Limited Lookahead Control of Discrete-Event Systems: Cost, Probability, and State Space

By

Creag Winacott

A thesis submitted to the Graduate Program in Computing in conformity with the requirements for the Degree of Master of Science

Queen’s University
Kingston, Ontario, Canada
January, 2012

Copyright © Creag Winacott, 2012
Abstract

Discrete-Event systems (DES) is a framework in which problems are modelled as finite-state automata and a solution in the form of a supervisory control scheme can be automatically synthesized via an exhaustive search through the state space of the system. Various extensions to the standard DES framework have been introduced to allow it to be applied to a greater variety of problems. When the system in question is very large or varies with time, a limited lookahead policy can be adopted, in which control decisions are made on-the-fly by looking at finite-step projections of the behaviour of the system’s underlying automata. This work presents a new approach to limited lookahead supervision which incorporates many of the extensions to DES that are already present in the literature, such as event probability and string desirability. When dealing with a limited lookahead technique, the projected system behaviour is represented as a lookahead tree with some depth limit decided on by the user. It can be difficult to strike a balance between the complexities associated with storing and analyzing the trees and the amount of information available to make decisions, both of which increase with depth. This work also presents a set of methods which are designed to aid in accurately estimating the state space of lookahead trees with the intent of simplifying the process of determining a favourable depth to use. Finally, the approaches introduced herein are applied to a simulation of an infectious disease outbreak, primarily to showcase them in action, but also for the possibility of illuminating any useful information for real-world health units.
Acknowledgements

Thank you to Ann, Ed and James, my loving family for supporting me throughout this endeavour.

Special thanks to my supervisor, Karen Rudie, and the Control of Discrete-Event Systems Research Group at Queens University for fostering a wonderful environment for developing ideas. In particular, I thank Behnam Behinaein, Daniel Cownden, Lenko Grigorov and Sarah-Jane Whittaker for their contributions.

In addition, I thank my endearing friends, Corey Brown, Susie Crowe, Jacob de Vos, Murilo Guimaraes, Alexandra Pompeu-Robinson, Jesse Toninger and Bryan Walsh, for occasionally dragging me back to reality and keeping lit the light at the end of the tunnel.

Finally, I must thank my defense committee, Juergen Dingel and Abdol-Reza Mansouri, for their enlightening inquiry and for catching everything I had missed.
Table of Contents

Abstract ................................................................. ii

Acknowledgements ....................................................... iii

Chapter 1  Introduction ................................................. 1

Chapter 2  Background ................................................... 3
  2.1  Representation .................................................. 4
  2.2  FSA ............................................................... 5
     2.2.1  Properties .................................................. 7
     2.2.2  Operations .................................................. 7
  2.3  Supervisory Control of DES .................................... 9
     2.3.1  Small Factory Example .................................... 11
  2.4  Online Control .................................................. 16
     2.4.1  Limited Lookahead Policy ................................ 18
     2.4.2  Variable Lookahead Policy ............................... 22
  2.5  String Desirability .............................................. 27
  2.6  Probabilistic DES ............................................... 29

Chapter 3  Recursive Utility-Based Limited Lookahead ............. 31
  3.1  RUBLL Framework ............................................... 32
  3.2  RUBLL Pseudocode .............................................. 32
  3.3  MACHINE Example ............................................... 34
     3.3.1  RUBLL Approach .......................................... 35
     3.3.2  Markov Chain Approach .................................. 36
  3.4  B-RUBLL ......................................................... 39
  3.5  Implementation .................................................. 40

Chapter 4  State-Space Analysis ...................................... 44
  4.1  Estimating Tree State-Space ................................... 45
     4.1.1  Spectral Analysis ......................................... 45
     4.1.2  $\tau$-Estimation .......................................... 47
     4.1.3  Computing $\tau$ for Primitive, Diagonalizable Adjacency Matrices 48
     4.1.4  Examples ................................................... 50
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Computing $\tau$ for General Case</td>
<td>54</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Non-Mixing</td>
<td>54</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Component Analysis</td>
<td>58</td>
</tr>
<tr>
<td>4.3</td>
<td>Complexity Analysis</td>
<td>61</td>
</tr>
<tr>
<td>4.4</td>
<td>Algorithm</td>
<td>63</td>
</tr>
<tr>
<td>4.5</td>
<td>Discussion</td>
<td>66</td>
</tr>
</tbody>
</table>

**Chapter 5**  
Outbreak Simulation and Results  
5.1 Simulation Description | 68 |
| 5.1.1 | Health Unit Responses | 69 |
| 5.1.2 | Model of an Exposed Individual | 70 |
| 5.1.3 | Simulation Parameters | 72 |
| 5.1.4 | Simulation Results | 75 |
| 5.2 | RUBLL Implementation | 80 |
| 5.2.1 | Benefit Values | 85 |
| 5.3 | Offline Analysis | 88 |
| 5.3.1 | Scenario One: Smallpox | 88 |
| 5.3.2 | Scenario Two: Influenza | 96 |
| 5.4 | Online Analysis | 102 |
| 5.5 | Analysis Comparison | 103 |

**Chapter 6**  
Conclusion and Future Directions | 106 |

Bibliography | 108 |

Appendix A | 111 |
List of Figures

Figure 2.1  FSA representation of the machine from the Small Factory problem. State names have been abbreviated as I=IDLE, W=WORKING, D=DOWNS   6

Figure 2.2  Result of the parallel composition operation on two machines from the Small Factory problem. Numerical subscripts indicate each machine.  8

Figure 2.3  Supervisory control loop  9

Figure 2.4  Small Factory manufacturing line: machines coupled by a buffer  11

Figure 2.5  Machine i  11

Figure 2.6  Plant: $G = M_1 || M_2$  12

Figure 2.7  Buffer specification $E_1$. Self-loops allow irrelevant events to occur freely. State labels indicate the number of workpieces in the buffer.  12

Figure 2.8  Repair specification $E_2$. State labels indicate whether or not $M_2$ is in state $D_2$.  13

Figure 2.9  Combined specification: $E = E_1 \times E_2$  13

Figure 2.10  Supremal Controllable Sublanguage for Small Factory problem.  15

Figure 2.11  3-step unwinding of FSA into lookahead tree  17

Figure 2.12  Limited lookahead supervisory control loop  17

Figure 2.13  Limited lookahead Tree with $N = 3$ computed with optimistic strategy for the Small Factory problem following string $\alpha_1\beta_1$.  20

Figure 2.14  Limited lookahead Tree with $N = 3$ computed with conservative strategy for the Small Factory problem following string $\alpha_1\beta_1$.  21
<table>
<thead>
<tr>
<th>Number</th>
<th>Figure Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.15</td>
<td>Limited lookahead Tree with $N = 3$ computed with optimistic strategy for the Small Factory problem following string $\alpha_1\beta_1\alpha_2$.</td>
<td>23</td>
</tr>
<tr>
<td>2.16</td>
<td>Limited lookahead Tree with $N = 3$ computed with optimistic strategy for the Small Factory problem following string $\alpha_1\beta_1\alpha_2$.</td>
<td>24</td>
</tr>
<tr>
<td>2.17</td>
<td>Limited lookahead Tree with $N = 3$ computed with VLP algorithm for the Small Factory problem following string $\alpha_1\beta_1$.</td>
<td>27</td>
</tr>
<tr>
<td>2.18</td>
<td>Limited lookahead Tree with $N = 3$ computed with VLP algorithm for the Small Factory problem following string $\alpha_1\beta_1\alpha_2$.</td>
<td>28</td>
</tr>
<tr>
<td>3.1</td>
<td>FSA representation of MACHINE example.</td>
<td>34</td>
</tr>
<tr>
<td>3.2</td>
<td>5-step lookahead tree for MACHINE in BROKEN state.</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>Transition matrix, normalized eigenvector, state and global utility for each control pattern. Eigenvector and global utility values truncated at the fourth decimal place.</td>
<td>38</td>
</tr>
<tr>
<td>3.4</td>
<td>Example trees unwound to depths (a) 2, (b) 3, and (c) 4.</td>
<td>41</td>
</tr>
<tr>
<td>4.1</td>
<td>(a) Example FSA; (c) resulting lookahead tree unwound from state 1 to depth 4; and (b) corresponding adjacency matrix.</td>
<td>45</td>
</tr>
<tr>
<td>4.2</td>
<td>The $\tau$ parameter for depths 0 through 20 for the FSA shown in Figure 4.1(a). The steady-state value for $\tau$ is determined by the expression in (4.5) in Section 4.1.3.</td>
<td>49</td>
</tr>
<tr>
<td>4.3</td>
<td>(a) Non-mixing, bipartite, alternating growth FSA; (b) resulting lookahead tree unwound from state 1 to depth 4; and (c) $\tau$ for depths 0 through 20.</td>
<td>56</td>
</tr>
<tr>
<td>4.4</td>
<td>(a) Non-mixing, bipartite FSA with four $\tau$ values and (b) resulting lookahead tree unwound from state 1 to depth 5.</td>
<td>58</td>
</tr>
<tr>
<td>4.5</td>
<td>FSAs with siphoning components. (a) has small growth siphoning component; (b) has large growth siphoning component.</td>
<td>59</td>
</tr>
<tr>
<td>4.6</td>
<td>FSA with identical components ${2, 3}$ and ${4, 5}$.</td>
<td>60</td>
</tr>
<tr>
<td>4.7</td>
<td>Non-invertible FSA with connected identical components ${3, 4}$ and ${5, 6}$.</td>
<td>61</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>FSA representation of an exposed individual [7]</td>
<td>71</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>Distribution of infected individuals over 10000 runs using quarantine strategy with immediate treatment for scenario two</td>
<td>79</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>Lookahead window for adapted RUBLL implementation</td>
<td>81</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>Probabilistically weighted exposure feedback arcs as they are appended to the model of an individual</td>
<td>91</td>
</tr>
<tr>
<td>Figure 5.5</td>
<td>Transition matrix for quarantine strategy with delayed treatment ($\gamma_3$) with weighted exposure feedback arcs, and normalized eigenvector corresponding to the largest eigenvalue.</td>
<td>91</td>
</tr>
<tr>
<td>Figure 5.6</td>
<td>Representative adjacency matrix for quarantine with delayed treatment strategy ($\gamma_3$).</td>
<td>93</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4.1</td>
<td>Estimated and actual state spaces for Figure 4.1(a) FSA</td>
<td>51</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Estimated and Actual state spaces and run times in milliseconds, and calculated % error of run times for RUBLL algorithm implemented on trees unwound from WORKING state in the MACHINE problem from Section 3.3</td>
<td>53</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Estimated and Actual state spaces and run times in milliseconds, and calculated % error of run times for RUBLL algorithm implemented on trees unwound from BROKEN state in the MACHINE problem from Section 3.3</td>
<td>54</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Assumed parameters for first simulation (smallpox)</td>
<td>74</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Assumed parameters for second simulation (influenza)</td>
<td>75</td>
</tr>
<tr>
<td>Table 5.3</td>
<td>Simulation results for scenario one (smallpox)</td>
<td>77</td>
</tr>
<tr>
<td>Table 5.4</td>
<td>Simulation results for scenario two (influenza)</td>
<td>78</td>
</tr>
<tr>
<td>Table 5.5</td>
<td>Simulation results for scenario two comparing $\xi_c = 0.99$, $\xi_c = 0.9$ and $\xi_c = 0.8$ using the quarantine strategy with delayed treatment</td>
<td>79</td>
</tr>
<tr>
<td>Table 5.6</td>
<td>Simulation results for scenario two comparing $\xi_c = 0.99$, $\xi_c = 0.9$ and $\xi_c = 0.8$ using the quarantine strategy with immediate treatment</td>
<td>80</td>
</tr>
<tr>
<td>Table 5.7</td>
<td>Utility values for depths 1 through 12 computed from the initial state of the simulation for Scenario One. Values are rounded to 2 decimal places.</td>
<td>88</td>
</tr>
<tr>
<td>Table 5.8</td>
<td>Estimated and actual state spaces for Scenario One with $\gamma_3$ rounded to 2 decimal places.</td>
<td>94</td>
</tr>
<tr>
<td>Table 5.9</td>
<td>Utility values for depths 1 through 11 computed from the initial state of the simulation for Scenario Two. Values are rounded to 2 decimal places.</td>
<td>96</td>
</tr>
</tbody>
</table>
Table 5.10 Utility values for depths 1 through 10 computed from the initial state of the simulation and growth rates for Scenario Two with $\xi$ set to 0.99, 0.9 and 0.8. Utility values are rounded to 2 decimal places and growth rates are rounded to 8. .......................... 98

