distribution in the interval [0,1] for tensor sizes ranging from 30×30×8 to 60×60×8 with increments of $N = 5$. Any combination of the 2-D assignment algorithms with the transportation algorithms resulted in the same assignments and objective function values. An example of the objective function values of a sample tensor of dimension 30×30×8 is shown in Fig. 2.4 when the algorithm was run to
obtain the top $10^4$ solutions. As shown in Fig. 2.4, even when $10^4$ assignments were obtained, the maximum and minimum objective function values obtained from the assignment solutions had minimal variation and the difference was relatively small for all tensor dimensions tested; however, as shown in Fig. 2.5, the $m$-best 3-D assignment algorithm, which invoked the JVC algorithm was, on average, 3 times faster when compared to the case when the auction algorithm was used. The RELAX-IV algorithm was used to solve the transportation problem in this experiment.

2.6.3 Transportation vs. Transauction vs. RELAX-IV Algorithm

Similar tests were performed to evaluate the best algorithm to solve the transportation problem. Assuming that the JVC algorithm would be invoked to solve the 2-D
Figure 2.5: The CPU runtime for the JVC and auction algorithms were compared as a function of varying tensor dimensions. The JVC algorithm consistently outperformed the auction algorithm.

The assignment portion of the problem, Fig. 2.6 demonstrates that the simplex-based transportation algorithm was significantly slower compared to both the Transauction and RELAX-IV algorithms. In general, the RELAX-IV algorithm had the fastest runtime speed. The maximum observed speedup of RELAX-IV in comparison to the simplex-based transportation and the Transauction algorithms was 21.4 and 2.4, respectively. Overall, the RELAX-IV algorithm dominated both the simplex-based transportation and the Transauction approaches to the transportation problem, on average solving it nearly 17 and 1.6 times faster, respectively. Based on these findings, the JVC and RELAX-IV algorithms were selected to solve the 2-D assignment and transportation problems within the $m$-best 3-D assignment problem, respectively, for the remaining computational experiments. We optimized the $m$-best 3-D assignment
Figure 2.6: The CPU runtime for the Transauction and simplex-based transportation algorithms were compared as a function of differing tensor dimensions. The Transauction algorithm remained relatively unaffected by the increase in the reward tensor size, while the transportation algorithm took orders of magnitude more time to find the same assignments.

algorithm via the methods detailed in Section 2.4

2.6.4 Solution quality evaluation for decomposition methods

A sample test tensor of dimension $30 \times 30 \times 8$ with elements uniformly distributed in the interval $[0, 1]$ was used to measure the solution quality and to compare the simulation runtimes of the $m$-best 3-D assignment algorithm when exploiting different combinations of the search space decomposition optimization methods. As shown in Fig. 2.7, the optimization combinations of A+B(iii) and A+B(iv) resulted in approximately a 10% reduction in the solution quality compared to the original Murty’s proposed search space decomposition method. This is because the dual value
in our problem setup did not serve as an accurate estimate of the initial upper bound. All other combinations of optimization methods were comparable to the Murty’s search space decomposition. Therefore, optimization method combinations A+B(iii) and A+B(iv) were removed from the remaining tests.

2.6.5 Runtime comparison for decomposition methods

Similar tests were performed on all the remaining combinations of optimization methods for tensor sizes varying from $30 \times 30 \times 8$ to $60 \times 60 \times 8$ with an increment of $N = 5$. Table 2.3 shows the simulation runtime in CPU seconds for $10^4$ solutions and for the various search space optimization combinations, given a sample test tensor for each incremented dimension. Methods A, A+B(i) and A+B(ii) on average ran 20%, 52% and 32% slower, respectively, within the $m$-best 3-D assignment algorithm, as
compared to the original Murty’s search space decomposition. As shown in Fig. 2.8, methods A+B(i)+C and A+B(ii)+C were able to obtain speedups with very minimum variation in the objective function values originally found by Murty’s search space decomposition for all tensor sizes except that of dimension 60×60×8. The reason for such a slow down is explained later in Section 2.6.6. Furthermore, combinations A+B(i)+C and A+B(ii)+C were able to obtain objective function values slightly better (higher) than the proposed method by Murty (on the order of 10^{-6}). This phenomenon is due to the Lagrangian relaxation algorithm’s approximation of the 3-D assignment problem. The search space decomposition method is suboptimal when applied to the 3-D assignment problem (due to the suboptimal nature of the Lagrangian relaxation algorithm), and so from our analysis we observed that, through the particular optimization method combinations of A+B(i)+C and A+B(ii)+C, better feasible solutions were found. These methods were also significantly faster, offering an average of 2.1 and 2.4 speedup, respectively, as illustrated in Fig. 2.8. To investigate these combinations more thoroughly, Monte Carlo runs were performed on these two combinations only.

2.6.6 Scalability with N

To measure both the overall scalability and consistency, 10 Monte Carlo runs were performed for each tensor size varying from 30×30×8 to 60×60×8 in increments of N = 10 and using the two specific optimization method combinations of A+B(i)+C and A+B(ii)+C. Each test tensor was generated with elements uniformly distributed in the interval [0,1] and 10^4 solutions were obtained for each tensor. In each run, the objective function values and the simulation runtime, were monitored and compared
### Table 2.3
Simulation runtime in CPU seconds for various combinations of decomposition methods

<table>
<thead>
<tr>
<th>Tensor Size</th>
<th>Original Murty</th>
<th>A</th>
<th>A+B(i)</th>
<th>A+B(ii)</th>
<th>A+B(i)+C</th>
<th>A+B(ii)+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>30×30×8</td>
<td>139.17</td>
<td>157.32</td>
<td>193.45</td>
<td>124.30</td>
<td>51.69</td>
<td>44.96</td>
</tr>
<tr>
<td>35×35×8</td>
<td>189.03</td>
<td>230.78</td>
<td>400.78</td>
<td>284.66</td>
<td>74.95</td>
<td>62.67</td>
</tr>
<tr>
<td>40×40×8</td>
<td>235.56</td>
<td>293.40</td>
<td>429.86</td>
<td>391.12</td>
<td>76.85</td>
<td>73.50</td>
</tr>
<tr>
<td>45×45×8</td>
<td>178.70</td>
<td>226.16</td>
<td>398.22</td>
<td>337.41</td>
<td>102.21</td>
<td>87.33</td>
</tr>
<tr>
<td>50×50×8</td>
<td>288.11</td>
<td>370.07</td>
<td>803.69</td>
<td>491.81</td>
<td>99.15</td>
<td>79.73</td>
</tr>
<tr>
<td>55×55×8</td>
<td>244.75</td>
<td>303.14</td>
<td>489.72</td>
<td>477.44</td>
<td>207.71</td>
<td>204.60</td>
</tr>
<tr>
<td>60×60×8</td>
<td>152.72</td>
<td>209.78</td>
<td>463.33</td>
<td>427.37</td>
<td>309.65</td>
<td>232.06</td>
</tr>
</tbody>
</table>

against both combinations, as well as with respect to Murty’s decomposition method.

Fig. 2.9 shows the percentage error of the two optimization methods as compared to the original Murty’s search space decomposition. The average percentage error increased with each increment in the tensor dimensions. Overall, the optimization method combination of A+B(ii)+C had a lower median compared to the combination of A+B(i)+C; however, the combination of variants A+B(i)+C had less variation with respect to the average percentage error. Table 2.4 details the minimum, maximum, and average runtimes observed in CPU seconds for the Monte Carlo runs. The method A+B(ii)+C had the fastest runtime, as shown in Fig. 2.10, with an observed maximum average of 3.1 speedup over Murty’s search space decomposition method, while having 2.14 speedup on average, when averaged over all Monte Carlo runs. The tensor of dimension 60×60×8 resulted in a slow down of 32% and 25%, respectively, with optimizations A+B(i)+C and A+B(ii)+C.

In the fully optimized m-best 3-D assignment algorithm, there exists a tradeoff
Figure 2.8: Percentage error compared against the speedup for the combinations of optimization methods tested for all tensor sizes, varied from $30 \times 30 \times 8$ to $60 \times 60 \times 8$ with an increment of $N = 5$.

(when $N \approx 55$) in computation time between obtaining a feasible solution and the $m$-best optimization methods (e.g., partitioning or sorting), as seen in Fig. 2.10. The increase in dimension $N$ does not necessarily mean an increase in the computation time of the 3-D assignment algorithm, since both optimization methods $A+B(i)+C$ and $A+B(ii)+C$ reduce the frequency of calling the 3-D assignment algorithm; however, partitioning and/or sorting the larger subproblems may become more difficult. A more favorable speedup may be observed if the algorithms were to be implemented in a fast object oriented-programming language. Overall, for a $30 \times 30 \times 8$ tensor, the $m$-best 3-D assignment algorithm utilizing optimization method $A+B(ii)+C$ took an average of 4.9 milliseconds to obtain a single solution to the 3-D assignment problem.
Table 2.4
Minimum, maximum, and average runtimes in CPU seconds to obtain $10^4$ solutions

<table>
<thead>
<tr>
<th>30×30×8</th>
<th>Original Murty</th>
<th>A+B(i)+C</th>
<th>A+B(ii)+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>111.84</td>
<td>53.72</td>
<td>45.01</td>
</tr>
<tr>
<td>Mean</td>
<td>146.03</td>
<td>63.50</td>
<td>49.64</td>
</tr>
<tr>
<td>Max</td>
<td>195.41</td>
<td>76.68</td>
<td>53.55</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>40×40×8</th>
<th>Original Murty</th>
<th>A+B(i)+C</th>
<th>A+B(ii)+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>147.44</td>
<td>78.07</td>
<td>69.83</td>
</tr>
<tr>
<td>Mean</td>
<td>245.27</td>
<td>103.74</td>
<td>79.19</td>
</tr>
<tr>
<td>Max</td>
<td>284.48</td>
<td>128.17</td>
<td>92.61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>50×50×8</th>
<th>Original Murty</th>
<th>A+B(i)+C</th>
<th>A+B(ii)+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>151.15</td>
<td>98.25</td>
<td>80.24</td>
</tr>
<tr>
<td>Mean</td>
<td>247.28</td>
<td>155.13</td>
<td>139.05</td>
</tr>
<tr>
<td>Max</td>
<td>320.79</td>
<td>271.20</td>
<td>243.48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>60×60×8</th>
<th>Original Murty</th>
<th>A+B(i)+C</th>
<th>A+B(ii)+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>142.19</td>
<td>226.88</td>
<td>203.46</td>
</tr>
<tr>
<td>Mean</td>
<td>214.01</td>
<td>314.57</td>
<td>284.93</td>
</tr>
<tr>
<td>Max</td>
<td>294.43</td>
<td>586.10</td>
<td>542.90</td>
</tr>
</tbody>
</table>

2.6.7 Scalability with R

As mentioned in Sections 2.2.1 and 2.3.1.4 the value of $m_k$ should be such that $\sum_{k=1}^{R} m_k \geq N$. For problems where $m_k = R \geq N$, the 3-D assignment problem reduces to a 2-D assignment problem. We analyze the scalability of the algorithms with respect to incrementing $R$ by performing 10 Monte Carlo runs for each increment and requesting $10^4$ solutions with tensor size $N = 30$ and $R = 6, 10, 15, 20, 25, 29$ ($R = 30$ is omitted because, as mentioned earlier, this devolves into a 2-D assignment problem). Fig. 2.11 shows the relative percentage error of the two optimization
Figure 2.9: Box plot for the average percentage error (as compared to the original Murty search space decomposition method) for the optimization method combinations A+B(i)+C and A+B(ii)+C.

methods, as compared to the original Murty’s search space decomposition. The average percentage error is zero for $R \geq 10$, since the problem constraint (2.4) is less likely to be violated, and therefore, the duality gap is zero. The minimum, average and maximum runtimes are listed in Table 2.5. As shown in Fig. 2.12 the speed of the original $m$-best 3-D assignment algorithm increases significantly with $R$. A maximum speedup of 10.8 and an average speedup of 7.5 were observed when comparing the optimization method A+B(ii)+C with the original $m$-best 3-D assignment algorithm. The total number of arcs input to the RELAX-IV algorithm is bounded above by $R^3$; hence increasing $R$ has an exponential impact on the complexity of the problem solved by the algorithm and, in turn, the CPU runtime of the original $m$-best 3-D assignment. On the other hand, the optimized $m$-best 3-D assignment is able to
reduce the need for the 3-D assignment routine invocation and, therefore, is able to obtain $10^4$ solutions in a relatively short amount of time ($< 5$ minutes). The tensor of dimension $30\times30\times6$ had an increase in average CPU runtime compared to a tensor of dimension $30\times30\times10$ when considering the optimized methods $A+B(i)+C$ and $A+B(ii)+C$. This is due to nonzero duality gap which impacts the partitioning procedure and subsequently requires more subproblems to be solved before obtaining all $m$-best solutions. Intuitively, due to the nature of the problem, a tensor of dimension $30\times30\times6$ is more likely to violate constraint (2.4).
Table 2.5
Minimum, maximum, and average runtimes in CPU seconds to obtain $10^4$ solutions

<table>
<thead>
<tr>
<th>Size</th>
<th>Original Murty</th>
<th>A+B(i)+C</th>
<th>A+B(ii)+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$30 \times 30 \times 6$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>83.33</td>
<td>78.24</td>
<td>62.51</td>
</tr>
<tr>
<td>Mean</td>
<td>103.77</td>
<td>93.48</td>
<td>79.00</td>
</tr>
<tr>
<td>Max</td>
<td>119.63</td>
<td>108.27</td>
<td>93.91</td>
</tr>
<tr>
<td>$30 \times 30 \times 10$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>161.03</td>
<td>56.47</td>
<td>42.03</td>
</tr>
<tr>
<td>Mean</td>
<td>232.45</td>
<td>65.78</td>
<td>51.33</td>
</tr>
<tr>
<td>Max</td>
<td>276.15</td>
<td>73.7</td>
<td>56.32</td>
</tr>
<tr>
<td>$30 \times 30 \times 15$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>477.91</td>
<td>87.00</td>
<td>68.20</td>
</tr>
<tr>
<td>Mean</td>
<td>640.89</td>
<td>102.21</td>
<td>75.71</td>
</tr>
<tr>
<td>Max</td>
<td>886.40</td>
<td>113.97</td>
<td>84.40</td>
</tr>
<tr>
<td>$30 \times 30 \times 20$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>747.44</td>
<td>133.35</td>
<td>108.31</td>
</tr>
<tr>
<td>Mean</td>
<td>1077.13</td>
<td>150.95</td>
<td>115.24</td>
</tr>
<tr>
<td>Max</td>
<td>1587.85</td>
<td>172.55</td>
<td>125.67</td>
</tr>
<tr>
<td>$30 \times 30 \times 25$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>1519.77</td>
<td>186.74</td>
<td>162.13</td>
</tr>
<tr>
<td>Mean</td>
<td>1902.02</td>
<td>218.85</td>
<td>176.63</td>
</tr>
<tr>
<td>Max</td>
<td>2507.46</td>
<td>230.53</td>
<td>187.83</td>
</tr>
<tr>
<td>$30 \times 30 \times 29$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td>1704.44</td>
<td>265.57</td>
<td>235.14</td>
</tr>
<tr>
<td>Mean</td>
<td>2572.39</td>
<td>298.28</td>
<td>249.87</td>
</tr>
<tr>
<td>Max</td>
<td>3343.91</td>
<td>330.11</td>
<td>268.99</td>
</tr>
</tbody>
</table>
Figure 2.11: Box plot for the average percentage error (as compared to the original Murty search space decomposition method) for the optimization method combinations $A+B(i)+C$ and $A+B(ii)+C$, where $N = 30$. 
Figure 2.12: The average CPU runtimes for 10 Monte Carlo runs with each increment of $R$ for the two optimization method combinations tested.
Chapter 3

Context-Aware Dynamic Asset Allocation for Maritime Surveillance Operations

3.1 Introduction

3.1.1 Motivation

The illicit drug trade is an extremely profitable industry and it is estimated that the consumers in the United States of America alone spend as much as 150 billion USD per year on black market drugs. Of this, it is estimated that 37 billion USD is spent on cocaine alone. It is a problem of national, and increasingly international, concern [171], [172]. This problem increased exponentially with the advent of narco-terrorism and the prospect of terrorists using narcotics smuggling techniques to transport terrorists or weapons of mass destruction into the country. Given the
reduction in the national resources allocated to the counter-narcotics threat, it is of paramount importance that smarter and faster decision support tools that integrate a wide variety of information are developed to assist in this challenge of using less to accomplish more. To do so requires effective hybrid human-machine systems.

The US Navy has shown a growing interest in mixed-initiative human-machine systems and mastering information dominance for effective context-driven operations [173]. To do so requires the transfer of the right data from the right sources in the right context to the right decision maker (DM) at the right time for the right purpose—a concept known as 6R [163]. If a dynamically developing operational context can be understood by the DM, appropriate courses of action can be carried out, given the unfolding events. In the context of maritime operations, DMs must assimilate information from a multitude of sources before making decisions on the strategy to be followed each day. If the DMs are better informed about what to expect given the currently accessible data, as well as what they might expect in the case of unforeseen events, effective decisions can be made on the courses of action.

Currently, much planning for narcotics seizures is performed by humans interpreting large amounts of data, including weather forecasts, intelligence, and recently reported contacts of interest. Each day, the targeting analysts must process and interpret all of this data and agree upon a course of action amounting to where limited detection aircraft and interdiction vessels should be allocated to disrupt the maximum amount of shipments over a multi-day planning cycle. The consolidation of large amounts of data and possible strategies into a single asset allocation optimizer is beneficial for both algorithmic purposes and human understanding. To support this transition to a human-machine collaborative mode of operation, we have developed an optimization-based modeling framework and the associated decision support software tool for
dynamic surveillance and interdiction resource management in counter-smuggling operations. This tool, named COAST or Courses Of Action Simulation Tool [160], and the corresponding algorithms are intended to support targeting analysts in identifying high probability areas of smuggler presence and to proactively develop a set of high-value courses of action.

The counter-smuggling problem presented in this chapter is viewed as a moving horizon stochastic control problem, as illustrated in Fig. 3.1, specifically from a strategic operations standpoint, i.e., decision making with regards to a schedule to follow for the upcoming time horizon. Here, each block is an entity, such as a decision maker (DM), sensor, or asset, and the link from each block represents the outcome of the block and its impact or influence on the next block. The problem can be decomposed into surveillance and interdiction asset allocation subproblems. This chapter focuses on the surveillance component, where DMs (also termed targeteers) are responsible for allocating multiple aircraft (viz., P-3 Orions (manned)) over a finite time horizon in an effort to detect the transportation of contraband. The interdiction
component, detailed in [161], involves the allocation of multiple heterogeneous surface assets (viz., Navy ships, Coast Guard cutters), to disrupt multiple drug smugglers of varying types, similar to that which is addressed in this chapter. The DMs in Fig. 3.1 choose which surveillance assets to allocate to which target(s) (smugglers) based on the target type and intelligence forecasting the target’s trajectory (specified in the form of probability of activity (PoA) surfaces [67,159]). After allocated assets attempt to search for potential targets, the mission environment changes due to any target detection that may occur or due to weather changes. These environment changes are recorded by sensors and operators, processed, and sent back to the DMs in the form of target types and tracks, and are combined into an updated PoA surface, providing a new forecast for the remainder of the planning time horizon. The process then repeats. Ideally, the results of this chapter feed that of [161] for coordinated smuggler detection and interdiction.

3.1.2 Related Research

The surveillance mission involves the search, detection, tracking and identification of potential smugglers within a large geographic region, which plays an essential role in the counter-smuggling operation. Airborne surveillance assets (e.g., helicopters, maritime patrol aircraft) are highly efficient at determining the sea surface traffic information. However, in a real world scenario, there is typically a limited number of surveillance assets and a large sea surface area that needs to be surveilled. The study of how to most effectively employ limited resources to locate an object, whose location is not precisely known, falls under the rubric of search theory.

The earliest foundations of search theory were built by Koopman [92] to aid the U.S.
Navy in efficiently locating enemy submarines during World War II, which was further generalized in [93]. There are two major categories of search theory: 1) the optimal allocation of effort problem, and 2) the best track problem [130]. For the optimal effort allocation problem, Blachman and Proschan [24] derived an optimum search pattern for a generalized problem of finding an object in one of $n$ boxes. Pollock [143] introduced a Bayesian approach to the optimal allocation problem, where allocation decisions are made sequentially based on observations up to the current time in order to minimize the expected cost of searching to satisfy a specified probability of detection. Charnes and Cooper [33] applied convex programming, along with the Kuhn-Tucker conditions, for the optimum distribution of effort computation. In this chapter, we adopt Charnes and Cooper’s method to compute the effort required for the optimal search in a discretized map.

Stone [166] made use of the calculus of variations, convexity properties, and generalized Lagrange multiplier techniques to formulate a systematic treatment of search theory. For the best track problem, Lukka [104] worked out the theory of optimal search for stationary targets, targets whose motion is known, and targets whose motion is almost known. The method relies on the theory of optimal control. Mangel [108] extended Lukka’s algorithms with the option of incorporating a detection rate that is either independent of or dependent on velocity.

In recent years, the problem of drug surveillance has been formulated from a variety of viewpoints. For example, Washburn [179] formulated the surveillance problem as a two-person zero-sum game and Pfeiff [137] applied search theory to a defender-attacker optimization model that maximizes the defender’s probability of success. Royset and Wood approach the problem as a network flow problem, wherein an interdictor must destroy a set of arcs on a network to minimize both the interdiction cost and minimize
the maximum flow of smugglers [153]. Jacobson [81] formulates the problem as a multiple traveling salesman problem with the objective of minimizing the overall search route cost for multiple platforms that visit every search location. Ng and Sancho [129] developed a dynamic programming method to solve the surveillance problem. However, the dynamic programming approach suffers from the curse of dimensionality for large problems and, consequently, near-optimal approximations are needed. A common way to overcome this curse is by approaching the problem via approximate dynamic programming with policy iteration as in [76], where they frame the problem in terms of stochastic control with partially observable Markov decision processes. Kress et al. [95] examine a discrete-time and discrete-space stochastic dynamic programming approach to coordinate the efforts of a single aerial search asset and a single surface interdiction asset. Other approaches, including the formulation of the surveillance problem as a resource-dependent orienteering problem [29, 141, 142], wherein reward depends on the resource expended at each visited node, have been investigated.

Optimal search problem formulations have become versatile in their ability to account for multiple cooperating searchers, multiple targets with different characteristics, as well as environmental effects on the search [128, 152, 155, 182]. For example, arc inspection is based on the inverse of the probabilities of detection as opposed to PoA surfaces accounting for weather and intelligence in [4, 67, 159]. Byers [27] extended the network modeling approach to drug interdiction by including Bayesian updating of the PoA surface. He considered a scenario with one unmanned aerial vehicle and one ground-based interceptor to interdict multiple targets with different deadlines. Bessman [22] developed a defender-attacker optimization model that uses the PoA surfaces as the basis for asset allocation against smugglers. He formulated a stochastic shortest path problem and represented smuggler behavior as the output of an all-to-one
label-correcting temporal dependence instead of one-step dependence. Three different sensor types (one interdiction and two surveillance) are considered for allocation to prosecute one type of target (among three possible). In this defender-attacker model, smugglers are assumed to have imperfect knowledge of possible sensor locations and are given the ability to modify their behavior in response to this information.

### 3.1.3 Chapter Organization

Similar to Pietz and Royset [141], we also discretized our maritime map. We adopt Charnes and Coopers’s method [33] to compute the effort required for optimal search in a discretized map. Our novel algorithmic contributions are the following:

1): Fast 1- and 2-step lookahead approximate dynamic programming algorithms for maritime surveillance composed of heterogeneous assets, heterogeneous targets, each of which is carrying not necessarily the same amount of contraband. Our algorithms exploit the fusion of intelligence and weather information available in the probability of activity (PoA) surfaces.

2): We measure the utility of our approach by way of comparison with more traditional branch-and-cut algorithms to solve the surveillance problems. We develop two variations of the approximate dynamic programming-based surveillance asset allocation algorithms, wherein real-world constraints on the assets (e.g., endurance and rest time) are explicitly considered.

The chapter is organized as follows. Section 3.2 describes the problem and the technical challenges addressed in the development of allocation algorithms underlying our decision support tool. In Section 3.3 we discuss solution approaches, including
exhaustive and greedy branch-and-cut and approximate dynamic programming. In Section 3.4, we present simulation results as applied to a benchmark scenario that has multiple targets, multiple surveillance assets and parameters that have multiple levels of uncertainty. We additionally conduct and present results from our sensitivity analysis relating to the scalability and performance of our solution approaches in a realistic mission scenario.

3.2 Problem Model and Formulation

3.2.1 Problem Definition and Solution Architecture

The complete maritime surveillance and interdiction problem is one of maritime drug trafficking disruption in the East Pacific Ocean and the Caribbean Sea. The general mission consists of two components: 1) surveillance (the detection, tracking, and identification of contacts of interest) and, 2) interdiction (the interception, investigation, and potential apprehension and repatriation of smugglers). In response to the need for information fusion, we proposed a decision support system (DSS) in [160], named COAST, to host and utilize algorithms to provide auxiliary support to JIATF-South targeteers. We proposed different forms of visualizations to enable DMs to understand the behavior of our algorithms and the presently evolving context, while also providing functionality for human input and interaction in order to effectively integrate both humans and decision support algorithms for mixed-initiative planning. The information flow for the complete maritime interdiction problem is illustrated in Fig. 3.2.

In COAST, we solve a moving horizon dynamic resource management problem for both surveillance and interdiction operations based on user-defined mission parameters.
We then provide suggested courses of action (COAs) that the DMs can interact with, adjust and fine tune to analyze various “what-if” scenarios and to obtain a satisfactory allocation. Visual and computational analytics are provided to communicate the reasons behind our algorithm’s behavior. From Fig. 3.2 continuously updated PoA surfaces (see Fig. 3.3 for an example), representing the posterior probabilities of smugglers’ presence, constitute the sufficient statistics for decision making [18] – that is, COAST does not need to know how specific intel or meteorology and oceanography (METOC) features, e.g., uncertainty associated with a drug trafficker, wave heights, currents, etc., and how these two inputs, along with asset and target models, are combined to produce the PoA surface. A targeteer can fine tune the allocations, the resulting COAs are executed, and observations from surveillance and interdiction assets are sent back to the reachback cell in the form of situational reports or SITREPs (e.g., detections or non-detections) which are used to update the PoAs. The targeteer can specify multiple objective functions. The objectives considered and analyzed in this chapter are:

**O1**: Maximize the normalized weight of the contraband detected (normalized by the total possible amount of contraband)

**O2**: Maximize the normalized number of detections (normalized by the total possible number of cases)

**O3**: Maximize the normalized number of smugglers detected (normalized by the total possible number of smugglers)

Let $\alpha_j$ and $\rho_j$ denote the expected contraband weight and expected number of smugglers for case $j$. Let $C$ be the total number of cases (i.e., predicted smuggler
tracks) to be searched. Then, the normalized priority weights for Objectives $O1–O3$, respectively, are as follows:

$$\lambda_j = \frac{\alpha_j}{\sum_{g=1}^{C} \alpha_g} \quad (3.1)$$

$$\lambda_j = \frac{1}{C} \quad (3.2)$$

$$\lambda_j = \frac{\rho_j}{\sum_{g=1}^{C} \rho_g} \quad (3.3)$$

### 3.2.2 Problem Formulation

The notation used in this chapter is listed in Table 3.1.
### Table 3.1
**Summary of Notations**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>Total number of surveillance assets</td>
</tr>
<tr>
<td>$A_j$</td>
<td>Total area to be searched for case $j$</td>
</tr>
<tr>
<td>$B_{ij}$</td>
<td>Great Circle distance from the base of asset $i$ to the centroid of case $j$</td>
</tr>
<tr>
<td>$C$</td>
<td>Total number of cases</td>
</tr>
<tr>
<td>$CPoSD(i, j)$</td>
<td>Cumulative probability of successful detection for a given asset $i$ allocated to case $j$</td>
</tr>
<tr>
<td>$d_{it}$</td>
<td>Landing time for asset $i$’s $t$-th flight</td>
</tr>
<tr>
<td>$i$</td>
<td>Surveillance asset index</td>
</tr>
<tr>
<td>$j$</td>
<td>Case index</td>
</tr>
<tr>
<td>$k$</td>
<td>Time epoch index</td>
</tr>
<tr>
<td>$K$</td>
<td>End of planning time horizon</td>
</tr>
<tr>
<td>$L_i$</td>
<td>Endurance of asset $i$</td>
</tr>
<tr>
<td>$PoA(q, k, j)$</td>
<td>Likelihood that a smuggler belonging to case $j$ is located in a cell $q$ at time $k$</td>
</tr>
<tr>
<td>$R_i$</td>
<td>Downtime of asset $i$</td>
</tr>
<tr>
<td>$s_{it}$</td>
<td>Remaining search time available within the current sortie for asset $i$</td>
</tr>
<tr>
<td>$S_{ij}$</td>
<td>Sweepwidth of asset $i$ searching for target $j$</td>
</tr>
<tr>
<td>$t_{ij}$</td>
<td>Travel time for traversing the distance $B_{ij}$</td>
</tr>
<tr>
<td>$v_{ia}$</td>
<td>Travel speed of asset $i$</td>
</tr>
<tr>
<td>$v_s$</td>
<td>Search speed of asset $i$</td>
</tr>
<tr>
<td>$w_{ijk}$</td>
<td>Reward of allocating asset $i$ to case $j$ at time $k$</td>
</tr>
<tr>
<td>$x_{ijk}$</td>
<td>Binary decision variable of allocating asset $i$ to case $j$ at time epoch $k$</td>
</tr>
<tr>
<td>$\lambda_j$</td>
<td>Priority weight of case $j$</td>
</tr>
<tr>
<td>$\tau_{i\ell}$</td>
<td>Departure time for asset $i$’s $\ell$-th flight</td>
</tr>
<tr>
<td>$\gamma(i, j, \ell)$</td>
<td>The set of search time indices for asset $i$ assigned to case $j$ for the $\ell$th flight</td>
</tr>
</tbody>
</table>

### 3.2.2.1 PoA Surface

The foundation for each asset allocation solution is the PoA surface over multiple time epochs. The PoA surface is the result of combining METOC information with actionable intelligence with regards to uncertain smuggler departure point(s),
departure times, waypoint(s), destination(s), and their behavior on the ocean. The
spatio-temporal probability surface, PoA, is calculated as the joint probability of two
discrete random events: 1) the case \( j \), with a corresponding binary random variable \( C_j \), i.e., how trustworthy the intelligence source is regarding a target, and 2) the target
(corresponding to case \( j \) at a location \( q \) at time epoch \( k \), with a corresponding binary
random variable \( X(q,k,j) \), i.e., given that the case \( j \) exists, the probability that the
target exists at a location \( q \) at time \( k \). The probability surface PoA is indexed by a
location \( q \), time \( k \), and case \( j \), and is defined in (3.4)–(3.7).

\[
PoA(q, k, j) = P(\mathcal{C}_j = 1 \cap X(q, k, j) = 1) \quad (3.4)
\]
\[
= E \{ \mathcal{C}_j \cdot X(q, k, j) \} \quad (3.5)
\]
\[
= E_{\mathcal{C}_j} \left\{ \mathcal{C}_j \cdot E_{X(q,k,j) | \mathcal{C}_j} (X(q, k, j) | \mathcal{C}_j) \right\} \quad (3.6)
\]
\[
= \sum_{c_j = \{0,1\}} c_j \cdot P(\mathcal{C}_j = c_j) \cdot \left( \sum_{h = \{0,1\}} h \cdot P(X(q, k, j) = h | \mathcal{C}_j = c_j) \right), \quad (3.7)
\]

where we separate the expectation in (3.6) based on the law of total expectation/iterated expectations.

