THE UNIVERSITY OF
SYDNEY

Belief Space Scheduling

Andrew W. Palmer

A thesis submitted in fulfilment
of the requirements of the degree of
Doctor of Philosophy

THE UNIVERSITY OF
SYDNEY

Australian Centre for Field Robotics
School of Aerospace, Mechanical and Mechatronic Engineering
The University of Sydney

Submitted July 2015; revised December 2015

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the University or other institute of higher learning, except where due acknowledgement has been made in the text.

Andrew W. Palmer

2 December 2015

Abstract

Andrew W. Palmer
The University of Sydney

Doctor of Philosophy
July 2015

Belief Space Scheduling

This thesis develops the belief space scheduling framework for scheduling under uncertainty in Stochastic Collection and Replenishment (SCAR) scenarios. A SCAR scenario is a generic resource transportation scheduling scenario consisting of multiple agents operating in the field and either using a resource such as fuel, battery charge, water, or collecting a resource such as data or physical samples. To enable these user agents to operate persistently, one or more replenishment agents travel from a centralised replenishment point to the user agents and replenish or collect their supply of the resource. Key characteristics of this scenario are that the agents operate persistently, the fleets of user and replenishment agents are heterogeneous, and that the agents have uncertainty associated with their current and future states. These characteristics have not been considered collectively in the literature.

Belief space scheduling is proposed as a framework for scheduling the actions of the replenishment agents in SCAR scenarios. Belief space scheduling performs optimisation on probability distributions over all possible states of the system rather than on individual states. It consists of three major components—estimation of the current system state given uncertain sensor readings, prediction of the future state given a schedule of tasks, and optimisation of the assignment or schedule of the replenishment agents to maximise the performance of the system.

The state estimation problem is complicated by a number of constraints that act on the state. For example, the capacity of each agent acts as a hard constraint on the state. The knowledge provided by these constraints is used to improve the state estimate through a constrained Kalman Filter (KF) approach using Probability Density Function (PDF) truncation. A novel extension of the truncation method is then developed for soft constraints that have uncertainty described by a Gaussian distribution. Soft constraints can arise from discrete sensors, such as float switches and proximity sensors, that are used in real-world SCAR scenarios. While existing truncation methods for soft constraints use numerical integration to constrain the state estimate, the developed method uses an analytical solution for the integration. This is shown to outperform existing methods, striking a balance between the high uncertainty of a KF that ignores the constraints and the overconfidence of a KF that ignores the uncertainty of the constraints.

To predict the future state of the system, a novel analytical, continuous-time framework is proposed. This framework uses multiple Gaussian approximations to propagate the probability distributions describing the system state into the future. It is compared with a Monte Carlo (MC) framework and is shown to provide similar discrimination performance while computing, in most cases, orders of magnitude faster.

Finally, several branch and bound tree search methods are proposed for the optimisation problem. These methods focus optimisation efforts on earlier tasks within a Model Predictive Control (MPC)-like framework. They are shown to outperform existing heuristic and meta-heuristic approaches. Using the proposed prediction framework is also shown to significantly improve the performance in situations where the replenishment agents are close to being fully-utilised. In multi-replenishment agent scenarios, a branch and bound approach which avoids sending replenishment agents to the same task at similar times is shown to produce excellent performance while avoiding a significant proportion of the computational burden associated with these scenarios.

Acknowledgements

I would like to start by thanking my supervisors, Andrew Hill and Steve Scheding, for their support and guidance over the past three and a half years. Our weekly discussions were crucial for keeping me on track. I would also like to thank Tom Allen for initially taking me on as a student, and Ben Upcroft for encouraging me to move to Sydney to do my PhD with the Australian Centre for Field Robotics (ACFR). My thanks go to the Rio Tinto Centre for Mine Automation, ACFR, and the University of Sydney for providing me with the opportunity to do this PhD.

To all of my friends at ACFR, thank you for creating such a welcoming and enjoyable atmosphere. To Zac, Rishi, Victor, Lloyd, and Alex, thank you for the enlightening lunchtime discussions. Thanks to Jen Jen, Nick, Chris, and Adrian for the many games and puzzles we completed and competed in. I would like to thank Dan for introducing me to the world of rock climbing, and Bryan for turning everything into an adventure. To Suchet, Dushyant, Lachlan, Dan, Kaushi, Joe, Tim, Will, and everyone else, thank you for the fun times and great memories. To my friends back home, thank you for your support and visits.

Finally, I would like to express my sincere gratitude to my family for their invaluable advice and encouragement. To my sister, thanks for the adventures. To my parents, thank you for believing in me and helping me through this journey.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

List of Acronyms

ATC	Apparent Tardiness Cost
AUV	Autonomous Underwater Vehicle
CDF	Cumulative Distribution Function
DBB	Deterministic Branch and Bound
EKF	Extended Kalman Filter
G	Greedy heuristic
GPS	Global Positioning System
KF	Kalman Filter
KL	Kullback-Leibler
LQR	Linear-Quadratic Regulator
MC	Monte Carlo
MDP	Markov Decision Process
Meta-RaPS	Meta-heuristic for Randomised Priority Search
MILP	Mixed-Integer Linear Programming
MINLP	Mixed-Integer Non-Linear Programming
MPC	Model Predictive Control
MS	Minimum Slack first
NC	No Combinations
PDF	Probability Density Function
PF	Particle Filter
POMDP	Partially Observable Markov Decision Process
PRM	Probabilistic RoadMap
QEKF	Quantised Extended Kalman Filter
RFID	Radio Frequency IDentification
RMSE	Root Mean Squared Error
RRT	Rapidly-exploring Random Tree
SA	Simulated Annealing
SATC	Stochastic Apparent Tardiness Cost
SBB	Stochastic Branch and Bound
SCAR	Stochastic Collection and Replenishment
TiMDP	Time-dependent Markov Decision Process

TSP	Travelling Salesman Problem
TSPN	Travelling Salesman Problem with Neighbourhoods
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UKF	Unscented Kalman Filter
WSPT	Weighted Shortest Processing Time first

Chapter 1

Introduction

The aim of this thesis is to develop a framework for scheduling under uncertainty in scenarios that involve moving a resource from one location to another. In such scenarios, one or more replenishment agents either transfer a resource such as fuel, battery charge, or water, from a centralised point to user agents that use this resource while operating out in the field, or transport the data or physical samples collected by the user agents out in the field back to the centralised point. The objective of the scheduling problem considered in this thesis is to minimise the downtime of the user agents resulting from them exhausting their supply or capacity of the resource.

The problem of scheduling under uncertainty is formulated in this thesis as belief space scheduling, where optimisation is performed on probability distributions describing the belief of the system state rather than on the individual system states themselves. Figure 1.1 shows the proposed framework of belief space scheduling, the three major aspects of which are estimation, prediction, and optimisation. The estimation component is concerned with producing a probabilistic estimate of the current state of the system using uncertain observations of the system state and the history of tasks that have been performed. This probabilistic estimate is known as a belief state, which is a probability distribution over all possible states of the system. The optimisation component uses this belief state to determine a task or schedule of tasks for each of the replenishment agents that will minimise an objective function.

The prediction component is used within the optimisation to evaluate each task or schedule of tasks that is considered. It predicts the future belief state of the system using the current belief state and the task or schedule currently under consideration. This thesis proposes and evaluates novel methods for these components.

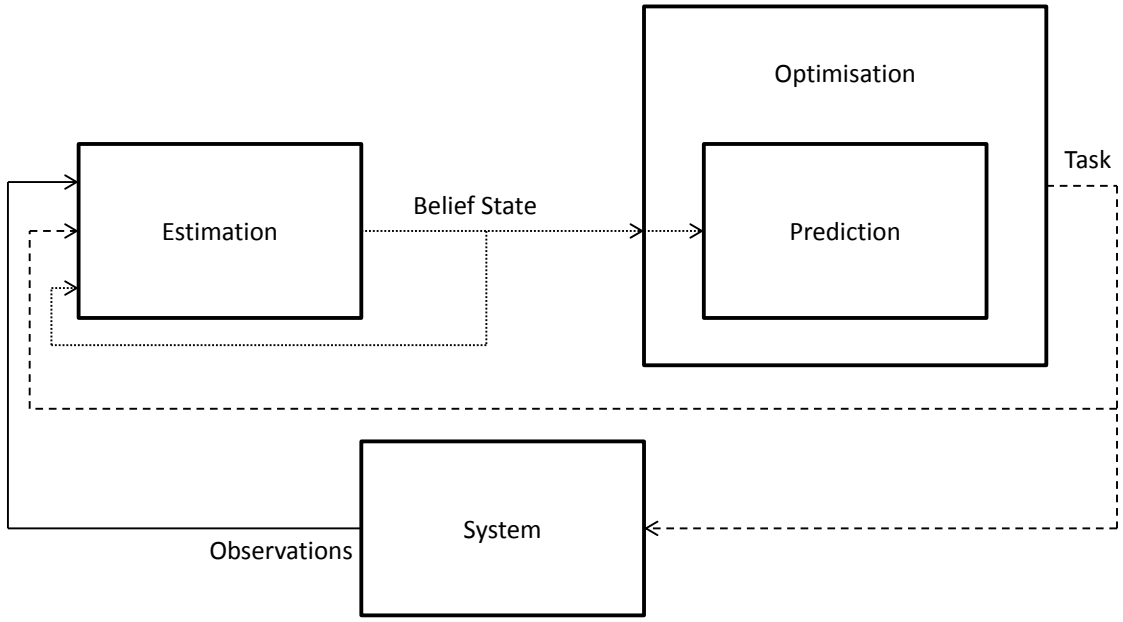


Figure 1.1 – Components of belief space scheduling problems

1.1 Motivation

Scheduling problems involving the transport of resources to agents in the field are found in many domains including robotics, defence, agriculture, and mining. Examples in the literature include recharging of robots [53], data collection from wireless sensor networks [116], aerial refuelling [54], satellite refuelling [89], autonomous harvesting [48, 110], and haul truck operations [14]. Many of these real-world scheduling problems have elements of uncertainty that have typically been ignored when optimising the schedule.

This thesis formulates these problems as a new, generic scheduling scenario called Stochastic Collection and Replenishment (SCAR). In a SCAR scenario, one or more

(a) **(b)**

(c) **(d)**

Figure 1.2 – Example scenarios. (a) and (b) are replenishment scenarios involving the transport of fuel to agents operating in the field. In (c) and (d), fruit picking machines and crop harvesters are used to collect food in the field. Photos reproduced from: (a) Geo Drill Global Services [38] (b) Rozen [86] (c) University of Florida [107] (d) Cheng [25]

replenishment agents transport a resource (e.g. fuel) from a central replenishment point (e.g. a petrol station) to user agents operating in the field. The aim of the SCAR scenario is to facilitate the persistent operation of the user agents without requiring them to leave the field and visit the replenishment point. SCAR scenarios have a number of important characteristics. Firstly, the replenishment agents are facilitating the persistent operation of the user agents over an infinite time-horizon. This means that the replenishment agents will visit the user agents more than once, the deadline for replenishing a user agent will depend on the history of when it was previously replenished, and the replenishment agents cannot be assumed to have infinite or sufficient capacity to replenish all user agents. Secondly, the resource level of the user agents varies over time as it is consumed—the later a user agent is replenished, the longer it will take to be replenished.

The final and most important characteristic of SCAR scenarios is that it models the uncertainty inherent in real-world scenarios. Parameters such as the speed and resource usage rate of an agent will vary depending on the environmental conditions, operator, and task, and therefore cannot be predicted with certainty. For the mining and agricultural scenarios that motivate this thesis, there can also be very poor sensing which contributes to the uncertainty of the scenario. For example, resource levels may have uncertain measurements from discrete level sensors. These measurements may also be affected by environmental conditions such as the slope of the ground that the agent is on. The characteristics listed above have not been considered collectively in the literature, with most existing work either ignoring some of these characteristics, or dealing with them in a superficial way.

Uncertainty in motion planning problems has typically been dealt with using belief space planning, where the state of an agent is described by a probability distribution [78]. However, the existing approaches either make restrictive assumptions that render them unsuitable for scheduling in SCAR scenarios [81], or are computationally intractable for realistic problem sizes [44]. This thesis takes inspiration from belief space planning methods by scheduling in the belief space rather than the state space. A heavy focus of this thesis is the use of analytical and approximation meth-

ods to decrease the computational requirements of the various algorithms. These are specifically developed for scenarios where the uncertain parameters are modelled by Gaussian distributions. However, the general framework presented in this thesis is not limited to Gaussian distributions, and any probability distribution can be used provided suitable analytical and approximation methods are available.

1.2 Thesis Contributions

This thesis introduces the SCAR scenario and proposes solutions to the estimation, prediction, and optimisation problems that comprise belief space scheduling for SCAR scenarios. The principal contributions of this thesis are:

- A rigorous problem definition of the SCAR scenario.
- An analytical constrained Kalman Filter (KF) using the truncation method for uncertain constraints that are described by a Gaussian distribution. This also yields a Gaussian approximation of the conditional probability distribution of a Gaussian distributed variable that is conditioned on other Gaussian distributed variables.
- A continuous-time, analytical framework for predicting the future state of a system given a schedule and uncertainty model. This is shown to provide comparable performance to a benchmark Monte Carlo (MC) approach, with the advantage of computing orders of magnitude faster.
- Branch and bound optimisation methods for single- and multi-replenishment agent scenarios. These are compared with existing heuristic and meta-heuristic methods in a detailed computational study. The branch and bound methods are shown to consistently result in the lowest downtime of the user agents, particularly when using the analytical prediction framework.

Other contributions of this thesis include:

- A Gaussian approximation of the Inverse Gaussian distribution.
- A Gaussian approximation of the generalised rectified Gaussian distribution.

1.3 Document Structure

The remainder of the thesis is organised as follows:

Chapter 2 reviews work related to the SCAR scenario and belief space scheduling concepts. It first explores the literature on collection and replenishment scenarios, and how they are commonly reformulated as classical manufacturing scheduling scenarios. Haul truck scheduling in particular is a collection scenario that has received significant attention in the literature. Despite being a simplified form of the SCAR scenario, this literature provides useful commentary on the characteristics necessary of a practical scheduling system. The treatment of uncertainty in collection and replenishment scenarios is also examined. Finally, the area of belief space planning is explored. Belief space planning methods are typically used in robotics for motion planning under uncertainty. While these methods are not directly transferable to SCAR scheduling problems, they can provide valuable insights into dealing with uncertainty.

Chapter 3 introduces a problem definition of the SCAR scenario. It first outlines the parameters of the various agents and the environment, then provides a mathematical description of the behaviour of the agents, and, finally, defines the objectives of the optimisation problem.

Chapter 4 develops methods for estimating the current state of the system in the presence of uncertainty in the system model and observations. After first introducing some background on constrained KF methods, a novel analytical constrained KF is presented for uncertain constraints that are described by a Gaussian distribution. The constrained KF method is tested on a single user agent where it is shown to outperform the existing unconstrained and hard-constrained KF methods.

Chapter 5 introduces methods for predicting the future state of the system given uncertainty in the system model. A MC approach for predicting the future state of the system is presented as a benchmark method. Novel analytical prediction methods which use Gaussian approximations are then developed. The Gaussian approximations used include approximations to the inverse Gaussian distribution, Gaussian ratio distribution, Gaussian product distribution, and rectified Gaussian distribution. Simulations of two single-replenishment agent scenarios and two multi-replenishment agent scenarios are then used to validate the performance of the prediction methods. The analytical prediction methods are shown to produce comparable results to the benchmark MC method while computing orders of magnitude faster.

Chapter 6 presents methods of optimising the behaviour of the agents in a SCAR scenario. The optimisation methods are used within a Model Predictive Control (MPC)-like framework where a new task is planned at the completion of the previous task so that unexpected changes to the state of the system can be readily incorporated. Heuristic and meta-heuristic optimisation methods from the literature are first introduced and modified for the SCAR scenario. Following this, a number of branch and bound tree search methods are developed for both single- and multi-replenishment agent SCAR scenarios. Using the same scenarios as in Chapter 5, the branch and bound optimisation methods are shown to consistently result in the lowest downtime results across all scenarios.

Chapter 7 concludes the thesis and discusses areas of future research.

Chapter 2

Literature Review

This chapter presents a review of the literature on collection and replenishment scenarios, scheduling and dispatching, and belief space planning. Section 2.1 examines the literature on collection and replenishment scenarios, their relation to classical manufacturing scheduling, and how uncertainty is generally handled in these scenarios. Section 2.2 then looks at how uncertainty is typically dealt with in robotic planning problems and how that can be applied to the scenarios under consideration in this thesis.

2.1 Collection and Replenishment Scenarios

Collection and replenishment scenarios have received limited study in the literature—replenishment scenarios have primarily focussed on aerial refuelling of aircraft, in-orbit refuelling of satellites, and recharging of Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs), while the main collection scenarios found in the literature were data collection from wireless sensor networks and haul truck dispatching. Of these scenarios, haul truck dispatching has received the most attention. This section reviews the literature on these collection and replenishment scenarios, and provides commentary on how uncertainty has typically been incorporated in scheduling problems.

2.1.1 Refuelling and Recharging

The aerial refuelling problem is the most studied of the problems in the replenishment literature. In general, it has been framed as an NP-hard combinatorial optimisation problem that resembles classical problems such as the restricted Travelling Salesman Problem (TSP) with time windows [31], or the parallel machine manufacturing job shop problem [54].

Jin et al. [46, 47] examined the problem of refuelling UAVs that are in formation behind a single tanker aircraft. They aimed to minimise a combination of two costs—the priority weighted time of refuelling of each UAV, and a similarity metric comparing the current schedule to the previous schedule. The purpose of the similarity metric was to minimise reshuffling costs that were incurred when the schedule was recalculated in response to UAVs joining or leaving the refuelling queue. They developed a backwards recursive dynamic programming algorithm to calculate the optimal schedule. This algorithm cannot be applied to most Stochastic Collection and Replenishment (SCAR) scenarios as the backwards recursive nature means that it cannot handle deadlines that are dependent on the time at which the user agents were previously replenished.

Kaplan and Rabadi [54, 55] considered a more complex aerial refuelling scenario consisting of multiple tanker aircraft and groups of aircraft to be refuelled. They modelled the problem as an identical parallel machine scheduling problem where the tanker aircraft are the machines and the aircraft to be refuelled are the jobs. The aircraft to be refuelled have ready times, a set refuelling time, and a pair of deadlines which correspond to their fuel level and mission. Their objective function was to minimise the total weighted tardiness of the jobs, and they noted that this problem formulation is NP-hard. A task is tardy if it is completed after its deadline, with the tardiness increasing the later the task is completed. In [54] they introduced a Mixed-Integer Linear Programming (MILP) model of the problem which can be used with commercial solvers to find the optimal solution to the problem. For large problems, however, this method is too computationally expensive to be practical—it took up to 10 minutes to calculate the schedule for a scenario with just 12 jobs. They introduced the Apparent Tardiness Cost (ATC) dispatching heuristic and the Simulated Anneal-

ing (SA) meta-heuristic as alternative solution methods. SA explores neighbouring schedules by swapping tasks, and avoids getting trapped in local minima by accepting worse schedules with a certain probability. They found that SA produced better results than ATC for small problem sizes, but was outperformed by ATC in larger problems while also taking 100 times longer than ATC to compute. In [55] they used the ATC heuristic to generate an initial schedule for the SA algorithm to improve the speed and performance of SA. This was compared with another meta-heuristic, Meta-heuristic for Randomised Priority Search (Meta-RaPS), with each method performing best in different circumstances. They did not compare the combined SA and ATC method with the ATC heuristic to quantify the improvement from combining the two methods.

The work of Jin et al. [46, 47] and Kaplan and Rabadi [54, 55] has several limitations. First, these formulations either ignored the time taken for the aircraft to travel to the tanker aircraft (by assuming that they are already in formation behind the tanker), or treated it as fixed regardless of which tanker aircraft it goes to. Second, the time taken to refuel the aircraft was treated as constant, regardless of when it was actually refuelled. Third, the tanker aircraft were assumed to have sufficient capacity to fully refuel all aircraft. Finally, Kaplan and Rabadi [54, 55] assumed that the tanker aircraft were identical. For many SCAR scenarios, these assumptions are not valid.

Wiley [115] and Barnes et al. [12] presented solutions to the inter-theatre aerial refuelling problem which addresses all of the issues above. The approach, originally developed by Wiley [115] and extended by Barnes et al. [12], used a group theoretic tabu search meta-heuristic which modelled the flight time of the tanker aircraft between planes as sequence-dependent setup times. Their formulation incorporated a heterogeneous fleet of tanker aircraft and allowed for the limited capacity of the tanker aircraft, as well as the variations in refuelling times and quantities based on when the aircraft were actually refuelled. They minimised a hierarchical objective function consisting of 12 criteria ranging from mission objectives, to number of aircraft and amount of fuel used. Their solution was tailored to the inter-theatre aerial refuelling problem and outperformed the existing methods in terms of solution quality

and computation time.

A similar scenario to the inter-theatre aerial refuelling problem is the in-orbit refuelling of a constellation of satellites. Shen and Tsiotras [89], Shen [88], and Tsiotras and De Nailly [106] studied this problem and noted that the additional constraints of the scenario render known TSP solutions ineffective. They solved the optimisation problem using a heuristic based on the total sweep angle of a servicing sequence. This heuristic is specific to satellites in a circular orbit and cannot be applied to the more general SCAR scenario.

A very similar scenario to the SCAR scenario was presented by Babic [10] where the problem of scheduling refuelling trucks at an airport was examined. They used branch and bound to find the optimal schedule for the refuelling trucks. SCAR scenarios are designed to facilitate the persistent operation of the agents using the resource. Thus, it is important to consider replenishing user agents multiple times in the optimisation horizon, particularly when there are large differences in the capacities and usage rates of the user agents. All of the work above, bar that of Barnes et al. [12] and Wiley [115], only considered visiting each agent once. These approaches would likely be sufficient for scenarios where the replenishment agents are heavily under-utilised. However, in scenarios where the replenishment agents are operating continuously, visiting each agent more than once is an important consideration as it results in deadlines that depend on when the agent was last visited.

Zebrowski and Vaughan [119] examined a deterministic form of the SCAR scenario where a tanker robot travels to worker robots and recharges or refuels them, with the added difficulty that the tanker robot does not initially know where the robots are and has to first search for them. The order in which the robots were recharged was selected using either a last-in, first-out strategy, or a first-in, last-out strategy, based on the order in which the robots were discovered. More sophisticated optimisation methods were not investigated. Litus et al. [64] considered a similar scenario where the robot locations are known. They used brute force to determine the optimal recharging order, and focused on developing methods for determining rendezvous locations that minimised the total travel cost of the tanker robot given a recharging

order. They showed that the problem of finding the optimal recharging order can be reduced to a TSP. In [65], they then developed a distributed heuristic for determining the rendezvous locations. In both [64] and [65], the capacity of the tanker robot was not considered.

In the work of Mathew et al. [69, 70], UGVs were used to rendezvous with and recharge UAVs performing a persistent surveillance task. They discretised the paths of the UAVs during their charging windows and planned paths for the UGVs to minimise their total travel cost. In [69] they developed an MILP to calculate an exact solution for recharging each UAV once, and showed that the single-charging agent case could be transformed into an asymmetrical TSP. In [70] they extended this to recharge the UAVs periodically, and used a receding horizon approach to minimise the computational requirement of the planner. A critical assumption of both of these papers was that the UGVs had sufficient capacity to recharge the UAVs. Using this assumption in scenarios where the UGVs do not have sufficient capacity to recharge all of the UAVs could result in one or more of the UAVs running out of charge mid-flight.

Modelling Replenishment Scenarios as Classical Scheduling Problems

In the work of Kaplan and Rabadi [54, 55], the aerial refuelling problem was modelled as a classical manufacturing scheduling problem. Many of the characteristics of SCAR scenarios can be found in the manufacturing literature—the travel time between user agents can be treated as a sequence-dependent setup time [63], replenishment times that depend on when the user agent was last replenished can be modelled as learning or deteriorating job effects [43], and the limited capacity of a replenishment agent can be modelled as periodic maintenance on the machines [57]. Sequence-dependent setup times have received extensive study [5], as have learning and deteriorating job effects [26]. The work of Toksarı and Güner [105] was the only literature found which combined both sequence-dependent setup times and learning effects. However, they made assumptions which are not compatible with the SCAR scenario, such as a common due-date for all tasks. No literature was found which combined sequence-dependent

setup times, learning effects, and machine maintenance. Other characteristics of the SCAR scenario such as sequence-dependent deadlines were also not found in the literature.

2.1.2 Data Collection from Wireless Sensor Networks

One of the main collection scenarios found in the literature was data collection from wireless sensor networks using a data mule. This strategy can be particularly useful for underwater wireless sensor networks where wireless communications over long distances is difficult, which was motivated by long-term monitoring of coral reefs. Such a scenario was presented by Vasilescu et al. [111], where an Autonomous Underwater Vehicle (AUV) was used to travel to each sensor node and retrieve the collected data. Similar data collection scenarios were presented in [19, 32, 66, 101, 102, 104, 116, 118].

Tirta et al. [104] compared three strategies for selecting the order to visit the sensor nodes—*round robin*, where each sensor node was visited once per cycle, *rate-based*, where each sensor node was visited in proportion to the rate that it collected data, and *min movement*, which combined the rate-based strategy with a cost based on the distance travelled by the collecting agent. In two very simple examples consisting of only three sensor nodes, the *min movement* approach was consistently outperformed by the *round robin* strategy. However, neither the *round robin* or *rate-based* strategies were able to consistently outperform the other.

Dunbabin et al. [32] selected the next sensor node to visit based on distance—the closest unvisited node was chosen. As AUVs are subject to significant uncertainty when travelling underwater, this strategy aimed to minimise the positioning error accumulated by the robot, and thus maximise the probability that the target sensor node was found at the expense of total travel distance.

Yuan et al. [118] treated the problem as a Travelling Salesman Problem with Neighbourhoods (TSPN). Here, neighbourhood refers to the distance around the sensor in which the data mule can wirelessly communicate with the sensor. They presented an approximate algorithm which generated a tour from the starting location, through all

of the nodes, and back to the starting location. This approach is unlikely to produce good results in scenarios where the rate of data collection varies between sensors as the algorithm implicitly assumes that the data collection rates are identical. Tekdas and Isler [101] and Ma et al. [66] used a similar strategy, calculating a TSP tour over the nodes which was repeated periodically. Bhadauria et al. [19] included the time taken to communicate the data from the sensor node to the data mule in their formulation of the problem and solved it using a modified TSPN algorithm. In [102], they extended this to use a two-ring communication model for the data transfer rate between the data mule and sensor nodes and again solved using a TSPN formulation.

Yan and Mostofi [116] used a clustering strategy to reduce the computational requirement of the scheduling algorithm. Their algorithm calculated points within clusters of nodes to visit, and then used a TSP solver to find the path which minimised the total energy requirement of the system, incorporating both travel cost and data transmission cost.

With the exception of the work of Tirta et al. [104], the sensor nodes were treated as identical. This means that the applicability of these TSP algorithms is limited to SCAR scenarios where the user agents are identical. The capacity of the data mule was also ignored. For these applications, it is unlikely that the storage capacity of the data mule will be exceeded, but by solving the problem as a TSP or TSPN they are unable to account for this case.

2.1.3 Haul Truck Dispatching

Surface mines have numerous scenarios that revolve around the transportation of a resource—haul trucks transport ore and waste from excavators to crushers and stockpiles, fuel trucks carry fuel to the excavators and drills, and water trucks transport water to the drills. Of these scenarios, only haul truck dispatching has received attention in the literature due to the large cost associated with haulage, which can account for more than half of the operating costs of open-pit mines [4, 73].

As noted by Alarie and Gamache [4], the haul truck dispatching problem is a simpli-

fication of scheduling problems found in other industries, such as SCAR scenarios, as the trucks pick up only one item. This eliminates the routing problem as the truck is always filled to capacity. Despite this, the literature on haul truck dispatching can provide a good indication of the requirements of a dispatching algorithm for real world systems. Surveys of haul truck dispatching algorithms are presented by Munirathinam and Yingling [73] and Alarie and Gamache [4]. Alarie and Gamache [4] identified two main approaches to the haul truck dispatching problem—single stage algorithms and multi-stage algorithms. Single stage algorithms typically use a heuristic based on one or more criteria to determine which shovel the truck requiring dispatching should be assigned to. Multi-stage algorithms, on the other hand, divide the dispatching problem into a number of sub-problems. Usually these consist of a high-level stage which determines the production targets for each shovel, and a low-level stage which dispatches the trucks such that the production targets are met.

The dispatching approaches were further divided by Alarie and Gamache [4] into three strategy types—1 truck to n shovels, m trucks to 1 shovel, and m trucks to n shovels. The 1 truck to n shovels strategy is generally used in single stage algorithms. In this strategy, heuristic methods are used to minimise a criteria such as the idle time of the shovels and the waiting time of the trucks. A summary of heuristics was presented by Munirathinam and Yingling [73]. Each individual heuristic generally works best in certain types of scenarios, and producing good performance across all scenarios requires a composite dispatching rule, essentially a weighted sum of individual heuristics. The difficulty with composite dispatching rules is calculating appropriate weightings—the weightings can be fixed or variable, and generally require extensive simulation to determine appropriate values [80].

The m trucks to 1 shovel strategy considers the next m trucks requiring assignment, and each shovel individually. This approach is used in the DISPATCH haul truck optimisation software produced by Modular Mining Systems [113, 114]. DISPATCH uses a multi-stage approach—the high-level stage uses linear programming to calculate the optimal flow rates of the material, and the low-level stage considers each shovel individually, selecting the truck which minimises the idle time of the shovel.

The m trucks to n shovels strategy is a combinatorial optimisation problem that resembles the replenishment and collection problems presented above. These problems are generally formed as an assignment problem or transportation problem. The combinatorial nature of these approaches means that they can be intractable when the values of m and n are too large [4].

More recent haul truck dispatching literature has taken advantage of increases in computation power to implement more advanced algorithms. *SmartMine*, a commercial dispatching package, performs an exhaustive search over all possible allocation combinations of the next n dispatching requests [98]. By considering future dispatch requests, the package aims to reduce myopic decision making. Each combination of allocations is simulated and then evaluated using a combination of objectives—queue time at the shovels, idle times of the shovels, and productivity of the truck fleet. While it produced good results in simulation, the authors did not provide information on the value of n used and the computation time of the algorithm. Given its brute-force nature, it is unlikely to scale well to large values of n and large truck fleets.

Other recent literature has focussed on incorporating uncertainty into the optimisation. Ta et al. [99] used a chance-constrained problem formulation to incorporate uncertainty in the truck cycle time and truckload, Bastos [15] and Bastos et al. [14] modelled uncertainty in the route taken by drivers and travel time using a single-dependent agent Time-dependent Markov Decision Process (TiMDP), and Arelovich et al. [7] produced turn by turn instructions for the truck drivers and incorporated a probability that the drivers would not follow the instructions.

The work of Bastos [15], Bastos et al. [14], and Arelovich et al. [7] raises an important human factors issue—human operators can be unreliable when following directions, and inconsistent in their performance of a task over time. Autonomous haul trucks have recently been introduced to several mine sites. These trucks move material in a more predictable and consistent manner than human operated haul trucks [17], and remove the need for the dispatching algorithms to consider these human factors.

2.1.4 Uncertainty

A crucial aspect of the SCAR scenario that has been ignored by most of the above literature is uncertainty, particularly uncertainty in the time taken to travel to the agents or sensors, and uncertainty in the amount of the resource the agents have remaining. For refuelling and recharging problems involving aircraft, ignoring uncertainty can potentially lead to the loss of aircraft. Mathew et al. [70] attempted to account for these uncertainties by introducing an arbitrary safety margin into their models for the estimation of the battery life and estimation of the time taken to dock and recharge. While this approach reduced plan failures, it sacrificed system performance without quantifying the risk associated with each task. In situations where the deadlines are tight, using a large safety margin may make it impossible to find a schedule which satisfies the constraints of the problem.

Even in the manufacturing literature, stochastic problems have received limited study due to their difficulty in comparison to deterministic problems [5]. Arnaout et al. [8] and Reddy and Narendran [84] considered manufacturing problems where the time taken to complete a job was uncertain. However, in both cases they simply evaluated deterministic heuristics using the mean values of the completion times and did not incorporate any information about the uncertainty in the heuristics.

A simple method to account for uncertainty is to generate a new plan when the state of the system changes unexpectedly [36]. However, this is a reactive method which does not take into account the risk associated with choosing one task over another. Pinedo [80] noted that the more uncertainty there is in a scenario, the simpler the scheduling rules should be. The reasoning behind this is that when a random event occurs it will necessitate the recalculation of the schedule, and it is desirable to have a scheduling algorithm that can react quickly to the unexpected change. Again, this does not take into account any risk and could potentially be sacrificing robustness for reactivity.

Some proactive methods which attempt to minimise risk include conservative estimates of the uncertain parameters [18, 70], estimation of an expected cost through

Monte Carlo (MC) simulation [13, 83], and chance constraints [34, 75]. Conservative estimates of the uncertain parameters can significantly overestimate the risk, and MC simulation is computationally expensive. Chance constraints provide a principled way of minimising an objective function while satisfying constraints with a given probability, but it requires calculation of certain probability distributions to determine the probability of success. For some problems, the only way to calculate these probability distributions may be through MC analysis, meaning it can inherit the computational issues of MC.

Hiatt et al. [42] took a different approach—they started by generating a schedule using a deterministic planner and then applied uncertainty analysis to make the schedule more robust. In the uncertainty analysis they used the expected reward as the cost of the schedule, where the expected reward is the reward weighted by its probability. However, this approach adds significant computation time on top of the time required to generate the initial schedule.

2.2 Belief Space Planning

Unlike scheduling under uncertainty, motion planning under uncertainty has received significant attention, particularly within the robotics community. While some planners quantify risk based on a probabilistic cost map [74], the majority of algorithms for planning under uncertainty deal with uncertainty arising from noisy sensors and stochastic dynamics. In such scenarios, the true state of the robot is not directly observable and must be inferred from the noisy observations. This leads to a probability distribution over the possible states of the robot, known as a belief state [24]. The general formulation of the belief space planning problem is known as a Partially Observable Markov Decision Process (POMDP).

A solution to a POMDP has two general components, shown in Figure 2.1—a state estimator, and a controller. The state estimator maintains a distribution over the possible states of the robot using the history of actions and observations of the robot.

The controller uses the state estimate and some form of policy or search method to determine what action the robot should take next.

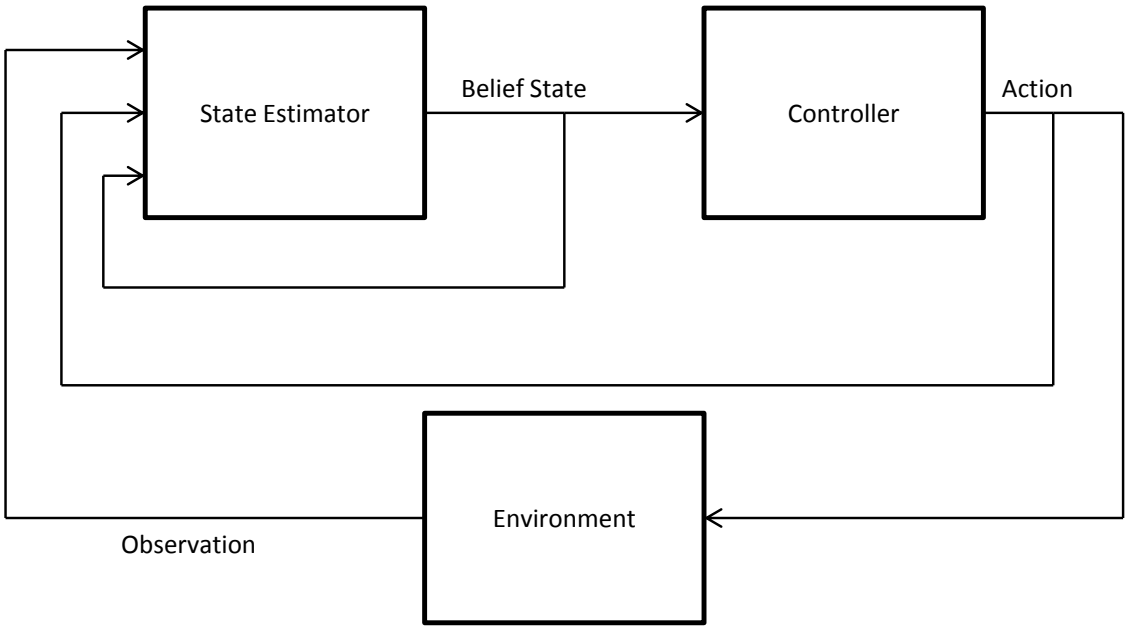


Figure 2.1 – POMDP estimation and control architecture

Early work on POMDPs considered only discrete states and actions. In these cases, the state estimator simply used Bayes' rule to propagate the effects of observations and actions on the state estimate. The controller typically used an optimal control policy generated through dynamic programming [24].

More recent work has considered continuous state, action, and observation spaces. State estimators used in these scenarios range from analytical methods such as the Kalman Filter (KF), through to MC methods such as the Particle Filter (PF). A KF is an optimal estimator for linear, time-invariant systems with Gaussian noise [6], whereas PFs are heavily used in non-linear high-dimensional systems [103]. Extensions to the KF include the Extended Kalman Filter (EKF) which linearises around the current state estimate to enable the standard KF to be used on non-linear systems [6, 50], and the Unscented Kalman Filter (UKF) which uses deterministically sampled points to represent the state [49].

To generate the controller, most methods reformulate the POMDP as a fully observ-

able Markov Decision Process (MDP) over belief space, resulting in a fully observable probabilistic control problem. However, computation of the optimal cost function and policy is a difficult problem as the belief space is infinite and continuous [21], and POMDPs have been shown to be PSPACE-complete [76]. Papadimitriou and Tsitsiklis [76] showed that an efficient online implementation of an MDP solver is likely to be impossible to create, even if an arbitrary amount of precomputation is allowed.

Patil et al. [79] identified six broad approaches to belief space planning:

Simulation-based approaches simulate the belief state of the robot for a number of potential plans. The best plan is then chosen according to a metric such as information gain [96, 108].

Regression-based planning algorithms use logical representations to discretise the belief state in order to quickly compute a policy [51].

Policy search methods optimise the parameters of a control policy using approximate inference [27, 29].

Point-based value iteration methods sample the belief space to build up a representative sample of belief points. These sampled beliefs are represented as a belief tree, where each node represents a belief and each edge represents an action-observation pair [56]. The state estimator is used to determine the evolution of the belief along the edges, and value updates are then iteratively applied to compute an optimal control policy. The main difference between the various point-based methods is how points are sampled from the belief space—uniform sampling is generally insufficient to accurately represent the belief state. Kurniawati et al. [59] used reachability analysis to guide the sampling process towards more useful points. They then introduced the Guided Cluster Sampling algorithm which used heuristics to reduce the size of the belief space to be sampled [60]. Bai et al. [11] computed an upper and lower bound on the belief and used heuristics to select observations which minimised the gap between the upper

and lower bounds, biasing the sampling towards regions which were likely to lead to improvement in the policy.

Sampling-based approaches use probabilistic motion planning methods, such as the Rapidly-exploring Random Tree (RRT) and Probabilistic RoadMap (PRM), to explore the belief space and find an optimal plan. Prentice and Roy [82] modified the PRM into the belief roadmap which computes the reachable belief space, allowing minimum expected cost paths to be found quickly. Agha-mohammadi et al. [3] created a PRM in belief space and used a Linear-Quadratic Regulator (LQR) controller to generate the optimal policy over the roadmap. Hauser [40] generated a belief-space search tree by randomly sampling open-loop controls. They also used MC simulation to estimate the likelihood of success of the policy. Bry and Roy [22] used an RRT in conjunction with a chance constraint on the probability of collision—paths that would collide with a probability above the chance-constraint were pruned from the search tree.

Trajectory optimisation methods calculate a locally optimal trajectory or policy, sacrificing global optimality for computational speed. Examples of these methods are presented in [33, 44, 45, 79, 81, 109]. These methods generally use an LQR to compute the locally optimal control policy, and require the control inputs to be continuous.

Most of the papers dealing with continuous state spaces make the assumption that the future observations that will be acquired by the robot are the most likely ones given the simulated belief of the robot state. This is known as maximum likelihood observations, and it significantly reduces the size of the belief tree as only one observation is simulated for each action. While using this assumption gives good results [81], significant improvements can be achieved by relaxing the assumption [109]. However, this comes at the expense of computational speed, with short planning horizons required to achieve acceptable planning times [44, 45].

A number of these approaches have used a MPC scheme where the robot planned an optimal sequence of controls for a number of look-ahead time steps, performed the

first action in the sequence, and then replanned the sequence of controls [44, 45, 78]. This approach reduced the need for the model to fully represent the problem, as unexpected observations were incorporated when the controller replanned [81].

There are two key differences between the SCAR scheduling problems considered in this thesis and the belief space planning problems in the above literature. Firstly, while the state and observation spaces are both continuous, the action space in a SCAR scenario is discrete. This means that the fast trajectory optimisation methods presented cannot be used. The scenario considered by Bai et al. [11] is a similar problem with discrete actions and continuous state and observations. However, their method requires significant offline computation. The configurations of the SCAR scenarios under consideration are likely to change regularly, meaning that the scheduler must be able to react quickly to these changes. This offline computation would be required each time the scenario configuration changes, making this approach infeasible for most practical systems. For scenarios where the environment and fleet of agents are fixed (for example, automated warehouses), these methods may be applicable. However, the level of uncertainty in such scenarios is generally quite low, reducing the need for schedulers that plan in belief space.

Secondly, in the above literature the robot performs an action and makes an observation at each time step, allowing the controller to decide to take actions which can reduce uncertainty in the state estimation. However, actions or tasks in scheduling scenarios generally take many time steps to complete, meaning that this feedback control cannot be achieved without frequently switching between tasks. This is undesirable (and potentially impossible) behaviour, particularly in manned scenarios. Another consequence of the longer time scale of the tasks is that the maximum likelihood observations assumption cannot be used to reduce the size of the belief tree. Since observations are made at each time step, simulating the maximum likelihood observation at every time step will lead to an unrepresentative estimate of the future state of the system by the time a task has been completed.

2.3 Summary

This chapter presented a review of the literature on collection and replenishment scenarios, and methods for dealing with uncertainty in planning and scheduling problems. Collection and replenishment scenarios have thus far received limited study, and none of the examined literature fully represented the SCAR scenario. Much of the literature on refuelling and recharging scenarios made restrictive assumptions, and the solutions for data collection from wireless sensor networks were typically based on variants of the TSP. The other collection scenario examined, haul truck dispatching, is a simplified form of the SCAR scenario as there is no routing problem. While not directly applicable to SCAR scenarios, the literature on haul truck dispatching provided valuable insight into the necessary characteristics of practical scheduling systems. The main deficiency of the examined collection and replenishment literature was that they ignored the uncertainty inherent to these scenarios. Finally, belief space planning was introduced as an example of how uncertainty is incorporated in other planning domains. Existing belief space planning methods are generally computationally very expensive, and most methods are simply incompatible with the discrete-action, continuous-state space that is characteristic of scheduling scenarios. Some of the strategies used, such as MPC, will be adapted in this thesis for use in SCAR scenarios. Next, Chapter 3 formally defines the SCAR scenario and introduces the optimisation objectives.

Chapter 3

Problem Definition of the SCAR Scenario

This chapter presents a detailed problem definition of the Stochastic Collection and Replenishment (SCAR) scenario. The problem definition outlines the behaviour of the system element highlighted in Figure 3.1 in response to tasks that are generated by the optimisation component. The structure of the SCAR scenario is motivated by autonomous mining and agricultural scenarios where the persistent operation of a fleet of user agents using a resource is facilitated by one or more replenishment agents which transport the resource from a replenishment point to the user agents. These scenarios typically have a large, heterogeneous fleet of user agents, a small, heterogeneous fleet of replenishment agents, and a central replenishment point. An example multi-replenishment agent SCAR scenario is shown in Figure 3.2. A key characteristic of SCAR scenarios is that the parameters of the agents are uncertain. The rest of this chapter is structured as follows: Section 3.1 outlines the parameters of the agents and environment, Section 3.2 provides a mathematical description of the behaviour of the system, and Section 3.3 outlines the definition of a schedule and the objectives of the SCAR scenario.

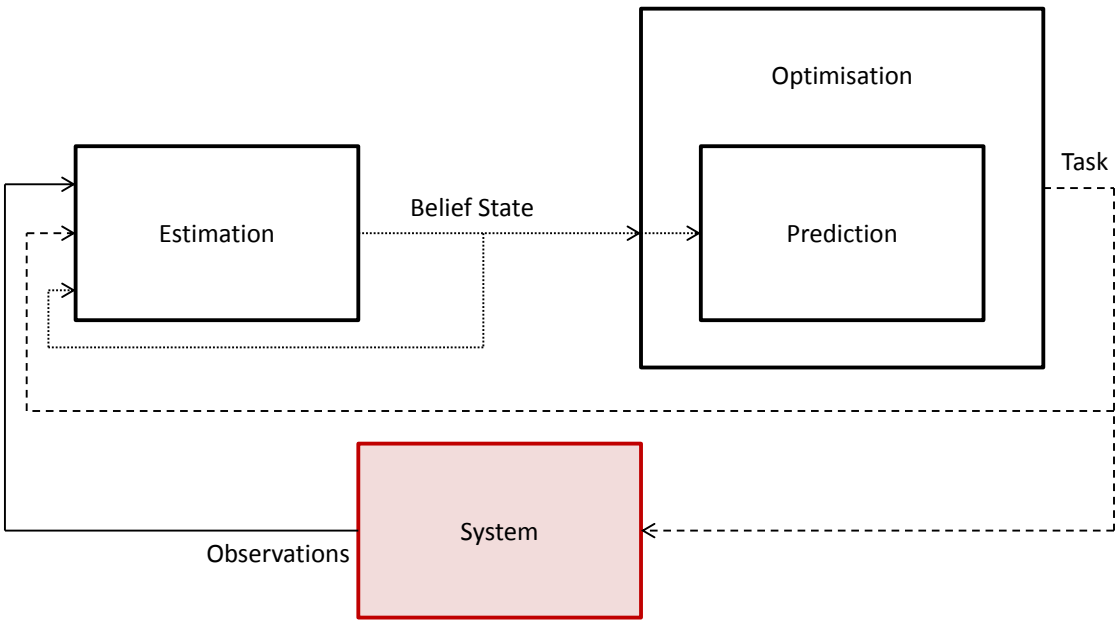


Figure 3.1 – The components of belief space scheduling. This chapter outlines the behaviour of the system (highlighted) and the desired outcomes of the belief space scheduling framework.

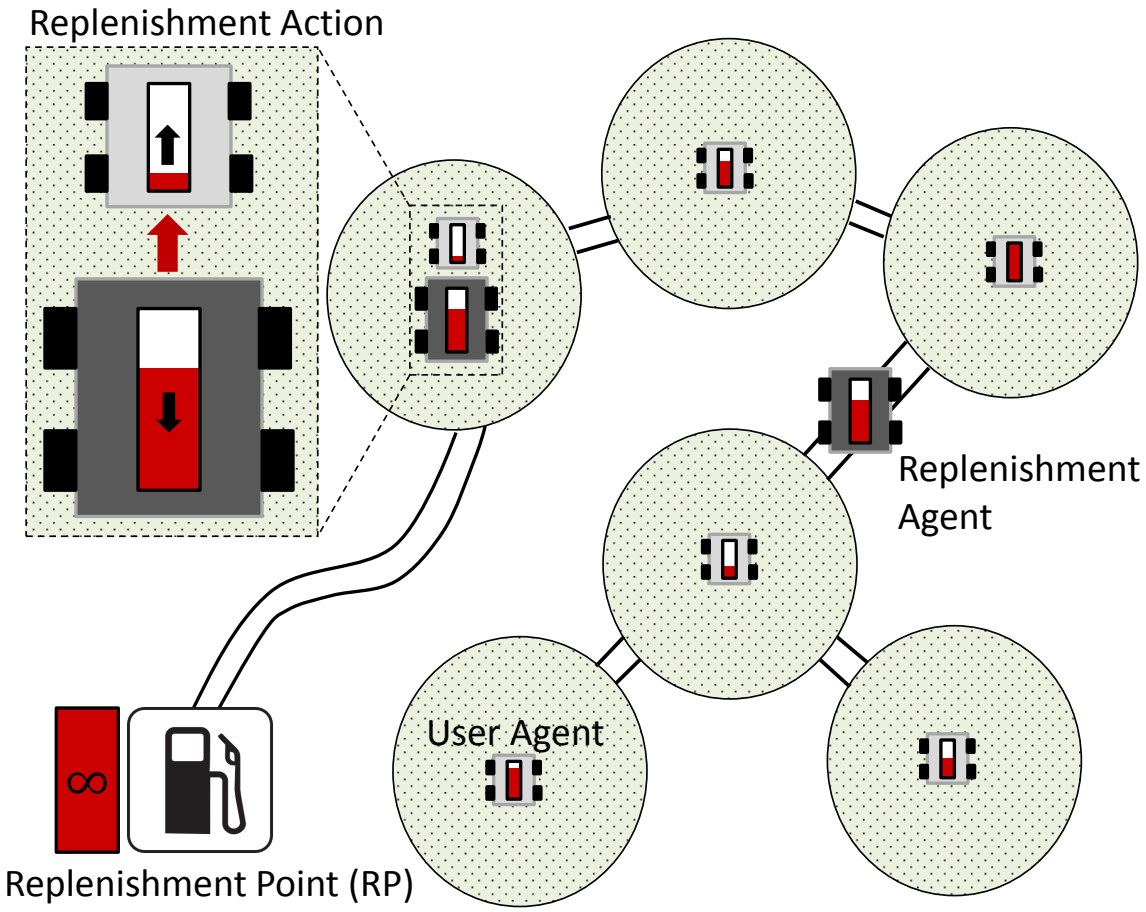


Figure 3.2 – An example SCAR scenario. Replenishment agents (dark grey) travel to, and replenish, the user agents (light grey) operating in the field. The replenishment agents can replenish their supply of the resource at the replenishment point (bottom left), which has infinite supply of the resource. In this example, the travel of the replenishment agents is restricted by roads between the operational areas of the user agents. Note also that each operational area may contain multiple user agents. Inset: The replenishment agent transfers the resource to the user agent, diminishing its supply of the resource and increasing the resource reserves of the user agent.

3.1 Parameters

This section outlines the parameters of the agents and environment in a multi-replenishment agent SCAR scenario. Note that the collection scenario can be modelled as a replenishment scenario if the resource under consideration is storage space. Uncertain parameters are denoted by an upper case letter and are assumed in this thesis to be modelled by a Gaussian probability distribution, while parameters which do not have uncertainty are denoted by a lower case letter. Note that Gaussian probability distributions have been used for convenience and other probability distributions may be used provided suitable arithmetic methods are available. Vectors are denoted by a bold lower case letter, and matrices by a bold upper case letter. All vectors and matrices do not have uncertainty associated with their elements. Units do not matter as long as they are consistent.

In an n-user agent, m-replenishment agent system, each user agent, i, has the following parameters:

- $c_{u,i}$ is the capacity of the resource
- $l_{u,i}$ is the current resource level, where $l_{u,i} \in [0, c_{u,i}]$
- $R_{u,i}$ is the resource usage rate

Each replenishment agent, j, has the following parameters:

- $c_{a,j}$ is the capacity of the resource
- $l_{a,j}$ is the current resource level, where $l_{a,j} \in [0, c_{a,j}]$
- $R_{a,j}$ is the resource replenishment rate to the user agent
- $T_{sa,j}$ and $T_{pa,j}$ are the setup and packup times that are incurred before and after the replenishment agent replenishes a user agent
- $V_{a,j}$ is the speed when travelling between tasks

The replenishment agents are assumed to have a separate supply of fuel or battery charge which is sufficient for their operations and is replenished when it visits the replenishment point.

The replenishment point has the following parameters:

- R_r is the resource replenishment rate into the replenishment agent
- T_{sr} and T_{pr} are the setup and packup times that are incurred before and after a replenishment agent is replenished

The user agents operate in areas that remain relatively fixed. For example, autonomous drills and excavators operate on a specific bench on a mine site, and autonomous tractors and harvesters operate within a specific field on a farm. The user agents are not required to remain in these fixed areas indefinitely—agents moving to new benches or fields within the scope of the schedule can be incorporated by treating the time that they arrive at their new location as the ready time of the agent. The distances between the operational areas of the user agents are generally larger than any distances that the user agents travel within their operational areas, and any variations in the travel times of the replenishment agents due to the movements of the user agents are assumed to be incorporated in the uncertain travel, setup, and packup times of the replenishment agents. Scenarios where the user agents do not operate in relatively fixed areas (for example, the inter-theatre aerial refuelling problem) are outside of the scope of this thesis.