Table 5.11 Estimated and actual state spaces for Scenario Two with $\gamma_4$ rounded to 2 decimal places. .............................. 101

Table 5.12 Estimated and actual run times for Scenario Two with $\gamma_4$, rounded to 2 decimal places. .............................. 102

A.1 Utility values for depths 1 through 50 computed from the initial state of the simulation for Scenario One. Values are rounded to 2 decimal places. .............................. 111

A.1 Utility values for depths 1 through 50 computed from the initial state of the simulation for Scenario One. Values are rounded to 2 decimal places. .............................. 112

A.2 Utility values for depths 1 through 50 computed from the initial state of the simulation for Scenario Two. Values are rounded to 2 decimal places. .............................. 112

A.2 Utility values for depths 1 through 50 computed from the initial state of the simulation for Scenario Two. Values are rounded to 2 decimal places. .............................. 113

A.2 Utility values for depths 1 through 50 computed from the initial state of the simulation for Scenario Two. Values are rounded to 2 decimal places. .............................. 114
Chapter 1

Introduction

Human beings are avid decision makers. Think of the sheer number of decisions that we make every day. We choose what to wear, what to eat, where to go, who to associate with and how to spend our time. Our lives are completely defined by these choices; making decisions is a staple of our existence. Another staple of human existence is the ongoing struggle to improve our lives by outsourcing work to tools and technology. Decision making is one of the more complex tasks that we perform, but with the advent of the microchip we have finally undertaken the task of designing technology systems to help make decisions for us.

In recent decades the development of new technologies has continued to accelerate, and the complexity of these systems has grown equivalently. With such a rapid increase in complexity it becomes more difficult to ensure that these systems will perform as intended, and yet they are increasingly being incorporated into our everyday lives in the form of communication networks, monitoring and control systems in automobiles [26], air traffic control systems [35], transportation networks [18] and so on. A set of operational rules is required to ensure the desired behavior of these systems, and to eliminate the possibility of system failure, which can typically result in large monetary losses, or even loss of life in the case of safety-critical systems. Traditionally these systems have been controlled by a set of rules modeled by differential and difference equations, but the increase in system complexity is making this approach less feasible.

As a result of the nature of human perception, the dynamics of these man-made systems can largely be interpreted as a set of discrete events. Discrete Event Systems (DES) is a relatively new approach which models these systems in an abstracted form so as to reduce the complexity inherent in representing them with a series of equations. The DES framework has been extended by Ramadge and Wonham [31],[38],[30] so that once a system has been modeled, along with its intended behavior, a control scheme to ensure the desired behavior can be generated automatically by an exhaustive search algorithm. However, systems are becoming so complex that it is not always feasible to represent them fully, especially in cases where the system is understood as a set of interacting modules. In these cases the modules themselves may be fairly simple to model, but the model representing every possible ordering of interactions between
them may be exceedingly large. Additionally, systems may not be fully understood at the time when a control scheme is required. In the case of dynamic systems, modules of the system may arbitrarily be added to or removed from the system at any point in time; this kind of time-varying behavior makes an exhaustive state-space search impossible. In cases where the finite-state automata vary over time or are very large, a limited lookahead supervision technique (as introduced by Chung, Lafortune and Lin [13]) can be employed. With this approach a supervisory decision is made for the current state by exhaustively exploring a finite-step projection of system behaviour (referred to as a lookahead tree), rather than exploring the full system itself. For further introduction to the limited lookahead approach, the reader is directed to [12], [11] and [23].

Much work has been done to extend the DES framework to apply to a wider variety of applications. As a result, the DES field has become segregated into specialized approaches to specific problems. The purpose of the work presented here is to advocate an amalgamation of the DES approaches by defining a limited lookahead algorithm that is as versatile as possible while still adhering to the existing frameworks. In particular, this work represents the first consolidation of limited lookahead, event probability and string desirability into a single supervisory control algorithm. This algorithm is referred to as the Recursive Utility-Based Limited Lookahead (RUBLL) algorithm. Chapter 2 deals with introducing the existing supervisory algorithms in detail from the ground up. Chapter 3 is devoted to the RUBLL algorithm which builds on the previous work on limited lookahead supervision and focuses on application to a wider class of problems. Chapter 4 presents an investigation into the state space of lookahead trees, which is an intrinsic aspect of the limited lookahead approach. This investigation results in a set of methods for calculating exact and estimated state-space values under different conditions, which are discussed in detail. Chapter 5 demonstrates the application of RUBLL to an infectious disease outbreak simulation. Finally, Chapter 6 summarizes the contributions of this work and provides some suggestions for future research. Some of the material in Chapter 3 first appeared in [36] and the material in Chapter 4 will appear in [37].
Chapter 2

Background

A system is some part of the universe that has been conceptually encapsulated in order to study or define its behaviour by how it interacts with the rest of the universe. By this definition essentially anything can be considered a system. Adopting a computer science approach, we can define a system’s interaction with the universe via inputs and outputs. The input is how the universe acts on the system and changes its state, and the output is the system’s response to the universe based on that state. Discrete-Event Systems (DES) are a class of systems whose inputs, states and outputs can be fully described by discrete values. More specifically, the state of a DES can be characterized by the serial occurrence of discrete events that happen instantaneously at arbitrary points in time. A discrete event is something that either happens or does not, whereas continuous events happen gradually over some amount of time, and so can partially have happened. Modelling of DES is a useful approach when dealing with problems in which continuous time is not important and only the order that events occur in matters. This may seem limiting at first, but consider that most man-made systems fall under this category. For example, the state of a digital system is unaffected by small variances in the speed of its components.

Many natural systems can be simplified and interpreted as discrete-event as well. For example, the spread of an infectious disease through a population has been modeled as a DES with probabilistic events [8]. Whether or not a system is suitably representable as discrete-event is a decision that must be made on a case by case basis by the modeller. From these definitions it follows that a large amount of the problems being modelled today can be interpreted accurately as discrete.

The remainder of this chapter will cover representation, operations, and the supervisory techniques already established in the literature. Throughout these sections the content will be illustrated using a running example; a classic problem from [31], which is referred to here as the Small Factory problem.

At the heart of the Small Factory problem is a machine which processes workpieces. The machine is initially in an IDLE state at which point it can accept a workpiece and start to process it, which places the machine in a WORKING state. While working the machine can either finish processing the workpiece, which returns it to the IDLE state, or it can break down which places it in the DOWN state. When the machine
is DOWN, repairing it will return it to the IDLE state.

2.1 Representation

A DES can be represented mathematically as a regular language, where the set of words expressible by the regular language define exactly the set of event strings producible by the DES. A regular language can be described by a regular expression, so the behaviour of a DES can be expressed as a regular expression. For example, the machine from the Small Factory can be represented as a regular expression.

\[(\alpha(\beta + \gamma\mu))^*\]

where
\[
\begin{align*}
\alpha &= \text{Start processing workpiece} \\
\beta &= \text{Finish processing workpiece} \\
\gamma &= \text{Break down} \\
\mu &= \text{Repair machine}
\end{align*}
\]

Regular expressions are not very visually intuitive to work with, so other representations are usually employed. In practice DESs have been successfully modeled using temporal logic [27], Petri-nets [29], and statecharts [7]; however, the majority of the work done has focused on using a finite-state automata (FSA) representation for a number of reasons:

1. FSAs are simple and intuitive. Both the creation of FSA models and their interpretation are in tune with human perception. When we look at a problem, our minds automatically start to break it down into discrete problem states and discrete occurrences that change one state into another. This makes translating a mental model of a problem into an FSA very straightforward. Conversely, when we look at a visual representation of an FSA, a state transition diagram, our eyes are automatically drawn towards the labeled ellipses (representing states) because they are visually salient. Once our gaze settles on a specific state, it is naturally drawn from that state to others via the arrows (representing events) connecting them. At first glance, our eyes will automatically run random simulations of the behaviour of an FSA.

2. FSAs are associated with a number of composition and analysis techniques. Many useful algorithms exist that allow FSAs to be minimized, combined or otherwise manipulated. These well established operations are instrumental in
creating a soundly abstracted FSA model of a system. For instance, a complicated system may be more readily modeled as a set of separate but interacting modules. Also, the structure of an FSA is essentially a directed graph, so when necessary there are a number of analysis techniques that can be adapted from graph theory.

3. FSAs are well established in language theory as being generatively equivalent to a regular language [5]. This means that all of the operations for regular expressions can be employed to reason about the behaviour of FSAs, which is of great use in proofs of correctness. For instance, an FSA minimization algorithm can be proved to be correct by showing that the minimized FSA will produce the same language as the original every time. Some work has been done in defining other representation techniques in terms of regular expressions. An extension of temporal logic, called Sugar, has been shown to be equivalent to regular expressions [3]; and an extension of regular expressions, called concurrent regular expressions have been shown to be equivalent to Petri nets [16] and statecharts [21]. Clearly there is a hierarchy of expressability of different representations; however, for the set of problems that can be expressed as a regular language (the set of problems we are interested in) it is most beneficial to use FSAs.

2.2 FSA

A finite-state automaton is represented as a 5-tuple.

\[ A = (\Sigma, Q, \delta, q_0, Q_m) \]

- \( \Sigma \) is the alphabet; a non-empty, finite set of symbols which correspond to events in the system.

- \( Q \) is the set of states; a non-empty, finite set of symbols which correspond to states in the system.

- \( \delta : \Sigma \times Q \rightarrow Q \) is the transition function. This function is only defined when the event in question transitions out of the state in question, so it is a partial function. For example, \( \delta(q, \sigma) = q' \) means that event \( \sigma \) transitions from state \( q \) to \( q' \). The notation \( \delta(q, \sigma)! \) is used to express that \( \delta(q, \sigma) \) is defined.

- \( q_0 \in Q \) is the initial state; the starting state of the system.
• $Q_m \subseteq Q$ is the set of marked states; states in which the system may terminate, indicating that the string of events leading to it is a member of the corresponding regular language.

For example, the machine from the Small Factory problem would be represented as follows:

$\Sigma = \{\alpha, \beta, \gamma, \mu\}$
$Q = \{\text{IDLE, WORKING, DOWN}\}$
$\delta(\text{IDLE}, \alpha) = \text{WORKING}$
$\delta(\text{WORKING}, \beta) = \text{IDLE}$
$\delta(\text{WORKING}, \gamma) = \text{DOWN}$
$\delta(\text{DOWN}, \mu) = \text{IDLE}$
$q_0 = \text{IDLE}$
$Q_m = \{\text{IDLE}\}$

![FSA representation of the machine from the Small Factory problem. State names have been abbreviated as I=IDLE, W=WORKING, D=DOWN](image)

The transition function is recursive in nature, i.e., $\delta(\delta(q, \alpha), \beta) = q'$ means that the string $\alpha\beta$ will transfer the system from state $q$ to $q'$ through some intermediate state. This nesting of functions can become very large with the size of the event string, so it makes sense to extend the transfer function to deal with strings: $\delta' : \Sigma^* \times Q \rightarrow Q$.

$\delta'(q, \epsilon) = q$

$\delta'(q, s\sigma) = \delta(\delta'(q, s), \sigma)$, where $s \in \Sigma^*$ and $\sigma \in \Sigma$

For example, $\delta'(\text{IDLE}, \alpha\beta) = \text{IDLE}$. Typically, $\delta'$ is also denoted by $\delta$, and is used when dealing with strings of events as input. An FSA $A$ has two languages associated with it; the generated language $L(A)$ and the accepted language $L_m(A)$. The generated language is the set of all possible strings of events that can be produced; it is defined
as \( L(A) := \{ s|\delta(q_0, s)! \} \). The accepted language is the set of all event strings that terminate in a marked state; it is defined as \( L_m(A) := \{ s|\delta(q_0, s) \in Q_m \} \). Generally, the set of marked states are chosen to recognize some interesting subset of behaviour of the system.