We assume that \( P(\mathcal{C}_j = 1) = 1 \), that is, the intelligence sources are always correct
with 100% certainty. Then, (3.7) reduces to,

\[
PoA(q, k, j) = P(X(q, k, j) = 1) \quad (3.8)
\]

Therefore, \( PoA(q, k, j) \) is a number that refers to the likelihood that a smuggler,
belonging to case \( j \), is located in a cell \( q \) at time \( k \). The PoA surfaces are computed as
detailed in [67] and represent all the relevant information for effective asset allocation. The DM can specify how many planning epochs to optimize over based on these PoA surfaces and the objective function to be optimized. A typical PoA surface $PoA(q,k,j)$, summed over all $k$, is shown in Fig. 3.3.

### 3.2.2.2 Optimal Search Effort Calculation

We assume the optimum distribution of search effort is known based on the model in [33]. Let $p_{jkq}$ denote the PoA of target $j$ in cell $q$ at time $k$. We first rank the nonzero PoA cells in decreasing order such that $p_{jk[1]} \geq p_{jk[2]} \geq \ldots$, where $[\kappa]$ denotes the $\kappa^{th}$ largest nonzero PoA cell. Let the total available effort to be expended by asset $i$ to search case $j$ be $\Phi_{ij}$. A critical threshold is then calculated to narrow the problem space and eliminate PoA cells not worth searching, by first finding an $n$ that satisfies the following inequality [33].

$$\sum_{v=1}^{n} v \left[ \ln p_{jk[v]} - \ln p_{jk[v+1]} \right] > \Phi_{ij} \quad (3.9)$$
Then, the critical probability, $\rho_{ijk}$, corresponding to the search of case $j$ by asset $i$ at time $k$, is as in (3.10).

$$\rho_{ijk} = \frac{1}{n} \left( \sum_{v=1}^{n} v \left( \ln p_{jk[v]} - \ln p_{jk[v+1]} \right) - \Phi_{ij} \right) + \ln p_{jk[n+1]} \quad (3.10)$$

We then select all the cells corresponding to case $j$ which have a probability of activity greater than the critical probability found in (3.10). This reduces the number of potential cells that need to be searched for each case $j$ by asset $i$. We then compute the patrol box that maximally covers the high probability cells for each case. The allocation of assets to patrol boxes is the subject of the optimization problem discussed next.

### 3.2.2.3 Optimization Problem

The case regions are labeled by aggregating the PoA surfaces over a discrete planning time period of length $K$ (e.g., 72 hours). Let us assume a moving horizon frame of reference, where $k = 0$ corresponds to the current time period of unit length ($\Delta = 1$ hour); $k = 1$ corresponds to the first planning period, and $k = K$ corresponds to the final period to be planned for. Let $A$ be the total number of surveillance assets, $C$ be the total number of cases and $q \in Q(j)$ be the set of cells in the patrol box for case $j$ as determined by the optimal search effort calculation algorithm. The size of the patrol box depends on the concept of operations and is assumed known. Let $w_{ijk}$ be the probability of successful detection (PoSD), which is the product of the PoA surface and the probability of detection (PD) when asset $i$ is assigned to search for
case \( j \) at time \( k \). That is,

\[
w_{ijk} = \sum_{q \in Q(j)} P_oA(q,k,j) PD(i,j,k),
\]

(3.11)

where

\[
PD(i,j,k) = 1 - e^{-\frac{S_{ijk}v_i\Delta}{d_j}}
\]

(3.12)

is the probability that asset \( i \) detects case \( j \) during the \( k^{th} \) time epoch interval (the \( PD(i,j,k) \) can only be collected at the end of the \( k^{th} \) time epoch interval). Let us assume that each asset travels to the search region at a speed \( v^a_i \) and searches in the search region at a speed \( v^s_i \). The PD equation is adopted from Koopman’s random search formula [92], and offers a lower bound on the probability of detection; advanced models may be used in place of (3.12) as in [166]. Here, \( S_{ijk} \) is the sweepwidth of asset \( i \) searching for case \( j \) at time epoch \( k \), and \( \Delta \) is the inter-epoch interval (=1 hour in this chapter).

Let \( B_{ij} \) represent the geodesic\(^1\) distance that asset \( i \) must traverse from its base to the centroid of case \( j \). The time it takes to traverse \( B_{ij} \), denoted by \( t_{ij} \), is given by,

\[
t_{ij} = \left\lceil \frac{B_{ij}}{v_i^a} \right\rceil,
\]

(3.13)

\(^1\)The geodesic distance is the shortest distance between two points on the surface of a sphere.
\[ \tau_{i\ell} = \begin{cases} k, & 0 < k \leq K \text{ if } i \text{ is assigned to a case during the } \ell^{th} \text{ flight} \\ \infty, & \text{otherwise} \end{cases} \] (3.14)

A similar definition applies to \( d_{i\ell} \). For each flight, the total search and travel time for each asset from its corresponding base to each case must not exceed the asset’s endurance, \( L_i \) (in hours), and, upon flight completion, it must rest for \( R_i \) consecutive hours before it can be scheduled to depart for the next search box. The assets are assumed to be manned aircraft with an associated rest time for the pilot; additionally, each aircraft requires periodic maintenance and refueling. The minimum time it may take for an asset to become available again for search is \( L_i + R_i \). Note that there is no feasible asset allocation for a case \( j \) and asset \( i \) if \( 2t_{ij} \geq L_i \), i.e., the total round trip travel time for a search region is greater than the maximum aloft time \( L_i \). With \( PoSD \) defined as in (3.11), the cumulative probability of successful detection (\( CPoSD \)) for a given asset \( i \) is

\[ CPoSD(i, j) = 1 - \prod_{k=1}^{K} (1 - w_{ijk}x_{ijk}) , \] (3.15)

where \( x_{ijk} \) is a binary decision variable such that \( x_{ijk} = 1 \) if asset \( i \) is assigned to case \( j \) at time epoch \( k \), and 0, otherwise. The total reward that asset \( i \) can collect over the planning time horizon is then

\[ r_i = \sum_{j} \lambda_j CPoSD(i, j) , \] (3.16)

where \( \lambda_j \) is the normalized priority weight of case \( j \). We wish to solve the following problem.
\[
\begin{align*}
\max_{x_{ijk}, \tau_{i\ell}, d_{i\ell}} J &= \max \sum_{i=1}^{A} r_i \quad (3.17) \\
\text{s.t. } &\sum_i x_{ijk} \leq 1 \ \forall j, k \quad (3.18) \\
&\sum_j x_{ijk} \leq 1 \ \forall i, k \quad (3.19) \\
&d_{i\ell} - \tau_{i\ell} \leq L_i \ \forall i, \ell \quad (3.20) \\
&\tau_{i\ell+1} - d_{i\ell} \geq R_i \ \forall i, \ell \quad (3.21) \\
&\tau_{i\ell}, d_{i\ell} \in \{0, \ldots, K\} \cup \{\infty\} \ \forall i, \ell \quad (3.22) \\
&x_{ijk} \in \{0, 1\} \quad (3.23)
\end{align*}
\]

In (3.17), we assume that the surveillance asset cannot detect targets while it is en route to the patrol box. Constraints (3.18) and (3.19) ensure that no more than one case is allocated to an asset at one time. Constraint (3.20) indicates that the maximum asset aloft time must not exceed \(L_i\). Constraint (3.21) ensures that there must be a minimum downtime of \(R_i\) between asset allocations for a particular asset \(i\) and that subsequent allocations must have a departure time later than the previous one(s), if any. The problem posed in (3.17)-(3.23) is NP-hard \[59\].
3.3 Solution Approach

3.3.1 Exhaustive Branch-and-Cut

The first solution approach we investigated is the exhaustive branch-and-cut method, herein referred to as E-B&C. This method involves the enumeration and evaluation of all feasible solutions and is illustrated in Fig. 3.4 with respect to asset $i$, where each completed branch is a feasible solution with the corresponding asset-case assignments $\langle i, j^* \rangle$, given the departure times $\tau_{i\ell}, \ell = 1, 2, \ldots$ for each flight of asset $i$. We enumerate a complete feasible flight schedule over all flights for each asset $i$ and calculate the total reward $r_i$ for a given asset $i$ using (3.15) and (3.16). The schedule with the highest $r_i$ is selected to be the best assignment for asset $i$. In order to find the optimal allocation, we repeat the process mentioned above with the full permutation of asset-case combinations. The pseudocode is shown in Algorithm 6. In the Algorithm 6’s pseudocode, line 1 generates the permutation of the ordering of assets for which to start the allocation. Lines 2–5 compute the best assignment for the selected asset $i$ using B&C and updates the PoA surface accordingly to avoid duplicate assignments (this is done by setting the allocated grid cells in the PoA surface to have no reward during the assigned search time(s)). Line 7 saves all the assignment for each asset sequence generated by the permutation function. Line 8 resets the PoA surface to the originally initialized surface prior to any updates in order to compute the next sequence generated by the permutation function.
Figure 3.4: Branching method with $\tau_{i1}$ and $\tau_{i2}$ being the departure time for the first and second flights and the corresponding case assignment $j_1$ and $j_2$. The $r_i$ is evaluated using (3.15) and (3.16) for each completed branch. The highest $r_i$ is then saved as the best assignment for asset $i$.

Algorithm 6 Exhaustive Branch-and-Cut (E-B&C)

1: PermSeq = Perm(1,...,A) $\triangleright$ Permutation of ordering of assets for which to start allocation
2: for each AssetSeq in PermSeq do
3:     for each $i \in$ AssetSeq do
4:         assign($i$) = B&C($i$)
5:         updatePoA(assign($i$)) $\triangleright$ Prevent overlap of patrol box assignments
6:     end for
7:     PotentialAssign $\leftarrow$ PotentialAssign + assign $\triangleright$ Save potential assignment given we allocated in order AssetSequence
8:     resetPoA $\triangleright$ Set PoA to originally initialized surface prior to any updates
9: end for
10: BestAssign = MaxReward(PotentialAssign) $\triangleright$ For all potential assignments found, search and find that which resulted in the maximum reward

3.3.2 Greedy Branch-and-Cut I

Similar to E-B&C, we repeat the asset allocation process for all the available assets and fix the assignment for an asset $i^*$ with the highest $r_i$. After the asset-case-time epoch assignment is fixed, we update the PoA surface to ensure that the assigned cases are no longer available for additional scheduling during the assigned search hours. The same process is then repeated until either no more assets are available or all cases
are fully allocated. We refer to this method as GB&C-I. The pseudocode is shown in Algorithm 7. In this pseudocode, line 1 states that while there are any unassigned assets, continue on to lines 2–7, where the best assignment for each unassigned asset is found using B&C. The best asset assignment is then selected in line 8 (i.e., $i^*$ becomes known among the explored potential assignments). In lines 9–11, the PoA surface is updated given the asset assignment found.

Algorithm 7 Greedy Branch-and-Cut I (GB&C-I)

1: while length(AssignedAsset) ≤ $A$ do
2: assign = ∅
3: for each $i$ ∈ {$1, \ldots, A$} do
4: if $i$ /∈ AssignedAsset then
5: assign($i$) = B&C($i$)
6: end if
7: end for
8: assignment($i^*$) = MaxReward(assign)
9: AssignedAsset ← AssignedAsset + $i^*$
10: BestAssign ← BestAssign + assignment($i^*$)
11: updatePoA(assignment($i^*$))
12: end while

3.3.3 Greedy Branch-and-Cut II

To reduce the runtime and problem complexity, we propose a second greedy Branch-and-Cut method, referred to as GB&C-II. This method is similar to the E-B&C method, except that we put an additional constraint on assets. Once we enumerate all the possible departure times and find the best assignment \{$j^*$\} corresponding to each departure time for an asset $i$, we fix the corresponding schedule. That is, we reduce the complexity of search with more than one asset from permutation ordering to a linear ordering. The same process is then repeated until all cases are fully allocated.
or there are no more assets available. The pseudocode is shown in Algorithm 8. Here, line 2 finds the best assignment for asset $i$ found in line 1. Line 3 updates the PoA surface and line 4 saves the best assignment found in line 2.

**Algorithm 8 Greedy Branch-and-Cut II (GB&C-II)**

1: for each $i \in \{1, \ldots, A\}$ do
2: assign($i$) = B&C($i$)
3: updatePoA(assignment($i$))
4: BestAssign ← BestAssign + assign($i$)
5: end for

### 3.3.4 Parallelized Greedy Branch-and-Cut II

To further improve the computation time, we develop a parallelized version of the GB&C-II algorithm. Parallelization involves dividing a large problem into multiple independent subproblems, where each subproblem is assigned to a processor. This substantially reduces the computation time and therefore can rapidly generate allocation solutions. We use a Master-Slave architecture for our parallelization with the following functionalities:

**Master Process**

- Pools subproblems for the slave processors to run
- Spawns the subproblems on multiple slave processors
- Collects the results from the slave processors

**Slave Process**

- Receives the subproblem from the master processor
• Executes the subproblem

• Returns the solution to the master processor

The serial GB&C-II algorithm, executed on a single processor, searches the branch-and-cut tree by expanding live nodes one at a time. In order to parallelize this problem on $M$ processors, we set each $\tau_{i1}$ to each processor and let each processor execute the subproblem. All processes share the same memory for the PoA and other read-only data. Lastly, the master processor collects all value returns from the slave processors to evaluate the best assignment for asset $i$.

3.3.5 Approximate Dynamic Programming (ADP)

Another approach to solve the problem is via approximate dynamic programming, more specifically, a one-step lookahead rollout algorithm. Note that the following formulation is for a single arbitrary asset $i$ and is thus assumed given throughout. Let $j_k$ be the asset-case assignment at time epoch $k$ and $z_k$ be the remaining aloft time for an asset at time epoch $k$. We have the state equation for $z_{k+1}$ as:

$$z_{k+1} = f(z_k, j_k)$$ (3.24)

where $j_k$ is the state-based control variable that selects a case $j$ at time epoch $k$ as

$$j_k = \mu_k(z_k, j_{k-1}), \quad j_k = 0, 1, \ldots, C$$ (3.25)

Here, $z_k = L_i$ and $j_k = 0$ implies that there is no asset-case assignment made and the asset is in the rest state at time epoch $k$. When $z_k \leq L_i$ and $j_k = 1, \ldots, C$, an
asset-case assignment has been made at time epoch \( k - 1 \) and the asset is currently in a flight state. The detailed control options are described in this section later (see (3.31) and (3.32)).

The approximate dynamic programming equation for the problem is defined as follows:

\[
g_k(z_k, j_k) = \lambda_{j_k} \left( 1 - \prod_{k \in s_{jk}} (1 - w_{ijk}) \right)
\]  
\[
J_k(j_k) = \max_{j_k} E \left\{ g_k(z_k, j_k) + \bar{J}_{k+1} (f(z_k, j_k, \Lambda(k))) \right\}
\]  

(3.26) (3.27)

where \( s_{jk} \) is the set of remaining search time indices available within the current sortie for asset \( i \) assigned to case \( j \) and \( \Lambda(k) \) is a function that indicates that the asset is currently flying its \( \ell^{th} \) flight at time \( k \). The variable \( \lambda_j \) is the normalized priority weight for case \( j \). Here, \( \bar{J}_{k+1} \) is the heuristic cost-to-go and is estimated based on the following assumptions:

\( \textbf{H1:} \) The asset will fly out for its maximum aloft time

\( \textbf{H2:} \) Each asset will stay on just one case for each flight

\( \textbf{H3:} \) Each asset will fly out immediately after it is fully rested.

\( \textbf{H4:} \) The case with the highest total reward will be selected for the \( \ell^{th} \) flight interval, as in (3.28)

\[
j^* = \arg \max_j \lambda_j \left( 1 - \prod_{k \in \gamma(i,j,\ell)} (1 - w_{ijk}) \right)
\]  

(3.28)

where \( \gamma(i, j, \ell) \) is the set of search time indices for asset \( i \) assigned to case \( j \) for the \( \ell^{th} \) flight. If the planning time horizon allows multiple flights, then we first compute
the best case for the next flight time defined by $H1$ to $H3$ using (3.28). The future cost-to-go for the $\ell^{th}$ flight is as follows.

$$
H(\ell) = \lambda^{*}_{j^{*}} \left( 1 - \prod_{k \in \gamma(i,j^{*},\ell)} (1 - w_{ij^{*}k}) \right)
$$

(3.29)

where $j^{*}$ is computed from (3.28). The heuristic cost-to-go given the current flight at time $k$ is $\Lambda(k)$ and is given by

$$
\bar{J}_{k+1}(f(z_k,j_k,\ell_k)) = \sum_{n=\Lambda(k)+1}^{\left\lceil \frac{K L_i + R_i}{\ell_i + R_i} \right\rceil} H(n)
$$

(3.30)

As mentioned before, the control variable $j_k$ is state-dependent. When an asset is at rest state at time $k$, the control variable $j_k$ comprises the actions of launching the asset or not with the intent of obtaining better reward at a later time epoch. That is,

$$
j_k = \begin{cases} 
0, & \text{do not launch the asset} \\
\beta, & \beta = 1, \ldots, C \text{ launch the asset}
\end{cases}
$$

(3.31)

A comparison of expected reward between launching the asset at the current hour versus the next hour is performed using rollout with the heuristic defined above. If launching the asset during the current time epoch results in a higher reward, then the asset will be assigned to the case with the highest total reward $r_i$ in (3.28) and assigned for the first search hour to the selected case. If launching the asset during the next time epoch results in a higher reward, then we simply increment the time epoch and repeat the process. Fig. 3.5 illustrates this rollout heuristic for determining the expected reward for launching at a different hour.
Figure 3.5: Illustration of rollout for deciding when to fly with the traveling time (green), search time (blue).

When the asset is in flight, for the second through final hour of the search, the control variable $j_k$ takes on a different set of values, detailed as follows:

$$j_k = \begin{cases} j_{k-1}, & \text{Stay on the current case} \\ \tilde{j} \neq j_{k-1} & \text{Switch to a different case with the cost of additional travel time} \end{cases}$$

(3.32)

We illustrate the computation of the heuristic for the 1-step lookahead rollout in Fig. 3.6. The first example illustrates the situation when the surveillance asset is searching for case $j$ and chooses to stay on case $j$ for the remaining search interval. The second example illustrates the situation, wherein the asset currently searching for case $j$ switches to a new case $\tilde{j} \neq j$, while considering the cost of additional travel time from case $j$ to case $\tilde{j}$. The travel time between the new case $\tilde{j}$ to the asset’s home base is then the new return travel time for the asset. The optimal control action is selected based on the maximum expected reward, as in (3.27). This process is repeated for each time epoch $k$ to obtain a feasible asset-case assignment over the planning horizon.
Figure 3.6: Illustration of one-step lookahead. 1) Stay at current case; 2) switch to a different case with the cost of additional traveling time; 3) return to the asset’s base.

3.3.6 Multi-Step Lookahead Approximate Dynamic Programming-I

We propose two multi-step lookahead ADP strategies to obtain near-optimal assignments for all assets. The first method begins with an $m$-length permutation of the asset order for which to start the allocation. That is, $m = 1$ corresponds to searching over each asset; $m = 2$ corresponds to searching over all possible pairs of assets and so on. The PoA is then updated with respect to each asset-case-time assignment to ensure that there are no duplicate asset-case-time tuples. The difference between the two methods lies in how the remaining assets are allocated. In the first proposed method, we exhaustively compute the feasible asset assignment for all the available assets and fix the allocation corresponding to the asset with the highest $r_i$. The PoA is then updated and the process is then repeated until either no more assets are available or all cases are fully allocated. Once all the assets are allocated, we reset the PoA surface to its original state and repeat the process from the beginning with the next possible $m$-length subset of assets to start the initial asset assignment over the time horizon. We refer to this method as $m$SLADP-I. The pseudocode is shown in Algorithm 9. In the Algorithm 9’s pseudocode, line 1 generates the $m$-length
permutation of asset order, where $m$ specifies the size of the subset permutation to be used for the initial asset allocation. Lines 4–7 find the best allocation given each asset $i$ in a specific asset order from line 1 and updates the PoA surface, accordingly. Then, lines 12–18 compute the best assignment for the remaining unassigned assets. Line 12 finds the best assignment for each unassigned asset and line 17 selects the best asset $i^*$ for allocation. The PoA surface is subsequently updated in line 18. Lines 20–24 save the complete assignment and reset the parameters for the next asset permutation sequence generated in line 1.

### 3.3.7 Multi-Step Lookahead Approximate Dynamic Programming-II

The second multi-step lookahead method (referred as $m$SLADP-II), as in the first method, begins with an $m$-length permutation of asset ordering. The difference between $m$SLADP-I and $m$SLADP-II is how the algorithm computes the asset allocation for the remaining assets. In $m$SLADP-II, we iteratively compute the best asset allocation for each $i$. Once the best assignment is found for asset $i$, we immediately fix the corresponding schedule and update the PoA surface. There is no additional loop to find the best asset-case assignment among all the remaining assets. Hence, $m$SLADP-II is faster and less complex than $m$SLADP-I. The same process is then repeated until all cases are fully allocated or there are no more assets available. The pseudocode is shown in Algorithm 10. In this pseudocode, lines 1–7 remain the same as Algorithm 9. The difference lies in lines 10–17, where, for each unassigned asset, we find the best assignment corresponding to an asset $i^*$ and update the PoA surface accordingly. Once all assets are assigned, we reset all the parameters for the next asset permutation
Algorithm 9 \textit{mSLADP-I}

1: PermSeq = \text{Perm}([1, \ldots, A], m) \triangleright m$-\text{length permutation of asset order, where } m \text{ specifies the size of the subset permutation to be used for initial asset allocation}$

2: \textbf{for each} AssetSeq in PermSeq \textbf{do}

3: \hspace{1em} \textbf{for each} \; i \in \text{AssetSeq} \textbf{do}

4: \hspace{2em} assign(i) = \text{ADP}(i)

5: \hspace{2em} BestAssign \leftarrow BestAssign + assign(i)

6: \hspace{2em} updatePoA(assign(i))

7: \hspace{2em} AssignedAsset \leftarrow AssignedAsset + i

8: \hspace{1em} \textbf{end for}

9: \textbf{while} \; \text{length}(AssignedAsset) \leq A \textbf{ do}

10: \hspace{1em} \textbf{for each} \; i \in [1, \ldots, A] \textbf{ do}

11: \hspace{2em} \textbf{if} \; i \notin \text{AssignedAsset} \textbf{ then}

12: \hspace{3em} assignTemp(i) = \text{ADP}(i)

13: \hspace{2em} \textbf{end if}

14: \hspace{1em} \textbf{end for}

15: \hspace{1em} b_{assign}(i^*) = \text{MaxReward}(assignTemp) \quad \triangleright \text{Given the previous allocations, select the asset assignment with the highest } r_i \text{ among the remaining available assets}$

16: \hspace{1em} AssignedAsset \leftarrow AssignedAsset + i^*

17: \hspace{1em} BestAssign \leftarrow BestAssign + b_{assign}(i^*)

18: \hspace{1em} updatePoA(assignment(i^*))

19: \hspace{1em} \textbf{end while}

20: PotentialAssign \leftarrow PotentialAssign + BestAssign

21: assign = \emptyset

22: AssignedAsset = \emptyset

23: BestAssign = \emptyset

24: resetPoA

25: \textbf{end for}

26: BestAssign = \text{MaxReward}(PotentialAssign)

sequence generated from line 1.
Algorithm 10 mSLADP-II

1: PermSeq = Perm({1, ..., A}, m)
2: for each AssetSeq in PermSeq do
   3:     for each $i \in$ AssetSeq do
   4:         assign($i$) = ADP($i$)
   5:         BestAssign $\leftarrow$ BestAssign + assign($i$)
   6:         updatePoA(assign($i$))
   7:         AssignedAsset $\leftarrow$ AssignedAsset + $i$
   8:     end for
   9: for each $i \in \{1, \ldots, A\}$ do
   10:     if $i \notin$ AssignedAsset then
   11:         assign($i$) = ADP($i$)
   12:     end if
   13: end for
   14: AssignedAsset $\leftarrow$ AssignedAsset + $i^*$
   15: BestAssign $\leftarrow$ BestAssign + assign($i$)
   16: updatePoA(assignment($i^*$))
   17: PotentialAssign $\leftarrow$ PotentialAssign + BestAssign
   18: assign = $\emptyset$
   19: AssignedAsset = $\emptyset$
   20: BestAssign = $\emptyset$
   21: resetPoA
   22: end for
23: BestAssign = MaxReward(PotentialAssign)

3.4 Simulation and Computational Results

The proposed algorithms were implemented in Python 2.7 on an Intel® Core™ i7-6600U CPU @ 2.60GHz × 4 with 32 GB RAM. Our computational results are organized as follows: We first describe the mission scenario. Then, we discuss the solution quality of various algorithms with respect to objectives $O1$–$O3$ and their runtimes. Additionally, we conduct scalability analyses of the algorithms by varying the number of assets and cases, as well as robustness of the various algorithms using a signal-to-noise ratio (SNR) metric from robust design [138].
3.4.1 Scenario Description(s)

There are two main areas of operation in the simulated scenario: the East Pacific Ocean and the Caribbean Sea. The PoA surfaces corresponding to this area of responsibility (AOR) were partitioned into a grid of $90 \times 138$ cells, where each cell is a square with a side length of 30 nautical miles. The total area of the AOR was $\approx 11$ million square nautical miles. The lower left corner of the rectangular AOR had a latitude and longitude of $10^\circ S, 110^\circ W$, respectively.

The PoA surfaces forecasted ten smuggler cases, of which five were located in the East Pacific Ocean and the remaining five were located in the Caribbean Sea. The details for each case can be found in Table 3.2 and Fig. 3.8. These cases are generated based on Navy intelligence, which typically comprises estimates of the expected number of smugglers on board and the size of the contraband shipment. Often there are few “active” cases, i.e., cases which targeteers deem to have sufficient actionable intelligence to allocate assets to. We assume the PoA surfaces reflect the spatio-temporal probabilities pertaining to such “active” cases. Four different types of smuggler vessels were considered: 1) Go Fast – small, fast boats capable of reaching high speeds, 2) Panga – modest-sized, fast boats, that are easy to build by the smugglers. 3) Self-Propelled Semi-Submersible (SPSS) – narco-submarines capable of shifting heavy loads long distances while almost submerged under the ocean’s surface [180], and 4) Fully submerged vessel – makeshift submarine-like vessels that can remain submerged with large quantities of cocaine aboard. Each case had a unique departure, destination, and waypoint combination. Waypoints are defined as possible areas in the ocean where the cargo is transferred to another vessel or a change in trajectory of the smuggler is predicted. Additionally, each case also had an associated
Table 3.2
SMUGGLER CASES

<table>
<thead>
<tr>
<th>Case</th>
<th>Case ID</th>
<th>Vessel Type</th>
<th>Speed (kts)</th>
<th>Payload (kg)</th>
<th># of Smugglers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GF1</td>
<td>GO FAST</td>
<td>30</td>
<td>1000</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>PG1</td>
<td>PANGA</td>
<td>20</td>
<td>450</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>GF2</td>
<td>GO FAST</td>
<td>30</td>
<td>1000</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>PG2</td>
<td>PANGA</td>
<td>20</td>
<td>450</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>PG3</td>
<td>PANGA</td>
<td>20</td>
<td>450</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>PG4</td>
<td>PANGA</td>
<td>20</td>
<td>450</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>SP1</td>
<td>SPSS(^a)</td>
<td>8</td>
<td>2500</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>FSV1</td>
<td>FSV(^b)</td>
<td>4</td>
<td>5000</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>PG5</td>
<td>PANGA</td>
<td>20</td>
<td>450</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>PG6</td>
<td>PANGA</td>
<td>20</td>
<td>450</td>
<td>2</td>
</tr>
</tbody>
</table>

\(^a\)Self-Propelled Semi-Submersible  
\(^b\)Fully Submerged Vessel

payload measured in kg of cocaine. This is relevant when we run the algorithm with objective \(O2\). An important fact to note is that each case had different start and end times. Fig. 3.7 details the time epochs when each smuggler case is deemed active. Cases with high uncertainty had wide bands of \(PoA\). The amount of uncertainty is dependent on the type of smuggler vessel (e.g., SPSSs can be extremely difficult to detect, and thus the corresponding \(PoA\) surfaces reflect this in long and broad bands of probability reflecting spatial and temporal uncertainty), and/or departure time uncertainty.

In the scenario, ten P-3 surveillance assets were considered as available for allocation during the planning horizon. The home bases of individual surveillance assets are detailed in Table 3.3. Each asset carries two different types of sensors with performance parameters detailed in Table 3.4.

We simulated the scenario with a granularity of one hour (i.e., the forecasted
Figure 3.7: Chart displaying when each smuggler case is active over the 72 hour time horizon. Cases are active up through time $K = 72$ and do not necessarily end at that time, but rather, due to the time horizon of the forecast data, are truncated.

Table 3.3

<table>
<thead>
<tr>
<th>Asset Home Base Location (Longitude, Latitude)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 6</td>
</tr>
<tr>
<td>2, 7</td>
</tr>
<tr>
<td>3, 8</td>
</tr>
<tr>
<td>4, 9</td>
</tr>
<tr>
<td>5, 10</td>
</tr>
</tbody>
</table>

surfaces were for each hour, on the hour, thus $\Delta = 1$ h). The forecasts extended to 72 h out from the current time (i.e., $K = 72$) and an asset allocation solution (e.g., $x_{ijk} = 0$ or 1) was required for each time epoch, $k$ in order for the algorithm to terminate.

Note that we omit E-B&C for large size scenarios in our results due to an exponential increase in computation times. Therefore, for E-B&C, we compute the solution for scenarios involving only up to 5 assets and 10 cases.
Table 3.4
Sensor-to-Target Sweepwidth (nm)

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>FSV&lt;sup&gt;a&lt;/sup&gt;</th>
<th>SPSS&lt;sup&gt;b&lt;/sup&gt;</th>
<th>FV&lt;sup&gt;c&lt;/sup&gt;</th>
<th>PANGA</th>
<th>GF&lt;sup&gt;d&lt;/sup&gt;</th>
<th>MV&lt;sup&gt;e&lt;/sup&gt;</th>
<th>SV&lt;sup&gt;f&lt;/sup&gt;</th>
<th>UNK&lt;sup&gt;h&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>APS 115</td>
<td>5</td>
<td>7.5</td>
<td>7.5</td>
<td>9</td>
<td>7.5</td>
<td>7.5</td>
<td>7.5</td>
<td>2.5</td>
</tr>
<tr>
<td>APS 137</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>18</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>5</td>
</tr>
</tbody>
</table>

<sup>a</sup> Fully submerged vessel  
<sup>b</sup> Self-propelled semi-submersible  
<sup>c</sup> Fishing vessel  
<sup>d</sup> Go-fast  
<sup>e</sup> Merchant vessel  
<sup>f</sup> Supply vessel  
<sup>g</sup> Historical data  
<sup>h</sup> Unknown (other)

Figure 3.8: Experiment scenario.

3.4.2 Solution Quality with Different Objective Function

Using the aforementioned values for the parameters, we ran the simulation for all the approaches to schedule the ten specified assets over the 72 h planning horizon. Tables 3.5-3.7 show the cumulative probability of successful detection ($CPoSD$) for the GB&C-II method for objectives $O1$, $O2$ and $O3$, respectively. Parallel GB&C-II has the same result as the sequential GB&C-II. Therefore, we omit the Parallel GB&C-II from the quality comparison.