The distances between and positions of the various agents are given by:

- $s_{au,j,i}$ is the distance between replenishment agent j and user agent i
- $s_{ar,j}$ is the distance between replenishment agent j and the replenishment point
- $x_{u,i}$ is the position of user agent i
- $x_{a,j}$ is the position of replenishment agent j
- x_r is the position of the replenishment point

Real-world SCAR scenarios may use a number of different sensor types for measuring the resource level of the agents. Of particular interest in this thesis are discrete sensors—discrete sensors can only tell whether the resource level is above or below the set-point of the sensor, an example of this is a float switch. In practice, these

sensors may have some form of uncertainty or noise in the measurements that are returned. For example, a discrete sensor may not be positioned exactly at the set-point specified, and the resource level may appear higher or lower if the agent is positioned on sloped ground. For all agents, it is assumed that the resource level is perfectly known when it is at 0% or 100%—these are reasonable assumptions as the agent ceases operating when it is at 0%, and further replenishment is impossible when it is at 100%.

For many of the real-world scenarios motivating this thesis, the positions of the replenishment agents are generally known to a high degree of accuracy through GPS or similar technology. This thesis therefore assumes perfect measurement of the position of the replenishment agents. The methods developed in this thesis can be used with uncertain position estimates by using a probability distribution describing the distance, S, instead of a scalar value, s.

The uncertain parameters in a SCAR scenario are treated differently to those in a standard Markov Decision Process (MDP). In a standard MDP, uncertain parameters are sampled at each time step. In a SCAR scenario, the uncertain parameters are instead sampled infrequently to simulate real-world behaviour. As an example, the water usage rate of a drill on a mine site will change based on the type of ground that it is drilling in—as it moves from hole to hole, the average water usage rate varies. This is the behaviour that is simulated by sampling the uncertain parameters infrequently.

3.2 Agent States, Constraints, and Behaviour

This section describes the behaviour of the agents in a SCAR scenario. A vector, $\boldsymbol{b}$, with elements $\boldsymbol{b}_j \in \{0, ..., n\}, \forall j \in \{1, ..., m\}$, stores which task each replenishment agent is currently allocated to. If $1 \leq \boldsymbol{b}_j \leq n$, then the task for replenishment agent j is to replenish user agent $\boldsymbol{b}_j$. If $\boldsymbol{b}_j = 0$, the task for replenishment agent j is to be replenished by the replenishment point. An m by $n+1$ matrix of binary variables, $\boldsymbol{E}$, captures which task each replenishment agent is currently performing. Note that this

only captures whether the replenishment agent is at the user agent or replenishment point and is currently setting up, replenishing, or packing up. If the replenishment agent is travelling or idling, it is not treated as currently performing the task. If the variable $\boldsymbol{E}_{j,0}$ is 1, then replenishment agent j is setting up, being replenished by, or is packing up at the replenishment point. Similarly, if the variables $\boldsymbol{E}_{j,1}, ..., \boldsymbol{E}_{j,n}$ are 1, then replenishment agent j is setting up, replenishing, or is packing up at user agents 1 to n respectively.

$$\boldsymbol{E}_{j,i} = \begin{cases} 1 & \text{if replenishment agent } j \text{ is setting up, replenishing, or packing} \\ & \text{up at task } i \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The matrix $\boldsymbol{E}$ is subject to the following constraints:

$$\sum_{i=0}^{n} \boldsymbol{E}_{j,i} \leq 1, \quad \forall j \in \{1, ..., m\} \tag{3.2}$$

$$\sum_{j=1}^{m} \boldsymbol{E}_{j,i} \leq 1, \quad \forall i \in \{1, ..., n\} \tag{3.3}$$

Equation 3.2 limits each replenishment agent to performing only one task at a time, while Equation 3.3 limits the number of replenishment agents servicing each user agent to one. For this thesis it is assumed that the replenishment point can service multiple replenishment agents at a time, so this is not limited in Equation 3.3.

The current action of replenishment agent j is described by a state machine with 8 states, $q_j \in \{1, ..., 8\}, \forall j \in \{1, ..., m\}$. The state transition diagram is shown in Figure 3.3, and the actions in each state are outlined below:

- $q_j = 1$: Travelling to the next task
- $q_j = 2$: Idling at the user agent

- $q_j = 3$: Setting up at the user agent
- $q_j = 4$: Replenishing the user agent
- $q_j = 5$: Packing up at the user agent
- $q_j = 6$: Setting up at the replenishment point
- $q_j = 7$: Being replenished at the replenishment point
- $q_j = 8$: Packing up at the replenishment point

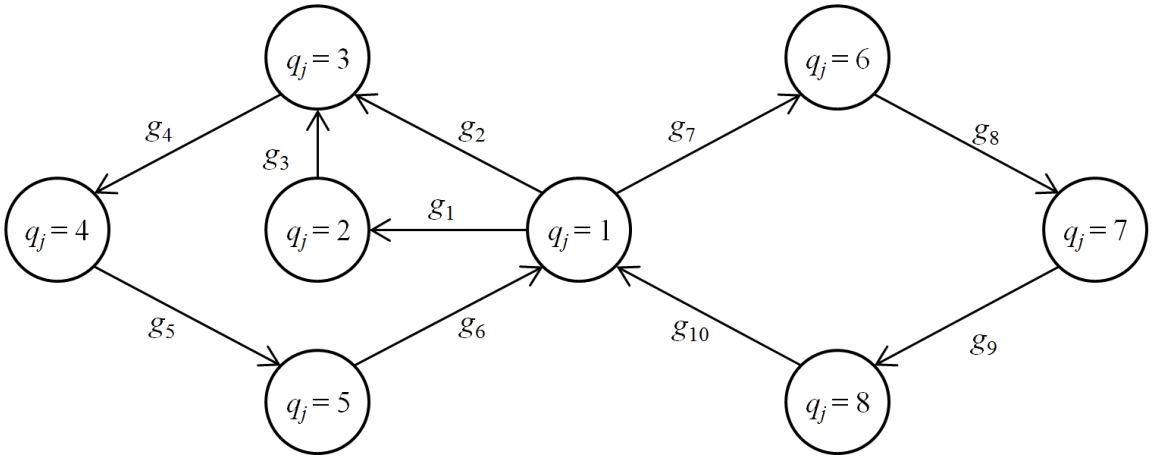


Figure 3.3 – State and transition conditions for replenishment agent j. Only the transition conditions to move to a different state are shown. Transition conditions required to stay in the same state are the complement of the conditions shown.

The state transition conditions, $g_1, ..., g_{10}$, are given below.

- g_1: $\left(x_{a,j} = x_{u,\boldsymbol{b}_j}\right) \wedge \left(\sum_{k=1,k\neq j}^{m} \boldsymbol{E}_{k,\boldsymbol{b}_j} \neq 0\right) \wedge (\boldsymbol{b}_j \neq 0)$
- g_2: $\left(x_{a,j} = x_{u,\boldsymbol{b}_j}\right) \wedge \left(\sum_{k=1,k\neq j}^{m} \boldsymbol{E}_{k,\boldsymbol{b}_j} = 0\right) \wedge (\boldsymbol{b}_j \neq 0)$
- g_3: $\left(\sum_{k=1,k\neq j}^{m} \boldsymbol{E}_{k,\boldsymbol{b}_j} = 0\right)$
- g_4: Setup is complete
- g_5: $\left(l_{u,\boldsymbol{b}_j} = c_{u,\boldsymbol{b}_j}\right) \vee (l_{a,j} = 0)$
- g_6: Packup is complete
- g_7: $(x_{a,j} = x_r) \wedge (\boldsymbol{b}_j = 0)$

- g_8: Setup is complete
- g_9: $(l_{a,j} = c_{a,j})$
- g_{10}: Packup is complete

Conditions g_1 and g_2 check that the replenishment agent is at the same position as the user agent. In addition, g_1 is satisfied when there is already another replenishment agent replenishing the user agent, while g_2 is satisfied when that user agent is not currently being replenished by a replenishment agent. Condition g_3 simply allows the replenishment agent to transfer out of the idle state when the previous replenishment agent has finished replenishing the user agent. Condition g_5 is triggered when either the user agent has been replenished to capacity, or the replenishment agent has exhausted its supply of the resource. Condition g_7 occurs when the replenishment agent is at the replenishment point, and condition g_8 is triggered when the replenishment agent has been replenished to capacity.

The rate of change of the resource supply of user agent i is given by:

$$\frac{dl_{u,i}}{dt} = \begin{cases} R_{a,j} - R_{u,i} & \text{if } q_j = 4 \text{ and } 0 < l_{u,i} < c_{u,i} \text{ and } l_{a,j} \neq 0 \text{ and} \\ & i = \boldsymbol{b}_j, \quad \forall j \in \{1, ..., m\} \\ R_{a,j} & \text{if } q_j = 4 \text{ and } l_{u,i} = 0 \text{ and } l_{a,j} \neq 0 \text{ and} \\ & i = \boldsymbol{b}_j, \quad \forall j \in \{1, ..., m\} \\ -R_{u,i} & \text{if } l_{u,i} > 0 \text{ and } i \neq \boldsymbol{b}_j, \quad \forall j \in \{1, ..., m\} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

The first and second cases occur when a replenishment agent is replenishing the user agent, while the third and fourth cases are the regular operation of the user agent when it has resources remaining and no resources remaining respectively. The user agents are assumed to be able to recover without consequence from exhausting their supply of the resource once they are replenished. This means that when they exhaust the resource they enter a safe zero-resource state, for example, a ground vehicle stopping by the side of the road.

The rate of change of the resource supply of replenishment agent j is given by:

$$\frac{dl_{a_j}}{dt} = \begin{cases} -R_{a,j} & \text{if } q_j = 4 \text{ and } l_{u,\boldsymbol{b}_j} \neq c_{u,\boldsymbol{b}_j} \text{ and } l_{a,j} \neq 0 \\ R_r & \text{if } q_j = 7 \text{ and } l_{a,j} \neq c_{a,j} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

The first condition corresponds to the replenishment agent replenishing a user agent while the second condition is when it is being replenished at the replenishment point. Outside of these conditions the replenishment agents do not consume the resource. Finally, the velocity of replenishment agent j is given by:

$$\frac{ds_j}{dt} = \begin{cases} V_{a,j} & \text{if } q_j = 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

3.3 Schedule Optimisation

This section introduces the objective of the SCAR scenario and describes the structure of a schedule. A schedule is a list of tasks in the order in which they are to be executed. In a multi-replenishment agent scenario, the schedule will consist of a list of tasks for each replenishment agent. The two types of tasks available to a replenishment agent are replenishing a user agent, and being replenished itself by the replenishment point. The task of replenishing a user agent is denoted by the index of the user agent, while the task of being replenished by the replenishment point is denoted by 0. An example schedule, $\boldsymbol{\theta}$, of a 6-user agent, 2-replenishment agent SCAR scenario is:

$$\boldsymbol{\theta} = \begin{bmatrix} [1, 4, 3, 6, 4, 1] \\ [0, 2, 6, 5, 3] \end{bmatrix} \tag{3.7}$$

It is assumed in this thesis that the replenishment agents take the shortest route between any two tasks. The routing problem could be included by treating each different route to a user agent as a separate task. This could potentially have an impact in situations where one route is shorter but has higher uncertainty than another.

Tasks may appear multiple times, or not at all, in a schedule, and one replenishment agent may have more tasks than the others. A particular task in the schedule is referenced by $\boldsymbol{\theta}_{j,k}$, where j is the index of the replenishment agent, and k is the task number. For example, $\boldsymbol{\theta}_{2,3}$ would refer to the 3rd task of replenishment agent 2, which is to perform the task of replenishing user agent 6 in the above schedule.

The objective of the SCAR scenario considered in this thesis is to minimise the weighted downtime of the user agents, λ:

$$\arg\min \lambda = \sum_{i=1}^{n} w_i t_{empty,i} \tag{3.8}$$

where the downtime of user agent i, $t_{empty,i}$, is the total time that the resource supply of user agent i is empty, and w_i is the priority weighting of user agent i.

Many other objective functions could be used in a SCAR scenario, and the objective function in Equation 3.8 is merely used to illustrate the utility of the methods developed in this thesis. Other objective functions that could be used include minimising the number of user agents that have downtime, and hierarchical objective functions that minimise first the total downtime of the user agents and then other objectives such as the total distance travelled by the replenishment agent.

3.4 Summary

This chapter presented a detailed problem definition of the SCAR scenario. The parameters of the various agents and environment were first defined. Following this, a mathematical description of the behaviour of the agents was outlined. A definition of the schedule of the replenishment agents was then presented, along with the objective of the optimisation problem. The next chapter introduces the estimation problem and develops methods for incorporating uncertain constraints on the system into the state estimate.

Chapter 4

Estimation

This chapter introduces methods for estimating the current state of the system in scheduling problems. Existing research on scheduling under uncertainty has typically neglected uncertainty in the current state of the system, with most approaches generally assuming perfect knowledge of the system state at the time of scheduling. The estimation component of belief space scheduling, highlighted in Figure 4.1, uses uncertain observations of the current system state in conjunction with the history of tasks that have been performed to produce a belief of the current state of the system. The belief state produced by the estimation component is a probability distribution over all possible states of the system. A commonly used method of estimation in belief space planning methods is the Kalman Filter (KF) [79, 109] which is an optimal estimator for linear dynamic systems with white process and measurement noise [94].

SCAR scenarios can have a number of different constraints acting on the system. The physical limits on the capacity of an agent act as hard constraints when estimating the resource level. Similarly, discrete sensors that have no uncertainty can be treated as hard constraints—at the moment the sensor changes reading, the state of the system is know to satisfy an equality with the set-point of the sensor, and while the sensor gives one reading the state is known to satisfy an inequality with the set-point. Here, the absence of a sensor reading is used to improve the state estimate. Incorporating hard constraints into the KF is a well studied area [39, 91, 93, 94]. However,

soft constraints, such as discrete sensors that have uncertainty, have typically been dealt with using numerical methods [20, 87]. The aim of this chapter is to develop novel analytical methods of incorporating soft constraints into the KF that avoid the computational burden of numerical methods.

Section 4.1 first presents background on the KF including the general KF equations and a truncation method for dealing with hard constraints. In Section 4.2, the truncation method is extended to soft constraints that have uncertainty described by Gaussian distributions. Finally, Section 4.3 presents results validating the performance of the developed truncation method for soft constraints.

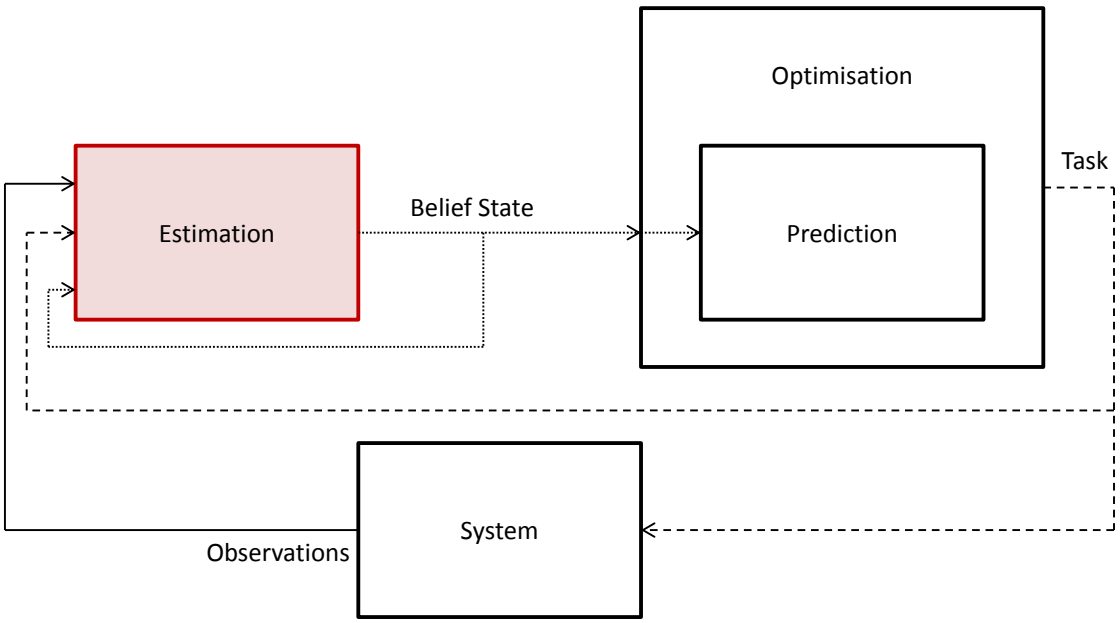


Figure 4.1 – The components of belief space scheduling. This chapter outlines methods for estimating the current state of the system.

4.1 Background

This section presents existing methods for state estimation. Section 4.1.1 provides a concise overview of the discrete KF. Section 4.1.2 then reviews the existing methods

for incorporating constraints on the state of the system and introduces the truncation method for dealing with hard constraints.

4.1.1 Kalman Filter

The KF was first derived by Kalman [52]. This section covers the basics of the discrete KF as presented in [6]. Consider a linear system given by:

$$\boldsymbol{x}_t = \boldsymbol{F}_t \boldsymbol{x}_{t-1} + \boldsymbol{B}_t \boldsymbol{u}_t + \boldsymbol{w}_t \tag{4.1}$$

where $\boldsymbol{x}_t$ is the state vector at time t, $\boldsymbol{u}_t$ is the vector of control inputs, $\boldsymbol{F}_t$ is the state transition matrix which applies the effect of each state parameter at time $t-1$ on the system at time t, $\boldsymbol{B}_t$ is the control input matrix which applies the effect of each control input parameter, and $\boldsymbol{w}_t$ is a vector containing the noise terms for the elements in the state vector. The noise is assumed to be drawn from a multivariate normal distribution with zero mean and a covariance matrix given by $\boldsymbol{Q}_t$.

Measurements are given by:

$$\boldsymbol{z}_t = \boldsymbol{H}_t \boldsymbol{x}_t + \boldsymbol{v}_t \tag{4.2}$$

where $\boldsymbol{z}_t$ is the measurement vector, $\boldsymbol{H}_t$ is the transformation matrix that maps the state vector parameters into the measurement domain, and $\boldsymbol{v}_t$ is the measurement noise vector. The measurement noise is assumed to be drawn from a multivariate normal distribution with zero mean and a covariance matrix given by $\boldsymbol{R}_t$.

The discrete KF consists of two stages—the prediction step and the measurement update. Performing these steps results in a state estimate, $\hat{\boldsymbol{x}}_t$, and associated variance, $\boldsymbol{P}_t$. The equations for the prediction stage of the KF are:

$$\hat{\boldsymbol{x}}_{t|t-1} = \boldsymbol{F}_t \hat{\boldsymbol{x}}_{t-1|t-1} + \boldsymbol{B}_t \boldsymbol{u}_t \tag{4.3}$$

$$\boldsymbol{P}_{t|t-1} = \boldsymbol{F}_t \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t^T + \boldsymbol{Q}_t \tag{4.4}$$

The measurement update equations are:

$$\hat{\boldsymbol{x}}_{t|t} = \hat{\boldsymbol{x}}_{t|t-1} + \boldsymbol{K}_t \left(\boldsymbol{z}_t - \boldsymbol{H}_t \hat{\boldsymbol{x}}_{t1|t-1} \right) \tag{4.5}$$

$$\boldsymbol{P}_{t|t} = \boldsymbol{P}_{t|t-1} - \boldsymbol{K}_t \boldsymbol{H}_t \boldsymbol{P}_{t|t-1} \tag{4.6}$$

where

$$\boldsymbol{K}_t = \boldsymbol{P}_{t|t-1} \boldsymbol{H}_t^T \left(\boldsymbol{H}_t \boldsymbol{P}_{t|t-1} \boldsymbol{H}_t^T + \boldsymbol{R}_t \right)^{-1} \tag{4.7}$$

4.1.2 Kalman Filter with Hard Constraints

Constraints arise in many state estimation problems including machine health monitoring [91], vision systems [90], robotics [20], discrete sensor networks [67], and object tracking [85]. A common example of hard constraints in the literature is obstacles in motion planning problems, which are treated as hard inequality constraints. Hard inequality constraints are well studied [91], where the main approaches are estimate projection [93], gain projection [39], and Probability Density Function (PDF) truncation [94]. Estimate and gain projection approaches incorporate the constraints into the derivation of the KF, resulting in a constrained optimisation problem that can be solved using quadratic programming, least squares approaches, and others [91, 92]. Truncation methods, on the other hand, are applied directly to the PDF resulting from a KF. They truncate the PDF at the constraints and calculate the mean and covariance of the truncated PDF, which then become the constrained state estimate and its covariance. The PDF truncation approach was shown by Simon and Simon [94] to, in general, outperform the estimate projection method. The truncation ap-

proach has been applied to probabilistic collision checking for robots [77], and has been extended to non-linear systems [97, 100].

Discrete sensors that have uncertainty on their set-point can be modelled as a soft constraint. When the sensor changes reading, the soft equality constraint can be incorporated as a noisy measurement [41, 91]. Soft inequality constraints are significantly more difficult to deal with, and numerical filters such as a Particle Filter (PF) have typically been used for these problems [87]. Several numerical methods have been examined for incorporating soft constraints into the KF. A numerical PDF truncation method was used by Boccadoro et al. [20] for robot localisation using Radio Frequency IDentification (RFID) tags, where the noise on the inequality constraints was highly non-Gaussian. Compared with a PF approach, the numerical PDF truncation method was 2 to 3 orders of magnitude faster while, in general, providing similar results. A similar RFID problem was examined by Manes and Martinelli [67], where they combined aspects of the Unscented Kalman Filter (UKF) and PF—the prediction step used the standard UKF step, while the correction step was modified to weight the sigma-points of the UKF in a similar manner to the weighting process in a PF. It was shown to outperform a PF as well as the Quantised Extended Kalman Filter (QEKF) presented by DiGiampaolo and Martinelli [30]. All of the literature examined focused on constraints with non-Gaussian distributions where the constrained state estimates are, by necessity, calculated using numerical methods. When considering soft constraints with uncertainty that is described by a Gaussian distribution, it is possible to derive analytical methods to perform the truncation to increase the speed of the algorithm. These analytical methods are developed in Section 4.2.

The remainder of this section presents the truncation method developed by Simon [91], Simon and Simon [94]. Consider the following s linearly independent constrains on the linear system defined in Equation 4.1:

$$a_m(k) \leq \boldsymbol{\phi}_m^T(k)\boldsymbol{x}(k) \leq b_m(k) \qquad m = 1, ..., s \tag{4.8}$$

A one sided constraint can be represented by setting $a_m = -\infty$, or $b_m = \infty$.

Given an estimate $\hat{\boldsymbol{x}}(k)$ with covariance $\boldsymbol{P}(k)$ at time k, the problem is to truncate the Gaussian PDF $\mathcal{N}(\hat{\boldsymbol{x}}(k), \boldsymbol{P}(k))$ using the s constraints described above, and then find the mean $\tilde{\boldsymbol{x}}(k)$ and covariance $\tilde{\boldsymbol{P}}(k)$ of the Gaussian distribution that approximates the truncated PDF. The calculated mean and covariance represent the constrained estimate of the state.

To apply the constraints via the truncation method, the state vector is first transformed so that the constraints are decoupled. This results in s transformed constraints that each involve only one transformed state, allowing the constraints to be enforced individually on each transformed state. It should be noted that the order in which constraints are applied can change the final state estimate. However, if the initial constraints are decoupled, the order of constraint application does not change the result [94].

Let the vector $\tilde{\boldsymbol{x}}_i(k)$ be the truncated state estimate, and the matrix $\tilde{\boldsymbol{P}}_i(k)$ be the covariance of $\tilde{\boldsymbol{x}}_i(k)$, after the first $i-1$ constraints have been enforced. To initialise the process:

$$\begin{aligned} i &= 1 \\ \tilde{\boldsymbol{x}}_i(k) &= \hat{\boldsymbol{x}}(k) \\ \tilde{\boldsymbol{P}}_i(k) &= \boldsymbol{P}(k) \end{aligned} \tag{4.9}$$

The transformed state vector is given by:

$$\boldsymbol{z}_i(k) = \boldsymbol{\rho}_i \boldsymbol{W}_i^{-1/2} \boldsymbol{T}_i^T (\boldsymbol{x}(k) - \tilde{\boldsymbol{x}}_i(k)) \tag{4.10}$$

where the matrices $\boldsymbol{T}_i$ and $\boldsymbol{W}_i$ are derived from the Jordan canonical decomposition of $\tilde{\boldsymbol{P}}_i(k)$:

$$\boldsymbol{T}_i \boldsymbol{W}_i \boldsymbol{T}_i^T = \tilde{\boldsymbol{P}}_i(k) \tag{4.11}$$

$\boldsymbol{T}_i$ is an orthogonal matrix, and $\boldsymbol{W}_i$ is a diagonal matrix. The matrix $\boldsymbol{\rho}_i$ is derived by the Gram-Schmidt orthogonalisation [72] which finds the orthogonal $\boldsymbol{\rho}_i$ that satisfies:

$$\boldsymbol{\rho}_i \boldsymbol{W}_i^{1/2} \boldsymbol{T}_i^T \boldsymbol{\phi}_i(k) = \begin{bmatrix} (\boldsymbol{\phi}_i^T(k) \tilde{\boldsymbol{P}}_i(k) \boldsymbol{\phi}_i(k))^{1/2} & 0 & ... & 0 \end{bmatrix}^T \tag{4.12}$$

The Gram-Schmidt orthogonalisation method is as follows:

$\boldsymbol{\rho}$ is an $n \times n$ matrix with rows $\boldsymbol{\rho}_j, \forall j \in (1, ..., n)$, where n is the length of $\tilde{\boldsymbol{x}}_i(k)$:

$$\boldsymbol{\rho} = \begin{bmatrix} \boldsymbol{\rho}_1 \\ \vdots \\ \boldsymbol{\rho}_n \end{bmatrix} \tag{4.13}$$

The first row of $\boldsymbol{\rho}$ is given by:

$$\boldsymbol{\rho}_j = \frac{\boldsymbol{\phi}_j^T(k) \boldsymbol{T} \boldsymbol{W}^{1/2}}{\left(\boldsymbol{\phi}_j^T(k) \tilde{\boldsymbol{P}}_j(k) \boldsymbol{\phi}_j(k) \right)^{1/2}} \tag{4.14}$$

Then, for $\alpha = 2, \ldots, n$ do the following:

1. The αth row of $\boldsymbol{\rho}$ is calculated by:

$$\boldsymbol{\rho}_\alpha = \boldsymbol{e}_\alpha^T - \sum_{j=1}^{\alpha-1} \left(\boldsymbol{e}_\alpha^T \cdot \boldsymbol{\rho}_j \right) \boldsymbol{\rho}_j \tag{4.15}$$

   where $\boldsymbol{e}_\alpha$ is an n-element unit vector where the αth element is 1:

$$\boldsymbol{e}_\alpha = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \alpha\text{th element} \tag{4.16}$$

2. If $\boldsymbol{\rho}_\alpha$ is zero, replace it with:

$$\boldsymbol{\rho}_\alpha = \boldsymbol{e}_1^T - \sum_{j=1}^{\alpha-1} \left(\boldsymbol{e}_1^T \cdot \boldsymbol{\rho}_j\right) \boldsymbol{\rho}_j \tag{4.17}$$

3. Normalise $\boldsymbol{\rho}_\alpha$:

$$\boldsymbol{\rho}_\alpha = \frac{\boldsymbol{\rho}_\alpha}{||\boldsymbol{\rho}_\alpha||_2} \tag{4.18}$$

After performing the above transformation, only one element of $\boldsymbol{z}_i(k)$ is constrained, and the states in the transformed state vector $\boldsymbol{z}_i(k)$ are independent standard normal distributions. The transformed states are given by:

$$c_i(k) \leq \boldsymbol{e}_i^T \boldsymbol{z}_i(k) \leq d_i(k) \tag{4.19}$$

where

$$c_i(k) = \frac{a_i(k) - \boldsymbol{\phi}_i^T(k)\tilde{\boldsymbol{x}}_i(k)}{\sqrt{\boldsymbol{\phi}_i^T(k)\tilde{\boldsymbol{P}}_i(k)\boldsymbol{\phi}_i(k)}} \tag{4.20}$$

and

$$d_i(k) = \frac{b_i(k) - \boldsymbol{\phi}_i^T(k)\tilde{\boldsymbol{x}}_i(k)}{\sqrt{\boldsymbol{\phi}_i^T(k)\tilde{\boldsymbol{P}}_i(k)\boldsymbol{\phi}_i(k)}} \tag{4.21}$$

Truncating the i-th element of the transformed state estimate results in a distribution which is the original PDF between the constraints, and 0 outside of them. An example is shown in Figure 4.2. The area of the remaining distribution is calculated as:

$$\int_{c_i(k)}^{d_i(k)} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\zeta^2}{2}\right) \mathrm{d}\zeta = \frac{1}{2}\left[\mathrm{erf}\left(\frac{d_i(k)}{\sqrt{2}}\right) - \mathrm{erf}\left(\frac{c_i(k)}{\sqrt{2}}\right)\right] \tag{4.22}$$

where erf(.) is the error function, defined as:

$$\operatorname{erf}(t) = \frac{2}{\sqrt{\pi}} \int_0^t \exp\left(-\tau^2\right) \mathrm{d}\tau \tag{4.23}$$

The truncated PDF is then given by:

$$\operatorname{PDF}(\zeta) = \begin{cases} \alpha \exp\left(-\frac{\zeta^2}{2}\right) & \zeta \in [c_i(k), d_i(k)] \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

where

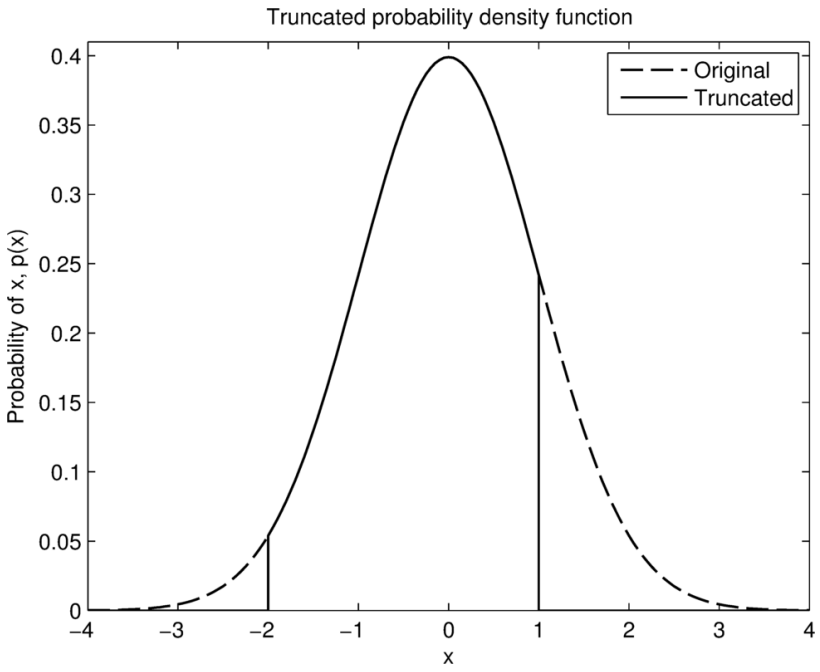
$$\alpha = \frac{\sqrt{2}}{\sqrt{\pi}\left[\operatorname{erf}\left(\frac{d_i(k)}{\sqrt{2}}\right) - \operatorname{erf}\left(\frac{c_i(k)}{\sqrt{2}}\right)\right]} \tag{4.25}$$

The mean and variance of the Gaussian distribution that approximates the truncated PDF is given by:

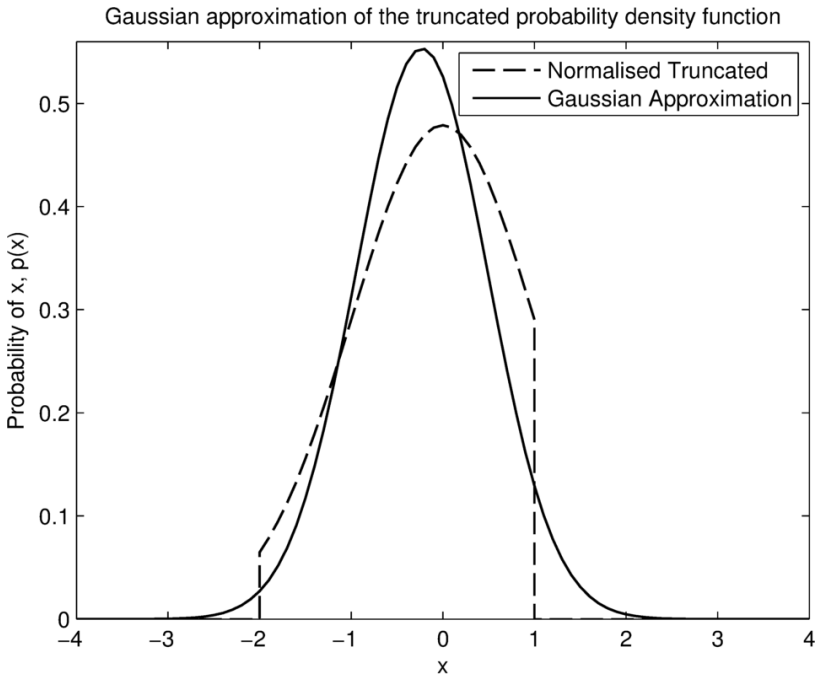
$$\begin{aligned} \mu_i &= \alpha \int_{c_i(k)}^{d_i(k)} \zeta \exp\left(-\frac{\zeta^2}{2}\right) \mathrm{d}\zeta \\ &= \alpha \left[\exp\left(\frac{-c_i^2(k)}{2}\right) - \exp\left(\frac{-d_i^2(k)}{2}\right)\right] \end{aligned} \tag{4.26}$$

$$\begin{aligned} \sigma_i^2 &= \alpha \int_{c_i(k)}^{d_i(k)} (\zeta - \mu_i)^2 \exp\left(-\frac{\zeta^2}{2}\right) \mathrm{d}\zeta \\ &= \alpha \left[\exp\left(\frac{-c_i^2(k)}{2}\right)(c_i(k) - 2\mu_i) - \exp\left(\frac{-d_i^2(k)}{2}\right)(d_i(k) - 2\mu_i)\right] + \mu_i^2 + 1 \end{aligned} \tag{4.27}$$

An example of the Gaussian approximation of the truncated PDF is shown in Figure 4.2b. The transformed state estimate, after the ith constraint has been applied, has the following mean and covariance:



(a)



(b)

Figure 4.2 – (a) Gaussian distribution that is truncated at $x = -2$ and $x = 1$. (b) Gaussian approximation of the truncated Gaussian distribution. The truncated Gaussian distribution has been normalised in this figure so that it has an area of 1.

$$\begin{aligned}\tilde{\boldsymbol{z}}_{i+1}(k) &= \mu_i \boldsymbol{e}_i \\ \tilde{\boldsymbol{G}}_{i+1}(k) &= \boldsymbol{I}_n + \left(\sigma_i^2 - 1\right) \boldsymbol{e}_i \boldsymbol{e}_i^T\end{aligned} \tag{4.28}$$

where $\boldsymbol{I}_n$ is an $n \times n$ identity matrix. Taking the inverse of the transformation in Equation 4.10 gives the mean and variance of the state estimate after the truncating with the ith constraint:

$$\begin{aligned}\tilde{\boldsymbol{x}}_{i+1}(k) &= \boldsymbol{T}_i \boldsymbol{W}_i^{1/2} \boldsymbol{\rho}_i^T \tilde{\boldsymbol{z}}_{i+1}(k) + \tilde{\boldsymbol{x}}_i(k) \\ \tilde{\boldsymbol{P}}_{i+1}(k) &= \boldsymbol{T}_i \boldsymbol{W}_i^{1/2} \boldsymbol{\rho}_i^T \tilde{\boldsymbol{G}}_{i+1}(k) \boldsymbol{\rho}_i \boldsymbol{W}_i^{1/2} \boldsymbol{T}_i^T\end{aligned} \tag{4.29}$$

This process (from Equation 4.10 to Equation 4.29) is repeated for the s constraints, incrementing i each time and using the constrained state estimate after constraint i has been applied as the input state estimate for constraint $i+1$. After the s constraints have been applied, the constrained state estimate is:

$$\begin{aligned}\tilde{\boldsymbol{x}}(k) &= \tilde{\boldsymbol{x}}_{s+1}(k) \\ \tilde{\boldsymbol{P}}(k) &= \tilde{\boldsymbol{P}}_{s+1}(k)\end{aligned} \tag{4.30}$$

4.2 Kalman Filter with Soft Constraints

This section presents a novel extension of the truncation method introduced in Section 4.1.2 to soft constraints that have uncertainty described by Gaussian distributions. Existing truncation methods for soft constraints have used numerical integration [20, 67]. Unlike these existing methods, the method developed in this section presents an analytical solution for the integration. In Section 4.2.1, the Gaussian

distributed constraints are transformed so that they are applied to a standard normal distribution. Section 4.2.2 presents an analytical method of truncating a standard normal distribution when there is one constraint, and Section 4.2.3 extends this to an interval constraint.

4.2.1 Transforming Soft Constraints

Consider the following s linearly independent constrains on the state of a system:

$$A_m(k) \leq \boldsymbol{\phi}_m^T(k)\boldsymbol{x}(k) \leq B_m(k) \qquad m = 1, ..., s \tag{4.31}$$

where the constraints are described by a Gaussian distribution:

$$\begin{aligned} A_m(k) &\sim \mathcal{N}(\mu_{a,m}, \sigma_{a,m}^2) \\ B_m(k) &\sim \mathcal{N}(\mu_{b,m}, \sigma_{b,m}^2) \end{aligned} \tag{4.32}$$

Equation 4.31 describes a two-sided constraint on the linear function of the state described by $\boldsymbol{\phi}_m^T(k)\boldsymbol{x}(k)$. One sided constraints can be represented by setting $\mu_{a,m} = -\infty$, or $\mu_{b,m} = \infty$, and hard constraints can be implemented by setting the standard deviation of the constraint $\sigma_{a,m} \approx 0$ or $\sigma_{b,m} \approx 0$ as required.

After transforming the state using the process described in Section 4.1.2, the transformed constraints are given by:

$$C_i(k) \leq \boldsymbol{e}_i^T \boldsymbol{z}_i(k) \leq D_i(k) \tag{4.33}$$

where

$$
\begin{aligned}
C_i(k) &\sim \mathcal{N}(\mu_{c,i}, \sigma^2_{c,i}) \\
\mu_{c,i}(k) &= \frac{\mu_{a,i}(k) - \boldsymbol{\phi}_i^T(k)\tilde{\boldsymbol{x}}_i(k)}{\sqrt{\boldsymbol{\phi}_i^T(k)\tilde{\boldsymbol{P}}_i(k)\boldsymbol{\phi}_i(k)}} \\
\sigma^2_{c,i}(k) &= \frac{\sigma^2_{a,i}(k)}{\boldsymbol{\phi}_i^T(k)\tilde{\boldsymbol{P}}_i(k)\boldsymbol{\phi}_i(k)}
\end{aligned} \tag{4.34}
$$

and

$$
\begin{aligned}
D_i(k) &\sim \mathcal{N}(\mu_{d,i}, \sigma^2_{d,i}) \\
\mu_{d,i}(k) &= \frac{\mu_{b,i}(k) - \boldsymbol{\phi}_i^T(k)\tilde{\boldsymbol{x}}_i(k)}{\sqrt{\boldsymbol{\phi}_i^T(k)\tilde{\boldsymbol{P}}_i(k)\boldsymbol{\phi}_i(k)}} \\
\sigma^2_{d,i}(k) &= \frac{\sigma^2_{b,i}(k)}{\boldsymbol{\phi}_i^T(k)\tilde{\boldsymbol{P}}_i(k)\boldsymbol{\phi}_i(k)}
\end{aligned} \tag{4.35}
$$

4.2.2 Applying a One-Sided Soft Constraint

First, consider the case where there is only one constraint on the transformed state:

$$
C_i(k) \leq \boldsymbol{e}_i^T \boldsymbol{z}_i(k) \tag{4.36}
$$

As each element of $\boldsymbol{z}_i(k)$ is a standard normal distribution, the resultant truncated function for the ith element of the constrained estimate, $\mathrm{Z}(\boldsymbol{e}_i^T \tilde{\boldsymbol{z}}_{i+1}(k))$, is given by:

$$
\begin{aligned}
\mathrm{Z}(\boldsymbol{e}_i^T \tilde{\boldsymbol{z}}_{i+1}(k)) &= \mathrm{PDF}(\boldsymbol{e}_i^T \boldsymbol{z}_i(k)) \times \mathrm{CDF}(C_i(k)) \\
&= \frac{1}{2\sqrt{2\pi}} \exp\left(-\zeta^2/2\right) \left[1 + \mathrm{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right]
\end{aligned} \tag{4.37}
$$

where CDF is the Cumulative Distribution Function. This function, once normalised, represents the conditional probability distribution of $\boldsymbol{e}_i^T \boldsymbol{z}_i(k)$ that satisfies $C_i(k) \leq \boldsymbol{e}_i^T \boldsymbol{z}_i(k)$. To normalise the function in Equation 4.37 to a PDF, the area is calculated as:

$$\begin{aligned}
&\int_{-\infty}^{\infty} \mathrm{Z}(\boldsymbol{e}_i^T \tilde{\boldsymbol{z}}_{i+1}(k))\mathrm{d}\zeta \\
&\quad = \int_{-\infty}^{\infty} \frac{1}{2\sqrt{2\pi}} \exp\left(-\zeta^2/2\right) \left[1 + \mathrm{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right] \mathrm{d}\zeta \\
&\quad = \frac{1}{2}\left[1 - \mathrm{erf}\left(\frac{\mu_{c,i}}{\sqrt{2\left(\sigma_{c,i}^2 + 1\right)}}\right)\right]
\end{aligned} \tag{4.38}$$

The truncated PDF is then given by:

$$\mathrm{PDF}(\zeta) = \alpha_i \exp\left(-\zeta^2/2\right) \left[1 + \mathrm{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right] \tag{4.39}$$

where

$$\alpha_i = \frac{1}{\sqrt{2\pi}\left[1 - \mathrm{erf}\left(\frac{\mu_{c,i}}{\sqrt{2\left(\sigma_{c,i}^2+1\right)}}\right)\right]} \tag{4.40}$$

The mean and variance of the ith element of $\boldsymbol{z}_i(k)$ after the truncation has been performed are calculated as follows:

$$\begin{aligned}
\mu_i &= \alpha_i \int_{-\infty}^{\infty} \zeta \exp\left(-\zeta^2/2\right) \left[1 + \mathrm{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right] \mathrm{d}\zeta \\
&= \frac{2\alpha_i}{\sqrt{\sigma_{c,i}^2 + 1}} \exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2 + 1)}\right)
\end{aligned} \tag{4.41}$$

$$\begin{aligned}\sigma_i^2 &= \alpha_i \int_{-\infty}^{\infty} (\zeta - \mu_i)^2 \exp\left(-\zeta^2/2\right) \left[1 + \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right] \mathrm{d}\zeta \\ &= \alpha_i \left[\sqrt{2\pi}\left(\left(1+\mu_i^2\right)\left(1 - \operatorname{erf}\left(\frac{\mu_{c,i}}{\sqrt{2(\sigma_{c,i}^2+1)}}\right)\right)\right)\right. \\ &\quad \left. + \frac{2}{\sqrt{\sigma_{c,i}^2+1}} \exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)\left(\frac{\mu_{c,i}}{\sigma_{c,i}^2+1} - 2\mu_i\right)\right]\end{aligned} \tag{4.42}$$

The derivation of the above integrals are presented in Appendix A. Figure 4.3 shows examples of a lower constraint being applied to a standard normal distribution. As can be seen, the developed method is a good approximation of the truncated distribution in these examples.

The equations for applying an upper constraint of the form:

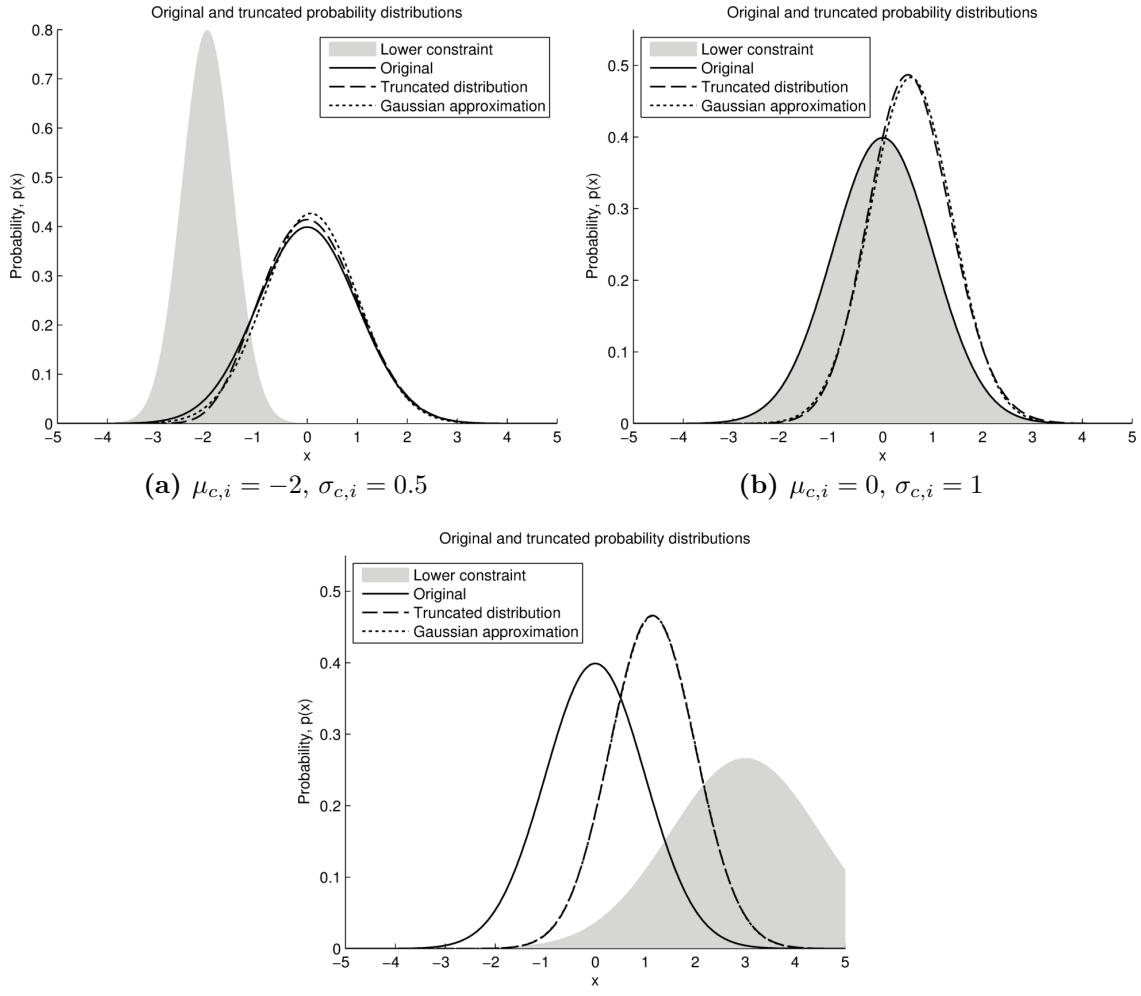
$$\boldsymbol{e}_i^T \boldsymbol{z}_i(k) \leq D_i(k) \tag{4.43}$$

are as follows:

$$\alpha_i = \frac{1}{\sqrt{2\pi}\left[1 + \operatorname{erf}\left(\frac{\mu_{d,i}}{\sqrt{2\left(\sigma_{d,i}^2+1\right)}}\right)\right]} \tag{4.44}$$

$$\mu_i = -\frac{2\alpha_i}{\sqrt{\sigma_{d,i}^2+1}} \exp\left(-\frac{\mu_{d,i}^2}{2(\sigma_{d,i}^2+1)}\right) \tag{4.45}$$

$$\begin{aligned}\sigma_i^2 = \alpha_i &\left[\sqrt{2\pi}\left(\left(1+\mu_i^2\right)\left(1 + \operatorname{erf}\left(\frac{\mu_{d,i}}{\sqrt{2(\sigma_{d,i}^2+1)}}\right)\right)\right)\right. \\ &\left. - \frac{2}{\sqrt{\sigma_{d,i}^2+1}} \exp\left(-\frac{\mu_{d,i}^2}{2(\sigma_{d,i}^2+1)}\right)\left(\frac{\mu_{d,i}}{\sigma_{d,i}^2+1} - 2\mu_i\right)\right]\end{aligned} \tag{4.46}$$



(a) $\mu_{c,i} = -2$, $\sigma_{c,i} = 0.5$

(b) $\mu_{c,i} = 0$, $\sigma_{c,i} = 1$

(c) $\mu_{c,i} = 3$, $\sigma_{c,i} = 1.5$

Figure 4.3 – A comparison of the actual truncated PDF and Gaussian approximation of the truncated PDF for various lower constraints.

4.2.3 Applying an Interval Soft Constraint

Now consider the case where there are two constraints. The two constraints acting on the transformed state are:

$$C_i(k) \leq \boldsymbol{e}_i^T \boldsymbol{z}_i(k) \leq D_i(k) \tag{4.47}$$

The resultant truncated function for the ith element of the constrained estimate, $\mathrm{Z}(\boldsymbol{e}_i^T \tilde{\boldsymbol{z}}_{i+1}(k))$, is given by:

$$\begin{aligned}
&\mathrm{Z}(\boldsymbol{e}_i^T \tilde{\boldsymbol{z}}_{i+1}(k)) \\
&\quad = \mathrm{PDF}(\boldsymbol{e}_i^T \boldsymbol{z}_i(k)) \times \mathrm{CDF}(C_i(k)) \times (1 - \mathrm{CDF}(D_i(k))) \\
&\quad = \frac{1}{4\sqrt{2\pi}} \exp\left(-\zeta^2/2\right) \left[1 + \mathrm{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right] \left[1 - \mathrm{erf}\left(\frac{\zeta - \mu_{d,i}}{\sigma_{d,i}\sqrt{2}}\right)\right]
\end{aligned} \tag{4.48}$$

This function, once normalised, represents the conditional probability distribution of $\boldsymbol{e}_i^T \boldsymbol{z}_i(k)$ given $C_i(k) \leq \boldsymbol{e}_i^T \boldsymbol{z}_i(k) \leq D_i(k)$.

The integrals required to calculate the area, mean, and variance of the above function contain integrals of the form:

$$\int_{-\infty}^{\infty} \exp\left(-x^2\right) \mathrm{erf}(ax + b)\mathrm{erf}(\alpha x + \beta)\mathrm{d}x \tag{4.49}$$

which does not have an analytical solution. The following approximation is proposed for the $\mathrm{CDF}(C_i(k)) \times (1 - \mathrm{CDF}(D_i(k)))$ term in Equation 4.48:

$$\begin{aligned}
&\left[1 + \mathrm{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right] \left[1 - \mathrm{erf}\left(\frac{\zeta - \mu_{d,i}}{\sigma_{d,i}\sqrt{2}}\right)\right] \\
&\approx 2\left[\mathrm{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) - \mathrm{erf}\left(\frac{\zeta - \mu_{d,i}}{\sigma_{d,i}\sqrt{2}}\right)\right]
\end{aligned} \tag{4.50}$$

This yields an approximation of the truncated function given by Equation 4.48 of:

$$\frac{1}{2\sqrt{2\pi}} \exp\left(-\zeta^2/2\right) \left[\text{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) - \text{erf}\left(\frac{\zeta - \mu_{d,i}}{\sigma_{d,i}\sqrt{2}}\right)\right] \tag{4.51}$$

This approximation relies on the condition that $\mu_{c,i} < \mu_{d,i}$, and the assumption that the distributions of the constraints do not significantly overlap. If these are satisfied, it is highly likely that one of the error function terms will be equal to 1 when the other is at an intermediate value, giving a good approximation of the actual distribution, as illustrated in Figure 4.4.