### 2.2.1 Properties

Two FSAs are said to be equivalent if they generate and recognize the same language. An FSA is said to be accessible if there is a path from the initial state to every state; that is, \( \delta(q_0, s) = q \) is defined by some \( s \) for all \( q \). An FSA is said to be coaccessible or non-blocking if there is a path from every state to some marked state; that is, \( \delta(q, s) \in Q_m \) for some \( s \). When an FSA is both accessible and coaccessible it is called trim, and has the interesting property that the prefix closure of its accepted language is equal to its generated language. Prefix closure is an operation that takes some language \( K \) and returns the language made up of all possible prefixes of strings of \( K \), denoted \( \overline{K} \). So a trim FSA \( A \) will have the property:

\[
L_m(A) = L(A)
\]

### 2.2.2 Operations

There are two primary operations that allow FSAs to be combined; product and parallel composition. The intersection of two FSAs, denoted \( A_1 \times A_2 \), is the FSA whose generated and accepted languages are equal to the intersection of the languages generated and accepted respectively by \( A_1 \) and \( A_2 \). i.e., \( L(A_1 \times A_2) = L(A_1) \cap L(A_2) \) and \( L_m(A_1 \times A_2) = L_m(A_1) \cap L_m(A_2) \). For a transition to be present in \( A_1 \times A_2 \), it must be present in both \( A_1 \) and \( A_2 \). Formally,

if \( A_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, Q_{m1}) \) and \( A_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, Q_{m2}) \),

then \( A_1 \times A_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{01}, q_{02}), Q_{m1} \times Q_{m2}) \),

where

\[
\delta((q_1, q_2), \sigma) = \begin{cases} 
\delta_1(q_1, \sigma), \delta_2(q_2, \sigma) & \text{if } \delta_1(q_1, \sigma)! \text{ and } \delta_2(q_2, \sigma)!
\end{cases}
\]

The product operation is useful for modeling two or more systems that must execute the same events at the same time. It is somewhat restrictive in practice because it only allows transitions that are common to both systems, and often results are not coaccessible. In order to allow events private to certain FSAs to happen independent of the others, while still maintaining the synchronicity of shared events,
parallel composition should be used. The parallel composition, or \textit{synchronous product} of FSAs \(A_1\) and \(A_2\) is given by \(A_1||A_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{01}, q_{02}), Q_{m_1} \times Q_{m_2})\), where

\[
\delta((q_1, q_2), \sigma) = \begin{cases} 
(q_1, \delta_1(q_1, \sigma), q_2) & \text{if } \delta_1(q_1, \sigma) \neq \text{undefined} \text{ and } \sigma \notin \Sigma_2 \\
(q_2, \delta_2(\sigma, q_1)) & \text{if } \delta_2(q_2, \sigma) \neq \text{undefined} \text{ and } \sigma \notin \Sigma_1 \\
(\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) & \text{if } \delta_1(q_1, \sigma) \neq \text{undefined} \text{ and } \delta_2(q_2, \sigma) \neq \text{undefined} \text{ and } \sigma \notin \Sigma_1 \\
\text{undefined} & \text{otherwise}
\end{cases}
\]

This operation is very useful in practice because it enforces the synchronization of events that are common to more than one FSA, but otherwise doesn’t affect the behaviour of the overall system. This is ideal for modeling a complex system as a set of interacting modules. When no events are shared, strings can interleave freely, and this is referred to as a \textit{shuffle} operation. Note that when all events are shared, parallel composition is equivalent to the product operation.

For example, we can introduce a second machine to the Small Factory problem as a separate FSA and calculate the parallel composition of the two automata to obtain a model for both machines running in parallel (depicted in Figure 2.2).

There are no shared events between the two machines, so this can be considered a shuffle operation.


CHAPTER 2. BACKGROUND

2.3 Supervisory Control of DES

A DES can be modeled as an FSA to give some insight into the structure of its behaviour. However, this process does not aid in determining how a system should be altered so that it behaves more desirably. Really what is required is to be able to specify how we would like a system to behave, and then determine how the system should be changed or controlled to ensure that it behaves as desired. This is exactly what Ramadge and Wonham accomplished with their supervisory control framework [31, 38, 30], referred to as the Ramadge-Wonham (RW) framework.

This approach introduces the concept of a *supervisor*, which is an entity that has the ability to exert some limited amount of control on the system, denoted $S$. Under this framework a DES is modeled as an FSA and is referred to as the *plant*, denoted $G = (\Sigma, Q, \delta, q_0, Q_m)$. The event set is divided into two disjoint subsets of *controllable* events, $\Sigma_c$, and *uncontrollable* events, $\Sigma_{uc}$; where $\Sigma = \Sigma_c \cup \Sigma_{uc}$, and $\Sigma_c \cap \Sigma_{uc} = \emptyset$. Controllable events correspond to occurrences that can be averted by exerting some kind of control on the system, so the supervisor has the ability to disable them. Uncontrollable events are those that cannot be disabled by any means; these usually correspond to unexpected or undesirable occurrences in the system, such as a failure of some kind. Ideally, the supervisor will operate on the plant by disabling certain events depending on the current state to ensure that only desired behaviour will result (depicted in Figure 2.3). This interaction is called a *closed-loop* system, denoted $S/G$.

![Figure 2.3: Supervisory control loop](image)

A supervisor can be represented as an FSA, the states of which correspond to states in $G$, and the presence of events indicates that they are enabled at that state. Alternatively, a supervisor can be thought of as a feedback map, a function that takes states and events as input and returns a disablement value. Desirable behaviour of the system is also modeled as an FSA $E$, called the *specification* or *legal language*. The specification is designed so that it generates all of the behaviours of the plant that are not forbidden, so $L(E) \subseteq L(G)$. States that are not in $E$ are referred to as *illegal* states. The problem of supervisory control is synthesizing a supervisor such
that

\[ L(S/G) \subseteq E \]

A specification \( E \) is said to be controllable with respect to \( G \) if \( s \in L(G) \) implies that \( s \sigma \in L(E) \) for all \( s \in L(E) \) and all \( \sigma \in \Sigma_{uc} \), i.e., if \( G \) is unable to uncontrollably enter an illegal state. Assuming that \( L(E) \subseteq L(G) \), \( L(E/G) = L(E) \) so \( E \) can be used as a supervisor that enforces itself. Otherwise \( E \) can be composed synchronously with \( G \) to produce a suitable supervisory automaton.

If \( E \) is not controllable with respect to \( G \) it means that there is an unsafe state in \( G \), meaning that it has an uncontrollable event that leads to another state which is either illegal or unsafe. When this is the case, a supervisor must be synthesized to ensure that no illegal or unsafe states can ever be reached by the system. This is accomplished by computing the supremal controllable sublanguage of \( E \), that is, the largest sublanguage of \( E \) that is controllable with respect to \( G \), denoted \( \text{sup} C(E) \). A formula for the computation of \( \text{sup} C(E) \) is given in [38], and a recursive algorithm for computing \( \text{sup} C(E) \) is given in [9] with stated complexity \( O(n^2m^2|\Sigma|) \), where \( n \) and \( m \) are the number of states in \( E \) and \( G \) respectively. Recursive computation of \( \text{sup} C(E) \) is accomplished by taking the synchronous composition of \( G \) and \( E \) and iteratively trimming off the illegal and unsafe states. It has been shown that if the languages in question are prefix-closed, \( \text{sup} C(E) \) can be computed non-recursively with complexity \( O(mn^2) \) [6]. The supremal controllable sublanguage can be used as a supervisor that allows the plant the most freedom of execution while ensuring that no illegal behavior results. This is referred to as a maximally permissive supervisor. This approach to supervisor synthesis can be used universally because if \( E \) is controllable with respect to \( G \), then \( \text{sup} C(E) = E \).

\[ \text{sup} C(E) = S \text{ such that } L(S/G) \subseteq E \]

The supremal controllable sublanguage represents a supervisory solution to the problem by defining which events should be disabled at each state to ensure the specifications are met. The supervisor \( \text{sup} C(E) \) can be represented as a control mapping, which can be more straightforwardly implemented in real-world problems. A control mapping \( \psi : Q \times \Sigma \rightarrow \{0, 1\} \) is a function that maps states and events in \( G \) to control decisions, where 0 represents ‘disable’ and 1 represents ‘enable’. It is defined as

\[
\psi(q, \sigma) = \begin{cases} 
1 & \text{if } \exists \ s \sigma \in \text{sup} C(E) \text{ such that } \delta(q_0, s) = q \\
0 & \text{otherwise}
\end{cases}
\]
This control mapping completes the cycle of translating a real-world problem into a representational form, computing a supervisor and converting it back to a real-world solution. Now the RW supervisory algorithm will be applied to the Small Factory problem in its entirety as it appeared in [31].

2.3.1 Small Factory Example

The Small Factory problem is made up of two machines \( M_1 \) and \( M_2 \) that are connected by a buffer. Both machines start off in an IDLE state, and accepting a workpiece places them in a WORKING state. Machine 1 accepts workpieces \( (\alpha_1) \) from an input line and processes them. When it completes a workpiece \( (\beta_1) \) it outputs it into the buffer, which has a capacity for one workpiece. Machine 2 accepts workpieces \( (\alpha_2) \) from the buffer and ejects them into an output bin when it finishes them \( (\beta_2) \). Either machine can break down \( (\gamma_i) \) while working on a part, which will cause the part to be discarded. Broken down machines can be repaired \( (\mu_i) \), returning them to the idle state. The manufacturing line is depicted in Figure 2.4.

![Figure 2.4: Small Factory manufacturing line: machines coupled by a buffer](image)

Each machine is modeled identically as shown in Figure 2.5. Accepting workpieces and repairing machines are considered controllable events \( (\Sigma_c = \{\alpha_i, \mu_i\}) \), while finishing workpieces and breaking down are uncontrollable \( (\Sigma_{uc} = \{\beta_i, \gamma_i\}) \). Controllable events are indicated by a perpendicular tick on the transition arc.

![Figure 2.5: Machine i](image)

The possible behaviours of the two machines are represented in Figure 2.6. Note that this figure is similar to Figure 2.2, but here information about which events are controllable is displayed. The convention on state labels is that the first letter indicates the state of machine 1, and the second indicates the state of machine 2. Now
the specified behaviour of the system should be modelled. There are two requirements of the system in order for it to run smoothly. The first is that no buffer overflow or underflow can occur, this specification is shown in Figure 2.7. The second requirement is that when both machines are broken down machine 2 should be repaired first, shown in Figure 2.8.

The two specifications can be combined using the product operation, which forces shared events to synchronize resulting in the restrictions of both persisting in the product, shown in Figure 2.9.

Now the controllability of the desired system behaviour $E$ can be checked with
Figure 2.8: Repair specification $E_2$. State labels indicate whether or not $M_2$ is in state $D_2$.

Figure 2.9: Combined specification: $E = E_1 \times E_2$
respect to the possible system behaviour $G$. Recall that if there exists a string $s \in L(E)$ and an event $\sigma \in \Sigma_{uc}$ such that $s\sigma \in L(G)$ and $s\sigma \notin L(E)$, then $E$ is not controllable with respect to $G$. This basically means that there is some string $s$ which is legal, but leads to a state with an uncontrollable event $\sigma$ which transitions into an illegal state. In the Small Factory problem there is an example of this which causes buffer overflow. When $M_1$ accepts a workpiece when the buffer is already full, this puts it in a position to uncontrollably finish processing the workpiece and overflow the buffer. More formally, when $s = \alpha_1\beta_1\alpha_1$ and $\sigma = \beta_1$, the conditions for controllability are violated, so $E$ is not controllable with respect to $G$. Note that $E_2$ is controllable with respect to $G$ because both repair events are controllable, so from state $DD\mu_1$ can always be disabled until $\mu_2$ has occurred. Also note that the avoidance of buffer underflow is controllable with respect to $G$ because $\alpha_2$ can be disabled when the buffer is empty. However, because buffer overflow is not controllable with respect to $G$, the combined specification $E$ is not either because it contains the same problem behaviour. In the case that $E$ is not controllable with respect to $G$, $\text{sup}_C(E)$ must be calculated, that is, the largest sublanguage of $G$ with respect to which $E$ is controllable. This supremal controllable sublanguage can be used as a supervisor that enforces the specifications. The supremal controllable sublanguage of the small factory problem is shown in Figure 2.10.