We refer to Tables 3.5-3.7 as COA matrices. The COA matrices aid the DM in understanding the reasoning behind the algorithm’s behavior and its output by giving metrics both for individual asset-case pairs, as well as overall, the probability an asset detects at least one case ($PDC$) and the probability that a case is detected by
### Table 3.5
**Objective O1: Maximize Weight of Contraband Detected**

<table>
<thead>
<tr>
<th>Asset</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
<th>Case 5</th>
<th>Case 6</th>
<th>Case 7</th>
<th>Case 8</th>
<th>Case 9</th>
<th>Case 10</th>
<th>PDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.60</td>
</tr>
<tr>
<td>3</td>
<td>0.09</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.25</td>
<td>0.29</td>
<td>-</td>
<td>-</td>
<td>0.60</td>
</tr>
<tr>
<td>4</td>
<td>0.15</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.28</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.47</td>
</tr>
<tr>
<td>5</td>
<td>0.21</td>
<td>0.14</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.33</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.61</td>
<td>-</td>
<td>-</td>
<td>0.61</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.40</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.40</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>0.09</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.21</td>
<td>-</td>
<td>-</td>
<td>0.28</td>
</tr>
<tr>
<td>9</td>
<td>0.17</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.30</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.42</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>-</td>
<td>0.20</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.11</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.06</td>
</tr>
<tr>
<td><strong>PDA</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.55</td>
<td>0.80</td>
<td>0.95</td>
<td>-</td>
<td>0.19</td>
</tr>
</tbody>
</table>

*Expected weight of contraband disrupted: 7,828 kg*
*Expected number of detections: 3.41*
*Expected number of smugglers: 8.21*

### Table 3.6
**Objective O2: Maximize Number of Detections**

<table>
<thead>
<tr>
<th>Asset</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
<th>Case 5</th>
<th>Case 6</th>
<th>Case 7</th>
<th>Case 8</th>
<th>Case 9</th>
<th>Case 10</th>
<th>PDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.76</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>0.27</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.38</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.61</td>
</tr>
<tr>
<td>3</td>
<td>0.20</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.28</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.43</td>
</tr>
<tr>
<td>4</td>
<td>0.15</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.10</td>
<td>-</td>
<td>-</td>
<td>0.35</td>
</tr>
<tr>
<td>5</td>
<td>0.19</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.33</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>0.13</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.20</td>
<td>-</td>
<td>0.31</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>0.29</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.37</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.55</td>
</tr>
<tr>
<td>8</td>
<td>0.13</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.23</td>
<td>-</td>
<td>-</td>
<td>0.33</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>-</td>
<td>0.10</td>
<td>-</td>
<td>0.18</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.25</td>
</tr>
<tr>
<td>10</td>
<td>0.17</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.05</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td><strong>PDA</strong></td>
<td>0.61</td>
<td>0.48</td>
<td>0.10</td>
<td>0.28</td>
<td>0.18</td>
<td>0.38</td>
<td>0.28</td>
<td>0.86</td>
<td>0.23</td>
<td>0.46</td>
<td></td>
</tr>
</tbody>
</table>

*Expected weight of contraband detected: 6,619 kg*
*Expected number of detections: 3.85*
*Expected number of smugglers: 8.67*
### Table 3.7

**Objective O3: Maximize The Number of Smugglers Detected**

<table>
<thead>
<tr>
<th>Asset</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
<th>Case 5</th>
<th>Case 6</th>
<th>Case 7</th>
<th>Case 8</th>
<th>Case 9</th>
<th>Case 10</th>
<th>PDC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.21</td>
<td>-</td>
<td>0.61</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.69</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.34</td>
<td>0.26</td>
<td>-</td>
<td>-</td>
<td>0.15</td>
<td>0.59</td>
</tr>
<tr>
<td>3</td>
<td>0.09</td>
<td>0.23</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td>0.48</td>
</tr>
<tr>
<td>4</td>
<td>0.11</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.21</td>
<td>-</td>
<td>-</td>
<td>0.10</td>
<td>-</td>
<td>-</td>
<td>0.37</td>
</tr>
<tr>
<td>5</td>
<td>0.18</td>
<td>0.10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td>0.25</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.58</td>
<td>-</td>
<td>0.15</td>
<td>0.64</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>0.27</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.11</td>
<td>0.35</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.16</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.30</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.22</td>
<td>-</td>
<td>0.36</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>0.09</td>
<td>0.16</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.41</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>-</td>
<td>0.09</td>
<td>0.16</td>
<td>0.18</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td>0.23</td>
</tr>
<tr>
<td>PDA</td>
<td>0.44</td>
<td>0.44</td>
<td>0.17</td>
<td>0.34</td>
<td>0.18</td>
<td>0.63</td>
<td>0.51</td>
<td>0.84</td>
<td>0.22</td>
<td>0.35</td>
<td></td>
</tr>
</tbody>
</table>

*a Expected weight of contraband detected: 7,036 kg
*b Expected number of detections: 4.13
*c Expected number of smugglers: 9.37

at least one asset (PDA). These matrices may be generated to assess the allocation performance at a particular time epoch, or, as shown in Tables 3.5-3.7, the cumulative asset allocation performance up to that point in time (in Tables 3.5-3.7 through $K = 72$).

Solving with respect to Objective $O1$ (Table 3.5) resulted in an asset allocation with the highest expected weight of contraband detected, totaling 7,828 kg of cocaine compared to Objectives $O2$ and $O3$ (Tables 3.6 and 3.7). This implies that we have a 64% success rate of detecting the transport of contraband with respect to the total possible for the experimental scenario of 12,200 kg of contraband. The asset allocations with respect to Objective $O1$ have 15.5% and 10.1% more contraband disrupted when compared to Objectives $O2$ and $O3$, respectively. In Table 3.5, Case 8 has the most amount of contraband (5000 kg) with a $CPoSD = 0.95$. Solving with respect to Objective $O3$ resulted in the detection of a higher expected weight of contraband

87
expected number of detections (6.8%), and expected number of smugglers (7.5%) compared to Objective $O_2$. This could be caused by the uniform priority weight vector used in objective $O_2$.

For the sake of compactness, we omit the COA matrices used in demonstrating the performance of the other approaches implemented and instead, quantify the goodness of the allocation by comparing the algorithms with that of GB&C-II algorithm as measured by the expected weight of the contraband detected, expected number of detections and the expected number of smugglers detected.

The sums of the totals for each objective for each algorithm are shown in Table 3.8. Fig. 3.9 shows a normalized representation of the results detailed in Table 3.8 where the largest possible number of detections and contraband detected was utilized as a basis for normalization of both metrics, respectively, to compare the expected number of detections and contraband weight detected. Note that Fig. 3.9 only contains the results for 1SLADP-I and 1SLADP-II; the detailed solutions of $m$SLADP with $m > 1$ are shown later in Section 3.4.3.

We illustrate in Table 3.8 and Fig. 3.9 that all branch-and-cut-based algorithms optimizing Objective $O_2$ are out-performed by the same algorithms optimizing Objective $O_3$ in terms of both the expected number of detections and expected number of smugglers. When comparing GB&C-I and GB&C-II, optimizing with respect to Objective $O_2$ resulted in 4% less expected number of detections and 1.2% less expected number of smugglers than when optimizing with respect to Objective $O_3$.

In terms of the amount of contraband detected, using the GB&C-I algorithm resulted in an allocation that obtained the highest expected amount of contraband detected when solving for Objective $O_1$; however, its solutions for maximizing the expected number of detections or expected number of smugglers were inferior to the
Figure 3.9: A normalized view comparing the performance of all the algorithms, with respect to the expected weight of contraband disrupted (O1), the expected number of interdictions (O2), the expected number of smugglers (O3).

ADP-based algorithms. In general, we see that the branch-and-cut-based methods are able to obtain more contraband when solving with respect to Objective O1, while the ADP-based method are able to get better solutions for the expected number of detections and expected number of smugglers when solving with respect to Objectives O2 and O3 with the exception of 1SLADP-I for Objective O3.

3.4.3 Scalability: Available Asset Sensitivity

In this section, we use Objective O1 for the scalability studies with respect to the number of assets. To measure the scalability, we limited the number of assets available for allocation for the scenario from 1 to 10 aircraft. Figs. 3.10 and 3.11 show the expected weight of contraband disrupted and the runtimes, respectively. The detailed
Table 3.8
Algorithm Comparison

<table>
<thead>
<tr>
<th>Objective</th>
<th>GB&amp;C-I</th>
<th>GB&amp;C-II</th>
<th>1SLADP-I</th>
<th>1SLADP-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>7,869</td>
<td>7,828</td>
<td>7,520</td>
<td>7,821</td>
</tr>
<tr>
<td>O2</td>
<td>6,658</td>
<td>6,619</td>
<td>7,185</td>
<td>7,610</td>
</tr>
<tr>
<td>O3</td>
<td>7,188</td>
<td>7,036</td>
<td>7,594</td>
<td>7,591</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objective</th>
<th>GB&amp;C-I</th>
<th>GB&amp;C-II</th>
<th>1SLADP-I</th>
<th>1SLADP-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>3.47</td>
<td>3.41</td>
<td>3.57</td>
<td>3.68</td>
</tr>
<tr>
<td>O2</td>
<td>3.87</td>
<td>3.84</td>
<td>3.72</td>
<td>4.08</td>
</tr>
<tr>
<td>O3</td>
<td>4.12</td>
<td>4.13</td>
<td>3.80</td>
<td>4.04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objective</th>
<th>GB&amp;C-I</th>
<th>GB&amp;C-II</th>
<th>1SLADP-I</th>
<th>1SLADP-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>8.35</td>
<td>8.21</td>
<td>8.37</td>
<td>8.72</td>
</tr>
<tr>
<td>O2</td>
<td>8.56</td>
<td>8.67</td>
<td>8.40</td>
<td>9.21</td>
</tr>
<tr>
<td>O3</td>
<td>9.50</td>
<td>9.37</td>
<td>8.70</td>
<td>9.12</td>
</tr>
</tbody>
</table>

Values are in Tables 3.9 and 3.10. In Fig. 3.10 and Table 3.9 we see that approximate dynamic programming-based algorithms (1SLADP-I, 1SLADP-II, 2SLADP-I and 2SLADP-II) are able to obtain similar amounts of contraband disrupted, differing by only up to 339.7 kg (4.6%) of contraband.

Similarly, the branch-and-cut-based algorithms (E-B&C, GB&C-I and GB&C-II) are able to obtain similar amounts of contraband disrupted, differing by only up to 279.1 kg of contraband among the three. E-B&C, intuitively, outperformed the other branch-and-cut variations (and all other algorithms for that matter) among the scenarios simulated until runtime became an issue. GB&C-II is able to obtain a better
result compared to GB&C-I when there are 2, 6, or 7 assets available for allocation. This is due to the nature of the scenario or the characteristics of the smuggler cases. Since GB&C-I iterates through all available assets, there is a tendency that closer (with respect to assets’ home base) cases are allocated first, since there is less travel time and, hence, are more rewarding. In turn, this may limit the options available to assets considered for allocation in later iterations since cases, previously in close proximity to their home base may already be allocated and, due to longer travel time, will be much less rewarding or not at all. Similar problems arose with 1SLADP-I algorithm, which obtains less expected contraband disrupted compared to 1SLADP-II algorithm when there are more than 6 assets available for allocation, differing by up to 314.9 kg of contraband. We are able to minimize the effect of this problem by applying a 2-step lookahead strategy. 2SLADP-I algorithm obtains less expected contraband disrupted compared to 2SLADP-II algorithm when there are more than 5 assets available for allocation, differing by up to 197.6 kg of contraband.

As Fig. 3.11 and Table 3.10 show, E-B&C has the slowest runtime. There is a maximum speed up of 34.8, 120.6, 210.6, 4,794, 6,146, 2,861, 3,711 and an average speed up of 9.8, 30.9, 53.7, 1,177, 1,542, 809, and 994 when comparing the runtimes of GB&C-I to GB&C-II, Parallel GB&C-II, 1SLADP-I, 1SLADP-II, 2SLADP-I, and 2SLADP-II, respectively. Over all the asset availability scenarios tested, the average speed up of GB&C-II, Parallel GB&C-II, 1SLADP-I, 1SLADP-II, 2SLADP-I, and 2SLADP-II are 3.6, 6.1, 87, 143, 52 and 72 times, respectively, faster compared to GB&C-I.

Our key finding here is that, with a 1.6% sacrifice in optimality on average, GB&C-II provides a solution nearly identical to that of E-B&C, while offering a solution in a fraction of the time (up to nearly 210.6 times faster among the simulated results).
Alternatively, at a cost of 2.5% suboptimality on average, but more than 6,146 times faster speedup, we can run the 1SLADP-II for a given scenario. Similarly, at a cost of 2.4% suboptimality on average, 2SLADP-II offers more than 3,711 times faster speedup.

In general, GB&C-II should be used when the total numbers of assets is less than 3 due to its minimal sacrifice in optimality (on average 1.6%). When the number of assets is greater than 3, 2SLADP-II should be used.

### 3.4.4 Scalability: Varying the Number of Cases

Here, we vary the number of cases from 1 to 10, while fixing the number of available assets to 10. Figs. 3.12 and 3.13 show the expected weight of contraband disrupted and the runtimes, respectively. The detailed values for each figure are in Tables 3.11 and 3.12.
Figure 3.11: The CPU runtimes for each algorithm by varying the number of available assets.

Table 3.9

<table>
<thead>
<tr>
<th># of Assets</th>
<th>E-B&amp;C</th>
<th>GB&amp;C-I</th>
<th>GB&amp;C-II</th>
<th>1SLADP-I</th>
<th>1SLADP-II</th>
<th>2SLADP-I</th>
<th>2SLADP-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3,806</td>
<td>3,806</td>
<td>3,806</td>
<td>3,806</td>
<td>3,806</td>
<td>3,806</td>
<td>3,806</td>
</tr>
<tr>
<td>2</td>
<td>5,240</td>
<td>4,747</td>
<td>5,240</td>
<td>4,747</td>
<td>4,747</td>
<td>4,747</td>
<td>4,747</td>
</tr>
<tr>
<td>3</td>
<td>5,997</td>
<td>5,997</td>
<td>5,935</td>
<td>5,952</td>
<td>5,988</td>
<td>5,988</td>
<td>5,988</td>
</tr>
<tr>
<td>4</td>
<td>6,436</td>
<td>6,349</td>
<td>6,253</td>
<td>6,306</td>
<td>6,341</td>
<td>6,341</td>
<td>6,341</td>
</tr>
<tr>
<td>5</td>
<td>6,762</td>
<td>6,617</td>
<td>6,483</td>
<td>6,583</td>
<td>6,658</td>
<td>6,658</td>
<td>6,693</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>6,816</td>
<td>6,847</td>
<td>6,748</td>
<td>6,823</td>
<td>6,823</td>
<td>6,858</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>7,239</td>
<td>7,265</td>
<td>6,983</td>
<td>7,298</td>
<td>7,292</td>
<td>7,323</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>7,501</td>
<td>7,490</td>
<td>7,195</td>
<td>7,494</td>
<td>7,396</td>
<td>7,519</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>7,734</td>
<td>7,691</td>
<td>7,369</td>
<td>7,648</td>
<td>7,535</td>
<td>7,673</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>7,809</td>
<td>7,828</td>
<td>7,520</td>
<td>7,821</td>
<td>7,648</td>
<td>7,846</td>
</tr>
</tbody>
</table>

respectively. From Fig. 3.12 and Table 3.11, we see that all the algorithms have very similar solution quality. We see a noticeable increase in contraband disruption for case 8 (5,000 kg of contraband). All algorithms obtained a similar amount of expected contraband disrupted.
Table 3.10
SIMULATION RUNTIME (s) FOR VARYING ASSET AVAILABILITY

<table>
<thead>
<tr>
<th># of Assets</th>
<th>E-B&amp;C</th>
<th>GB&amp;C-I</th>
<th>GB&amp;C-II</th>
<th>Parallel GB&amp;C-II</th>
<th>1SLADP-I</th>
<th>1SLADP-II</th>
<th>2SLADP-I</th>
<th>2SLADP-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.23</td>
<td>3.70</td>
<td>3.48</td>
<td>1.86</td>
<td>0.08</td>
<td>0.08</td>
<td>0.09</td>
<td>0.08</td>
</tr>
<tr>
<td>2</td>
<td>15.1</td>
<td>10.9</td>
<td>7.54</td>
<td>4.49</td>
<td>0.18</td>
<td>0.18</td>
<td>0.16</td>
<td>0.17</td>
</tr>
<tr>
<td>3</td>
<td>69.6</td>
<td>22.8</td>
<td>11.8</td>
<td>6.75</td>
<td>0.38</td>
<td>0.26</td>
<td>0.30</td>
<td>0.29</td>
</tr>
<tr>
<td>4</td>
<td>418</td>
<td>45.7</td>
<td>16.7</td>
<td>9.82</td>
<td>0.53</td>
<td>0.36</td>
<td>0.51</td>
<td>0.47</td>
</tr>
<tr>
<td>5</td>
<td>2639</td>
<td>75.9</td>
<td>21.9</td>
<td>12.5</td>
<td>0.55</td>
<td>0.43</td>
<td>0.92</td>
<td>0.71</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>89.4</td>
<td>23.0</td>
<td>13.9</td>
<td>0.94</td>
<td>0.53</td>
<td>1.58</td>
<td>1.03</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>111</td>
<td>25.5</td>
<td>15.1</td>
<td>1.46</td>
<td>0.66</td>
<td>2.91</td>
<td>1.62</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>139</td>
<td>30.3</td>
<td>16.4</td>
<td>1.35</td>
<td>0.83</td>
<td>4.91</td>
<td>2.22</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>181</td>
<td>31.0</td>
<td>18.32</td>
<td>1.73</td>
<td>0.93</td>
<td>7.75</td>
<td>2.90</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>220</td>
<td>34.7</td>
<td>20.9</td>
<td>2.22</td>
<td>0.99</td>
<td>13.0</td>
<td>5.50</td>
</tr>
</tbody>
</table>

Fig. 3.13 and Table 3.12 show the runtimes. As expected, GB&C-I has the slowest runtimes while the 1SLADP-II algorithm has the fastest runtime (< 1 s). There are maximum speed ups of 7, 11, 99, 221, 17 and 40 when comparing the runtimes of GB&C-I to GB&C-II, Parallel GB&C-II, 1SLADP-I, 1SLADP-II, 2SLADP-I, and 2SLADP-II, respectively. On average the speed ups of the GB&C-II, Parallel GB&C-II, 1SLADP-I, 1SLADP-II, 2SLADP-I, and 2SLADP-II algorithms were 4.3, 6, 33, 71.8, 6, and 16.5 times, respectively.

The key point here is that the algorithm 2SLADP-I is very efficient and is recommended for scenarios when the number of cases is less than or equal to the number of assets, which is often the case.
Figure 3.12: The expected weight of contraband disrupted for each algorithm by varying the number of available cases.

Table 3.11
EXPECTED WEIGHT OF CONTRABAND DISRUPTED (KG) FOR VARYING CASE AVAILABILITY

<table>
<thead>
<tr>
<th># of Cases</th>
<th>GB&amp;C-I</th>
<th>GB&amp;C-II</th>
<th>1SLADP-I</th>
<th>1SLADP-II</th>
<th>2SLADP-I</th>
<th>2SLADP-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>653.2</td>
<td>558.7</td>
<td>653.2</td>
<td>629.6</td>
<td>657.7</td>
<td>653.2</td>
</tr>
<tr>
<td>2</td>
<td>1,000</td>
<td>775.8</td>
<td>968.7</td>
<td>999.4</td>
<td>1,037</td>
<td>999.4</td>
</tr>
<tr>
<td>3</td>
<td>1,332</td>
<td>1,189</td>
<td>1,302</td>
<td>1,334</td>
<td>1,355</td>
<td>1,359</td>
</tr>
<tr>
<td>4</td>
<td>1,459</td>
<td>1,425</td>
<td>1,462</td>
<td>1,479</td>
<td>1,471</td>
<td>1,504</td>
</tr>
<tr>
<td>5</td>
<td>1,540</td>
<td>1,452</td>
<td>1,474</td>
<td>1,549</td>
<td>1,537</td>
<td>1,610</td>
</tr>
<tr>
<td>6</td>
<td>1,680</td>
<td>1,633</td>
<td>1,707</td>
<td>1,655</td>
<td>1,727</td>
<td>1,685</td>
</tr>
<tr>
<td>7</td>
<td>3,145</td>
<td>3,090</td>
<td>3,049</td>
<td>3,133</td>
<td>3,125</td>
<td>3,188</td>
</tr>
<tr>
<td>8</td>
<td>7,725</td>
<td>7,694</td>
<td>7,311</td>
<td>7,648</td>
<td>7,665</td>
<td>7,716</td>
</tr>
<tr>
<td>9</td>
<td>7,725</td>
<td>7,694</td>
<td>7,320</td>
<td>7,633</td>
<td>7,612</td>
<td>7,754</td>
</tr>
<tr>
<td>10</td>
<td>7,869</td>
<td>7,828</td>
<td>7,520</td>
<td>7,821</td>
<td>7,648</td>
<td>7,846</td>
</tr>
</tbody>
</table>
Figure 3.13: The CPU runtimes for each algorithm by varying the number of available cases.

Table 3.12
Simulation Runtime (s) for Varying Case Availability

<table>
<thead>
<tr>
<th># of Cases</th>
<th>GB&amp;C-I</th>
<th>GB&amp;C-II</th>
<th>Parallel GB&amp;C-II</th>
<th>1SLADP-I</th>
<th>1SLADP-II</th>
<th>2SLADP-I</th>
<th>2SLADP-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.85</td>
<td>0.34</td>
<td>1.08</td>
<td>0.68</td>
<td>0.23</td>
<td>1.99</td>
<td>0.86</td>
</tr>
<tr>
<td>2</td>
<td>4.09</td>
<td>1.36</td>
<td>4.47</td>
<td>0.66</td>
<td>0.32</td>
<td>3.74</td>
<td>1.27</td>
</tr>
<tr>
<td>3</td>
<td>5.88</td>
<td>2.67</td>
<td>1.89</td>
<td>0.85</td>
<td>0.42</td>
<td>4.13</td>
<td>1.58</td>
</tr>
<tr>
<td>4</td>
<td>11.8</td>
<td>3.93</td>
<td>3.06</td>
<td>1.00</td>
<td>0.54</td>
<td>5.24</td>
<td>1.91</td>
</tr>
<tr>
<td>5</td>
<td>18.0</td>
<td>4.77</td>
<td>4.01</td>
<td>1.48</td>
<td>0.57</td>
<td>5.51</td>
<td>2.16</td>
</tr>
<tr>
<td>6</td>
<td>24.7</td>
<td>7.48</td>
<td>5.31</td>
<td>1.34</td>
<td>0.64</td>
<td>6.95</td>
<td>2.38</td>
</tr>
<tr>
<td>7</td>
<td>56.0</td>
<td>11.7</td>
<td>7.27</td>
<td>1.45</td>
<td>0.72</td>
<td>7.79</td>
<td>2.55</td>
</tr>
<tr>
<td>8</td>
<td>108</td>
<td>13.2</td>
<td>8.56</td>
<td>1.67</td>
<td>0.80</td>
<td>9.45</td>
<td>3.13</td>
</tr>
<tr>
<td>9</td>
<td>183</td>
<td>15.9</td>
<td>9.85</td>
<td>1.92</td>
<td>0.89</td>
<td>11.9</td>
<td>4.00</td>
</tr>
<tr>
<td>10</td>
<td>220</td>
<td>34.7</td>
<td>20.9</td>
<td>2.22</td>
<td>0.99</td>
<td>13.0</td>
<td>5.50</td>
</tr>
</tbody>
</table>

3.4.5 Robustness: Monte Carlo Evaluation of Asset Allocation Strategies

To test the robustness of each asset allocation algorithm, we simulated 100,000 trajectories of smugglers (10,000 from each case) behaving as in our benchmark
scenario. Sampling from the PoA surfaces, we obtained waypoints for each smuggler at each time epoch and joined them together to obtain a full path. From these paths, we measured whether the smuggler traversed through any allocated patrol boxes during the allocated search time, and if so, what was the aircraft’s probability of detecting the target during those time epoch(s). Table 3.13 shows the detailed performance statistics for each algorithm over the 100,000 Monte Carlo simulations. A useful metric to measure an algorithm’s goodness is that of nominal-the-best signal-to-noise ratio (SNR), that is,

$$SNR = 10 \log_{10} \frac{\mu^2}{\sigma^2}$$

Nominal-the-best SNR is a useful measure when the goal is to maximize a mean and minimize the variation. Note that maximization of this metric seeks to minimize the coefficient of variation (=standard deviation/mean) and is thus a measure of robustness of a solution. From the results of 100,000 Monte Carlo runs, we found that algorithm 2SLADP-II performs the best with respect to objective $O1$, when measured using the signal-to-noise ratio. The 2SLADP-II algorithm obtained, on average, 7,612 kg of contraband (out of a total of 12,200 kg purportedly transported).

As Table 3.13 shows, all algorithms were able to obtain a similar expected amount of contraband, with 2SLADP-II proving to be the most robust, as measured in terms of nominal-the-best SNR.
<table>
<thead>
<tr>
<th>Objective</th>
<th>Mean of Contraband Detected ($\mu$) in kg</th>
<th>Standard Deviation of Contraband Detected ($\sigma$) in kg</th>
<th>SNR (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GB&amp;C-I</td>
<td>7,616</td>
<td>246.3</td>
<td>29.8</td>
</tr>
<tr>
<td>GB&amp;C-II</td>
<td>7,632</td>
<td>252.5</td>
<td>29.6</td>
</tr>
<tr>
<td>1SLADP-I</td>
<td>7,645</td>
<td>244.2</td>
<td>29.9</td>
</tr>
<tr>
<td>1SLADP-II</td>
<td>7,610</td>
<td>218.4</td>
<td>30.8</td>
</tr>
<tr>
<td>2SLADP-I</td>
<td>7,648</td>
<td>240.3</td>
<td>30.1</td>
</tr>
<tr>
<td>2SLADP-II</td>
<td>7,612</td>
<td>203.3</td>
<td>31.5</td>
</tr>
</tbody>
</table>
4.1 Introduction

4.1.1 Motivation

Navy planners strive to optimize ship routes with respect to multiple objectives, e.g., fuel efficiency, time, distance, safety, etc. When the task is one of trying to optimize multiple objectives, humans are notoriously poor at decision making, especially if the task is dynamic and has inherent uncertainty [2, 25, 99, 162]. Consequently, decision support tools are needed to collaboratively optimize routes by evaluating and recommending multiple courses of action (COAs) from which a navy planner can select one. To support such mixed-initiative planning, the tool(s) must aid the
human planner to create COAs and evaluate his or her own plan against optimized ones, or to combine both human expectation of the forecast, geographic hazards, and possible uncertainty with the automated algorithm output for hybrid human-machine consensus on what routes to consider for one or more shipping vessels (as in the case of aircraft strike group path routing). Although the problem is formulated for ship routing, the path planning algorithm is applicable to unmanned aerial vehicle and helicopter routing, among other Navy missions.

The scope of this chapter is limited to that of many-objective ship routing problems in uncertain environments, where many-objective refers to 15 or more, to be simultaneously optimized and traded-off. Such problems are rarely undertaken in practice due to the computational complexity. Motivated by carrier strike group missions and fuel cost optimization, which also falls under the category of many-objective ship routing, we treat different weather parameters as individual objectives to be optimized (e.g., minimize relative wind speed, current speed, etc.). Each such weather-based objectives involves varying spatio-temporal uncertainty over a finite horizon with some correlation. Thus, our problem can be succinctly stated as follows: Given a graph (e.g., as in [158]), a departure point, and a destination point, find a representative set of Pareto optimal shortest paths in a reasonable amount of time to optimize as many as 15 or more objectives.

The ship routing problem falls under the rubric of a multi-objective shortest path problem under uncertainty with time windows, speed and bearing as additional control variables, that is, with time-varying stochastic and non-convex costs at nodes and along arcs in the network, the evolution of which is similar to the timescale of the ship’s transit. What makes the problem intractable is that arc costs are time-dependent, non-convex, and many-dimensioned. The complexity of the problem space
and concurrent constraints renders the majority of provably optimal multi-objective approaches unusable, thus necessitating the need for tools that allow for navigational planning and replanning that consider both economic and practical needs of naval and commercial shipping [158].

4.1.2 Related Research

The single-objective shortest path problem is widely studied in the literature and was researched extensively, for example [15,43,49,70,100,122,156], during the mid-1950s to late-1960s. By optimizing the path with two or more objectives that are usually in conflict, we obtain the multi-objective path planning problem, which, is a key component of the ship routing problem considered here. Solution approaches to the multi-objective path planning in the literature primarily fall into one of two categories: 1) Generation methods, 2) Conversion to a single-objective shortest path problem.

Generation method refers to the direct generation of the Pareto front by solving the shortest path problem. In this vein, Hansen [69] first examined the case of two objectives and the concomitant computational complexity of the problem. Based on Hansen’s work, Henig [75] proposed a dynamic programming approach, where performance improvements were obtained when the arc costs are quasiconcave/quasiconvex. Kostreva and Wiecek [94] proposed a generalized dynamic programming approach (both backward and forward) to obtain multi-objective shortest paths on networks with (known) time-dependent arc costs.

Aside from these approaches, a majority of the research around multi-objective shortest path problems are of the label setting (e.g., [109]) or label correcting variety. Martins’ algorithm [109] is a label setting algorithm in the spirit of Dijkstra’s shortest
path algorithm, but in lieu of a single cost, a label with multiple entries, corresponding to each of many objective costs, is set on each vertex. Stewart and White extended the A* algorithm to a multi-objective variant (MOA*) \[165\], where they devised an intelligent method to select nodes to expand as part of their algorithm. Most recently, Mandow proposed a new approach to multi-objective A* (NAMOA*), where the algorithm smartly expands selected paths using various heuristic evaluation functions \[106,107\]. We use this method as a reference method for comparison with the NAPO algorithm of this chapter.

Generation of the full Pareto-frontier suffers from rapidly increasing computation time and storage due to the NP-hard nature of the multi-objective shortest path problem \[63,154\]. Therefore, approximation methods for rapidly generating the Pareto-frontier are desired to make them practical in real-world applications. Warburton \[178\] approximated the Pareto-frontier and bounded the problem complexity to polynomial time by introducing the $\epsilon$-nondominated scaling procedure. Based on Warburton’s work, Hassin presented two alternative approximation algorithms with fully polynomial complexity \[71\]. Even these modifications can result in intractable computational complexity.

The second approach that bypasses direct generation is the second aforementioned category of multi-objective path planning, which involves converting the multi-objective shortest path problem into a single objective shortest path problem either through a utility function \[120,151\] or objective weights based on user preference \[38,62\]. This method may be the fastest in term of computation runtime; however, these methods often significantly reduce the Pareto-optimality of the resulting Pareto front, that is, the number of distinct routing options available to the navigator can be substantially smaller than what can be obtained.
In this chapter, we present a fast approximate method for the Pareto-front generation utilizing a combination of approximate dynamic programming (ADP) techniques (i.e., one-step lookahead, rollout) and clustering techniques (i.e., Gaussian mixture model (GMM), silhouette score). This combination substantially reduces the computation time, while enlarging the number of distinct courses of action that the navigator can choose from.