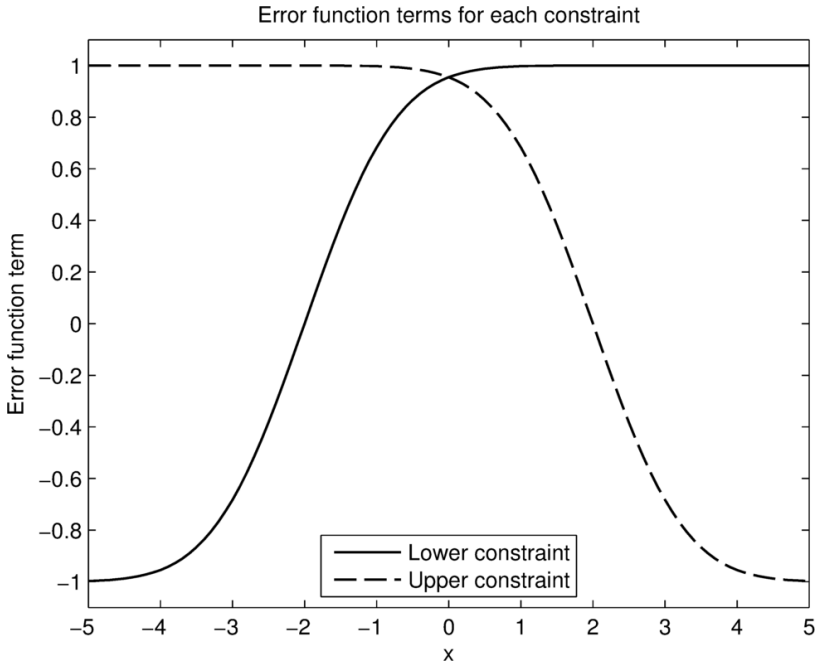


Figure 4.4 – Error function terms for a lower constraint with $\mu_{c,i} = -2$ and $\sigma_{c,i} = 1$ and an upper constraint with $\mu_{d,i} = 2$ and $\sigma_{d,i} = 1$. In this case, one of the error function terms is always close to 1.

An overlap metric, γ, is defined as:

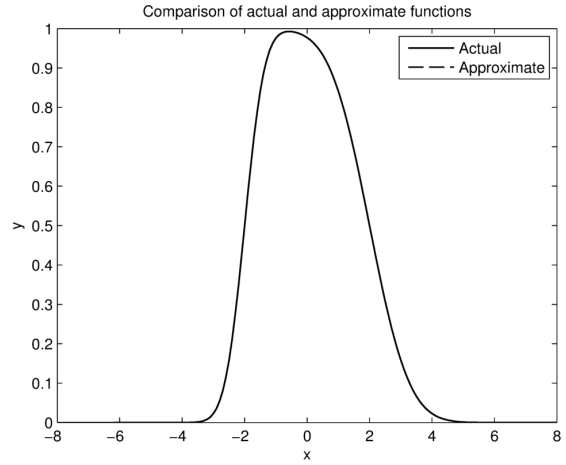
$$\gamma = \frac{\mu_{d,i} - \mu_{c,i}}{\sigma_{d,i} + \sigma_{c,i}} \tag{4.52}$$

and a shape metric, δ, is defined as:

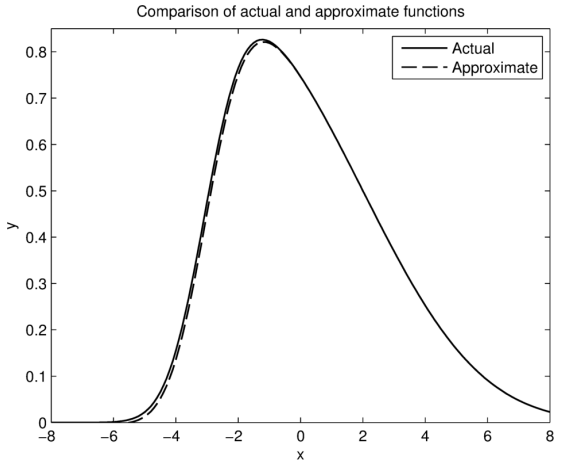
$$\delta = \left| \log \left(\frac{\sigma_{c,i}}{\sigma_{d,i}} \right) \right| \tag{4.53}$$

γ is a measure of how much the probability distributions of the two constraints overlap, and δ is a measure of how different the shapes of the probability distributions of the constraints are. Figure 4.5 shows examples of this approximation applied to several different cases of γ and δ. As can be seen, the approximation is an almost perfect approximation in Figure 4.5a, with the approximation degrading as γ decreases in the other examples. In Figure 4.5d, the constraint distributions substantially overlap one another and the approximation suffers as a result. Figure 4.6 compares the PDF generated using the approximation with the actual truncated PDF for the same values of γ and δ as Figure 4.5. For the first two cases, the approximate PDF is an almost perfect approximation of the actual PDF. In Figure 4.6c, approximate PDF is also quite close to the actual PDF, despite the difference in the approximations shown in Figure 4.5c. Finally, in Figure 4.6d, there is a large deviation between the approximate and actual PDFs.

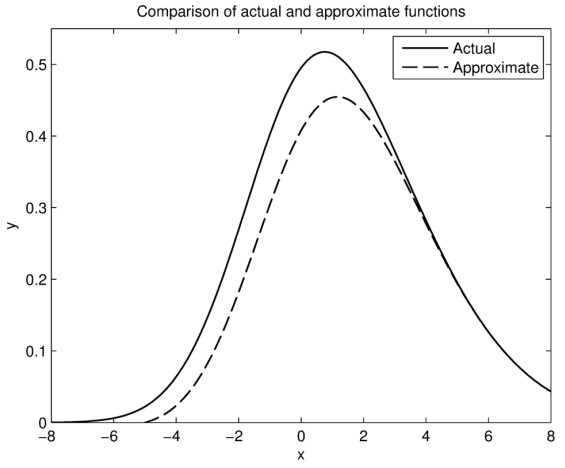
Figure 4.7 shows the Kullback-Leibler (KL) divergence [58] between the actual and approximate distributions for various δ and γ. The KL divergence is a measure of the difference between two distributions, and as it nears 0, the approximation approaches the actual distribution. Increasing γ and decreasing δ improves the approximation. For $\gamma \geq 3$, the approximation very closely matches the actual distribution, regardless of δ.



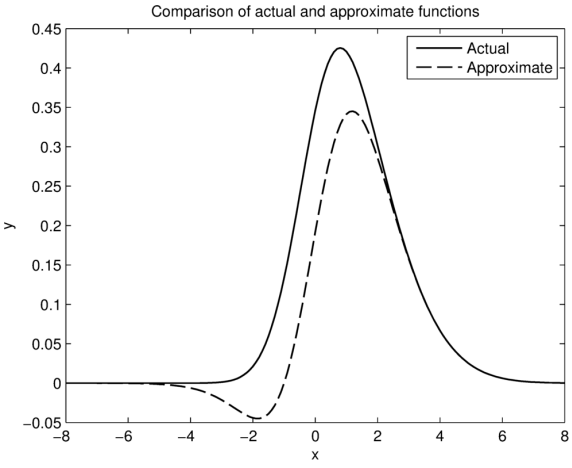
(a) $\mu_{c,i} = -2$, $\sigma_{c,i} = 0.5$, $\mu_{d,i} = 2$, $\sigma_{d,i} = 1$, corresponding to $\gamma = 2.67$ and $\delta = 0.30$



(b) $\mu_{c,i} = -3$, $\sigma_{c,i} = 1$, $\mu_{d,i} = 2$, $\sigma_{d,i} = 3$, corresponding to $\gamma = 1.25$ and $\delta = 0.48$

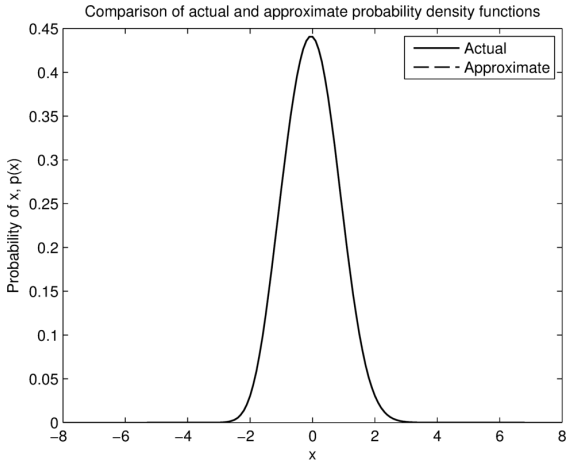


(c) $\mu_{c,i} = -1$, $\sigma_{c,i} = 2$, $\mu_{d,i} = 2$, $\sigma_{d,i} = 3.5$, corresponding to $\gamma = 0.55$ and $\delta = 0.24$

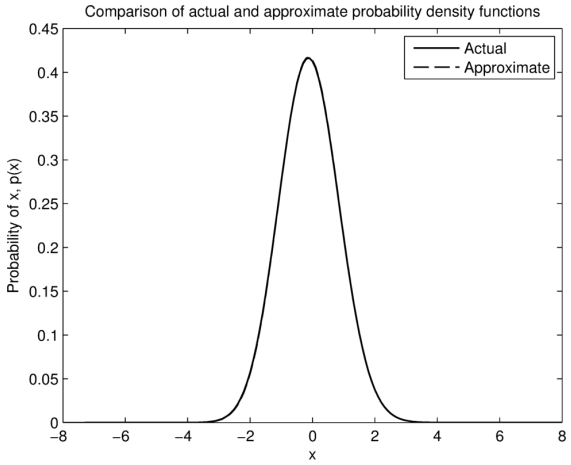


(d) $\mu_{c,i} = 0$, $\sigma_{c,i} = 1$, $\mu_{d,i} = 1$, $\sigma_{d,i} = 2$, corresponding to $\gamma = 0.33$ and $\delta = 0.30$

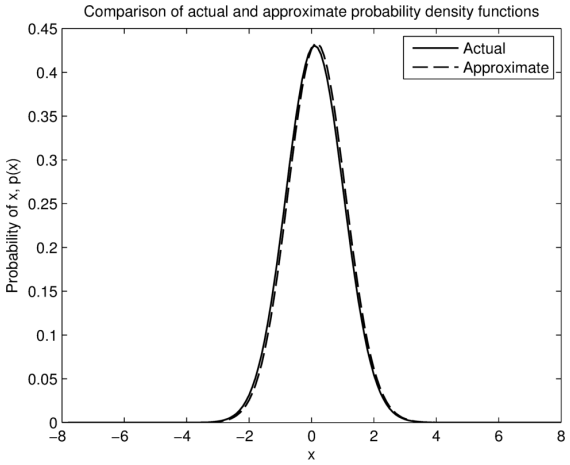
Figure 4.5 – A comparison of the actual and approximate functions for the interval constraint given in Equation 4.50 for various overlap metrics, γ, and shape metrics, δ. In (a), the approximate function is an almost perfect approximation of the actual function. In (d), the constraint distributions overlap significantly.



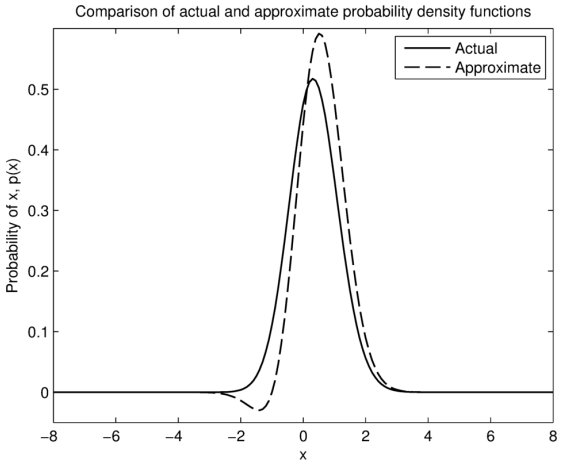
(a) $\mu_{c,i} = -2$, $\sigma_{c,i} = 0.5$, $\mu_{d,i} = 2$, $\sigma_{d,i} = 1$, corresponding to $\gamma = 2.67$ and $\delta = 0.30$



(b) $\mu_{c,i} = -3$, $\sigma_{c,i} = 1$, $\mu_{d,i} = 2$, $\sigma_{d,i} = 3$, corresponding to $\gamma = 1.25$ and $\delta = 0.48$



(c) $\mu_{c,i} = -1$, $\sigma_{c,i} = 2$, $\mu_{d,i} = 2$, $\sigma_{d,i} = 3.5$, corresponding to $\gamma = 0.55$ and $\delta = 0.24$



(d) $\mu_{c,i} = 0$, $\sigma_{c,i} = 1$, $\mu_{d,i} = 1$, $\sigma_{d,i} = 2$, corresponding to $\gamma = 0.33$ and $\delta = 0.30$

Figure 4.6 – A comparison of the actual and approximate truncated PDFs for the interval constraint for the same overlap metrics, γ, and shape metrics, δ as Figure 4.5. In (a) and (b), the approximate PDF is an almost perfect approximation of the actual PDF.

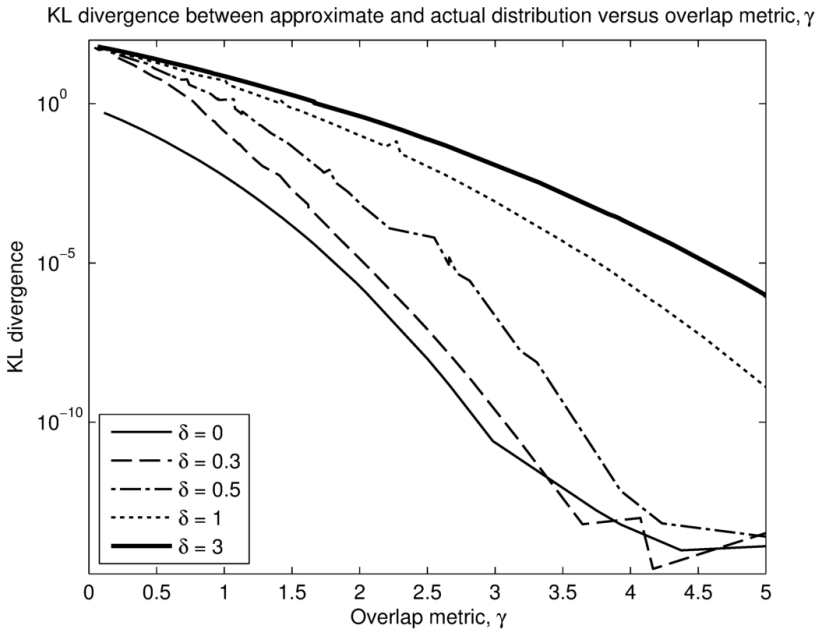


Figure 4.7 – Comparison of the KL divergence between the actual and approximate distributions as a function of the overlap metric, γ, where the lines are of constant shape metric, δ. For $\delta > 3$, the KL divergence is approximately the same as for $\delta = 3$. The unevenness of the lines is due to sampling noise.

To normalise the approximated function to a PDF, the area is given by:

$$\begin{aligned}
&\int_{-\infty}^{\infty} \mathrm{Z}(\boldsymbol{e}_i^T \tilde{\boldsymbol{z}}_{i+1}(k))\mathrm{d}\zeta \\
&\quad = \int_{-\infty}^{\infty} \frac{1}{4\sqrt{2\pi}} \exp\left(-\zeta^2/2\right) \left[2\left(\mathrm{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) - \mathrm{erf}\left(\frac{\zeta - \mu_{d,i}}{\sigma_{d,i}\sqrt{2}}\right)\right)\right] \mathrm{d}\zeta \qquad (4.54) \\
&\quad = \frac{1}{2}\left[\mathrm{erf}\left(\frac{\mu_{d,i}}{\sqrt{2\left(\sigma_{d,i}^2 + 1\right)}}\right) - \mathrm{erf}\left(\frac{\mu_{c,i}}{\sqrt{2\left(\sigma_{c,i}^2 + 1\right)}}\right)\right]
\end{aligned}$$

The mean of the PDF is then given by:

$$\begin{aligned}\mu_i &= \alpha_i \int_{-\infty}^{\infty} \zeta \exp\left(-\zeta^2/2\right) \left[\left(\operatorname{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) - \operatorname{erf}\left(\frac{\zeta-\mu_{d,i}}{\sigma_{d,i}\sqrt{2}}\right)\right)\right] \mathrm{d}\zeta \\ &= 2\alpha_i \left(\frac{1}{\sqrt{\sigma_{c,i}^2+1}} \exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right) - \frac{1}{\sqrt{\sigma_{d,i}^2+1}} \exp\left(-\frac{\mu_{d,i}^2}{2(\sigma_{d,i}^2+1)}\right)\right) \end{aligned} \tag{4.55}$$

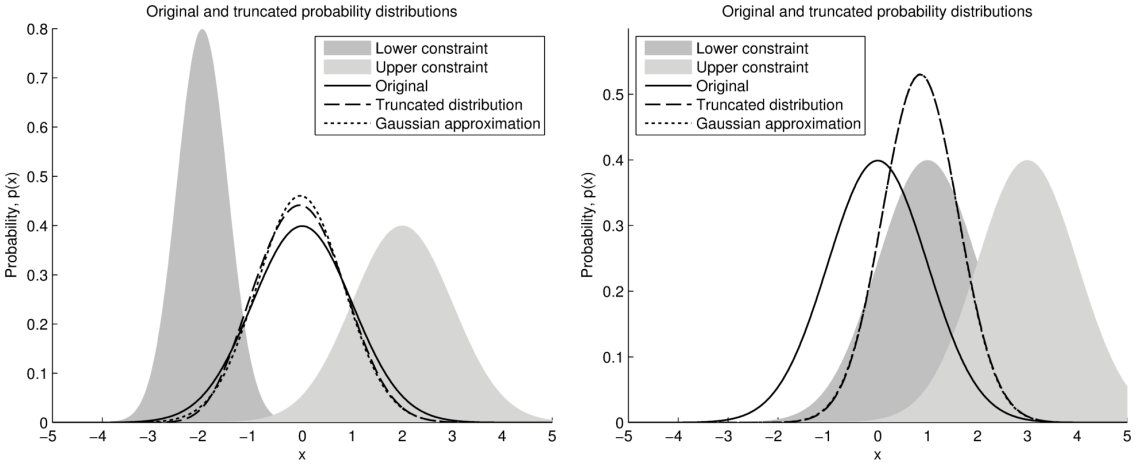
where

$$\alpha_i = \frac{1}{\sqrt{2\pi}\left[\operatorname{erf}\left(\frac{\mu_{d,i}}{\sqrt{2\left(\sigma_{d,i}^2+1\right)}}\right) - \operatorname{erf}\left(\frac{\mu_{c,i}}{\sqrt{2\left(\sigma_{c,i}^2+1\right)}}\right)\right]} \tag{4.56}$$

Finally, the variance is given by:

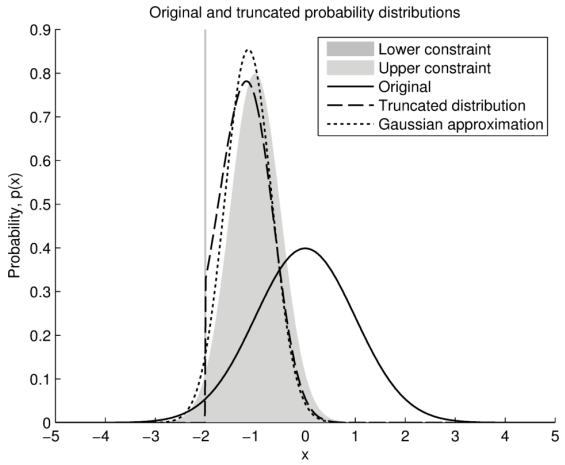
$$\begin{aligned}\sigma_i^2 &= \alpha_i \int_{-\infty}^{\infty} (\zeta-\mu_i)^2 \exp\left(-\zeta^2/2\right) \left[\left(\operatorname{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) - \operatorname{erf}\left(\frac{\zeta-\mu_{d,i}}{\sigma_{d,i}\sqrt{2}}\right)\right)\right] \mathrm{d}\zeta \\ &= \alpha_i \left[\sqrt{2\pi}\left(\left(1+\mu_i^2\right)\left(\operatorname{erf}\left(\frac{\mu_{d,i}}{\sqrt{2(\sigma_{d,i}^2+1)}}\right) - \operatorname{erf}\left(\frac{\mu_{c,i}}{\sqrt{2(\sigma_{c,i}^2+1)}}\right)\right)\right)\right. \\ &\qquad + \frac{2}{\sqrt{\sigma_{c,i}^2+1}} \exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)\left(\frac{\mu_{c,i}}{\sigma_{c,i}^2+1} - 2\mu_i\right) \\ &\qquad \left. - \frac{2}{\sqrt{\sigma_{d,i}^2+1}} \exp\left(-\frac{\mu_{d,i}^2}{2(\sigma_{d,i}^2+1)}\right)\left(\frac{\mu_{d,i}}{\sigma_{d,i}^2+1} - 2\mu_i\right)\right] \end{aligned} \tag{4.57}$$

The derivations of these integrals are not provided, but can be easily derived using the solutions provided in Appendix A for the one-sided constraint. Example intervals constraints applied to a standard normal distribution are shown in Figure 4.8. In Figure 4.8c, the lower constraint is effectively a hard constraint, and the actual truncated PDF is highly non-Gaussian in the left tail.



(a) $\mu_{c,i} = -2$, $\sigma_{c,i} = 0.5$, $\mu_{d,i} = 2$, $\sigma_{d,i} = 1$, corresponding to $\gamma = 2.67$ and $\delta = 0.30$

(b) $\mu_{c,i} = 1$, $\sigma_{c,i} = 1$, $\mu_{d,i} = 3$, $\sigma_{d,i} = 1$, corresponding to $\gamma = 1$ and $\delta = 0$



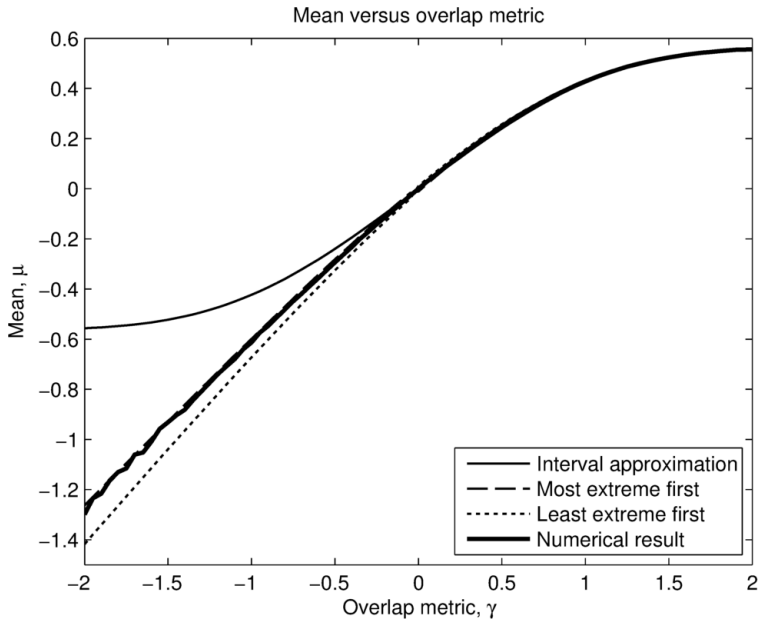
(c) $\mu_{c,i} = -2$, $\sigma_{c,i} = 0.001$, $\mu_{d,i} = -1$, $\sigma_{d,i} = 0.5$, corresponding to $\gamma = 2$ and $\delta = 2.70$

Figure 4.8 – A comparison of the actual truncated PDF and Gaussian approximation of the truncated PDF for various interval constraints.

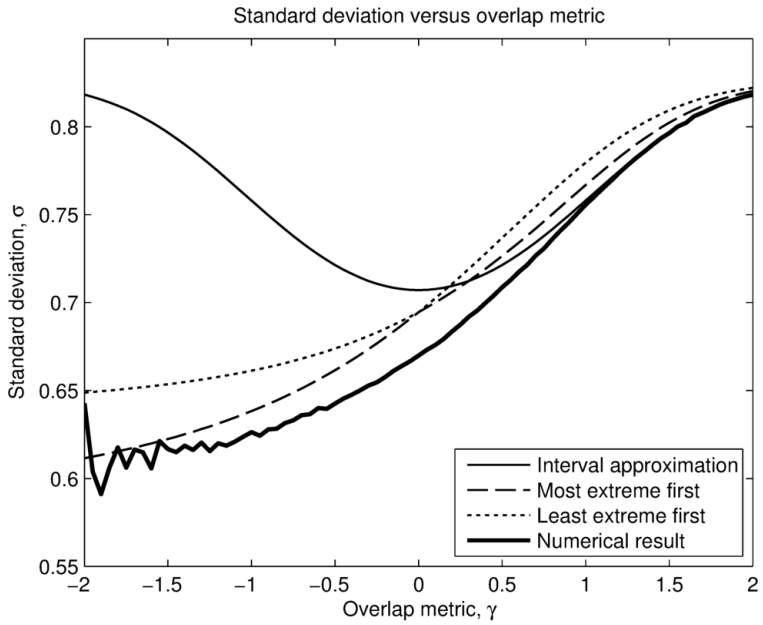
What to do when the interval approximation conditions are not satisfied

The approximation in Equation 4.51 is a good approximation when $\mu_{c,i} < \mu_{d,i}$ and the two constraint distribution do not significantly overlap. When these conditions are not met, the approximation can be poor. In particular, when $\gamma \leq 0$, the approximation is not valid. In such cases, a better approximation of the constrained distribution can be achieved by applying the constraints individually as one-sided constraints instead of an interval constraint. It was found that it is generally best to apply the highest variance constraint first and the lowest variance constraint second if $\delta > 0.5$. If $\delta < 0.5$, then applying the constraints from most extreme to least extreme tends to produce the best results. The lower constraint is considered the most extreme constraint if $\mu_{c,i} > -\mu_{d,i}$, otherwise the upper constraint is the most extreme constraint.

Figure 4.9 shows the calculated mean and standard deviation when $\delta = 0$ using the various methods and compared against a numerically calculated mean and standard deviation. These clearly show the degradation in performance of the interval constraint approximation, particularly for $\gamma \leq 0$. Of the two possible orders of applying the constraints, applying the most extreme constraint first consistently yields the best results. Figure 4.10 and Figure 4.11 compare the methods for $\delta = 0.5$. The interval approximation very quickly diverges from the actual result when $\gamma \leq 1$. For this case, applying the highest variance constraint first produces results closest to the numerical answer.

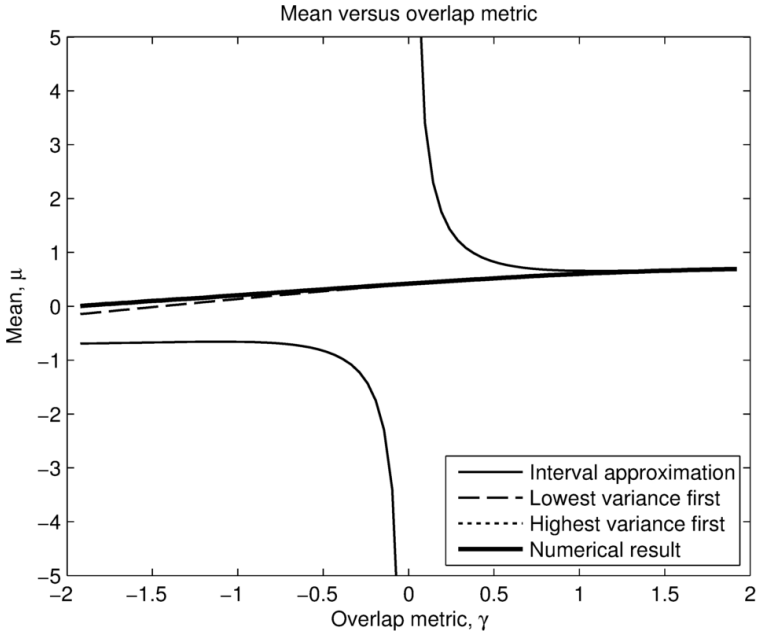


(a) Comparing the mean, μ

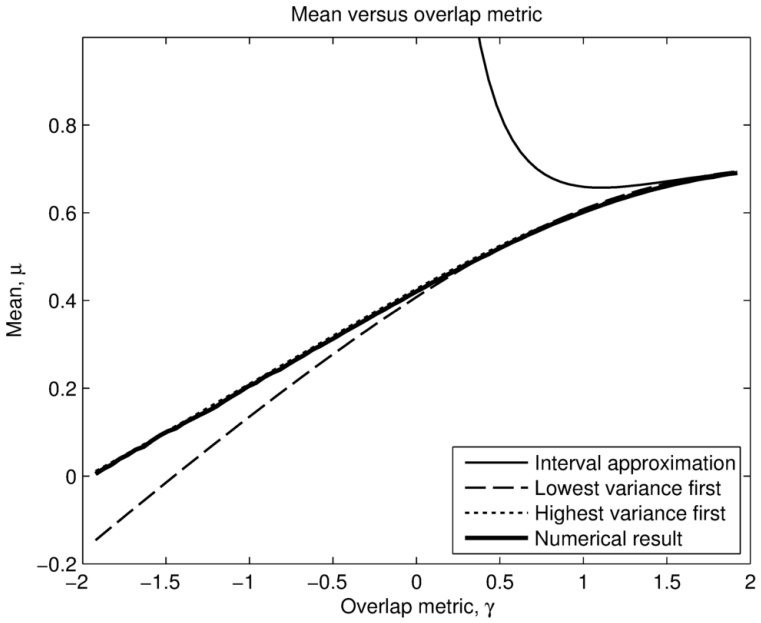


(b) Comparing the standard deviation, σ

Figure 4.9 – (a) A comparison of the resultant mean, μ, when applying an interval constraint for $\delta = 0$. Note that the result for applying the most extreme constraint first is essentially the same as the numerical result. (b) A comparison of the resultant standard deviation, σ, when applying an interval constraint for $\delta = 0$. The methods tested were applying the constraints using the interval constraint formulation, and as two one-sided constraints applying either the most extreme or least extreme constraint first. Numerical results were obtained through MC analysis.

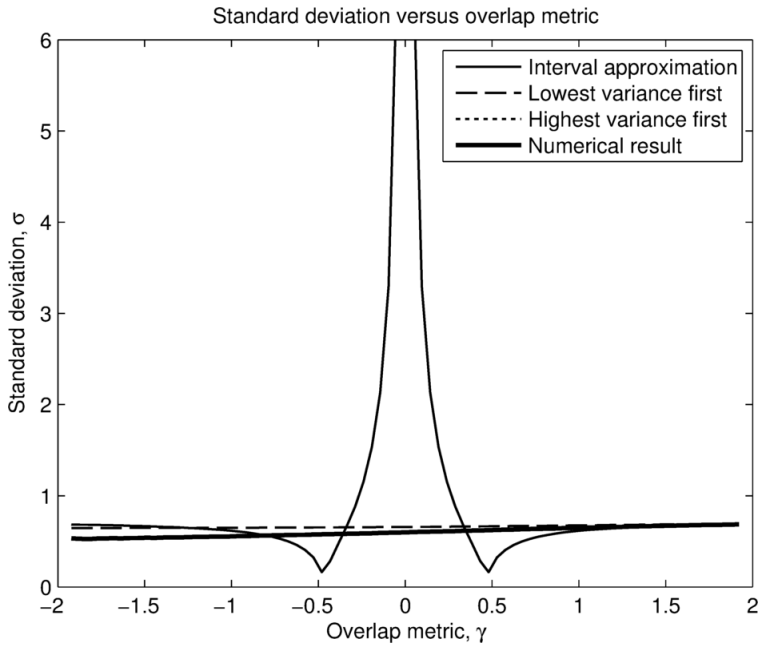


(a) Comparing the mean, μ

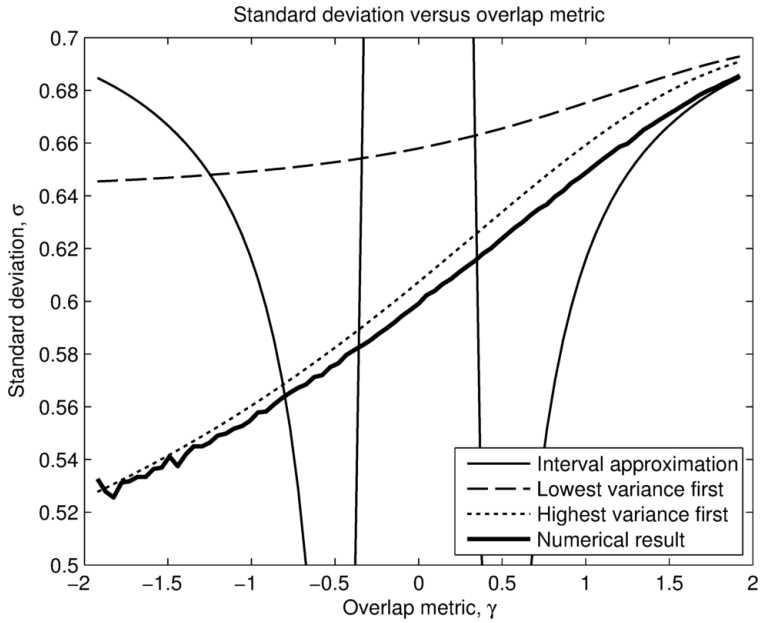


(b) Zoomed in version comparing the mean, μ

Figure 4.10 – A comparison of the resultant mean, μ, when applying an interval constraint for $\delta = 0.5$. (b) is a zoomed in version of (a). The methods tested were applying the constraints using the interval constraint formulation, and as two one-sided constraints applying either the lowest variance or highest variance constraint first. Numerical results were obtained through MC analysis. The result obtained by applying the highest variance constraint first is mostly hidden by the numerical result.



(a) Comparing the standard deviation, σ



(b) Zoomed in version comparing the standard deviation, σ

Figure 4.11 – A comparison of the resultant standard deviation, σ, when applying an interval constraint for $\delta = 0.5$. (b) is a zoomed in version of (a). The methods tested were applying the constraints using the interval constraint formulation, and as two one-sided constraints applying either the lowest variance or highest variance constraint first. Numerical results were obtained through MC analysis. Applying the highest variance constraint first yielded results closest to the numerical result.

4.3 Results

The soft-constrained KF was compared against an unconstrained KF and a hard-constrained KF that treats the soft constraints as hard constraints. The resource level of a user agent with a capacity of 1000L was estimated using noisy discrete sensors placed at every 10% of capacity from 10% to 90%. The methods were evaluated with uncertainty on the set-point of the discrete sensors of 0%, 0.5%, 1%, 1.5%, 2%, and 2.5%. Perfect knowledge was assumed at levels of 0% and 100%. The user agent used the resource at a mean rate of 0.5L/s with a standard deviation of 0.05L/s. The operation of the user agent was simulated 1000 times for 30 minutes with the resource level starting at 100% with no uncertainty. After approximately 27 minutes, the user agent was replenished by a replenishment agent which has a mean replenishment rate of 10L/s and a standard deviation of 0.5L/s. The estimate and uncertainty provided by the soft constrained KF for one of the simulations is shown in Figure 4.12.

Each method was tested in 1000 simulations for each sensor uncertainty, and the Root Mean Squared Error (RMSE) for each method was calculated. The RMSE for each method is shown in Table 4.1. The RMSE and percentage improvement of each method are graphed in Figure 4.13. When the sensors had no uncertainty, the hard- and soft-constrained methods produced the exact same result, outperforming the unconstrained KF resulting by 14.5%. The tracking performance of all methods degraded as the uncertainty of the sensors was increased—the information provided by the constraint is reduced as the uncertainty is increased. The behaviour of the soft-constrained KF approaches that of the unconstrained KF when the sensors have an uncertainty of 2.5%, while the hard-constrained KF is actually outperformed by the unconstrained KF in this case by 7.4%. In this case, the soft-constrained KF outperforms the unconstrained and hard-constrained KFs by 3.9% and 11.1% respectively. In all cases, the soft-constrained KF performs as good as, or better than, the other methods.

The main advantage of the soft-constrained KF is that it produces more confident estimates than the unconstrained KF when the estimate is close to the constraints,

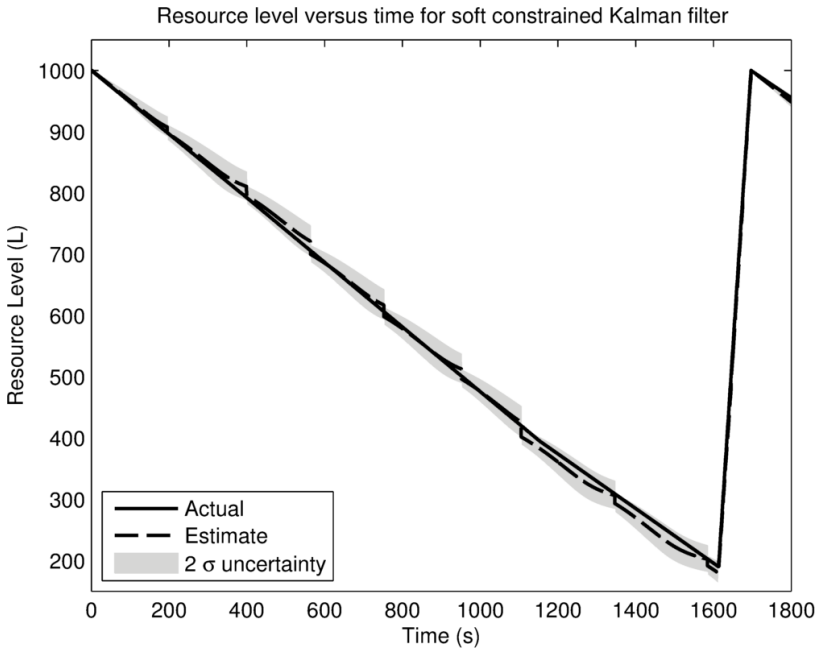


Figure 4.12 – Soft-constrained KF example with sensor uncertainty of 1% of maximum capacity. The dashed line is the estimate of the resource level, and the grey shading represents an uncertainty envelope around the estimate of two standard deviations either side. The soft-constrained KF uses the positions and uncertainty of the sensors to improve the estimate of the resource level as it approaches the sensors.

Table 4.1 – RMSE (L) for each Kalman filter method

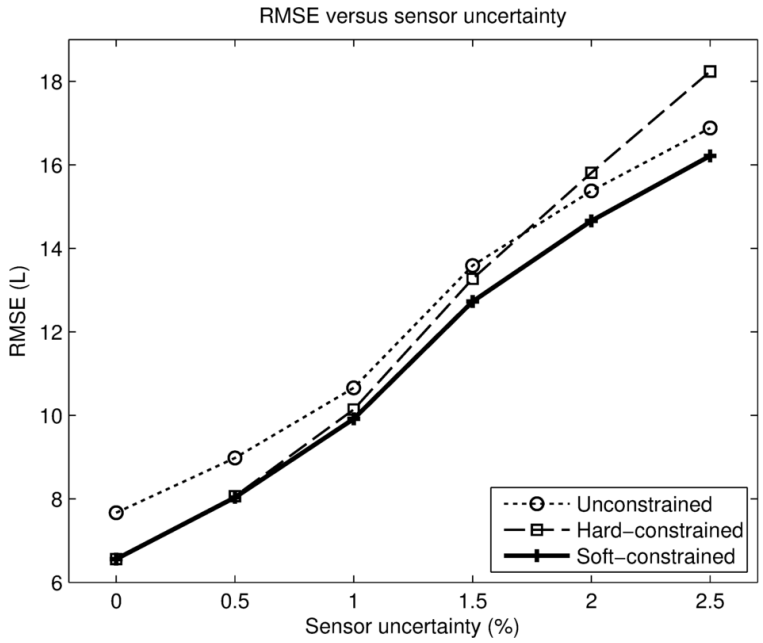
Sensor uncertainty	0%	0.5%	1%	1.5%	2%	2.5%
Unconstrained	7.67	8.98	10.66	13.58	15.37	16.88
Hard-constrained	6.56	8.06	10.14	13.27	15.81	18.24
Soft-constrained	6.56	8.04	9.91	12.72	14.66	16.22

without producing overconfident estimates like the hard-constrained KF. This is illustrated in the first example shown in Figure 4.14. When the estimate is far away from the constraints, the soft-constrained KF produces almost exactly the same result as the unconstrained KF. As the estimate approaches the constraint, the behaviour of the methods diverge. The unconstrained KF does not use the information provided by the constraint so the uncertainty of the estimate continues to grow, even though most of the uncertainty lies beyond the constraint. The soft-constrained method on the other hand, uses the knowledge of the constraint to produce an estimate that is

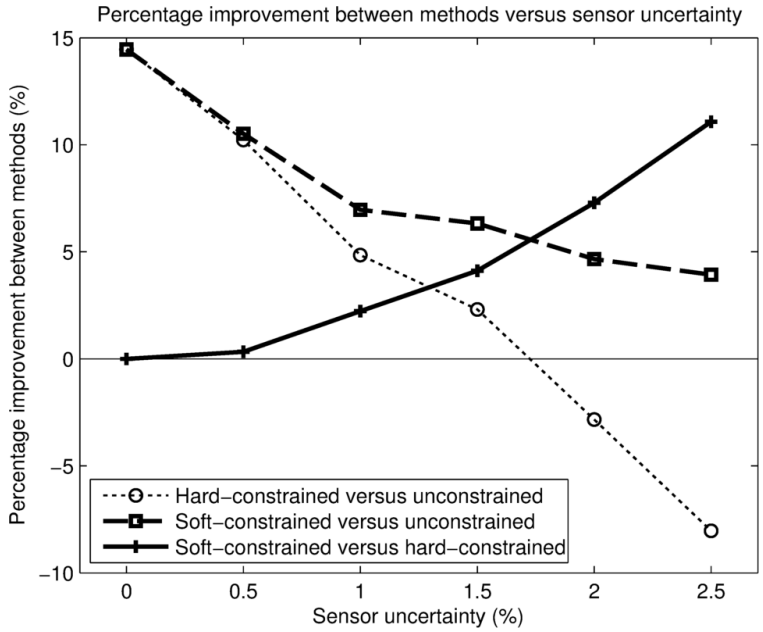
closer to the actual value with higher confidence. The hard-constrained KF uses the mean value of the constraint to try to improve the quality of the estimate. However, it results in overconfident estimates which stray from the actual resource level.

In cases where the estimate of the unconstrained KF lies close to the actual value, the constrained methods can produce worse results in terms of RMSE. The second example shown in Figure 4.15 demonstrates this. In this example, the hard-constrained KF again produces an overconfident estimate close to the constraint. While the soft-constrained KF has a higher tracking error than the unconstrained KF close to the constraint, it produces a more confident estimate while still containing the actual value within a two-σ confidence interval.

Additional results of the soft-constrained KF applied to other applications are presented in Appendix B.

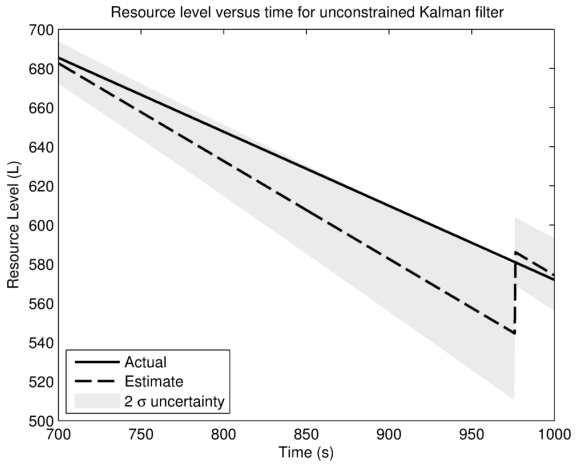


(a) RMSE

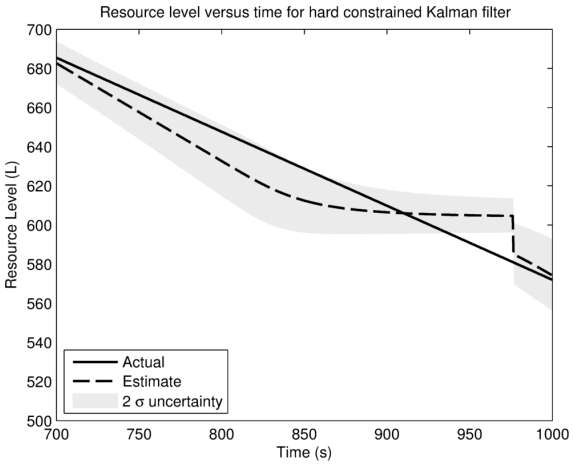


(b) Percent difference

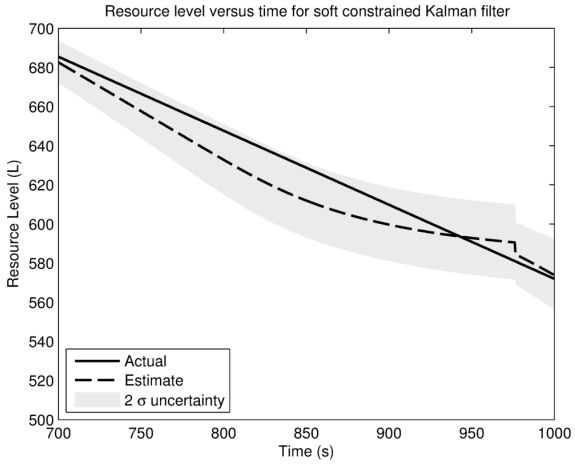
Figure 4.13 – Constrained KF results. (a) shows the RMSE of each method versus the sensor uncertainty. The sensor uncertainty is a percentage of the total capacity of the user agent. (b) shows the comparative percentage improvement of each method versus the sensor uncertainty. Points positioned above 0 are an improvement over the comparison method, while points positioned below 0 are a decrease in performance compared to the comparison method. The soft-constrained KF performs as good as, or better than, the unconstrained and hard-constrained methods, while the hard-constrained KF is outperformed by the unconstrained KF when the sensor uncertainty is high.



(a) Unconstrained KF

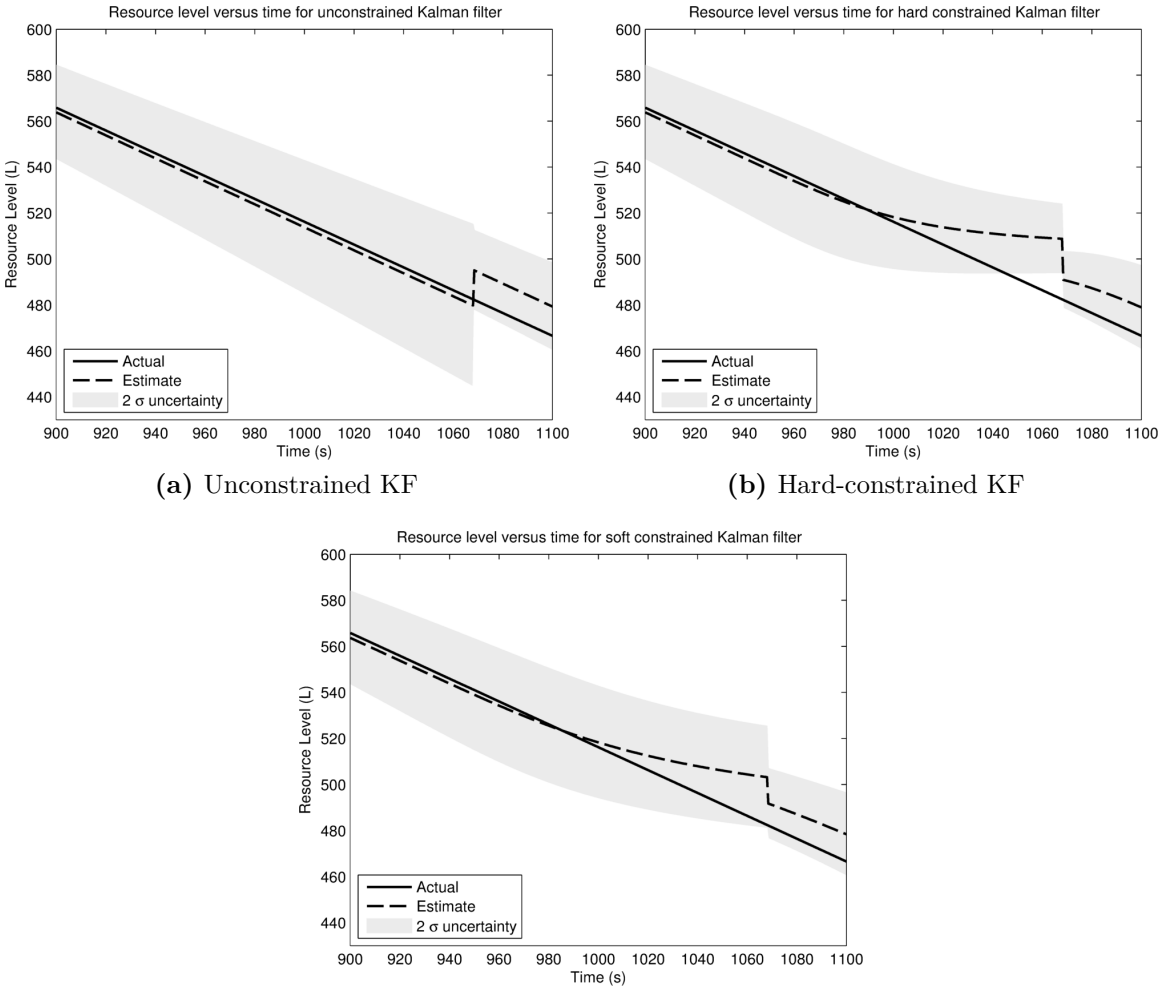


(b) Hard-constrained KF



(c) Soft-constrained KF

Figure 4.14 – This example shows the behaviour of the various KF approaches as it approaches an uncertain sensor at a resource level of 60% of maximum capacity (600L), where the sensor has uncertainty of 1%. (a) shows the unconstrained KF. In (b), the hard-constrained KF is overconfident, resulting in a two-σ confidence interval that does not include the actual resource level. Using the soft-constrained KF in (c), the actual resource level stays within the uncertainty of the estimate.



(a) Unconstrained KF

(b) Hard-constrained KF

(c) Soft-constrained KF

Figure 4.15 – This example shows the behaviour of the various KF approaches as it approaches an uncertain sensor at a resource level of 50% of maximum capacity (500L), where the sensor has uncertainty of 1%. In this example, the estimate produced by the unconstrained KF in (a) lies very close to the actual value. In (b), the hard-constrained KF is overconfident, resulting in a two-σ confidence interval that does not include the actual resource level. The actual resource level lies within the two-σ confidence interval produced by the soft-constrained KF in (c), despite the estimate diverging from the actual resource level.

4.4 Summary

This chapter introduced methods for estimating the current state of a system. Section 4.1 presented the general formulation of the KF and an existing truncation method for dealing with hard constraints on the state. Hard constraints are common in SCAR scenarios as resource levels are subject to hard limits. Some sensors that are used for measuring the resource level can be modelled as a soft constraint on the system state. Section 4.2 extended the truncation method to soft constraints, deriving novel analytical methods for approximating the resultant truncated Gaussian distribution.

The soft-constrained KF was shown in Section 4.3 to consistently outperform a regular unconstrained KF and a KF that assumed hard constraints. It resulted in a lower RMSE of up to 14.5% and 11.1% compared to the unconstrained and hard-constrained KFs respectively. The soft constraints were used to reduce uncertainty in the state estimate compared to the unconstrained KF without producing the overconfident results of the hard-constrained KF.

The truncation approach developed for constraining the state estimate is not exclusive to KFs, and can be applied in any situation that deals with Gaussian distributions that have constraints. In the most general sense, the truncation approach yields a Gaussian approximation of the conditional probability distribution of a Gaussian distributed variable that is conditioned on other Gaussian distributed variables. This is used in Chapter 5 for conditioning the probability distributions of the arrival times of replenishment agents at a user agent based on their order of arrival.

Chapter 5 next presents methods for predicting the future state of the system, taking the estimated system state from the soft constrained KF method developed in this chapter as an input. The prediction methods make use of Gaussian approximations to quickly calculate the expected future state.

Chapter 5

Prediction

This chapter develops methods for predicting the future state of the system. As shown in Figure 5.1, the prediction component of belief space scheduling sits within the optimisation component, and is used by the optimisation methods to evaluate the tasks and schedules under consideration. It does this by predicting a belief of the future state of the system based on the belief of the current state, the task or schedule of tasks, and a model of the system.

The majority of approaches to prediction within the scheduling literature either ignore uncertainty [8], use Monte Carlo (MC) simulation [83], or get a worst-case estimate by using conservative estimates of the uncertain parameters [70]. Belief space planning approaches, on the other hand, typically use a Kalman Filter (KF) to both estimate the current state of the agent and predict the future evolution of the agent given a plan. They use a discrete-time approach, predicting the outcome of an action and simulated observation at each time-step. Many of these approaches simulate multiple observations at each time-step to build an accurate representation of the future belief [11, 59]. These approaches require significant computation time and generally use very short planning horizons [44, 45]. An assumption that is commonly used to reduce the computational requirements of belief space planning methods is the maximum likelihood observation assumption, where the simulated observation is the most likely observation given the belief of the agent [81].