This supervisor is not very illuminating when presented as an automaton, which is a good reason to translate it to a control mapping. In the small factory problem, much of the legal language $E$ is controllable with respect to $G$, so it makes sense to define the control mapping in terms of how it differs from $E$. First $\psi$ is initialized with $E$,

$$\psi(q,\sigma) = \begin{cases} 1 & \text{if } \exists s\sigma \in E \text{ such that } \delta(q_0, s) = q \\ 0 & \text{otherwise} \end{cases}$$

Then $\psi$ can be overridden with the following three entries:

$$\psi(II2, \alpha_1) = 0$$
$$\psi(IW2, \alpha_1) = 0$$
$$\psi(ID2, \alpha_1) = 0$$

These entries illustrate how disallowing $M_1$ to start a new workpiece when the buffer is full is the only control decision required to ensure $E$ beyond those defined by $G \times E$. These entries can be found by looking at $E - \text{sup}_C(E)$, or they can be kept track of during the trimming process of the computation of $\text{sup}_C(E)$.

This example has shown how the RW framework can be utilized to compute a solution to the small factory problem. There are some situations when the RW
Figure 2.10: Supremal Controllable Sublanguage for Small Factory problem.
framework is not applicable however. The algorithm for computing $\text{sup}\mathcal{C}(E)$ requires an exhaustive state-space exploration of the system, and sometimes the system is too large for this to be feasible. For instance, a system can be represented modularly as a set of interacting components. The state space of the full system will be equal to the Cartesian product of the number of states in each module, so it can become very large very quickly. This dilemma is referred to as state-space explosion, and it is addressed by the limited lookahead approach to supervisory control, a technique that only considers strings of limited length within some lookahead window. The other problem with traditional DES supervision is that it requires the plant to be fully defined at the outset. This makes it impossible for use on systems that change over time, referred to as Dynamic Discrete Event Systems (DDESs) [18]. Time-varying systems are defined by DDES as a set of interacting modules that can appear or disappear from the system at arbitrary points in time. This is another problem that is rectified by a limited lookahead approach.

2.4 Online Control

Chung, Lafortune and Lin address the problems with controlling large or dynamic systems by introducing the limited lookahead policy (LLP) supervisor [13], followed by the more refined variable lookahead policy (VLP) supervisor [11]. We will discuss the original LLP supervisor first, but it should be noted that some terms introduced in [11] are used.

Traditional RW framework DES supervisors are computed entirely before the system begins execution, this is called offline control. Limited lookahead supervision is referred to as online control because a control action is computed on-the-fly while the system is executing, after the occurrence of each event. Online control is applicable to large systems because it only looks at a limited portion of the system at a time, and it is applicable to dynamic systems because it recalculates control after each event transition. Each control action is determined by analyzing an $N$-step lookahead tree of the system’s projected behaviour from the current state. In order to unwind an FSA into a tree, event strings must be split up over unique branches that never converge. So each state at depth $N$ in the tree corresponds to a unique string of length $N$ generated by the plant. A tree is a useful structure because it distinguishes states based on their event history; a tree makes no assumptions about the equivalence of permuted event strings. An example of an FSA being unwound into a lookahead tree is shown in Figure 2.11.

The tree is represented as a 5-tuple $T = (\Sigma, X, \xi, x_0, X_m)$, where $\Sigma$ is the set of
Figure 2.11: 3-step unwinding of FSA into lookahead tree

transitions, $X$ is the set of states, $\xi : X \times \Sigma \rightarrow X$ is the transition function, $x_0$ is the root node and $X_m$ is the set of marked states. Once the lookahead tree has been generated, the task is very similar to offline supervisor synthesis. The control action should disable events in order to direct control towards marked states and away from illegal states. How this is accomplished by analyzing the lookahead tree depends on the specific controller being used. Once the best control action is computed, it is enforced at the current state and the plant is allowed to make its next event transition. After each event, a new lookahead tree is generated for the new state and the whole process is repeated. This forms a control loop which is illustrated in Figure 2.12.

Figure 2.12: Limited lookahead supervisory control loop

This interaction between the plant and supervisor represents the central concept behind online control. Now we can delve into a more detailed description of how LLP supervision determines the best control actions.
2.4.1 Limited Lookahead Policy

Once a lookahead tree has been generated, it is up to the supervisor to decide which control action should be enforced. This is accomplished by LLP by fully traversing the lookahead tree and assigning a cost-to-go value of 0 to marked states, and infinity to illegal states. States that are neither marked nor illegal, called transient states, inherit a cost-to-go value from their subsequent states in the following manner: If they contain any outgoing uncontrollable events, the worst case is inherited from the corresponding states. If they contain only controllable events, the best case is inherited from the corresponding states. Intuitively, if a state contains an uncontrollable event that leads to an illegal state (with cost-to-go of infinity), that state inherits the cost-to-go of infinity. Conversely, if a controllable event leads to an illegal state, but there is another event that leads to a marked state, the state inherits the cost-to-go of 0 from the marked state, because the control action can disable the bad event and direct control into the marked state. The cost-to-go function can be computed for every state in the tree in this manner by propagating the values backwards to the current state of the system. The cost-to-go function \( ctg \) is defined for state \( x \) below, where \( m \) is the number of outgoing uncontrollable events, and \( n \) is the number of outgoing controllable events.

\[
ctg(x) = \begin{cases} 
0 & \text{if } x \in Q_m \\
\infty & \text{if } s \notin L(E), \text{ where } \xi(q_0, s) = x \\
\min_{\sigma \in \Sigma_{uc}} \{ctg(\xi(x, \sigma))\} & \text{if } (\exists \sigma \in \Sigma_{uc}) \ (\xi(x, \sigma)! \\
\max_{\sigma \in \Sigma_c} \{ctg(\xi(x, \sigma))\} & \text{otherwise}
\end{cases}
\]

Once the outgoing events of the current state of the system have been labeled with their corresponding values, the least-cost control action can be calculated and enforced, and the next event in the system can be allowed to fire.

Not every transient state will have a cost-to-go value defined by its subsequent states, particularly those located at the boundary of the lookahead window. This can result in pending traces of events, that is, strings that cannot be labeled safe because they may have an uncontrollable continuation to an illegal state beyond the current lookahead window. In order to deal with pending traces LLP can adopt either an optimistic or conservative attitude. The optimistic attitude assumes the best case: that these pending traces lead to a marked state, and so assigns them the value 0. The conservative attitude assumes the worst case: that these pending traces lead to an illegal state, and so assigns them the cost-to-go infinity. Intuitively
speaking, the optimistic strategy tends to move forward unless it sees some illegal region, but this may be too late to avoid some string of uncontrollable events that lead to it. With an insufficient lookahead size the optimistic strategy can produce invalid behaviours; increasing the window size reduces the amount of over-acceptance. The conservative strategy tends to avoid pending traces which can lead to the invalidation of strings that actually lead to a marked state. With an insufficient lookahead size the conservative strategy can be overly restrictive; increasing the window size increases the number of valid traces that are allowed.

Under certain conditions either strategy can produce invalid control schemes, that is, a supervisor that is less optimal than one computed in an offline fashion. A supervisor is over-accepting if it allows an event string that can uncontrollably lead to an illegal state. A supervisor is overly-restrictive if it disallows an event string that can controllably lead to a marked state. With an insufficient lookahead depth the optimistic strategy can be over-accepting and the conservative strategy can be overly-restrictive. Under the optimistic strategy, $N$ must be at least the length of the longest string of uncontrollable events that lead to an illegal state to ensure the specifications are met. Under the conservative strategy, $N$ must be at least the length of the longest string between two marked states to avoid being overly-restrictive. So, given some background information about the system to be controlled, the LLP supervisor can be as optimal as a solution computed offline. Computation of the offline solution requires a very large amount of memory, while the online solution requires far less time to compute but must be computed at every step.

**Small Factory Example**

The limited lookahead policy supervisor can be applied to the Small Factory problem from Section 2.3.1. In problems such as the small factory example, it is not important for the system to reach specific goal states, rather that the system continues to operate without errors. Typically in these cases, all states will be marked to help facilitate the forward progress of the system. In this example we will leave only state II marked to better illustrate the propagation of values throughout the trees.

Figures 2.13 and 2.14 both show lookahead trees unwound to depth 3 from state II following event string $a_1\beta_1$; so the buffer has a workpiece in it. Illegal states are indicated by an X through the state, and any subsequent states are greyed out because it follows that they are also illegal. Figure 2.13 shows how the values propagate under the optimistic strategy and Figure 2.14 illustrates the conservative strategy. The cost-to-go values of each tree are calculated by first assigning values to the leaf nodes based
on the employed strategy, the legality of states and their markings. These values are displayed to the right of the leaf nodes. Then the cost-to-go values for either strategy can be propagated over the entire tree using the $ctg$ function from Section 2.4.1. These values are displayed beneath their corresponding states.

![Limited look-ahead Tree with $N = 3$ computed with optimistic strategy for the Small Factory problem following string $\alpha_1\beta_1$.](image)

There are three situations that need to be avoided to enforce the specifications. The repair specification can be enforced in one step by disabling $\mu_1$ when both machines are down. Buffer underflow can be avoided in one step by disabling $\alpha_2$ when the buffer is empty. Ensuring there is no buffer overflow requires a two step lookahead because the problem occurs when Machine 1 finishes a workpiece uncontrollably. This must be avoided by disallowing Machine 1 to start a workpiece when there is already one in the buffer.

The supervisors shown in Figures 2.13 and 2.14 both produce the same control action after the occurrence of $\alpha_1\beta_1$; disable $\alpha_1$ and enable $\alpha_2$. This control action is not *over-accepting* or *overly-restrictive* because it enforces the specifications correctly and does not disallow any valid strings. With a lookahead limit of 1, the optimistic strategy will assume that a marked state can be controllably reached after string
Figure 2.14: Limited lookahead Tree with $N = 3$ computed with conservative strategy for the Small Factory problem following string $\alpha_1\beta_1$. 
\( \alpha_1 \beta_1 \alpha_1 \), so it would enable both events and be over-accepting here. This can be avoided by ensuring that the lookahead depth is at least 2, which is given by the conditions for validity in [13]; in the Small Factory example the length of the longest string of uncontrollable events that lead to an illegal state is 2.

Once the control action has been calculated, it is enforced and the system is allowed to make the next transition. Only \( \alpha_2 \) is enabled so the system makes this transition. Now the system is in a new state so a new control action must be calculated by generating another lookahead tree. Figures 2.15 and 2.16 show lookahead trees unwound from this new state to depth 3 using the optimistic and conservative strategies, respectively.

Figure 2.15 shows that the optimistic strategy produces an optimal control action here. However, the conservative strategy shown in Figure 2.16 is sub-optimal because it disables \( \alpha_1 \). This is an example of how the conservative strategy can be overly-restrictive. The conditions for validity given in [13] state that the conservative strategy will not be overly-restrictive if the lookahead depth is equal to or greater than the length of the longest string between two marked states. In the Small Factory example there exist loops of events such that a marked state is never again reached so it is impossible to ensure that the conservative strategy will not be overly-restrictive. For instance, the set of strings \( \alpha_1 \beta_1 \alpha_2 (\alpha_1 \gamma_1 \mu_1)^* \) transition through a marked state after two events and then never reach another one. Even with an arbitrarily large lookahead depth, these strings can always continue past the lookahead boundary and will hence be marked with a cost-to-go of infinity, which can result in an overly-restrictive control action.