4.1.3 Chapter Organization

The chapter is organized as follows. In Section 4.2, we provide the environmental setup and modeling. Then, in Section 4.3, we introduce the problem formulation. The detail of our solution approach is discussed in Section 4.4. Then, we apply our approach to a ship routing example with complex weather conditions and compare the results with those obtained via NAMOA* in Section 4.5.

4.2 Environmental Setup and Modeling

4.2.1 Environmental Impact

In this chapter, accurate short and medium range weather predictions are used in conjunction with ship models (e.g., how a ship’s speed may be impacted by the expected wind, wave, and current conditions). Commercial ship voyage planning modules are used to calculate the impacts of weather, ship’s hull form, cargo, and (power) plant characteristics on fuel costs. Broad categories of impacting weather include, but are not limited to, winds, waves, and currents. Among wind features, wind
speed and direction are used in impact calculation. Among wave features, amplitude, period, and direction are considered. Additionally, current direction and speed are used in ship impact calculations. Environmental parameters are forecast by multiple models [13][170][181].

Besides weather, bathymetry data is crucial in calculating the optimized ship routes. Bathymetry, extracted from the Oceanographic Atmospheric Master Library (NRL-MRY, NRL-Stennis Space Center), is divided into four categories: 1) shallow water, 2) water deeper than a certain threshold (in this chapter, twelve feet), 3) land, and 4) unknown. From this data, TMPLAR extracts possible safe paths that do not cross over land, and in alignment with the navigator’s desire to allow travel over shallow water or not, and routes ships in a mixed-initiative decision making cycle.

### 4.2.2 Fuel Cost Calculation

A high priority concern for navigators is to save on fuel to reduce expenditures, while also increasing operational endurance and asset availability [41]. Fuel cost calculation is complex and involves the ship’s hydrodynamics, nonlinear combinations of model parameters, and one or more exogenous variables. TMPLAR utilizes the Smart Voyage Planning Decision Aid (SVPDA), explained in detail in [117], for fuel consumption calculation. Input parameters include swell heights and periods, surface wind speeds and directions, wave directions, heights and periods, and current speeds and directions. In high fidelity fuel consumption models, relative wind and sea-state calculations have a direct impact on ship speed depending on the ship’s bearing, e.g., the wind and current may aid a ship along its course if its bearing is the same; however, if the ship is against the wind/current direction, it will have to expend much more fuel to get to
its destination in time. Fuel consumption is thus highly sensitive to the ship’s speed (both when traveling at slow and fast speeds) \[37\].

The overall power needed to maintain a speed from one node to the next is calculated as in (4.1):

\[
P_{Total} = P_{CW} + P_{Sea} + P_{Swell} + P_{Wind},
\]  

(4.1)

where \(P_{Total}\) represents the total power required, and \(P_{CW}\) represents the power required to navigate at the specified ship speed in calm water and current. Here, \(P_{Sea}\), \(P_{Swell}\), and \(P_{Wind}\) represent the additional resistances due to the sea, swell, and wind components, respectively. \(P_{CW}\) is dependent on the relative direction and velocity of the current with respect to the ship; \(P_{Sea}\), \(P_{Swell}\), and \(P_{Wind}\) are similarly so, using direction, height, and period of the sea and swell(s), respectively, for the former two components, and speed and direction for the latter. SVPDA even considers the ambient air temperature to incorporate the effect of temperature on HVAC (heating, ventilation, and air conditioning) loads. Details on ship’s power calculation can be found in \[37,61,83,103\].

### 4.2.3 Problem Setup

In conjunction with navigator’s input, the problem space is set up as follows. The coordinates of a departure and destination and a Great Circle route is constructed between the two (assuming no land obstacles prevent doing so). Using the Great Circle route as a basis, a specified number of “stages” are inserted equi-spaced along

---

1 A Great Circle route is one that is the shortest distance between two points lying on the surface of a sphere, often used in ship navigation and air travel. It is also known as the geodesic distance.
the track. Additionally, within each stage, a specifiable number of nodes are inserted at a predefined distance cross track (perpendicular) on each side of the Great Circle route. In this manner, a navigator-definable multi-stage grid is constructed. The start and end stages consist of one node each and the specifiable number of nodes (including the node associated with the Great Circle route) is added to each stage. The resulting grid system is a trellis with a finite number of stages, wherein each node in one stage is connected to all the nodes in the next. This provides a grid system for finding the Pareto optimal paths between the source and the destination.

Ship safety is the highest priority among all objectives the navigator considers when routing ships. When searching for viable paths (arcs) to traverse between stages, bathymetry and weather conditions are checked both at the node locations and along the connecting arcs. Doing so reduces the problem space further, eliminating options that should not be explored due to infeasibility or safety concerns. Due to the severity of the consequences if a ship is not routed safely, hard thresholds/constraints are enforced. If any threshold is exceeded, the location is removed from consideration, e.g., if the wave height exceeds a threshold specified by the planner, the corresponding node and/or arc is removed from the problem space. The safety constraint is thus modeled as a Heaviside function, where there are two types of nodes/arcs: those that are passable by the ship and those that are impassable.
4.3 Problem Formulation

4.3.1 Deterministic Problem

Adapting the formulation from [158], let \( G = (N, E) \). We write the (forward, which is feasible due to a deterministic forecast) dynamic programming equation as follows.

4.3.1.1 States

Let \( x_s \) be the two-dimensional state which consists of the node \( n_s \) at stage \( s \) and the arrival time \( \tau_s \). Note that the arrival time at a node is the same as the departure time at a node. That is, generally for the current stage \( s \), the state can be written as follows.

\[
x_s = (x_{1,s}, x_{2,s}) = (n_s, \tau_s)
\]  (4.2)

4.3.1.2 Controls

Let \( S \) be the required number of nodes in a path connecting the origin and the destination. The control variables at stage \( s \) are: 1) which node \( n_{s+1} \) to traverse to; 2) the power plant configuration \( \rho_s \) needed to efficiently traverse to the next stage \( s + 1 \) departing at \( \tau_s \). Control variables determine the speed and bearing of the ship at node \( n_s \). That is,

\[
u_s = (u_{1,s}, u_{2,s}) = (n_{s+1}, \rho_s)
\]  (4.3)
4.3.1.3 Transition Dynamics

Then, the transition dynamics for stage $s+1$ are given by,

$$x_{s+1} = f_s(x_s, u_s) \quad (4.4)$$

where

$$x_{1,s+1} = u_{1,s} \quad (4.5)$$

$$x_{2,s+1} = x_{2,s} + b(x_{1,s}, u_{1,s}, u_{2,s}) \quad (4.6)$$

where $b$ is the transit time from node $n_s$ at stage $s$ such that we arrive at node $n_{s+1}$ at time $\tau_{s+1}$ in stage $s+1$. Note that $\tau_{s+1} \in (0, T]$ is an integer multiple of a time resolution $\Delta > 0$ and $T \geq \Delta$ is a given integer denoting the maximum amount of time specified to transit the route. We assume transit time to be nonnegative.

4.3.1.4 Objective Function

We denote the $d$-dimensional cost to traverse arc $\langle x_{1,s}, x_{1,s+1} \rangle$ as $c(x_s, u_s)$, where $d$ is the total number of objectives to consider, and $c^i(x_s, u_s)$ as the cost pertaining to a particular objective $i \in \{1, 2, \ldots, d\}$. The costs $c^i(x_s, u_s)$, $i \in \{1, \ldots, d\}$ are assumed to be nonnegative. The power plant configuration $u_{2,s}$ is used for both fuel efficiency and to achieve certain top speeds, given weather impacts (i.e., it is a proxy for time of arrival $\tau_{s+1}$).

We now define the shortest path (with respect to objective $i$) to the from the start node $n_1$ to a node $n_{s+1}$ in stage $s+1 \leq S, s \neq 1$ as $J_{s+1}^i(x_{s+1})$. The cost is found by
solving (4.7).

\[
J^*_s(x_{s+1}) = \min_{x_s} \left\{ J^*_s(x_s) + c^i(x_s, u_s) \right\}
\]  
(4.7)

s.t.  
\[
0 < x_{2,s} + b(x_{1,s}, u_{1,s}, u_{2,s}) \leq T
\]
(4.8)

\[
u_{2,s} \in P
\]
(4.9)

\[
x_{1,s}, x_{1,s+1} \in N_a
\]
(4.10)

\[
\langle x_{1,s}, x_{1,s+1} \rangle \in E_a
\]
(4.11)

where \( P \) is the set of allowable power plant configurations, and \( N_a \) and \( E_a \) are the set of safe nodes and arcs, respectively (i.e., for a given a ship class, those nodes/arcs whose bathymetry is of a certain depth or greater). The recursion is initiated with the initial condition \( J_1(x_1) = 0 \). Our constraints include arriving by a time \( \tau_{s+1} \) that satisfies (4.8), choosing an allowable plant configuration (4.9), and only traversing along feasible safe nodes and edges in the network (4.10)–(4.11).

Although constraint (4.8) results in a large problem space, we show later that, by exploiting time discretization and earliest possible arrival/latest allowable departure times via forward-backward Dijkstra algorithm, we can significantly reduce the required computation.

### 4.3.2 Multi-objective Extension

To solve the multiple objective problem, we must use a labeling algorithm and find the set of Pareto optimal labels (solutions) for stage \( s \). Let label \( g_{s,\ell s}(x_s) \), \( \ell = 1, \ldots, \mathcal{L} \), of \( \mathcal{L} \) Pareto optimal labels in stage \( s \), be as in (4.12).

\[
g_{s,\ell s}(x_s) = [J^1_s(x_s), \ldots, J^i_s(x_s), \ldots, J^d_s(x_s)]
\]
(4.12)
where each label $g_{s,\ell^*}$ has cardinality equal to the number of objectives $d$.

Adapting the multi-objective notation from [106], a set of one or more labels will be stored at each node. In general, a cost vector $g_\ell$ is said to dominate $g_{\ell'}$, denoted as $g_\ell \prec g_{\ell'}$, if and only if

$$g_\ell \prec g_{\ell'} \iff \forall_i \ J^i \leq J'^i \land g_\ell \neq g_{\ell'}$$

(4.13)

where $J^i$ denotes the $i^{th}$ element of the label (or cost vector) $g_\ell$. Then, given a set of vectors, denoted by $F$, we shall define $\text{nondom}(F)$ as the set of nondominated vectors in set $F$, that is,

$$\text{nondom}(F) = \{ g_{\ell^*} \in F | \not\exists g_{\ell'} \in F, g_{\ell'} \prec g_{\ell^*} \}$$

(4.14)

(i.e., there does not exist a cost vector $g_{\ell'}$ that dominates $g_{\ell^*}$ in label set $F$). Then, by augmenting the control vector to include $\ell$ as the third control variable, and letting $F_{s+1}(x_{s+1})$ be the nondominated label set over state $x_{s+1}$, we can rewrite (4.7) as

$$F_{s+1}(x_{s+1}) = \text{nondom}_{u_s}\left\{ g_{s,\ell^*}(x_s) + c(x_s, u_s) \right\}$$

(4.15)

subject to (4.8)-(4.11).

### 4.4 Fast Approximate Method for the Pareto-frontier Generation

We propose a 1-Step Lookahead (1SL) combined with rollout strategy to solve the dynamic programming problem in (4.15). To further reduce the problem complexity,
we utilize a Gaussian mixture model (GMM), along with the silhouette score, to cluster the potential nondominated paths. A subset of the paths are then selected to be the representative paths for the next stage.

4.4.1 Time Windows

Without loss of generality, we limit the time window to take on only integer multiples of the time resolution $\Delta > 0$. Let $\mathcal{T}(n_s, n_{s+1})$ be the minimum time to traverse from $n_s$ to $n_{s+1}$. To incorporate deadlines and feasibility, the problem is then to find optimal times of arrival and departure at each node subject to (4.8)–(4.11), while also satisfying (4.16), which corresponds to enforcing an earliest possible time of arrival at each node and a latest allowable departure from each node to reach the destination $n_S$ by time $T$.

$$\mathcal{T}(n_1, n_s) \leq \tau_s \leq T - \mathcal{T}(n_s, n_S),$$ (4.16)

The upper and lower bounds in (4.16) can be computed via Dijkstra’s algorithm.

Thus, the time window at $n_S$ is $[\lceil \mathcal{T}(n_1, n_S)/\Delta \rceil \ast \Delta, \lfloor T/\Delta \rfloor \ast \Delta]$, where $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ denote the ceiling and floor functions, respectively. Instituting time windows at each node in the network, $\tau_s$ is constrained as,

$$[\lceil \mathcal{T}(n_1, n_s)/\Delta \rceil \ast \Delta \leq \tau_s \leq \lfloor T - \mathcal{T}(n_s, n_S)/\Delta \rfloor],$$ (4.17)

That is, we may only consider discretized times that fall within the time window, at time intervals of arbitrary length $\Delta$, for each node when deciding the time to depart the previous node and the time to arrive at the next, given the latest allowable time of arrival, $T$, at $n_S$. Thus, the time window on the destination propagates through
the network.

4.4.2 1-Step Lookahead with Rollout Strategy

To solve (4.15), let us write the approximate dynamic programming equation as,

$$\phi(x_s, u_s) = g_{s, \ell_s}(x_s) + \ell(x_s, u_s)$$ (4.18)

$$F_{s+1}(x_{s+1}) = \text{nondom} \left\{ \phi(x_s, u_s) + \bar{H}_{s+1}(f_s(x_s, u_s)) \right\}$$ (4.19)

subject to (4.8)–(4.11). \(\bar{H}_{s+1}(x_s, u_s)\) is the heuristic cost-to-go and is estimated as follows. Let the A* shortest path algorithm [70] solve the single objective shortest route. For each objective \(i\), let us denote the optimal cost to be \(A_i = \{a_{i_s}^{i}\}_{s=1}^{S}\), which consists of the objective function’s cumulative cost at each stage and the optimal path \(R_i = \{r_{i_s}\}_{s=1}^{S}\). Note that this step only needs to be performed once; the single objective optimal route/cost information can be stored and reused. Let us assume that, at node \(n_{s+1}\) in state \(x_{s+1}\) (which can be computed given current state \(x_s\) and \(u_s\) using (4.4)), we travel to the node \(r_{i_{s+2}}^{i}\) (arriving at optimal arrival time \(\tau_{s+2}^{*}\)). We denote this assumed (and known \textit{a priori}) control as \(\bar{u}_{s+1}\) (that is, we connect the path in its current state back to its corresponding optimal single objective path), then, the heuristic cost-to-go at \(x_{s+1}\) for objective \(i\) is defined as

$$\bar{H}_{s+1}(x_{s+1}) = \left[ h^1(x_{s+1}, \bar{u}_{s+1}), \ldots, h^d(x_{s+1}, \bar{u}_{s+1}) \right]$$ (4.20)

$$h^i(x_{s+1}, \bar{u}_{s+1}) = a_{s}^{i} - a_{s+2}^{i} + c^i(x_{s+1}, \bar{u}_{s+1})$$ (4.21)

In other words, the heuristic cost-to-go from stage \(s + 1\) consists of the cost to travel to the next stage’s optimal node and assumes optimal traversal for the remainder of the path to the terminal node.
4.4.3 Gaussian Mixture Model and Silhouette Score to Reduce Problem Space

To further reduce the problem complexity, we cluster the non-dominated paths and select a subset of paths for further expansion. Let $L_s$ be the Pareto optimal set of labels that collects all non-dominated labels at stage $s$ by iterating over set $F_s(x_s)$ for all nodes $n \in N_s$, where $N_s$ denotes the set of nodes in stage $s$, and for all feasible departure times $\tau_s$ at stage $s$,

$$L_s = \text{nondom} \{F_s(x_s)\} \forall n_s \in N_s,$$

that is, the set of non-dominated states available at stage $s$. We then cluster the non-dominated paths associated with labels in $L_s$ into 2 to a maximum number $K$ ($K$ is a hyper parameter) clusters via Gaussian mixture models (GMMs). The number of clusters is determined by the silhouette score, which measures the quality of GMM clusters. Rouseeuw [150] proposes that each cluster have a corresponding silhouette value that indicates which data points lie well within a cluster and those that are between clusters. The average silhouette score for each solution represents the clustering solution’s validity and can serve as the basis for selecting the number of clusters that best fits the data points when utilizing a GMM clustering algorithm.

For each non-dominated path in $L_s$ with the corresponding label $g_{s \ell s}(x_s)$, let $z(\ell)$ be the average dissimilarity of path $\ell$ with all other path within some cluster $k$. This can be any of several distance metrics, but for simplicity, let $z(\ell)$ be the average Euclidean distance from path $\ell$ to any other path within the same cluster. Let $y(\ell)$ denote the lowest average dissimilarity amongst all clusters. As with $z(\ell)$, the distance from path $\ell$ to all other paths not within the same cluster are calculated and the
lowest distance is then selected. Essentially, \( z(\ell) \) signifies how well a point is assigned to a cluster and \( y(\ell) \) signifies which cluster path \( \ell \) best fits in. The silhouette score, \( w(\ell) \), for path \( \ell \), is then defined as

\[
w(\ell) = -\frac{y(\ell) - z(\ell)}{\max \{ z(\ell), y(\ell) \}}
\]

(4.23)

It is evident from (4.23) that \(-1 \leq w(\ell) \leq 1\). If the silhouette score \( w(\ell) \) is 0, this implies that path \( \ell \) is on the border between two clusters. If \( w(\ell) \) is close to 1, path \( \ell \) is badly matched to clusters other than the one it has already been assigned to and is considered to be appropriately labeled; however, if \( w(\ell) \) is close to -1, it implies that path \( \ell \) should have been labeled as belonging to some cluster other than the one it is currently assigned to. Utilizing the average silhouette value, given in (4.24), it is possible to interpret the “goodness” of the GMM cluster scheme by aggregating all the silhouette values for each label and taking the average.

\[
\frac{1}{\mathcal{D}} \sum_{\ell=1}^{\mathcal{L}} w(\ell)
\]

(4.24)

Other metrics that may suit a given problem space include geometric or harmonic averages.

For each stage, we cluster the non-dominated paths with the total number of clusters ranging from 2 to \( K \) and use the number of clusters with the highest average silhouette value. Then, we pick the top \( m \) number of paths from each cluster for the next stage’s calculations, where the top here refers to the highest silhouette scores.

Additionally, we also consider \( \lambda \) stages of freedom, that is, we do not perform the clustering technique when the algorithm is \( \lambda \) stages from the destination node.
Alternately worded, only stages 1 to $S - \lambda$ result in the invocation of GMM clustering with silhouette score calculations. This is applied to increase the number of Pareto-optimal labels set at the destination.

4.5 Simulation and Computational Results

4.5.1 Scenario Description

The test scenario used a time resolution of $\Delta = 15$ minutes with a specified departure time of midnight on August 1, 2017 at Roanoke Island, North Carolina, USA ($36.0^\circ$ N, $75.0^\circ$ W), and a deadline of midnight on August 3, 2017 at Rock Sound, The Bahamas ($25.0^\circ$ N, $76.0^\circ$ W). A trellis was constructed between the start and destination, such that there were 6 equi-spaced stages between the origin and destination, and with a Great Circle route connecting through the middle. Five nodes were inserted to the west and east of the Great Circle track, per stage, at a distance of 20 nautical miles. In Fig. 4.1, the grid used for our analyses is illustrated. The test scenario used forecast information available up to the departure time and coincided with the impacts and aftereffects of Tropical Storm Emily, a rapidly-forming storm system that passed through the Florida panhandle into the western Atlantic Ocean. The comparison and analyses in this section were performed on an Intel Corporation Xeon E3-1200 v5/E3-1500 v5/6th Gen Core Processor with 32 GB RAM.

The graph was generated and post-processed to remove nodes and arcs deemed unsafe (in the form of passable or impassable nodes/arcs). We tested the NAPO algorithm on up to 15 objectives and Table 4.1 shows the single-objective minimum cost route computed using A* with the corresponding time for each objective. Table
Figure 4.1: Graph network with start and destination and the connecting edges, starting at South Carolina and directed towards The Bahamas.

Figure 4.1 also shows that some objectives require more runtime compared to others. This is likely due to the nature of the data, how it is stored, the range of values associated with the objective (e.g., there are physical limits in meteorology on how high sea height can be, but distance is an unbounded metric since it is always possible to add more waypoints to a graph).

We set $\lambda = 2$, $K = 10$, and $m = 3$, that is, we do not perform the clustering technique for the last two stages of the graph, the silhouette score is evaluated when attempting to cluster the label set into 2 to 10 clusters, and we select 3 nodes from the $k$ clusters with the best silhouette score. Note that smaller $K$ significantly reduces
Table 4.1  
**Single Objective Cost and Simulation Runtime**

<table>
<thead>
<tr>
<th>Obj. Index</th>
<th>Obj. Names</th>
<th>Min. Route Total Cost</th>
<th>Simulation Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fuel Consumption</td>
<td>68.7 (kgal)</td>
<td>244.5</td>
</tr>
<tr>
<td>2</td>
<td>Sig. Wave Height</td>
<td>7.11 (m)</td>
<td>254.8</td>
</tr>
<tr>
<td>3</td>
<td>Sea Height</td>
<td>0.50 (m)</td>
<td>43.86</td>
</tr>
<tr>
<td>4</td>
<td>Sea Period</td>
<td>3.57 (s)</td>
<td>37.76</td>
</tr>
<tr>
<td>5</td>
<td>Swell Height</td>
<td>3.40 (m)</td>
<td>230.5</td>
</tr>
<tr>
<td>6</td>
<td>Swell Period</td>
<td>46.9 (s)</td>
<td>280.3</td>
</tr>
<tr>
<td>7</td>
<td>Swell Height 2</td>
<td>1.66 (m)</td>
<td>211.2</td>
</tr>
<tr>
<td>8</td>
<td>Swell Period 2</td>
<td>40.7 (s)</td>
<td>286.9</td>
</tr>
<tr>
<td>9</td>
<td>Swell Height 3</td>
<td>0.12 (m)</td>
<td>47.50</td>
</tr>
<tr>
<td>10</td>
<td>Swell Period 3</td>
<td>9.24 (s)</td>
<td>127.1</td>
</tr>
<tr>
<td>11</td>
<td>Wind Speed</td>
<td>40.41 (m/s)</td>
<td>177.4</td>
</tr>
<tr>
<td>12</td>
<td>Current Speed</td>
<td>1.46 (m/s)</td>
<td>133.9</td>
</tr>
<tr>
<td>13</td>
<td>Air Temperature</td>
<td>2101 (K)</td>
<td>285.1</td>
</tr>
<tr>
<td>14</td>
<td>Total Time</td>
<td>27.25 (hr)</td>
<td>79.43</td>
</tr>
<tr>
<td>15</td>
<td>Total Distance</td>
<td>662.8 (nm)</td>
<td>316.2</td>
</tr>
</tbody>
</table>

the simulation runtime at the cost of solution quality.

### 4.5.2 Solution Quality: NAMOA* vs NAPO

We ran the test scenario with both NAMOA* and the new approximate dynamic programming-based Pareto optimization (NAPO) method proposed in the chapter. Figure 4.2 shows the Pareto-optimal costs obtained by both algorithms. To compare the two Pareto-fronts, let us construct a vector consisting of all the single objective minimum costs as reference costs, that is, \( \mu = [68.7, 7.11]' \) for two objectives. Then, we measure the percentage difference between each Pareto-point solution cost and the
reference cost. That is, for $\ell = 1, \ldots, \mathcal{L}$,

$$
\epsilon^\ell = \frac{(g_{S,\ell}^* (x_S) - \mu) / \mu}{(4.25)}
$$

The average $\epsilon$ for NAMOA* are 0.16 and 0.16, for objectives 1 and 2, respectively. For NAPO, the average $\epsilon$ are 0.27 and 0.15 for objectives 1 and 2, respectively. The 88 Pareto-points found via NAMOA* were reduced down to 36 Pareto-points with a relatively small sacrifice in optimality via the proposed NAPO algorithm. We perform the same test for 3 objectives, where the average $\epsilon$ for NAMOA* is 0.23, 0.15 and 2.49, for objectives 1, 2 and 3, respectively. For NAPO, the average $\epsilon$ is 0.22, 0.19 and 5.04 for objectives 1, 2 and 3, respectively. NAMOA* output 608 Pareto-points for the 3-objective planning problem, while the proposed NAPO algorithm terminated with 42 Pareto-points, with minimal sacrifice in optimality.
### Table 4.2

**Total Number of Pareto-points and Simulation Runtime Comparison for 2 to 3 Objectives**

<table>
<thead>
<tr>
<th># of Obj</th>
<th>Total # of Pareto-points</th>
<th>Simulation Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NAMOA*</td>
<td>NAPO</td>
</tr>
<tr>
<td>2</td>
<td>88</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>608</td>
<td>42</td>
</tr>
</tbody>
</table>

#### 4.5.3 Scalability Analysis

To gauge the scalability of the NAPO algorithm, we varied the number of objectives from 2 to 15. A comparison of the number of Pareto-solutions and the simulation runtimes between the two algorithms is shown in Table 4.2. Note that the we were unable to obtain solutions from NAMOA* for more than 3 objectives due to the prohibitive and exponential simulation runtime of the algorithm. We list the remaining results for the NAPO algorithm in Table 4.3 when the number of objectives was varied from 3 to 15. Overall, the proposed NAPO algorithm’s runtime was strongly correlated with the number of GMM clusters. From Table 4.3, we were able to approximate a Pareto front for the complex 15-objective ship routing problem around the time of Tropical Storm Emily in 15 minutes, as shown in Table 4.3, where the Pareto-front comprised 866 solutions. A visual of the runtimes between the two algorithms is illustrated in Fig. 4.3, where no times were obtained for NAMOA* after 3 objectives due to its intractability in the context of the ship routing problem. The runtime of the NAPO algorithm was approximately sub-log linear, while NAMOA*’s runtime was definitively exponential.
Figure 4.3: Simulation runtime for total number of objective varies from 2 to 15.

Table 4.3
NAPO Algorithm Total Number of Pareto-points and Simulation Runtime for 4 to 15 Objectives

<table>
<thead>
<tr>
<th># of Objectives</th>
<th>Total # of Pareto-points</th>
<th>Simulation Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>54</td>
<td>341.5</td>
</tr>
<tr>
<td>5</td>
<td>51</td>
<td>411.8</td>
</tr>
<tr>
<td>6</td>
<td>282</td>
<td>503.1</td>
</tr>
<tr>
<td>7</td>
<td>221</td>
<td>563.0</td>
</tr>
<tr>
<td>8</td>
<td>120</td>
<td>618.0</td>
</tr>
<tr>
<td>9</td>
<td>664</td>
<td>666.1</td>
</tr>
<tr>
<td>10</td>
<td>500</td>
<td>686.1</td>
</tr>
<tr>
<td>11</td>
<td>173</td>
<td>710.3</td>
</tr>
<tr>
<td>12</td>
<td>241</td>
<td>735.9</td>
</tr>
<tr>
<td>13</td>
<td>114</td>
<td>810.7</td>
</tr>
<tr>
<td>14</td>
<td>843</td>
<td>857.2</td>
</tr>
<tr>
<td>15</td>
<td>866</td>
<td>902.4</td>
</tr>
</tbody>
</table>
Chapter 5

On the Identification of Noise Covariances and Adaptive Kalman Filtering: A New Look at a 50 Year-old Problem

5.1 Introduction

The Kalman filter (KF) \[85\] is the optimal state estimator for linear dynamic systems driven by Gaussian white noise with measurements corrupted by Gaussian white noise\[1\]. In the classical design of a Kalman filter, the noise covariance matrices are assumed known and they, along with the system dynamics, determine the achievable filter’s accuracy. However, in many practical situations, including noisy feature data in machine learning, the statistics of the noise covariances are often unknown or only

---

\[1\]The KF is also the best linear estimation algorithm when the noises are non-Gaussian with known covariances \[11\].
partially known. Thus, noise identification is an essential part of adaptive filtering. Adaptive filtering has numerous applications in engineering [121], machine learning [36], econometrics [44], weather forecasting [42,79,115,149], to name a few.

We were motivated by the following learning problem: Given a vector time series and a library of models of system dynamics for the data (e.g., a Wiener process, a white noise acceleration model, also called nearly constant velocity model, or a white noise jerk model, also called nearly constant acceleration model), find a suitable process noise and measurement noise model and the best system dynamics for the time series. The problem we consider in this chapter is limited to finding a suitable process noise and measurement noise covariance for a given dynamic model.

5.1.1 Previous Work

The approaches for estimating the noise covariance matrices for a Kalman filter can be broadly classified into four general categories: Bayesian inference, maximum likelihood estimation, covariance-matching, and correlation methods. The first two categories pose the noise covariance estimation problem as a parameter estimation problem.

In the Bayesian inference approach [77], the covariance estimation problem is solved by obtaining the posterior probability density function (pdf) of the unknown parameters (in this case, the noise covariance matrix elements) from their prior pdf and the observed measurements using the Bayes’ formula recursively. In 2013, Matisko and Havlena [110] proposed a new Bayesian method to estimate the unknown covariance matrices. They first use a Monte Carlo method to generate a grid of possible unknown covariance matrix pairs \((Q, R)\) with more density near the highest prior probability. Then, they compute the likelihood and posterior probability after performing state
estimation for each pair using a Kalman filter. In general, the Bayesian approach suffers from the curse of dimensionality and is computationally intractable due to the fact that it involves numerical integration or Monte Carlo simulations over a very large parameter space.

In maximum likelihood estimation [88, 164], the noise statistics are obtained by maximizing the probability density function of the measurement residuals generated by the filter, which is the likelihood of the filter parameters [11]. These filter-based maximum likelihood methods require nonlinear programming based optimization and are computationally intractable. Shumway and Stoffer [157] utilize the expectation maximization (EM) algorithm [48], which requires the smoothed estimates of the system state. This approach starts with the smoothed estimation of the system state given an estimate of the initial state and noise covariance matrices. Then, the unknown parameters are estimated via maximum likelihood estimation using the smoothed state estimates obtained from the expectation step. Later, Ghahramani and Hinton [60] present an extension of [157] that can account for an unknown observation matrix in linear dynamic systems. They then go on to use forward and backward recursions to estimate the noise covariance matrices. This process is repeated until the estimated parameters converge. In addition to computational complexity, this method suffers from convergence to a local optimum.

The basic idea of the covariance-matching techniques [125] is that the sample covariance of the innovations should be consistent with its theoretical value. In [125], the unknown noise covariances are estimated from the sample covariance computed from the innovation sequences accumulated over the entire historical data (or in a moving time window). In this method, if the estimated innovation covariance value is much larger than the theoretical value, then the process noise covariance is increased.
The convergence has never been proved for this method.

With regard to correlation methods, Heffes [74] derived an expression for the covariance of the state error and of the innovations of any suboptimal filter as a function of noise covariances. This expression serves as a fundamental building block in the correlation methods. The first innovation-based technique to estimate the optimal Kalman filter gain and the unknown noise covariance matrices via the correlations of innovations from an arbitrary initial stabilizing filter gain was introduced by Mehra [113]. Another procedure to carry out the identification of unknown optimal Kalman filter gain and the noise covariance matrices is by Carew and Bélanger [31]. Their strategy calculates the Kalman filter gain based on the estimation error that is defined as the discrepancy between the optimal state estimates obtained from the optimal Kalman filter gain and the state estimates obtained from an arbitrary suboptimal Kalman filter gain. There is a question as to whether the correlation method is sensitive to the initial Kalman filter gain selection. Mehra suggested to repeat the noise covariance estimation steps with the obtained gain from the first attempt to improve the estimation. However, Carew and Bélanger [31] claim that if the optimal Kalman filter gain is used as the initial condition, then the approximations in Mehra’s approach are such that the correctness of the optimal gain will not be confirmed.