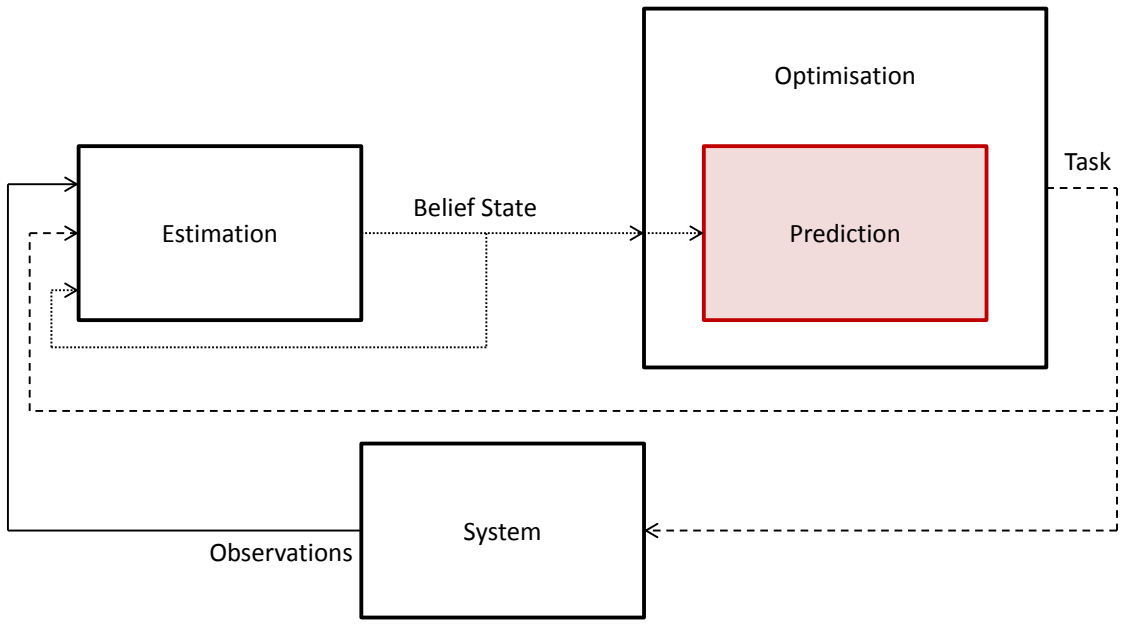


Figure 5.1 – The components of belief space scheduling. This chapter outlines methods for predicting the future state of the system.

Unlike belief space planning problems, where a different action is taken at each time-step, the tasks in scheduling problems may take many time-steps to be completed. This makes predicting the future state of the system using a discrete-time approach problematic—in the time that one task is completed, many hundreds or thousands of observations may have been received. Simulating multiple observations to accurately represent the future belief is simply infeasible—predicting 100 steps into the future with only 10 sampled observations per time-step would lead to a belief tree with 10^{100} nodes. Using maximum likelihood observations is also not appropriate as it can lead to unrepresentative simulated beliefs. Consider an extreme example where the system has perfect sensing—using the maximum likelihood observation assumption leads to a simulated belief that has no uncertainty, thus ignoring the uncertainty in the system parameters.

Given the issues with simulating observations discussed above, the methods developed in this chapter assume that no future observations are made. This enables the use of continuous-time methods which calculate probability distributions for the times at

which events occur. Two types of methods are explored—a MC approach and an analytical approach. The MC approach uses multiple samples of the parameter and belief distributions to construct a belief of the future state. The analytical method, instead of sampling the distributions, uses approximations to enable the entire probability distributions to be propagated into the future.

Section 5.1 first formulates the MC prediction method for SCAR scenarios. Section 5.2 then presents the analytical method for single-replenishment agent SCAR scenarios, including the approximations required to use parameters and beliefs that have uncertainty described by Gaussian distributions. Section 5.3 then extends the analytical method to multi-replenishment agent SCAR scenarios. Finally, Section 5.4 introduces several scenarios and evaluates the analytical methods against the MC approach.

5.1 Monte Carlo Method

This section outlines the MC approach for predicting the future state in a SCAR scenario given a schedule. The basic idea behind MC simulation is to simulate the evolution of the system many times using a deterministic model with parameter values sampled from the probability distributions for each parameter. By using enough samples, eventually an accurate distribution of the future state will be obtained. MC approaches have been widely used for the estimation problem [9, 103], and the error of the solutions produced by these results is known to decrease as a function of the square root of the number of samples [28]. Consequently, achieving accurate results in high dimensional problems can be computationally very expensive. It is introduced in this thesis primarily as a benchmark for the analytical methods developed in Section 5.2 and Section 5.3.

The algorithm for the MC method is outlined in Algorithm 5.1. It begins by sampling a state from the initial state on line 3. It then considers the tasks in the schedule in the order that the replenishment agents arrive.

Algorithm 5.1: Monte Carlo method

`MonteCarlo(`$\boldsymbol{\psi}_{initial}, \boldsymbol{\theta}, n_{samples}$`)`
input : Current system state, $\boldsymbol{\psi}_{initial}$; schedule, $\boldsymbol{\theta}$; number of samples, $n_{samples}$
output: Weighted downtime, λ; state, $\boldsymbol{\psi}$

$\lambda \leftarrow 0$
for $n \in \{1, ..., n_{samples}\}$ **do**
  sample $l_{u,i}$ and $l_{a,j}$ from $\boldsymbol{\psi}_{initial}$ $\forall i, j$
  while *there are tasks remaining in the schedule,* $\boldsymbol{\theta}$ **do**
    $t_{a,j} \leftarrow t_{l,j} + \dfrac{s}{v_{a,j}}\ \forall j$ `// calculate arrival time`
    $j \leftarrow$ index of replenishment agent that will arrive at its next task first
    if *the task is to by replenished by the replenishment point* **then**
      $t_{f,j} \leftarrow t_{a,j} + t_{sr} + \dfrac{c_{a,j} - l_{a,j}}{r_r} + t_{pr}$ `// replenishment finish time`
      $l_{a,j} \leftarrow c_{a,j}$
    else
      $i \leftarrow$ index of user agent that replenishment agent j is replenishing
      if *there are already replenishment agents replenishing or waiting to replenish user agent* i *when replenishment agent* j *arrives* **then**
        $K \leftarrow$ set of replenishment agents at user agent i
        $t_{a,j} \leftarrow \max\left(t_{a,j}, t_{l,k}\ \forall k \in K\right)$ `// wait for other agents`
      $t_{b,j,i} \leftarrow t_{a,j} + t_{sa,j}$ `// time after setup`
      $t_{d,i} \leftarrow t_{f,i} + \dfrac{l_{u,i}}{r_{u,i}}$ `// deadline`
      $t_{c,j,i} \leftarrow \max\left(0,\ t_{b,j,i} - t_{d,i}\right)$ `// downtime of user agent`
      $\lambda \leftarrow \lambda + w_i t_{c,j,i}$ `// weighted downtime`
      $l_{u,i} \leftarrow \max\left(0,\ l_{u,i} - t_{f,i} r_{u,i}\right)$ `// level before replenishment`
      $t_{r,j,i} \leftarrow \min\left(\dfrac{l_{a,j}}{r_{a,j}},\ \dfrac{c_{u,i} - l_{u,i}}{r_{a,j} - r_{u,i}}\right)$ `// replenishment time`
      $l_{u,i} \leftarrow \min\left(c_{u,i},\ l_{u,i} + t_{r,j,i}\left(r_{a,j} - r_{u,i}\right)\right)$ `// user agent level`
      $l_{a,j} \leftarrow \max\left(0,\ l_{a,j} - t_{r,j,i} r_{a,j}\right)$ `// replenishment agent level`
      $t_{f,i} \leftarrow t_{b,j,i} + t_{r,j,i}$ `// update last replenishment time`
      $t_{l,j} \leftarrow t_{f,i} + t_{pa,j}$ `// time after packup`
    remove the task $\boldsymbol{\theta}_{j,1}$ from the schedule, $\boldsymbol{\theta}$
  $t = \min\left(t_{l,j}\ \forall j\right)$ `// schedule end time`
  forall the *user agents,* i **do**
    $t_{d,i} \leftarrow t_{f,i} + \dfrac{l_{u,i}}{r_{u,i}}$ `// deadline`
    $t_{c,i} \leftarrow \max\left(0,\ t - t_{d,i}\right)$ `// downtime`
    $\lambda \leftarrow \lambda + w_i t_{c,i}$ `// weighted downtime`
$\lambda \leftarrow \dfrac{\lambda}{n_{samples}}$ `// average downtime`
$\psi \leftarrow$ average of the predicted state of the system

The time that a replenishment agent arrives at its next task, $t_{a,j}$, is given on line 5. Here, $s = s_{ar,j}$ if the task is to be replenished by the replenishment point, and $s = s_{au,j,i}$ if the task is to replenish a user agent. $t_{l,j}$ is the time that the replenishment agent left its last task, and $v_{a,j}$ is sampled from the probability distribution for $V_{a,j}$.

If the task of the replenishment agent is to be replenished by the replenishment point (line 7), then the time that the task is completed at, $t_{f,j}$, is given on line 8 where t_{sr}, r_r, and t_{pr} are sampled from their respective distributions. Note that, if this is the first task in the schedule for replenishment agent j, $l_{a,j}$ is sampled from the belief of the current resource level, $L_{a,j}$, generated using the estimation methods in Chapter 4. The resource level of the replenishment agent after this task is simply $c_{a,j}$ (line 9).

If the replenishment agent arrives at the user agent while one or more replenishment agents are replenishing or waiting to replenish the user agent, then the replenishment agent must wait until these agents have finished replenishing the user agent before commencing setting up (line 14). The time at which the replenishment agent begins replenishing the user agent is given on line 15, where $t_{sa,j}$ is sampled from the parameter distribution. The time at which the user agent exhausts its supply of the resource is given on line 16, where the resource level, $l_{u,i}$, is sampled from the belief of the current resource level, and the resource usage rate, $r_{u,i}$, is sampled from the parameter distribution. $t_{f,i}$ is the time at which the last replenishment of user agent i finished. If the user agent has not been visited by a replenishment agent yet in the schedule, then $t_{f,i} = 0$.

If the user agent exhausts its supply of the resource before the replenishment agent begins to replenish it, the user agent will cease operation and will incur downtime (line 17). The resource level of the user agent before the replenishment begins is calculated on line 19. The time taken to replenish the user agent, $t_{r,j,i}$, is then given on line 20, where $r_{a,j}$ is sampled from the parameter distribution. Note that the time taken to replenish the user agent depends on whether the replenishment agent has a sufficient supply of the resource to fully replenish the user agent. In addition, the user agent is assumed to continue using the resource while being replenished. The new levels of the user agent and replenishment agent after the replenishment are presented

on line 21 and line 22 respectively.

The time that the user agent was last replenished is then updated on line 23, and the time that the replenishment agent finishes packing up is calculated on line 24. Finally, any downtime incurred to the end of the schedule is calculated on line 29 by using the completion time of the replenishment agent that finishes its schedule first.

5.2 Single-Agent Analytical Method

This section presents a novel analytical method for calculating the future state of single-replenishment agent SCAR scenarios. This prediction method is outlined in Algorithm 5.2. The major difference between this approach and the MC approach detailed in Algorithm 5.1 is that, instead of sampling the initial belief and parameter values, this method uses the entire probability distribution for the initial belief and for each parameter. Note that the presented method does not assume a specific type of probability distribution, and any distribution can be used provided methods exist for performing the necessary operations. Section 5.2.1 introduces approximations that enable Gaussian distributions to be used to represent the beliefs and parameters.

The distribution for the time that a replenishment agent arrives at its next task, $T_{a,j}$, is given on line 3. If the task of the replenishment agent is to by replenished by the replenishment point, then $s = s_{ar,j}$, otherwise $s = s_{au,j,i}$. The distance to the replenishment point, $s_{ar,j}$, and the distance to the operational area of the user agent, $s_{au,j,i}$, are scalar values here as the replenishment agent is leaving its previous task and the distances between these points are assumed to be known to a high degree of certainty. If the replenishment agent was partway along a road and had poor sensing such that its position was not known to a high degree of certainty, probability distributions of $S_{ar,j}$ and $S_{au,j,i}$ respectively would be used for these distance values.

If the task of the replenishment agent is to be replenished by the replenishment point (line 4), then the distribution for the time that that task is completed at, $T_{f,j}$, is given on line 5, where $L_{a,j}$ is the belief of the resource level. The resource level of the

Algorithm 5.2: Analytical method

`Analytical(`$\boldsymbol{\psi}_{initial}, \boldsymbol{\theta}, n_{samples}$`)`
input : Current system state, $\boldsymbol{\psi}_{initial}$; schedule, $\boldsymbol{\theta}$
output: Weighted downtime, λ; state, $\boldsymbol{\psi}$

$\lambda \leftarrow 0$, $j \leftarrow 1$
while *there are tasks remaining in the schedule,* $\boldsymbol{\theta}$ **do**
  $T_{a,j} \leftarrow T_{l,j} + \dfrac{s}{V_{a,j}}$ `// calculate arrival time`
  if *the task is to by replenished by the replenishment point* **then**
    $T_{f,j} \leftarrow T_{a,j} + T_{sr} + \dfrac{c_{a,j} - L_{a,j}}{R_r} + T_{pr}$ `// replenishment finish time`
    $L_{a,j} \leftarrow c_{a,j}$
  else
    $i \leftarrow$ index of user agent that replenishment agent j is replenishing
    $T_{b,j,i} \leftarrow T_{a,j} + T_{sa,j}$ `// time after setup`
    $T_{d,i} \leftarrow T_{f,i} + \dfrac{L_{u,i}}{R_{u,i}}$ `// deadline`
    $T_{c,j,i} \leftarrow T_{b,j,i} - T_{d,i}$ `// downtime distribution`
    $E(\text{downtime}) \leftarrow \int_0^\infty t T_{c,j,i} \, \mathrm{d}t$ `// expected downtime`
    $\lambda \leftarrow \lambda + w_i E(\text{downtime})$ `// weighted downtime`
    $L_{u,i} \leftarrow \left(L_{u,i} - \left(T_{b,j,i} - T_{f,i}\right) R_{u,i}\right)^{\#}$ `// level before replenishment`
    $Q_{u,i,j} \leftarrow \left(c_{u,i} - L_{u,i}\right) \dfrac{R_{a,j}}{R_{a,j} - R_{u,i}}$ `// replenishment quantity`
    $Q^*_{u,i,j} \leftarrow \left(Q_{u,i,j}\right)^{*\leq L_{a,j}}$ `// adjusted quantity`
    $T_{r,j,i} \leftarrow \dfrac{Q^*_{u,i,j}}{R_{a,j}}$ `// replenishment time`
    $L_{u,i} \leftarrow \left(L_{u,i} - \left(T_{b,j,i} - T_{f,i}\right) R_{u,i}\right)^{\#}$ `// user agent level`
    $L_{a,j} \leftarrow \left(L_{a,j} - Q_{u,i,j}\right)^{\#}$ `// replenishment agent level`
    $T_{f,i} \leftarrow T_{b,j,i} + T_{r,j,i}$ `// update last replenishment time`
    $T_{l,j} \leftarrow T_{f,i} + T_{pa,j}$ `// time after packup`
  remove the task $\boldsymbol{\theta}_{j,1}$ from the schedule, $\boldsymbol{\theta}$
forall the *user agents,* i **do**
  $T_{d,i} \leftarrow T_{f,i} + \dfrac{L_{u,i}}{R_{u,i}}$ `// deadline`
  $T_{c,i} \leftarrow T_{l,j} - T_{d,i}$ `// downtime distribution`
  $E(\text{downtime}) \leftarrow \int_0^\infty t T_{c,i} \, \mathrm{d}t$ `// expected downtime`
  $\lambda \leftarrow \lambda + w_i E(\text{downtime})$ `// weighted downtime`
$\psi \leftarrow \left(L_{a,j}, L_{u,i} \; \forall i\right)$ `// predicted final belief state`

replenishment agent after this task is then given by $c_{a,j}$ (line 6).

If the task is instead to travel to and replenish a user agent, the distribution for the time at which the replenishment agent begins replenishing the user agent is shown on line 9. The deadline for the user agent is given on line 10. If it has not been visited yet by a replenishment agent, then $T_{f,i} = 0$. The distribution for the downtime incurred by the user agent, $T_{c,j,i}$, is presented on line 11. Only values in the positive domain contribute to the downtime, and values below zero should be ignored. This distribution can be used in a chance-constrained formulation by calculating the probability that downtime will be incurred as:

$$P(\text{downtime}) = \int_0^\infty T_{c,j,i} \mathrm{d}t \tag{5.1}$$

This thesis uses the expected downtime so that the total expected downtime of the system can be minimised, where the expected downtime is calculated using the integral on line 12 and E is the expectation operator.

The level of user agent i at time $T_{b,j,i}$ is given on line 14, where the # operator denotes that the distribution has been adjusted to account for hard constraints on the state of the system. Note that this adjustment is only required when using distributions that do not already fit within the hard constraints. A method for adjusting Gaussian distributions against hard constraints is discussed in Section 5.2.1. The quantity of the resource required to fully replenish the user agent, $Q_{u,i,j}$, is given on lines 15 and 16. Here, $* \leq L_{a,j}$ means that $Q_{u,i,j}$ is adjusted so that it does not exceed the resource level of the replenishment agent, $L_{a,j}$, as the replenishment agent may have insufficient supply of the resource to fully replenish the user agent. A method for performing this soft adjustment for Gaussian distributions is outlined in Section 5.2.1.

The time taken to replenish the user agent is then given on line 17. The new levels of the user agent and replenishment agent after the replenishment are given by line 18 and line 19 respectively. Note that the unadjusted $Q_{u,i,j}$ is used on line 19 as using $Q^*_{u,i,j}$ can underestimate the amount of the resource transferred by the replenishment

agent. The # adjustment ensures that $L_{a,j}$ remains primarily in the positive domain. The time that the user agent was last replenished is updated on line 20, and the time that the replenishment agent finishes packing up is given on line 21. Finally, any additional expected downtime at the end of a schedule is calculated for each user agent on line 26.

5.2.1 Gaussian Approximations

From this point forward, this thesis assumes that the stochastic parameters can be described by independent Gaussian distributions. Gaussian distributions were chosen as the addition and subtraction of Gaussian distributed variables results in a Gaussian distributed variable. Other operations such as division, multiplication, and inversion, however, result in non-Gaussian distributions. To facilitate the propagation of the distributions in the analytical prediction method, it is convenient to approximate the resulting non-Gaussian distributions as Gaussian distributions. The following subsections discuss these approximation methods.

It should be noted that the probability distributions for quantities such as the time when the user agent was last replenished and the time taken to replenish the user agent are not independent and will have some level of covariance. In this thesis the covariance between all calculated distributions has been assumed to be zero. The quality of the results presented later in Section 5.4 demonstrate that these approximations and assumptions are acceptable.

Inverse Gaussian Distribution

A Gaussian approximation of the probability distribution resulting from the inverse of a Gaussian distributed variable was not found in the examined literature, so the following approximation was formulated. Consider:

$$I = \frac{c}{G} \tag{5.2}$$

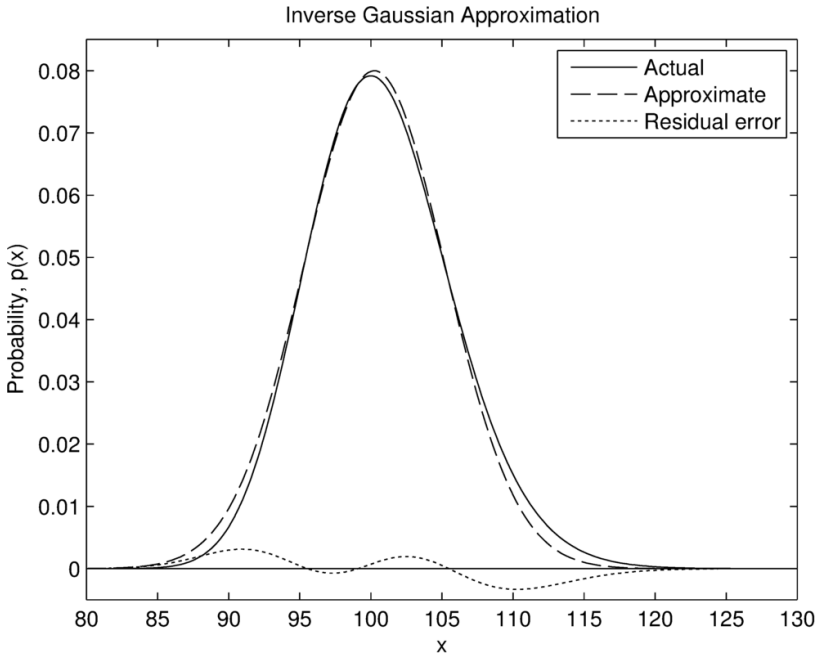
where I is an inverse Gaussian distributed variable, G is a Gaussian distributed variable described by mean, μ_G, and standard deviation, σ_G, and c is a constant. If the resultant inverse Gaussian distribution is assumed to be a Gaussian distribution, the points on I for $\mu_G - \sigma_G$ and $\mu_G + \sigma_G$ will be positioned at points $\mu_I + \sigma_I$ and $\mu_I - \sigma_I$ respectively. Using this assumption yields the following approximations for the mean and standard deviation:

$$\begin{aligned}\mu_I &= \frac{1}{2}\left((\mu_I + \sigma_I) + (\mu_I - \sigma_I)\right) \\ &= \frac{1}{2}\left(\frac{c}{\mu_G - \sigma_G} + \frac{c}{\mu_G + \sigma_G}\right) \\ &= \frac{c\mu_G}{\mu_G^2 - \sigma_G^2}\end{aligned} \tag{5.3}$$

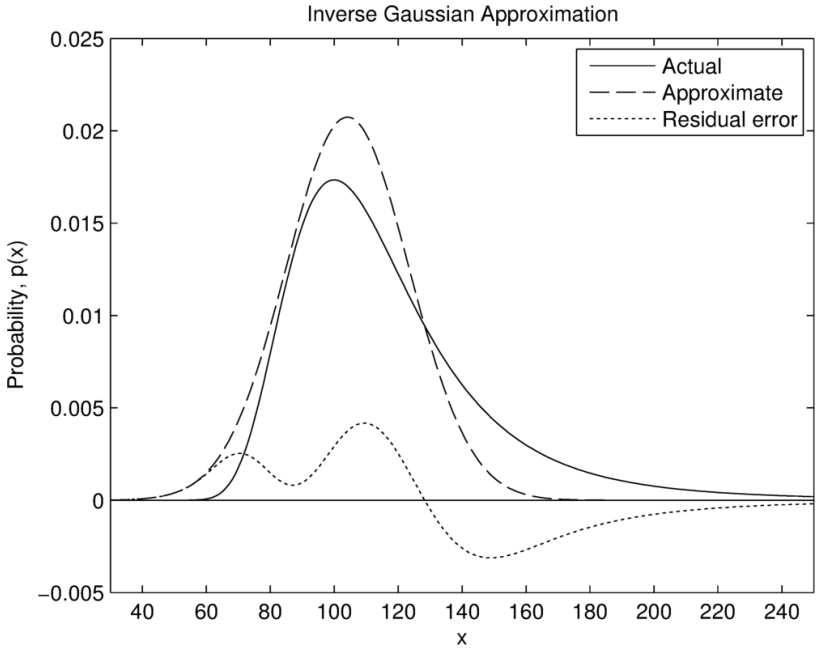
$$\begin{aligned}\sigma_I &= \frac{1}{2}\left((\mu_I + \sigma_I) - (\mu_I - \sigma_I)\right) \\ &= \frac{1}{2}\left(\frac{c}{\mu_G - \sigma_G} - \frac{c}{\mu_G + \sigma_G}\right) \\ &= \frac{c\sigma_G}{\mu_G^2 - \sigma_G^2}\end{aligned} \tag{5.4}$$

Figure 5.2 shows the inverse Gaussian distribution, the resultant approximation and the residual error between the two distributions for two μ_G/σ_G ratios. In Figure 5.2a, the approximation is very good, matching the shape of the inverse Gaussian distribution closely. Using a smaller μ_G/σ_G ratio in Figure 5.2b, the resultant inverse Gaussian distribution is skewed to the right. The approximation cannot account for the skewness and suffers as a result.

The KL divergence as a function of the μ_G/σ_G ratio is shown in Figure 5.3. As can be seen, the approximation improves as the μ_G/σ_G ratio increases. As $\mu_G/\sigma_G \to 0$, large portions of the distribution of G are in the negative domain. However, when G represents a physical quantity such as velocity, it does not make sense for this



(a) $\mu_G/\sigma_G = 20$



(b) $\mu_G/\sigma_G = 5$

Figure 5.2 – The approximation for the inverse Gaussian distribution. (a) shows an example of the approximation for $\mu_G/\sigma_G = 20$, and (b) shows an example of the approximation for $\mu_G/\sigma_G = 5$.

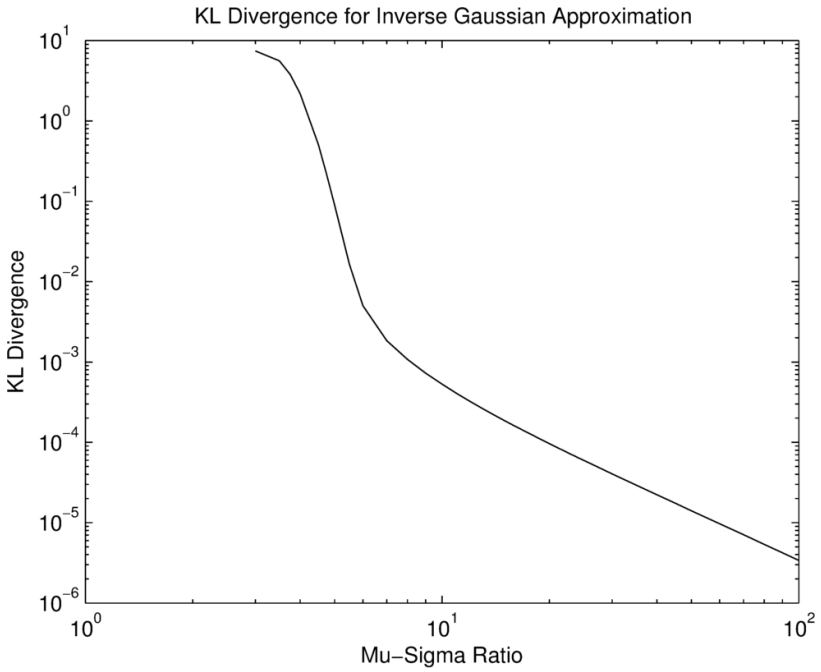


Figure 5.3 – KL divergence for the inverse Gaussian approximation

quantity to be negative as, in the case of velocity, it implies that the agent would be travelling away from its destination. Thus, the usefulness of this approximation is limited to situations where the majority of G is in the positive domain. This is likely to be the case for most practical scenarios. For scenarios where this is not the case, a Gaussian distribution may not be the best representation of the physical quantity under consideration.

Gaussian Ratio Distribution

A method for approximating a ratio of Gaussian distributed variables is given in [68]. A ratio distribution, R:

$$R = \frac{E}{F} \tag{5.5}$$

with distributions $E \sim \mathcal{N}(\mu_E, \sigma_E)$ and $F \sim \mathcal{N}(\mu_F, \sigma_F)$, and correlation between E and F of $\rho = 0$, can be approximated with a Gaussian distribution where:

$$r = \frac{\sigma_F}{\sigma_E} \tag{5.6}$$

$$a = \frac{\mu_E}{\sigma_E} \tag{5.7}$$

$$b = \frac{\mu_F}{\sigma_F} \tag{5.8}$$

$$\mu_R = \frac{a}{r(1.01b - 0.2713)} \tag{5.9}$$

$$\sigma_R = \frac{1}{r}\sqrt{\frac{a^2+1}{b^2+0.108b-3.795} - r^2\mu_R^2} \tag{5.10}$$

This approximation is only valid for $a < 2.5$, $b > 4$ [68]. As $a \to \infty$, the ratio distribution approaches the inverse Gaussian distribution and can be approximated using the inverse Gaussian approximation presented above. For situations where $a \geq 2.5$, this thesis uses the inverse Gaussian approximation method with E treated as a scalar, $e = \mu_E$.

Gaussian Product Distribution

An approximation to the product of two Gaussian distributed variables is presented in [112]. For a Gaussian product distribution, M:

$$M = EF \tag{5.11}$$

Then:

$$\mu_M = \mu_E \mu_F \tag{5.12}$$

$$\sigma_M^2 = \sigma_E^2 \sigma_F^2 (1 + \delta_E^2 + \delta_F^2) \tag{5.13}$$

where

$$\delta_x = \frac{\mu_x}{\sigma_x} \tag{5.14}$$

Ware and Lad [112] note that the approximation improves as δ_E and δ_F become large.

Expected Value

The expected downtime for a user agent is calculated using the integral on line 12 of Algorithm 5.2. If the distribution for the downtime is a Gaussian distribution with μ and σ, this becomes:

$$E(G) = \int_0^\infty \frac{x}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \, \mathrm{d}x \tag{5.15}$$

Solving the definite integral gives:

$$E(G) = \frac{\mu}{2}(1 + \mathrm{erf}(\frac{\mu}{\sigma\sqrt{2}})) + \frac{\sigma}{\sqrt{2\pi}} \exp\left(-\frac{\mu^2}{2\sigma^2}\right) \tag{5.16}$$

Adjusting the Distribution against Hard Constraints

Several distributions in the analytical framework presented in this section were required to be adjusted to take into account hard limitations on the state. Line 14 in Algorithm 5.2, for example, specifies the resource level of the user agent before it is replenished. The minimum value that this resource level can take is 0, and the maximum value is $c_{u,i}$. However, when using Gaussian distributions, which have a domain of $(-\infty, \infty)$, to represent these distributions, some of the probability distribution lies outside of the physical limits of system. Where distributions have been adjusted to

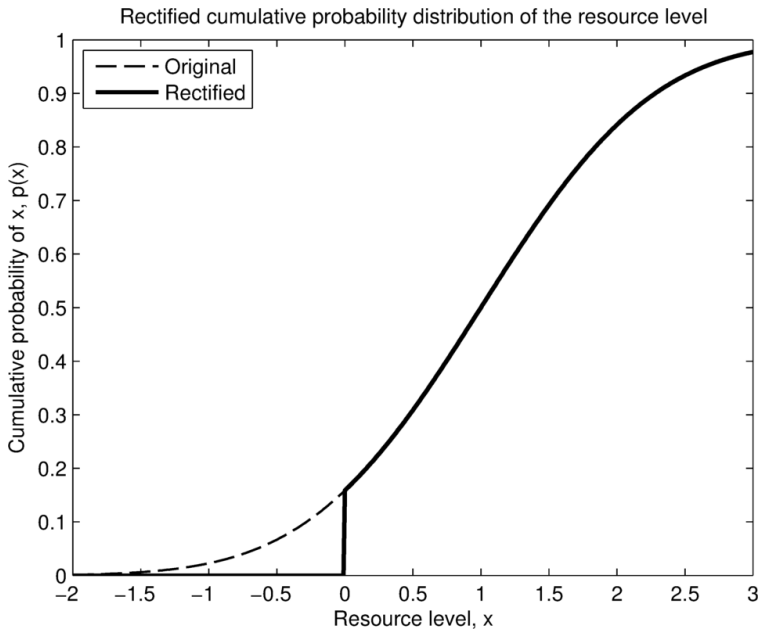
account for hard limits, denoted by a #, a similar approach to the truncation method for the KF presented in Section 4.1.2 has been used. Unlike in the truncation method for the constrained KF, the parts of the probability distribution that lie outside of the physical limits of the system are not discarded, and are instead grouped together as a discrete probability that the system is at the physical limit. This is similar to the rectified Gaussian distribution introduced by Socci et al. [95] and used by Meng et al. [71]. A rectified Gaussian distribution restricts the domain to be non-negative, grouping any probability in the negative domain at 0. The generalised rectified Gaussian distribution is introduced here, where the domain is restricted to be between two arbitrary points, with the probability outside of those points grouped at the respective point.

Figure 5.4 illustrates the rectification concept. In this example, the probability distribution describing the predicted future resource level extends into the negative domain. It is appropriate to group the probability in the negative domain at zero as it represents the probability that the agent has used up its supply of the resource. If the truncation approach were used instead, the resultant probability distribution would overestimate the resource level as it ignores the probability that the resource supply has been exhausted. In Figure 5.4a, the jump in the CDF of the rectified distribution at $x = 0$ represents the discrete probability that the resource supply is empty. In Figure 5.4b, this discrete probability is represented by a Dirac delta function.

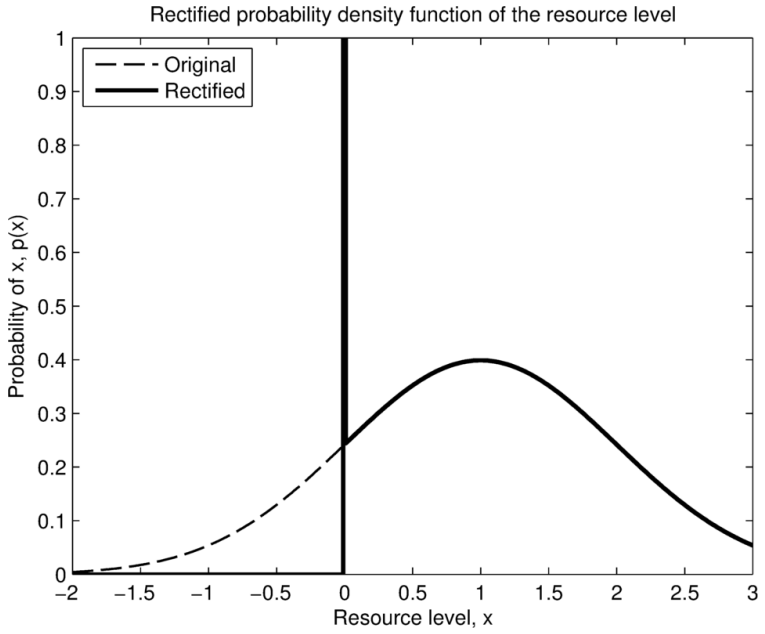
If the original CDF is $F(x)$, the problem is to calculate a new Gaussian PDF, $\mathcal{N}(\mu_R, \sigma_R^2)$, that approximates the PDF of the generalised rectified Gaussian distribution that satisfies the following CDF, $F_R(x)$:

$$F_R(x) = \begin{cases} 0 & \text{if } x < a \\ F(x) & \text{if } a \leq x < b \\ 1 & \text{if } x \geq b \end{cases} \tag{5.17}$$

where a and b are the limits on the state. Following the transformation process from Section 4.1.2 for a one-dimensional Gaussian distribution, the distribution being



(a) CDF of the rectified Gaussian distribution



(b) PDF of the rectified Gaussian distribution

Figure 5.4 – Rectifying the resource level at $x = 0$. The CDF and PDF of the rectified Gaussian distribution are shown in (a) and (b) respectively. The jump in the rectified CDF in (a) when $x = 0$ corresponds to the probability that $x = 0$. The spike in the rectified PDF in (b) at $x = 0$ is a Dirac delta function and corresponds to the probability of $x \leq 0$ in the original distribution.

adjusted is transformed to a standard normal distribution, and the constraints are transformed to c and d respectively:

$$c = \frac{a - \mu_A}{\sigma_A} \tag{5.18}$$

$$d = \frac{b - \mu_A}{\sigma_A} \tag{5.19}$$

where μ_A and σ_A are the mean and standard deviation of the distribution being adjusted. The mean and variance of the Gaussian approximation of the rectified Gaussian distribution are then given by:

$$\begin{aligned}\mu_z &= \int_c^d \frac{\zeta}{\sqrt{2\pi}} \exp\left(-\frac{\zeta^2}{2}\right) \mathrm{d}\zeta + \frac{c}{2}\left(1 + \operatorname{erf}\left(\frac{c}{\sqrt{2}}\right)\right) + \frac{d}{2}\left(1 - \operatorname{erf}\left(\frac{d}{\sqrt{2}}\right)\right) \\ &= \left(\frac{1}{\sqrt{2\pi}}\right)\left(\exp\left(\frac{-c^2}{2}\right) - \exp\left(\frac{-d^2}{2}\right)\right) + \frac{c}{2}\left(1 + \operatorname{erf}\left(\frac{c}{\sqrt{2}}\right)\right) + \frac{d}{2}\left(1 - \operatorname{erf}\left(\frac{d}{\sqrt{2}}\right)\right)\end{aligned} \tag{5.20}$$

$$\begin{aligned}\sigma_z^2 &= \int_c^d (\zeta - \mu)^2 \exp\left(-\frac{\zeta^2}{2}\right) \mathrm{d}\zeta + \frac{(c-\mu)^2}{2}\left(1 + \operatorname{erf}\left(\frac{c}{\sqrt{2}}\right)\right) \\ &\quad + \frac{(d-\mu)^2}{2}\left(1 - \operatorname{erf}\left(\frac{d}{\sqrt{2}}\right)\right) \\ &= \frac{\mu^2 + 1}{2}\left(\operatorname{erf}\left(\frac{d}{\sqrt{2}}\right) - \operatorname{erf}\left(\frac{c}{\sqrt{2}}\right)\right) \\ &\quad - \frac{1}{\sqrt{2\pi}}\left(\exp\left(-\frac{d^2}{2}\right)(d - 2\mu) - \exp\left(-\frac{c^2}{2}\right)(c - 2\mu)\right) \\ &\quad + \frac{(c-\mu)^2}{2}\left(1 + \operatorname{erf}\left(\frac{c}{\sqrt{2}}\right)\right) + \frac{(d-\mu)^2}{2}\left(1 - \operatorname{erf}\left(\frac{d}{\sqrt{2}}\right)\right)\end{aligned} \tag{5.21}$$

Taking the inverse of the transformation gives:

$$\mu_R = \mu_z \sigma_A + \mu_A \tag{5.22}$$

$$\sigma_R^2 = \sigma_z^2 \sigma_A^2 \tag{5.23}$$

Adjusting the Distribution against Soft Constraints

The other type of adjustment used adjusts one distribution so that it either does not exceed, or is not exceed by, another distribution. This soft adjustment is denoted by a * followed by the distribution that it is adjusted against. This is used on line 16 of Algorithm 5.2 to ensure that the quantity of the resource used to replenish a user agent does not exceed the current capacity of the replenishment agent. Consider a distribution, A, that is adjusted so that it does not exceed the distribution B. The proposed method is as follows:

$$\mu_A^{*\leq B} = \begin{cases} \mu_A & \text{if } \mu_A - 3\sigma_A < \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A < \mu_B + 3\sigma_B \\ \mu_B & \text{if } \mu_A - 3\sigma_A > \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A > \mu_B + 3\sigma_B \\ \frac{\mu_B - 3\sigma_B + \mu_A + 3\sigma_A}{2} & \text{if } \mu_A - 3\sigma_A > \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A < \mu_B + 3\sigma_B \\ \frac{\mu_A - 3\sigma_A + \mu_B + 3\sigma_B}{2} & \text{if } \mu_A - 3\sigma_A < \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A > \mu_B + 3\sigma_B \end{cases} \tag{5.24}$$

$$\sigma_A^{*\leq B} = \begin{cases} \sigma_A & \text{if } \mu_A - 3\sigma_A < \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A < \mu_B + 3\sigma_B \\ \sigma_B & \text{if } \mu_A - 3\sigma_A > \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A > \mu_B + 3\sigma_B \\ \frac{\mu_A + 3\sigma_A - (\mu_B - 3\sigma_B)}{6} & \text{if } \mu_A - 3\sigma_A > \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A < \mu_B + 3\sigma_B \\ \frac{\mu_B + 3\sigma_B - (\mu_A - 3\sigma_A)}{6} & \text{if } \mu_A - 3\sigma_A < \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A > \mu_B + 3\sigma_B \end{cases} \tag{5.25}$$

This method ensures that $P(A \leq x) \leq P(B \leq x)$ for x within 3 standard deviations of the mean of both A and B. Examples of the four cases are given in Figure 5.5. In Figure 5.5a, the tails of the upper constraint (where the tails are defined as $\mu \pm 3\sigma$) are both higher than those of the original distribution, resulting in an adjusted distribution that is the same as the original distribution. In Figure 5.5b, the tails of the upper constraint are consistently below those of the original distribution. The adjusted distribution in this case is therefore given by the distribution of the upper

constraint. In Figure 5.5c, the original distribution is above the upper constraint in the left tail and below the upper constraint in the right tail. In this case the adjusted distribution is calculated by using the left tail of the constraint distribution and the right tail of the original distribution. Finally, in Figure 5.5d, the original distribution is below the upper constraint in the left tail, but above the upper constraint in the right tail. The adjusted distribution is calculated using the left tail of the original distribution and the right tail of the constraint distribution.

For adjusting distribution A such that it is not lower than distribution B, the mean and standard deviation are calculated as follows:

$$\mu_A^{*\geq B} = \begin{cases} \mu_A & \text{if } \mu_A - 3\sigma_A > \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A > \mu_B + 3\sigma_B \\ \mu_B & \text{if } \mu_A - 3\sigma_A < \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A < \mu_B + 3\sigma_B \\ \frac{\mu_B - 3\sigma_B + \mu_A + 3\sigma_A}{2} & \text{if } \mu_A - 3\sigma_A < \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A > \mu_B + 3\sigma_B \\ \frac{\mu_A - 3\sigma_A + \mu_B + 3\sigma_B}{2} & \text{if } \mu_A - 3\sigma_A > \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A < \mu_B + 3\sigma_B \end{cases} \tag{5.26}$$

$$\sigma_A^{*\geq B} = \begin{cases} \sigma_A & \text{if } \mu_A - 3\sigma_A > \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A > \mu_B + 3\sigma_B \\ \sigma_B & \text{if } \mu_A - 3\sigma_A < \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A < \mu_B + 3\sigma_B \\ \frac{\mu_A + 3\sigma_A - (\mu_B - 3\sigma_B)}{6} & \text{if } \mu_A - 3\sigma_A < \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A > \mu_B + 3\sigma_B \\ \frac{\mu_B + 3\sigma_B - (\mu_A - 3\sigma_A)}{6} & \text{if } \mu_A - 3\sigma_A > \mu_B - 3\sigma_B \text{ and } \mu_A + 3\sigma_A < \mu_B + 3\sigma_B \end{cases} \tag{5.27}$$

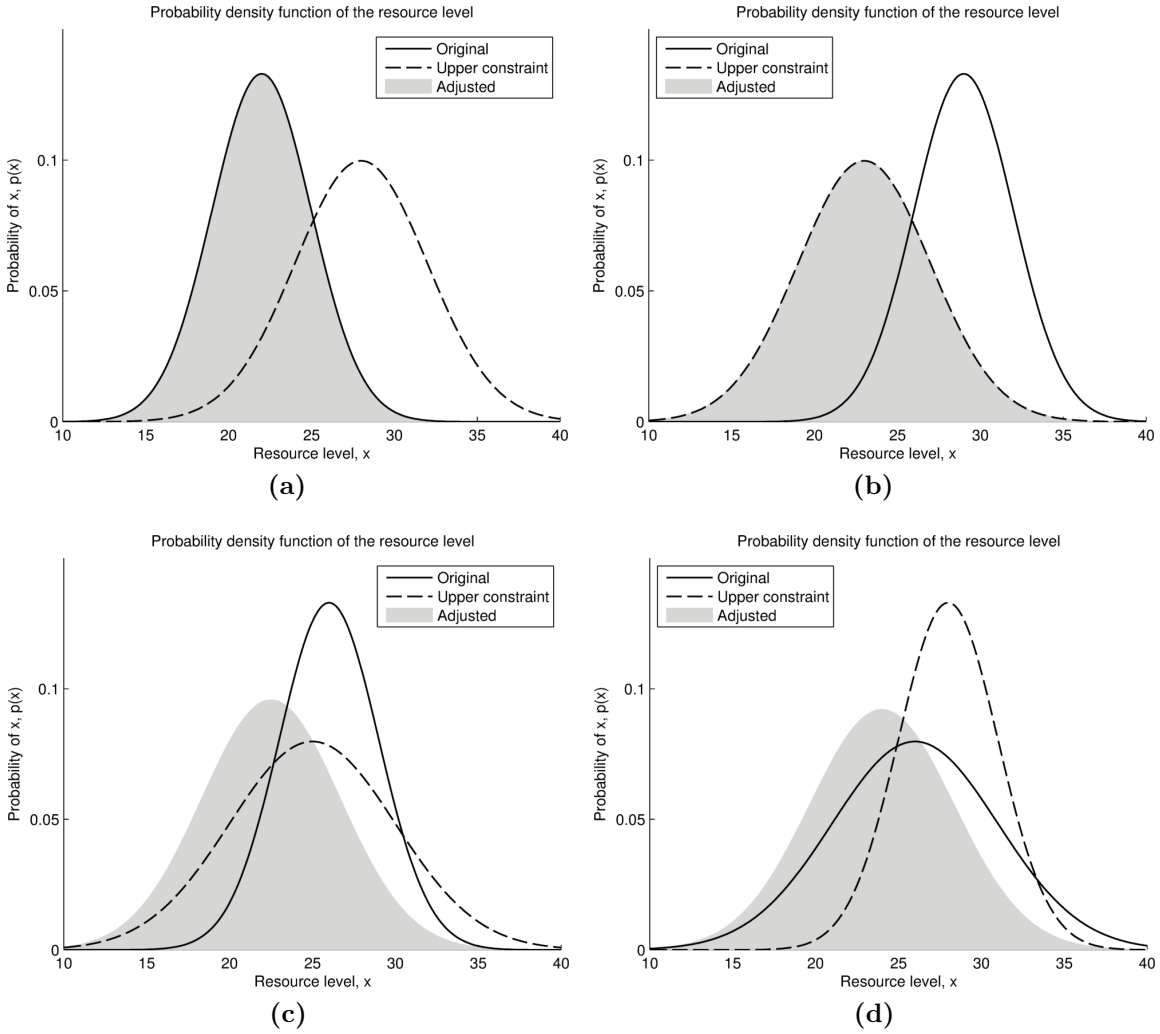


Figure 5.5 – Adjusting the resource level using an upper constraint distribution. In (a), the original distribution is fully below the constraint in the range $\mu_A \pm 3\sigma_A$, so the adjusted distribution is equal to the original distribution. In (b), the adjusted distribution is equal to the upper constraint as the original distribution is fully above the constraint. In (c), the left tail of the constraint distribution and the right tail of the original distribution are used to calculate the adjusted distribution. In (d), the left tail of the original distribution and the right tail of the constraint distribution are used to calculate the adjusted distribution.

5.3 Multi-Agent Analytical Methods

Extending the single-replenishment agent framework presented in Section 5.2 to multi-replenishment agent scenarios requires consideration of the order in which each user agent is visited by the replenishment agents. For any given schedule, there may be multiple possible orders in which the replenishment agents can arrive at each user agent, which each have different probabilities of occurring and different expected costs. The proposed method calculates the expected cost of a schedule by summing the expected cost of each order of arrival multiplied by the probability of that order occurring.

5.3.1 Calculating the Probability of an Arrival Order

The probability of a given order of arrival can be calculated by examining the probability distributions for the arrival times of the replenishment agents at each user agent. Consider probability distributions for the arrival times of two replenishment agents, j and k, at a particular user agent of $T_{a,j}$ and $T_{a,k}$ respectively. The probability that replenishment agent j arrives before replenishment agent k is given by:

$$P(T_{a,j} < T_{a,k}) = \int_{-\infty}^{0} p\left(T_{a,j} - T_{a,k}\right) \mathrm{d}t \tag{5.28}$$

where t is time. Extending this to three replenishment agents with arrival time probabilities of $T_{a,j}$, $T_{a,k}$ and $T_{a,l}$, the probability that j arrives before k, and k arrives before l, is given by:

$$P(T_{a,j} < T_{a,k} \cap T_{a,k} < T_{a,l}) = P(T_{a,j} < T_{a,k}) \times P(T_{a,k} < T_{a,l} | T_{a,j} < T_{a,k}) \tag{5.29}$$

This is equivalent to finding the CDF of the multivariate Gaussian distribution of $T_{a,j} - T_{a,k}$ and $T_{a,k} - T_{a,l}$. The mean and covariance matrices of the resultant multi-

variate Gaussian distribution, $\boldsymbol{\mu}_D$ and $\boldsymbol{\Sigma}_D$, are calculated through an affine transformation as follows.

Let $D_1 = T_{a,j} - T_{a,k}$ and $D_2 = T_{a,k} - T_{a,l}$, where $T_{a,j}$, $T_{a,k}$ and $T_{a,l}$ have means and standard deviations of $\mu_{a,j}$, $\mu_{a,k}$, $\mu_{a,l}$, and $\sigma_{a,j}$, $\sigma_{a,k}$, $\sigma_{a,l}$ respectively:

$$\boldsymbol{D} = \begin{bmatrix} D_1 \\ D_2 \end{bmatrix} = \boldsymbol{S}\boldsymbol{T} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} T_{a,j} \\ T_{a,k} \\ T_{a,l} \end{bmatrix} \tag{5.30}$$

with

$$\boldsymbol{\mu}_A = \begin{bmatrix} \mu_{a,j} \\ \mu_{a,k} \\ \mu_{a,l} \end{bmatrix} \qquad \boldsymbol{\Sigma}_A = \begin{bmatrix} \sigma_{a,j}^2 & 0 & 0 \\ 0 & \sigma_{a,k}^2 & 0 \\ 0 & 0 & \sigma_{a,l}^2 \end{bmatrix} \tag{5.31}$$

The matrices after the affine transformation has been applied are:

$$\boldsymbol{\mu}_D = \boldsymbol{S}\boldsymbol{\mu}_A \tag{5.32}$$

and

$$\boldsymbol{\Sigma}_D = \boldsymbol{S}\boldsymbol{\Sigma}_A\boldsymbol{S}^T \tag{5.33}$$

The probability can then be calculated by evaluating the CDF of the multi-variate Gaussian distribution with $\boldsymbol{\mu}_D$ and $\boldsymbol{\Sigma}_D$ for $D_1 \leq 0$ and $D_2 \leq 0$. No analytical solution exists for the CDF of a multi-variate Gaussian distribution [37]. However, approximate methods are readily available in Matlab and Python. Using this framework, the cost calculation method can be extended to many replenishment agents. The multi-variate CDF function, `MVNCDF`, is computationally expensive, especially for high-dimensional multi-variate distributions. Algorithm 5.3 reduces the computational requirement by calculating the probability for the order of arrival at each user

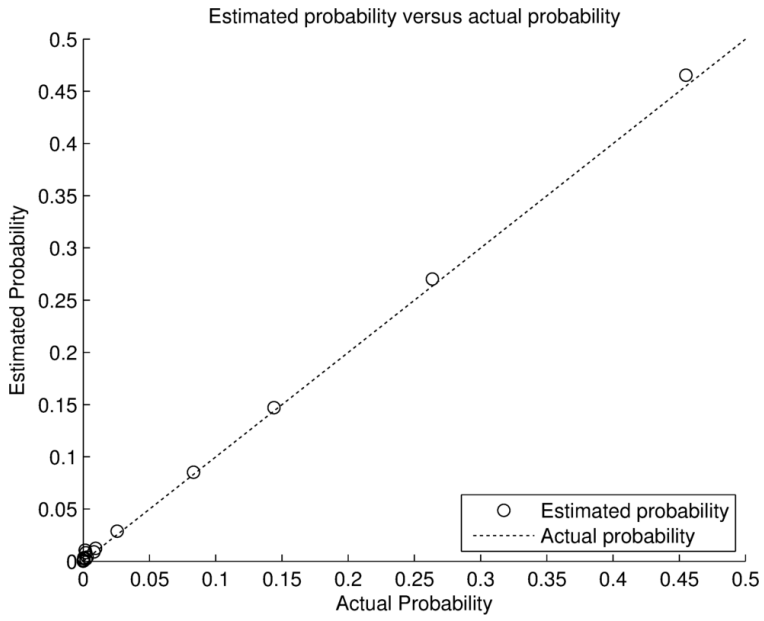
Algorithm 5.3: Calculate probability of arrival order

`CalcProb(`$\boldsymbol{T}$, n`)`
input : List of lists of arrival times ordered by user agent, $\boldsymbol{T}$; number of user agents, n
output: Probability, P

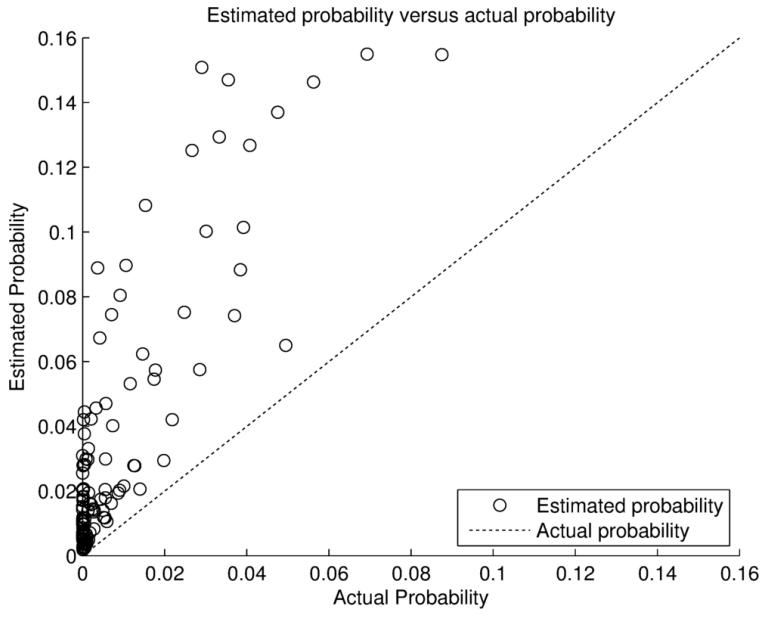
$P \leftarrow 1$
for $i \in \{1, ..., n\}$ **do**
  $\boldsymbol{\mu}, \boldsymbol{\Sigma} \leftarrow$ affine transformation of $\boldsymbol{T}_i$
  $P \leftarrow P \times$`MVNCDF(`$\boldsymbol{\mu}, \boldsymbol{\Sigma}$`)` `// multi-variate CDF for the arrival times at user agent` i

agent independently. However, this process can still be quite slow, taking several milliseconds to compute in some scenarios. Other methods to reduce the dimensionality, such as discarding arrival times which are isolated, may produce further increases in speed.