As these examples show, there are cases in which either the optimistic or conservative strategy will be more optimal at equal depths; it depends on the characteristics of the problem. A favourable strategy and depth can be determined by analyzing the problem beforehand with respect to the conditions for validity in [13] and the required speed of online computations determined by the state space of the lookahead trees. The obvious choice when approaching the Small Factory problem is to employ an optimistic strategy with a lookahead depth of 2. Continued work on the limited lookahead framework has eliminated the need for a control strategy to be selected prior to online execution, which is discussed in the following section.

### 2.4.2 Variable Lookahead Policy

The Variable Lookahead Policy (VLP) supervisor is very similar to LLP, but with a few important improvements. Instead of adopting a control attitude and treating
Figure 2.15: Limited lookahead Tree with $N = 3$ computed with optimistic strategy for the Small Factory problem following string $\alpha_1 \beta_1 \alpha_2$. 
Figure 2.16: Limited lookahead Tree with \( N = 3 \) computed with optimistic strategy for the Small Factory problem following string \( \alpha_1 \beta_1 \alpha_2 \).
pending traces as either marked or illegal, VLP assigns pending traces a cost-to-go of \( U \) for undecided. The value propagation algorithm is essentially the same: uncontrol-
rollable events pass the worst-case value, while states with only controllable events pass
the best-case value, where \( U \) is evaluated to be between 0 and infinity. If any \( U \) values
are propagated back to the current node, then one of two actions must be taken. If
it results in an unambiguous control decision then \( U \) can be treated as a value that is
greater than 0 and less than \( \infty \), although this approach can yield a control decision
that is overly-restrictive. Alternatively, either an optimistic or conservative strategy
must be adopted, in which case \( U \) is replaced with 0 or infinity respectively. This
defines the LLP control attitudes as special cases of VLP and so the same bounds on
\( N \) to ensure validity are applicable here.

Implementation of the \( U \) value allows for two optimizations on the algorithm to be
applied. The first being that the entire lookahead tree need not always be traversed
for a control decision to be made. Progressing outwards in a breadth-first manner,
values are propagated back to the current state, and once none of the outgoing events
from the current state are labeled with \( U \) a control decision can be made without
any further computations. The second optimization is that it becomes much easier
to reuse computations from previous iterations of the algorithm. Because no values
of 0 or infinity are incorrectly assigned to pending traces, it is always safe to reuse
these values in subsequent iterations. In this manner, only the states with a value of
\( U \) and the states at level \( N \) in the lookahead tree need to be computed at each step.

There have been a few extensions to the VLP supervisor. Hadj-Alouane, Lafortune
and Lin [19] developed a framework called VLP with state information, abbreviated
as VLP-S. The state information is essentially a statement about the safety of the
state, which is computed by recursively unwinding the plant starting at the state in
question. Expansion continues until a state that has already been expanded, or a
state with a known cost is encountered. Known cost states include marked states,
illegal states, and states which are known to be either safe or unsafe from previous
expansions. This state expansion typically has to proceed further than the lookahead
window allows, so the majority of their findings are concerned with using VLP-S to
more efficiently compute solutions offline, which is accomplished by expanding every
state in the plant. The complexity of VLP-S is shown to have the same worst case
complexity as standard offline supervision, and is potentially more efficient otherwise.
Of course, VLP-S can be used online as well, and the addition of a seen state list
is very beneficial for dealing with certain problems. For instance, if two states have
uncontrollable events leading to each other, they will generate an infinitely long string
of uncontrollable events in the lookahead tree that will always extend beyond the
lookahead window. If these states are safe a conservative attitude will be overly restrictive, and if they are unsafe an optimistic attitude will be invalid. When VLP-S extends one of these states, it will only extend the uncontrollable string two steps until it is back where it started. It will then extend along the other outgoing events in a depth-first manner until it can unambiguously assign the states an actual safety cost, and generate a control scheme from this.

**Small Factory Example**

Now we will apply the standard VLP algorithm to the Small Factory problem from Section 2.3.1. Figure 2.17 shows a lookahead tree unwound to depth 3 from state II following event string $\alpha_1, \beta_1$; so the buffer has a workpiece in it. This is the same tree as depicted in Figures 2.13 and 2.14. Using VLP, the values are calculated for the tree in much the same way as LLP, but in this case pending traces are assigned a cost-to-go of U instead of assigning a 0 or $\infty$. The optimistic LLP supervisor (Figure 2.13) can be obtained by simply replacing all of the U values with 0 values, and the conservative LLP supervisor (Figure 2.14) can be obtained by replacing them with $\infty$ values. Selecting a supervision strategy is not necessary in this case because no undecided values are propagated to the decision node. This VLP supervisor makes the same control decision as LLP and disables $\alpha_1$ forcing the system into state IW.

Figure 2.18 shows a lookahead tree unwound to depth 3 from state IW following event string $\alpha_1\beta_1\alpha_2$. This tree will help to illustrate optimizations of VLP that make it favourable over LLP. First, because no decided values have been “incorrectly” assigned through the application of a strategy, any 0 or $\infty$ values from the tree in Figure 2.17 can be reused. This includes states resulting from strings $\alpha_1\beta_1\alpha_2\beta_2$, $\alpha_1\beta_1\alpha_2\beta_2\beta_2$, $\alpha_1\beta_1\alpha_2\gamma_2$ and $\alpha_1\beta_1\alpha_2\gamma_2\beta_2\mu_2$, which is a total of 4 reusable states. Second, not all of the tree must be traversed in order to obtain a control decision; only branches following states with U values need to be calculated. This means that only the branches following string $\alpha_1\beta_1\alpha_2\alpha_1$ need to be explored before making a control decision because the other branches from the root node are already labelled with 0 values. After exploring these states the branch is still labelled with a U value, so a strategy should be selected in order to make a control decision. The conditions for validity of LLP control strategies from [13] are still applicable here, so we know that an optimistic strategy is the better choice, which results in $\alpha_1$ being enabled.

For this tree with 34 states, 4 states have values that can be reused, and 15 need not be calculated because they occur after a state with a decided value. This leaves a total of 17 states out of 34 that need to be explored, so VLP with its optimizations
accomplishes as much as LLP while only performing half as many calculations. It can be seen how even more computational savings are possible under certain circumstances, and that VLP will make decisions as optimal as LLP, so VLP represents a strict improvement over LLP.

Figure 2.17: Limited lookahead Tree with $N = 3$ computed with VLP algorithm for the Small Factory problem following string $\alpha_1 \beta_1$.

2.5 String Desirability

Traditionally strings that terminate in a marked state are considered desirable, and strings that could not be generated by the legal language $E$ are considered undesirable. Not all problem situations are such that outcomes can be defined as either good or bad and nowhere in between. Recent work has aimed to extend the concept of string desirability to be more continuous.

Sengupta and Lafortune [33] introduce a different approach to defining the desirability of strings called cost, which is a performance measure that corresponds with undesirability. The event cost function $C_e$ associates values with events, while the control cost function $C_c$ associates values with the disablement of events by the
Figure 2.18: Limited lookahead Tree with $N = 3$ computed with VLP algorithm for the Small Factory problem following string $\alpha_1\beta_1\alpha_2$. 
supervisor:
\[ C_e : \Sigma \rightarrow \mathbb{R}^+ \cup \{0\} \quad \text{and} \quad C_c : \Sigma \rightarrow \mathbb{R}^+ \cup \{0, \infty\}, \]
where \( \mathbb{R}^+ \) is the set of positive real numbers. Uncontrollable events are associated with an infinite disablement cost, i.e., if \( \sigma \in \Sigma_{uc} \) then \( C_c(\sigma) = \infty \). Using these functions the cost of any string of events generated by the controlled system can be determined, and the supervisor is tasked with minimizing this cost.

Similar work has been conducted by Chen and Lin [10]. They define two functions, \( \text{cost} \) and \( \text{effect} \), which associate each event in the system with two corresponding real values:
\[ C : \Sigma \times Q \rightarrow \mathbb{R} \quad \text{and} \quad E : \Sigma \times Q \rightarrow \mathbb{R} \]
where \( \mathbb{R} \) is the set of real numbers. The cost of event \( \sigma \) in state \( q \), \( C(\sigma, q) \), describes the level of undesirability of the event via the amount of resources it consumes. The effect of \( \sigma \) in \( q \), \( E(\sigma, q) \), describes the level of desirability of the event via the amount of good it accomplishes. Because it is not possible to optimize on two values at once, Chen and Lin define a controller that disables events in order to minimize cost while maintaining a pre-specified minimum level of effectiveness. This is accomplished by calculating the cumulative cost and effect for every state in the lookahead tree, and trimming off states from which the desired effectiveness cannot be attained. The resulting tree is referred to as the effective tree, and it is then used to compute the control actions that minimize cost. Alternatively, a maximum cost value can be specified and used to trim the tree, and then control decisions can be made to maximize effect.

2.6 Probabilistic DES

Traditionally the events of a DES define all the possible behaviours of the system, and control actions are enforced to shape these possible behaviours. For some problems, it is more beneficial for the solution to be concerned with what is likely to happen, instead of what is able to happen. Of course, for this set of problems the system must be defined in terms of what is likely to happen, so probabilities of event occurrences must be known or estimated. These systems are referred to as probabilistic discrete event systems (PDES) by Lawford and Wonham, who introduce the concept of probabilistic events [25]. In this framework each event is associated with a probability of occurrence and the plant is referred to as a probabilistic generator \( G = (Q, \Sigma, \delta, q_0, Q_m, p) \), where \( p(q, \sigma) \) is the probability of event \( \sigma \) in state \( q \). Each probabilistic generator corresponds to a probabilistic language, where \( L_p(G) \) is the
probabilistic language generated by $G$. The probability that string $s$ is generated by $G$ is denoted by $L_p(G)(s)$. With traditional languages, $L_p(G)(s)$ would be equal to 1 if $s \in L(G)$, and 0 otherwise. With probabilistic languages the values can be anywhere in between, i.e., $L_p(G)(s) \in [0, 1]$, where $L_p(G)(s) > 0$ iff $s \in L(G)$. In the probabilistic framework the probabilities of events exiting from state $q$ should add up to 1, otherwise the system is said to terminate in $q$ with the leftover probability:

$$1 - \sum_{\sigma \in \Sigma} p(q, \sigma) = \text{probability of termination in state } q$$

Lawford and Wonham show how a probabilistic supervisor $S_p$ can influence the likelihood of occurrence of events in $G$ by probabilistically disabling events, i.e., for each string $s$ and event $\sigma$, $S_p(s, \sigma)$ yields the probability that $S_p$ enables $\sigma$ after $s$ has occurred. Under certain conditions a probabilistic supervisor $S_p$ can be computed from a probabilistic generator $G$ and a legal language $E$, which defines the desired probabilities of events. This concept of a probabilistic supervisor is not applicable in most of the problem situations discussed in the introduction, where desirable decisions need to be made at every step along the way. A probabilistic supervisor will favourably influence event probabilities in the long term, but decisions made in the short term are subject to chance.

Similar work has been conducted by Kumar and Garg [24], but they are more concerned with the language-theoretic side of things. They define the conditions under which a supervisor exists that ensures illegal strings have an occurrence probability of zero, and desirable strings occur with some pre-specified minimum probability.
Chapter 3

Recursive Utility-Based Limited Lookahead

Much of the work discussed in the previous chapter was aimed at extending the DES framework to a wider range of problems. Supervisory algorithms have been designed to incorporate probability of events, to handle a more diverse definition for the benefit of occurrences, and to tackle extremely large problems. This chapter introduces an algorithm which combines and builds on these improvements, referred to as the Recursive Utility-Based Limited Lookahead (RUBLL) algorithm.