Later, Neethling and Young [126] suggested to combine the noise covariance matrices in a vector and solve a single least squares or weighted least squares problem to improve the performance of Mehra and Carew–Bélanger’s approaches. In 2006, Odelson et al. [131,132] developed the autocovariance least squares method to estimate the noise covariance matrices by applying the suggestions of [126] on Mehra’s approach and using the Kronecker operator. The algorithm defines a multistep autocovariance
function between the measurements, which is used to develop a linear least squares formulation to estimate the noise covariance matrices. Duník et al. [52] compared the method presented by Odelson, Rajamani, and Rawlings [132] to a combined state and parameter estimation approach.

An interesting variant of the correlation methods is to utilize the output correlations. In 1972, Mehra [114] proposed an output correlation technique to directly estimate the optimal Kalman filter gain. This method has the advantage of being non-recursive compared to the innovation correlation techniques. However, the poor estimates of sample output correlation functions can lead to an ill-conditioned Riccati equation.

The contributions of the present chapter are as follows:

1. A necessary and sufficient condition for the identifiability of unknown noise covariances is provided for a Gauss-Markov system. This involves the rank of the auto and cross-covariances of the weighted sum of innovations of a suboptimal filter, where the weights are the coefficients of the minimal polynomial of the state transition matrix.

2. A novel six-step solution approach via a successive approximation and adaptive gradient descent scheme with a new objective function to obtain the unknown noise covariance matrices $Q$ and $R$, as well as the steady-state Kalman filter gain $W$, and the steady-state state prediction covariance matrix $\bar{P}$ or the updated state covariance matrix $P$, is proposed. This ensures positive definite $Q$ and positive definite $R$, as well as $\bar{P}$ and $P$.

3. Several novel approaches to estimate the unknown noise covariance matrix $R$ are derived via utilization of the post-fit residual, which has not yet been discussed in the literature.
4. Convergence proofs in [31] assumed that time averages are the same as ensemble averages. This is only approximate with finite data. Consequently, these methods either diverge or result in largely inaccurate estimates of unknown covariances.

5. Our approach can enforce structural assumptions on $Q$ and $R$ (e.g., diagonality of $Q$ and $R$, symmetry and positive definiteness).

The chapter is organized as follows. In Section 5.2, we provide an overview of the Kalman filter and derive a new Riccati equation for the updated state covariance. Then, in Section 5.3, we discuss the necessary and sufficient condition for the unknown noise covariances’ estimation. We briefly discuss different approaches to obtaining the unknown covariance parameters in Section 5.4. Then, in Section 5.5, we discuss a convergent version of Mehra’s method to estimate the optimal Kalman filter gain. In Section 5.6, we derive five different ways to obtain $R$. Section 5.7 provides a method to estimate the process noise covariance matrix $Q$ and the steady-state updated state covariance $P$, iteratively. All these methods are combined in Section 5.8, where we present a systematic process to find the optimal filter gain $W$, the innovation covariance $S$, the measurement noise covariance $R$, the steady-state state prediction (or updated state) covariance $\bar{P}$ ($P$) and the process noise covariance $Q$. In Section 5.9, we specialize the approach to estimate $W$, $R$, $Q$ and $P$ for a process, where all the states are observed and for a Wiener process. Lastly, we apply our approach to five numerical examples from the literature in Section 5.10. The iteration variable is superscript with (·) to differentiate the notation from exponents.
5.2 Plant and Measurement Model for the Kalman Filter

The notation used in the remainder of this chapter is listed in Table 5.1.

Consider the discrete-time linear dynamic system

\[ x(k + 1) = Fx(k) + Γv(k) \quad (5.1) \]
\[ z(k) = Hx(k) + w(k) \quad (5.2) \]

(i.e., a Gauss-Markov system) where \( x(k) \) is an \( n_x \)-dimensional state vector, \( F \) is the state transition matrix of the system, \( H \) is the \( n_z \times n_x \) measurement matrix, and \( Γ \) is the \( n_x \times n_v \) dimensional noise gain matrix. The sequences \( v(k), k = 0, 1, \ldots, \) and \( w(k), k = 0, 1, \ldots, \) are zero-mean white Gaussian noises with covariance matrices \( Q \) and \( R \), respectively. The two noise sequences and the initial state are assumed to be mutually independent. The matrices \( Q \) and \( R \) are assumed to be positive definite. Note that even if \( Q \) is positive definite, \( ΓQΓ' \) need not be; it can be positive semi-definite. We assume that the system is observable and \( (F, ΓQ^{1/2}) \) is controllable \(^2\).

Given the estimate \( \hat{x}(k|k) \), the Kalman filter \(^{[11]}\) estimates the state at the

\(^2\)Detectability and stabilizability are all that are needed for a stable Kalman filter (i.e., state observability is not needed).
### Table 5.1
**Summary of Notation**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>state transition matrix of the system</td>
</tr>
<tr>
<td>$H$</td>
<td>measurement matrix</td>
</tr>
<tr>
<td>$J$</td>
<td>objective function value</td>
</tr>
<tr>
<td>$M$</td>
<td>number of lags in sample covariance matrix computation</td>
</tr>
<tr>
<td>$n_x$</td>
<td>dimension of the vector $x$</td>
</tr>
<tr>
<td>$N$</td>
<td>number of observed samples</td>
</tr>
<tr>
<td>$P$</td>
<td>steady-state updated state covariance matrix</td>
</tr>
<tr>
<td>$\overline{P}$</td>
<td>steady-state state prediction covariance matrix</td>
</tr>
<tr>
<td>$v$</td>
<td>zero-mean white Gaussian noises with covariance matrices $Q$</td>
</tr>
<tr>
<td>$w$</td>
<td>zero-mean white Gaussian noises with covariance matrices $R$</td>
</tr>
<tr>
<td>$W$</td>
<td>Kalman filter gain</td>
</tr>
<tr>
<td>$x$</td>
<td>state vector</td>
</tr>
<tr>
<td>$\hat{x}$</td>
<td>estimate of $x$</td>
</tr>
<tr>
<td>$\tilde{x}$</td>
<td>error corresponding to the estimate of $x$</td>
</tr>
<tr>
<td>$z$</td>
<td>measurement vector</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>stepsize</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>noise gain matrix</td>
</tr>
<tr>
<td>$\bar{\epsilon}$</td>
<td>averaged NIS</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>regularization parameter</td>
</tr>
<tr>
<td>$\mu$</td>
<td>post-fit residual sequence with covariance matrix $G$</td>
</tr>
<tr>
<td>$\nu$</td>
<td>innovation sequence with covariance matrix $S$</td>
</tr>
<tr>
<td>$\mathcal{J}$</td>
<td>noise covariance identifiability matrix</td>
</tr>
<tr>
<td>Cov</td>
<td>covariance</td>
</tr>
<tr>
<td>$\delta_{kj}$</td>
<td>Kronecker delta function</td>
</tr>
<tr>
<td>diag($\cdot$)</td>
<td>diagonal of a matrix</td>
</tr>
<tr>
<td>$E[\cdot]$</td>
<td>expectation</td>
</tr>
<tr>
<td>tr($\cdot$)</td>
<td>trace (of a matrix)</td>
</tr>
<tr>
<td>vec($\cdot$)</td>
<td>linear transformation to convert a matrix into a column vector</td>
</tr>
<tr>
<td>$</td>
<td>\cdot</td>
</tr>
<tr>
<td>$</td>
<td></td>
</tr>
<tr>
<td>${\nu(k)}_{k=1}^{N}$</td>
<td>the sequence $\nu(k), k = 1, \ldots, N$</td>
</tr>
<tr>
<td>$\triangleq$</td>
<td>equal by definition</td>
</tr>
<tr>
<td>$\nabla_W$</td>
<td>gradient with respect to (the matrix) $W$</td>
</tr>
<tr>
<td>RMSE</td>
<td>root mean square error</td>
</tr>
<tr>
<td>NIS</td>
<td>normalized innovation squared</td>
</tr>
</tbody>
</table>
next time instant $k + 1$ as

\begin{align*}
\hat{x}(k + 1|k) &= F\hat{x}(k|k) \\
\nu(k + 1) &= z(k + 1) - H\hat{x}(k + 1|k) \\
\hat{x}(k + 1|k + 1) &= \hat{x}(k + 1|k) + W(k + 1)\nu(k + 1) \\
P(k + 1|k) &= FP(k|k)F' + \Gamma Q\Gamma' \\
S(k + 1) &= HP(k + 1|k)H' + R \\
W(k + 1) &= P(k + 1|k)H'S(k + 1)^{-1} \\
P(k + 1|k + 1) &= P(k + 1|k) - W(k + 1)S(k + 1)W(k + 1)'
\end{align*}

where the estimate $\hat{x}(k + 1|k)$ is the one-step extrapolated estimate of the state vector $x(k)$ based on the measurements up to $k$, $W(k), k = 1, \ldots, N$ is the sequence of Kalman filter gains, $\nu(k), k = 1, \ldots, N$ is the innovation sequence, $P(k + 1|k)$ is the state prediction covariance, $S(k + 1)$ is the measurement prediction (or innovation) covariance, and $P(k + 1|k + 1)$ is the updated state error covariance.

The six-step approach in this chapter is designed specifically for a steady-state Kalman filter. The steady-state state prediction covariance matrix $\bar{P}$ satisfies an algebraic Riccati equation.

\begin{equation}
\bar{P} = F[\bar{P} - \bar{P}H'(H\bar{P}H' + R)^{-1}H\bar{P}]F' + \Gamma Q\Gamma'
\end{equation}

The steady-state updated state covariance, denoted as $P$, can also be computed via another algebraic Riccati equation (see Appendix A.1).

\begin{equation}
P = FPF' - PH'(R - HPH')^{-1}HP' + \Gamma Q\Gamma'
\end{equation}
Evidently,

\[ P = \bar{P} - WSW' \]  
\[ = (I_{n_x} - WH)\bar{P}(I_{n_x} - WH)' + WRW' \]  

where (5.13) is known as the Joseph form; \( W \) and \( S \) are the steady-state optimal gain, and the steady-state innovation covariance, respectively, and are given by

\[ W = \bar{P}H'S^{-1} \]
\[ = \bar{P}H'(H\bar{P}H' + R)^{-1} \]
\[ = PH'R^{-1} \]  

\[ S = E[\nu(k)\nu(k)'] = H\bar{P}H' + R \]  

Note that \( (I_{n_x} - WH) \) is invertible, but need not be stable (i.e., eigenvalues need not be inside the unit circle).

### 5.3 Identifiability of \( Q \) and \( R \)

One major issue in the previous literature involves the necessary conditions to estimate the unknown covariance matrices. Mehra [113] claimed that the system must be observable and controllable; however, Odelson [132] provided a counter-example wherein the system was observable and controllable, but the full \( Q \) matrix was not estimable. Following the ideas in [168], we prove that the necessary and sufficient condition (as detailed in Appendix A.2) to estimate the unknown covariance matrices in a system is directly related to its minimal polynomial of

\[ \bar{F} = F(I_{n_x} - WH), \]  

130
its stable closed-loop filter matrix $\bar{F}$, and a transformation of the innovations based on the coefficients of the minimal polynomial. Let us define $\tilde{x}(k+1|k)$ to be the predicted error between the state $x(k+1)$ and its predicted state $\hat{x}(k+1|k)$, that is,

$$\tilde{x}(k+1|k) = x(k+1) - \hat{x}(k+1|k) \quad (5.17)$$

We can rewrite $\hat{x}(k+1|k)$ in terms of $\tilde{x}$, that is,

$$\hat{x}(k+1|k) = F\tilde{x}(k|k-1) + FW\tilde{x}(k|k-1) + FWw(k) \quad (5.18)$$

Then, substituting $(5.18)$ into $(5.17)$ and using $(5.1)$, we have

$$\tilde{x}(k+1|k) = \bar{F}\tilde{x}(k|k-1) + \Gamma v(k) - FWw(k) \quad (5.19)$$

where $\bar{F}$ is defined in $(5.16)$. We can also write $\nu(k)$ in terms of $\tilde{x}$, that is

$$\nu(k) = H\tilde{x}(k|k-1) + w(k) \quad (5.20)$$

Let us define the $m^{th}$ order minimal polynomial of $\bar{F}$ as

$$\sum_{i=0}^{m} a_i \bar{F}^{m-i} = 0; \quad a_0 = 1 \quad (5.21)$$

Then, the innovations $\nu(k)$ can be written as

$$\nu(k) = H\bar{F}^m\tilde{x}(k-m|k-m-1)$$

$$+ \left\{ H \sum_{j=0}^{m-1} \bar{F}^{m-1-j} [\Gamma v(k-m+j) - FWw(k-m+j)] \right\} + w(k) \quad (5.22)$$

Note that we apply the minimal polynomial of $\bar{F}$ to ensure that the innovation in
(5.22) is stationary. Let us define $\xi(k)$ as

$$\xi(k) = \sum_{i=0}^{m} a_i \nu(k - i)$$

(5.23)

$$= \sum_{i=0}^{m} a_i \left[ H \left\{ \sum_{j=0}^{m-i-1} \tilde{F}^{m-i-j} \left[ \Gamma \nu(k - m + j) - FWw(k - m + j) \right] \right\} + w(k - i) \right]$$

(5.24)

$$= \sum_{i=0}^{m} a_i \left[ H \left\{ \sum_{l=i+1}^{m-1} \tilde{F}^{l-i-1} \left[ \Gamma \nu(k - l) - FWw(k - l) \right] \right\} + w(k - i) \right]$$

(5.25)

$$= \sum_{l=1}^{m} \mathcal{B}_l \nu(k - l) + \sum_{l=0}^{m} \mathcal{G}_l w(k - l)$$

(5.26)

where $\mathcal{B}_l$ and $\mathcal{G}_l$ are the sum of two moving average processes driven by the process noise and the measurement noise, that is,

$$\mathcal{B}_l = H \left( \sum_{i=0}^{l-1} a_i \tilde{F}^{l-i-1} \right) \Gamma$$

(5.28)

$$\mathcal{G}_l = \left[ a_l I_{n_z} - H \left( \sum_{i=0}^{l-1} a_i \tilde{F}^{l-i-1} \right) FW \right]$$

(5.29)

$$\mathcal{G}_0 = I_{n_z}$$

(5.30)

Denoting $L_j = E [\xi(k) \xi(k-j)]$, for $j = 0, 1, 2, \ldots, m$, we have

$$L_j = \sum_{i=j+1}^{m} \mathcal{B}_i Q \mathcal{B}_j' + \sum_{i=j}^{m} \mathcal{G}_i R \mathcal{G}_j'$$

(5.31)

We know that $Q = [q_{ij}]$ is an $n_v \times n_v$ positive semi-definite and symmetric matrix, and $R = [r_{ij}]$ is an $n_z \times n_z$ positive definite and symmetric matrix. Utilizing the symmetry of $Q$ and $R$, and letting $b_{i,l}$ and $g_{i,l}$ denote the $l$-th column of $\mathcal{B}_i$ and $\mathcal{G}_i$, respectively,
we can rewrite (5.31) as

$$L_j = \sum_{l=1}^{n_v} \sum_{p=1}^{n_v} q_{lp} \left[ \sum_{i=j+1}^{m} b_{i,l}b'_{i-j,p} \right] + \sum_{l=1}^{n_z} \sum_{p=1}^{n_z} r_{lp} \left[ \sum_{i=j}^{m} g_{i,l}g'_{i-j,p} \right]$$

(5.32)

$$= \sum_{l=1}^{n_v} \left\{ \sum_{p=1}^{l} q_{lp} \left[ \sum_{i=j+1}^{m} b_{i,l}b'_{i-j,p} \right] + \sum_{p=l+1}^{m} q_{lp} \left[ \sum_{i=j+1}^{m} b_{i,l}b'_{i-j,p} \right] \right\}$$

$$+ \sum_{l=1}^{n_z} \left\{ \sum_{p=1}^{l} r_{lp} \left[ \sum_{i=j}^{m} g_{i,l}g'_{i-j,p} \right] + \sum_{p=l+1}^{m} r_{lp} \left[ \sum_{i=j}^{m} g_{i,l}g'_{i-j,p} \right] \right\}$$

(5.33)

$$= \sum_{l=1}^{n_v} \left\{ q_{lt} \left[ \sum_{i=j+1}^{m} b_{i,l}b'_{i-j,l} \right] + \sum_{p=l+1}^{m} q_{lp} \left[ \sum_{i=j+1}^{m} b_{i,l}b'_{i-j,p} + b_{i,p}b'_{i-j,l} \right] \right\}$$

$$+ \sum_{l=1}^{n_z} \left\{ r_{ll} \left[ \sum_{i=j}^{m} g_{i,l}g'_{i-j,l} \right] + \sum_{p=l+1}^{m} r_{lp} \left[ \sum_{i=j}^{m} g_{i,l}g'_{i-j,p} + g_{i,p}g'_{i-j,l} \right] \right\}$$

(5.34)

From (5.34), we can form the noise covariance identifiability matrix $\mathbf{J}$ of dimension $(m + 1)n_z^2 \times \frac{1}{2} [n_v(n_v + 1) + n_z(n_z + 1)]$, as in Algorithm 11. Algorithm 11 uses the vec(A) function to convert a matrix $A$ into a column vector. For a $p \times n$ matrix $A$,

$$\text{vec}(A) \triangleq [a_{11}, \ldots, a_{p1}, a_{12}, \ldots, a_{p2}, \ldots, a_{1n}, \ldots, a_{pn}]'$$

(5.35)

Using (5.34) and collecting terms corresponding to each $q_{lp}$, $p = l, l + 1, \ldots, n_v$ and each $r_{lp}$, $p = l, l + 1, \ldots, n_z$ into the corresponding columns of $\mathbf{J}$, we obtain the following identifiability condition that must be satisfied by $Q$ and $R$,

$$\mathbf{J} \begin{bmatrix} \text{vec}(Q) \\ \text{vec}(R) \end{bmatrix} = \begin{bmatrix} L_0 \\ L_1 \\ \vdots \\ L_m \end{bmatrix}$$

(5.36)
Algorithm 11 Construction of the noise covariance identifiability matrix $\mathcal{J}$

1: for $j := 0 : m$ do
2:     $r = j * n_z^2$
3:     $k \leftarrow 0$
4:     for $l := 1 : n_v$ do
5:         $k \leftarrow k + 1$
6:         $b = \sum_{i=j+1}^{m} [b_i b_{i-j}']$
7:         $\mathcal{J}(r+1 : r + n_z^2, k) \leftarrow \text{vec}(b)$
8:     end for
9:     for $p := l + 1 : n_v$ do
10:        $k \leftarrow k + 1$
11:        $c_{j,l,i}(p) = [b_i b_{i-j,p} + b_{i,p} b_{i-j,l}]'$
12:        $d = \sum_{i=j+1}^{m} c_{j,l,i}(p)$
13:        $\mathcal{J}(r+1 : r + n_z^2, k) \leftarrow \text{vec}(d)$
14:     end for
15: end for
16: for $l := 1 : n_z$ do
17:     $k \leftarrow k + 1$
18:     $g = \sum_{i=j}^{m} [g_i g_{i-j,p}]'$
19:     $\mathcal{J}(r+1 : r + n_z^2, k) \leftarrow \text{vec}(g)$
20:     for $p := l + 1 : n_z$ do
21:         $k \leftarrow k + 1$
22:         $h_{j,l,p}(i) = [g_i g_{i-j,p} + g_{i,p} g_{i-j,l}]'$
23:         $f = \sum_{i=j}^{m} h_{j,l,p}(i)$
24:         $\mathcal{J}(r+1 : r + n_z^2, k) \leftarrow \text{vec}(f)$
25:     end for
26: end for

The linearity of \((5.36)\) implies the full rank condition on $\mathcal{J}$. Since $R$ is always estimable because $G_m$ (recall that $m$ is the order of minimal polynomial) is invertible\(^3\), the maximum number of unknowns in $Q$ that can be estimated must be less than or equal to the minimum number of independent measurements minus the number of unknowns in $R$. That is

$$\text{rank}(\mathcal{J}) - n_R > n_Q$$ \hspace{1cm} (5.37)

\(^3\)See Appendix [A.3] for a detailed proof.
where \(n_R\) is the number of unknowns in \(R\), and \(n_Q\) is the number of unknowns in \(Q\).

To illustrate the necessity and sufficiency of this condition, consider an example system from [132],

\[
x(k) = \begin{bmatrix} 0.9 & 0 & 0 \\ 1 & 0.9 & 0 \\ 0 & 0 & 0.9 \end{bmatrix} x(k-1) + v(k-1) \\
z(k) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} x(k) + w(k)
\]

with \(Q\) being a full \(3 \times 3\) positive definite symmetric matrix and \(R\) being a full \(2 \times 2\) positive definite symmetric matrix. Since the rank of \(J\) is not affected by \(W\) (the observability condition is independent of the filter gain matrix), one can examine the rank of \(J\) for \(W = 0\) for convenience. In this case, the minimal polynomial coefficients are

\[
\begin{bmatrix} a_0 & a_1 & a_2 \end{bmatrix}' = \begin{bmatrix} 1 & -1.8 & 0.81 \end{bmatrix}'
\]

The \(\mathcal{B}\) and \(\mathcal{G}\) matrices are

\[
\mathcal{B}_1 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathcal{B}_2 = \begin{bmatrix} 1 & -0.9 & 0 \\ 0 & 0 & -0.9 \end{bmatrix}
\]

\[
\mathcal{G}_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathcal{G}_1 = \begin{bmatrix} -1.8 & 0 \\ 0 & -1.8 \end{bmatrix}
\]

\[
\mathcal{G}_2 = \begin{bmatrix} 0.81 & 0 \\ 0 & 0.81 \end{bmatrix}
\]
Here, \( \mathcal{F} \) is a \( 12 \times 9 \) matrix with a rank of 8. Since there are 9 unknown variables (6 in \( Q \) and 3 in \( R \)), the covariance matrix elements are not identifiable. However, if \( E[v(k)v(k)'] \) is diagonal, as is typically the case, then the covariance matrix elements are identifiable because there are only 6 unknown variables (full \( R \) matrix and three diagonal elements of \( Q \)).

Another example to illustrate the necessity and sufficiency of this condition is to consider the system

\[
x(k) = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.2 \end{bmatrix} x(k - 1) + \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} v(k) \tag{5.44}
\]

\[
z(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(k) + w(k) \tag{5.45}
\]

with \( Q \) being a diagonal \( 2 \times 2 \) positive definite diagonal matrix and \( R \) being a scalar. Similarly, we examine the rank of \( \mathcal{F} \) for \( W = 0 \) and obtain the minimal polynomial coefficients,

\[
\begin{bmatrix} a_0 & a_1 & a_2 \end{bmatrix}' = \begin{bmatrix} 1 & -0.3 & 0.02 \end{bmatrix}' \tag{5.46}
\]

The \( \mathcal{B} \) and \( \mathcal{G} \) matrices are

\[
\mathcal{B}_1 = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \mathcal{B}_2 = \begin{bmatrix} -0.2 & 0 \end{bmatrix} \tag{5.47}
\]

\[
\mathcal{G}_0 = 1 \quad \mathcal{G}_1 = -0.3 \quad \mathcal{G}_2 = 0.02 \tag{5.48}
\]
Here,
\[
\mathbf{J} = \begin{bmatrix}
1.04 & 0 & 1.09 \\
-0.2 & 0 & -0.31 \\
0 & 0 & 0.02
\end{bmatrix}
\] (5.49)
has a rank of 2. Since there are 3 unknown variables (2 in \(Q\) and 1 in \(R\)), the covariance matrix elements are not identifiable.

Note that the minimal polynomial can be used to estimate the unknown covariances \(R\) and \(Q\) via quadratic programming techniques. Furthermore, it can be used to estimate the optimal gain \(W\), as in [168] and Appendix A.4; however, reliable and accurate estimation of the parameters of vector moving average processes is still an unresolved problem [58, 87, 105, 144].

5.4 Approaches to Obtain Filter Parameters

There are two competing approaches for the estimation of the filter parameters \(W\), \(R\), \(Q\), and \(\bar{P}\). The first approach is to estimate the noise covariance matrices first and subsequently the Kalman filter gain \(W\) and the predicted state covariance \(\bar{P}\) are computed given the estimated noise covariance matrices [110, 164]. This method has an underlying problem in that it involves the sum of two moving average processes. Additionally, the autoregressive moving average (ARMA) approach, pioneered in the econometric literature, does not extend naturally to sums of moving average processes and we have found the resulting algorithms [58, 87, 105, 144] to have erratic computational behavior.

The second approach is to estimate the Kalman filter gain \(W\) from the measured
data first [31][113]. Given the optimal $W$, we can compute $R$, $Q$ and $\bar{P}$ (this approach is applied in this chapter). The proposed $R$, $Q$ and $\bar{P}$ estimates in this chapter are valid as long as an optimal gain $W$ is provided. There are many ways to obtain the optimal Kalman filter gain $W$. The techniques listed in this chapter to obtain the optimal $W$, that is, Section 5.5 and Appendix A.4 are by no means all-inclusive, and several such methods may be suitable for a given problem. For example, the optimal gain $W$ can be obtained from the suboptimal Kalman filter residual [35], solving the minimal polynomial problem [168], utilizing the least squares method on the observable form [30], and utilizing a second Kalman filter to track the error in the estimated residual of the first Kalman filter [139], to name a few.

### 5.5 Estimation of $W$

This section includes the discussion of two different approaches to estimate the optimal Kalman filter gain $W$, namely, the minimal polynomial approach and the successive approximation, coupled with an adaptive gradient descent scheme, on a criterion based on innovation correlations. The derivation of the minimal polynomial approach is detailed in Appendix A.4. This approach assumes the system to be purely driven by the optimal innovation. In doing so, the estimation of the optimal Kalman gain can be achieved via a vector auto-regressive model approximation of a vector moving average process. However, from limited testing on examples chosen in this chapter, this approach was found to be numerically unstable, only performing well on systems with no eigenvalues close to unity. In fact, the vector auto-regressive model has various numerical problems and an accurate and reliable algorithm to obtain the solution still
remains to be developed [87]. Therefore, we omit this approach from the chapter and focus on minimization of the innovation correlations using a successive approximation and adaptive gradient descent method.

In the sequel, we describe in detail the approach of our chapter using the correlation-based criterion. If the Kalman filter gain \( W \) is not optimal, the innovation sequence \( \{\nu(k)\}_{k=1}^{N} \) is correlated. We can use the innovation sequence of any stable suboptimal Kalman filter and compute \( M \) sample covariance matrices, as in [113]:

\[
\hat{C}(i) = \frac{1}{N - M} \sum_{j=1}^{N-M} \nu(j)\nu(j + i), \quad i = 0, 1, 2, \ldots, M - 1 
\]  

(5.50)

We know that the optimal Kalman filter gain \( W \) makes the autocorrelation function \( \hat{C}(i), i = 0, 1, 2, \ldots, M - 1 \) vanish for all \( i \neq 0 \). Given the correlation matrix for \( i \geq 1 \) as in [113], that is

\[
C(i) = E[\nu(k)\nu(k - i)']
\]

(5.51)

\[
= H\bar{F}^{i-1}F [\bar{P}H' - WC(0)]
\]

(5.52)

where \( \bar{F} \) is as in (5.16). We define the objective function \( J \) to be minimized as

\[
J = \frac{1}{2} \text{tr} \left\{ \sum_{i=1}^{M-1} \left[ \text{diag} \left( \hat{C}(0) \right) \right]^{-\frac{1}{2}} \hat{C}(i)' \left[ \text{diag} \left( \hat{C}(0) \right) \right]^{-1} \hat{C}(i) \left[ \text{diag} \left( \hat{C}(0) \right) \right]^{-\frac{1}{2}} \right\}
\]

(5.53)

where \( \text{diag}(C) \) is the Hadamard product of an identity matrix, of same dimension as \( C \), with \( C \)

\[
\text{diag}(C) = I \odot C
\]

(5.54)

This objective function is selected to minimize the sum of the normalized \( \hat{C}(i) \) with
respect to the corresponding diagonal elements of \( \hat{C}(0) \) for \( i > 0 \). The optimal \( J \) becomes 0 as the sample size \( N \) tends to \( \infty \) because the time averages are the same as ensemble averages given infinite data. Substituting (5.52) into (5.53) and utilizing the cyclic property of trace, we have

\[
J = \frac{1}{2} \text{tr} \left\{ \sum_{i=1}^{M-1} \Theta(i) X \mathcal{E}^2 X' \right\}
\]  

(5.55)

where

\[
\Theta(i) = \Phi(i)' \mathcal{E}^2 \Phi(i)
\]

(5.56)

\[
\Phi(i) = \bar{H} \bar{F}^{i-1} F
\]

(5.57)

\[
X = \Psi - WC(0)
\]

(5.58)

\[
\Psi = \bar{P} H'
\]

(5.59)

\[
\mathcal{E} = [\text{diag } (C(0))]^{-\frac{1}{2}}
\]

(5.60)

For ill-conditioned systems, a regularization term \( \lambda_W \text{tr}(WW') \) can be added to convexify the objective function. Taking the gradient\(^4\) of (5.55) with respect to \( W \), we get

\[
\nabla_W J = - \sum_{i=1}^{M-1} \Phi(i)' \mathcal{E}^2 C(i) \mathcal{E}^2 C(0) - F'ZF X - \sum_{\ell=0}^{i-2} [C(\ell + 1)' \mathcal{E}^2 C(i)' \mathcal{E}^2 H \bar{F}^{i-\ell-2}]'
\]

(5.61)

and \( Z \) is given by the Lyapunov equation

\[
Z = \bar{F}'Z \bar{F} + \frac{1}{2} \sum_{i}^{M-1} \Phi(i)' \mathcal{E}^2 \tilde{C}(i) \mathcal{E}^2 H + \left( \Phi(i)' \mathcal{E}^2 \tilde{C}(i) \mathcal{E}^2 H \right)'
\]

(5.62)

\(^4\)Detailed steps on the gradient computation are provided in Appendix A.5.
and $X$ is obtained by rewriting (5.52) as

$$
\begin{bmatrix}
HF \\
H\bar{F}F \\
\vdots \\
H\bar{F}^{M-1}F
\end{bmatrix} \cdot X =
\begin{bmatrix}
\hat{C}(1) \\
\hat{C}(2) \\
\vdots \\
\hat{C}(M-1)
\end{bmatrix}
$$

(5.63)

Then, we can obtain $X$ as

$$
X = \begin{bmatrix}
HF \\
H\bar{F}F \\
\vdots \\
H\bar{F}^{M-1}F
\end{bmatrix}^\dagger \begin{bmatrix}
\hat{C}(1) \\
\hat{C}(2) \\
\vdots \\
\hat{C}(M-1)
\end{bmatrix}
$$

(5.64)

where $A^\dagger$ is the pseudoinverse of $A$, defined by

$$A^\dagger = (A'A)^{-1}A'$$

(5.65)

which exists, since we assume the system to be completely observable and $M \geq n_x$.

The gradient direction can be used to obtain the optimal Kalman filter gain $W$ iteratively through the bold driver method in [12,97,177]. Details of this application can be found in Section 5.8.3.2.
5.6 Estimation of $R$

5.6.1 General $R$

Given the steady-state optimal gain $W$ and the innovation covariance $S$, whose estimation is explained later in Section 5.8, let $\mu(k), k = 1, \ldots, N$ be the sequence of post-fit residuals of the Kalman filter, that is,

\[
\mu(k) = z(k) - H\hat{x}(k|k) = (I_{n_z} - HW)\nu(k) \tag{5.66}
\]

\[
= (I_{n_z} - HW)\nu(k) \tag{5.67}
\]

Note that $(I_{n_z} - HW)$ is invertible (rank $n_z$) because $(I_{n_z} - HW) = RS^{-1}$ (proven below) and due to the assumption that $R$ is positive definite.