It is also convenient to be able to estimate what the probability of an order of arrival is without using the full multi-variate CDF. Algorithm 5.4 outlines a method of estimating the probability using adjacent pairs of arrival times and Equation 5.28. This method is guaranteed not to underestimate the probability of an order of arrival as it ignores any conditional probabilities. While the results are not accurate compared to the probability calculated from Algorithm 5.3, it is significantly faster and is useful for identifying low probability orders of arrival. Figure 5.6 shows the quality of the estimates for two groups of arrival times. In Figure 5.6a, the estimated probability is an almost perfect approximation of the actual probability as any arrival times that are close to each other only occur in pairs. This indicates that for scenarios with only 2 replenishment agents, `EstimateProb` is a very good approximation of the actual probability. In Figure 5.6b, the distribution for each arrival time overlaps with multiple other arrival times. While the estimated probability is significantly overestimated in most cases, low probability combinations are still easily identified.



(a)



(b)

Figure 5.6 – Comparing the estimated probability of a particular order of arrival with the actual probability. This is performed for all possible orders of arrival using arrival times of $[(50, 10), (60, 10), (100, 10), (105, 10), (130, 10)]$ in (a) and $[(80, 10), (90, 20), (100, 10), (110, 20), (120, 10)]$ in (b), where the first number of each pair is the mean of the arrival time distribution and the second number is the standard deviation. In (a) the arrival times are spread out, resulting in few likely arrival orders. The arrival time distributions are much closer to one another in (b). The dotted line has a slope of 1:1.

Algorithm 5.4: Estimate probability of arrival order

```
EstimateProb(T, n)
input  : List of lists of arrival times ordered by user agent, T; number of user
         agents, n
output: Probability, P
P ← 1
for i ∈ {1, ..., n} do
    for k ∈ {2, ...,Length(T_i)} do
        μ ← μ_{T_{i,k-1}} − μ_{T_{i,k}}
        σ² ← σ²_{T_{i,k-1}} + σ²_{T_{i,k}}
        P ← P×CDF(μ, σ²)     // CDF for the pair of arrival times
```

5.3.2 Conditioning the Arrival Times

The probability distributions for the arrival times that have been used above are the average arrival time distributions for all possible orders of arrival. The arrival time distributions for a particular order of arrival are conditioned on that order of arrival. Consider two replenishment agents j and k with arrival time distributions $T_{a,j}$ and $T_{a,k}$ respectively. The conditional probability distribution for the arrival time of agent j given that it arrives before agent k is:

$$p(t_{a,j}|T_{a,j} < T_{a,k}) = \frac{p\left(T_{a,j}\right) \times P(t_{a,j} < T_{a,k})}{P(T_{a,j} < T_{a,k})} \tag{5.34}$$

where

$$P(t_{a,j} < T_{a,k}) = \int_{t_{a,j}}^{\infty} p\left(T_{a,k}\right) \mathrm{d}t = 1 - \mathrm{CDF}_{T_{a,k}}\left(t_{a,j}\right) \tag{5.35}$$

$$P(T_{a,j} < T_{a,k}) = P(T_{a,k} - T_{a,j} > 0) = \int_{0}^{\infty} p(T_{a,k} - T_{a,j})\mathrm{d}t = 1 - \mathrm{CDF}_{T_{a,k}-T_{a,j}}\left(0\right) \tag{5.36}$$

Terms of the form $\mathrm{CDF}_x\left(y\right)$ are the CDF of x evaluated at y. If $T_{a,j}$ and $T_{a,k}$

have mean and standard deviations of μ_j, σ_j and μ_k, σ_k respectively, the conditional probability distribution is given by:

$$p(t_{a,j}|T_{a,j} < T_{a,k}) = \frac{1}{\sigma_j\sqrt{2\pi}} \exp\left(-\frac{(t_{a,j}-\mu_j)^2}{2\sigma_j^2}\right) \frac{1-\operatorname{erf}\left(\frac{t_{a,j}-\mu_k}{\sigma_k\sqrt{2}}\right)}{1+\operatorname{erf}\left(\frac{\mu_k-\mu_j}{\sqrt{2\left(\sigma_j^2+\sigma_k^2\right)}}\right)} \tag{5.37}$$

Approximating this probability distribution as a Gaussian distribution is equivalent to applying a one-sided soft constraint in Section 4.2.2, where the arrival time distribution for k is treated as an upper constraint on the arrival time distribution for j. Note that both distributions should be constrained—$T_{a,j}$ should be constrained using $T_{a,k}$ as the upper constraint, and $T_{a,k}$ should be constrained using $T_{a,j}$ as the lower constraint.

For three or more replenishment agents, the arrival time distributions are conditioned on multiple distributions. For example, if j arrives before k and k arrives before l, then the arrival time distribution for j is conditioned on the arrival time distributions for both k and l. For k, the interval constraint method from Section 4.2.3 can be applied using $T_{a,j}$ as the lower constraint and $T_{a,l}$ as the upper constraint. For j and l, the constraints applied are $T_{a,k}|T_{a,k} < T_{a,l}$ and $T_{a,k}|T_{a,k} > T_{a,j}$ respectively, where $T_{a,k}|T_{a,k} < T_{a,l}$ is simply $T_{a,k}$ with the upper constraint of $T_{a,l}$ applied to it, and $T_{a,k}|T_{a,k} > T_{a,j}$ is $T_{a,k}$ with the lower constraint of $T_{a,j}$ applied to it. Consecutively applying the constraints in this way enables the arrival times for any number of replenishment agents to be constrained.

Once the arrival times of the agents have been conditioned on the order of arrival, the future state of the system can be predicted. For cases where multiple replenishment agents arrive at the same task at similar times to one another, the time when each replenishment agent actually begins setting up at the task, $T_{s,j}$, depends on when the previous replenishment agent finished packing up at that task, $T_{f,k}$. The conditioned

arrival time of the later replenishment agent, $T_{a,j}$, should be adjusted against the finish time of the earlier replenishment agent: $T_{s,j} = T_{a,j}^{*\geq T_{f,k}}$.

5.3.3 Proposed Algorithm

An easy implementation of calculating the expected cost of a schedule is to generate every possible order of arrival and then calculate the cost and probability of each order, conditioning the arrival times as necessary. This may be sufficient for schedules where there are very few possible orders of arrival, but can take too long to compute for schedules with many possible orders. In such cases, there are generally many orders that are highly unlikely to occur. The proposed method, presented in Algorithm 5.5, speeds up the process by generating the orders as the cost is being calculated and discarding unlikely orders based on the computed arrival times.

Algorithm 5.5 first starts by updating the predicted state with any tasks that are currently being performed, excluding those where the replenishment agent is still travelling to the task (line 6). Following this, it predicts the effects of the remaining tasks in the schedule. On line 11, tasks which are not performed by other replenishment agents, tasks where the replenishment agent arrives far enough in advance of any other replenishment agents such that it can be assumed to always arrive first, or the task of being replenished at the replenishment point, are performed.

If no task was performed, then line 15 gets the index of the replenishment agent that is expected to arrive at its next task first. Using the next task for this replenishment agent, line 16 checks whether the other replenishment agents which also have this task in their schedule will arrive at that task at a similar time, and whether it is the first task in their schedule. If these conditions are satisfied, the state is copied and updated for every possible order of arrival in `DuplicateAndUpdateState` on line 18.

If the conditions on line 16 are not satisfied, the algorithm again copies and updates the state for every possible order of arrival on line 21, ignoring replenishment agents where the task under consideration is not the first task in their schedule. This is an approximation that is necessary to avoid the algorithm reaching a deadlock state. The

Algorithm 5.5: Multi-agent cost calculation

`MultiCostCalc(`$\boldsymbol{\psi}_{initial}, \boldsymbol{\theta}$`)`
input : Current system state, $\boldsymbol{\psi}_{initial}$; schedule, $\boldsymbol{\theta}$
output: Weighted downtime, λ; state, $\boldsymbol{\psi}$
$\boldsymbol{\alpha}[1] \leftarrow \boldsymbol{\psi}_{initial}, \boldsymbol{\beta}[1] \leftarrow \boldsymbol{\theta}$ `// array of states and array of schedules`
$d \leftarrow 1$ `// length of arrays`
`// update initial state with tasks in progress`
for $j \in \{1, ..., m\}$ **do**
  if *replenishment agent j is currently setting up, replenishing a user agent, or packing up* **then**
    update $\boldsymbol{\alpha}[1]$ with task $\boldsymbol{\beta}[1]_{j,1}$, remove task $\boldsymbol{\beta}[1]_{j,1}$ from $\boldsymbol{\beta}$
`// generate possible orders, predict their future states`
while *there are tasks remaining in* $\boldsymbol{\beta}$ **do**
  for $i \in \{1, ..., d\}$ *where the number of tasks in* $\beta[i]$ *is larger than the minimum number of tasks in* $\beta[j], \forall j \in 1, ..., d$ **do**
    for $j \in \{1, ..., m\}$ **do**
      if *task* $\boldsymbol{\beta}[i]_{j,1}$ *does not exist in* $\boldsymbol{\beta}[i]_k \forall k \in \{1, ..., m\}, k \neq j$ *or the agent will arrive before any other instances of the task or* $\boldsymbol{\beta}[i]_{j,1} = 0$ **then**
        update $\boldsymbol{\alpha}[i]$ with task $\boldsymbol{\beta}[i]_{j,1}$, remove task $\boldsymbol{\beta}[i]_{j,1}$ from $\boldsymbol{\beta}$
        break
    if *the state was not updated with a task* **then**
      $j \leftarrow$ index of first replenishment agent to arrive at its next task
      if $\boldsymbol{\beta}[i]_{j,1}$ *exists in* $\boldsymbol{\beta}[i]_k, \forall k \in \{1, ..., m\}, k \neq j$ *and j will arrive at a similar time to k and* $\boldsymbol{\beta}[i]_{k,1} = \boldsymbol{\beta}[i]_{j,1}$ **then**
        for *all possible arrival orders,* $\boldsymbol{\delta}$*, of task* $\boldsymbol{\beta}[i]_{j,1}$ **do**
          $\boldsymbol{\alpha}, \boldsymbol{\beta}, d \leftarrow$ `DuplicateAndUpdateState(`$\boldsymbol{\alpha}, \boldsymbol{\beta}, d, i, \boldsymbol{\delta}$`)`
      else
        for *all possible arrival orders,* $\boldsymbol{\delta}$*, of task* $\boldsymbol{\beta}[i]_{j,1}$*, ignoring agents where the task is not the first in their schedule* **do**
          $\boldsymbol{\alpha}, \boldsymbol{\beta}, d \leftarrow$ `DuplicateAndUpdateState(`$\boldsymbol{\alpha}, \boldsymbol{\beta}, d, i, \boldsymbol{\delta}$`)`
        remove state $\boldsymbol{\alpha}[i]$ and schedule $\boldsymbol{\beta}[i]$, $d \leftarrow d - 1$
  $(\boldsymbol{\alpha}, \boldsymbol{\beta}, d) \leftarrow$ `Reduce(`$\boldsymbol{\alpha}, \boldsymbol{\beta}, d$`)`
`// calculate average cost and state`
$\lambda \leftarrow 0$
for $i \in \{1, ..., d\}$ **do**
  $t \leftarrow$ `GetMinTime(`$\boldsymbol{\alpha}[i]$`)`
  update $\boldsymbol{\alpha}[i]$ to time t
  $\lambda \leftarrow \lambda +$ `GetDowntime(`$\boldsymbol{\alpha}[i]$`)` $\times$ `CalcProb(`$\boldsymbol{\alpha}[i], n$`)`
$\boldsymbol{\psi} \leftarrow$ probability weighted average of $\boldsymbol{\alpha}$

original state and schedule which is copied in the `DuplicateAndUpdateState` function is removed on line 22. The `Reduce` function on line 23 removes any duplicate orders, as well as low probability orders.

Once there are no more tasks remaining in the schedules, the minimum time taken by a replenishment agent to execute the schedule is returned by `GetMinTime` on line 27 and used on line 28 to update the expected system state to the completion time of the schedule. The `GetDowntime` and `CalcProb` functions on line 29 return the downtime and probability of each expected state based on the order that the tasks were performed in, and contribute to the probability weighted expected downtime, λ. Finally, the average future state is calculated on line 30.

The check for whether the replenishment agents will arrive at similar times or not (line 11 and line 16) is performed using Equation 5.28. If the probability is greater than some threshold (99.9% is used in this thesis), then it can be assumed that the first replenishment agent will arrive far enough in advance of the second one such that the reverse order can be ignored.

The `DuplicateAndUpdateState` function is outlined in Algorithm 5.6. The original state is copied on line 3, and the original schedule is copied on line 4. Then line 5 to line 7 updates the copy of the state with the first task of the replenishment agents in order of their arrival, specified by $\boldsymbol{\delta}$, and removes that task from the copy of the schedule. Note that the first task for each of the replenishment agents listed in $\boldsymbol{\delta}$ should be identical.

A `Reduce` function is outlined in Algorithm 5.7. Duplicate orders are first removed on lines 4 to 9. It then eliminates low probability orders on lines 11 to 15 based on the threshold multiplier, γ. The actual threshold used to determine whether an order is removed is calculated by multiplying the probability of the most likely order by the threshold multiplier (line 12). A nominal value of $\gamma = 0.01$ has been specified in the algorithm. The probability of each arrival order is estimated using the `EstimateProb` algorithm outlined in Algorithm 5.4 instead of using `CalcProb` to calculate the actual probability in order to reduce computation time. Provided γ is small enough, using the estimated probability is unlikely to result in arrival orders that actually have a

Algorithm 5.6: Duplicate and update the state

`DuplicateAndUpdateState(`$\boldsymbol{\alpha}, \boldsymbol{\beta}, d, i, \boldsymbol{\delta}$`)`

input : Array of states, $\boldsymbol{\alpha}$; array of schedules, $\boldsymbol{\beta}$; length of array, d; state to duplicate, i; list of replenishment agents in order of arrival, $\boldsymbol{\delta}$

output: Array of states, $\boldsymbol{\alpha}$; array of schedules, $\boldsymbol{\beta}$; length of array, d

```
calculate the conditioned arrival times
d ← d + 1
copy state α[i] and store in α[d]
copy schedule β[i] and store in β[d]
for j ∈ δ do
    update state α[d] with the task β[d]_{j,1}
    remove task β[d]_{j,1} from β[d]
```

Algorithm 5.7: Reduce

`Reduce(`$\boldsymbol{\alpha}, \boldsymbol{\beta}, d$`)`

input : Array of states, $\boldsymbol{\alpha}$; array of schedules, $\boldsymbol{\beta}$; length of arrays, d

output: Array of states, $\boldsymbol{\alpha}$; array of schedules, $\boldsymbol{\beta}$; length of arrays, d

```
γ ← 0.01                          // threshold multiplier
p ← GetMaxProbability(α)
// remove duplicate orders
for i ∈ {1, ..., d} do
    for j ∈ {i + 1, ..., d} do
        if α[i] == α[j] then
            remove state α[i] from α
            remove schedule β[i] from β
            d ← d − 1
// remove low probability orders
for i ∈ {1, ..., d} do
    if EstimateProb(α[i]) ≤ γ × p then
        remove state α[i] from α
        remove schedule β[i] from β
        d ← d − 1
```

high probability of occurring being removed. Other possible `Reduce` functions could restrict the number of orders to the n most likely, discard the cumulative $\gamma\%$ least likely, or a combination of the above. The advantage of the approach detailed in Algorithm 5.7 over these other possible `Reduce` functions is that it retains all orders that have a probability close to the highest probability order. Retaining only the n most likely orders can result in relatively high probability orders being discarded, and discarding the $\gamma\%$ least likely may retain many low probability orders in the case where one order is significantly more likely than the others.

5.3.4 Conditioning the Previous Arrival Times

Algorithm 5.5 assumes that the arrival time distributions for all of the tasks of a single replenishment agent are independent. In practice, this assumption is not valid. Let $T_{a,j,1}$ be the arrival time distribution for replenishment agent j at task 1, $T_{a,j,2}$ be the arrival time distribution for j at task 2, and $T_{a,k,2}$ be the arrival time distribution for replenishment agent k at task 2. If j arrives before k at task 2, then the arrival time distributions for j and k at task 2 can be adjusted using Equation 5.34. The arrival time distribution $T_{a,j,1}$ can be adjusted using a similar process. Let $T_{a,j,2-1}$ be the time between the arrival at task 1 to the arrival at task 2 for replenishment agent j: $T_{a,j,2-1} = T_{a,j,2} - T_{a,j,1}$. Then:

$$p(t_{a,j,1}|T_{a,j,2} < T_{a,k,2}) = \frac{p(t_{a,j,1})P(t_{a,j,1} + T_{a,j,2-1} < T_{a,k,2})}{P(T_{a,j,2} < T_{a,k,2})} \tag{5.38}$$

where

$$\begin{aligned} P(t_{a,j,1} + T_{a,j,2-1} < T_{a,k,2}) &= P(t_{a,j,1} < T_{a,k,2} - T_{a,j,2-1}) \\ &= \int_{t_{a,j,1}}^{\infty} p(T_{a,k,2} - T_{a,j,2-1})\mathrm{d}t \\ &= 1 - \mathrm{CDF}_{T_{a,k,2} - T_{a,j,2-1}}(t_{a,j,1}) \end{aligned} \tag{5.39}$$

$$\begin{aligned} P(T_{a,j,2} < T_{a,k,2}) &= P(T_{a,k,2} - T_{a,j,2} > 0) \\ &= \int_0^\infty p(T_{a,k,2} - T_{a,j,2})\mathrm{d}t \\ &= 1 - \mathrm{CDF}_{T_{a,k,2}-T_{a,j,2}}(0) \end{aligned} \tag{5.40}$$

This process can be applied for an arbitrary number of arrival times. The problem with this process is that it is a chicken-and-egg problem—calculating the arrival times of the earlier tasks requires the arrival times of the later tasks to already be calculated. However, the later arrival times depend on the arrival times of the earlier tasks. Some form of iterative method would be required for the arrival time distributions to converge to the actual distributions, resulting in a significant increase in the computation time of the algorithm compared to Algorithm 5.5.

5.3.5 Alternative Greedy Algorithm

A downside of Algorithm 5.5 is that it will potentially be computationally very expensive in cases where there are many possible orders of arrival. Algorithm 5.8 outlines an alternative method which greedily selects the order of tasks based on the arrival times. Regardless of the difference in arrival times between two agents, the one which is more likely to arrive first is selected. In the case that two agents have the exact same mean arrival time, the one with the lowest index is chosen. This method ignores all of the other possible orders, and does not necessarily end up with the most probable order. However, if the schedule can be chosen such that replenishment agents do not arrive at similar times, this approach will give accurate results. The advantage of this method is the reduced computation time. An optimisation method which avoids replenishment agents arriving at similar times and uses `MultiCostCalcGreedy` is introduced in Chapter 6.

Algorithm 5.8: Multi-agent cost calculation with greedily chosen order

`MultiCostCalcGreedy(`$\boldsymbol{\psi}_{initial}, \boldsymbol{\theta}$`)`
input : Current system state, $\boldsymbol{\psi}_{initial}$; schedule, $\boldsymbol{\theta}$
output: Weighted downtime, λ; state, $\boldsymbol{\psi}$

$\boldsymbol{\psi} \leftarrow s_{initial}$ `// state`
$\boldsymbol{\beta} \leftarrow \boldsymbol{\theta}$ `// schedule`
`// update initial state with tasks in progress`
for $j \in \{1, ..., m\}$ **do**
  if *replenishment agent j is currently setting up, replenishing a user agent, or packing up* **then**
    update $\boldsymbol{\psi}$ with task $\boldsymbol{\beta}_{j,1}$
    remove task $\boldsymbol{\beta}_{j,1}$ from $\boldsymbol{\beta}$
`// get the earliest task and predict the future state`
while *there are tasks remaining in $\boldsymbol{\beta}$* **do**
  $j \leftarrow$ index of replenishment agent that will arrive at its next task first
  update $\boldsymbol{\psi}$ with task $\boldsymbol{\beta}_{j,1}$
  remove task $\boldsymbol{\beta}_{j,1}$ from $\boldsymbol{\beta}$
`// calculate final cost and state`
$t \leftarrow$ `GetMinTime(`$\boldsymbol{\alpha}$`)`
update $\boldsymbol{\psi}$ to time t
$\lambda \leftarrow$ `GetDowntime(`$\boldsymbol{\alpha}[i]$`)`

5.4 Results

This section evaluates the performance of the prediction algorithms developed in this chapter. Section 5.4.1 first introduces a number of scenarios that will be used to evaluate the algorithms. Section 5.4.2 then compares the analytical prediction method with the MC approach in single-replenishment agent scenarios. Finally, Section 5.4.3 evaluates the analytical methods in multi-replenishment agent scenarios. All methods were implemented in Python and calculation times were calculated on a 2.8GHz Intel i7-640M.

5.4.1 Scenarios

Four scenarios were used to evaluate the performance of the prediction methods developed in this chapter. These scenarios are also used to evaluate the optimisation methods presented in Chapter 6. Of the four scenarios, Scenarios S1 and S2 are single-replenishment agent scenarios, and Scenarios M1 and M2 are multi-replenishment agent scenarios. The replenishment agents are assumed to initially be positioned at the replenishment point. For the purpose of testing the prediction methods, the initial resource levels of all agents were initialised to a random value between 0% and 100% of the agent capacity. The rest of this section outlines the parameters and layout of the user agents, replenishment agents, and the replenishment point for each scenario.

Scenario S1

The first scenario consists of multiple agents performing a persistent collection task, with the resource under consideration being storage capacity. Each agent operates in their own small area where the items being collected are concentrated, so they are relatively fixed in space compared to the whole system. The replenishment agent must travel along roads to each user agent—the map of the roads is shown in Figure 5.7. The set of user agents is heterogeneous and their parameters are shown in Table 5.1. The replenishment agent parameters are shown in Table 5.2, and the parameters of the replenishment point are shown in Table 5.3.

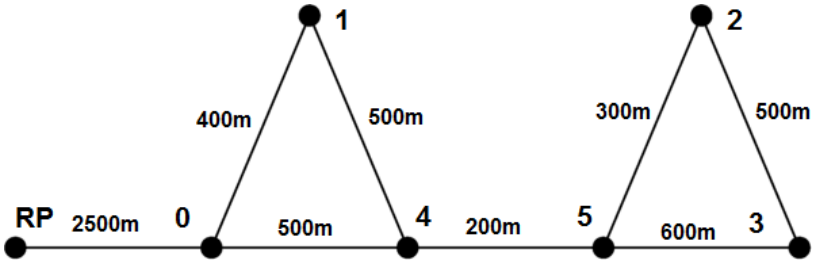


Figure 5.7 – Layout showing the location of the Replenishment Point (RP) and user agents for Scenario S1. The numbered nodes designate the operational areas of the user agents.

Table 5.1 – User agent parameters for Scenario S1

User agent, i	1	2	3	4	5	6
$c_{u,i}$ (L)	1000	1200	700	1200	1000	800
$R_{u,i}$ mean (L/s)	0.5	0.5	0.3	0.5	0.4	0.4
$R_{u,i}$ standard deviation (L/s)	0.05	0.05	0.05	0.02	0.08	0.04

Table 5.2 – Replenishment agent parameters for Scenario S1

Parameter	Mean	Standard Deviation
c_a (L)	5000	-
R_a (L/s)	10	0.5
T_{sa} (s)	60	20
T_{pa} (s)	20	5
V_a (m/s)	15	0.5

Table 5.3 – Replenishment point parameters for Scenario S1

Parameter	Mean	Standard Deviation
T_{sr} (s)	30	10
T_{pr} (s)	10	1
R_r (L/s)	20	1

The scenario was run using the first 4, 5, and 6 user agents to represent under-utilised, fully-utilised, and over-utilised scenarios respectively. In the under-utilised scenario the replenishment agent should be able to prevent all of the user agents from exhausting their supply of the resource, while in the over-utilised scenario the replenishment agent will not be able to prevent the user agents from incurring downtime. The fully-utilised case sits between the other two.

Scenario S2

The second single-replenishment agent scenario involves the delivery of generator fuel to 20 remote villages by truck. The quality of roads between villages is highly variable which results in travel speeds that are very uncertain, while the villages use the fuel at a relatively predictable rate. Figure 5.8 shows the network of roads connecting the villages.

Three different size trucks were considered with parameters shown in Table 5.4. The different truck sizes vary only in their capacity—the resource usage rates, setup and packup times, and velocity, are the same for all truck sizes. The large, medium, and small truck sizes roughly correspond to under-, fully-, and over-utilised scenarios respectively. The parameters of the replenishment point are shown in Table 5.5. There are four different sizes of villages, each with approximately 80 days supply of fuel. The parameters of each size of village are shown in Table 5.6, and the size of each village is outlined in Table 5.7. The truck is assumed to operate for 12 hours each day.

Scenario M1

This scenario is designed to be more representative of a mining scenario, with multiple fuel trucks refuelling various pieces of equipment throughout a mine. The road layout for this scenario is the same as for Scenario S1, shown in Figure 5.7. The user agent, replenishment agent, and replenishment point parameters are shown in Table 5.8, Table 5.9, and Table 5.10 respectively. Note that, in this scenario, there are multiple

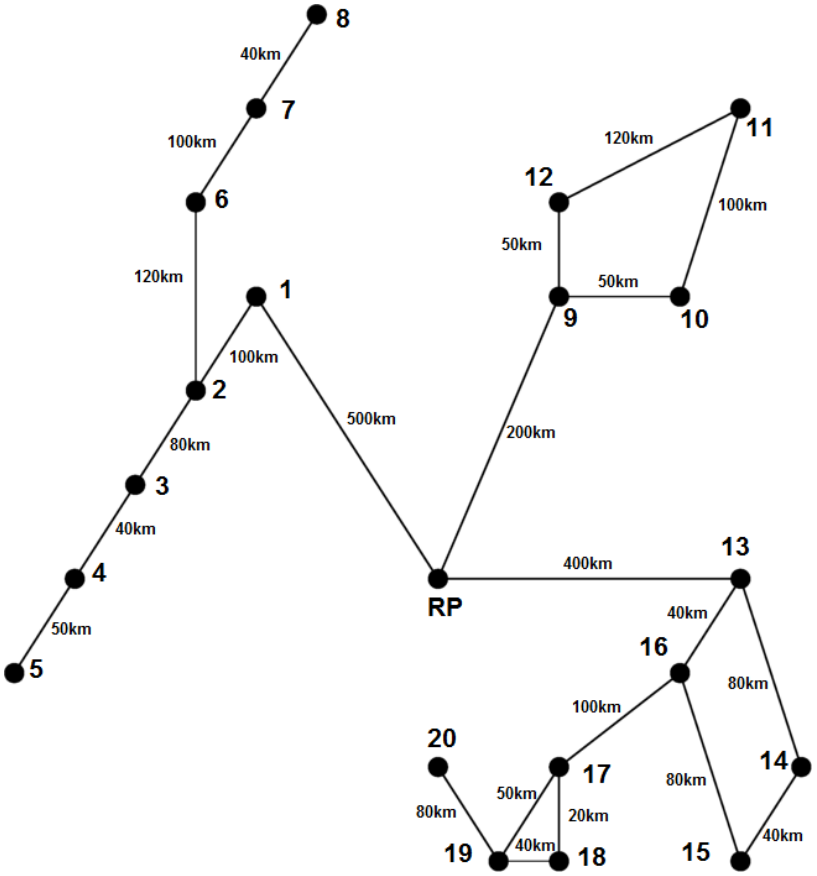


Figure 5.8 – Layout showing the location of the Replenishment Point (RP) and villages for Scenario S2.

Table 5.4 – Replenishment agent parameters for Scenario S2

Parameter	Mean	Standard Deviation
c_a Large (L)	23000	-
c_a Medium (L)	15000	-
c_a Small (L)	10000	-
R_a (L/hr)	600	60
T_{sa} (min)	120	10
T_{pa} (min)	40	3
V_a (km/hr)	16	4

Table 5.5 – Replenishment point parameters for Scenario S2

Parameter	Mean	Standard Deviation
T_{sr} (min)	120	20
T_{pr} (min)	60	10
R_r (L/hr)	10000	1000

Table 5.6 – User agent parameters for Scenario S2

Parameter	Mean	Standard Deviation
Small Village (S)		
$c_{u,i}$ (L)	4000	-
$R_{u,i}$ (L/day)	50	2.5
Medium Village (M)		
$c_{u,i}$ (L)	5000	-
$R_{u,i}$ (L/day)	62.5	2.5
Large Village (L)		
$c_{u,i}$ (L)	6000	-
$R_{u,i}$ (L/day)	75	2.5
Extra Large Village (XL)		
$c_{u,i}$ (L)	8000	-
$R_{u,i}$ (L/day)	100	2.5

Table 5.7 – User agent types for Scenario S2

Village	1	2	3	4	5	6	7	8	9	10
Size	S	L	S	M	M	S	M	L	XL	L
Village	11	12	13	14	15	16	17	18	19	20
Size	L	XL	M	L	M	S	L	M	L	M

user agents at some nodes to reflect multiple drills or excavators operating on the same bench in a mine.

Table 5.8 – User agent parameters for Scenario M1

User agent, i	1	2	3	4
Position	1	1	2	2
$c_{u,i}$ (L)	1000	1000	1200	1000
$R_{u,i}$ mean (L/s)	0.5	0.5	0.5	0.5
$R_{u,i}$ standard deviation (L/s)	0.05	0.05	0.05	0.05
User agent, i	5	6	7	8
Position	3	4	5	6
$c_{u,i}$ (L)	1000	800	700	1200
$R_{u,i}$ mean (L/s)	0.4	0.4	0.3	0.5
$R_{u,i}$ standard deviation (L/s)	0.08	0.04	0.05	0.02

Table 5.9 – Replenishment agent parameters for Scenario M1

Replenishment agent, j	1	2
$c_{a,j}$ (L)	5000	2500
$R_{a,j}$ mean (L/s)	10	10
$R_{a,j}$ standard deviation (L/s)	0.5	0.5
$T_{sa,j}$ mean (s)	60	60
$T_{sa,j}$ standard deviation (s)	20	20
$T_{pa,j}$ mean (s)	20	20
$T_{pa,j}$ standard deviation (s)	5	5
$V_{a,j}$ mean (m/s)	15	7
$V_{a,j}$ standard deviation (m/s)	0.5	0.5

Table 5.10 – Replenishment point parameters for Scenario M1

T_{sr} mean (s)	30
T_{sr} standard deviation (s)	10
T_{pr} mean (s)	30
T_{pr} standard deviation (s)	10
R_r mean (L/s)	20
R_r standard deviation (L/s)	1

Scenario M2

The second multi-replenishment agent scenario has a homogeneous fleet of 5 replenishment agents and a heterogeneous set of 3 user agents. It is designed to highlight potential issues with the developed algorithms and is not representative of real-world scenarios. The layout of the scenario is shown in Figure 5.9. The parameters of the agents and replenishment point are shown in Table 5.11, Table 5.12, and Table 5.13.

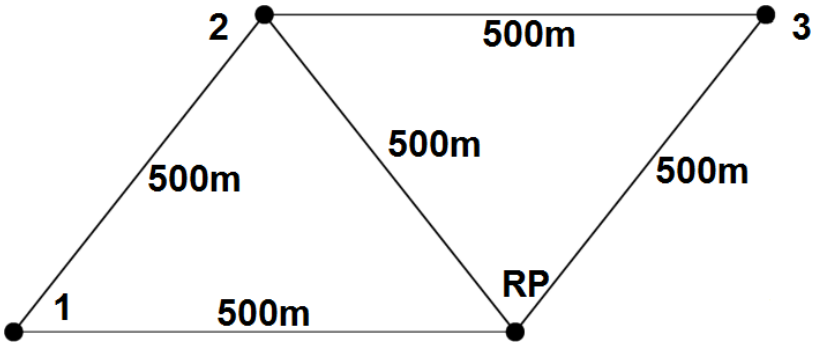


Figure 5.9 – Layout showing the location of the Replenishment Point (RP) and user agents for Scenario M2. A user agent is positioned at each node.

Table 5.11 – User agent parameters for Scenario M2

User agent, i	1	2	3
$c_{u,i}$ (L)	100	125	150
$R_{u,i}$ mean (L/s)	0.6	1	1.2
$R_{u,i}$ standard deviation (L/s)	0.1	0.1	0.1

Table 5.12 – Replenishment agent parameters for Scenario M2

$c_{a,j}$ (L)	200
$R_{a,j}$ mean (L/s)	5
$R_{a,j}$ standard deviation (L/s)	1
$T_{sa,j}$ mean (s)	10
$T_{sa,j}$ standard deviation (s)	2
$T_{pa,j}$ mean (s)	10
$T_{pa,j}$ standard deviation (s)	2
$V_{a,j}$ mean (m/s)	10
$V_{a,j}$ standard deviation (m/s)	2

Table 5.13 – Replenishment point parameters for Scenario M2

T_{sr} mean (s)	10
T_{sr} standard deviation (s)	2
T_{pr} mean (s)	10
T_{pr} standard deviation (s)	2
R_r mean (L/s)	10
R_r standard deviation (L/s)	1

5.4.2 Single-Agent Results

The methods were tested by using random schedules of a fixed number of tasks. To allow comparison of the costs generated by these schedules, the following ratio objective function is defined:

$$\arg\min \phi = \frac{\sum_{i=1}^{n} w_i t_{empty,i}}{n t_{total}} = \frac{\lambda}{n t_{total}} \tag{5.41}$$

where w_i is the weighting applied to user agent i, $t_{empty,i}$ is the downtime of user agent i, t_{total} is the total time taken to execute the schedule, and n is the number of user agents. Provided the weights sum to n, then the objective function is bound between 0 and 1:

$$\text{if } \sum_{i=1}^{n} w_i = n \text{ then } 0 \leq \phi \leq 1 \tag{5.42}$$

If $w_1 = w_2 = ... = w_n$, the ratio cost is equivalent to the proportion of time during the schedule that the user agents are expected to be non-operational. In all of the scenarios examined, each user agent had a weighting of 1.

The MC method uses multiple samples to calculate the expected cost of a schedule, with the accuracy of the expected cost improving as the number of samples is increased. Figure 5.10 shows the absolute error of a cost calculation using the MC method as a function of the number of samples used. The computational complexity of the MC method is $\mathcal{O}((a+n)b)$ where a is the number of tasks in the schedule, n

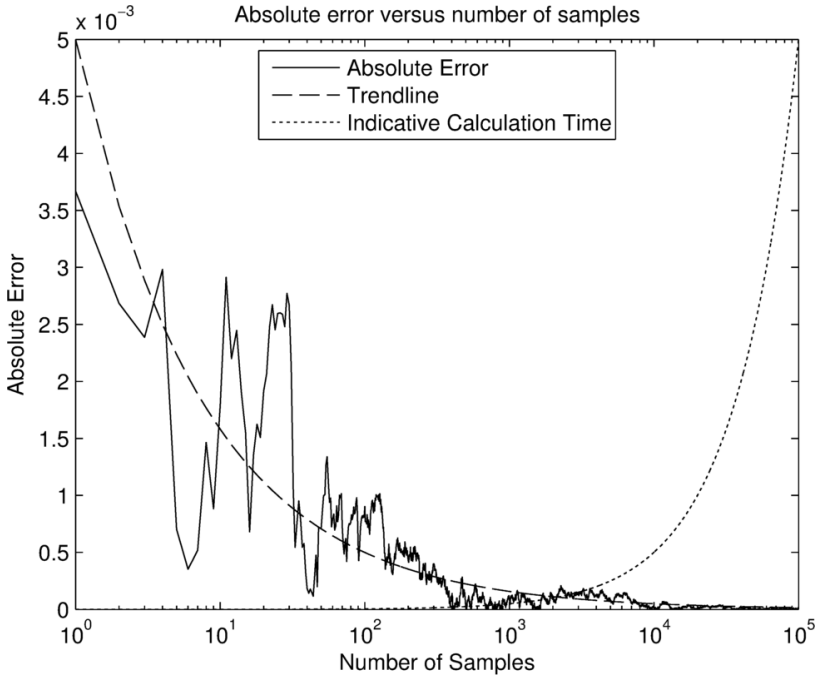


Figure 5.10 – Absolute error versus number of samples for the Monte Carlo method in Scenario S1. The calculation time line shown is indicative of the trend in calculation time rather than the actual calculation time.

is the number of user agents, and b is the number of samples used. 1000 samples was chosen as it offers a good trade-off between absolute error and calculation time, with an absolute error of approximately 1.5×10^{-4} and a calculation time of 1.5s.

For evaluating the prediction methods, Scenario S1 was used with 6 user agents, and Scenario S2 was used with the large truck. For each scenario, the cost of 10,000 randomly generated schedules was calculated using both the MC and analytical frameworks. Schedules consisted of 8 tasks in Scenario S1, and 20 tasks in Scenario S2. The difference between the analytical and MC methods was then calculated for each schedule. The distribution of differences is approximately normally distributed with means and standard deviations shown in Table 5.14.

As can be seen in Table 5.14, the analytical framework produced costs which were very close to the MC method in Scenario S1. Based on the distribution of differences, over 99.8% of costs calculated by the analytical method were within 0.005 of the MC method. In Scenario S2, the analytical method had a larger deviation from the MC

Table 5.14 – Proposed cost method minus Monte Carlo cost method

Scenario	Method	Mean ($\times 10^{-3}$)	Standard Deviation ($\times 10^{-3}$)	Comparison Accuracy	Computation Time (s)
S1	Analytical	0.08	1.52	99.6%	0.007
S1	MC	–	–	–	0.678
S2	Analytical	-1.96	1.92	99.4%	0.009
S2 (mod)	Analytical	0.71	0.72	99.8%	0.009
S2	MC	–	–	–	1.506

method. This was a result of the Gaussian approximations starting to break down. The standard deviation of the velocity of the replenishment agent in this scenario was very high, resulting in inverse Gaussian distributions that are heavily skewed to one side. The Gaussian approximation of the inverse Gaussian distribution loses a lot of the skewed tail, resulting in this difference (see Figure 5.2b). The degradation resulting from the approximation was shown by the improved results in the modified Scenario S2 in Table 5.14 where the standard deviation of the velocity was reduced from 4km/hr to 2km/hr. With the reduction in standard deviation, the difference between the two methods was significantly reduced.

The most important aspect of the cost calculation framework is its ability to accurately discriminate between two schedules as it will be used later within optimisation methods to select the lowest cost schedule. The comparison accuracy of the analytical framework was tested by comparing the costs calculated for each schedule with the cost of every other schedule. A correct result was recorded if both the MC and analytical methods agreed on which schedule produced the lowest cost. With the 10,000 schedules, almost 50,000,000 comparison were made. Correct comparison results were recorded in 99.6% of cases for Scenario S1 and 99.8% of cases for the modified Scenario S2. In the cases where the analytical method did not produce the same result as the MC method, the average difference in the proportion of downtime was 1.7×10^{-3} for Scenario S1 and Scenario S2, and 6.3×10^{-4} for the modified Scenario S2, for both methods. These results show that the analytical method is highly accurate with respect to the MC method. In the cases where it produced the wrong comparison result, the differences in the costs are small enough to be considered negligible.

The main advantage of the analytical framework over the MC approach is the computation time. The analytical method was almost 100 times faster than the MC method in Scenario S1, and over 150 times faster in Scenario S2. The computational complexity of the analytical framework in the single-replenishment agent scenarios is $\mathcal{O}(a+n)$ where a is the number of tasks in the schedule and n is the number of user agents.

5.4.3 Multi-Agent Results

The analytical framework was tested in Scenarios M1 and M2 using threshold multipliers of $\gamma \in \{0.01, 0.1, 1\}$, and using the framework with greedily chosen order so that there are no arrival order combinations to be considered (denoted by NC). 10,000 random schedules consisting of 10 tasks per replenishment agent for Scenario M1, and 5 tasks per replenishment agent for Scenario M2, were generated for each scenario, with the MC method again tested using 1,000 samples. The mean and standard deviation of the difference between the analytical and MC methods, the comparison accuracy, and the computation time for each method are shown in Table 5.15. Like Scenario S2, the velocity of the replenishment agents in Scenario M2 has a high standard deviation which degrades the performance of the inverse Gaussian approximation. A version of Scenario M2 with a lower velocity standard deviation of 0.5m/s for the replenishment agents was also tested (see M2 (mod) results) to show the accuracy of the prediction methods when the inverse Gaussian approximation produces better results.

The effect of the threshold multiplier, γ, is clear in the results. When $\gamma = 1$, the analytical method only considered the most likely arrival order and ignored any others. Replenishment agents in Scenario M2 were highly likely to arrive at similar times to one another, so using a γ of 1 had clear detrimental effects in this case. As γ was reduced and more arrival orders were considered in parallel, the difference between the MC and analytical methods decreased and the comparison accuracy increased. The trade-off was that the computation time increased. In Scenario M2, the increase in computation time was significant, with it actually taking longer than the MC method

when $\gamma = 0.1$ and $\gamma = 0.01$.

The cost calculation method using the greedily chosen order (NC) generally resulted in costs that were higher than those generated by the full cost calculation. In many cases, the NC method and $\gamma = 1$ case calculated the cost using the same order of arrival. The difference between these methods was a result of the NC method not conditioning the arrival times as per Section 5.3.2. For schedules where the replenishment agents do not arrive at similar times, these two methods give similar results.

The comparison accuracy in Scenario M1 was similar to the single-replenishment agent scenarios, while the best comparison accuracy dropped to 98.0% in Scenario M2 and 98.8% in Scenario M2 (mod). The average difference in cost between schedules for incorrect comparisons was 7.5×10^{-3} for Scenario M2 and 3.7×10^{-3} for Scenario M2 (mod). Figure 5.11 shows the accuracy of the MC method as a function of the number of samples for Scenario M2. The average error at 1,000 samples was approximately

Table 5.15 – Proposed cost method minus Monte Carlo cost method

Scenario	γ	Mean ($\times 10^{-3}$)	Standard Deviation ($\times 10^{-3}$)	Comparison Accuracy	Computation Time (s)
M1	1	-0.74	5.00	99.2%	0.020
M1	0.1	-0.71	2.12	99.6%	0.026
M1	0.01	-0.69	2.07	99.6%	0.036
M1	NC	-0.05	4.86	99.2%	0.014
M1	MC	–	–	–	1.995
M2	1	-9.43	28.5	92.4%	0.193
M2	0.1	-5.91	10.9	97.7%	2.210
M2	0.01	-3.73	10.0	98.0%	5.813
M2	NC	10.5	29.8	92.3%	0.010
M2	MC	–	–	–	1.540
M2 (mod)	1	-4.81	26.8	93.0%	0.063
M2 (mod)	0.1	-3.13	4.57	98.7%	0.579
M2 (mod)	0.01	-2.50	4.12	98.8%	0.878
M2 (mod)	NC	1.08	27.4	92.8%	0.010
M2 (mod)	MC	–	–	–	1.714

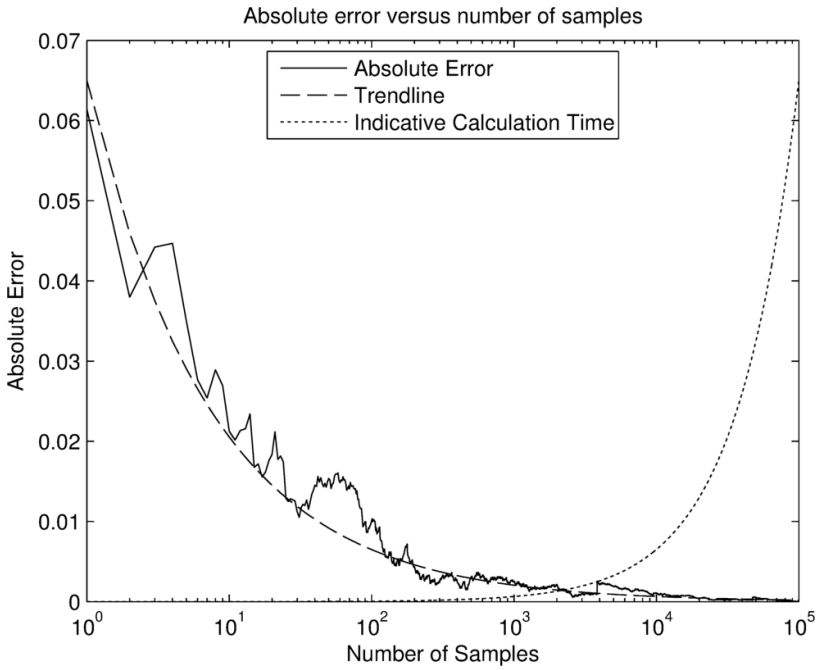


Figure 5.11 – Absolute error versus number of samples for the Monte Carlo method in Scenario M2. The calculation time line shown is indicative of the trend in calculation time rather than the actual calculation time.

2.5×10^{-3}, with similar results for Scenario M2 (mod). A significant portion of the incorrect comparisons, particularly in Scenario M2 (mod), were within the noise of the MC method where it is impossible to differentiate between them using the MC method with 1,000 samples with any confidence. To get the accuracy of the MC to a similar level as in Scenario S1 would require on the order of 100,000 samples.

In Scenario M1, the analytical framework had a clear advantage in computation time, computing at least 70 times faster than the MC framework. In Scenario M2, the computation time increased significantly as γ was decreased. The long computation times were a result of the large number of possible arrival orders that were considered. The analytical method is $\mathcal{O}(a+n)$ in the best case where no replenishment agents perform the same task (typical for mining and agricultural scenarios where there are significantly more user agents than replenishment agents), and $\mathcal{O}(a!(1+n))$ in the worst case where all replenishment agents perform the same task (for scenarios where there are significantly more replenishment agents than user agents). Increasing the thresh-

old multiplier, γ, reduces the computational complexity at the cost of accuracy, with the computation complexity approaching $\mathcal{O}(a+n)$ when $\gamma = 1$. It should be noted that calculating the CDF of the multivariate Gaussian distribution using `MVNCDF` was responsible for approximately 70% of the computation time of the analytical method in Scenario M2. The analytical method with greedily chosen order (NC) is $\mathcal{O}(a+n)$.

5.5 Summary

This chapter presented and evaluated methods for predicting the future state of a system given an estimate of the current state, a schedule for the replenishment agents, and a model of the uncertainty. A MC approach was first introduced in Section 5.1 as a benchmark for the methods that were subsequently developed. Section 5.2 then considered the single-replenishment agent case and developed a novel analytical prediction method. This approach made use of Gaussian approximations to the inverse Gaussian distribution, ratio of Gaussian distributions, product of Gaussian distributions, and generalised rectified Gaussian distribution, to enable the uncertainty to be propagated into the future. New Gaussian approximations to the inverse Gaussian and generalised rectified Gaussian distributions were also formulated.

Following this, Section 5.3 extended the analytical prediction method to scenarios with multiple replenishment agents. A key difficulty in multi-replenishment agent scenarios is that the uncertainty in these scenarios means that replenishment agents may arrive in many different orders at the user agents for any given schedule. This section presented a method for calculating the probability of an order of arrival given the arrival times, and used the truncation method from Chapter 4 to condition the arrival times on the order of arrival. These methods were then used to develop a prediction framework that efficiently incorporated the different arrival orders. An alternative algorithm which ignores the multiple possible orders of arrival and simply greedily selects an order was also introduced.

Finally, the developed methods were evaluated in Section 5.4 using four scenarios—two single-replenishment agent scenarios and two multi-replenishment agent scenarios.

In the single-replenishment agent scenarios, the analytical method produced results that were up to 99.8% accurate in comparison to the benchmark MC method. The analytical approach was also shown to require orders of magnitude less computation time. The results for the multi-replenishment agent scenarios broadly mirrored those for the single-replenishment agent cases. The analytical approach was up to 99.6% accurate in comparison to the MC, but suffered severe degradation in computation time for scenarios where there were many possible orders of arrival.

The analytical methods developed in this chapter provide a fast and accurate way of incorporating uncertainty when predicting the future state of the system. These methods are now ready to be used within optimisation methods to quickly evaluate the risk associated with each schedule under consideration. Without these analytical methods, the optimisation methods would be infeasible due to the excessive computational time required by the MC method. In multi-replenishment agent scenarios with many possible orders of arrival, the analytical approach can be very slow. In these cases, structuring the optimisation method to avoid these cases should enable the NC analytical framework to be used without any detrimental impact to its prediction capability, reducing the computational requirement of the algorithms.

The following chapter presents methods for optimising the schedule of the replenishment agents. The chapter ultimately completes the belief space scheduling framework, using the estimation and prediction methods from Chapter 4 and this chapter in conjunction with the optimisation methods.

Chapter 6

Optimisation

The purpose of this chapter is to develop and evaluate optimisation strategies for minimising the risk of the system incurring downtime. As shown in Figure 6.1, the estimation and prediction components of belief space scheduling which were introduced in the previous chapters are critical to the optimisation component. The estimation component generates a belief of the current state of the system which is used by the optimisation and prediction components when determining the best task or tasks to be performed next. The optimisation methods use the prediction component to evaluate the tasks under consideration by predicting the belief of the future state.

Some of the literature examined in Chapter 2 formulated their problems using Mixed-Integer Linear Programming (MILP) models and used commercial solver packages such as CPLEX [1] and Gurobi [2] to find the optimal solution. A downside of these approaches is that they are computationally very expensive. In the case of Kaplan and Rabadi [54] for example, their MILP model was only solvable for small instances, taking up to 10 minutes to solve a scenario with only 12 jobs. A deterministic form of the SCAR scenario could be implemented as an MILP similar to [54], but with the use of a rolling horizon. However, to incorporate uncertainty using the prediction framework from Chapter 5 would require a Mixed-Integer Non-Linear Programming (MINLP) model which is not solvable by these commercial packages [23]. Consequently, three alternative types of optimisation methods are considered in this chapter—heuristics,

meta-heuristics, and tree search methods. Heuristics are simple rules of thumb which typically only consider which task should be performed next, while meta-heuristic and tree search methods consider combinations of tasks in the form of schedules to reduce myopia in decision making. Meta-heuristics, such as simulated annealing, use randomness to find near-optimal solutions for non-convex problems, while tree search methods, such as branch and bound, are guaranteed to find the optimal schedule. These methods are used within a Model Predictive Control (MPC)-like framework, where a new task is generated after the previous one has been completed.

The rest of this chapter is structured as follows: Section 6.1 first outlines the general framework of the optimisation methods and discusses the choice of finite horizon used. Section 6.2 then introduces and modifies three heuristics for the SCAR scenario—a greedy heuristic which allocates the replenishment agent to the user agent that is expected to exhaust its supply of the resource first, the Apparent Tardiness Cost (ATC) heuristic which balances the deadline of a user agent with the length of time taken to service it, and a modified form of the ATC heuristic which incorporates elements of uncertainty. Section 6.3 presents a simulated annealing meta-heuristic algorithm which aims to find a good schedule by making random changes to the schedule, and Section 6.4 then introduces branch and bound tree search methods. Finally, these optimisation methods are evaluated in Section 6.5.