The RUBLL algorithm uses the concept of utility to make control decisions. The utility of an event is essentially equal to the probability that it occurs times the desirability of the resulting state, so it is synonymous with expected value. The utility of a state is equal to the probability-weighted sum of the values of its subsequent states. For instance, winning a dollar on a coin flip has a utility of fifty cents. The RUBLL algorithm proceeds very similarly to VLP, but instead of propagating values corresponding to the safety of states it propagates utility values. Utility is calculated for every state in the lookahead tree starting at the leaf nodes and working backwards. As in VLP, each state propagates the highest value that the supervisor can attain by controlling the events. The supervisor exercises its control by selecting from a set of event probabilities defined by the system. For instance, if there are two control actions for some state, the supervisor will need to decide between the two corresponding sets of probabilities for all events exiting that state. This control framework is consistent with having a few distinct control options that influence the probability of events, which is more compatible with real-world probabilistic control than Lawford and Wonham’s concept of a probabilistic supervisor [25]. The utility for each set of probabilities is calculated by taking the probability-weighted sum of the resulting state’s utilities, and incorporating the desirability associated with the events themselves. The RUBLL framework uses a desirability function similar to Sengupta and Lafortune’s [33], in that each event and control decision is associated with a benefit value. The benefit of an event or control action defines the amount of good that is accomplished, and cost is defined as negative benefit to simplify the optimization problem. The RUBLL framework is designed so that it is applicable to a wide variety of real-world problems, but it is also defined in such a way so that the problems covered by previous controllers are handled as special cases.
CHAPTER 3. RECURSIVE UTILITY-BASED LIMITED LOOKAHEAD

3.1 RUBLL Framework

The RUBLL framework extends the definition of a plant to incorporate probability and benefit functions: \( G = (\Sigma, Q, \delta, q_0, P, B_c, B_e) \). Note that the concept of marked states has been replaced by the benefit functions, and the controllability of events is defined by the probability function. As in previous lookahead algorithms the plant is unwound into an \( N \)-step lookahead tree \( T \) which is used to calculate the best control decision at the current state. The tree is represented as a 7-tuple \( T = (\Sigma, X, \xi, x_0, P, B_c, B_e) \), where \( \Sigma \) is the set of events, \( X \) is the set of states, \( \xi : X \times \Sigma \to X \) is the transition function, \( x_0 \) is the root node, \( P \) is the probability function and \( B_c \) and \( B_e \) are the benefit functions. The probability function, benefit functions, and event set are unaffected by the unwinding of \( G \) into \( T \); but the set of states, the root node and the transition function will be redefined.

The event benefit function \( B_e \) associates each event with a benefit value. The control benefit function \( B_c \) associates each state with a set of benefit values, each corresponding to a specific control strategy. The probability function \( P \) associates each event in each state with a set of probability values, each corresponding to a specific control strategy. So if there are \( m \) possible control choices, \( P \) and \( B_c \) will each return an \( m \)-tuple of values.

\[
P : X \times \Sigma \to [0,1] \times [0,1] \times \cdots \times [0,1]
\]

\[
B_c : X \to \mathbb{R} \times \mathbb{R} \times \cdots \times \mathbb{R}
\]

\[
B_e : \Sigma \to \mathbb{R}
\]

The set of \( m \) control patterns is defined as \( \Gamma = \{ \gamma_1, \ldots, \gamma_m \} \). Each control pattern \( \gamma_i : X \times \Sigma \to [0,1] \times \mathbb{R} \) maps the possible probability and benefit values of an event to specific values associated with the specific control pattern. In other words it maps a tuple of probabilities to a single probability, and a tuple of benefit values to a single benefit value. We use \( P^i(x, \sigma) \) to denote the probability of event \( \sigma \) occurring at state \( x \) under control pattern \( \gamma_i \), and \( B_c^i(x) \) to denote the control benefit at state \( x \) under control pattern \( \gamma_i \). For all \( \sigma \in \Sigma \), for all \( x \in X \), if \( P(x, \sigma) = (p_1, \ldots, p_m) \) and \( B_c(x) = (b_1, \ldots, b_m) \) then \( \gamma_i(x, \sigma) = (p_i, b_i) = (P^i(x, \sigma), B^i_c(x)) \).

3.2 RUBLL Pseudocode

The RUBLL algorithm operates by recursively calculating the utility of each state based on the utility of its subsequent states. It continues until the utilities for each
control pattern are known for the root node, which is also called the decision node. At this point the best decision can be made and enforced, and the plant is allowed to execute its next event.

**RUBLL Pseudocode**

**MAIN**:
1. Construct lookahead tree \( T \) to depth \( N \)
2. Compute \( \text{bestU}(x, depth) \) for current state \( x \) (as detailed in Algorithm 3.2.1)
3. Enforce \( \gamma_i \) with \( U^i = \text{bestU}(x, depth) \)
4. Allow system to execute next step probabilistically and return to 1

**Algorithm 3.2.1: bestU(x, depth)**

```
if depth = 0
    then return (0)
for each \( \gamma_i \in \Gamma \)
do \( U^i = B^i_c(x) + \sum_{\sigma} (P^i(x, \sigma) \cdot (\text{bestU}(\xi(x, \sigma), depth - 1) + B_e(\sigma))) \)
return (maximum \( U^i \) for all \( i \))
```

The MAIN function is in charge of unwinding the system into a lookahead tree and determining the optimal control action using the bestU function. The bestU function calculates the current state’s utility for each control pattern and returns the maximum attainable utility. The bestU function will continue to call itself recursively until it reaches the lookahead depth \( N \), at which point the value 0 is returned as the utility. These values are propagated back along the tree until the bestU value for the decision node is obtained. The recursive approach of the bestU function of breaking the problem down into smaller subproblems is an example of dynamic programming [4]. When the bestU value for the decision node is known, the MAIN function enforces the strategy that produces this maximum attainable utility, and allows the system to execute the next event. This pseudocode represents the most straightforward interpretation of the RUBLL algorithm because it shows how utility is propagated by a single equation. However, implementation of this code as is will result in exponentially more redundant calculations as the depth of the tree increases. For pseudocode that avoids these redundant calculations when implemented the reader is directed to Section 3.5.
3.3 MACHINE Example

Consider the example problem shown in Figure 3.1 to which the RUBLL algorithm will be applied. The problem is represented by an FSA with additional information attached to its states and events.

![Figure 3.1: FSA representation of MACHINE example.](image)

This problem is one in which an imaginary machine is tasked with processing a series of parts. The machine takes a part from an input bin and starts to work on it. From this WORKING state one of three things can happen: the machine finishes the part, breaks down, or has to do some additional work on the part. Finishing a part returns the machine to the IDLE state and breaking down places the machine in the BROKEN state in which repair attempts can be made to try to return to the IDLE state. If any additional work on the part is required it places the machine in the STILL WORKING state in which the machine can finish the part or break down. Note that the machine is more likely to break down when dealing with 'problem' parts that require more work. Each event in the system has a probability and a benefit associated with them, found in the round and square brackets, respectively. Multiple probabilities indicate that a control decision can be made, where any of the probabilities can be applied but must be done so with their corresponding benefit values. For instance, from the WORKING state there is a 3% chance that the machine will break down and enter the BROKEN state. There is no immediate benefit or cost associated with this event, indicated by the absence of any square brackets. Alternatively, there is a 10% chance (indicated by 0.1, the first probability associated with event `moreWork`) that the machine will enter the STILL WORKING state, unless...
the decision to preprocess the part has been made. In the case of preprocessing
the chance of the moreWork event is dropped down to 1% (indicated by 0.01, the
second probability associated with event moreWork), but there is a cost of 10 (which
translates to a benefit of -10) associated with part preprocessing. Some of the events
have an undetermined probability, denoted by (*). This means that the probability
is determined by which control decisions are enforced for the other events exiting the
current state. Since the probability of events leaving a state must add up to 1 in
a non-terminating system, undetermined probabilities absorb any remaining chance
to ensure that this is always the case. For instance, at the WORKING state the
probability of finishing a preprocessed part is 96%, while the chance of finishing a
part that is not preprocessed is 87%. In the BROKEN state one of two handymen
can be hired to fix the machine. Technician A has a 40% chance of succeeding and
each attempt has a cost of 60, while technician B has 70% chance to succeed but
costs 120. Finishing a part has a benefit of 100. The problem presented here has
two control decisions that can be made: the decision to preprocess a part or not,
and the decision to hire technician A or B when the machine breaks down. Each of
these decisions have pros and cons associated with them and it is not readily obvious
which decisions will result in the greatest overall benefit. The decision to preprocess
a part has an immediate cost of 10, but this may be offset by the benefits of avoiding
the STILL WORKING state more often, in which the machine is using more time to
process the part and is more likely to break down. In the BROKEN state the decision
to hire technician A or B must be made. Hiring A has a lower immediate cost of 60,
but results in the machine being more likely to stay broken for longer. Hiring B has
a higher immediate cost of 120, but increases the chance of successfully repairing the
machine. It may be beneficial to spend more now to reduce the amount of time spent
in the BROKEN state because more parts will be finished the less time the machine
is broken. It is therefore necessary to apply an analysis technique to determine the
most beneficial control strategy. RUBLL is one such analysis technique that will
perform this task; another is the Markov chain approach. Both of these methods will
be applied to solve the problem.

3.3.1 RUBLL Approach

The first step of applying the RUBLL algorithm is to formalize the problem into
the framework discussed in Section 3.1. If you recall, for all \( \sigma \in \Sigma \) and for all
\( x \in X \), if \( P(x, \sigma) = (p_1, \ldots, p_m) \) and \( B_c(x) = (b_1, \ldots, b_m) \) then \( \gamma_i(x, \sigma) = (p_i; b_i) = (P_i^i(x, \sigma), B_{c_i}(x)) \). For example,
CHAPTER 3. RECURSIVE UTILITY-BASED LIMITED LOOKAHEAD

\[ B_e(\text{finishPart}) = 100 \]
\[ B_c(\text{BROKEN}) = (-60, -120, -60, -120) \]
\[ P(\text{WORKING}, \text{finishPart}) = (0.87, 0.87, 0.96, 0.96) \]
\[ P(\text{BROKEN}, \text{fixSuccess}) = (0.4, 0.7, 0.4, 0.7) \]
\[ \gamma_1(\text{WORKING}, \text{finishPart}) = (0.87, 100) \]
\[ P^1(\text{WORKING}, \text{finishPart}) = 0.87 \]
\[ \gamma_2(\text{BROKEN}, \text{fixSuccess}) = (0.7, -120) \]
\[ B^2_c(\text{BROKEN}) = -120 \]

where
\[ \gamma_1 = \text{No Preprocessing, Technician A} \]
\[ \gamma_2 = \text{No Preprocessing, Technician B} \]
\[ \gamma_3 = \text{With Preprocessing, Technician A} \]
\[ \gamma_4 = \text{With Preprocessing, Technician B} \]

Unlike previous lookahead algorithms, RUBLL determines the best control decision by calculating the utility of all of the states in the tree using Algorithm 3.2.1. At this point the control decision which maximizes the decision node utility can be determined and enforced. Figure 3.2 shows the utility values calculated in this manner on a tree of depth 5 for a machine in the broken state.

Utility values are shown in curly brackets below their corresponding states. Benefit values are shown in square brackets above their corresponding state or event. States at depth N are assigned a utility of zero. The utilities of states at depth 1 are similarly used to calculate the optimal decision to be made. Using Algorithm 3.2.1 we get

\[ U^1 = -60 + 0.4(168.2) + 0.6(-72) = -35.92 \]
\[ U^2 = -120 + 0.7(168.2) + 0.3(-72) = -23.86 \]

Control pattern \( \gamma_2 \) produces a larger utility, so it is a more beneficial decision to hire technician B. It is also clear from this example that a depth of at least 5 is required to make the decision to hire technician B because the BROKEN state at depth 1 does not have sufficient depth to make the optimal decision.