**Proposition 1**: Given the optimal steady-state Kalman filter gain $W$, the post-fit residual sequence $\mu(k)$, and the innovation sequence $\nu(k)$, the joint covariance of these two sequences is

\[
\text{Cov} \left( \begin{bmatrix} \nu(k) \\ \mu(k) \end{bmatrix} \right) = \begin{bmatrix} S & R \\ R & R - HPH' \end{bmatrix} \tag{5.68}
\]

**Proof**: On the right hand side of (5.68), the $(1,1)$ block is simply the definition of the innovation covariance matrix in (5.15). Using (5.67), the $(1,2)$ block in (5.68) is, given by

\[
E[\mu(k)\nu(k)'] = (I_{n_z} - HW)E[\nu(k)\nu(k)'] \tag{5.69}
\]

\[
= (I_{n_z} - HW)S \tag{5.70}
\]
Using (5.7) and (5.8),

\[ E[\mu(k)\nu(k)'] = (I_{n_x} - H\bar{P}H'S^{-1})S \]
\[ = S - H\bar{P}H' = R \]

(5.71a)
(5.71b)

The (2,2) block of (5.68) is obtained as follows.

\[ G = E[\mu(k)\mu(k)'] \]
\[ = E \{ [(I_{n_x} - HW)\nu(k)] [(I_{n_x} - HW)\nu(k)'] \} \]
\[ = (I_{n_x} - HW)S(I_{n_x} - HW)' \]
\[ = R(I_{n_x} - HW)' = R - RW'H' \]

(5.72)
(5.73)
(5.74)
(5.75)

which, given (5.14), simplifies to

\[ G = R - HPH' \]

(5.76)

Note that by using the Schur determinant identity [19, 175], the determinant of (5.68) is

\[ \begin{vmatrix} S & R \\ R & R - HPH' \end{vmatrix} = |S||G - RS^{-1}R| = 0 \]

(5.77)

where the relationship \( G = R - HPH' = RS^{-1}R \) is proved in (5.76) and Proposition 2 below.

**Proposition 2:** Given the optimal steady-state Kalman filter gain \( W \) and the corresponding post-fit residual \( \mu(k) \) and innovation \( \nu(k) \) sequences, the covariance
matrix $R$ can be computed in the following five ways:

**R1**: $R = (I_n z - HW) S$ \hspace{1cm} (5.78)

**R2**: $R = \frac{1}{2} \left\{ E[\mu(k)\nu(k)'] + E[\nu(k)\mu(k)'] \right\}$ \hspace{1cm} (5.79)

**R3**: Obtain $R$ from $G = RS^{-1} R$ \hspace{1cm} (5.80)

**R4**: $R = \frac{1}{2} \left( G + S - HW SW' H' \right)$ \hspace{1cm} (5.81)

**R5**: $R = \frac{1}{2} \left\{ G(I_n z - W'H')^{-1} + (I_n z - HW)^{-1} G \right\}$ \hspace{1cm} (5.82)

**Proof**: **R1** is proven in (5.70). Method **R2** to estimate $R$ is by symmetrizing (5.70). For method **R3** to estimate $R$, we can substitute (5.8) in (5.70) and rewrite $G$ as

$$G = R - H\bar{P}H' + HW SW' H'$$ \hspace{1cm} (5.83)

Then, by substituting (5.14) into (5.83)

$$G = R - H\bar{P}H' + HPH'S^{-1}HPH'$$ \hspace{1cm} (5.84)

We also know from (5.15)

$$(S - R) = H\bar{P}H'$$ \hspace{1cm} (5.85)

By substituting (5.85) into (5.84), we can write $G$, defined in (5.73), as

$$G = R - (S - R) + (S - R)S^{-1}(S - R)$$ \hspace{1cm} (5.86)

$$= RS^{-1} R$$ \hspace{1cm} (5.87)

$$S = RG^{-1} R$$ \hspace{1cm} (5.88)
Note that (5.87) is a continuous-time algebraic Riccati equation\(^5\). Therefore, we can estimate \(R\) by solving the continuous-time Riccati equation, as in [6], or Kleinman’s method \(^90\). Some additional methods to solve the continuous-time algebraic Riccati equation can be found in [101]. We can also interpret (5.80) in terms of a Linear Quadratic Regulator (LQR) optimal control problem, where we can obtain \(R\) as the solution of the continuous-time algebraic Riccati equation associated with the optimal gain in the LQR problem. The computation of \(R\) is also related to the simultaneous diagonalization problem\(^6\) in linear algebra [175]. Note that, in the scalar case, \(R\) is the geometric mean of the variance of the post-fit residual and the innovation, as in the (1,2) block of (5.68).

For \(R_4\), we substitute (5.85) into (5.83) and rewrite \(G\) as

\[
G = R - (S - R) + HWSW'H' \tag{5.89}
\]

\[
= 2R - S + HWSW'H' \tag{5.90}
\]

Solving for \(R\), we obtain

\[
R = \frac{1}{2} \{G + S - HWSW'H'} \tag{5.91}
\]

thus, proving \(R_4\).

\(^5R_0 + R_0 - RS^{-1}R + G = 0\)

\(^6\)The solution via Cholesky decomposition and eigen decomposition or simultaneous diagonalization can be found in Appendix A.6 and Appendix A.7, respectively.
For $\mathbf{R5}$, recall (5.70). We can rewrite (5.74) as

\begin{align*}
G &= (I_{n_z} - HW)S(I_{n_z} - HW)' \\ 
&= R(I_{n_z} - HW)' = (I_{n_z} - HW)R \\
\end{align*}

Thus, we can compute $R$ as

\begin{align*}
\hat{R} &= (I_{n_z} - HW)^{-1} G \\ 
&= G (I_{n_z} - W'H')^{-1} \\
\end{align*}

We can symmetrize the estimate of $R$ by

\begin{equation}
\hat{\hat{R}} = \frac{1}{2} \left\{ G (I_{n_z} - W'H')^{-1} + (I_{n_z} - HW)^{-1} G \right\} 
\end{equation}

proving $\mathbf{R5}$.

Note that $\mathbf{R1-R5}$ are theoretically the same; however, they are numerically different. We recommend $\mathbf{R3}$, since it ensures the positive definiteness of $R$.

### 5.6.2 Diagonal $R$

When $R$ is diagonal, we solve the least squares problem of

\[
\min_{X \geq 0} ||X - R||_F^2
\]

where $F$ indicates the Frobenius norm. The positive definite $R$ can be estimated from $\mathbf{R3}$, given in Proposition 2. The solution is simply the diagonal elements of the estimated $R$ from $\mathbf{R3}$. This can also be interpreted as the masking operation to
impose structural constraints on $R$, as discussed in the context of the estimation of $Q$ in Section 5.7.

5.6.3 Use of Smoothed State Estimate with One-Step-Lag Post-Fit Residuals

Note that $R$ can also be estimated using one-step-lag smoothing on the post-fit residuals. Let us define the one-step-lag smoothed residual $s(k)$ as in [123], that is,

$$s(k) = z(k) - H\hat{x}(k|k + 1)$$  \hspace{1cm} (5.98)

$$= z(k) - H\hat{x}(k|k) - HW_1\nu(k + 1)$$  \hspace{1cm} (5.99)

$$W_1 = \bar{P}\bar{F}^TP^{-1}W$$  \hspace{1cm} (5.100)

where $\bar{F}$ is defined as

$$\bar{F} = (I_{nx} - WH)F = F^{-1}FF$$  \hspace{1cm} (5.101)

From (5.66), we can also write $s(k)$ as a one-step moving average process

$$s(k) = \mu(k) - HW_1\nu(k + 1)$$  \hspace{1cm} (5.102)

$$= (I_{nx} - HW)\nu(k) - HW_1\nu(k + 1)$$  \hspace{1cm} (5.103)

Therefore,

$$E[s(k)\nu(k)'] = (I_{nx} - HW)C(0) - HW_1C(1)'$$  \hspace{1cm} (5.104)

and for the optimal Kalman filter gain $W$, we can write (5.104) as

$$E[s(k)\nu(k)'] = (I_{nx} - HW)S = R$$  \hspace{1cm} (5.105)
A similar expression can be derived for $E[s(k)\mu(k)']$, that is,

$$E[s(k)\mu(k)'] = (I_{nz} - HW)C(0)(I_{nz} - HW)' - HW_1C(1)'(I_{nz} - HW)'$$  \hspace{1cm} (5.106)$$

and for the optimal Kalman filter gain $W$, we have

$$E[s(k)\mu(k)'] = (I_{nz} - HW)S(I_{nz} - HW)'$$

$$= RS^{-1}R = G$$  \hspace{1cm} (5.107)$$

Lastly, the expression for $E[s(k)s(k)']$ is

$$E[s(k)s(k)'] = (I_{nz} - HW)C(0)(I_{nz} - HW)' + HW_1C(0)W_1'H'$$

$$- (I_{nz} - HW)C(1)(W_1)'H' - HW_1C(1)'(I_{nz} - HW)'$$  \hspace{1cm} (5.108)$$

and with the optimal Kalman filter gain $W$, combined with (5.14), we get

$$E[s(k)s(k)'] = RS^{-1}R + H\bar{P}\bar{F}'\bar{P}^{-1}WSW'\bar{P}^{-1}WSW'\bar{P}^{-1}\bar{F}\bar{P}H'$$  \hspace{1cm} (5.109)$$

$$= RS^{-1}R + RW'F'(I_{nz} - H'W'H'R^{-1}SR^{-1}H(I_{nz} - WH)FWR$$  \hspace{1cm} (5.110)$$

$$= RS^{-1}R + RW'F'H'S^{-1}SS^{-1}HFWR$$  \hspace{1cm} (5.111)$$

$$= R(S^{-1} + W'F'H'S^{-1}HFW)R$$  \hspace{1cm} (5.112)$$

Note that $E[s(k)s(k)']$ can be used in a manner similar to the algorithm in Section 5.5 to obtain the optimal Kalman filter gain $W$. More investigation is needed into this approach.
5.7 Estimation of Q, P and \( \bar{P} \)

In this section, we discuss a method to estimate the process noise covariance \( Q \) and the state prediction (updated) covariance \( \bar{P} \) (\( P \)). Unlike the case of a Wiener process and for a process with \( H = I \), where both \( Q \) and \( \bar{P} \) can be estimated separately and without iteration, as shown in Section 5.9.1.3, \( Q \) and \( \bar{P} \) (\( P \)) are coupled in the general case, requiring multiple iterations for the estimation to converge. The relationship between the steady-state state prediction covariance matrix \( \bar{P} \) and the steady-state updated state covariance matrix \( P \) with the process noise covariance matrix \( Q \) is

\[
\bar{P} = FPF' + \Gamma Q\Gamma' \\
= F(\bar{P}^{-1} + H' R^{-1} H)^{-1} F' + \Gamma Q\Gamma' \\
= FPF' + FWRW'F' + \Gamma Q\Gamma' 
\]

Similarly, the steady-state updated state covariance matrix can be written as

\[
P = \bar{P} = FP\bar{F}' + WRW' + (I_{n_x} - WH)\Gamma Q\Gamma'(I_{n_x} - WH)' \\
= (\bar{P}^{-1} + H' R^{-1} H)^{-1} \\
= \left[(FPF' + \Gamma Q\Gamma')^{-1} + H' R^{-1} H\right]^{-1}
\]

where \( \bar{F} \) is defined as in (5.101) and (5.117) is derived utilizing (5.14) and the fact (from [11]) that

\[
P = (I_{n_x} - WH)\bar{P} 
\]
We also define \( \tilde{P} \) as

\[
\tilde{P} \triangleq FPF' = \tilde{F} \tilde{P} \tilde{F}' + FWRW'F' + \tilde{F} \Gamma Q \Gamma' \tilde{F}'
\]

(5.120)

Given \( \tilde{P} \) and \( S \), or \( P \) and \( S \), or \( \tilde{P} \) and \( S \), we can compute \( \Gamma Q \Gamma' \) in the following ways:

**Q1** : \( \Gamma Q \Gamma' = F^{-1} \tilde{P} (F^{-1})' + WSW' - \tilde{P} \)

(5.121)

**Q2** : \( \Gamma Q \Gamma' = P + WSW' - FPF' \)

(5.122)

**Q3** : \( \Gamma Q \Gamma' = \tilde{P} - F\tilde{P}F' + FWSW'F' \)

(5.123)

where **Q1** - **Q3** are derived from (5.6).

In this chapter, we utilize the updated state covariance matrix to estimate \( Q \) and \( P \), iteratively. Let \( t = 0, 1, \ldots \) and \( \ell = 0, 1, \ldots \) denote the (two loop) iteration indices, and let us assume the initial estimate \( \Gamma Q^{(0)} \Gamma' = WSW' \) (this is the Wiener process solution for the estimation of \( Q \), as shown in Section 5.9). Let us initialize \( P \) by solving the Lyapunov equation (starting with \( t = 0 \) and \( \ell = 0 \))

\[
P^{(0)} = \tilde{F} P^{(0)} \tilde{F}' + WRW' + (I_{nx} - WH) \Gamma Q^{(t)} \Gamma' (I_{nx} - WH)' \]

(5.124)

for \( P^{(0)} \). We compute \( P^{(\ell+1)} \) utilizing (5.118) until the value converges, that is,

\[
P^{(\ell+1)} = \left[ (FP^{(t)}F' + \Gamma Q^{(t)} \Gamma')^{-1} + H'R^{-1}H \right]^{-1}
\]

(5.125)

Given the converged \( P \), let us denote \( D^{(t+1)} \) as

\[
D^{(t+1)} = P + WSW' - FPF'
\]

(5.126)
Then, we can update \( Q^{(t+1)} \) from (5.122)

\[
Q^{(t+1)} = \Gamma^\dagger D^{(t+1)}(\Gamma')^\dagger \quad (5.127)
\]

A mask matrix \( A \) can shape \( Q \) to enforce the structural constraints (e.g., diagonal covariance). The mask matrix comprises binary matrix elements with a 1 in the desired positions and 0, elsewhere, for example, as in an identity matrix. Then \( Q \) is structured by

\[
Q^{(t+1)} = A \odot Q^{(t+1)} \quad (5.128)
\]

where \( \odot \) is the Hadamard product. We subsequently set \( \ell = 0 \) and recompute \( P \) using \( Q^{(t+1)} \) in (5.125), and this process repeats until the estimate of \( Q \) converges. For ill-conditioned systems, a tuning (regularization) parameter \( \lambda_Q \) can be used in (5.127), that is

\[
Q^{(t+1)} = \Gamma^\dagger \left[ D^{(t+1)} + \lambda_Q I_{n_x} \right] (\Gamma')^\dagger \quad (5.129)
\]

After the estimate of \( Q \) converges, we can estimate \( \bar{P} \) using either (5.113), (5.114) or (5.115).

### 5.8 Iterative Algorithm to Estimate Steady-State \( W, S, P (\bar{P}), Q \) and \( R \)

Given the methods to obtain estimates of \( R \) and \( Q \) in Sections 5.6 and 5.7, we summarize our method into a six-step solution approach to obtain the optimal steady-state \( W, S, P (\bar{P}), Q, \) and \( R \).
5.8.1 Step 1

Start with iteration $r = 0$ and initialize with a $W^{(0)}$ to stabilize the system as in [91].

We execute the Kalman filter for samples $k = 1, 2, \ldots, N$ as

$$\hat{x}^{(r)}(k+1|k) = F\hat{x}^{(r)}(k|k)$$  \hfill (5.130)

$$\nu^{(r)}(k + 1) = z(k + 1) - H\hat{x}^{(r)}(k + 1|k)$$  \hfill (5.131)

$$\hat{x}^{(r)}(k + 1|k + 1) = \hat{x}^{(r)}(k + 1|k) + W^{(r)}\nu^{(r)}(k + 1)$$  \hfill (5.132)

$$\mu^{(r)}(k + 1) = z(k + 1) - H\hat{x}^{(r)}(k + 1|k + 1)$$  \hfill (5.133)

5.8.2 Step 2

Compute $M$ sample covariance matrices, as in [5.50].

5.8.3 Step 3

In this step, we check whether any of the termination conditions given below are met. If none of the termination conditions are met, we update the Kalman filter gain via the proposed method, detailed later in Section 5.8.3.2.

5.8.3.1 Termination Conditions

There are five conditions that result in algorithm termination, subsequently yielding a Kalman filter gain $W$ for $R$, $Q$ and $\bar{P}$ estimates in later steps:

Condition 1: The converged Kalman filter gain is within a specified threshold $\zeta_W$.

Condition 2: The gradient of Kalman filter gain [5.61] is within a specified threshold
Condition 3: The objective function value in (5.53) is within a specified threshold $\zeta_J$ from zero.

Condition 4: The objective function value stops improving, given a specified “patience” (number of epochs, detailed in Section 5.8.3.2) for the adaptive gradient method.

Condition 5: The maximum number of iterations is reached.

5.8.3.1.1 Condition 1: Let $\Delta W$ be the change in the Kalman filter gain from iteration $r$ to $r + 1$, that is

$$
\Delta W = W^{(r+1)} - W^{(r)}
$$

(5.134)

then

$$
\delta_W = ||\Delta W ./ (W^{(r)} + \epsilon_W)||
$$

(5.135)

where ./ indicates element-wise division and $||\cdot||$ is a matrix norm (In this chapter, the authors use the Euclidean norm) and $\epsilon_W$ is a very small value to protect against zeros in the denominator. When $\delta_W$ is less than a specified threshold $\zeta_W$, the Kalman filter gain is assumed to have converged and we terminate the algorithm; otherwise, we update the Kalman filter gain $W$ for the next iteration.

5.8.3.1.2 Condition 2: We also examine the gradient of the Kalman filter gain $\nabla W J$ for convergence. We assume the Kalman filter gain to be converged when
the Euclidean norm of $\nabla_W J$ is less than a sufficiently small threshold $\zeta_\Delta$, that is,

$$\|\nabla_W J\|_2 < \zeta_\Delta$$

(5.136)

5.8.3.1.3 **Condition 3:** Similar to $W$, we can compute the change in the objective function $J$ from iteration $r$ to $r + 1$. The Kalman filter gain is assumed to have converged when $J^{(r)}$ is less than a specified threshold $\zeta_J$; otherwise, we update the Kalman filter gain for the next iteration.

5.8.3.1.4 **Condition 4:** The fourth termination condition is related to the step size for the proposed approximation method. We adapt the bold driver method in [12,97,177] and the method considers a “patience” parameter to indicate that the objective function value $J^{(r)}$ has stopped improving (detailed in Section 5.8.3.2). The algorithm is terminated with the Kalman filter gain corresponding to minimum $J^{(r)}$.

5.8.3.1.5 **Condition 5:** This condition is implemented to ensure that the algorithm terminates within a reasonable number of iterations, denoted by $n_L$. Typically, the number of iterations required to reach the optimal Kalman filter gain $W$ increases proportionally with $n_x$.

5.8.3.2 **Kalman Filter Gain Update**

When any of the above conditions are met, we terminate the algorithm. Otherwise, we update the Kalman filter gain $W$ for the next iteration $r + 1$ via the gradient direction in (5.61). Given the gradient direction, the Kalman filter gain at iteration $r + 1$ is
updated by

\[ W^{(r+1)} = W^{(r)} - \alpha^{(r)} \nabla_{W} J \tag{5.137} \]

where \( \alpha^{(r)} \) is the step size for the proposed method. The step size is initialized as

\[ \alpha^{(0)} = \min \left( c \left( \frac{N}{N_s} \right)^\beta, c \right) \tag{5.138} \]

where \( c \) is a positive constant and is used to update the Kalman filter gain in the first iteration, \( N_s \) is a hyperparameter on the number of observations, and \( \beta \) is a positive constant to adapt the initial step size to the number of observations. Note that (5.138) is selected to automatically tune the initial step size. When only a small subset of samples are observed, we want to use a smaller step size to prevent large steps that could result in unstable gains. If a line search is used instead, initialization is not necessary. Use of stochastic approximation type step sizes will enable one to extend the estimation method to on-line situations and the extended Kalman filter.

Subsequently, \( \alpha^{(r)} \) is computed using the bold driver method in [12, 97, 177]. That is, after each iteration, we compare the \( J^{(r)} \) to its previous value, \( J^{(r-1)} \), and set

\[ \alpha^{(r)} = \begin{cases} 0.5\alpha^{(r-1)}, & \text{if } J^{(r)} > J^{(r-1)} \\ \max(1.1\alpha^{(r-1)}, \bar{c}), & \text{otherwise} \end{cases} \tag{5.139} \]

where \( \bar{c} \) is the maximum step size defined as,

\[ \bar{c} = \min \left( \left( \frac{N}{N_s} \right)^\beta, c_{\max} \right) \tag{5.140} \]

and \( c_{\max} \) is a positive constant between 0 and 1.
Once we update the Kalman filter gain $W$, we go back to Step 1 by setting $r = r + 1$ and repeat the same process until any of the five termination conditions are met.

Note that each time $J^{(r)} \leq J^{(r-1)}$, we save the corresponding Kalman filter gain $W^{(r)}$ and $J^{(r)}$, and we halve the step size each time $J^{(r)} > J^{(r-1)}$ in the hope of observing a decrease in $J^{(r)}$. If the value of $J^{(r)}$ has consecutively increased for a specified number of iterations (i.e., given a “patience” factor), we select the best Kalman filter gain $W$ by

$$ W = \arg \min_r J^{(r)} $$

(5.141)

We then terminate the iteration and move onto Step 4 after repeating Steps 1 and 2 with the corresponding $W$. Note that adaptive stochastic gradient descent methods can be applied to compute the optimal Kalman filter gain $W$ as in $[80, 89, 127, 169, 183]$.

5.8.4 Step 4

Once we obtain the optimal steady-state Kalman filter gain $W$ and the corresponding innovation covariance $S$, we can compute the unknown $R$, as in Section 5.6.

5.8.5 Step 5

Given the covariance matrix $R$, computed in Step 4, we can compute the covariance matrix $Q$ and steady-state state prediction covariance matrix $\bar{P}$, as detailed in Section 5.7.
5.8.6 Step 6

We implement a successive approximation as follows: an outer-loop is used to reinitialize with the $R$ and $Q$ obtained from Step 5 and then reinvoke Steps 1-5. We keep track of the best $J^{(r)}$ among the outer-loop iterations. The Kalman filter gain associated with the minimum $J^{(r)}$ is selected to be the optimal Kalman filter gain. The algorithm terminates when the difference between the best $J^{(r)}$ from each outer-loop is less than $\zeta_J$ or the maximum number of outer-loop iterations is reached.

5.9 Special Cases: Wiener Process and $H = I_{n_x}$ cases

In this section, we consider two special cases below. The first case is when the state transition matrix $F$ and the measurement matrix $H$ are both identity matrices, $I_{n_x}$ and $I_{n_z}$, where $n_x = n_z$. This considerably simplifies our method to estimate $R$ and $Q$. The second special case is when only the measurement matrix $H$ is the identity matrix, while the state transition matrix $F$ remains general. Note that we can extend either case to that of one assuming perfect measurements, that is, when $H = I_{n_x}$, we have no measurement noise, and thus, $R = 0$.

5.9.1 Case 1: Wiener Process

For a Wiener process, we have $F = I_{n_x}$ and $H = I_{n_x}$.
5.9.1.1 Kalman Filter Gain Update for a Wiener Process

To get the optimal Kalman filter gain, for \( k = 1, 2, \ldots, N \),

\[
\dot{x}(k|k-1) = \dot{x}(k-1|k-2) + W \nu(k-1) \tag{5.142}
\]

\[
\tilde{z}(k|k-1) = \hat{x}(k|k-1) \tag{5.143}
\]

\[
z(k) = \tilde{z}(k|k-1) + \nu(k) = \hat{x}(k|k-1) + \nu(k) \tag{5.144}
\]

Define

\[
\xi(k) = z(k) - z(k-1) \tag{5.145}
\]

\[
= \hat{x}(k|k-1) + \nu(k) - \hat{x}(k-1|k-2) - \nu(k-1) \tag{5.146}
\]

\[
= \nu(k) + (W - I_{n_x}) \nu(k-1) \tag{5.147}
\]

Then, let us define \( L_0 \) and \( L_1 \) as

\[
L_0 = E[\xi(k)\xi(k)'] = S + (W - I_{n_x})S(W - I_{n_x})' \tag{5.148}
\]

\[
L_1 = E[\xi(k)\xi(k-1)'] = (W - I_{n_x})S \tag{5.149}
\]

Note that both \( L_0 \) and \( L_1 \) can be computed from samples. Additionally, we can obtain the optimal \( W \) from \( L_1 \) as

\[
W = I_{n_x} + L_1S^{-1} \tag{5.150}
\]
Substituting $W$ in (5.150) into (5.148), we can write the relationship between $L_0$ and $L_1$ as

$$L_0 = S + L_1S^{-1}SS^{-1}L'_1$$  \hspace{1cm} (5.151)

$$= S + L_1S^{-1}L'_1$$  \hspace{1cm} (5.152)

Note that (5.152) is in a form related to the discrete algebraic Riccati equation and has a positive definite solution [53].

5.9.1.2 Estimation of $R$ for a Wiener Process

**Proposition 3**: For a Wiener process where both the state transition matrix $F$ and the measurement matrix $H$ are both the identity matrices, $I_{n_x}$ and $I_{n_z}$, respectively, where $n_x = n_z$, and given the optimal steady-state Kalman filter gain $W$, and the concomitant post-fit residual sequence $\mu(k)$ and innovation sequence $\nu(k)$, the covariance matrix $R$ can be computed in the following ways:

**SR1**: $R = (I_{n_z} - W)S$  \hspace{1cm} (5.153)

**SR2**: $R = \frac{1}{2} \{ E[\mu(k)\nu(k)'] + E[\nu(k)\mu(k)'] \}$  \hspace{1cm} (5.154)

**SR3**: $G = RS^{-1}R$  \hspace{1cm} (5.155)

**SR4**: $R = \frac{1}{2} [G + S - WSW'']$  \hspace{1cm} (5.156)

**SR5**: $R = G - WSW'' + \frac{1}{2}(WS + S'W')$  \hspace{1cm} (5.157)

**Proof**: SR1-SR4 are directly proven by substituting $H = I_{n_z}$ into R1-R4. For
SR5, we know from (5.8) that

\[ WS = \bar{P} \]  

(5.158)

and we also know from (5.15) that,

\[ S = \bar{P} + R \]  

(5.159)

Then,

\[ G = (I_{n_z} - W)S(I_{n_z} - W)' \]  

(5.160)

\[ = (I_{n_z} - W)S - (I_{n_z} - W)SW' \]  

(5.161)

\[ = S - WS - SW' + WS \]  

(5.162)

\[ = R - SW' + WSW' \]  

(5.163)

Then, we can compute \( R \) as

\[ R = G + WS - WSW' \]  

(5.164)

Symmetrizing (5.164),

\[ R = G - WSW' + \frac{1}{2}(WS + S'W') \]  

(5.165)

hence, \textbf{SR5} is proven.
5.9.1.3 Estimation of $\bar{P}$ and $Q$ for a Wiener Process

Unlike the general case, where multiple iterations are needed to estimate both $Q$ and $\bar{P}$, in the case of a Wiener process, we can estimate them with no iteration.

**Proposition 4:** For a Wiener process, where the state transition matrix $F$ and the measurement matrix $H$ are both identity matrices, $I_{n_x}$ and $I_{n_z}$, respectively, and given the optimal steady-state Kalman filter gain $W$, and the corresponding innovation sequence $\nu(k)$, the steady-state state prediction covariance and the process noise covariance $Q$ can be computed as:

\[
\bar{P} = WS \\
Q = WSW' \tag{5.166}
\]

**Proof:** Given the relationship in (5.8) and knowing that, for a Wiener process $H = I_{n_z}$, using (5.8), we have (5.166).

For a Wiener process, we can rewrite the Riccati equation (5.10) as

\[
P = P - P(P + R)^{-1}P + Q \tag{5.168}
\]

Using the relationship of (5.15) and (5.166) in (5.168) yields

\[
P = P - WSW' + Q \tag{5.169}
\]
Thus, for a Wiener process, $Q$ can be estimated as

$$Q = WSW'$$  \hspace{1cm} (5.170)

Hence, (5.167) is proven. Note that (5.167) is used as $Q^{(0)}$ in the general case for iteratively computing $Q$. Also note that when $R = 0$ (i.e., perfect measurement case), we have,

$$W = I_{nx}$$ \hspace{1cm} (5.171)

$$P = 0$$ \hspace{1cm} (5.172)

$$G = 0$$ \hspace{1cm} (5.173)

$$Q = S = \bar{P}$$ \hspace{1cm} (5.174)

5.9.2 Case 2: $H = I_{nx}$

In the second case, only $H$ is the identity matrix, but $F$ is not necessarily so.

5.9.2.1 Kalman filter Gain Update for the $H = I_{nx}$ Case

To get the optimal Kalman filter gain, for $k = 1, 2, \ldots, N$,

$$\hat{x}(k + 1|k) = F\hat{x}(k|k - 1) + FW\nu(k)$$ \hspace{1cm} (5.175)

$$z(k) = x(k) + w(k)$$

$$= \hat{x}(k|k - 1) + \nu(k)$$ \hspace{1cm} (5.176)
Let $\xi(k)$ be

$$\xi(k) = z(k) - Fz(k - 1) \quad (5.177)$$

Define

$$\xi(k) = \hat{x}(k | k - 1) + \nu(k) - F\hat{x}(k - 1 | k - 2) - F\nu(k - 1) \quad (5.178)$$

$$= \nu(k) - \bar{F}\nu(k - 1) \quad (5.179)$$

where

$$\bar{F} = F(I_{nx} - W) \quad (5.180)$$

We can write $L_0 = E\{\xi(k)\xi(k)\}'$ as

$$L_0 = S + \bar{F}SF' \quad (5.181)$$

Similarly, $L_1 = E\{\xi(k)\xi(k - 1)\}'$ can be computed based on (5.179) as,

$$L_1 = -\bar{F}S \quad (5.182)$$

From the right hand side of (5.181), we can find $S$ by solving

$$S + L_1S^{-1}L_1' = L_0 \quad (5.183)$$

Upon calculating $S$, we can find the optimal Kalman filter gain $W$ as

$$W = I_{nx} + F^{-1}L_1S^{-1} \quad (5.184)$$
and we can calculate $R$ from $R_3$, in (5.87). $G$ can be obtained by running the filter given the optimal Kalman filter gain. Note that, we can also write $\xi(k)$ as

$$\xi(k) = x(k) - Fx(k-1) + w(k) - Fw(k-1)$$ (5.185)

$$= \Gamma v(k-1) + w(k) - Fw(k-1)$$ (5.186)

Then, $L_0$ is

$$L_0 = \Gamma Q\Gamma' + R + FRF'$$ (5.187)

Equating (5.181) and (5.187), we can compute $\Gamma Q\Gamma'$ as

$$\Gamma Q\Gamma' = S + \bar{F}FSF' - (R + FRF')$$ (5.188)

$$= S + FGF' - (R + FRF')$$ (5.189)

Equation (5.189) follows from

$$G = (I_{n_z} - W)S(I_{n_z} - W)'$$ (5.190)

Note that when $R = 0$ (i.e., perfect measurement case), we have $W$, $P$ and $G$ as in (5.171)–(5.173), respectively, and

$$L_0 = S = \Gamma Q\Gamma' = \bar{P}$$ (5.191)

### 5.10 Numerical Examples

In this section, we illustrate our method on the following five cases:
1. A second-order kinematic system (a white noise acceleration or nearly constant velocity model) by varying the lags $M$ in the correlation.