6.1 Framework

A strategy that has commonly been used in the literature is MPC [78]. In a MPC framework, the planner generates an optimal sequence of actions for a number of look-ahead steps [44]. Some earlier methods, such as FF-Replan [117] and methods that use the maximum likelihood observation assumption [81], used the MPC framework as their primary method of dealing with uncertainty—they treated the problem as deterministic and frequently replanned to incorporate any unexpected changes to the system. All of the methods presented in this chapter are used within a MPC-like framework. Unlike the MPC methods above that replan at every time-step, it is in-

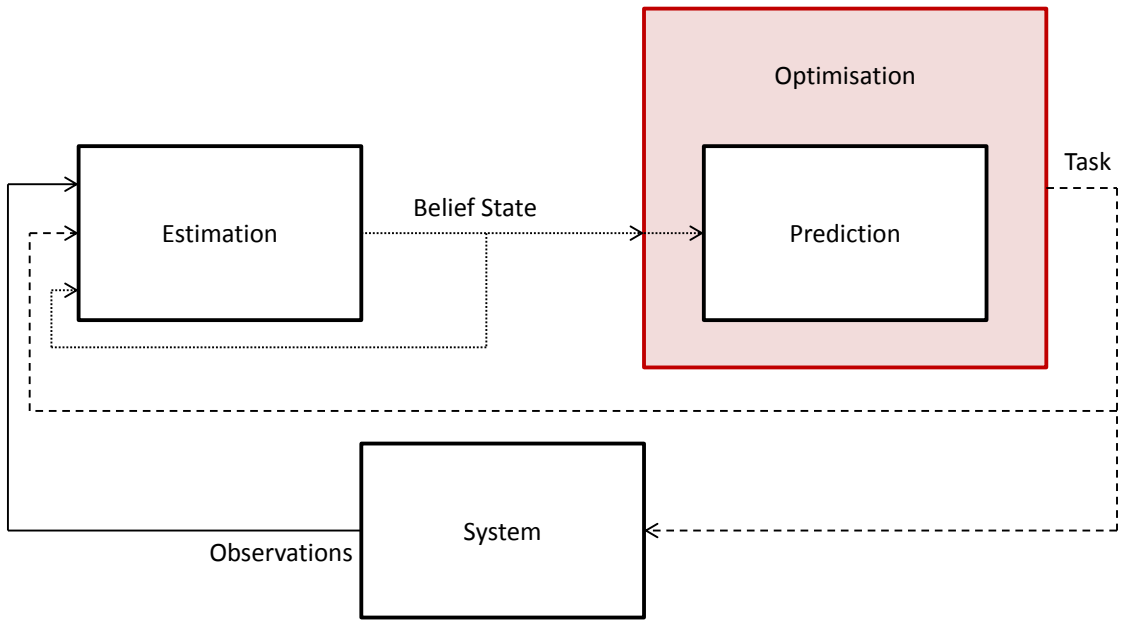


Figure 6.1 – The components of belief space scheduling. This chapter outlines methods for optimising the task or tasks that are performed by the system.

tended in this thesis that the replanning should occur after each task is complete. The primary reason for this is that the time-step in SCAR scenarios is usually significantly shorter than the time taken to perform a task, and replanning at every time-step may lead to the task of any given replenishment agent changing frequently. This behaviour is undesirable for two reasons: Firstly, on mine sites and for other scenarios where the movement of the replenishment agents is restricted by roads or other obstacles, switching to a new task may require the replenishment agent to immediately change direction which, in many cases, is either not possible or is prohibited. Secondly, in manned scenarios it is generally desirable to avoid changing the assignment of a vehicle once the vehicle is en route as frequently changing the task can lead to operator confusion.

In belief space planning methods, the number of look-ahead steps, or finite horizon, is equally defined by a look-ahead time or number of actions as an action is generally taken at each time-step. For scheduling problems, the choice of finite horizon is more complicated as tasks do not necessarily take equal lengths of time to complete.

Using a finite horizon that is defined by a length of time can present issues when using simulated annealing for reasons that will be discussed in Section 6.3. Thus, a finite horizon that specifies the number of tasks in a schedule has been used for the single-replenishment agent case to enable fair comparison between the methods. For the multi-replenishment agent case, the simulated annealing algorithm was no longer considered and a finite horizon based on time has been used.

Unlike traditional scheduling problems, replenishment agents in SCAR scenarios may perform the same task multiple times within a schedule. Intuitively, it does not make sense for a replenishment agent to perform the same task as the one that it has just completed. Therefore, the methods presented in this chapter restrict the next assignment to tasks other than the one just performed. The meta-heuristic and tree search methods take this one step further, restricting successive tasks in the generated schedule to be different:

$$\boldsymbol{\theta}_{j,i-1} \neq \boldsymbol{\theta}_{j,i} \quad \forall i \in \{2, 3...a\}, j \in \{1, ..., m\} \tag{6.1}$$

where a is the number of tasks in the schedule and m is the number of replenishment agents. Disallowing these redundant tasks has the benefit of reducing the size of the search space for these algorithms.

The general structure of the optimisation framework is shown in Algorithm 6.1. This algorithm is called each time a replenishment agent requires an assignment. If the current level of the replenishment agent is below a threshold, $l_{a,j,thresh}$ (line 1), the only valid task is to send the replenishment agent to the replenishment point to be replenished (line 2). This threshold is necessary as the heuristics are not able to incorporate the task of replenishing the replenishment agent. If the resource level is above the threshold, a schedule or assignment is generated on line 3 using the various scheduling methods discussed in the remainder of this chapter.

The objective of the optimisation, as was defined by Equation 3.8 in Section 3.3, is to minimise the downtime of the user agents over an infinite horizon. The heuristics presented in Section 6.2 evaluate each task using a rule of thumb which does not

Algorithm 6.1: Optimisation framework

`Optimiser(`$\boldsymbol{\psi}, j$`)`
input : Current system state, $\boldsymbol{\psi}$; index of replenishment agent requiring assignment, j
output: Task, θ
1 **if** $l_{a,j} < l_{a,j,thresh}$ **then** `// if the level of the replenishment agent is below the threshold`
2 $\quad \theta \leftarrow 0$
else
3 $\quad \boldsymbol{\theta} \leftarrow$ `GetSchedule(`s,j`)` `// calculate a schedule or assignment`
4 $\quad \theta \leftarrow \boldsymbol{\theta}_{j,1}$

necessarily correspond to minimising the downtime of the user agents. The combinatorial optimisation methods, on the other hand, compare the schedules using an objective function. To allow comparison of schedules using a finite horizon defined by the number of tasks, the ratio objective function defined in Equation 5.41 has been used.

6.2 Heuristics

Heuristics are rules of thumb which are used to select the best task for the agent to perform next. They are computationally cheap and are heavily used in complex scenarios where other optimisation methods are too computationally expensive [73]. While heuristics have been used in stochastic scenarios, they have typically ignored any sources of uncertainty and simply used mean parameter values to determine the next task assignment [8, 84]. This section introduces three heuristics—a simple greedy heuristic, a modified version of the ATC heuristic used by Kaplan and Rabadi [54], and a version of the ATC heuristic that incorporates uncertainty.

6.2.1 Greedy Heuristic

The Greedy heuristic (G) selects the next task of the replenishment agent based on which of the user agents is expected to run out of the resource first. This is equivalent to finding the deadline of the user agent as given on line 16 of Algorithm 5.1. This is reproduced here for convenience. For each user agent, i, the deadline, $t_{d,i}$, is calculated as:

$$t_{d,i} = \frac{l_{u,i}}{r_{u,i,\mu}} \tag{6.2}$$

where $l_{u,i}$ is the mean value of the belief of the current resource level. The user agent with the earliest deadline is then selected as the next task for the replenishment agent.

In multi-replenishment agent systems, it is necessary to consider the actions of the other replenishment agents. One method would be to restrict the user agents under consideration to those that do not already have a replenishment agent assigned to them. This method is suitable for scenarios consisting of more user agents than replenishment agents, but will give undesirable results in scenarios where the replenishment agents outnumber the user agents. In such cases, the actions of the replenishment agents should be simulated using the framework in Section 5.1 with the mean parameter values to calculate the deadline of each user agent given the assignments of the other replenishment agents, assuming that the replenishment agents that are already assigned to tasks will arrive before the replenishment agent under consideration.

6.2.2 Apparent Tardiness Cost Heuristic

The ATC heuristic used by Kaplan and Rabadi [54] for the aerial refuelling problem is a combination of the Weighted Shortest Processing Time first (WSPT) and Minimum Slack first (MS) rules. It calculates priorities for each task based on the following formula:

$$\pi_i = \frac{w_i}{t_{l,i}} \phi_i \tag{6.3}$$

where π_i is the priority of task i, w_i is the weight assigned to that task, $t_{l,i}$ is the total processing time of the task, and ϕ_i is the marginal cost of delay. The marginal cost of delay used in [54] combined soft and hard deadlines with a ready time for each task. The SCAR scenarios under consideration in this thesis only have a soft deadline, yielding a marginal cost of delay of:

$$\phi_i = \exp\left(-\frac{\max(0, t_{d,i} - t_{b,j,i})}{k\overline{t_{b,j}}}\right) \tag{6.4}$$

where $t_{b,j,i}$ is the time at which the replenishment agent begins replenishing the user agent, $\overline{t_{b,j}}$ is the average start time for all replenishment tasks for that replenishment agent, $t_{d,i}$ is the deadline for the task, and k is a scaling factor. The scaling factor biases the behaviour of the ATC heuristic towards the WSPT rule if k is very large, and towards the MS rule if k is very small. Typical values of k range between 1 and 7. It should be noted that the deadline used by Kaplan and Rabadi [54] is the time by which the task must be completed, whereas the deadline in a SCAR scenario is the time before which the replenishment agent must begin replenishing the user agent. The deadline is calculated in the same way as the greedy heuristic using Equation 6.2, while the start and processing times are calculated using the mean parameter values in the equations on line 15 and line 24 respectively in Algorithm 5.1.

Multi-replenishment agent scenarios are treated similarly to the greedy heuristic—the actions of the other replenishment agents are simulated using the framework in Section 5.1 with the assumption that the replenishment agents already assigned to user agents will arrive before the replenishment agent under consideration. In some cases this will not be a valid assumption. However, it is often desirable to avoid sending multiple replenishment agents to the same user agent, and this assumption helps bias the heuristic away from assignments where this is not the case.

6.2.3 Stochastic Apparent Tardiness Cost Heuristic

In an attempt to incorporate uncertainty into the ATC heuristic, the analytical framework developed in Chapter 5 was used to create the Stochastic Apparent Tardiness Cost (SATC) heuristic. This is a different approach to that of Reddy and Narendran [84] and Arnaout et al. [8] who simply used deterministic heuristics in uncertain scenarios, ignoring any uncertainty. For the SATC heuristic, uncertainty is incorporated into the marginal cost of delay function as follows:

$$\phi_i = \exp\left(-\frac{\int_0^\infty (T_{d,i} - T_{b,j,i})\mathrm{d}t}{k\overline{t_{b,j}}}\right) \tag{6.5}$$

where $T_{b,j,i}$ and $T_{d,i}$ are calculated as per line 9 and line 10 respectively in Algorithm 5.2, $\overline{t_{b,j}}$ is calculated using the mean values for each $T_{b,j,i}$ calculated, and the integral is calculated using Equation 5.16. The rest of the heuristic remains the same as the deterministic version in Section 6.2.2.

6.3 Meta-Heuristic

This section describes the Simulated Annealing (SA) method that was used by Kaplan and Rabadi [54, 55]. SA is a global search algorithm that was inspired by the heating and cooling process used to anneal metals. It makes random moves to neighbour solutions by replacing one randomly selected task in the schedule with one of the other possible tasks, and it accepts worse solutions with a decreasing probability over time to prevent the algorithm from being trapped in local optima. Algorithm 6.2 outlines the general SA algorithm used in this thesis.

An initial schedule is generated using the ATC heuristic on line 1, and the cost of this initial schedule is calculated on line 2. The memory is then initialised with the initial schedule and cost (line 3) and the initial temperature is calculated (line 5). Then, while the number of iterations does not exceed the maximum number of iterations

Algorithm 6.2: Simulated annealing

`SA(`$\boldsymbol{\psi}, a, k, \alpha, i_{max}, j_{max}$`)`
input : Current system state, $\boldsymbol{\psi}$; finite horizon, a; initial temperature coefficient, k; temperature cooling coefficient, α; maximum number of iterations, i_{max}; maximum number of inner loop iterations, j_{max}
output: Best schedule, $\boldsymbol{\theta}_{best}$

1 Get an initial schedule, $\boldsymbol{\theta}$, of a tasks using the ATC heuristic `ATC(`$\boldsymbol{\psi}, a$`)`
2 $\omega \leftarrow$ `CostCalc(`$\boldsymbol{\psi}, \boldsymbol{\theta}$`)`
3 $\boldsymbol{M} \leftarrow \{\boldsymbol{\theta}, \omega\}$ `// initialise the memory with the initial schedule and cost`
4 $i \leftarrow 0$ `// initialise the number of iterations`
5 $T \leftarrow k \times \omega$ `// calculate the initial temperature by multiplying the cost by the initial temperature coefficient`
6 **while** $i < i_{max}$ **do** `// while less than the maximum number of iterations`
7 $\quad j \leftarrow 0$ `// initialise the number of inner loop iterations`
8 $\quad$ **while** $j < j_{max}$ **do** `// while less than the maximum number of inner loop iterations`
9 $\quad\quad \boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$ `// duplicate the schedule`
10 $\quad\quad$ Replace a randomly selected task in $\boldsymbol{\theta}'$ with another valid task
11 $\quad\quad \omega' \leftarrow$ `CostCalc(`$\boldsymbol{\psi}, \boldsymbol{\theta}'$`)`
12 $\quad\quad$ **if** $\omega' < \omega$ *or* `Rand(0,1)` $\leq \exp((\omega' - \omega)/T)$ **then**
13 $\quad\quad\quad$ Add the schedule and cost to the memory, $\boldsymbol{M} \leftarrow \{\boldsymbol{\theta}', \omega'\}$
14 $\quad\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}', \omega \leftarrow \omega'$
15 $\quad\quad i \leftarrow i + 1, j \leftarrow j + t$
16 $\quad T \leftarrow \alpha \times T$ `// update the temperature`
17 $\boldsymbol{\theta}_{best} \leftarrow$ lowest cost schedule in $\boldsymbol{M}$

(line 6), and the number of inner loop iterations does not exceed the maximum number of inner loop iterations (line 8), neighbour schedules are generated (line 10) and costed (line 11). Neighbour schedules are generated by replacing one task in the schedule with one of the other possible tasks, ensuring that the tasks either side are not the same as the new task.

If the cost of the new schedule is lower than the previously added schedule, or the probabilistic test is satisfied (line 12), the new schedule is added to the memory (line 13) and the current schedule and cost are updated (line 14). The iterators are incremented on line 15 and the temperature is updated on line 16. Finally, the lowest

cost schedule stored in memory is returned on line 17.

In [54], the finite horizon was simply the total number of tasks, as none of the tasks were repeated. For SCAR scenarios, the choice of finite horizons is slightly more complicated. There are two main options of finite horizon—specifying the number of tasks in the schedule, and specifying the execution time of the schedule. The difficulty with using a finite horizon based on the execution time with the simulated annealing method is that the various tasks may take different lengths of time to be executed. When one task is replaced with another, it can lead to a schedule which now no longer reaches the finite horizon. This then requires methods of selecting a new task or tasks at the end of the now incomplete schedule. Using a finite horizon based on the number of tasks in the schedule, on the other hand, can never result in an incomplete schedule. While this does result in schedules which take varying lengths of time to execute, the ratio cost defined in Equation 5.41 facilitates comparison of the schedules.

6.4 Branch and Bound

This section introduces branch and bound methods for both single-replenishment agent and multi-replenishment agent SCAR scenarios. Branch and bound is a depth-first tree search optimisation method, first developed by Land and Doig [62], which minimises the search space that is explored by ignoring branches where the lower bound on the cost is higher than the current best cost. One of the desirable characteristics of branch and bound is that it is an anytime optimisation method—a valid but likely sub-optimal schedule is available at any point during the optimisation process.

The set of all possible schedules for a given finite horizon can be formed as a tree (see Figure 6.2) where each node of the tree is a system state and each edge represents the tasks that can be performed from that system state. Along each edge of the tree, the evolution of the system state is simulated using one of the frameworks presented in Chapter 5. If the predicted resource level of the replenishment agents is below a

threshold, the only valid task is to return to the replenishment point. Similarly, the set of valid tasks excludes the task previously performed by that agent. This helps to reduce the size of the tree so that unnecessary tasks are not considered.

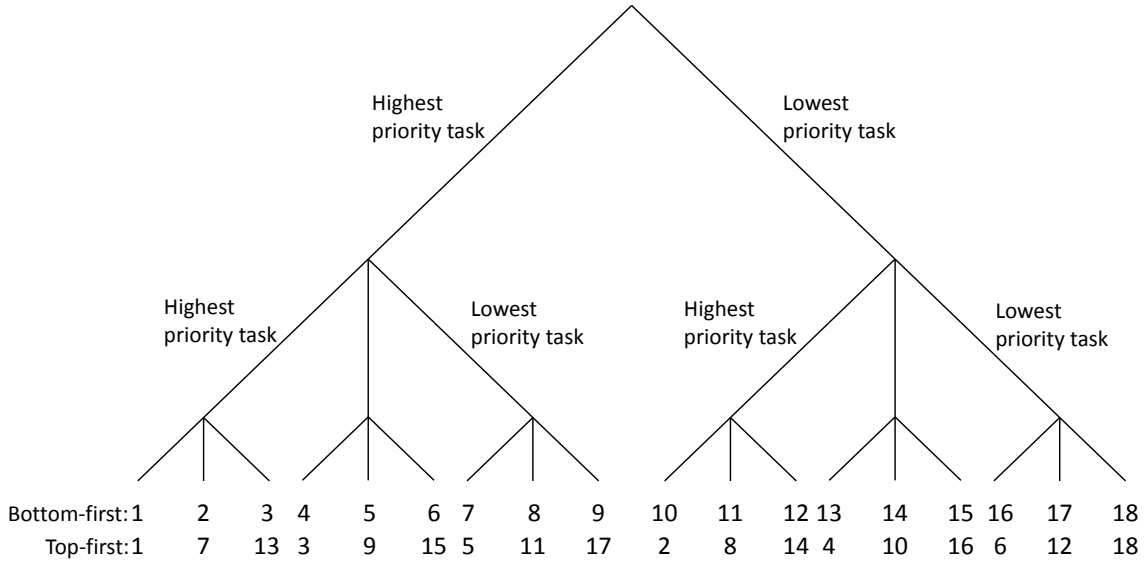


Figure 6.2 – An example tree of solutions—each node is a system state, and each edge is a different task. The evolution of the system is simulated along each edge. The numbers at the bottom of the tree outline two different approaches for searching through a tree. The first line shows the exploration order using the bottom-first approach, while the second line shows the exploration order using the top-first approach. The leftmost branch at each node is the highest priority (as calculated by a heuristic), and the rightmost is the lowest priority.

The order in which the branches of the tree are searched has a significant impact on the characteristics of the branch and bound algorithm. Two different approaches, shown in Figure 6.2, were examined. The first approach, bottom-first, searches through the leaves of one branch first before gradually searching higher in the tree. This approach has the advantage of minimising the amount of data that has to be stored in the tree as only the parent nodes of the current schedule under consideration must be stored. This gives a memory complexity of $\mathcal{O}(an)$, where a is the number of tasks in the schedule and n is the number of user agents. However, it has the disadvantage of focussing initial optimisation efforts on the later tasks in the schedule.

The other approach, top-first, explores the nodes in priority order with changes ini-

tially occurring at the start of the schedule in the top of the tree. Each successive solution examined is in a different high level branch to the previous solution, ensuring that the breadth of the tree is explored rapidly. This approach requires the calculated costs, states, and lower bounds of every node visited to be stored in a tree, which results in a memory complexity of $\mathcal{O}(n^a)$.

The cost of the current best schedule versus the number of nodes explored for the two methods is compared in Figure 6.3. Both approaches initially explore the same branches for the first schedule. However, where the bottom-first approach finds neighbour schedules which make minor incremental improvements to the cost, the top-first approach finds substantially better schedules in other branches of the tree. The bottom-first approach has many desirable characteristics for small optimisation problems—low memory usage and minimal computational overhead associated with having to search through the tree. In larger problems, however, it may be computationally intractable to search through the entire tree and the anytime characteristic of branch and bound must be exploited. In these cases, the top-first approach is more desirable as it generally finds lower cost schedules for the same number of nodes explored as the bottom-first approach. In addition, since it focusses on earlier tasks, it fits quite well into the MPC-like framework used in this thesis—optimisation efforts are focussed on the next tasks to be performed rather than the tasks at the end of the schedule.

6.4.1 Single-Agent Algorithm

The single-replenishment agent branch and bound algorithm is detailed in Algorithm 6.3. It takes as input the current state of the system, a finite horizon, and an optimisation depth. For the single-replenishment agent case, the finite horizon is the maximum number of tasks in the schedule to allow comparison with the simulated annealing method. The other input, the optimisation depth, determines how far down the tree the search goes. It uses the top-first approach to generate the tree of schedules on the fly, considering all possible combinations of the tasks within

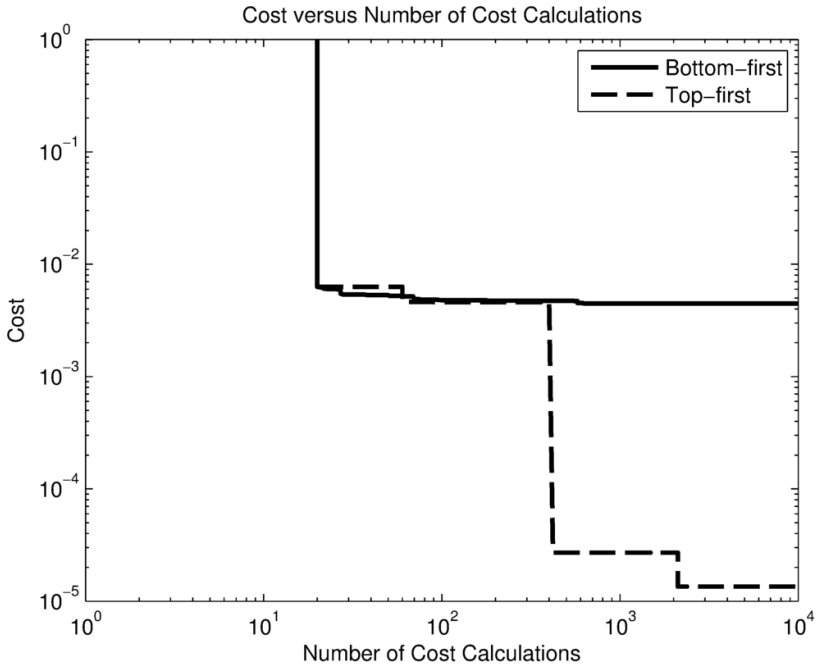


Figure 6.3 – A comparison of the two different approaches for searching through a tree. The top-first approach finds lower cost solutions in fewer iterations than the bottom-first approach.

the optimisation depth and selecting the remaining tasks using a heuristic. This is illustrated in Figure 6.4. If the optimisation depth is equal to the finite horizon, the algorithm will find the optimal schedule for that finite horizon.

Algorithm 6.3: Single-replenishment agent branch and bound

```
   BBSingle(ψ_initial, a, d_max)
   input  : Current system state, ψ_initial; finite horizon (number of tasks), a;
            optimisation depth, d_max
   output: Best schedule, θ_best
 1 θ_initial ← empty, θ_best ← empty, ω_best ← ∞
 2 T ← empty                          // initialise the tree
 3 for all possible combinations of the first d_max tasks do
 4 |   d ← 0, θ ← empty, ψ ← ψ_initial
 5 |   while d < a do
 6 |   |   p ← ATC(ψ)                 // array of tasks in priority order
 7 |   |   if d < d_max then
 8 |   |   |   b ← next branch to explore
 9 |   |   |   append p_b to θ_1
   |   |   else
10 |   |   |   append p_1 to θ_1
11 |   |   if T(θ) does not exist then
12 |   |   |   (ω, ψ) ← CostCalc(ψ_initial, θ)
13 |   |   |   T(θ) ← (ω, ψ)    // store the cost and state in the tree
14 |   |   d ← d + 1
15 |   |   if LowerBound(T, θ) > ω_best then
16 |   |   |   break
17 |   if LowerBound(T, θ) > ω_best then
18 |   |   continue
19 |   if ω < ω_best then
20 |   |   ω_best ← ω, θ_best ← θ
```

On line 6, the ATC heuristic is used to calculate a priority order for searching the branches at that node. If the level of the replenishment agent is below the level threshold, $l_{a,j,thresh}$, this priority order will only consist of the task of being replenished by the replenishment point. Otherwise, it will consist of all of the tasks in a priority order except for the last task performed. The ATC heuristic is used here to improve the search speed of the algorithm through the tree—higher priority tasks

are more likely to result in a lower cost schedule, reducing the total number of nodes explored to find the optimal schedule. If the current schedule has fewer tasks than the optimisation depth (line 7), the next branch to be explored is selected, otherwise only the highest priority branch is selected (line 10).

Since this algorithm is used within an MPC-like framework and only the first task in the schedule is used by the scheduler, it is also valid to terminate the optimisation process once all bar one of the branches from the root node have been eliminated. The one remaining branch is the first task in the optimal schedule, meaning that the later tasks do not need to be optimised.

As discussed previously, if low-cost schedules are found quickly within the optimisation, the branch and bound algorithm will end up exploring less branches to find the optimal solution than if it did not find the low-cost schedules early on. Therefore, a

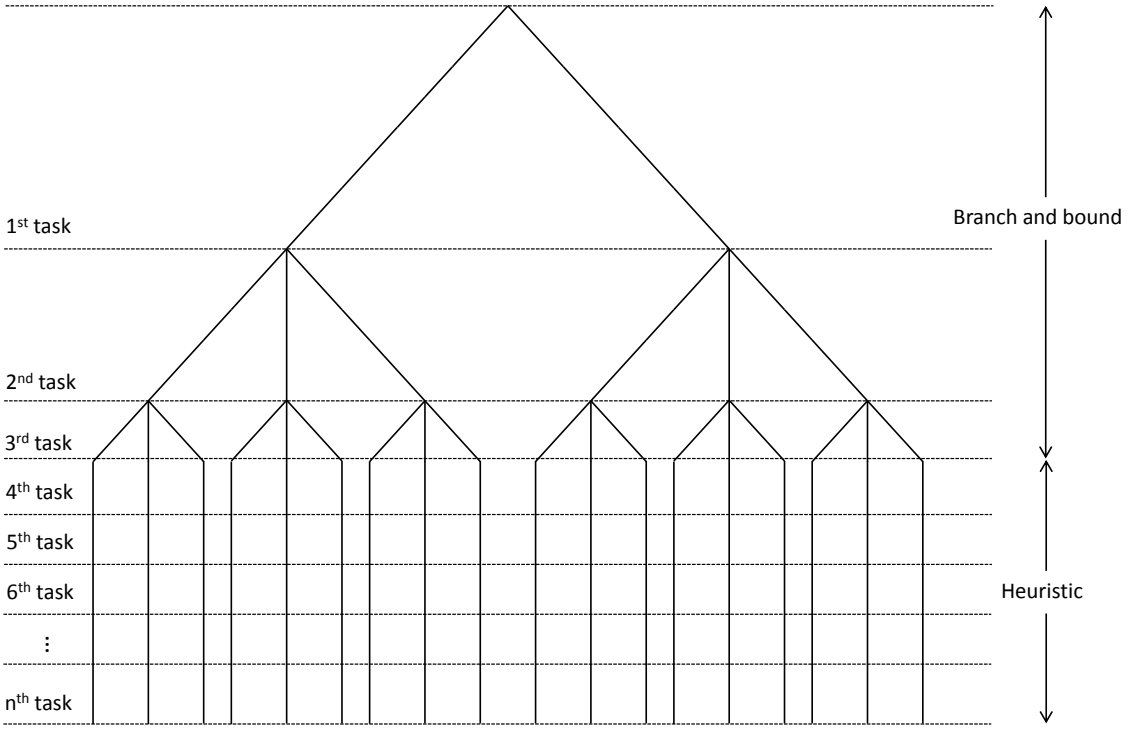


Figure 6.4 – Branch and bound when the optimisation depth is smaller than the schedule length. The optimisation depth in this example is three tasks—branch and bound optimises the first three tasks in the schedule, and the remaining tasks in the schedule are selected using a heuristic.

good initial schedule can significantly reduce the search time. The ATC heuristic used on line 6 goes some way towards guiding the initial search towards a good schedule. Another option is to use the schedule calculated for the previously performed task as a seed solution, with the first task removed and a final task added using the ATC heuristic. If this approach is used, it is important to perform a verification on the initial schedule to ensure that the resource level of the replenishment agent is above the level threshold whenever the task is to replenish a user agent.

The most important part of branch and bound is estimating the lower bound on the cost for each node (line 15). This lower bound represents the lowest possible cost of a complete schedule starting with the sequence of tasks described by that node. The more accurate the estimate of the lower bound is, the more branches can be pruned, resulting in a faster search time. A naïve heuristic that assumes the maximum possible time is taken to complete the schedule, as well as assuming that user agents incur no additional downtime to the finite horizon of the schedule, has been used for this thesis. This guarantees that the estimated minimum cost of a schedule is below the actual cost.

When calculating the lower bound, it is important that the time taken to complete the schedule, t_{total}, is not underestimated as it is on the denominator of the cost ratio defined in Equation 5.41. The ratio cost is reproduced here for convenience:

$$\arg\min \phi = \frac{\sum_{i=1}^{n} w_i t_{empty,i}}{n t_{total}} \tag{6.6}$$

The maximum value of t_{total} can be calculated by using the longest time each task can take when following on from a previous task to find the sequence of tasks which will take the longest time. The longest time each task can take is calculated by assuming that the user agent or replenishment agent is completely empty. Selecting the sequence of tasks cannot be performed in a greedy manner as this can underestimate the maximum schedule time. Instead, combinatorial optimisation is used to calculate the maximum time. While a method like branch and bound could be used to calculate

the maximum time in each instance, it is more efficient to use dynamic programming to precompute a lookup table of the maximum times to complete a schedule based on the previous task and the number of tasks to be selected.

A standard recursive dynamic programming approach similar to that used in [46] has been used in the single-replenishment agent case for precomputing the maximum times. Consider the example maximum task times shown in Table 6.1. The dynamic programming approach is illustrated in Table 6.2. It first calculates the single longest task to perform after each of the tasks. If the last task was task 0, for example, then it compares the next possible tasks of 1 and 2. Task 1 is selected as it has a total time of 150s compared to the time of 100s for task 2. This is performed for all of the tasks. In the next step, the best pair of tasks to be performed is determined. Again, if the last task was task 0, the two possible next tasks are task 1 and task 2, and the four possible pairs of tasks are [1,0], [1,2], [2,0], and [2,1]. From the first stage, it is already known that the optimal task to perform after performing task 1 is task 2, and the optimal task to perform after performing task 2 is task 0. Therefore, the four possible pairs of tasks are reduced to just [1,2] and [2,0]. This process is repeated up to the finite horizon of the schedule. Using this process means that, even as the number of tasks is increased, there are still only two options that need to be considered for each task in each step as the other options have already been shown to be suboptimal.

Table 6.1 – Example maximum task times in seconds

Previous task	0	1	2
Next task			
0	-	100	300
1	150	-	150
2	100	200	-

Table 6.2 – Dynamic programming example showing the next tasks and total time (in parentheses) for the tasks

Number of tasks	1	2	3
Previous task			
0	**1 (150)**	1,2 (350)	**1,2,0 (650)**
	2 (100)	**2,0 (400)**	2,0,1 (550)
1	0 (100)	0,1 (250)	0,2,0 (500)
	2 (200)	**2,0 (500)**	**2,0,1 (650)**
2	**0 (300)**	**0,1 (450)**	**0,2,0 (700)**
	1 (150)	1,2 (350)	1,2,0 (650)

6.4.2 Multi-Agent Algorithms

This section introduces two branch and bound methods for multi-replenishment agent scenarios. The first method, outlined in Algorithm 6.4, is similar to the branch and bound algorithm outlined for single-replenishment agent scenarios. The second method avoids selecting tasks such that replenishment agents arrive at similar times to one another, allowing the cost calculation detailed in Algorithm 5.8 to be used.

For the multi-replenishment agent case it is necessary to use a finite horizon based on the total execution time of the schedule to ensure that the schedules of each individual replenishment agent take similar lengths of time. Using a finite horizon based on the number of tasks could result in a schedule where one replenishment agent takes significantly longer to complete the schedule compared to the other replenishment agents. The finite horizon used in Algorithm 6.4 is the time after which the last task in the schedule for each replenishment agent must finish. Calculating the lower bound on the cost of the schedule is simplified by this choice of finite horizon—instead of requiring a dynamic programming approach to estimate the maximum execution time of a partial schedule, the maximum time in this case can be estimated by assuming that the longest possible task starts at the finite horizon.

Algorithm 6.4 begins by inserting into the schedule any tasks that are currently being performed by the replenishment agents (line 5). It then updates the state with the

Algorithm 6.4: Multi-replenishment agent branch and bound

`BBMulti(`$\boldsymbol{\psi}_{initial}, t_{horizon}, d_{max}$`)`
input : Current system state, $\boldsymbol{\psi}_{initial}$; finite horizon (time), $t_{horizon}$; optimisation depth, d_{max}
output: Best schedule, $\boldsymbol{\theta}_{best}$

$\boldsymbol{\theta}_{initial} \leftarrow empty, \boldsymbol{\theta}_{best} \leftarrow empty, \omega_{best} \leftarrow \infty$
$\boldsymbol{T} \leftarrow empty$ `// initialise the tree`
for $j \in \{1, ..., m\}$ **do**
  if *replenishment agent j is currently setting up, replenishing a user agent, or packing up* **then**
    append task to $\boldsymbol{\theta}_{initial,j}$
$(\omega_{initial}, \boldsymbol{\psi}_{initial}) \leftarrow$ `CostCalc(`$\boldsymbol{\psi}_{initial}, \boldsymbol{\theta}_{initial}$`)`
$\boldsymbol{T}(\boldsymbol{\theta}_{initial}) \leftarrow (\omega_{initial}, \boldsymbol{\psi}_{initial})$ `// store the cost and state in the tree`
for *all possible combinations of the first* d_{max} *tasks* **do**
  $t \leftarrow 0, d \leftarrow 0, \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{initial}, \boldsymbol{\psi} \leftarrow \boldsymbol{\psi}_{initial}$
  while $t < t_{horizon}$ **do**
    $j \leftarrow$ earliest free replenishment agent
    $\boldsymbol{p} \leftarrow$ `ATC(`$\boldsymbol{\psi}, j$`)` `// array of tasks in priority order`
    if $d < d_{max}$ **then**
      $b \leftarrow$ next branch to explore
      append $\boldsymbol{p}_b$ to $\boldsymbol{\theta}_j$
    else
      append $\boldsymbol{p}_1$ to $\boldsymbol{\theta}_j$
    if $\boldsymbol{T}(\boldsymbol{\theta})$ *does not exist* **then**
      $(\omega, \boldsymbol{\psi}) \leftarrow$ `CostCalc(`$\boldsymbol{\psi}_{initial}, \boldsymbol{\theta}$`)`
      $\boldsymbol{T}(\boldsymbol{\theta}) \leftarrow (\omega, \boldsymbol{\psi})$ `// store the cost and state in the tree`
    else
      $(\omega, \boldsymbol{\psi}) \leftarrow \boldsymbol{T}(\boldsymbol{\theta})$
    $d \leftarrow d + 1, t \leftarrow$ `GetMinTime(`$\boldsymbol{\psi}$`)`
    if `LowerBound(`$\boldsymbol{T}, \boldsymbol{\theta}$`)` $> \omega_{best}$ **then**
      break
  if `LowerBound(`$\boldsymbol{T}, \boldsymbol{\theta}$`)` $> \omega_{best}$ **then**
    continue
  if $\omega < \omega_{best}$ **then**
    $\omega_{best} \leftarrow \omega, \boldsymbol{\theta}_{best} \leftarrow \boldsymbol{\theta}$

initial schedule (line 6) and stores this in the tree (line 7). Then, at each node in the tree, the algorithm selects a task for the replenishment agent that is expected to finish its previous task first (line 11) and calculates whether it is possible for that partial schedule to lead to a lower cost schedule than the current best schedule (line 22). It follows this process until the lowest cost schedule is found.

The second branch and bound method follows the exact same structure as Algorithm 6.4 with one key difference—the `ATC` function on line 12 is modified to only return tasks where the replenishment agent will not arrive at a user agent at a similar time to any of the other replenishment agents. The cost calculations on line 6 and line 18 are then executed using the `MultiCostCalcGreedy` method outlined in Algorithm 5.8. The reasoning behind this approach is that it should reduce the computational impact of adding replenishment agents to the scenarios by avoiding the consideration of multiple arrival orders. This should not compromise the performance of the algorithm as replenishment agents are allocated to user agents that are not being serviced by another replenishment agent.

6.5 Results

This section compares the performance of the heuristic, meta-heuristic, and branch and bound optimisation methods in the single-replenishment and multi-replenishment agent scenarios that are outlined in Section 5.4.1. Section 6.5.1 presents the results for Scenarios S1 and S2, and Section 6.5.2 presents the results for Scenarios M1 and M2. Each method was tested 50 times in each scenario. All methods were implemented in Python and calculation times were calculated on a 2.8GHz Intel i7-640M.

6.5.1 Single-Agent Scenarios

The algorithms that were tested in the single-replenishment agent scenarios are:

- Greedy heuristic, G
- Deterministic ATC heuristic, ATC
- Stochastic ATC heuristic, SATC
- Simulated annealing using a cost function that incorporates uncertainty, SA
- Branch and bound using a cost function ignoring uncertainty, DBB
- Branch and bound using a cost function that incorporates uncertainty, SBB

The cost function used in the DBB method assumes that there is no uncertainty in the parameters of the agents and uses the mean values of the agent parameters. The SA and SBB methods both use the analytical cost calculation method developed in Section 5.2.

Of all of the optimisation methods examined, the G heuristic is the only method which does not take any parameters as an input. The ATC and SATC methods both have a scaling parameter that modifies the behaviour. As the SA method uses the ATC heuristic to generate the initial schedule, it also takes as input the scaling parameter,

along with the schedule length, the initial temperature coefficient, temperature cooling coefficient, maximum number of inner loop iterations, and maximum number of iterations. Kaplan and Rabadi [54] give suggestions for these parameters. An initial temperature coefficient of 50, a temperature cooling coefficient of 0.95, a maximum number of inner loop iterations of 15, and a maximum number of iterations of 2000 were found to give good results in the examined scenarios. Finally, the DBB and SBB methods take as input parameters the schedule length, the number of tasks to optimise, and the scaling parameter for the ATC heuristic.

Scenario S1

The user agents in this scenario were each equipped with discrete sensors at 20%, 40%, 60% and 80% of capacity. Each sensor had an uncertainty standard deviation of 1% of the maximum capacity. The replenishment agent was assumed to have perfect sensing for both position and resource level. Each simulation lasted for 5 hours of simulated time, with the initial resource level of each agent initialised to a random value between 50% and 100% of maximum capacity to simulate realistic, in-progress resource levels.

The k scaling parameter for the ATC and SATC methods was first tuned by running multiple simulations with k values between 1 and 7. As shown in Figure 6.5, the best results were achieved using a k value of approximately 2.5 for the 4-user agent scenario, and approximately 5.5 for the 5- and 6-user agent scenarios. This means that the behaviour is biased more towards the MS rule than the WSPT rule for the 4-user agent scenario in comparison to the 5- and 6-agent scenarios.

The SA, DBB, and SBB methods were run using finite horizons of $n+1$, $n+2$, and $n+3$ tasks, where n is the number of user agents in the system. The number of tasks optimised in the DBB and SBB methods was equal to the schedule length for this scenario, meaning that the schedules that were returned by these algorithms were the optimal schedules given the finite horizon.

The results for the 4-user agent scenario are shown in Figure 6.6. The performance

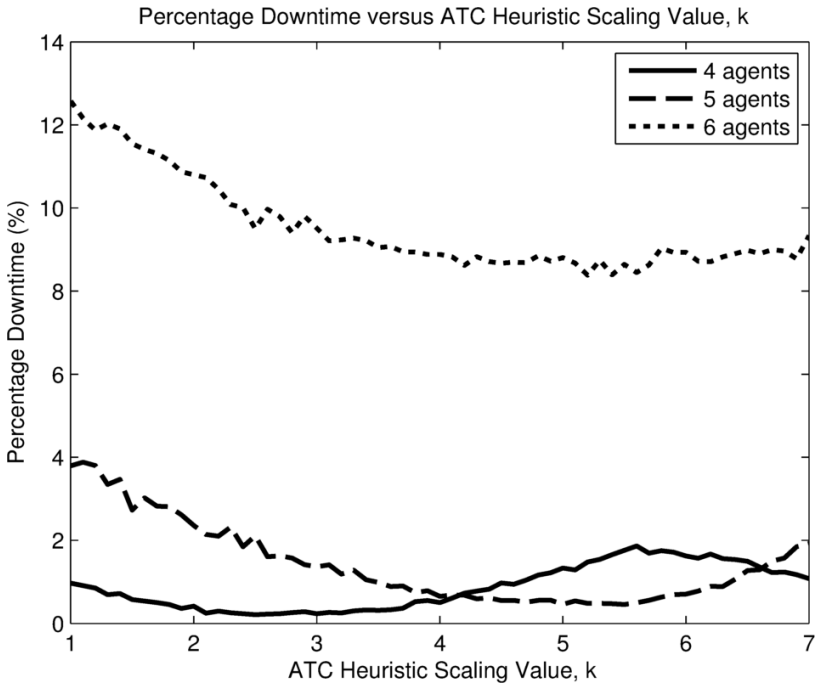


Figure 6.5 – Percentage downtime versus k value for the ATC heuristic in the 4-, 5-, and 6-user agent cases for Scenario S1.

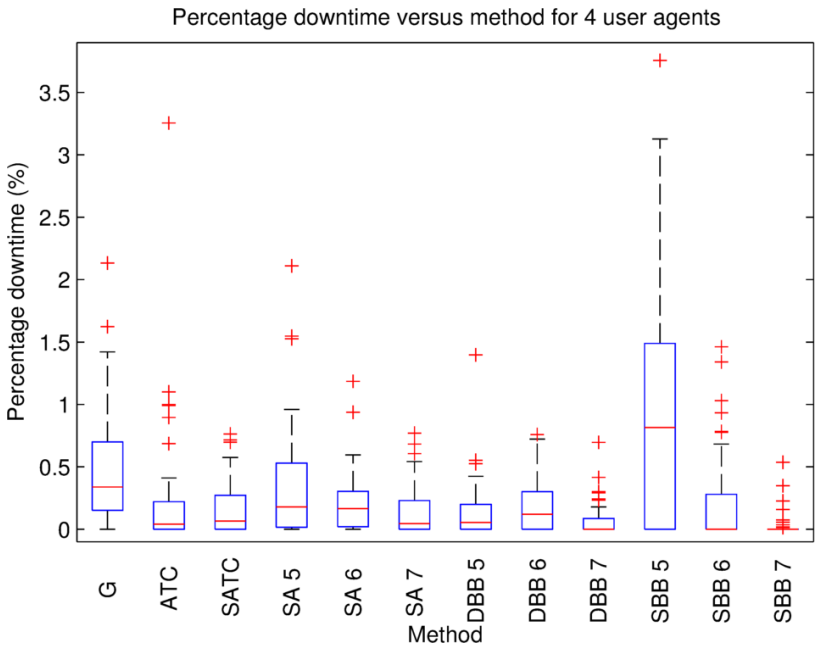
of the SA, DBB, and SBB methods improved as the finite horizon was increased. In the case of SBB, the improvement was substantial—when using a finite horizon of 5 tasks it actually had the worst performance of all of the methods. However, with a finite horizon of 7 tasks, the SBB method produced the overall best result. The poor performance of SBB when using the short finite horizon was due to the algorithm oscillating between two short tasks. This behaviour is demonstrated in Figure 6.7. When a longer finite horizon is used in Figure 6.7b, the oscillations are removed. This is side-effect of using the number of tasks in the schedule as the finite horizon. For a fixed number of tasks, the shorter the tasks are the less time there is for downtime to be accumulated. However, the short tasks are generally not good choices as they are typically oscillating between just two user agents that are close together, at the expense of the remaining user agents. As the finite horizon was increased, the algorithms were able to see the negative effects of choosing short tasks, resulting in no oscillations. DBB does not suffer as much from these oscillations as the initial schedule produced using the ATC heuristic in the DBB algorithm is usually

a zero-cost schedule that is returned immediately. The ATC and SATC heuristics outperformed the G heuristic, and actually produced comparable results to SA. This is an interesting result as it indicates that SA was unable to find a substantially better schedule than the initial schedule generated by ATC.

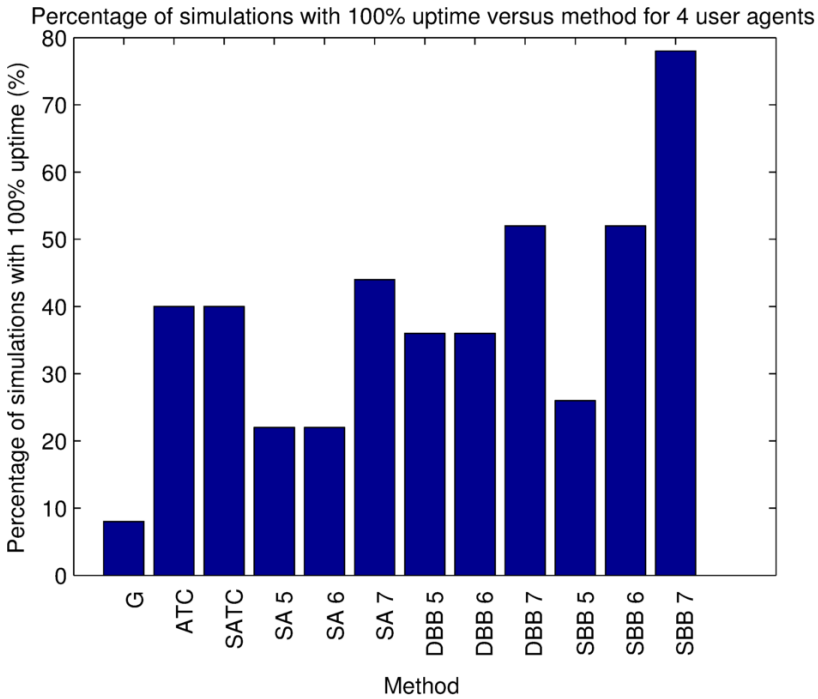
The 5-user agent scenario results, shown in Figure 6.8, follow the same trends as the 4-user agent scenario—the results for SA, DBB, and SBB improved as the finite horizon was increased, with SBB 8 producing the best results, particularly in terms of the number of simulations with 100% uptime. SA was outperformed by the branch and bound methods, and was not able to match the performance of the ATC heuristic. The G heuristic was again outperformed by the ATC and SATC heuristics, with significantly worse performance than in the previous scenario. The results for the 4- and 5-user agent scenarios clearly show the benefit of using an objective function that incorporates uncertainty over one that does not. As DBB is unable to differentiate between zero cost solutions, it instead relies on the priorities generated by the ATC heuristic to select a good task in cases where downtime is unlikely. The advantage of SBB over DBB is illustrated in Figure 6.9. SBB will find that selecting a schedule that will replenish the user agent at point a is significantly less risky than at point b, whereas DBB will return a zero cost for both schedules and is unable to differentiate between them. Therefore, point b may be chosen sometimes by the deterministic cost estimate, leading to an incurred cost when the actual resource level is as shown in Figure 6.9.

Finally, the results for the 6-user agent scenario are shown in Figure 6.10. In this scenario, none of the methods were able to prevent the user agents from incurring downtime. This scenario is very interesting—while the G heuristic has terrible performance, the remaining methods all have approximately the same performance. In particular, there is no noticeable difference between the DBB and SBB methods. This is to be expected as, in over-utilised scenarios like this, the objective function incorporating uncertainty will return an expected cost that is very close to the mean, and hence similar to that returned by an objective function that ignores uncertainty.

The calculation times for the methods are shown in Table 6.3. The heuristics have

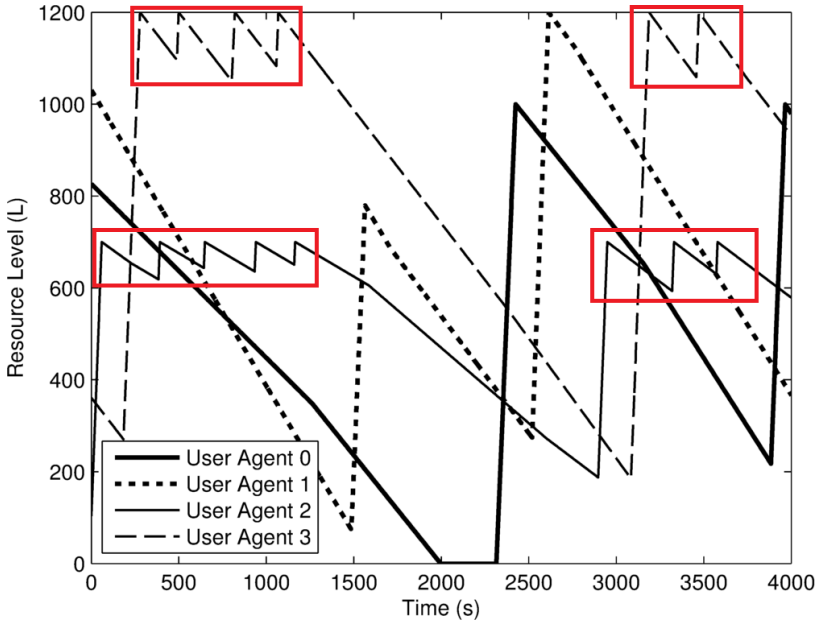


(a) Percentage downtime for each method

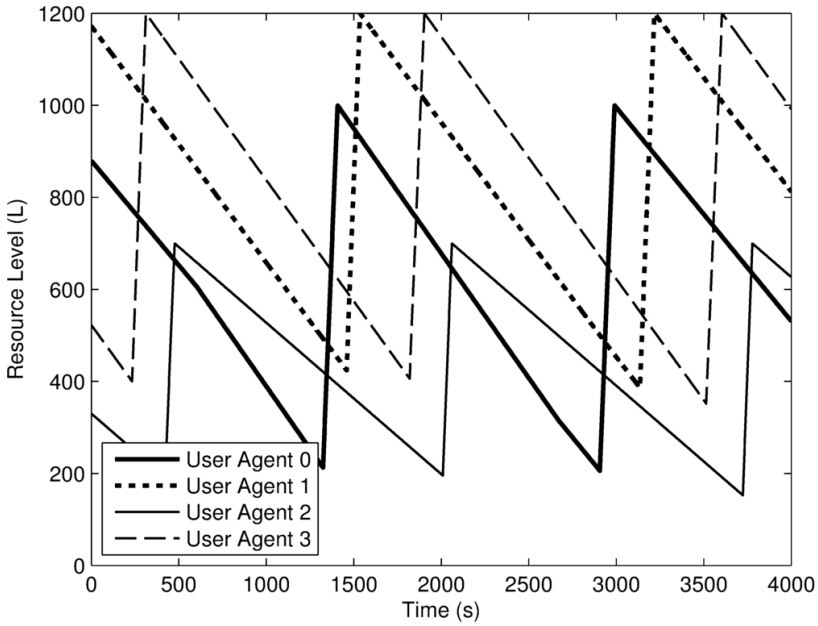


(b) Percentage of simulations with 100% uptime for each method

Figure 6.6 – (a) shows the box and whisker plots for the percentage downtime in Scenario S1 with 4 user agents. The number next to the SA, DBB, and SBB methods is the number of tasks in the schedule. (b) shows the percentage of simulation runs with 100% uptime for each method.