3.3.2 Markov Chain Approach

Using the Markov chain approach it is possible to calculate the utility or expected benefit value resulting from the steady state distribution for each of the four global control strategies. In order to do this, the FSA must be translated into a transition matrix representation. One transition matrix must be generated for each of the control
Figure 3.2: 5-step lookahead tree for MACHINE in BROKEN state.
CHAPTER 3. RECURSIVE UTILITY-BASED LIMITED LOOKAHEAD

patterns which are shown in Figure 3.3. In order to calculate the expected benefit for a given control strategy an eigenvector must be extracted from the transition matrix. The values of this eigenvector $v$ represent the proportion of time that the system spends in each corresponding state.

$$ P_\infty(x) = \frac{v_x}{(v_1 + \ldots + v_{|X|})} $$

Normalized eigenvectors for each matrix are shown in Figure 3.3. The utility corresponding to each state can be calculated by multiplying the probability of an outgoing event by the benefit associated with it, and adding these values for all outgoing events. The utility of state $x$ is denoted by $U_x$:

$$ U_x = \sum_\sigma P_\sigma \cdot B_\sigma, \text{ for each } \sigma \text{ exiting state } x, $$

where $P_\sigma$ and $B_\sigma$ represent the probability and benefit values, respectively, associated with event $\sigma$. State utilities are shown in Figure 3.3. Knowing the utility of each state, and the probability of being in each state based on the steady state distribution, it is now possible to calculate the global steady state utility $U_\infty$

$$ U_\infty = \sum_x U_x \cdot P_\infty(x), \text{ for all states } x \text{ in the system}. $$

Global utilities are shown in Figure 3.3. The Markov chain approach determines that the best strategy is to not preprocess the parts and to hire technician B. These control
decisions must be applied uniformly throughout execution of the system, whereas with a limited lookahead approach different control strategies can be applied at different points during execution.

The Markov chain approach will always find the optimal steady state solution, whereas RUBLL will not necessarily be optimal depending on the lookahead depth. The upside to RUBLL is that different control decisions can be applied at different times. While these approaches differ in this respect, we speculate that they are mathematically consistent with each other by the following: If state utilities are initialized as the expected state benefits found in Figure 3.3 and updated using the four equations derived from the lookahead tree, after sufficient iterations the utility of every state will approach the global utility values in Figure 3.3. The number of iterations required differs with the amount of oscillation in the system. For instance, with control pattern $\gamma_1$ every state’s utility is equal to the Markov global utility up to four decimal places after 114 iterations. With $\gamma_3$ the most oscillation is observed and 582 iterations are required until the numbers converge.

3.4 B-RUBLL

The benefit of using RUBLL over Markov chains is that different control decisions can be made at different points during the execution. The Markov chain solution requires that the control strategy be applied globally over the system, so the same control decision must be made at every point during the execution. A global control solution is suitable for the MACHINE example because it is a relatively simple one. In light of this, a simplification of the RUBLL algorithm in which controls are applied uniformly over the entire lookahead tree can be used. This version of the algorithm, in which each control pattern is applied like a blanket over the entire tree is referred to as the Blanket-RUBLL algorithm, or B-RUBLL, which is shown below:

**B-RUBLL Pseudocode**

**MAIN:**
1. Construct lookahead tree $T$ to depth $N$
2. Initialize global utility array $U^T[i]$
3. Compute $\text{BLANKETU}(x, depth, \gamma_i, 0, 1)$ for current state $x$
4. Enforce $\gamma_i$ with greatest $U^T[i]$
5. Allow system to execute next step probabilistically and return to 1
CHAPTER 3. RECURSIVE UTILITY-BASED LIMITED LOOKAHEAD

Algorithm 3.4.1: `BLANKETU(x, depth, \gamma_i, B, P)`

```latex
\textbf{Algorithm 3.4.1:} \textsc{blanketu}(x, depth, \gamma_i, B, P)
\begin{algorithmic}
  \If {depth = 0}
    \Then \(U^T[i] = U^T[i] + (B \cdot P)\)
  \EndIf
  \State \(B = B + B_c(x)\)
  \For {\textbf{each} }\(\sigma \in \Sigma\) \textbf{with} \(\xi(x, \sigma)\)!
    \Do {\textsc{blanketu}(\xi(x, \sigma), depth - 1, \gamma_i, B + B_c(\sigma), P \cdot P^i(x, \sigma))}
  \EndFor
\end{algorithmic}
```

This algorithm calculates the utility for each global control pattern by traversing the branches of the tree and keeping track of the cumulative benefit and cumulative probability of each path as it goes. When it reaches the depth of the lookahead tree, it multiplies the cumulative benefit with the cumulative probability for the current branch and adds this value to the global utility value. Once every branch of the tree has been traversed, the global utility value will equal the utility associated with the current control pattern.

The RUBLL algorithm must compute the best utility for each state based on the best attainable utilities of its subsequent states. This means that the utilities must be calculated in a backwards manner from the states at the horizon of the lookahead tree back to the decision node. The B-RUBLL algorithm applies the same control decision at every state which means that it can calculate the utility of a tree in a forwards manner, updating a global utility variable as it goes. This means that B-RUBLL can be efficiently implemented in its natural form, while RUBLL must be modified in order to be implemented so that no redundant calculations are performed. This implementable form of RUBLL is discussed in Section 3.5, along with complexity considerations for the three algorithms.

3.5 Implementation

The RUBLL algorithm presented in its simplest form (Algorithm 3.2.1) performs a lot of redundant calculations when it is implemented directly. In this section we will discuss these redundancies and how they can be avoided when implementing the algorithm.

The problem with implementing this code is that the recursive calls on successive states are made for each of the control choices, even though the control decision for the current state has no effect on the utilities of subsequent states. Therefore, if \(C\) is the number of control choices at each state, then every calculation must be performed \(C\) extra times for each additional depth in the lookahead tree. The computational
complexity of this algorithm can be calculated in terms of $C$ and the growth rate of the tree, $G$. The growth rate of a tree is equal to the average number of events exiting each state weighted by the proportion of each state’s appearance in the tree. Note that if $G$ is the weighted average of the number of events per state then the state space of the tree will increase by a factor of $G$ with each additional depth. In most cases, $G$ is equal to the dominant eigenvalue of the adjacency matrix of the underlying FSA. For a more detailed discussion on when this is the case, the reader is directed to Chapter 4.

Figure 3.4 shows a simple tree with $G = 2$ unwound to depths 2, 3 and 4. When applied to (a), RUBLL 3.2.1 will make $C$ recursive calls of the bestU function to each of the $G$ subsequent states. This yields a complexity of $C \cdot G$. When applied to (b), $C$ recursive calls are made to each of the $G$ trees of depth 1, which yields a complexity of $C \cdot G(C \cdot G)$. When applied to (c), $C$ recursive calls are made to each of the $G$ trees of depth 2, which yields $C^3 \cdot G^3$. Obviously the complexity of using the simplest implementation of the RUBLL algorithm is $O(C^n G^n)$. The factor $C^n$ represents all of the redundant calculations performed by this algorithm, and can successfully be removed by implementing RUBLL more neatly. Below is shown the version of the bestU function that is best for implementation. Note that the recursive call is made only once for each outgoing event, and the result is stored before it is used to determine the most favourable control decision for the current state.
Algorithm 3.5.1: \textsc{bestU}(x, depth)

\begin{algorithm}
\begin{algorithmic}
\If{depth = 0}
\State \Return (0)
\EndIf
\State Initialize array \texttt{bestUtility}[|\sigma|]
\For{} \texttt{\sigma} \in \Sigma with \xi(x, \sigma)!
\State \texttt{bestUtility}[\sigma] = \textsc{bestU}(\xi(x, \sigma), depth - 1)
\EndFor
\For{} \texttt{control pattern} \gamma_i \in \Gamma
\State \left\{ \begin{array}{l}
U^i = B^c_i(x) \\
\text{for each} \ \sigma \in \Sigma with \xi(x, \sigma)!
\Endarray \right.
\Do{} \left\{ \begin{array}{l}
U^i = U^i + P^c_i(x, \sigma) \cdot (\texttt{bestUtility}[\sigma] + B_e(\sigma)) \\
\text{if} \ U^i > U_{\text{max}} \then \ U_{\text{max}} = U^i
\Endarray \right.
\EndDo
\Return (U_{\text{max}})
\end{algorithmic}
\end{algorithm}

This implementation of RUBLL will actually run faster than B-RUBLL in many cases. The B-RUBLL algorithm traverses the full state space of the tree once for each control pattern. In terms of recursive calls, B-RUBLL has a complexity of $O(S^n C)$, where $S^n$ is defined as the state space of the lookahead tree unwound to depth $n$. In most cases (see Chapter 4) $G^n$ is the dominant factor in the definition of $S^n$, so these terms can be considered equivalent in complexity.

The complexity of RUBLL 3.5.1 can be similarly calculated. The entire tree must be traversed only once, but up to $C$ utility calculations must be performed at each state. This results in a worst-case complexity of $O(S^n C)$. In the general case there will be fewer than $C$ calculations at most states because not every global control pattern will have a unique control decision for every state in the system. For instance, look at the MACHINE example; there are 4 global control patterns, but only a maximum of 2 control choices in any single state.

Despite the fact that RUBLL can be implemented to run faster in many cases, it still makes sense to use B-RUBLL when a blanket control strategy is fitting for the problem at hand because B-RUBLL is easier to implement and debug, requires less memory, and is not substantially slower.

Analysis of complexity and many other considerations when using an online control policy are based on the state space of the lookahead trees. Therefore it is instrumental to investigate the nature of the growth of lookahead trees. For instance, with actual values for $S^n$ the complexity of online algorithms can be estimated much
more accurately, and the different algorithms could be compared across different approaches using different trees. This necessitates a method for easily calculating the size of lookahead trees, and this is the subject of the next chapter.
Chapter 4

State-Space Analysis

This chapter introduces and discusses a technique for estimating the state space of a lookahead tree. The technique presented here is intended as a tool to help address state-space explosion. Recall that the Ramadge and Wonham supervisory control algorithm requires a traversal of the full state space of the problem. The exhaustive search through the system is not possible when the state space is subject to change partway through the execution, or if the state space is too large to feasibly search. It is easy for a state space to become very large when dealing with systems made up of many different interacting modules because as two modules interact their behaviour is defined by an FSA with state space equal to the Cartesian product of the number of states in each module. As more modules are introduced, the state space of the overall system may increase exponentially; this is a problem referred to as state-space explosion. For example, ten modules with ten states each would combine to produce a system with ten billion states. In these cases a limited lookahead approach (as introduced by Chung, Lafortune and Lin [13]) can be employed, which means that the supervisory algorithm will be executing on lookahead trees. Tree state-space is dependent on only the state-transition structure of the FSA, so we can disregard initial and terminal states from our discussion and diagrams and focus on the directed graph induced by the states and transitions of the FSA. When it is clear from context, we may use the term “FSA” to refer to the directed graph associated with the FSA.

Increasing the depth of the tree used will increase the amount of information available to make decisions which, in turn, generally results in more optimal control (however, there exist counterexamples, as discussed in [18]). Increasing the lookahead depth used will also increase the space required to store the tree and the time required to compute decisions. The computational time and space requirements are dependent on the implementation used, but both are functions of the state space of the tree. This chapter presents a method by which the state space of a lookahead tree can be accurately estimated by analyzing the adjacency matrix of the underlying directed graph.

Recent work also applies DES theory to solve concurrency control problems [1]. There, a limited lookahead algorithm is used to explore the many possible interleavings of program threads and determine the controls required to meet the specifications,
such as deadlock avoidance. An estimate for the number of states in the lookahead window would allow the algorithms in [1] to be adjusted to maximally utilize computer resources. In addition, our method for estimating the size of state space may be useful to the model checking community. In model checking, a depth-first search of state space is performed in order to verify program properties; it is common to limit the space-state traversal to a certain depth [22, 20].

The remainder of this chapter is organized as follows: Section 4.1 is devoted to various algebraic definitions of the state space of lookahead trees. A parameter, called $\tau$, which will help to characterize the size of the state space, is introduced and a formula for the steady state $\tau$ value of a graph is derived. Section 4.1.4 is a simple example of the $\tau$-estimate in application. Section 4.1.4 illustrates how the $\tau$-estimate can be used to assist a limited lookahead approach by applying it to the MACHINE example. Section 4.2 covers the general case for calculating a $\tau$-estimate, and how to calculate $\tau$ for special cases. Section 4.3 presents a step-by-step guide to estimating tree state-space while considering the computational complexity of the various approaches. Section 4.4 presents an algorithm which performs the $\tau$ state-space estimation, and represents a culmination of the methods discussed in this chapter. Section 4.5 discusses the contributions of the chapter and outlines possible future directions of research.