2. A system described in Neethling [126].

3. A five-state system from [113] and [14] with diagonal $Q$ and $R$.

4. A detectable, but not completely observable, system from [132].

5. A three-state system from [132].

Each case is simulated with 100 Monte Carlo (MC) runs with an assumed “patience” of 5, $\zeta_J = 10^{-6}$, $\zeta_W = 10^{-6}$, $\zeta_\Delta = 10^{-6}$, $c = 0.01$, $c_{\text{max}} = 0.2$, $\beta = 2$ and the maximum outer-loop iteration limit is set to 20. Case 5 is simulated with 200 MC runs to be compatible with the results in [132].

For each test case, we examine the condition number of the system’s observability and controllability matrices, as well as matrix $\mathcal{J}$. The condition number of matrix $A$ is computed as

$$\kappa(A) = \|A\|\|A^\dagger\|$$

where $A^\dagger$ is the pseudoinverse of $A$ and $\|\cdot\|$ is a Euclidean norm. The rank of matrix $\mathcal{J}$ is also examined for each test case. For each test case result, we compute the 95% probability interval (PI) via the highest probability interval method and denote by $r$ and $\bar{r}$ the corresponding lower and upper limits, respectively. We also provide the mean and the root mean squared error (RMSE) of each distribution. The averaged normalized innovation squared (NIS) is also provided to measure the consistency of

---

7 The highest probability interval is, assuming unimodality, the minimum width interval such that the estimates of the parameter within the interval have a specified higher probability density than points outside of the interval.
the Kalman filter,
\[ \hat{\epsilon}(k) = \frac{1}{n_{MC}} \sum_{i=1}^{n_{MC}} \nu(k)'S^{-1}\nu(k) \tag{5.193} \]
where \( n_{MC} \) is the number of MC runs. The elements of each matrix \( A \) are denoted as \( a_{ij} \), representing the element in the \( i^{th} \) row and the \( j^{th} \) column of \( A \).

### 5.10.1 Case 1

We simulated a second-order kinematic system described by

\[
x(k) = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} x(k-1) + \begin{bmatrix} \frac{1}{2}T^2 \\ T \end{bmatrix} v(k-1) \tag{5.194}
\]
\[
z(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(k) + w(k) \tag{5.195}
\]

with sampling period \( T = 0.1 \), where

\[
E[v(k)v(j)'] = 0.0025\delta_{kj} \tag{5.196}
\]
\[
E[w(k)w(j)'] = 0.01\delta_{kj} \tag{5.197}
\]

where \( \delta_{kj} \) is the Kronecker delta function. The mean of the process and the measurement noises are assumed to be zero and the corresponding variances are given in (5.196) and (5.197), respectively. Note that the system has the condition number of 20.1 for its observability matrix and 20.2 for its controllability matrix. The noise
covariance identifiability matrix $J$, given the initial Kalman filter gain in (5.199), is

$$J = \begin{bmatrix} 5 \cdot 10^{-5} & 6 \\ 2.5 \cdot 10^{-5} & -4 \\ 0 & 1 \end{bmatrix}$$  \hspace{1cm} (5.198)$$

which has a rank of 2, and we have 2 unknown variables to estimate, implying that $Q$ and $R$ are identifiable. The condition number for $J$ is $1.5 \cdot 10^5$. The least squares problem using the minimal polynomial approach is ill-conditioned.

5.10.1.1 Varying the Number of Lags in the Correlations

We performed 100 MC runs, where each run contained $N = 1000$ sample observations. We set $n_L = 100$, $N_s = 1000$, and vary the lags, $M = 10, 20, 30, 40, 50, 100$, with an initial Kalman Filter gain $W^{(0)} = \begin{bmatrix} 0.1319 \\ 0.0932 \end{bmatrix}$, obtained by solving the Riccati equation with $Q^{(0)} = 0.1$ and $R^{(0)} = 0.1$. Figs.5.1 and 5.2 show the box plots of the estimated $R$ using R3 and $Q$ of 100 MC runs, respectively, with varying $M$.

The bottom and top of each “box” are the first (denoted $Q_1$) and third (denoted $Q_3$) quartiles of the estimate, respectively. The line in the middle of each box is the median estimate. The distances between the tops and bottoms are the interquartile ranges ($IQR = Q_3 - Q_1$). The whiskers are lines extending above and below each box and are drawn from each end of the interquartile ranges to the upper ($Q_3 + 1.5IQR$)

\footnote{All (R1–R5) obtain the same values.}
Figure 5.1: 100 Monte Carlo runs for the Kalman filter $R$ estimation using method $\textbf{R3}$ with various $M$.

Table 5.2

| Monte Carlo Simulation for Case 1 Varying the Number of Lags $M$ (Method $\textbf{R3}$) |
|-----------------------------------|---|---|---|---|---|---|
|                                  | 10 | 20 | 30 | 40 | 50 | 100 |
| $R$                              | 0.0100 | 0.0100 | 0.0100 | 0.0100 | 0.0100 | 0.0100 |
| $Q$                              | 0.0048 | 0.0030 | 0.0027 | 0.0026 | 0.0025 | 0.0025 |

and lower ($Q_1 - 1.5\text{IQR}$) adjacent values. Estimates beyond the whisker length are marked as outliers (indicated by the “+” symbols). The accuracies of the estimates of both $R$ and $Q$ increase with an increase in $M$. Table 5.2 shows the mean value of the estimates of both $R$ and $Q$. The smallest error of the median of the estimates of $R$ and the variability of the estimates of $Q$ are obtained with $M \geq 100$. 

168
Figure 5.2: 100 Monte Carlo runs for the Kalman filter $Q$ estimation with various $M$.

5.10.1.2 Estimation of $W$ and $\bar{P}$

Given $M = 100$, for 100 MC runs with the initial Kalman Filter gain as in (5.199), we found that $R_1$–$R_5$ estimate the same $R$ values. The true values of $R$ all lie within the 95% PI associated with the distribution of estimates. Fig.5.3 shows the $Q$ versus $R$ plot of each estimate. The true values are marked by “+” symbols. The reason the estimated $Q$ varies so much is that its value is very small compared to the measurement noise. Fig.5.4 shows the averaged NIS and its 95% probability region, which proves that the filter is consistent.
### Table 5.3
**Monte Carlo Simulation for Case 1 with $M = 100$ and PI = 2σ (100 Runs)**

<table>
<thead>
<tr>
<th></th>
<th>$R$</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td>R4</td>
<td>R5</td>
</tr>
<tr>
<td>Truth</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>0.0092</td>
<td>0.0092</td>
<td>0.0092</td>
<td>0.0092</td>
<td>0.0092</td>
</tr>
<tr>
<td>Mean</td>
<td>0.0100</td>
<td>0.0100</td>
<td>0.0100</td>
<td>0.0100</td>
<td>0.0100</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>0.0109</td>
<td>0.0109</td>
<td>0.0109</td>
<td>0.0109</td>
<td>0.0109</td>
</tr>
<tr>
<td>RMSE</td>
<td>$4.41 \cdot 10^{-4}$</td>
<td>$4.41 \cdot 10^{-4}$</td>
<td>$4.41 \cdot 10^{-4}$</td>
<td>$4.41 \cdot 10^{-4}$</td>
<td>$4.41 \cdot 10^{-4}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>$Q$</th>
<th></th>
<th>-------</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_{11}$</td>
<td></td>
<td></td>
<td></td>
<td>$W_{21}$</td>
<td>0.0100</td>
</tr>
<tr>
<td>Truth</td>
<td>0.0952</td>
<td></td>
<td>0.0476</td>
<td></td>
<td>-------</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>0.0697</td>
<td></td>
<td>0.0255</td>
<td></td>
<td>-------</td>
</tr>
<tr>
<td>Mean</td>
<td>0.0925</td>
<td></td>
<td>0.0465</td>
<td></td>
<td>-------</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>0.1250</td>
<td></td>
<td>0.0634</td>
<td></td>
<td>-------</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.0147</td>
<td></td>
<td>0.0100</td>
<td></td>
<td>-------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>$\bar{P}_{11}$</th>
<th></th>
<th></th>
<th>$\bar{P}_{22}$</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth</td>
<td>0.0011</td>
<td></td>
<td>5.13 $\cdot 10^{-4}$</td>
<td></td>
<td></td>
<td>5.13 $\cdot 10^{-4}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>8.47 $\cdot 10^{-4}$</td>
<td></td>
<td>2.82 $\cdot 10^{-4}$</td>
<td></td>
<td></td>
<td>2.82 $\cdot 10^{-4}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>0.0011</td>
<td></td>
<td>5.13 $\cdot 10^{-4}$</td>
<td></td>
<td></td>
<td>5.13 $\cdot 10^{-4}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>0.0013</td>
<td></td>
<td>8.72 $\cdot 10^{-4}$</td>
<td></td>
<td></td>
<td>8.72 $\cdot 10^{-4}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>1.26 $\cdot 10^{-4}$</td>
<td></td>
<td>1.60 $\cdot 10^{-4}$</td>
<td></td>
<td></td>
<td>1.60 $\cdot 10^{-4}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.10.2 Case 2

We simulated the system described in Neethling [126],

\[
x(k) = \begin{bmatrix} 0.8 & 1 \\ -0.4 & 0 \end{bmatrix} x(k-1) + \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} v(k-1) \tag{5.200}
\]

\[
z(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(k) + w(k) \tag{5.201}
\]

where

\[
E[v(k)v(j)'] = \delta_{kj} \tag{5.202}
\]

\[
E[w(k)w(j)'] = \delta_{kj} \tag{5.203}
\]
The system’s condition numbers for its observability and controllability matrices are 2.18 and 2.56, respectively. Here, $\mathcal{J}$, given the initial Kalman filter gain, is

$$\mathcal{J} = \begin{bmatrix} 1.25 & 1.8 \\ 0.5 & -1.12 \\ 0 & 0.4 \end{bmatrix} \quad (5.204)$$

and the rank is 2. The number of unknown variables is 2, therefore, the system noise variances are estimable. The condition number of $\mathcal{J}$ is 2.3 and indeed the minimal polynomial approach works well for this problem. We simulated 100 Monte Carlo runs with $N = 1000$, $n_L = 100$, $N_s = 1000$, and an initial suboptimal Kalman filter gain

$$W^{(0)} = \begin{bmatrix} 0.9 \\ 0.5 \end{bmatrix} \quad (5.205)$$
Table 5.4
Monte Carlo Simulation for Case 2 with $M = 100$ and PI = $2\sigma$ (100 Runs)

<table>
<thead>
<tr>
<th></th>
<th>$R$</th>
<th>$Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>Truth</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>0.56</td>
<td>0.56</td>
</tr>
<tr>
<td>Mean</td>
<td>1.05</td>
<td>1.05</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>1.51</td>
<td>1.51</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$W_1$</th>
<th>$W_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth</td>
<td>0.65</td>
<td>0.09</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>0.49</td>
<td>−0.03</td>
</tr>
<tr>
<td>Mean</td>
<td>0.63</td>
<td>0.10</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>0.80</td>
<td>0.25</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.08</td>
<td>0.07</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$\bar{P}_{11}$</th>
<th>$\bar{P}_{22}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth</td>
<td>1.89</td>
<td>0.35</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>1.59</td>
<td>0.30</td>
</tr>
<tr>
<td>Mean</td>
<td>1.87</td>
<td>0.35</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>2.07</td>
<td>0.39</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.12</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Table 5.4 shows the estimated noise variances. Similar to the Case 1 result, the mean values of each of the estimated parameters are very close to their corresponding true values. As seen in Table 5.4, the true values lie within the 95% PI associated with the distribution of estimates for each variable $Q$, $R$, $W$ and $P_{ii}$. Fig.5.5 shows the $Q$ and $R$ estimates for each MC run. As shown in Fig.5.6, the Kalman filter is considered consistent.
5.10.3 Case 3

In Case 3, we test on the example in [113]. The system matrices are assumed to be as follows.

\[
F = \begin{bmatrix}
0.75 & -1.74 & -0.3 & 0 & -0.15 \\
0.09 & 0.91 & -0.0015 & 0 & -0.008 \\
0 & 0 & 0.95 & 0 & 0 \\
0 & 0 & 0 & 0.55 & 0 \\
0 & 0 & 0 & 0 & 0.905
\end{bmatrix}
\] (5.206)
The condition number for the observability matrix is 42.6, and the condition number for the controllability matrix is 54.6. The system has a \( \text{rank}(\mathcal{F}) \) equal to 5 (utilizing}
the constraint that both $R$ and $Q$ are diagonal), with a total of 5 unknowns. Hence, the $Q$ and $R$ parameters are identifiable. The condition number of the noise covariance identifiability matrix $\mathbf{J}$ is 808. The initial Kalman filter gain is obtained by solving the Riccati equation with

$$Q^{(0)} = \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.75 \end{bmatrix} \quad (5.210)$$

$$R^{(0)} = \begin{bmatrix} 0.4 & 0 \\ 0 & 0.6 \end{bmatrix} \quad (5.211)$$

5.10.3.1 Minimum Number of Observation Samples Needed for Mehra’s and Bélanger’s Methods to Converge

Both Mehra’s [113] and Bélanger’s [14] methods to update the Kalman filter gain $W$ can be unstable unless a large number of data samples are observed. This is due to the fact that the time average converges slowly to the ensemble average. We conducted 100 MC simulations with 10,000 data samples in each run given the five-state system described in (5.206)–(5.209). We then varied the number of observed samples from 100 to 10,000 and updated the Kalman filter gain using both the Mehra [113] and Bélanger [14] methods. We measure the percentage of unstable Kalman filter gains by checking if any of the eigenvalues of $\bar{F}$ are outside of the unit circle for each run over the 100 MC runs. The results are shown in Figs.5.7 and 5.8. We only display up to 5,000 samples for both methods because each approach terminated with a stable gain when the total observation samples exceeded 5,000. The minimum number of samples required to obtain a stable gain from these methods were about 4,500. Our proposed
method always results in a stable Kalman filter gain; hence, it is not included in the comparison of methods.

5.10.3.2 Comparison of Proposed, Mehra’s, and Belanger’s Gain Update Methods

Given the 100 MC simulations with 10,000 observation samples generated in 5.10.3.1 and setting $n_L = 500$, $N_s = 10000$, Table 5.5 shows the estimation of the Kalman filter gain $W$ over 100 Monte Carlo runs, given three different gain update methods: the proposed method with $M = 40$, Mehra’s method with $M = 40$ [113] and Bélanger’s method with $M = 5$ [14].

In Table 5.5 we see that all methods have the true values staying within its 95% PI; however, our proposed method is able to obtain the Kalman filter gain closest to the optimal Kalman filter gain and the RMSE are, on average, 8 and 4 times smaller
Figure 5.8: Percentage of unstable Kalman filter gains obtained from [14] for varying the total number of observed samples ($M = 5$).

compared to Mehra’s and Bélanger’s, respectively. The very small gains $W_{21}$ and $W_{41}$ are (similarly to the small $Q$ from Case 1) very hard to estimate — they are essentially buried in noise.

We test and compare the proposed method with that of Mehra’s and Bélanger’s for the estimation of $R$, $Q$ and $P$ using the methodology described in Sections 5.6.2 and 5.7, combined with the converged Kalman filter gain from Table 5.5. The results are shown in Table 5.6 and Mehra’s method results in the true value of $P_{33}$ staying outside of the 95% PI. In comparison to Bélanger’s method, the proposed method is vastly more accurate with lower RMSE (2 to 9 times smaller) for all $R$, $Q$, and $P$, while Mehra’s method obtained a result that is less accurate than Bélanger’s method, as expected from the Kalman filter gain results. The reason $r_1$ is so difficult to estimate is that $S_1$ is dominated by the state uncertainty ($S_1 = 65$, $r_1 = 1$), i.e. the measurement
Table 5.5

| W Estimation Monte Carlo Simulation for Case 3 (100 Runs; 10,000 Samples) |
|-----------------|-----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
|                 | W_{1}           | W_{21}         | W_{31}         | W_{41}         | W_{51}         | W_{12}         | W_{22}         | W_{32}         | W_{42}         | W_{52}         |
| Truth           | 0.95 2.80 \cdot 10^{-3} | -2.86 -1.76 \cdot 10^{-4} | 0.03 0.77 0.34 | -1.49 0.25 | -0.77 |
| Proposed Method |                 |                 |                 |                 |                 |                 |                 |                 |                 |                 |
|                 | W_{11} 0.93 -8.65 \cdot 10^{-3} | -2.94 -0.02 | 0.02 0.71 0.31 | -1.57 0.18 | -0.84 |
|                 | W_{21} 2.52 \cdot 10^{-3} | -2.86 5.29 \cdot 10^{-4} | 0.03 0.77 0.34 | -1.50 0.25 | -0.76 |
|                 | W_{31} 0.97 0.01 | -2.80 0.02 | 0.05 0.84 0.38 | -1.41 0.30 | -0.68 |
|                 | W_{41} 5.33 \cdot 10^{-3} | 0.04 0.31 \cdot 10^{-3} | 9.60 \cdot 10^{-3} | 0.03 0.02 | 0.05 0.03 | 0.04 |
| Mehra’s Method  |                 |                 |                 |                 |                 |                 |                 |                 |                 |                 |
|                 | W_{11} 0.92 -0.04 | -3.38 -0.07 | -0.11 0.18 | 0.04 -2.61 | -0.12 -1.17 |
|                 | W_{21} 3.79 \cdot 10^{-4} | -3.15 3.59 \cdot 10^{-3} | -0.02 0.62 | 0.31 -1.34 | 0.28 -0.62 |
|                 | W_{31} 1.08 0.06 | -2.83 0.08 | 0.06 1.11 | 0.60 0.19 | 0.80 -0.11 |
|                 | W_{41} 0.07 0.03 | 0.33 0.04 | 0.07 0.30 | 0.14 0.79 | 0.22 0.34 |
| Bélanger’s Method |                 |                 |                 |                 |                 |                 |                 |                 |                 |                 |
|                 | W_{11} 0.89 -0.02 | -3.02 -0.05 | -0.04 0.45 | 0.13 -2.27 | -0.02 -1.15 |
|                 | W_{21} 1.46 \cdot 10^{-4} | -2.85 3.85 \cdot 10^{-3} | 0.03 0.77 | 0.33 -1.44 | 0.26 -0.77 |
|                 | W_{31} 1.01 0.04 | -2.70 0.04 | 0.10 1.13 | 0.52 -0.32 | 0.56 -0.32 |
|                 | W_{41} 0.03 0.02 | 0.09 0.02 | 0.03 0.17 | 0.09 0.48 | 0.14 0.20 |

noise is “buried” in a much larger innovation. In the case of r_{2} = 1, one has S_{2} = 2.45, i.e., r_{2} is “visible” in the innovations.

5.10.3.3 Varying The Number of Samples Observed

In this section, we vary the number of samples observed, N = 500, 2500, 5000, 10000 using our six-step approach. The results are detailed in Tables 5.7 and 5.8. As expected, the accuracy increases with an increase in N. The estimation is greatly degraded for N < 5000. Fig 5.9 illustrates that the Kalman filter is consistent.
Table 5.6

<table>
<thead>
<tr>
<th>Method</th>
<th>$R$</th>
<th>$Q$</th>
<th>$\bar{P}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$r_1$</td>
<td>$r_2$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>Truth</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>Proposed Method</td>
<td>$\bar{r}$</td>
<td>0.043</td>
<td>0.918</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>1.067</td>
<td>1.008</td>
</tr>
<tr>
<td></td>
<td>$\bar{\tau}$</td>
<td>1.976</td>
<td>1.107</td>
</tr>
<tr>
<td></td>
<td>RMSE</td>
<td>0.554</td>
<td>0.052</td>
</tr>
<tr>
<td>Mehra’s Method</td>
<td>$\bar{r}$</td>
<td>0.102</td>
<td>0.676</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>1.597</td>
<td>1.024</td>
</tr>
<tr>
<td></td>
<td>$\bar{\tau}$</td>
<td>3.681</td>
<td>1.199</td>
</tr>
<tr>
<td></td>
<td>RMSE</td>
<td>1.484</td>
<td>0.212</td>
</tr>
<tr>
<td>Bélanger’s Method</td>
<td>$\bar{r}$</td>
<td>0.0270</td>
<td>0.755</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>1.171</td>
<td>1.008</td>
</tr>
<tr>
<td></td>
<td>$\bar{\tau}$</td>
<td>2.631</td>
<td>1.254</td>
</tr>
<tr>
<td></td>
<td>RMSE</td>
<td>0.789</td>
<td>0.122</td>
</tr>
</tbody>
</table>

5.10.4 Case 4

For case 4, we simulate the unobservable (but detectable) system in [132],

$$x(k) = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.2 \end{bmatrix} x(k - 1) + \begin{bmatrix} 1 \\ 2 \end{bmatrix} v(k - 1)$$  \hspace{1cm} (5.212)

$$z(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(k) + w(k)$$  \hspace{1cm} (5.213)
Table 5.7
R, Q AND P ESTIMATION WHEN VARYING THE NUMBER OF SAMPLES OBSERVED N, MONTE CARLO SIMULATION FOR CASE 3 (100 RUNS: 500–10,000 SAMPLES)

<table>
<thead>
<tr>
<th>N</th>
<th>r_1</th>
<th>r_2</th>
<th>q_1</th>
<th>q_2</th>
<th>q_3</th>
<th>P_{11}</th>
<th>P_{22}</th>
<th>P_{33}</th>
<th>P_{44}</th>
<th>P_{55}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>72.31</td>
<td>1.143</td>
<td>1213</td>
<td>0.932</td>
<td>11.74</td>
</tr>
<tr>
<td>N = 500</td>
<td>r_1</td>
<td>r_2</td>
<td>q_1</td>
<td>q_2</td>
<td>q_3</td>
<td>P_{11}</td>
<td>P_{22}</td>
<td>P_{33}</td>
<td>P_{44}</td>
<td>P_{55}</td>
</tr>
<tr>
<td>Γ</td>
<td>0.075</td>
<td>0.708</td>
<td>0.715</td>
<td>0.593</td>
<td>0.431</td>
<td>62.97</td>
<td>0.965</td>
<td>855.5</td>
<td>0.579</td>
<td>7.505</td>
</tr>
<tr>
<td>Mean</td>
<td>2.246</td>
<td>1.014</td>
<td>0.919</td>
<td>1.343</td>
<td>1.120</td>
<td>73.25</td>
<td>1.396</td>
<td>1139</td>
<td>1.263</td>
<td>13.45</td>
</tr>
<tr>
<td>τ</td>
<td>5.221</td>
<td>1.358</td>
<td>1.137</td>
<td>2.847</td>
<td>1.542</td>
<td>89.88</td>
<td>2.204</td>
<td>1336</td>
<td>2.677</td>
<td>20.37</td>
</tr>
<tr>
<td>RMSE</td>
<td>2.060</td>
<td>0.182</td>
<td>0.133</td>
<td>0.663</td>
<td>0.308</td>
<td>6.998</td>
<td>0.398</td>
<td>136.6</td>
<td>0.626</td>
<td>3.511</td>
</tr>
<tr>
<td>N = 2,500</td>
<td>r_1</td>
<td>r_2</td>
<td>q_1</td>
<td>q_2</td>
<td>q_3</td>
<td>P_{11}</td>
<td>P_{22}</td>
<td>P_{33}</td>
<td>P_{44}</td>
<td>P_{55}</td>
</tr>
<tr>
<td>Γ</td>
<td>0.041</td>
<td>0.767</td>
<td>0.864</td>
<td>0.635</td>
<td>0.719</td>
<td>64.94</td>
<td>0.941</td>
<td>1071</td>
<td>0.660</td>
<td>8.554</td>
</tr>
<tr>
<td>Mean</td>
<td>1.453</td>
<td>1.010</td>
<td>0.977</td>
<td>1.127</td>
<td>1.049</td>
<td>73.24</td>
<td>1.245</td>
<td>1195</td>
<td>1.056</td>
<td>12.49</td>
</tr>
<tr>
<td>τ</td>
<td>3.354</td>
<td>1.173</td>
<td>1.103</td>
<td>1.685</td>
<td>1.431</td>
<td>80.49</td>
<td>1.653</td>
<td>1342</td>
<td>1.638</td>
<td>16.44</td>
</tr>
<tr>
<td>RMSE</td>
<td>1.076</td>
<td>0.094</td>
<td>0.070</td>
<td>0.392</td>
<td>0.181</td>
<td>4.472</td>
<td>0.203</td>
<td>77.80</td>
<td>0.280</td>
<td>2.088</td>
</tr>
<tr>
<td>N = 5,000</td>
<td>r_1</td>
<td>r_2</td>
<td>q_1</td>
<td>q_2</td>
<td>q_3</td>
<td>P_{11}</td>
<td>P_{22}</td>
<td>P_{33}</td>
<td>P_{44}</td>
<td>P_{55}</td>
</tr>
<tr>
<td>Γ</td>
<td>0.043</td>
<td>0.885</td>
<td>0.919</td>
<td>0.578</td>
<td>0.726</td>
<td>65.55</td>
<td>0.856</td>
<td>1126</td>
<td>0.550</td>
<td>8.864</td>
</tr>
<tr>
<td>Mean</td>
<td>1.100</td>
<td>1.011</td>
<td>0.997</td>
<td>1.008</td>
<td>0.981</td>
<td>72.27</td>
<td>1.148</td>
<td>1211</td>
<td>0.942</td>
<td>11.62</td>
</tr>
<tr>
<td>τ</td>
<td>2.580</td>
<td>1.161</td>
<td>1.084</td>
<td>1.397</td>
<td>1.265</td>
<td>80.13</td>
<td>1.390</td>
<td>1329</td>
<td>1.293</td>
<td>14.85</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.757</td>
<td>0.077</td>
<td>0.045</td>
<td>0.218</td>
<td>0.140</td>
<td>3.692</td>
<td>0.136</td>
<td>52.74</td>
<td>0.197</td>
<td>1.530</td>
</tr>
<tr>
<td>N = 10,000</td>
<td>r_1</td>
<td>r_2</td>
<td>q_1</td>
<td>q_2</td>
<td>q_3</td>
<td>P_{11}</td>
<td>P_{22}</td>
<td>P_{33}</td>
<td>P_{44}</td>
<td>P_{55}</td>
</tr>
<tr>
<td>Γ</td>
<td>0.043</td>
<td>0.918</td>
<td>0.941</td>
<td>0.662</td>
<td>0.802</td>
<td>66.06</td>
<td>0.930</td>
<td>1141</td>
<td>0.633</td>
<td>9.639</td>
</tr>
<tr>
<td>Mean</td>
<td>1.067</td>
<td>1.008</td>
<td>0.998</td>
<td>1.000</td>
<td>0.994</td>
<td>72.36</td>
<td>1.146</td>
<td>1212</td>
<td>0.933</td>
<td>11.72</td>
</tr>
<tr>
<td>τ</td>
<td>1.976</td>
<td>1.107</td>
<td>1.058</td>
<td>1.261</td>
<td>1.166</td>
<td>77.97</td>
<td>1.334</td>
<td>1290</td>
<td>1.172</td>
<td>13.76</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.554</td>
<td>0.052</td>
<td>0.031</td>
<td>0.170</td>
<td>0.097</td>
<td>2.906</td>
<td>0.106</td>
<td>37.87</td>
<td>0.153</td>
<td>1.083</td>
</tr>
</tbody>
</table>

with

\[ E[v(k)v(j)'] = \delta_{kj} \]  
\[ E[w(k)w(j)'] = \delta_{kj} \]  

(5.214)  
(5.215)
Table 5.8

<table>
<thead>
<tr>
<th></th>
<th>$W_{11}$</th>
<th>$W_{21}$</th>
<th>$W_{31}$</th>
<th>$W_{41}$</th>
<th>$W_{51}$</th>
<th>$W_{12}$</th>
<th>$W_{22}$</th>
<th>$W_{32}$</th>
<th>$W_{42}$</th>
<th>$W_{52}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth</td>
<td>0.95</td>
<td>2.80·10^{-3}</td>
<td>-2.86</td>
<td>-1.76·10^{-4}</td>
<td>0.03</td>
<td>0.77</td>
<td>0.34</td>
<td>-1.49</td>
<td>0.25</td>
<td>-0.77</td>
</tr>
<tr>
<td>$N$ = 500</td>
<td>$\bar{r}$</td>
<td>-0.04</td>
<td>-2.97</td>
<td>-0.05</td>
<td>-0.03</td>
<td>0.56</td>
<td>0.27</td>
<td>-1.64</td>
<td>0.14</td>
<td>-0.95</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>0.96</td>
<td>-1.57·10^{-3}</td>
<td>-2.74</td>
<td>7.53·10^{-3}</td>
<td>0.04</td>
<td>0.79</td>
<td>0.33</td>
<td>-1.49</td>
<td>0.26</td>
</tr>
<tr>
<td></td>
<td>$\tau$</td>
<td>1.01</td>
<td>0.03</td>
<td>-2.56</td>
<td>0.08</td>
<td>0.09</td>
<td>1.00</td>
<td>0.41</td>
<td>-1.27</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>RMSE</td>
<td>0.03</td>
<td>0.02</td>
<td>0.17</td>
<td>0.03</td>
<td>0.03</td>
<td>0.11</td>
<td>0.03</td>
<td>0.11</td>
<td>0.08</td>
</tr>
<tr>
<td>$N$ = 2,500</td>
<td>$\bar{r}$</td>
<td>-0.02</td>
<td>-2.97</td>
<td>-0.03</td>
<td>-6.51·10^{-3}</td>
<td>0.64</td>
<td>0.28</td>
<td>-1.64</td>
<td>0.14</td>
<td>-0.90</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>0.96</td>
<td>8.95·10^{-4}</td>
<td>-2.82</td>
<td>2.17·10^{-3}</td>
<td>0.03</td>
<td>0.77</td>
<td>0.33</td>
<td>-1.50</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>$\tau$</td>
<td>1.01</td>
<td>0.02</td>
<td>-2.62</td>
<td>0.05</td>
<td>0.07</td>
<td>0.95</td>
<td>0.40</td>
<td>-1.31</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>RMSE</td>
<td>0.02</td>
<td>0.01</td>
<td>0.10</td>
<td>0.02</td>
<td>0.02</td>
<td>0.08</td>
<td>0.03</td>
<td>0.09</td>
<td>0.05</td>
</tr>
<tr>
<td>$N$ = 5,000</td>
<td>$\bar{r}$</td>
<td>-0.02</td>
<td>-2.98</td>
<td>-0.03</td>
<td>8.68·10^{-3}</td>
<td>0.67</td>
<td>0.30</td>
<td>-1.63</td>
<td>0.15</td>
<td>-0.89</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>0.95</td>
<td>2.02·10^{-3}</td>
<td>-2.85</td>
<td>1.15·10^{-3}</td>
<td>0.03</td>
<td>0.77</td>
<td>0.34</td>
<td>-1.49</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>$\tau$</td>
<td>0.98</td>
<td>0.01</td>
<td>-2.77</td>
<td>0.03</td>
<td>0.06</td>
<td>0.90</td>
<td>0.39</td>
<td>-1.32</td>
<td>0.32</td>
</tr>
<tr>
<td></td>
<td>RMSE</td>
<td>0.02</td>
<td>7.08·10^{-3}</td>
<td>0.06</td>
<td>0.01</td>
<td>0.01</td>
<td>0.06</td>
<td>0.02</td>
<td>0.08</td>
<td>0.05</td>
</tr>
<tr>
<td>$N$ = 10,000</td>
<td>$\bar{r}$</td>
<td>-8.65·10^{-3}</td>
<td>-2.94</td>
<td>-0.02</td>
<td>0.02</td>
<td>0.71</td>
<td>0.31</td>
<td>-1.57</td>
<td>0.18</td>
<td>-0.84</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>0.95</td>
<td>2.52·10^{-3}</td>
<td>-2.86</td>
<td>5.29·10^{-4}</td>
<td>0.03</td>
<td>0.77</td>
<td>0.34</td>
<td>-1.50</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>$\tau$</td>
<td>0.97</td>
<td>0.01</td>
<td>-2.80</td>
<td>0.02</td>
<td>0.05</td>
<td>0.84</td>
<td>0.38</td>
<td>-1.41</td>
<td>0.30</td>
</tr>
<tr>
<td></td>
<td>RMSE</td>
<td>0.01</td>
<td>5.33·10^{-3}</td>
<td>0.04</td>
<td>9.31·10^{-3}</td>
<td>9.60·10^{-3}</td>
<td>0.03</td>
<td>0.02</td>
<td>0.05</td>
<td>0.03</td>
</tr>
</tbody>
</table>

With the initial Kalman filter gain, the system has

$$J = \begin{bmatrix} 1.04 & 1.09 \\ -0.20 & -0.31 \\ 0 & 0.02 \end{bmatrix} \quad (5.216)$$

The rank of $J$ is 2 and we have a total of 2 unknown variables. The condition number for $J$ is 23.4. We simulated 100 MC runs with observed samples $N = 1000$.
in each run. We set $n_L = 100$, $N_s = 1000$, and $\lambda_Q = 0.1$. Table 5.9 shows the estimated parameters with the initial Kalman filter gain obtained by solving the Riccati equation with $R(0) = 0.2$, and $Q(0) = 0.4$. Note that the system is not fully observable, i.e., the condition number for the observability matrix is infinity, while that for the controllability matrix is 25.8. In Table 5.9, the true values lie within the 95% PI associated with each distribution. Fig. 5.10 shows a wide variation of $Q$ and $R$ estimates; however, the NIS in Fig. 5.11 shows that the Kalman filter is consistent.
Figure 5.10: $Q$ and $R$ estimation for Case 4.