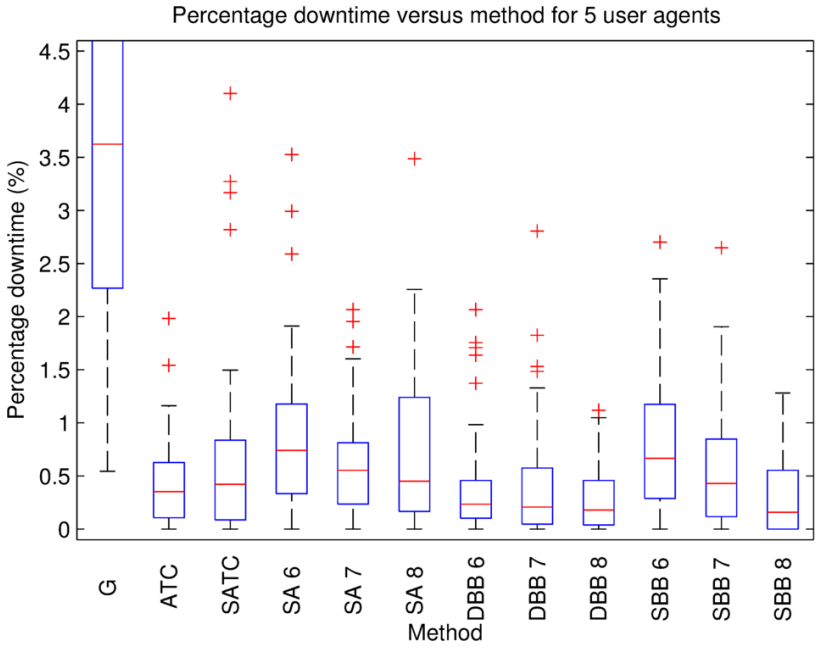


(a) Schedule length of 5 tasks

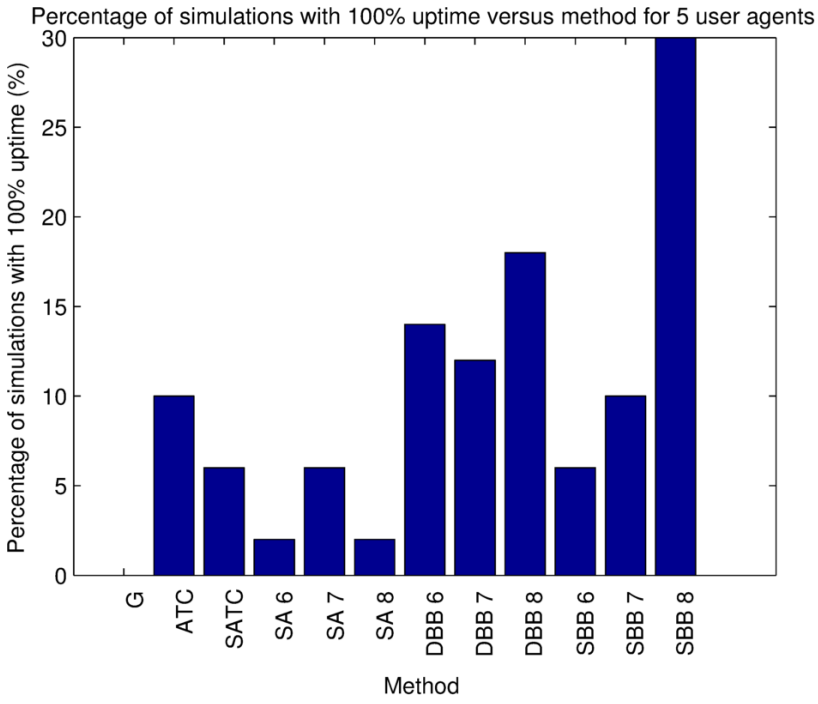


(b) Schedule length of 12 tasks

Figure 6.7 – Partial simulation results showing the user agent resource levels over time using the SBB method in Scenario S1 with 4 user agents. In (a), a schedule length of 5 tasks was used. Note the oscillations between the tasks of replenishing user agents 2 and 3, marked in red. In (b), a schedule length of 12 tasks was used and the oscillations between short tasks are no longer present.



(a) Percentage downtime for each method



(b) Percentage of simulations with 100% uptime for each method

Figure 6.8 – (a) shows the box and whisker plots for the percentage downtime in Scenario S1 with 5 user agents. The number next to the SA, DBB, and SBB methods is the number of tasks in the schedule. Note that some of the results for the G method have been cut-off to improve readability. (b) shows the percentage of simulation runs with 100% uptime for each method.

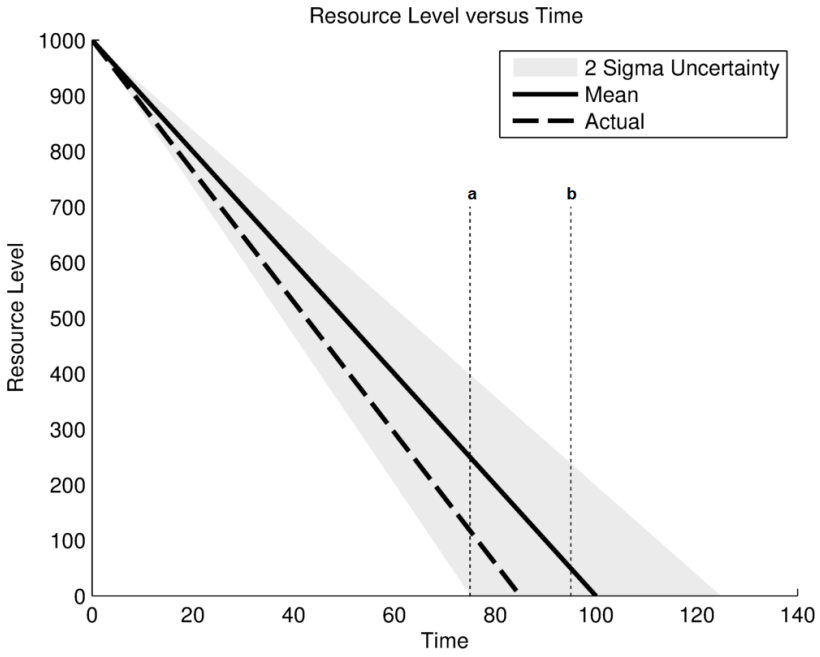


Figure 6.9 – Level of a user agent showing predicted level, uncertainty, and actual level. Replenishing at points a and b is treated identically if uncertainty is ignored. If uncertainty is considered, replenishing at point a is preferred over replenishing at point b.

the shortest calculation time, taking just a fraction of a second to generate a task allocation. The calculation time of SA slightly increases with both the number of user agents and the number of tasks in the schedule. The DBB and SBB methods show a much larger variation in the time taken. In the 4- and 5-user agent scenarios, DBB takes significantly less time than SBB. In these scenarios, the DBB algorithm generally finds a zero-cost schedule within the first few schedules tested and is able to return this schedule immediately without having to explore the rest of the tree. The SBB algorithm, on the other hand, will never find a zero-cost schedule and will only return a schedule once the entire tree has been sufficiently explored. In the 6-user agent case, the calculation times for DBB have increased significantly as it is highly unlikely that the DBB algorithm will find a zero-cost schedule. SBB still has a longer calculation time which is a direct consequence of the more computationally expensive prediction framework used.

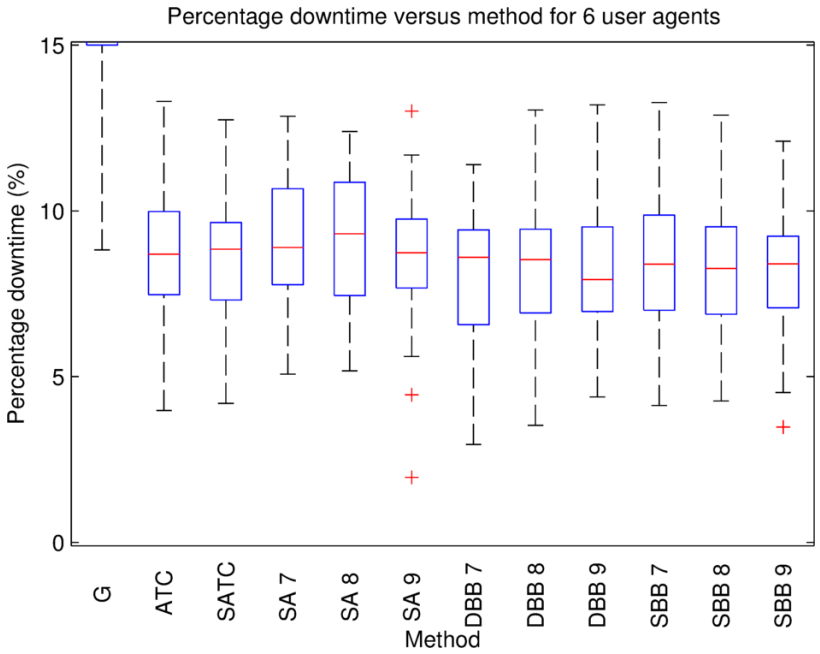


Figure 6.10 – Box and whisker plots for the percentage downtime in Scenario S1 with 6 user agents. The number next to the SA, DBB, and SBB methods is the number of tasks in the schedule. Note that some of the results for the G method have been cut-off to improve readability. None of the methods were able to prevent the user agents from exhausting their supply of the resource, so these results are omitted.

Table 6.3 – Calculation times in seconds for Scenario S1

Method	G	ATC	SATC	SA	SA	SA
Schedule Length	-	-	-	$n+1$	$n+2$	$n+3$
No. of User Agents, n						
4	4.55E-5	1.37E-4	3.08E-4	4.98	5.85	6.55
5	2.38E-5	1.48E-4	2.73E-4	6.54	7.62	7.96
6	5.00E-5	1.84E-4	3.20E-4	7.05	8.35	9.68
Method	DBB	DBB	DBB	SBB	SBB	SBB
Schedule Length	$n+1$	$n+2$	$n+3$	$n+1$	$n+2$	$n+3$
No. of User Agents, n						
4	1.70E-2	1.90E-2	2.80E-2	0.157	0.421	0.884
5	3.20E-2	4.90E-2	4.60E-2	0.605	1.12	2.51
6	1.51	2.06	7.46	8.11	20.2	37.8

Scenario S2

All of the agents in this scenario were assumed to have perfect sensors such that there is no uncertainty in the current state. The initial resource levels of all agents were initialised to a random value between 50% and 100% of maximum capacity, and the simulation lasted for 180 days of simulated time.

Before testing the optimisation methods, the ATC scaling parameter, k, was tuned. Values of $k = 3$ for the large and medium replenishment agents, and $k = 5$ for the small replenishment agent were found to give good results. Various schedule lengths were also tested. Figure 6.11 shows the number of simulations with 100% uptime for various schedule lengths using the SBB method with an optimisation depth of 2 tasks. As can be seen, the peak value is achieved at a schedule length of 25 tasks. At short schedule lengths, the performance is very poor due to oscillations from the preference for shorter tasks. Beyond 25 tasks, it is likely that the suboptimal decisions made by the ATC heuristic accumulate to such a point that the benefit of using a longer look-ahead distance is negated. SA, DBB, and SBB were all tested using a schedule length of 25 tasks. Finding the optimal schedule in this scenario is intractable and therefore the anytime nature of branch and bound was used to limit the runtime of the algorithms. This was achieved in two ways—the optimisation depth was varied, and a hard size limit of 10,000 nodes was placed on the solution tree. This limit of 10,000 nodes is used for all remaining scenarios.

Figure 6.12, Figure 6.13, and Figure 6.14 show the results for each method which broadly mirror those for Scenario S1. The benefit of the directed optimisation of the branch and bound methods on an initial schedule generated by the ATC heuristic is evident here. Even if only the first task is optimised, the branch and bound methods provided a significant decrease in downtime over the ATC heuristics. The SA method, on the other hand, struggled in these larger scenarios as it does not focus the optimisation on the earlier tasks in the schedule and was unable to sufficiently explore the search space to yield an improvement over the ATC heuristics.

The SATC heuristic outperformed the deterministic ATC heuristic in the medium and

small replenishment agent cases in terms of downtime, but was outperformed by the ATC heuristic in the large replenishment agent case. DBB was clearly outperformed by SBB in the large and medium replenishment agent cases, particularly in terms of percentage of scenarios with 100% uptime. Both methods had similar performance in the small replenishment agent case, corroborating the results obtained in Scenario S1.

The computation times for each optimisation method are detailed in Table 6.4. While the G, ATC, and SATC heuristics computed very quickly, the combinatorial optimisation methods took significantly longer. The results for the branch and bound methods mirror those for Scenario S1—DBB is generally very fast except in the small replenishment agent scenario where it is unable to quickly find a zero-cost schedule.

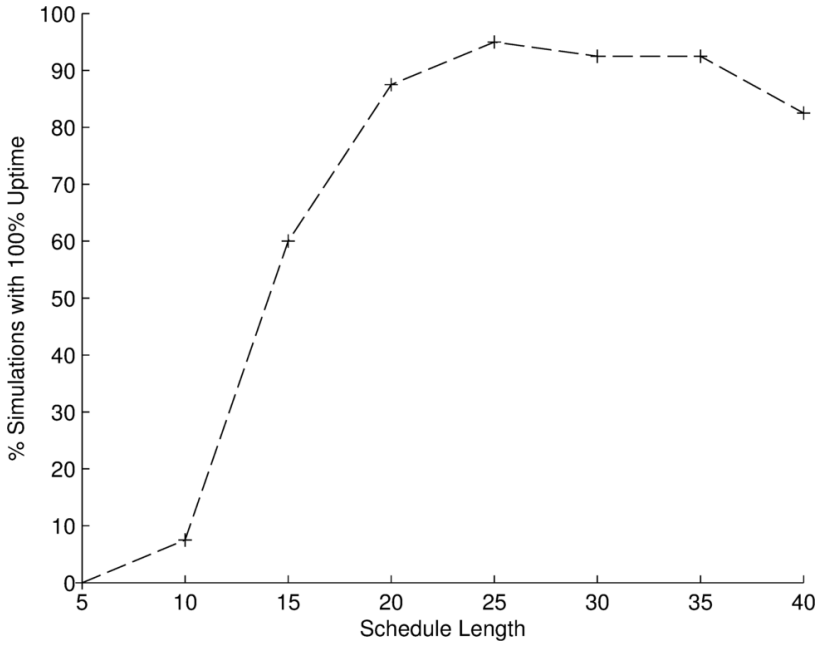
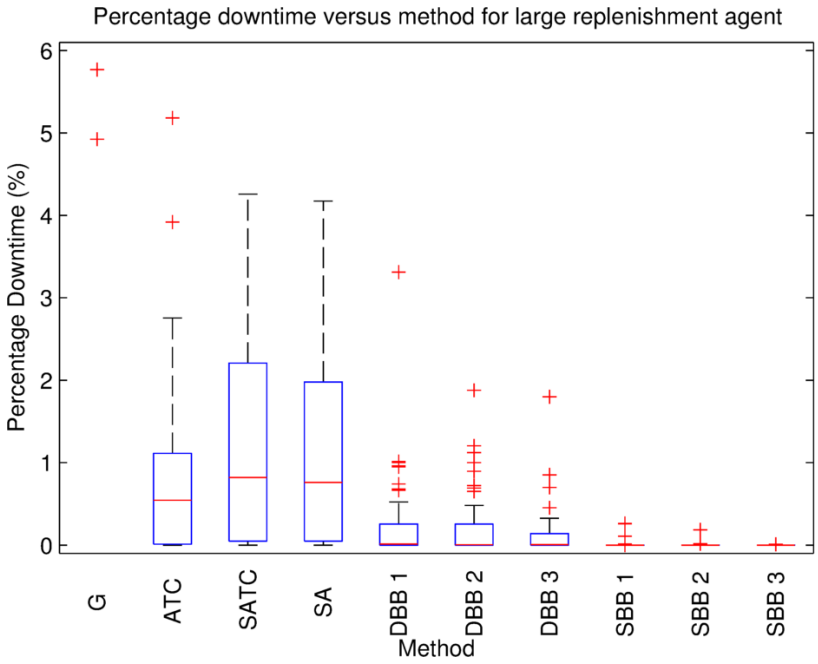
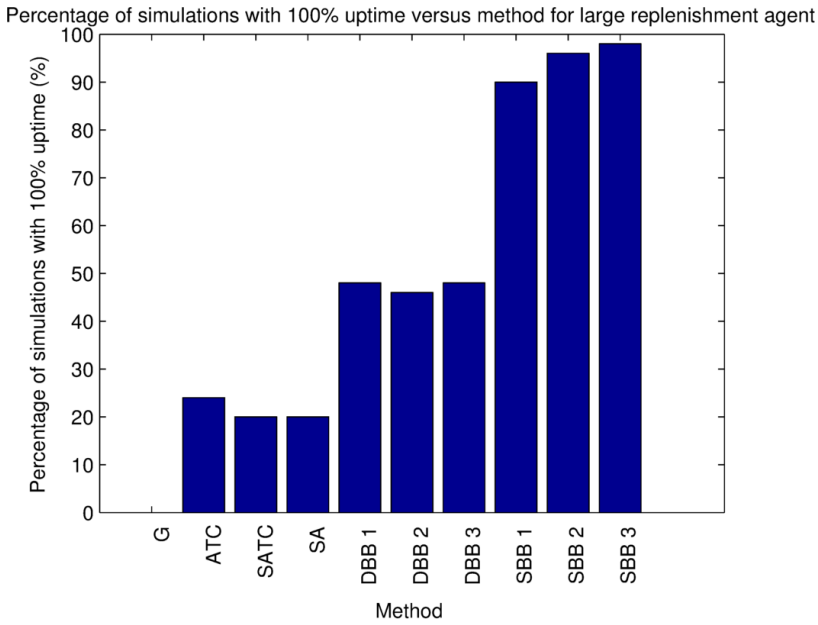


Figure 6.11 – Percentage of simulations with 100% uptime in Scenario S2 with the large replenishment agent for various schedule lengths using SBB with an optimisation depth of 2 tasks.

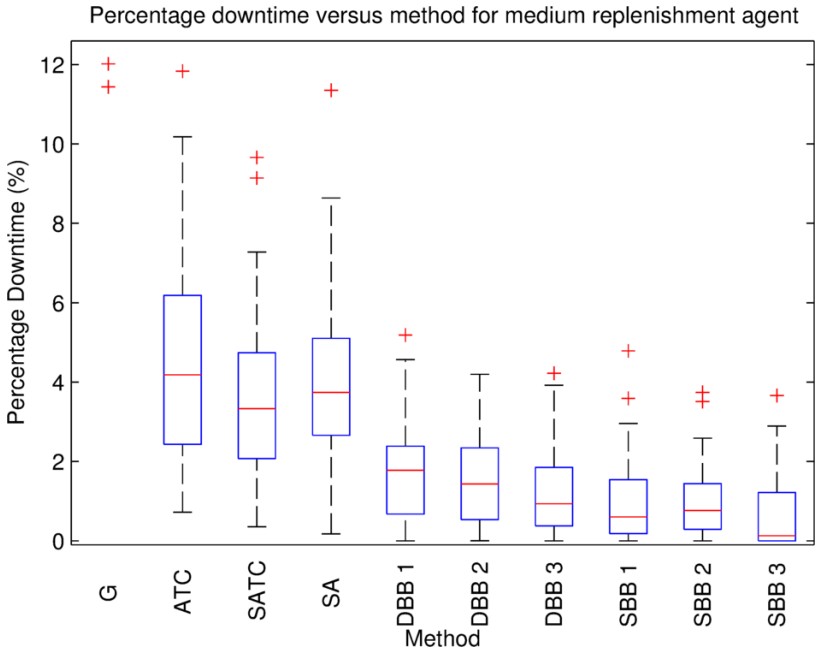


(a) Percentage downtime for each method

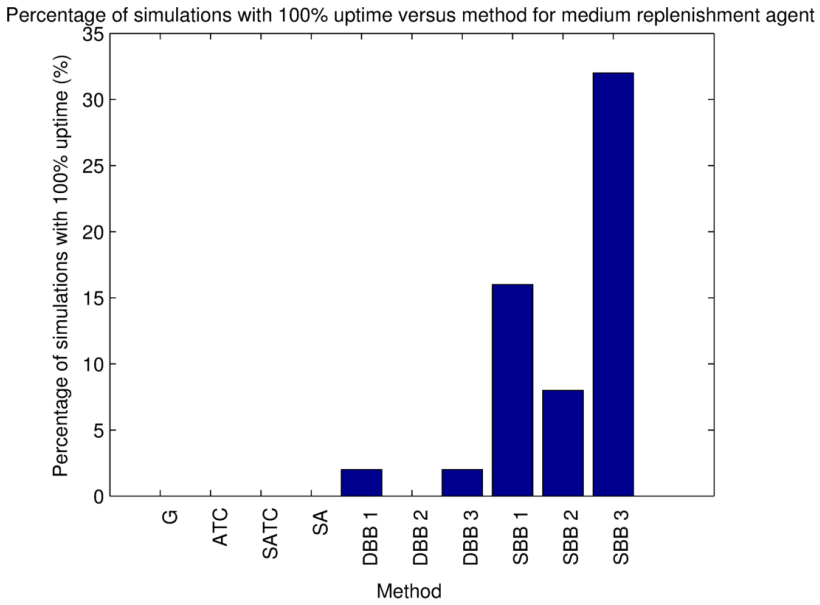


(b) Percentage of simulations with 100% uptime for each method

Figure 6.12 – (a) shows the box and whisker plots for the percentage downtime in Scenario S2 with the large replenishment agent. The number next to the DBB and SBB methods is the optimisation depth of the branch and bound algorithm. Note that some of the results for the G method have been cut-off to improve readability. (b) shows the percentage of simulation runs with 100% uptime for each method.



(a) Percentage downtime for each method



(b) Percentage of simulations with 100% uptime for each method

Figure 6.13 – (a) shows the box and whisker plots for the percentage downtime in Scenario S2 with the medium replenishment agent. The number next to the DBB and SBB methods is the optimisation depth of the branch and bound algorithm. Note that some of the results for the G method have been cut-off to improve readability. (b) shows the percentage of simulation runs with 100% uptime for each method.

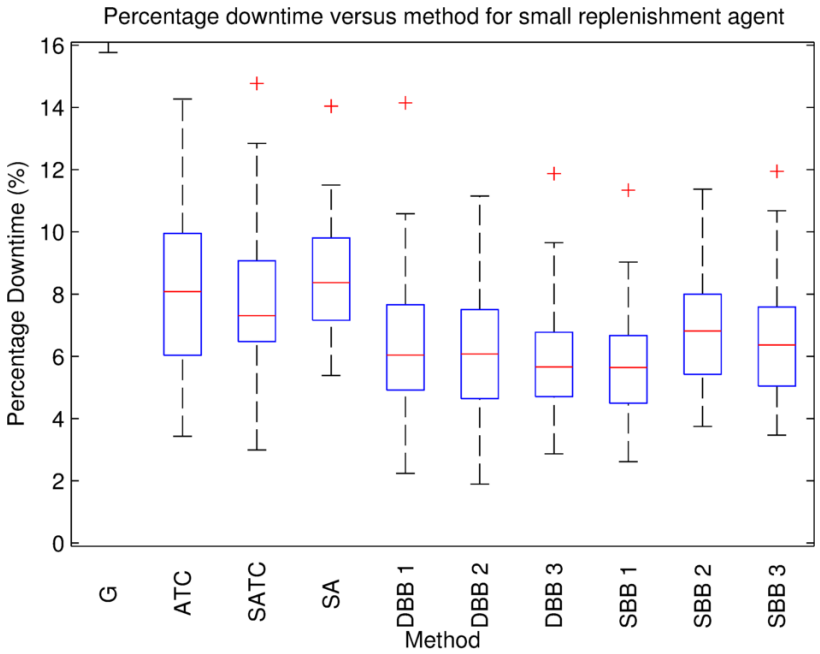


Figure 6.14 – Box and whisker plots for the percentage downtime in scenario S2 with the small replenishment agent. The number next to the DBB and SBB methods is the number of tasks in the schedule that are optimised. Note that the G results have been cut-off to improve readability. None of the methods were able to prevent the user agents from exhausting their supply of the resource, so these results are omitted.

Table 6.4 – Calculation times in seconds for Scenario S2

Method	G	ATC	SATC	SA		
Replenishment Agent						
Large	9.71E-4	1.61E-3	2.09E-3	13.5		
Medium	1.03E-3	1.50E-3	2.10E-3	13.9		
Small	1.03E-3	1.64E-3	2.00E-3	13.1		
Method	DBB 1	DBB 2	DBB 3	SBB 1	SBB 2	SBB 3
Replenishment Agent						
Large	0.664	0.674	0.668	1.68	11.9	12.2
Medium	0.671	0.639	0.638	1.88	21.7	39.5
Small	2.38	24.2	29.8	1.68	20.1	48.6

Single-Agent Summary

The above results clearly demonstrate the benefit of considering more than just the next task in the optimisation process. The heuristics consistently produced the worst performance, with the G heuristic performing significantly worse than the other heuristics. The SATC heuristic appeared to have an advantage over the deterministic ATC heuristic in some scenarios, but was outperformed in others. Overall, the benefit of incorporating uncertainty in the ATC heuristic appears to be minimal.

Surprisingly, the SA method was only able to match the performance of the ATC and SATC heuristics, despite using the ATC heuristic to generate the initial schedule. The branch and bound optimisations that focus on the earlier tasks in the schedule produced significantly better results for a similar computation time to the SA method. For the multi-replenishment agent scenarios, the G, SATC, and SA methods are no longer considered.

6.5.2 Multi-Agent Scenarios

The algorithms tested in the multi-replenishment agent scenarios were:

- Deterministic ATC heuristic, ATC
- Branch and bound using a cost function ignoring uncertainty, DBB
- Branch and bound using a cost function that incorporates uncertainty, SBB
- Branch and bound using a cost function that incorporates uncertainty and avoids sending replenishment agents to the same task at similar times, SBB NC

The schedule length for the DBB and SBB methods in these cases was based on time rather than the number of tasks. The SBB method was tested using γ values of 1, 0.1, and 0.01.

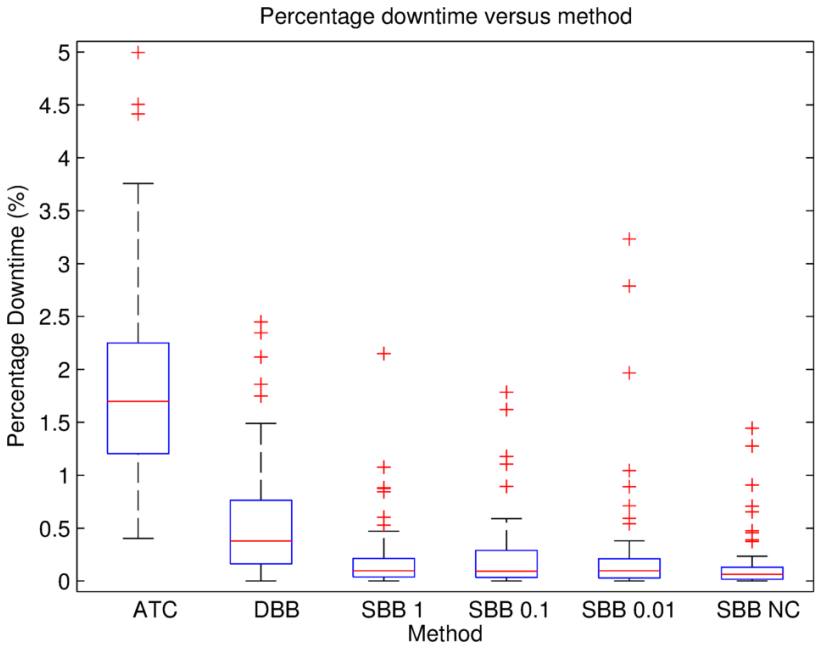
Scenario M1

The agents in this scenario had the same suite of sensors as the user agents in Scenario S1. Similarly, the simulations were run for 5 hours of simulated time with the initial resource levels of all agents initialised to a random value between 50% and 100% of maximum capacity. The ATC heuristic was used with a scaling value, k, of 3. A schedule horizon of 2000s, and an optimisation depth of 9 tasks, were used for the DBB and SBB methods. The results for this scenario are shown in Figure 6.15. These results follow the same trend as the single-replenishment agent scenarios—the ATC heuristic had the worst performance, followed by the DBB method, with the SBB methods producing the lowest and most consistent downtime results. Varying γ did not impact the results for the SBB method, and the SBB NC algorithm produced almost identical results to the full SBB algorithms. As can be seen in Figure 6.15b, the SBB methods again produced the best results in terms of percentage of simulations with 100% uptime.

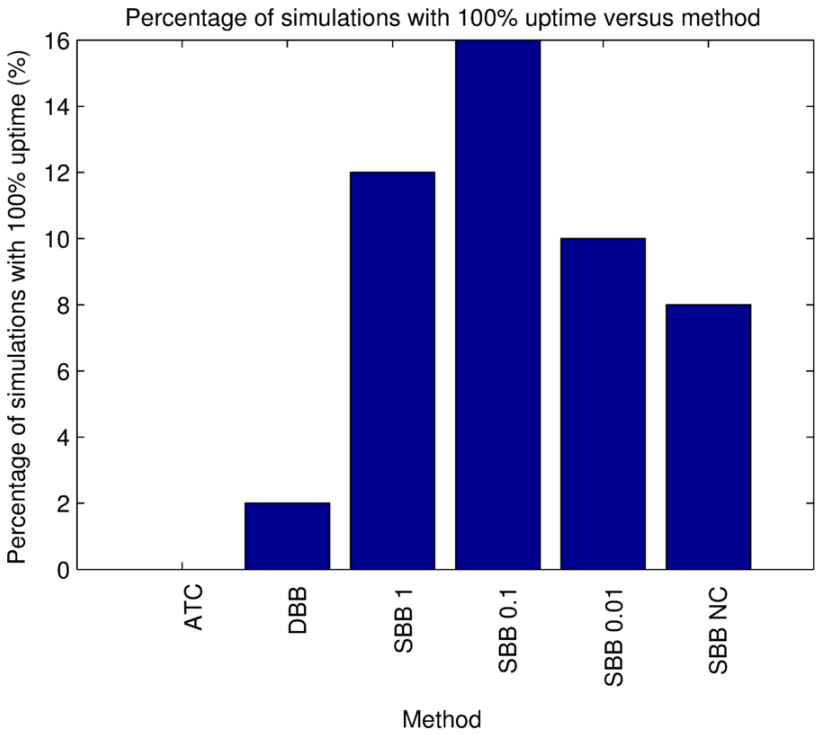
The calculation times for each method are detailed in Table 6.5. The ATC and DBB methods took the least amount of time, with the SBB algorithms all taking between 13 and 14 seconds. The significantly lower calculation time of the DBB method was due to the same reasons as discussed in Scenario S1—the algorithm was able to quickly find a zero-cost schedule and return this schedule immediately. As there were only two replenishment agents in this scenario, most schedules had only one or two likely arrival orders, resulting in minimal difference between the various SBB methods.

Table 6.5 – Calculation times in seconds for Scenario M1

ATC	DBB	SBB 1	SBB 0.1	SBB 0.01	SBB NC
1.46E-3	0.427	13.9	13.3	13.2	13.9



(a) Percentage downtime for each method



(b) Percentage of simulations with 100% uptime for each method

Figure 6.15 – (a) shows the box and whisker plots for the percentage downtime in Scenario M1. The number next to the SBB methods is the threshold multiplier value, γ. SBB NC refers to the multi-replenishment agent SBB algorithm that avoids sending replenishment agents to the same user agent at similar times. (b) shows the percentage of simulation runs with 100% uptime for each method.

Scenario M2

All of the agents in this scenario were assumed to have perfect sensors and each simulation was run for 1 hour of simulated time. The initial resource level of each replenishment agent was set to 100%, and the initial resource levels of the user agents was set to a random value between 50% and 100% of maximum capacity. Unlike in the other scenarios, two of the replenishment agents started at the replenishment point, with the remaining three each starting at a different user agent. An ATC scaling value of 2 was found to produce good performance. For the DBB and SBB methods, a schedule horizon of 400s and an optimisation depth of 5 tasks were used. Initially, the SBB NC algorithm would sometimes fail to find a valid schedule. The requirement for successive tasks of the replenishment agents to be different was relaxed slightly for the SBB NC method to enable it to find valid schedules—it was allowed to perform the task of being replenished by the replenishment point multiple times in a row. This highlights a potential failing of the SBB NC algorithm—it may be impossible to find a good schedule in scenarios with very high uncertainty and few user agents.

The results for Scenario M2 are shown in Figure 6.16. Interestingly, the ATC heuristic actually outperformed the DBB algorithm in terms of median downtime in this scenario. Varying the γ parameter had a small effect on the SBB results with the median downtime decreasing as γ was decreased. The SBB NC algorithm produced the best results, outperforming the other SBB methods by a significant margin. It was actually the only method to achieve runs with 100% uptime—6% of the simulations had 100% uptime.

Calculation times for the methods are detailed in Table 6.6. Here there is a large variation between the SBB methods, with the SBB NC approach taking far less time than the full SBB methods. In particular, the SBB approaches took significantly more time as γ was decreased and more arrival orders were considered. The intuitive strategy of avoiding replenishment agents arriving at the same time was clearly beneficial, both in terms of downtime and calculation time.

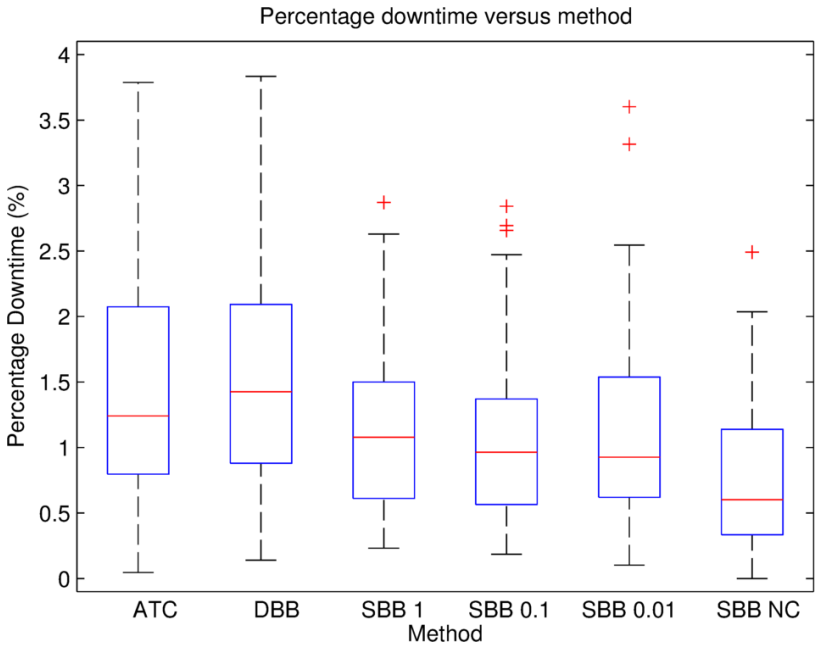


Figure 6.16 – Box and whisker plots for the percentage downtime for Scenario M2. The number after SBB is the threshold multiplier value, γ. SBB NC refers to the multi-replenishment agent SBB algorithm that avoids sending replenishment agents to the same user agent at similar times.

Table 6.6 – Calculation times in seconds for Scenario M2

ATC	DBB	SBB 1	SBB 0.1	SBB 0.01	SBB NC
2.32E-4	0.486	20.2	37.8	99.6	12.2

6.6 Summary

This chapter introduced and evaluated methods for optimising the actions of the replenishment agents in SCAR scenarios. After introducing the optimisation framework in Section 6.1, Section 6.2 presented several heuristics, including a new form of the ATC heuristic that attempted to incorporate some aspects of the uncertainty. Section 6.3 then introduced a SA meta-heuristic that had been successfully applied to similar scenarios in the literature. This method used the ATC heuristic to generate an initial schedule, and then performed random manipulations of the schedule to find a good schedule. Finally, several branch and bound approaches were developed

in Section 6.4. Branch and bound strategies which focussed search efforts on earlier tasks were introduced for both single-replenishment agent and multi-replenishment agent scenarios. In addition, the SBB NC branch and bound approach which actively avoids sending replenishment agents to the same task at similar times was detailed.

The above methods were evaluated in Section 6.5 using several single- and multi-replenishment agent scenarios. The branch and bound methods using the framework developed in Chapter 5 were shown to result in the lowest downtime. In particular, these methods consistently resulted in the highest number of simulations in which none of the user agents exhausted their supply of the resource. The branch and bound method which ignored uncertainty in its prediction framework had the next best performance, with SA and the heuristics following it. The undirected optimisation approach of SA struggled to find good solutions, particularly in scenarios with a large search space, while the heuristics made myopic decisions resulting in high downtime. In the multi-replenishment agent scenarios, the SBB NC approach was shown to outperform the other methods, highlighting the benefit of avoiding replenishment agents arriving at similar times.

The SBB algorithms, making use of the prediction framework from Chapter 5, were of most benefit in scenarios where the replenishment agents were on the cusp of being able to prevent the user agents from incurring downtime. In situations where they were completely unable to prevent the user agents from exhausting their supply of the resource, the branch and bound method that ignored uncertainty, DBB, produced similar results to the SBB algorithms. Similarly, in situations where there is very little risk of the user agents exhausting their supply of the resource there is unlikely to be any difference between the DBB and SBB methods, and even the ATC heuristic would likely be sufficient for preventing downtime.

The downside of the branch and bound approaches is that they are computationally expensive in comparison to the heuristics, particularly in large scenarios. Larger scenarios have not been investigated as they are computationally infeasible with this implementation, and the benefit of examining them is limited as the tested scenarios already demonstrate the value of the developed methods. In the multi-replenishment

agent scenarios, the calculation time of the prediction framework grows significantly as more orders of arrival are considered. As the SBB NC method does not suffer from this, it consequently offers the best computational performance of the branch and bound methods using the developed prediction framework.

All of the algorithms were implemented in Python and are currently unoptimised with many opportunities existing for improving their computational performance. The following optimisations are outside the scope of this thesis and are listed here as a guide to a practical implementation. The current implementation does not store the predicted state of the system at each node of the tree and instead calculates the cost at each node by simulating from the current state. By storing the predicted state at each node and only simulating the next task rather than the entire schedule up to that point could increase the speed by an estimated 5 times in large scenarios, with the speed-up increasing as the schedule length is increased. Using a language such as C could increase the speed of the algorithm by between 10 and 1000 times [16], and using parallel branch and bound algorithms can give a speed-up proportional to the number of processors used [61]. Parallel methods could also be easily applied to the multi-replenishment agent prediction framework. Using conservative estimates, a speed-up of at least 100 times is feasible, resulting in sub-second calculation times for all of the scenarios examined.

Chapter 7

Conclusion

The purpose of this thesis was to develop methods for scheduling under uncertainty in scenarios that are typical in robotic, agricultural, defence, and mining domains. The Stochastic Collection and Replenishment (SCAR) scenario was introduced to model these scenarios, and novel methods for the estimation, prediction, and optimisation problems that compose belief space scheduling were presented. Taking inspiration from the field of belief space planning, this thesis used probability distributions over the state of the system rather than individual states as is common in the scheduling literature. A key focus of this thesis was the development and use of analytical and approximation methods to address the excessive computational requirements of existing methods. The methods developed in this thesis resulted in a scheduling system that produced superior results to existing scheduling methods within a reasonable time frame.

This chapter presents a summary of the thesis in Section 7.1, followed in Section 7.2 by an overview of the contributions that have been made. Finally, Section 7.3 concludes by proposing avenues for future work.

7.1 Summary

Chapter 2 reviewed the current literature in the fields of scheduling and belief space planning. Existing methods of scheduling, particularly in collection and replenishment scenarios, were shown to ignore many of the characteristics of SCAR scenarios, the most significant being the uncertainty inherent in real-world scenarios. Literature on haul truck scheduling provided observations on the necessary characteristics of a practical scheduling system. Belief space planning methods were then reviewed. These methods were either prohibitively expensive in terms of computation time, or shown to make inappropriate assumptions for them to be applicable to the problem of scheduling in SCAR scenarios.

Chapter 3 presented a rigorous problem definition of the SCAR scenario. The problem definition covered the parameters and behaviour of the agents, and the objectives of the optimisation problem.

An analytical constrained Kalman Filter (KF) for uncertain constraints was developed in **Chapter 4** for estimating the state of the system. This method applies to constraints that are described by Gaussian distributions. Approximations for applying interval constraints were presented and were shown to be good approximations compared to numerical methods. This soft-constrained KF was then compared against unconstrained and hard-constrained KFs for tracking the resource level of an agent with several uncertain discrete sensors. When the sensors had no uncertainty, the hard- and soft-constrained KFs had identical results, outperforming the unconstrained KF by 14.5%. As the uncertainty of the sensors was increased, the tracking performance of the soft-constrained KF approached that of the unconstrained KF, outperforming it by only 3.9% when the uncertainty of the sensors was 2.5%. The performance of the soft-constrained KF relative to the hard-constrained KF increased as the sensor uncertainty was increased, with it outperforming the hard-constrained KF by 11.1% when the sensor uncertainty was 2.5%. The main advantage of the soft-constrained KF is that it is able to incorporate the uncertain constraints to improve the state estimate compared to the unconstrained KF without producing the

overconfident estimates that are the result of ignoring the constraint uncertainty.

Chapter 5 developed two types of methods for predicting the future state of a system given a schedule. The first, a Monte Carlo (MC) approach, used multiple samples from the parameter probability distributions to build up the distribution of the future state. The second approach was an analytical method that used several approximations to propagate the entire probability distributions without having to sample the distributions. As part of this, new Gaussian approximations to the inverse Gaussian distribution and generalised rectified Gaussian distribution were introduced. In multi-replenishment agent scenarios it is necessary to consider the order in which the replenishment agents arrive at their tasks. Methods for calculating the probability of an order of arrival, and then subsequently adjusting the arrival times for a specific order of arrival, were introduced. An alternative framework that was designed to be used in conjunction with an optimisation method that avoids multiple orders of arrival was also developed. In several single-replenishment agent and multi-replenishment agent scenarios, the analytical method was shown to produce comparable results to the MC approach while generally computing significantly faster. When comparing schedules, it was up to 99.8% accurate in comparison to the MC approach. In multi-replenishment agent scenarios with many replenishment agents and high uncertainty, consideration of the multiple orders of arrival resulted in a significant increase in the computation time of the analytical method, rendering it infeasible for practical scenarios.

Finally, **Chapter 6** introduced heuristic, meta-heuristic, and branch and bound optimisation methods that were combined with the estimation and prediction methods from Chapter 4 and Chapter 5. The heuristic and meta-heuristic methods were adapted from the existing literature on collection and replenishment scenarios primarily as benchmarks for the branch and bound methods. All of these methods were used within a Model Predictive Control (MPC)-like framework where a new allocation or schedule was calculated after each task was performed. The branch and bound methods were designed to exploit the MPC-like framework by focusing optimisation efforts on earlier tasks within the schedule so that good schedules could be

found quickly. A branch and bound method which avoids multiple orders of arrival of the replenishment agents was also developed for use with the alternative prediction framework in multi-replenishment agent scenarios. Of the methods tested, the branch and bound approaches using the analytical prediction framework from Chapter 5 consistently produced the best results. The heuristics produced myopic decisions, the undirected optimisation of the meta-heuristic was unable to sufficiently explore the search space, and the branch and bound approaches ignoring uncertainty were unable to differentiate between low- and high-risk schedules. In multi-replenishment agent scenarios, the branch and bound algorithm that avoids multiple orders of arrival was shown to have the best performance of the branch and bound approaches in terms of minimising both the downtime of the user agents and the computation time of the algorithm.

7.2 Contributions

The major contribution of this thesis was a framework for scheduling under uncertainty in SCAR scenarios, known as belief space scheduling. More specifically, the thesis contributions were:

- A mathematical formulation of the SCAR scenario, including a description of the optimisation objectives. The SCAR scenario relaxed many of the assumptions of existing scenarios in the literature by using an infinite time-horizon and incorporating uncertainty into the formulation.
- A method of approximating the conditional probability distribution of a Gaussian distributed variable that is conditioned on one or more Gaussian distributed variables as a Gaussian distribution. This was applied to the constrained KF problem using the truncation method where it was shown to outperform existing unconstrained and hard-constrained KF methods. Unlike existing methods for soft constraints which use numerical integration, this approach presented an analytical solution to the integration. The constrained KF was used in the

belief space scheduling framework to estimate the current state of the system in the presence of uncertain discrete sensors.

- A continuous-time, analytical framework for predicting the future state of a system given a schedule and uncertainty model. Where existing methods are computationally slow or unprincipled in their treatment of uncertainty, the proposed framework was computationally efficient and calculates a risk-weighted cost. When benchmarked against a MC approach, it was shown to produce similar discrimination performance while computing orders of magnitude faster. This framework enabled uncertainty to be accurately and efficiently incorporated into the optimisation methods.

- Several branch and bound optimisation methods for use within a MPC-like framework for single- and multi-replenishment agent SCAR scenarios. These were compared with existing heuristic and meta-heuristic methods where the benefit of the directed optimisation of the branch and bound methods in comparison to the myopic decisions of the heuristics and the undirected optimisation of the meta-heuristic was clear. The branch and bound approaches that used the analytical prediction framework to incorporate uncertainty into the optimisation also outperformed the branch and bound algorithms that ignored uncertainty.

- A Gaussian approximation for inverse Gaussian distributed variables.

- A Gaussian approximation of the generalised rectified Gaussian distribution.

7.3 Future Work

The work presented in this thesis opens a variety of future research avenues:

- The soft-constrained KF developed in this thesis is specifically for constraints and state estimates that are described by Gaussian distributions, and existing solutions for constrained KFs using other probability distributions have typically used numerical integration. Depending on the distribution types, it may be possible to derive analytical expressions for use in a soft-constrained KF. This would enable sensors which have non-Gaussian uncertainty to be incorporated efficiently.

- As shown in the results of Chapter 5, a source of error in the prediction framework results from the approximation of inverse Gaussian distributed variables degrading when the standard deviation of the original variable is large. Developing an improved approximation would lead to reduced error in comparison to the MC method.

- One of the current limitations of the multi-replenishment agent prediction framework is that the calculation of the probability of an order of arrival using existing CDF approximation methods is extremely slow. It has already been shown that, for some scenarios, it is viable to structure the optimisation in such a way that only one arrival order is highly probable, negating the need for this calculation. However, for scenarios where the ratio of replenishment agents to user agents is very high and there is high uncertainty, this may not be possible. Such scenarios will require faster methods of approximating the CDF, or algorithmic improvements that can minimise the need for the CDF calculation, to achieve real-time operation.

- An interesting avenue of future work on the optimisation side is to use a chance-constrained framework, where a schedule must satisfy a probability that the user agents will not exhaust their supply of the resource. This then enables the scheduler to optimise other aspects of the problem such as the distance travelled

or the operating cost of the replenishment agents, or use a hierarchical objective function which combines multiple objectives. The analytical prediction method presented in this thesis can easily be used in a chance-constrained framework—the probability of a user agent incurring downtime can be calculated from the distributions for the downtime of each user agent that are already generated by the framework.

- The current optimisation methods all require the Apparent Tardiness Cost (ATC) heuristic scaling parameter as an input. It may be possible to select an appropriate scaling parameter each time the optimisation algorithm is run by performing an optimisation using schedules purely generated by the ATC heuristic and varying the scaling parameter. This would go part way towards a system that does not require calibration by operators when the scenario changes.

- The method used to calculate the lower bound on the cost of a schedule in this thesis is very conservative due to the difficulty of estimating the cost to complete a partial schedule. Methods which produce a less conservative lower bound will enable the branch and bound algorithm to prune more branches, improving the computational performance.

- Some interesting algorithmic areas of future work include partitioning multi-replenishment agent problems into multiple single-replenishment agent problems, decentralised methods, and utilising parallel processing methods. These methods may reduce the computational requirements of the solution. Parallel versions of branch and bound already exist and offer the opportunity for significant computational speed gains.

List of References

[1] IBM CPLEX Optimizer. URL `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`.

[2] Gurobi Optimization. URL `http://www.gurobi.com/`.

[3] Ali-akbar Agha-mohammadi, Suman Chakravorty, and Nancy M Amato. FIRM: Sampling-based feedback motion planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, 33:268–304, 2013.

[4] Stephane Alarie and Michel Gamache. Overview of solution strategies used in truck dispatching systems for open pit mines. *International Journal of Surface Mining, Reclamation and Environment*, 16(1):59–76, 2002.

[5] Ali Allahverdi, C.T. Ng, T.C.E. Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.

[6] Brian D.O. Anderson and John B. Moore. *Optimal Filtering*. Courier Corporation, New York, 2005.

[7] Ariel Arelovich, Favio Masson, Osvaldo Agamennoni, Stewart Worrall, and Eduardo Nebot. Heuristic rule for truck dispatching in open-pit mines with local information-based decisions. In *Proceedings of the 2010 13th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1408–1414, 2010.

[8] Jean-Paul Arnaout, Ghaith Rabadi, and Ji Hyon Mun. A dynamic heuristic for the stochastic unrelated parallel machine scheduling problem. *International Journal of Operations Research*, 3(2):136–143, 2006.

[9] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[10] Obrad Babic. Optimization of Refuelling Truck Fleets at an Airport. *Transportation Research Part B: Methodological*, 218(6):479–481, 1987.

[11] Haoyu Bai, Hsu David, and Wee Sun Lee. Integrated Perception and Planning in the Continuous Space: A POMDP Approach. *The International Journal of Robotics Research*, 33(9):1288–1302, 2014.

[12] J. Wesley Barnes, Victor Duane Wiley, James T. Moore, and David M. Ryer. Solving the aerial fleet refueling problem using group theoretic tabu search. *Mathematical and Computer Modelling*, 39(6):617–640, 2004.

[13] Matthew H. Bassett, Joseph F. Pekny, and Gintaras V. Reklaitis. Using detailed scheduling to obtain realistic operating policies for a batch processing facility. *Industrial Engineering and Chemical Research*, 36(5):1717–1726, 1997.

[14] Guilherme S. Bastos, Luiz E. Souza, Fabio T. Ramos, and Carlos H. C. Ribeiro. A single-dependent agent approach for stochastic time-dependent truck dispatching in open-pit mining. In *Proceedings of the 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1057–1062, 2011.

[15] Guilherme Sousa Bastos. *Methods for Truck Dispatching in Open-Pit Mining*. PhD thesis, Aeronautics Institute of Technology, São José dos Campos, 2010.

[16] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython : The Best of Both Worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[17] Drew Bellamy and Luka Pravica. Assessing the impact of driverless haul trucks in Australian surface mining. *Resources Policy*, 36(2):149–158, 2011.

[18] Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1-3):49–71, September 2003.

[19] Deepak Bhadauria, Onur Tekdas, and Volkan Isler. Robotic data mules for collecting data over sparse sensor fields. *Journal of Field Robotics*, 28(3): 388–404, 2011.

[20] Mauro Boccadoro, Francesco Martinelli, and Stefano Pagnottelli. Constrained and quantized Kalman filtering for an RFID robot localization problem. *Autonomous Robots*, 29(3-4):235–251, 2010.

[21] Blai Bonet and Héctor Geffner. Planning with Incomplete Information as Heuristic Search in Belief Space. *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 52–61, 2000.

[22] Adam Bry and Nicholas Roy. Rapidly-exploring Random Belief Trees for motion planning under uncertainty. In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 723–730, 2011.

[23] Michael R. Bussieck and Stefan Vigerske. MINLP Solver Software. 2012.

[24] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. *AAAI*, 94, 1994.

[25] Howard Cheng. Problem H: Harvesting Crops. 2013. URL `http://psc.cpsc.ucalgary.ca/acpc/2013/contest/problems/H.html`.

[26] T.C.E Cheng, Q Ding, and B.M.T Lin. A concise survey of scheduling with time-dependent processing times. *European Journal of Operational Research*, 152(1):1–13, 2004.

[27] Patrick Dallaire, Camille Besse, Stephane Ross, and Brahim Chaib-draa. Bayessian Reinforcement Learning in Continuous POMDPs with Gaussian Processes. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2604–2609, 2009.

[28] Fred Daum and Jim Huang. Curse of Dimensionality and Particle Filters. In *Proceedings of the 2003 IEEE Aerospace Conference*, volume 4, pages 1979–1993, 2003.

[29] Marc Peter Deisenroth and Jan Peters. Solving Nonlinear Continuous State-Action-Observation POMDPs for Mechanical Systems with Gaussian Noise. In *Proceedings of the 2012 European Workshop on Reinforcement Learning (EWRL 2012)*, pages 1–14, 2012.

[30] Emidio DiGiampaolo and Francesco Martinelli. A passive UHF-RFID system for the localization of an indoor autonomous vehicle. *IEEE Transactions on Industrial Electronics*, 59(10):3961–3970, 2012.