Figure 5.11: Averaged NIS for Case 4.
Table 5.9
MO lE CARLO SIMULATION FOR CASE 4 WITH $M = 100$ AND PI = 2σ (100 RUNS; 1,000 SAMPLES)

<table>
<thead>
<tr>
<th></th>
<th>$R$</th>
<th>$Q$</th>
<th>$W_1$</th>
<th>$W_2$</th>
<th>$\bar{P}_{11}$</th>
<th>$\bar{P}_{22}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth</td>
<td>1.00</td>
<td>1.00</td>
<td>0.50</td>
<td>1.01</td>
<td>1.01</td>
<td>4.08</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>3.27 $\cdot 10^{-3}$</td>
<td>0.26</td>
<td>0.05</td>
<td>0.32</td>
<td>0.27</td>
<td>1.09</td>
</tr>
<tr>
<td>Mean</td>
<td>1.04</td>
<td>1.02</td>
<td>0.50</td>
<td>1.00</td>
<td>1.03</td>
<td>4.16</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>2.05</td>
<td>2.09</td>
<td>1.22</td>
<td>2.00</td>
<td>2.09</td>
<td>8.37</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.60</td>
<td>0.53</td>
<td>0.32</td>
<td>0.52</td>
<td>0.53</td>
<td>2.11</td>
</tr>
</tbody>
</table>

5.10.5 Case 5

In Case 5, we simulate the system from [132],

$$x(k) = \begin{bmatrix} 0.1 & 0 & 0.1 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.3 \end{bmatrix} x(k-1) + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} v(k-1) \quad (5.217)$$

$$z(k) = \begin{bmatrix} 0.1 & 0.2 & 0 \end{bmatrix} x(k) + w(k) \quad (5.218)$$

with

$$E[v(k)v(j)'] = 0.5\delta_{kj} \quad (5.219)$$

$$E[w(k)w(j)'] = 0.1\delta_{kj} \quad (5.220)$$

The condition number for observability and controllability matrices are 362 and 561, respectively; hence it is an ill-conditioned case. With the initial Kalman filter gain,
the noise covariance identifiability matrix $\mathbf{J}$ is

$$
\mathbf{J} = \begin{bmatrix}
0.28 & 1.37 \\
-0.09 & -0.67 \\
0.006 & 0.11 \\
0 & -0.006
\end{bmatrix}
$$

(5.221)

The rank of $\mathbf{J}$ is 2 and we have a total of 2 unknown variables indicating that both $Q$ and $R$ are identifiable (albeit due to the high condition number, not very well relative to the other systems tested). The condition number for $\mathbf{J}$ is 36.4. We simulated 200 MC runs with $N = 1000$ observed samples for each run. We set $M = 15$ to be consistent with the setup in [132]. We also set the maximum number of iterations $n_L = 100$, $N_s = 1000$, and the regularization term from (5.129) is $\lambda_Q = 0.3$. Table 5.10 shows the estimated parameters with the initial Kalman filter gain obtained by solving the Riccati equation with $R^{(0)} = 0.1$, and $Q^{(0)} = 0.5$. The results are detailed in Table 5.10 where the true value stays within the 95% PI. Fig. 5.12 shows the scatter plot for the estimates of $R$ and $Q$ of each MC run. The plot is similar to the estimates in [132]. However, the upper bound on $Q$ is less than that of [132] (about 0.2), which does not provide the detailed results presented in Table 5.10. Fig. 5.13 shows that the Kalman filter is consistent.
Table 5.10

Monte Carlo Simulation for Case 5 with $M = 100$ and PI = $2\sigma$ (100 Runs; 1,000 Samples)

<table>
<thead>
<tr>
<th></th>
<th>$R$</th>
<th>$Q$</th>
<th>$W_1$</th>
<th>$W_2$</th>
<th>$W_3$</th>
<th>$\bar{P}_{11}$</th>
<th>$\bar{P}_{22}$</th>
<th>$\bar{P}_{33}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth</td>
<td>0.10</td>
<td>0.50</td>
<td>1.14</td>
<td>2.24</td>
<td>3.39</td>
<td>0.54</td>
<td>2.04</td>
<td>4.69</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>0.07</td>
<td>0.26</td>
<td>0.54</td>
<td>1.05</td>
<td>1.60</td>
<td>0.29</td>
<td>1.07</td>
<td>2.50</td>
</tr>
<tr>
<td>Mean</td>
<td>0.11</td>
<td>0.49</td>
<td>1.05</td>
<td>2.06</td>
<td>3.12</td>
<td>0.52</td>
<td>1.99</td>
<td>4.58</td>
</tr>
<tr>
<td>$\bar{r}$</td>
<td>0.17</td>
<td>0.65</td>
<td>1.40</td>
<td>2.76</td>
<td>4.13</td>
<td>0.69</td>
<td>2.65</td>
<td>6.09</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.03</td>
<td>0.11</td>
<td>0.27</td>
<td>0.54</td>
<td>0.80</td>
<td>0.11</td>
<td>0.45</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Figure 5.12: $Q$ and $R$ estimation for Case 5.
Figure 5.13: Averaged NIS for Case 5.
Chapter 6

Conclusion

In Chapter 2, we formulated a 3-D assignment problem and developed an efficient method to solve the problem, including 1) a rigorous mathematical formulation that is applicable to multiple domains; 2) a novel two-phase solution approach to obtain a large number of ranked solutions. The first phase of our solution approach involves partitioning the original problem space into a series of subproblems via Murty’s $m$-best decomposition procedure, while in the second phase, we solve each of these subproblems using a combination of relaxed 2-D assignments through reformulation into either a transportation problem. The solution converges with a sufficiently small duality gap. We compared the simulation runtime of the $m$-best 3-D assignment algorithm when relaxing either the assignment constraints or the transportation constraints. We also compared the performance of different combinations of 2-D assignment algorithms with a given transportation algorithm and found the combination of JVC and RELAX-IV algorithms, while relaxing the transportation constraints, to be the best performing combination when one is interested in solving the $m$-best 3-D assignment problem.
for a large number of solutions (in Chapter 2, we were interested in obtaining $10^4$ ranked solutions). We also evaluated different decomposition methods and compared their scalability and consistency with Murty’s search space decomposition. From our analysis, it can be seen that, when solving for a large number of solutions within a 3-D assignment problem, utilizing dual variable inheritance, tight upper bounds on the feasible reward, and partitioning in an optimized order offer the best performance, solving for all $m$-best solutions in a fraction of the time of the original Murty’s decomposition method, with little to no sacrifice in solution quality. These optimizations offered a maximum speedup of 10.8 over Murty’s search space decomposition. On average, it took 49.64 s to obtain $10^4$ solutions for a tensor of dimension $30\times30\times8$ required for the nuclear fuel loading problem, which was well within the 10 minute time limit placed on the algorithm.

In Chapter 3, we proposed five asset allocation algorithms to the maritime surveillance problem: 1) Exhaustive Branch-and-Cut: enumerate all possible asset-case combinations over all times and find the optimal allocation. 2) Greedy Branch-and-Cut-I: enumerate and solve for the best allocation for each asset and compute the probability of successful detection for each asset, and iteratively generate a schedule based on the highest probability of successful detection. 3) Greedy Branch-and-Cut-II: similar to Exhaustive Branch-and-Cut, except the algorithm directly enforces the asset schedule once the best allocation is found. 4) Multi-Step Lookahead Approximate Dynamic Programming-I: utilize Multi-Step Lookahead Rollout in a heuristic to iteratively schedule asset-case assignments for individual time epochs. 5) Multi-Step Lookahead Approximate Dynamic Programming-II: similar to Multi-Step Lookahead Approximate Dynamic Programming-I, except that the algorithm directly enforces the asset schedule based on the highest incremental reward. We validated each algorithm
and solved the NP-hard counter-smuggling surveillance problem in a relatively short amount of time for any of the three objectives examined – maximizing the contraband disrupted, number of detections, or the number of smugglers detected. We found that branch-and-cut-based methods are able to obtain more contraband when optimizing the amount of contraband disrupted, while the approximate dynamic approaches are better at optimizing over the number of smugglers and the number of detections. We conducted scalability and robustness analyses to evaluate the solution quality, runtimes, and contraband detection performance behavior of each algorithm. We found that the algorithms scale reasonably well with the problem size. We also found that Approximate Dynamic Programming-based approaches are able to obtain effective asset allocations within seconds of computation time with a minimal sacrifice in optimality, while proving to provide the most robust solution as measured by the $SNR$ metric. Additionally, we found the 2SLADP-II algorithm to be the best when measuring with respect to nominal-the-best $SNR$. Our future work includes further sensitivity analyses with varying asset types, aloft times, number of unavailable assets, and rest times, and spatio-temporal variations in the PoA surface (e.g., scenario-based asset allocation to handle uncertainty in PoA surfaces). Additionally, higher fidelity simulations could easily be analyzed for more accurate detection models and other operational PoA surfaces (e.g., historical flow surface and active cases). Future work also includes the incorporation of UAVs, either as in [73], where solely UAVs collaborate, or in a mixed initiative sense, an augmentation of our proposed approach.

In Chapter 4, we proposed a new approximate dynamic programming-based approach to Pareto optimization, named NAPO. The algorithm excels for the many-objective ship routing problem under complex and uncertain weather impacts. We formulated the problem as an approximate dynamic programming problem with
uncertain, nonconvex stage costs, and developed a methodology that exploits the use of A*, GMMs and silhouette scores to approximate many-objective Pareto fronts. We applied the method on a real world weather event of Tropical Storm Emily off the coast of Florida in early August of 2017, and discussed our findings. We found our algorithm performs several orders of magnitude faster than any other multiobjective algorithm due to smart search and selection of Pareto labels to expand, resulting in minimal sacrifice to Pareto optimality. With this method, up to 15 objectives are able to be optimized under a dynamic and weather-impacted uncertain environment, in 15 minutes. Future work includes adding wait time as a decision variable (i.e., not assuming the time of arrival is the time of departure at each node), and parallelization of the NAPO algorithm. Much of the approach is highly parallelizable and much time can be saved if cost calculations are performed in parallel as opposed to sequentially. Further extension to Monte Carlo tree search and Q-learning in [23] will also be investigated.

In Chapter 5 we derived necessary and sufficient conditions for the identification of the process and measurement noise covariances for a Gauss-Markov system. We also provide a novel six-step successive approximation method, coupled with an adaptive gradient method, to estimate the steady-state Kalman filter gain $W$, unknown noise covariance matrices $R$, and $Q$, as well as the state prediction (or updated) covariance matrix $\bar{P}$ (or $P$) when $Q$ and $R$ are identifiable. Moreover, we developed a novel iterative approach to obtain positive definite $Q$, $R$ and $\bar{P}$, while ensuring that the structural assumptions on $Q$ and $R$ are enforced (e.g., diagonality of $Q$ and $R$, if appropriate, symmetry and positive definiteness). We provided several approaches to estimate the unknown noise covariance $R$ via post-fit residuals. We examined previous methods from the literature and heretofore undiscussed assumptions of these methods.
that result in largely inaccurate or unstable estimates of the unknown parameters. The proposed method significantly outperformed the previous ones, given the same system assumptions. We validated the proposed method on five different test cases and were able to obtain parameter estimates where the truth stays within the 95% probability interval based on the estimates. In the future, we plan to pursue a number of research avenues, including 1) estimating $Q$ and $R$ using one-step lag smoothed residuals; 2) exploring vector moving average estimation algorithms using the minimal polynomial approach and/or truncating the effects of state; 3) replacing the batch innovation covariance estimates by their individual or mini-batch estimates, as is done in machine learning, to enable real-time estimation; 4) investigating accelerated gradient methods (e.g., Adam \cite{89}, AdaGrad \cite{51}, RMSProp \cite{169}, conjugate gradient, memoryless quasi-Newton, and trust region methods \cite{19}); 5) automatic model selection from a library of models; and 6) extension to nonlinear dynamic models.
Appendix A

Kalman Filter Derivations

A.1 Steady-State Updated State Covariance Riccati Equation

From (5.6) and (5.9), we can write the steady-state updated state covariance matrix as

\[
\bar{P} = P + WSW' = FPF' + \Gamma Q \Gamma'
\]  

(A.1)

Thus,

\[
P = FPF' - WSW' + \Gamma Q \Gamma'
\]  

(A.2)

which, given (5.14) and (5.76), we can rewrite as
\[ P = FPF' - PH'R^{-1}SR^{-1}HP' + \Gamma Q\Gamma' \]  
\[ = FPF' - PH'G^{-1}HP' + \Gamma Q\Gamma' \]  
\[ = FPF' - PH'(R - HPH')^{-1}HP' + \Gamma Q\Gamma' \]

**A.2 Proof of Necessary and Sufficient Condition for Identifiability of Unknown Covariances**

**Proposition 5**: The necessary and sufficient condition to estimate the unknown covariance matrices in a system is directly related to its minimal polynomial of its stable closed-loop filter matrix \( \bar{F} \) and a transformation of the innovations based on the coefficients of the minimal polynomial.

**Proof**: To prove necessity, let us assume that \( Q \) and \( R \) are uniquely estimable, but \( \mathcal{I} \) does not have a full column rank. This implies the nullspace of \( \mathcal{I} \), \( \mathcal{N}(\mathcal{I}) \), contains nonzero vectors. Consequently, the column vectors of \( \mathcal{I} \) are dependent and there is an infinite number of \( Q \) and \( R \) estimates that satisfy the linear equations, which contradicts our original assumption.

Now, to prove sufficiency, let us assume that \( \mathcal{I} \) has a full column rank. This implies that \( \mathcal{N}(\mathcal{I}) \) contains only the null vector, and thus, a unique solution exists for the \( Q \) and \( R \) concatenated into a column vector in \([5.36]\). Therefore, \( Q \) and \( R \) must be estimable.
A.3 Proof of Estimability of $R$

Without loss of generality, let us assume that $a_m \neq 0$ and the closed-loop transition matrix $\bar{F}$ is invertible. Note that $W$ should be such that $\bar{F}$ does not correspond to a deadbeat observer (which has no noise assumption) or an observer with zero eigenvalues for $\bar{F}$. Since $R$ is assumed to be positive definite, $\bar{F}$ is always invertible [84,136,176].

When the Kalman filter gain $W = 0$, it is evident that

$$G_m = a_m I_{nz} \quad \text{(A.6)}$$

Then,

$$L_m = a_m R \quad \text{(A.7)}$$

and $R$ is clearly identifiable. When $W$ is not zero, using (5.21) in (5.29), we have

$$G_m = a_m [I_{nz} + H(I_{nz} - WH)^{-1}W] \quad \text{(A.8)}$$

$$= a_m (I_{nz} - HW)^{-1} \quad \text{(A.9)}$$

Recall that $(I_{nz} - HW)$ is invertible because it relates the innovations and post-fit residuals (see (5.67)). So, we have

$$(I_{nz} - HW) L_m = a_m R \quad \text{(A.10)}$$

Thus, $R$ is estimable.
A.4 Procedure to Obtain $W$ using minimal polynomial

Let $W_s$ be the suboptimal Kalman filter gain and $\tilde{e}$ be the difference of the state predictions between the optimal and suboptimal filter, that is,

$$\tilde{e}(k+1|k) = \bar{F}_s \tilde{e}(k|k-1) + F(W - W_s)\nu(k) \quad (A.11)$$

where $\bar{F}_s$ is defined as

$$\bar{F}_s = F(I_{n_x} - W_s H) \quad (A.12)$$

We can write the suboptimal innovation $\nu_s(k)$ in terms of $\tilde{e}(k|k-1)$

$$\nu_s(k) = H\tilde{e}(k|k-1) + \nu(k) \quad (A.13)$$

Then, using the minimal polynomial of $\bar{F}_s$ from \(5.21\), $\nu_s(k-i)$ can be written as

$$\nu_s(k-i) = H \left[ \bar{F}_s^{m-i} \tilde{e}(k-m|k-m-1) + \sum_{l=i+1}^{m} \bar{F}_s^{l-i-1} F(W - W_s) \nu(k-l) \right] + \nu(k-i) \quad (A.14)$$

Let us define $\xi(k)$ as

$$\xi(k) = \sum_{i=0}^{m} a_i \nu_s(k-i) = \sum_{i=0}^{m} a_i \left\{ \left[ H \sum_{l=i+1}^{m} \bar{F}_s^{l-i-1} F(W - W_s) \nu(k-l) \right] + \nu(k-i) \right\} \quad (A.15)$$

$$= \sum_{l=0}^{m} \left[ a_l I_{n_x} + H \sum_{i=0}^{l-1} a_i \bar{F}_s^{l-i-1} F(W - W_s) \right] \nu(k-l) \quad (A.16)$$

$$= \sum_{l=0}^{m} V_l \nu(k-l) \quad (A.17)$$
where
\[ V_l = a_l I_{n_z} + H \sum_{i=0}^{l-1} a_i F_{s}^{l-i-1} F(W - W_s) \] (A.18)

From (5.15), we can write (A.17) in terms of z-transform, that is
\[ \xi(z) = \sum_{l=0}^{m} V_l z^{-l} \nu(z) \] (A.19)

Note that we can write \( \xi(k) \) as a vector auto-regressive process of infinite order (which can be truncated to \( M^{th} \) order), that is,
\[ \xi(k) = \sum_{j=1}^{\infty} Y_j \xi(k - j) + \nu(k) \] (A.20)

The z-transform of (A.20) is,
\[ \xi(z) = \left[ I_{n_z} - \sum_{j=1}^{\infty} Y_j z^{-j} \right]^{-1} \nu(k) \] (A.21)

Also, note the relationship between (A.21) and (A.19),
\[ \left( I_{n_z} - \sum_{j=1}^{\infty} Y_j z^{-j} \right) \sum_{l=0}^{m} V_l z^{-l} = I_{n_z} \] (A.22)

By equating coefficients, we have
\[ Y_j = V_j - \sum_{l=1}^{j-1} Y_{j-l} V_l \quad j = 0, 1, 2, \ldots, m \]
\[ = -\sum_{l=1}^{m} Y_{j-l} V_l \quad j = m + 1 \]
We can truncate the infinite vector auto-regressive model at $\mathcal{M} \gg m$, for $i = 1, 2, \ldots, \mathcal{M}$,

$$E[\xi(k)\xi(k-i)'] = E\left\{ \sum_{j=1}^{\mathcal{M}} Y_j \xi(k-j)\xi(k-i)' + \nu(k)\xi(k-i)' \right\} \quad (A.23)$$

Then, we obtain the estimates of $\{Y_i\}_{i=1}^{\mathcal{M}}$ by solving

$$\sum_{j=1}^{i} Y_j L_{i-j} + \sum_{j=i+1}^{m+i} Y_j L'_{j-i} = L_i \quad i = 1, 2, \ldots, m$$

$$\sum_{j=i-m}^{i} Y_j L_{m-j+1} + \sum_{j=i+1}^{m+i} Y_j L'_{j-i} = 0 \quad i = m+1, m+2, \ldots, \mathcal{M}$$

Let $\hat{\nu}(k)$ be

$$\hat{\nu}(k) = \xi(k) - \sum_{j=1}^{\mathcal{M}} Y_j \xi(k-j) \quad (A.24)$$

and recall (A.18) and note that

$$\hat{C}_l = H \sum_{i=0}^{l-1} a_i \hat{F}_s^{l-i-1} F \quad (A.25)$$

where $\{\hat{C}_l\}$ are the sample covariance matrices. Then,

$$\text{vec} \left[ \xi(k) - \sum_{l=0}^{m} a_l \hat{\nu}(k-l) + \sum_{l=1}^{m} \hat{C}_l FW_s \hat{\nu}(k-l) \right] = \left[ \sum_{l=0}^{m} \hat{\nu}(k-l)' \otimes \hat{C}_l \right] \text{vec}(W) \quad (A.26)$$

Alternately, $W$ can be computed via $\{V_l\}$. To compute $V_l$, we know that

$$V_0 = I_{n_z} \quad (A.27)$$

$$V_l = \sum_{i=0}^{l-1} V_i Y_{l-i} \quad l = 1, 2, \ldots, m \quad (A.28)$$

199
Recalling (A.18), we have the following relationship,

\[ \tilde{V}_l = V_l - a_l I_{n_z} + \hat{C}_l W_s = \hat{C}_l W \quad l = 1, 2, \ldots, m \]  

(A.29)

Then,

\[
\begin{align*}
\text{vec} \begin{pmatrix}
\tilde{V}_1 \\
\tilde{V}_2 \\
\vdots \\
\tilde{V}_m
\end{pmatrix} &= \begin{bmatrix}
I_{n_z} \otimes \hat{C}_1 \\
I_{n_z} \otimes \hat{C}_2 \\
\vdots \\
I_{n_z} \otimes \hat{C}_m
\end{bmatrix} \text{vec} (W) \\
&= \hat{C}_a \text{vec} (W)
\end{align*}
\]

(A.30)

(A.31)

where the vec(·) function converts \( W \) into a column vector as in (5.35) and \( \otimes \) is the Kronecker product. We can obtain the optimal Kalman filter gain \( W \) by solving the least squares problem, where a unique solution exists if \( \hat{C}_a \) has full column rank.

A.5 Objective Function Gradient Computation

Note that \( \Theta(i) \), \( \Psi \), and \( W \hat{C}(0) \) are all functions of \( W \) in (5.55). Thus,

\[
\delta J = \frac{1}{2} \text{trace} \left\{ \sum_{i=1}^{M-1} \left[ \delta \Theta(i) \Omega + \Theta(i) \delta \Omega \right] \right\}
\]

(A.32)

200
where

\[
\Omega = [\Psi - WC(0)] \mathcal{E}^2 [\Psi' - C(0)W']
\]  
(A.33)

\[
\delta\Omega = \sum_{i=1}^{M-1} \left\{ [\delta\Psi - \delta WC(0)] \mathcal{E}^2 [\Psi' - C(0)W'] + [\Psi - WC(0)] \mathcal{E}^2 [\delta\Psi' - C(0)\delta W'] \right\}
\]  
(A.34)

and

\[
\sum_{i=1}^{M-1} \delta\Theta(i)\Omega = \sum_{i=1}^{M-1} \left\{ [F'(I_n - (W + \delta W)H)'F']^{-1} H'\mathcal{E}^2 H [F(I_n - (W + \delta W)H)]^{-1} F - F'(\bar{F}'i-1 H'\mathcal{E}^2 H\bar{F}^{i-1} F) \right\} \Omega
\]  
(A.35)

To first order, (A.35) can be approximated by

\[
\sum_{i=1}^{M-1} -F' [H'\delta W'(\bar{F}')^{i-2} + \bar{F}'H'\delta W'(\bar{F}')^{i-3} + \cdots + (\bar{F}')^{i-2} H'\delta W'] H'\mathcal{E}^2 H F^{i-1} \Omega - F'(\bar{F}')^{i-1} H'\mathcal{E}^2 H \delta WH F^{i-2} + \bar{F}\delta WH \bar{F}^{i-3} + \cdots + \bar{F}^{i-2} \delta WH \right\} \Omega
\]  
(A.36)

Then,

\[
\sum_{i=1}^{M-1} \delta\Theta(i)\Omega \approx - \sum_{i=1}^{M-1} \sum_{\ell=0}^{i-2} [F'(\bar{F}')^{\ell} H'\delta W'(\bar{F}')^{i-2-\ell} H'\mathcal{E}^2 H F^{i-1} \Omega + F'(\bar{F}')^{i-1} H'\mathcal{E}^2 H \bar{F}^{i-2-\ell} \delta W H \bar{F}^{i-2-\ell} \right] \Omega
\]  
(A.37)
So,
\[
\frac{1}{2} \text{trace} \left( \sum_{i=1}^{M-1} \delta \Theta(i) \Omega \right) = - \text{trace} \left[ \delta W' \sum_{i=1}^{M-1} \sum_{\ell=0}^{i-2} (F')^{i-2-\ell} H^\ell \tilde{F}^{i-1} F \tilde{F}' (F')^{\ell} H' \right] \tag{A.38}
\]
\[
= - \text{trace} \left[ \delta W' \sum_{i=1}^{M-1} \sum_{\ell=0}^{i-2} (F')^{i-2-\ell} H^\ell \Omega^2 C(i) \Omega^2 C(\ell + 1)' \right] \tag{A.39}
\]

For \(\sum_{i=1}^{M-1} \Theta(i) \delta \Omega\), we have
\[
\sum_{i=1}^{M-1} \Theta(i) \delta \Omega = \sum_{i=1}^{M-1} \Theta(i) \left\{ [\delta PH' - \delta WC(0)] \Omega^2 (\Psi' - C(0)W') \right. \\
- [\Psi - WC(0)] \Omega^2 \left[ H \delta P - C(0) \delta W' \right] \tag{A.40}
\]

where,
\[
\delta P = \tilde{F} P \tilde{F}' - F \delta W (\Psi' - C(0)W') F' + F (\Psi - WC(0)) \delta W' F' \tag{A.41}
\]
\[
= - \sum_{b=0}^{\infty} F^b [F \delta W (\Psi' - C(0)W') F' + F (\Psi - WC(0)) \delta W' F'] (F')^b \tag{A.42}
\]

Then,
\[
\frac{1}{2} \text{trace} \left( \sum_{i=1}^{M-1} \Theta(i) \delta \Omega \right) = \text{trace} \left\{ -\delta W' \sum_{i=1}^{M-1} \Theta(i) [\Psi - WC(0)] \Omega^2 \tilde{C}(0) \\
+ \frac{1}{2} \left[ \Theta(i)(\Psi - WC(0)) \Omega^2 H + H'\Omega^2 (\Psi' - C(0)W') \Theta(i) \right] \delta P \right\} \tag{A.43}
\]
Substituting (A.42) into (A.46), we get

\[
\frac{1}{2} \text{trace} \left( \sum_{i=1}^{M-1} \Theta(i) \delta \Omega \right) = -\text{trace} \left\{ \delta W' \sum_{i=1}^{M-1} \Theta(i)(\Psi - WC(0))\varepsilon^2 C(0) \right\} \\
- \text{trace} \left\{ [F \delta W'(\Psi' - C(0)W')]F' + F(\Psi - WC(0))\varepsilon \varepsilon' Z \right\} \tag{A.44}
\]

\[
Z = \sum_{b=0}^{\infty} (\bar{F}')(b) \left[ \frac{1}{2} \sum_{i=1}^{M-1} \left[ \Theta(i)(\Psi - WC(0))\varepsilon^2 H + H'\varepsilon^2 (\Psi' - C(0)W')\Theta(i) \right] \bar{F}^b \right] \tag{A.45}
\]

We can solve for \( Z \) via a Lyapunov equation as in (5.62). Then, by substituting \( Z \) into (A.44), we have

\[
\frac{1}{2} \text{trace} \left( \sum_{i=1}^{M-1} \Theta(i) \delta \Omega \right) = -\text{trace} \left\{ -\delta W' \sum_{i=1}^{M-1} \Theta(i)X\varepsilon^2 C(0) + F'ZF X \right\} \tag{A.46}
\]

where \( X \) can be estimated using (5.64). Then, by substituting (A.39) and (A.46) into (A.32), we get (5.61).

### A.6 Cholesky Decomposition and Eigen Decomposition

To solve for \( R \) using R3, we first perform Cholesky decomposition of \( S^{-1} \). That is,

\[
S^{-1} = \mathcal{L}\mathcal{L}' \tag{A.47}
\]

Then,

\[
\mathcal{L}'RS^{-1}\mathcal{L} = (\mathcal{L}'\mathcal{R})^2 = \mathcal{L}'G\mathcal{L} \tag{A.48}
\]
Let us perform eigen decomposition on \((A.48)\), that is
\[
\mathcal{L}'G\mathcal{L} = U\Lambda U'
\]  
(A.49)

Then, we have
\[
\mathcal{L}'R\mathcal{L} = U\Lambda^{1/2}U'
\]  
(A.50)

and \(R\) can be computed as
\[
R = (\mathcal{L}')^{-1}U\Lambda^{1/2}U'\mathcal{L}^{-1}
\]  
(A.51)

\section*{A.7 Simultaneous Diagonalization}

To solve for \(R\) using \(R3\), we first perform eigen decomposition on \(S^{-1}\). That is,
\[
S^{-1} = U_1\Lambda_1 U_1'
\]
\[
= (U_1\Lambda_1^{1/2}U_1')^2
\]  
(A.52)\hspace{1cm} (A.53)

Noting that
\[
S^{-1/2}GS^{-1/2} = (S^{-1/2}RS^{-1/2})^2
\]  
(A.54)

we perform another eigen decomposition on \(U_1\Lambda_1^{1/2}U_1'GU_1\Lambda_1^{1/2}U_1'\) to get
\[
U_1\Lambda_1^{1/2}U_1'GU_1\Lambda_1^{1/2}U_1' = (U_2\Lambda_2^{1/2}U_2')^2
\]
\[
= (U_1\Lambda_1^{1/2}U_1'RU_1\Lambda_1^{1/2}U_1')^2
\]  
(A.55)\hspace{1cm} (A.56)
and $R$ can be computed as

$$R = U_1 \Lambda_1^{-1/2} U_1' U_2 \Lambda_2^{1/2} U_2' U_1 \Lambda_1^{-1/2} U_1'$$

(A.57)
Bibliography


223


228