[31] Yvan Dumas, Jacques Desrosiers, Eric Gelinas, and Marius M. Solomon. An Optimal Algorithm for the Traveling Salesman Problem with Time Windows. *Operations Research*, 43(2):367–371, 1995.

[32] Matthew Dunbabin, Peter Corke, Iuliu Vasilescu, and Daniela Rus. Data muling over underwater wireless sensor networks using an autonomous underwater vehicle. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2091–2098, 2006.

[33] Tom Erez and William D. Smart. A scalable method for solving high-dimensional continuous POMDPs using local approximation. In *Proceedings of the 2010 International Conference on Uncertainty in Artificial Intelligence*, 2010.

[34] Cheng Fang, Peng Yu, and Brian C. Williams. Chance-Constrained Probabilistic Simple Temporal Problems. *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2264–2270, 2014.

[35] Hatem A. Fayed and Amir F. Atiya. An evaluation of the integral of the product of the error function and the normal probability density with application to the bivariate normal integral. *Mathematics of Computation*, 83 (285):235–250, 2014.

[36] Dave Ferguson, Maxim Likhachev, and Anthony Stentz. A Guide to Heuristic-based Path Planning. In *Proceedings of the International Workshop on Planning Under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS)*, pages 9–18, 2005.

[37] Alan Genz and Frank Bretz. *Computation of multivariate normal and t probabilities*. Springer Science & Business Media, 2009.

[38] Geo Drill Global Services. Mining. 2013. URL `http://www.geodrillglobal.com/index_files/mining-drill-rig.jpg`.

[39] Nachi Gupta and Raphael Hauser. Kalman Filtering with Equality and Inequality State Constraints. Technical Report 07, Oxford University Computing Laboratory, 2007.

[40] Kris Hauser. *Randomized belief-space replanning in partially-observable continuous spaces*, volume 68. Springer Berlin Heidelberg, 2011.

[41] Yaacov Hel-Or, Ari Rappoport, and Michael Werman. Relaxed parametric design with probabilistic constraints. In *Proceedings on the second ACM symposium on Solid modeling and applications*, pages 261–270, 1993.

[42] Laura M. Hiatt, Terry L. Zimmerman, Stephen F. Smith, and Reid Simmons. Strengthening schedules through uncertainty analysis. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 175–180, 2009.

[43] Y.S. Hsu and B.M.T. Lin. Minimization of maximum lateness under linear deterioration. *Omega*, 31(6):459–469, December 2003.

[44] Vadim Indelman, Luca Carlone, and Frank Dellaert. Planning Under Uncertainty in the Continuous Domain : a Generalized Belief Space Approach. In *Proceedings of the 2014 IEEE International Conference on Robotics & Automation (ICRA)*, pages 6763–6770, 2014.

[45] Vadim Indelman, Luca Carlone, and Frank Dellaert. Planning in the continuous domain: A generalized belief space approach for autonomous navigation in unknown environments. *The International Journal of Robotics Research*, 34(7):849–882, 2015.

[46] Zhipu Jin, Tal Shima, and Corey J. Schumacher. Optimal scheduling for refueling multiple autonomous aerial vehicles. *IEEE Transactions on Robotics*, 22(4):682–693, 2006.

[47] Zhipu Jin, Tal Shima, and Corey J. Schumacher. Scheduling and sequence reshuffle for autonomous aerial refueling of multiple UAVs. In *Proceedings of the 2006 American Control Conference*, pages 2177–2182, 2006.

[48] David A. Johnson, David J. Naffin, Jeffrey S. Puhalla, Julian Sanchez, and Carl K. Wellington. Development and implementation of a team of robotic tractors for autonomous peat moss harvesting. *Journal of Field Robotics*, 26 (6-7):549–571, 2009.

[49] Simon J. Julier and Jeffrey K. Uhlmann. A New Extension of the Kalman Filter to Nonlinear Systems. In *Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, pages 182–193, 1997.

[50] Simon J. Julier and Jeffrey K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[51] Leslie Pack Kaelbling and Tomas Lozano-Perez. Integrated task and motion planning in belief space. *The International Journal of Robotics Research*, 32 (9-10):1194–1227, 2013.

[52] Rudolf E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[53] Balajee Kannan, Victor Marmol, Jaime Bourne, and M Bernardine Dias. The Autonomous Recharging Problem: Formulation and a Market-based Solution. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3488–3495, 2013.

[54] Sezgin Kaplan and Ghaith Rabadi. Exact and heuristic algorithms for the aerial refueling parallel machine scheduling problem with due date-to-deadline window and ready times. *Computers & Industrial Engineering*, 62(1):276–285, 2012.

[55] Sezgin Kaplan and Ghaith Rabadi. Simulated annealing and metaheuristic for randomized priority search algorithms for the aerial refuelling parallel machine scheduling problem with due date-to-deadline windows and release times. *Engineering Optimization*, 45(1):67–87, 2013.

[56] Dimitri Klimenko, Joshua Song, and Hanna Kurniawati. TAPIR: A Software Toolkit for Approximating and Adapting POMDP Solutions Online. In *Proceedings of the 2014 Australasian Conference on Robotics and Automation*, 2014.

[57] Mikhail A. Kubzin and Vitaly A. Strusevich. Planning machine maintenance in two-machine shop scheduling. *Operations Research*, 54(4):789–800, 2006.

[58] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, pages 79–86, 1951.

[59] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Proceedings of Robotics: Science and Systems*, 2008.

[60] Hanna Kurniawati, Tirthankar Bandyopadhyay, and Nicholas M. Patrikalakis. Global motion planning under uncertain motion, sensing, and environment map. *Autonomous Robots*, 33(3):255–272, 2012.

[61] Ten-Hwang Lai and Sartaj Sahni. Anomalies in Parallel Branch-and-Bound Algorithms. *Communications of the ACM*, 27(6):594–602, 1984.

[62] AH Land and AG Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, 28(3):497–520, 1960.

[63] Xiaoping Li and Yi Zhang. Adaptive Hybrid Algorithms for the Sequence-Dependent Setup Time Permutation. *IEEE Transactions on Automation Science and Engineering*, 9(3):578–595, 2012.

[64] Yaroslav Litus, Richard T. Vaughan, and Pawel Zebrowski. The Frugal Feeding Problem: Energy-efficient, multi-robot, multi-place rendezvous. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA)*, pages 27–32, 2007.

[65] Yaroslav Litus, Pawel Zebrowski, and Richard T. Vaughan. A distributed heuristic for energy-efficient multirobot multiplace rendezvous. *IEEE Transactions on Robotics*, 25(1):130–135, 2009.

[66] Ming Ma, Yuanyuan Yang, and Miao Zhao. Tour planning for mobile data-gathering mechanisms in wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 62(4):1472–1483, 2013.

[67] Costanzo Manes and Francesco Martinelli. State estimation under quantized measurements: A Sigma-Point Bayesian approach. In *Proceedings of the 52nd IEEE Conference on Decision and Control*, pages 5024–5029, 2013.

[68] George Marsaglia. Ratios of normal variables. *Journal of Statistical Software*, 16(4):1–10, 2006.

[69] Neil Mathew, Stephen L. Smith, and Steven L. Waslander. A graph-based approach to multi-robot rendezvous for recharging in persistent tasks. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3482–3487, 2013.

[70] Neil Mathew, Stephen L. Smith, and Steven L. Waslander. Multirobot Rendezvous Planning for Recharging in Persistent Tasks. *IEEE Transactions on Robotics*, 31(1):128–142, 2015.

[71] Jia Meng, Jianqiu (Michelle) Zhang, Yidong Chen, and Yufei Huang. Bayesian non-negative factor analysis for reconstructing transcription factor mediated regulatory networks. *Proteome Science*, 9(Suppl 1):S9, 2011.

[72] Todd K. Moon and Wynn C. Stirling. *Mathematical Methods and Algorithms for Signal Processing*. New Jersey: Prentice Hall, Upper Saddle River, 2000.

[73] Mohan Munirathinam and Jon C. Yingling. A review of computer-based truck dispatching strategies for surface mining operations. *International Journal of Surface Mining, Reclamation and Environment*, 8(1):1–15, 1994.

[74] Liz Murphy and Paul Newman. Risky Planning on Probabilistic Costmaps for Path Planning in Outdoor Environments. *IEEE Transactions on Robotics*, 29 (2):445–457, 2013.

[75] Seza Orçun, I. Kuban Altinel, and Öner Hortaçsu. Scheduling of batch processes with operational uncertainties. *Computers & chemical engineering*, 20(96):2–7, 1996.

[76] Christos H. Papadimitriou and John N. Tsitsiklis. The Complexity of Markov Decision Processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[77] Sachin Patil, Jur Van Den Berg, and Ron Alterovitz. Estimating probability of collision for safe motion planning under Gaussian motion and sensing uncertainty. In *Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, pages 3238–3244, 2012.

[78] Sachin Patil, Yan Duan, John Schulman, Ken Goldberg, and Pieter Abbeel. Gaussian Belief Space Planning with Discontinuities in Sensing Domains. In *Proceedings of the 2014 IEEE International Conference on Robotics & Automation (ICRA)*, pages 6483–6490, 2014.

[79] Sachin Patil, Gregory Kahn, Michael Laskey, and John Schulman. *Scaling up Gaussian Belief Space Planning through Covariance-Free Trajectory Optimization and Automatic Differentiation*. Springer International Publishing, 2015.

[80] Michael Pinedo. *Scheduling: theory, algorithms, and systems*. Prentice Hall, Upper Saddle, N.J., 2nd edition, 2002.

[81] Robert Platt Jr, Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Proceedings of the 6th Robotics: Science and Systems Conference*, 2010.

[82] Samuel Prentice and Nicholas Roy. The Belief Roadmap: Efficient Planning in Belief Space by Factoring the Covariance. *The International Journal of Robotics Research*, 28(11-12):1448–1465, 2009.

[83] Eric Raboin, Petr Švec, Dana S. Nau, and Satyandra K. Gupta. Model-predictive asset guarding by team of autonomous surface vehicles in environment with civilian boats. *Autonomous Robots*, 38:261–282, 2014.

[84] Vijayender Reddy and T. T. Narendran. Heuristics and sequence-dependent set-up jobs in flow line cells. *International Journal of Production Research*, 41 (1):193–206, 2003.

[85] Victor Romero-Cano, Gabriel Agamennoni, and Juan Nieto. A variational approach to simultaneous multi-object tracking and classification. *The International Journal of Robotics Research*, pages 1–18, 2015.

[86] Oren Rozen. Boeing 707 refueling F-15s, June 28, 2011, Israeli Air Force Flight Academy ranks ceremony. 2011. URL `https://en.wikipedia.org/wiki/File:Dargot_28062011_707_Refueling.jpg`.

[87] Xinguang Shao, Biao Huang, and Jong Min Lee. Constrained Bayesian state estimation - A comparative study and a new particle filter based approach. *Journal of Process Control*, 20(2):143–157, 2010.

[88] Haijun Shen. *Optimal scheduling for satellite refuelling in circular orbits*. PhD thesis, Georgia Institute of Technology, 2003.

[89] Haijun Shen and Panagiotis Tsiotras. Optimal Scheduling for Servicing Multiple Satellites in a Circular Constellation. In *Proceedings of the 2002 AIAA/AAS Astrodynamics Specialists Conference*, pages 1–10, 2002.

[90] Nobutaka Shimada and Yoshiaki Shirai. Hand gesture estimation and model refinement using monocular camera-ambiguity limitation by inequality constraints. In *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition*, 1998.

[91] Dan Simon. *Optimal State Estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, Hoboken, New Jersey, 2006.

[92] Dan Simon. Kalman filtering with state constraints: a survey of linear and nonlinear algorithms. *IET Control Theory & Applications*, 4(8):1303–1318, 2010.

[93] Dan Simon and Donald L. Simon. Kalman filtering with inequality constraints for turbofan engine health estimation. *IEE Proceedings - Control Theory and Applications*, 153(3):371–378, 2006.

[94] Dan Simon and Donald L. Simon. Constrained Kalman filtering via density function truncation for turbofan engine health estimation. *International Journal of Systems Science*, 41(2):159–171, 2010.

[95] Nicholas D. Socci, Daniel D. Lee, and H. Sebastian Seung. The Rectified Gaussian Distribution. In *Advances in Neural Information Processing Systems 10*, pages 350–356. 1998.

[96] Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. Information Gain-based Exploration Using Rao-Blackwellized Particle Filters. In *Proceedings of the 2005 Robotics: Science and Systems Conference*, pages 65–72, 2005.

[97] Ondřej Straka, Jindřich Duník, and Miroslav Šimandl. Truncation nonlinear filters for state estimation with nonlinear inequality constraints. *Automatica*, 48(2):273–286, 2012.

[98] Robert F. Subtil, Diego M. Silva, and Julio C. Alves. A Practical Approach to Truck Dispatch for Open Pit Mines. In *Proceedings of the 35th APCOM Symposium*, pages 765–777, Wollongong, NSW, 2011.

[99] C. H. Ta, J. V. Kresta, J. F. Forbes, and H. J. Marquez. A stochastic optimization approach to mine truck allocation. *International Journal of Surface Mining, Reclamation and Environment*, 19(3):162–175, 2005.

[100] Bruno O. S. Teixeira, Leonardo A. B. Tôrres, Luis A. Aguirre, and Dennis S. Bernstein. Unscented filtering for interval-constrained nonlinear systems. In *Proceedings of the IEEE Conference on Decision and Control*, pages 5116–5121, 2008.

[101] Onur Tekdas and Volkan Isler. Using mobile robots to harvest data from sensor fields. *IEEE Wireless Communications*, 16(1):22–28, 2009.

[102] Onur Tekdas, Deepak Bhadauria, and Volkan Isler. Efficient data collection from wireless nodes under the two-ring communication model. *The International Journal of Robotics Research*, 31(6):774–784, 2012.

[103] Sebastian Thrun. Particle Filters in Robotics. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 511–518, 2002.

[104] Yuldi Tirta, Zhiyuan Li, Yung-Hsiang Lu, and Saurabh Bagchi. Efficient collection of sensor data in remote fields using mobile collectors. In *Proceedings on the 13th IEEE International Conference on Computer Communications and Networks (ICCCN)*, pages 515–519, 2004.

[105] M. Duran Toksarı and Ertan Güner. Parallel machine scheduling problem to minimize the earliness/tardiness costs with learning effect and deteriorating jobs. *Journal of Intelligent Manufacturing*, 21(6):843–851, 2009.

[106] Panagiotis Tsiotras and Arnaud De Nailly. A Comparison Between Peer-to-Peer and Single-Spacecraft Refueling Strategies. In *Proceedings of Infotech at Aerospace, AIAA*, pages 26–29, 2005.

[107] University of Florida. International Symposium on Mechanical Harvesting & Handling Systems of Fruits and Nuts. 2012. URL `http://conference.ifas.ufl.edu/harvest/`.

[108] Rafael Valencia, Marti Morta, Juan Andrade-Cetto, and Josep M. Porta. Planning reliable paths with Pose SLAM. *IEEE Transactions on Robotics*, 29 (4):1050–1059, 2013.

[109] Jur van den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012.

[110] E. J. Van Henten, J. Hemming, B. a J Van Tuijl, J. G. Kornet, J. Meuleman, J. Bontsema, and E. a. Van Os. An autonomous robot for harvesting cucumbers in greenhouses. *Autonomous Robots*, 13:241–258, 2002.

[111] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke. Data collection, storage, and retrieval with an underwater sensor network. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems - SenSys '05*, pages 154–165, 2005.

[112] Robert Ware and Frank Lad. Approximating the distribution for sums of products of normal variables. Technical Report 1978, The University of Queensland, 2003.

[113] J. W. White and J. P. Olson. Computer-based dispatching in mines with concurrent operating objectives. *Mining Engineering*, 38(11):1045–1054, 1986.

[114] J. W. White, J. P. Olson, and S. I. Vohnout. On improving truck shovel productivity in open pit mines. *CIM Bulletin*, 86(973):43–49, 1993.

[115] Victor Duane Wiley. *The Aerial Fleet Refueling Problem.* PhD thesis, University of Texas at Austin, 2001.

[116] Yuan Yan and Yasamin Mostofi. An Efficient Clustering and Path Planning Strategy for Data Collection in Sensor Networks Based on Space-Filling Curves. In *Proceedings of the 2014 IEEE Conference on Decision and Control*, pages 6895–6901, 2014.

[117] Sungwook Yoon, Alan Fern, and Robert Givan. FF-Replan: A baseline for probabilistic planning. *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 352–359, 2007.

[118] Bo Yuan, Maria Orlowska, and Shazia Sadiq. On the Optimal Robot Routing Problem in Wireless Sensor Networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(9):1252–1261, 2007.

[119] Pawel Zebrowski and Richard T. Vaughan. Recharging robot teams: A tanker approach. In *Proceedings of the 12th International Conference on Advanced Robotics, ICAR '05*, pages 803–810, 2005.

Appendix A

Integrals

A.1 Calculation of the Area

The area of the distribution given by Equation 4.38 is calculated as follows:

$$\begin{aligned}
&\int_{-\infty}^{\infty} \frac{1}{2\sqrt{2\pi}} \exp\left(-\zeta^2/2\right) \left[1 + \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right] \mathrm{d}\zeta \\
&= \frac{1}{2}\left(\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta + \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\zeta^2/2\right) \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) \mathrm{d}\zeta\right)
\end{aligned}$$

The first integral results in the CDF of the standard normal distribution and is equal to 1:

$$\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta = 1 \tag{A.1}$$

The second integral does not have an indefinite integral, but a definite integral is provided in [35]:

$$\int_{-\infty}^{\infty} \exp\left(-(\alpha t-\beta)^2\right) \operatorname{erf}(at+b)\,\mathrm{d}t = \frac{\sqrt{\pi}}{\alpha}\operatorname{erf}\left[\frac{\alpha b-\beta a}{\sqrt{\alpha^2+a^2}}\right] \tag{A.2}$$

Therefore:

$$\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\zeta^2/2\right) \operatorname{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) \mathrm{d}\zeta = \operatorname{erf}\left(\frac{-\mu_{c,i}}{\sqrt{2\left(\sigma_{c,i}^2+1\right)}}\right) \tag{A.3}$$

To summarise:

$$\boxed{\begin{aligned}\int_{-\infty}^{\infty} \frac{1}{2\sqrt{2\pi}} \exp\left(-\zeta^2/2\right) \left[1+\operatorname{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right] \mathrm{d}\zeta \\ = \frac{1}{2}\left[1-\operatorname{erf}\left(\frac{\mu_{c,i}}{\sqrt{2\left(\sigma_{c,i}^2+1\right)}}\right)\right]\end{aligned}} \tag{A.4}$$

A.2 Calculation of the Mean

The integral in Equation 4.41 is calculated as follows:

$$\begin{aligned}&\int_{-\infty}^{\infty} \zeta \exp\left(-\zeta^2/2\right) \left[1+\operatorname{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right] \mathrm{d}\zeta \\ &\quad = \int_{-\infty}^{\infty} \zeta \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta \qquad \text{(A.5)} \\ &\quad\quad + \int_{-\infty}^{\infty} \zeta \exp\left(-\zeta^2/2\right) \operatorname{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) \mathrm{d}\zeta \qquad \text{(A.6)}\end{aligned}$$

The first integral, Equation A.5, is evaluated as:

$$\int_{-\infty}^{\infty} \zeta \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta = \left[-\exp\left(-\zeta^2/2\right)\right]_{\infty}^{-\infty} = 0 \tag{A.7}$$

The second integral, Equation A.6, requires integration by parts. Let

$$u = \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) \qquad \mathrm{d}v = \zeta \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta \tag{A.8}$$

then

$$\mathrm{d}u = \frac{1}{\sigma_{c,i}}\sqrt{\frac{2}{\pi}} \exp\left(-\frac{(\zeta - \mu_{c,i})^2}{2\sigma_{c,i}^2}\right) \mathrm{d}\zeta \qquad v = -\exp\left(-\zeta^2/2\right) \tag{A.9}$$

This gives:

$$\begin{aligned}\int_{-\infty}^{\infty} \zeta \exp\left(-\zeta^2/2\right) \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) \mathrm{d}\zeta = &\left[-\exp\left(-\zeta^2/2\right) \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right]_{-\infty}^{\infty} \\ &+ \int_{-\infty}^{\infty} \frac{\exp\left(-\zeta^2/2\right)}{\sigma_{c,i}}\sqrt{\frac{2}{\pi}} \exp\left(-\frac{(\zeta - \mu_{c,i})^2}{2\sigma_{c,i}^2}\right) \mathrm{d}\zeta\end{aligned} \tag{A.10}$$

where

$$\left[-\exp\left(-\zeta^2/2\right) \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right]_{-\infty}^{\infty} = 0 \tag{A.11}$$

For the remaining integral, completing the square in the exponent gives:

$$\begin{aligned}&\int_{-\infty}^{\infty} \frac{\exp\left(-\zeta^2/2\right)}{\sigma_{c,i}}\sqrt{\frac{2}{\pi}} \exp\left(-\frac{(\zeta - \mu_{c,i})^2}{2\sigma_{c,i}^2}\right) \mathrm{d}\zeta \\ &\qquad = \sqrt{\frac{2}{\pi}} \frac{\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sigma_{c,i}} \int_{-\infty}^{\infty} \exp\left(-\frac{\left(\frac{\zeta(\sigma_{c,i}^2+1)-\mu_{c,i}}{\sigma_{c,i}}\right)^2}{2(\sigma_{c,i}^2+1)}\right) \mathrm{d}\zeta\end{aligned} \tag{A.12}$$

Let

$$u = \frac{\zeta(\sigma_{c,i}^2 + 1) - \mu_{c,i}}{\sigma_{c,i}}$$

then

$$\sqrt{\frac{2}{\pi}}\frac{\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sigma_{c,i}}\int_{-\infty}^{\infty}\exp\left(-\frac{\left(\frac{\zeta(\sigma_{c,i}^2+1)-\mu_{c,i}}{\sigma_{c,i}}\right)^2}{2(\sigma_{c,i}^2+1)}\right)\mathrm{d}\zeta$$
$$=\sqrt{\frac{2}{\pi}}\frac{\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sigma_{c,i}^2+1}\int_{-\infty}^{\infty}\exp\left(-\frac{u^2}{2(\sigma_{c,i}^2+1)}\right)\mathrm{d}u \quad \text{(A.13)}$$

Let

$$v=\frac{u}{\sqrt{2\left(\sigma_{c,i}^2+1\right)}}$$

then

$$\sqrt{\frac{2}{\pi}}\frac{\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sigma_{c,i}^2+1}\int_{-\infty}^{\infty}\exp\left(-v^2\right)\mathrm{d}u$$
$$=\frac{\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sqrt{\sigma_{c,i}^2+1}}\left[\mathrm{erf}(v)\right]_{-\infty}^{\infty}$$
$$=\frac{2\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sqrt{\sigma_{c,i}^2+1}} \quad \text{(A.14)}$$

To summarise:

$$\boxed{\int_{-\infty}^{\infty}\zeta\exp\left(-\zeta^2/2\right)\left[1+\mathrm{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right]\mathrm{d}\zeta=\frac{2}{\sqrt{\sigma_{c,i}^2+1}}\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)} \quad \text{(A.15)}$$

A.3 Calculation of the Variance

The integral in Equation 4.42 is calculated as follows:

$$\begin{aligned}
&\int_{-\infty}^{\infty} (\zeta - \mu_i)^2 \exp\left(-\zeta^2/2\right) \left[1 + \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right] \mathrm{d}\zeta \\
&\qquad = \int_{-\infty}^{\infty} \zeta^2 \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta && \text{(A.16)} \\
&\qquad + \int_{-\infty}^{\infty} \zeta^2 \exp\left(-\zeta^2/2\right) \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) \mathrm{d}\zeta && \text{(A.17)} \\
&\qquad - \int_{-\infty}^{\infty} 2\mu_i \zeta \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta && \text{(A.18)} \\
&\qquad - \int_{-\infty}^{\infty} 2\mu_i \zeta \exp\left(-\zeta^2/2\right) \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) \mathrm{d}\zeta && \text{(A.19)} \\
&\qquad + \int_{-\infty}^{\infty} \mu_i^2 \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta && \text{(A.20)} \\
&\qquad + \int_{-\infty}^{\infty} \mu_i^2 \exp\left(-\zeta^2/2\right) \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) \mathrm{d}\zeta && \text{(A.21)}
\end{aligned}$$

Now consider each of these integrals separately. The integral in Equation A.16 is given by:

$$\begin{aligned}
\int_{-\infty}^{\infty} \zeta^2 \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta &= \left[\sqrt{\frac{\pi}{2}} \operatorname{erf}\left(\frac{\zeta}{\sqrt{2}} - \zeta \exp\left(-\zeta^2/2\right)\right)\right]_{-\infty}^{\infty} \\
&= \sqrt{2\pi} && \text{(A.22)}
\end{aligned}$$

For the integral in Equation A.17, let

$$u = \operatorname{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) \qquad \mathrm{d}v = \zeta^2 \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta \qquad \text{(A.23)}$$

then

$$\mathrm{d}u = \frac{1}{\sigma_{c,i}}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{(\zeta-\mu_{c,i})^2}{2\sigma_{c,i}^2}\right)\mathrm{d}\zeta \qquad v = \sqrt{\frac{\pi}{2}}\mathrm{erf}\left(\frac{\zeta}{\sqrt{2}}\right) - \zeta\exp\left(-\zeta^2/2\right) \tag{A.24}$$

$$\begin{aligned}
&\int_{-\infty}^{\infty} \zeta^2\exp\left(-\zeta^2/2\right)\mathrm{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\mathrm{d}\zeta \\
&= \left[\mathrm{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\left(\sqrt{\frac{\pi}{2}}\mathrm{erf}\left(\frac{\zeta}{\sqrt{2}}\right) - \zeta\exp\left(-\zeta^2/2\right)\right)\right]_{-\infty}^{\infty} && \text{(A.25)} \\
&\quad - \int_{-\infty}^{\infty} \frac{1}{\sigma_{c,i}}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{(\zeta-\mu_{c,i})^2}{2\sigma_{c,i}^2}\right)\left(\sqrt{\frac{\pi}{2}}\mathrm{erf}\left(\frac{\zeta}{\sqrt{2}}\right) - \zeta\exp\left(-\zeta^2/2\right)\right)\mathrm{d}\zeta && \text{(A.26)}
\end{aligned}$$

Equation A.25 calculates to:

$$\left[\mathrm{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\left(\sqrt{\frac{\pi}{2}}\mathrm{erf}\left(\frac{\zeta}{\sqrt{2}}\right) - \zeta\exp\left(-\zeta^2/2\right)\right)\right]_{-\infty}^{\infty} = 0 \tag{A.27}$$

The integral in Equation A.26 can be split into the following integrals:

$$\begin{aligned}
&\int_{-\infty}^{\infty} \frac{1}{\sigma_{c,i}}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{(\zeta-\mu_{c,i})^2}{2\sigma_{c,i}^2}\right)\left(\sqrt{\frac{\pi}{2}}\mathrm{erf}\left(\frac{\zeta}{\sqrt{2}}\right) - \zeta\exp\left(-\zeta^2/2\right)\right)\mathrm{d}\zeta \\
&\qquad = \int_{-\infty}^{\infty} \frac{\exp\left(-\frac{(\zeta-\mu_{c,i})^2}{2\sigma_{c,i}^2}\right)}{\sigma_{c,i}}\mathrm{erf}\left(\frac{\zeta}{\sqrt{2}}\right)\mathrm{d}\zeta && \text{(A.28)} \\
&\qquad\quad - \int_{-\infty}^{\infty} \sqrt{\frac{2}{\pi}}\frac{\zeta}{\sigma_{c,i}}\exp\left(-\frac{(\zeta-\mu_{c,i})^2}{2\sigma_{c,i}^2}\right)\exp\left(-\zeta^2/2\right)\mathrm{d}\zeta && \text{(A.29)}
\end{aligned}$$

Using the formula provided in [35], the integral in Equation A.28 equates to:

$$\int_{-\infty}^{\infty} \frac{\exp\left(-\frac{(\zeta-\mu_{c,i})^2}{2\sigma_{c,i}^2}\right)}{\sigma_{c,i}} \mathrm{erf}\left(\frac{\zeta}{\sqrt{2}}\right) \mathrm{d}\zeta = \sqrt{2\pi}\mathrm{erf}\left(\frac{\mu_{c,i}}{\sqrt{2(\sigma_{c,i}^2+1)}}\right) \tag{A.30}$$

The integral in Equation A.29 has the same exponent as in Equation A.12. Let

$$u = \frac{\zeta(\sigma_{c,i}^2+1) - \mu_{c,i}}{\sigma_{c,i}}$$

then

$$\begin{aligned} &\int_{-\infty}^{\infty} \sqrt{\frac{2}{\pi}} \frac{\zeta}{\sigma_{c,i}} \exp\left(-\frac{(\zeta-\mu_{c,i})^2}{2\sigma_{c,i}^2}\right) \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta \\ &\qquad = \int_{-\infty}^{\infty} \sqrt{\frac{2}{\pi}} \frac{\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{(\sigma_{c,i}^2+1)^2} (\sigma_{c,i}u + \mu_{c,i}) \exp\left(-\frac{u^2}{2(\sigma_{c,i}^2+1)}\right) \mathrm{d}u \end{aligned} \tag{A.31}$$

Let

$$v = \frac{u}{\sqrt{2\left(\sigma_{c,i}^2+1\right)}}$$

then

$$\int_{-\infty}^{\infty} \sqrt{\frac{2}{\pi}} \frac{\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{(\sigma_{c,i}^2+1)^2} (\sigma_{c,i}u + \mu_{c,i}) \exp\left(-\frac{u^2}{2(\sigma_{c,i}^2+1)}\right) \mathrm{d}u$$

$$= \int_{-\infty}^{\infty} \sqrt{\frac{2}{\pi}} \frac{\sqrt{2\left(\sigma_{c,i}^2+1\right)} \exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{(\sigma_{c,i}^2+1)^2} (\sigma_{c,i}\sqrt{2\left(\sigma_{c,i}^2+1\right)}v + \mu_{c,i}) \exp\left(-v^2\right) \mathrm{d}v \tag{A.32}$$

$$= \int_{-\infty}^{\infty} \frac{2\sqrt{2}\sigma_{c,i} \exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sqrt{\pi}(\sigma_{c,i}^2+1)} v \exp\left(-v^2\right) \mathrm{d}v \tag{A.33}$$

$$+ \int_{-\infty}^{\infty} \frac{2\mu_{c,i} \exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sqrt{\pi}(\sigma_{c,i}^2+1)^{3/2}} \exp\left(-v^2\right) \mathrm{d}v \tag{A.34}$$

The integral in Equation A.33 equates to:

$$\int_{-\infty}^{\infty} \frac{2\sqrt{2}\sigma_{c,i} \exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sqrt{\pi}(\sigma_{c,i}^2+1)} v \exp\left(-v^2\right) \mathrm{d}v$$

$$= \frac{2\sqrt{2}\sigma_{c,i} \exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sqrt{\pi}(\sigma_{c,i}^2+1)} \left[-\frac{\exp\left(-v^2\right)}{2}\right]_{-\infty}^{\infty} = 0 \tag{A.35}$$

The integral in Equation A.34 equates to:

$$\begin{aligned}\int_{-\infty}^{\infty} \frac{2\mu_{c,i}\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sqrt{\pi}(\sigma_{c,i}^2+1)^{3/2}}\exp\left(-v^2\right)\mathrm{d}v \\ &= \frac{2\mu_{c,i}\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{\sqrt{\pi}(\sigma_{c,i}^2+1)^{3/2}}\left[\frac{1}{2}\sqrt{\pi}\mathrm{erf}(v)\right]_{-\infty}^{\infty} \\ &= \frac{2\mu_{c,i}\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{(\sigma_{c,i}^2+1)^{3/2}} \end{aligned} \tag{A.36}$$

To summarise, the integral in Equation A.17 simplifies to:

$$\begin{aligned}\int_{-\infty}^{\infty} \zeta^2\exp\left(-\zeta^2/2\right)\mathrm{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\mathrm{d}\zeta \\ &= \frac{2\mu_{c,i}\exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2+1)}\right)}{(\sigma_{c,i}^2+1)^{3/2}} - \sqrt{2\pi}\mathrm{erf}\left(\frac{\mu_{c,i}}{\sqrt{2(\sigma_{c,i}^2+1)}}\right) \end{aligned} \tag{A.37}$$

The integral in Equation A.18 equates to:

$$\int_{-\infty}^{\infty} 2\mu_i\zeta\exp\left(-\zeta^2/2\right)\mathrm{d}\zeta = \left[-\mu_i\exp\left(-\zeta^2/2\right)\right]_{-\infty}^{\infty} = 0 \tag{A.38}$$

The integral in Equation A.19 is similar to the integral evaluated in Section A.2 and equates to:

$$\int_{-\infty}^{\infty} 2\mu_i\zeta\exp\left(-\zeta^2/2\right)\mathrm{erf}\left(\frac{\zeta-\mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\mathrm{d}\zeta = \frac{4\mu_i}{\sqrt{\sigma_{c,i}^2+1}}\exp\left(-\frac{\mu_{c,i}^2}{\sqrt{\sigma_{c,i}^2+1}}\right) \tag{A.39}$$

The integral in Equation A.20 is the integral of a Gaussian distribution:

$$\int_{-\infty}^{\infty} \mu_i^2 \exp\left(-\zeta^2/2\right) \mathrm{d}\zeta = \left[\mu_i^2 \sqrt{\frac{\pi}{2}} \mathrm{erf}\left(\frac{\zeta}{\sqrt{2}}\right)\right]_{-\infty}^{\infty} = \mu_i^2 \sqrt{2\pi} \tag{A.40}$$

Using the formula provided in [35], the integral in Equation A.21 equates to:

$$\int_{-\infty}^{\infty} \mu_i^2 \exp\left(-\zeta^2/2\right) \mathrm{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right) \mathrm{d}\zeta = \mu_i^2 \sqrt{2\pi} \mathrm{erf}\left(\frac{-\mu_{c,i}}{\sqrt{2(\sigma_{c,i}^2 + 1)}}\right) \tag{A.41}$$

Summarising:

$$\begin{aligned} &\int_{-\infty}^{\infty} (\zeta - \mu_i)^2 \exp\left(-\zeta^2/2\right) \left[1 + \mathrm{erf}\left(\frac{\zeta - \mu_{c,i}}{\sigma_{c,i}\sqrt{2}}\right)\right] \mathrm{d}\zeta \\ &\qquad = \left[\sqrt{2\pi}\left(\left(1 + \mu_i^2\right)\left(1 - \mathrm{erf}\left(\frac{\mu_{c,i}}{\sqrt{2(\sigma_{c,i}^2 + 1)}}\right)\right)\right)\right. \\ &\qquad\quad \left. + \frac{2}{\sqrt{\sigma_{c,i}^2 + 1}} \exp\left(-\frac{\mu_{c,i}^2}{2(\sigma_{c,i}^2 + 1)}\right)\left(\frac{\mu_{c,i}}{\sigma_{c,i}^2 + 1} - 2\mu_i\right)\right] \end{aligned} \tag{A.42}$$

Appendix B

Additional Kalman Filter Results

This appendix presents additional results for the soft-constrained Kalman Filter (KF) developed in Chapter 4. Consider a robot moving along a corridor. The corridor has a wall 10m in front of the initial position of the robot, and discrete position sensors placed at 1m intervals. The position sensors can detect which side of the sensor the robot is on. The robot has an initial velocity of 10cm/s and accelerates at 1cm/s^2 for 20 seconds, then decelerates at 1cm/s^2 for 20 seconds, before again accelerating at 1cm/s^2 until it reaches the far wall. Two types of robots were considered—one with standard deviations on the acceleration and initial velocity of $\sigma_a = 1\text{cm/s}^2$ and $\sigma_v = 3\text{cm/s}$ respectively (Robot A), and a more certain one with standard deviations on the acceleration and initial velocity of $\sigma_a = 0.5\text{cm/s}^2$ and $\sigma_v = 1.5\text{cm/s}$ respectively (Robot B). The standard deviation of the set-point of the sensors was also varied, with standard deviations (in cm) of $\sigma_s \in \{0, 5, 10, 15, 20, 25, 30\}$ tested. The position of each robot was tracked using the following KF run at 10Hz:

$$\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} \ddot{x} \end{bmatrix} \quad \boldsymbol{F} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \quad \boldsymbol{G} = \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix} \tag{B.1}$$

The covariance of the process noise was given by:

$$\boldsymbol{Q} = \boldsymbol{G}\boldsymbol{G}^T\sigma_a^2 = \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & \Delta t^2 \end{bmatrix} \sigma_a^2 \tag{B.2}$$

The KF was initialised with:

$$\hat{\boldsymbol{x}} = \begin{bmatrix} 0 \\ 0.1 \end{bmatrix} \quad \boldsymbol{P} = \begin{bmatrix} 0 & 0 \\ 0 & \sigma_v^2 \end{bmatrix} \tag{B.3}$$

A position sensor changing its reading was incorporated as a noisy measurement of the position:

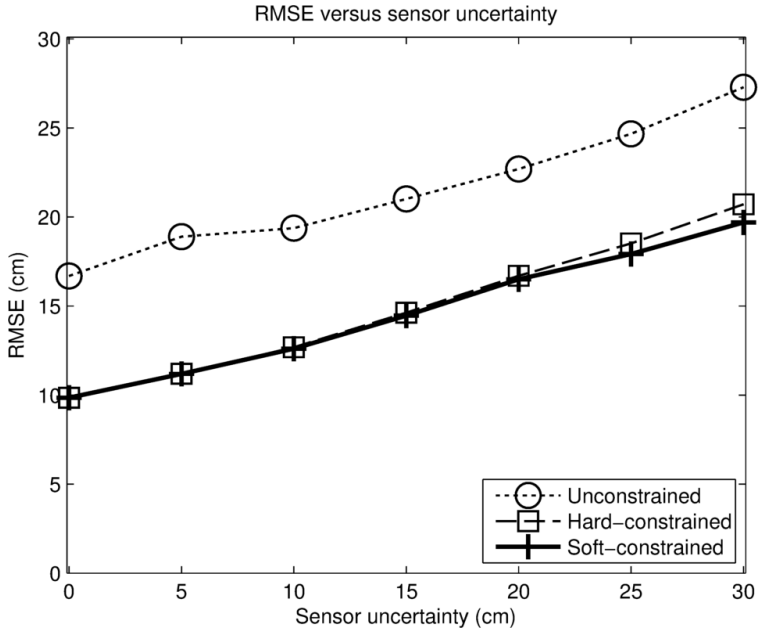
$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \boldsymbol{R} = \begin{bmatrix} \sigma_s^2 \end{bmatrix} \tag{B.4}$$

At each time step, position sensors whose reading did not change were not incorporated into the KF. The aim of the constrained KF approach here was to use the absence of measurements to improve the state estimate—while the robot was in between sensors, the sensors were treated as constraints on the state of the system. The truncation method presented in Chapter 4 was compared with an unconstrained KF, and a hard-constrained KF. Each combination of robot, sensor uncertainty, and KF method was tested 1000 times.

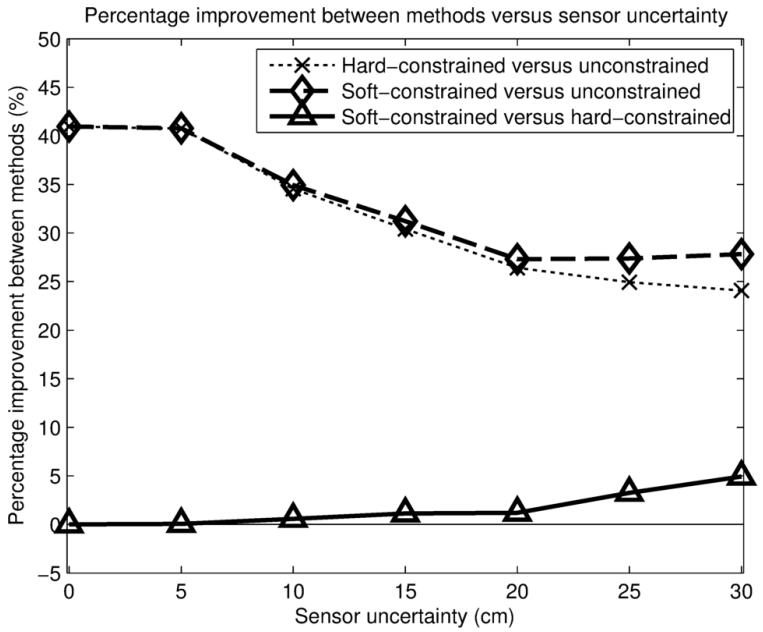
The time-average Root Mean Squared Error (RMSE) and percentage improvement between methods for Robot A are shown in Figure B.1, and the results for Robot B are shown in Figure B.2. For Robot A, both the hard-constrained and soft-constrained methods provided a significant benefit over the unconstrained KF, with an improvement of over 40% in tracking performance when the sensors have no uncertainty. As the uncertainty of the sensors was increased, the soft-constrained method slightly outperformed the hard-constrained method. The process noise for Robot B was significantly less than Robot A. As a result, the uncertainty of the sensors played a larger role in determining the performance of the methods. As can be seen in Figure B.2, the hard-constrained KF was significantly outperformed by the unconstrained KF once

the sensor uncertainty was above 10cm. In these cases, the estimate produced by the hard-constrained KF was overconfident, and the soft-constrained KF outperformed the hard-constrained KF by over 17%. An example of the overconfident estimates produced by the hard-constrained method is shown in Figure B.3. The soft-constrained KF strikes a balance between the high uncertainty of the unconstrained KF and the overconfident estimates of the hard-constrained KF.

One might argue that the truncated state estimate should be fed back into the KF as this will result in more confident estimates. Figure B.4 shows the effects of this feedback. As can be seen, the resultant estimate is more confident. However, this estimate can become overconfident and fail to accurately represent the actual state. For further discussion on this, please see Simon and Simon [94].

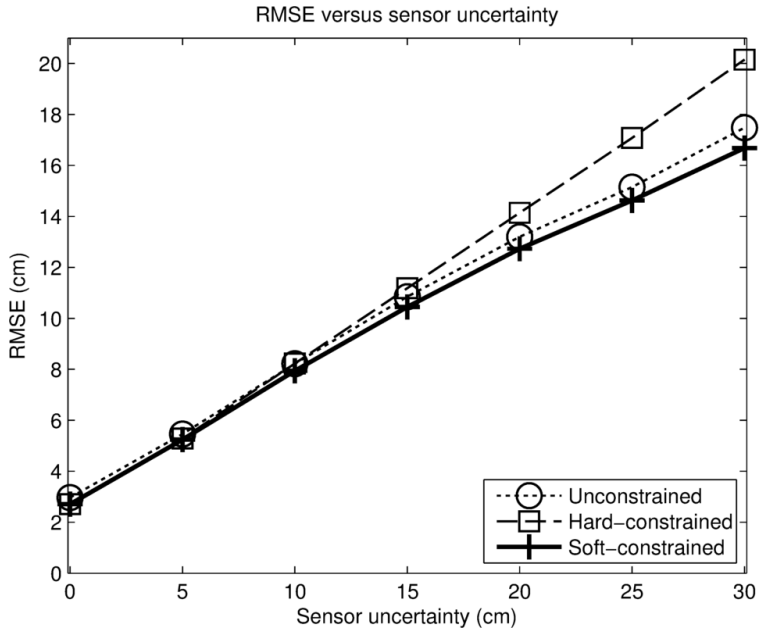


(a) RMSE

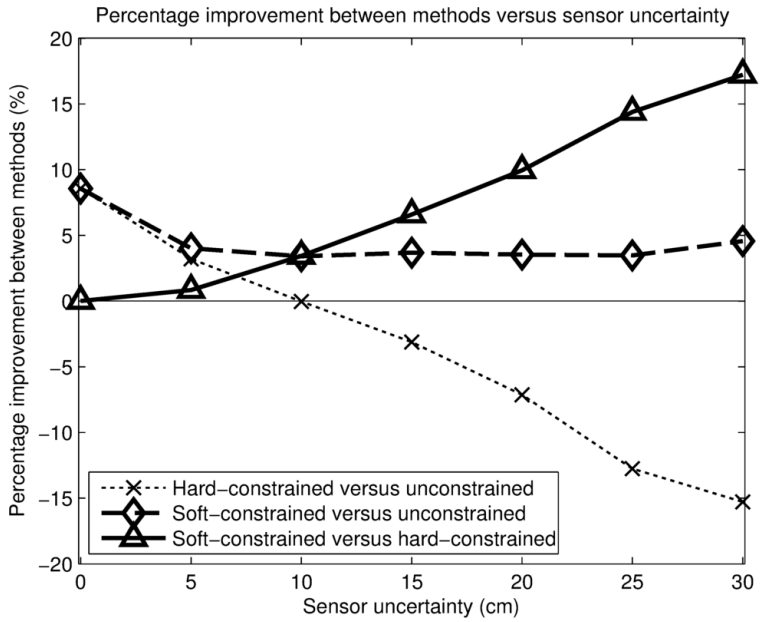


(b) Percentage improvement between methods

Figure B.1 – RMSE and percentage improvement between methods for Robot A as the sensor uncertainty is varied.

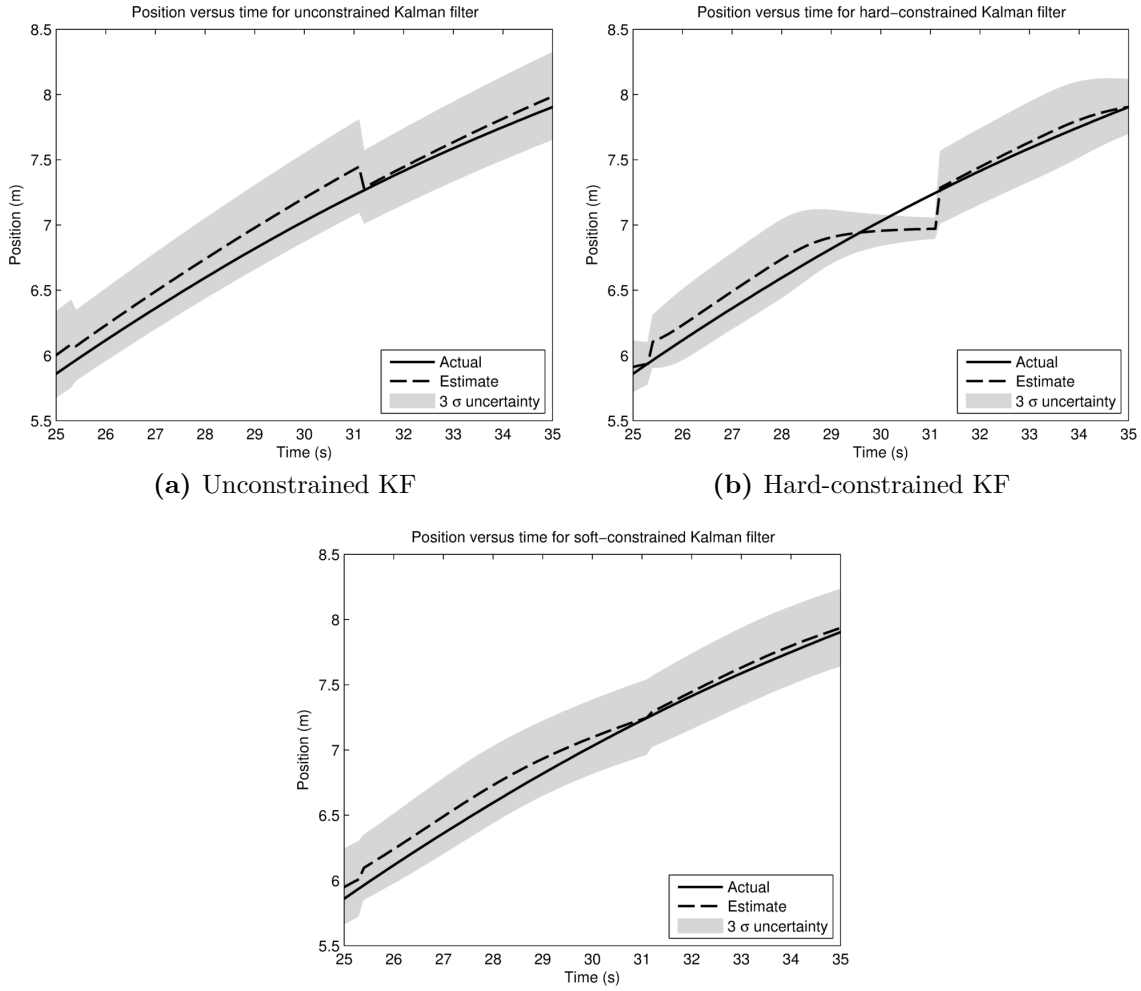


(a) RMSE



(b) Percentage improvement between methods

Figure B.2 – RMSE and percentage improvement between methods for Robot B as the sensor uncertainty is varied.

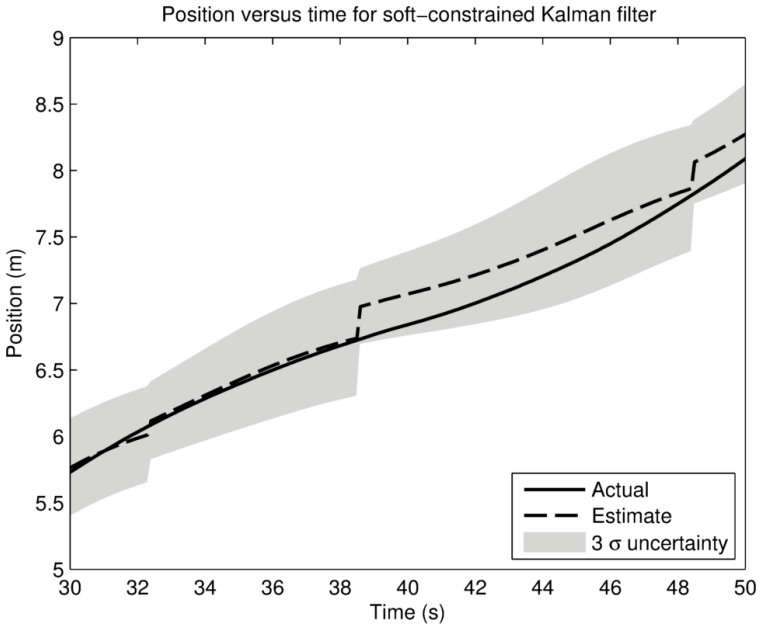


(a) Unconstrained KF

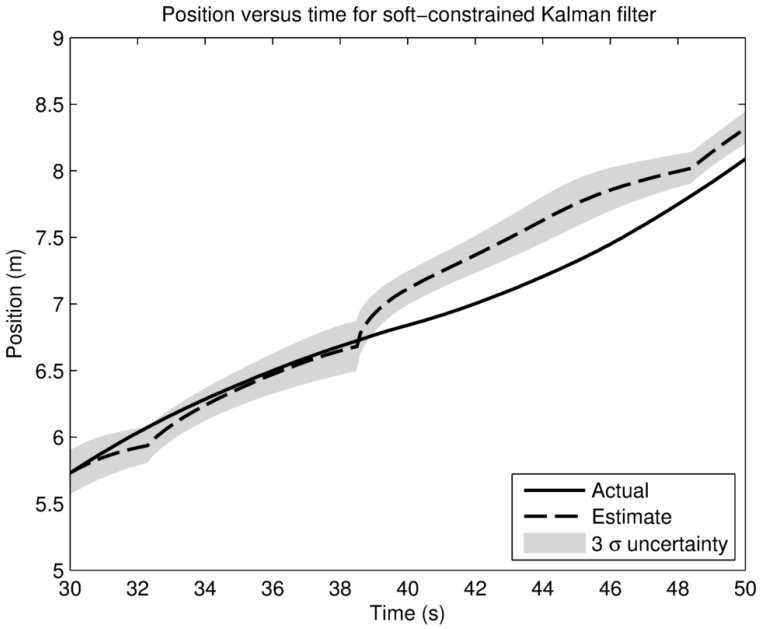
(b) Hard-constrained KF

(c) Soft-constrained KF

Figure B.3 – Comparison of the actual and estimated positions with uncertainty for Robot B with sensor uncertainty of 15cm. (a) shows the unconstrained KF. In (b), the hard-constrained KF is overconfident, resulting in a two-σ confidence interval that does not include the actual position. Using the soft-constrained KF in (c), the actual position stays within the uncertainty of the estimate.



(a) Soft-constrained KF without feedback



(b) Soft-constrained KF with feedback

Figure B.4 – Comparison of the actual and estimated positions with uncertainty for Robot A with sensor uncertainty of 15cm. In (a), the truncated estimate is not fed back into the Kalman filter, while in (b), the truncated estimate is used by the Kalman filter.