4.1 Estimating Tree State-Space

4.1.1 Spectral Analysis

The structure of any FSA with $M$ states can be represented as an $M \times M$ adjacency matrix $A$, where a non-negative integer $t$ in the $(i,j)$-entry indicates $t$ events leading from state $i$ to state $j$. An adjacency matrix is differentiable from a transition matrix (as mentioned in Section 3.3.2) because it is only concerned with the number of events between states, where a transition matrix incorporates event probabilities. For example, the FSA in Figure 4.1(a) has an adjacency matrix shown in Figure 4.1(c).

![Figure 4.1: (a) Example FSA; (b) resulting lookahead tree unwound from state 1 to depth 4; and (c) corresponding adjacency matrix](image-url)
Raising $A$ to the power of $n$ will yield a matrix whose $(i,j)$-entry contains the number of unique strings of events of length $n$ leading from state $i$ to state $j$ [32]. If we expand on this, the number of states at depth $n$ in a lookahead tree unwound from state $x$ is $\sum_{j=1}^{M} a_{xj}^{(n)}$, where $a_{xj}^{(n)}$ denotes the $(x,j)$-entry in the matrix resulting from $A$ being raised to the power of $n$. Say $S^n_x$ defines the number of states in the entire $n$-depth lookahead tree unwound from state $x$, then the number of states at every depth up to $n$ in the tree must be calculated and added:

$$S^n_x = \sum_{i=1}^{M} \sum_{j=0}^{n} a_{xi}^{(j)}$$  \hspace{1cm} (4.1)

In this manner, the exact state space of a lookahead tree can be calculated, but requires computation at every step in the tree which becomes expensive at larger depths. Say $S^n = \sum_{j=0}^{n} A^j = I + A + A^2 + \cdots + A^n$. We can simplify the computation of $S^n$ by multiplying by $(A - I)$ on the right side and cancelling (provided $A - I$ is invertible).

$$S^n(A - I) = (A + A^2 + \cdots + A^{n+1}) - (I + A + \cdots + A^n)$$
$$S^n(A - I) = A^{n+1} - I$$
$$S^n = (A^{n+1} - I)(A - I)^{-1}$$

This provides a method for calculating $S^n_x$ with only $\log_2 (n + 1)$ matrix multiplications required to get $A^{n+1}$.

$$S^n_x = \sum_{i=1}^{M} ((A^{n+1} - I)(A - I)^{-1})_{xi}$$  \hspace{1cm} (4.2)

This method is only applicable when $A - I$ is invertible, which is true when none of the eigenvalues of $A$ are equal to 1. No relationship between FSA structure and $A - I$ invertibility has been observed. It is still possible to simplify the calculation of $S^n_x$ when $A - I$ is not invertible as long as $A$ is diagonalizable. In this case, $A$ can be decomposed into the form $VDV^{-1}$, where $D$ (canonical form) is the diagonal matrix of the eigenvalues $(\lambda_1, \lambda_2, \ldots, \lambda_M)$ of $A$, and each column in $V$ (modal matrix) is an eigenvector of $A$ [15].
\[ A = V D V^{-1} \]
\[ A^n = V D^n V^{-1} \]
\[ \sum_{i=0}^{n} A^i = V \Delta^n V^{-1} \]

where \( \Delta^n = \sum_{i=0}^{n} D^i \). Say the \((i,j)\)-entries of \( V \) and \( V^{-1} \) are \( v_{ij} \) and \( u_{ij} \) respectively, and the \((i,i)\)-entry of \( \Delta^n \) is \( \delta_i^n \), then the lookahead tree state-space can be defined as

\[ S^n_x = \sum_{i=1}^{M} (v_{xi} \cdot \delta_i^n \cdot \sum_{j=1}^{M} u_{ij}) \quad (4.3) \]

Using (4.3) requires only the iterative computation of \( \Delta^n \), which amounts to summing the powers of the eigenvalues of \( A \) as opposed to summing the powers of \( A \) itself. We are interested in defining a good state-space estimator that requires no iterative computation. A few simple observations lead to what we call the \( \tau \)-estimate for lookahead tree state-space.

### 4.1.2 \( \tau \)-Estimation

We will now introduce the \( \tau \) parameter which will help to characterize the size of the state space of lookahead trees. It should be noted at this time that the growth rate of a lookahead tree can be either exponential or polynomial, but no intermediate rate is possible [17]. Gawrychowski et al. present polynomial time algorithms for determining if growth is polynomial and calculating the exact order of polynomial growth in [17]. We present a method of \( \tau \)-estimation that can be used when the growth is exponential, which is true when \( \lambda_k > 1 \). We first assume in section 4.1.3 that \( A \) is \emph{primitive}, meaning that it is a non-negative square matrix with the property that every element in \( A^n \) is greater than 0 for some \( n \). Because \( A \) is defined by an FSA, it is guaranteed to be square and non-negative, and every element in \( A^n \) will be greater than 0 for some \( n \) assuming that the FSA is \emph{strongly connected}, meaning that it is possible to reach any state from every other state. This property is referred to as \emph{irreducibility} in Markov Chains, and is characterized by all states being able to \emph{communicate}. By the Perron-Frobenius theorem [28] we know that if \( A \) is primitive then it has a unique largest eigenvalue \( \lambda_k \).
By unravelling a variety of FSAs into trees and examining their associated adjacency matrices, we observed two trends in the state space of exponential growth lookahead trees with respect to their decompositions when $A$ is primitive.

1. At large depths, lookahead tree state space increases by a factor of $\lambda_k$ for each additional step, i.e., $S^n_x \approx S^{n-1}_x \cdot \lambda_k$. If $\lambda_k = 1$ then there is no exponential growth in the lookahead tree.

2. At large depths, the state spaces of the lookahead trees unwound from each state are in the same proportions as the values of the eigenvector corresponding to $\lambda_k$, i.e., $S^n_x \propto v_{xk}$. Note that $v_{xk}$ refers to the entry corresponding to state $x$ in the dominant eigenvector, which is found in column $k$ of the modal matrix, $v$.

From the observations that $S^n_x$ is related to $\lambda^n_k$ and $v_{xk}$, it is convenient to factor these terms and call the remainder $\tau$. This produces a new expression for state space:

$$S^n_x = \lambda^n_k \cdot v_{xk} \cdot \tau$$  \hspace{1cm} (4.4)

Every FSA has a $\tau$ parameter, where $\tau = \frac{S^n_x}{\lambda^n_k \cdot v_{xk}}$. Many FSAs will have a $\tau$ that converges onto a single value at large depths, while some will have a $\tau$ that settles into oscillation between more values; in both cases we refer to this behaviour as the steady state of $\tau$, denoted by $\tau^\infty$. Figure 4.2 shows $\tau$ for the FSA in Figure 4.1(a), and is indicative of the convergence behaviour of $\tau$ for many FSAs, although larger FSAs may take more time to reach a steady state. When $\tau^\infty$ is used in (4.4), the left-hand side of the equation is denoted by $\tilde{S}^n_x$, i.e.,

$$\tilde{S}^n_x = \lambda^n \cdot v_{xk} \cdot \tau^\infty$$  \hspace{1cm} (4.5)

### 4.1.3 Computing $\tau$ for Primitive, Diagonalizable Adjacency Matrices

In cases when $\tau$ converges to a single value it is possible to compute the asymptotic value of $\tau$ from the decomposition of $A$. Note that this is only necessary when the lookahead tree state space is increasing exponentially, i.e., $\lambda_k > 1$. This is formalized as follows.
Figure 4.2: The $\tau$ parameter for depths 0 through 20 for the FSA shown in Figure 4.1(a). The steady-state value for $\tau$ is determined by the expression in (4.5) in Section 4.1.3.

**Theorem 1.** If the adjacency matrix $A$ of an FSA is primitive, then the asymptotic value of $\tau$, denoted $\tau^\infty$, is given by $\tau^\infty = \frac{\lambda_k}{\lambda_k - 1} \cdot \sum_{j=1}^{M} u_{kj}$.

**Proof.** To calculate the asymptotic value of $\tau$ we must solve for $\tau$ in $\lim_{n \to \infty} \{ \tilde{S}_n \} = \lim_{n \to \infty} \{ S_n \}$. From (4.3) and (4.5) we have

$$\lim_{n \to \infty} \{ \lambda^n_k \cdot v_{xk} \cdot \tau^\infty \} = \lim_{n \to \infty} \{ \sum_{i=1}^{M} (v_{xi} \cdot \delta^n_i \cdot \sum_{j=1}^{M} u_{ij}) \}$$

$$\tau^\infty = \lim_{n \to \infty} \{ \sum_{i=1}^{M} \left( \frac{v_{xi}}{v_{xk}} \cdot \delta^n_i \cdot \sum_{j=1}^{M} u_{ij} \right) \}$$

When the largest eigenvalue $\lambda_k$ is distinct in magnitude (which is the case when $A$ is primitive, by the Perron-Frobenius theorem [28]), then cases when $i$ is not equal to $k$ can be ignored because $\lambda_k > \lambda_i$, therefore as $n \to \infty$, $\delta^n_i / \lambda_k \to 0$. When $i$ is equal to $k$ the fraction $v_{xi} / v_{xk}$ equates to 1, which yields

$$\tau^\infty = \lim_{n \to \infty} \{ \frac{\delta^n_k}{\lambda_k^n} \} \cdot \sum_{j=1}^{M} u_{kj}$$

Since $\delta^n_k = \sum_{i=0}^{n} \lambda_i^n$, and $\lim_{n \to \infty} \{ \frac{i=0}{\lambda_k^n} \} = \lim_{n \to \infty} \{ 1 + \frac{1}{\lambda_k} + \frac{1}{\lambda_k^2} + \cdots + \frac{1}{\lambda_k^n} \} = \frac{\lambda_k}{\lambda_k - 1}$ (by the geometric series result, since $|\frac{1}{\lambda_k}| < 1$), we arrive at an equation for steady state $\tau$

$$\tau^\infty = \frac{\lambda_k}{\lambda_k - 1} \cdot \sum_{j=1}^{M} u_{kj} \quad (4.6)$$
CHAPTER 4. STATE-SPACE ANALYSIS

Once $\tau^\infty$ is known, it can be used with $\tilde{S}_x^n$ to calculate state space estimates for all $x$ and for all $n$. Estimates will be more accurate for larger $n$ values, which are the cases that would require more iterative calculations otherwise.

4.1.4 Examples

Simple Example

A $\tau$-estimate will be computed for the FSA in Figure 4.1(a). Throughout this example values are rounded to six decimal places. First we find

$$A = V D V^{-1}$$

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
-0.752938 & 0.400447 & -0.707107 \\
0.465341 & 0.647936 & 0 \\
0.465341 & 0.647936 & 0.707107
\end{bmatrix}
\begin{bmatrix}
-0.618034 & 0 & 0 \\
0 & 1.618034 & 0 \\
0 & 0 & -1.618034
\end{bmatrix}
\begin{bmatrix}
-0.961045 & 1.555003 & -0.961045 \\
0.690212 & 0.426575 & 0.690212 \\
0 & -1.414214 & 1.414214
\end{bmatrix}
\]

The largest eigenvalue $\lambda_k = 1.618034 = \lambda_2$. Now $\tau^\infty$ can be calculated using (4.5) with $k = 2$.

$$\tau^\infty = \frac{\lambda_2}{\lambda_2 - 1} \cdot \sum_{j=1}^{M} u_{2j}$$

$$= \frac{1.618034}{0.618034} \cdot 0.690212 + 0.426575 + 0.690212$$

$$= 4.730785$$

Having obtained the steady state $\tau$ value corresponding to this FSA, the state space of any of its lookahead trees can be estimated. Say we want to estimate the state space of the tree unwound from state 1 to depth 30:

$$\tilde{S}^{30}_{1} = \lambda_2^{30} \cdot v_{12} \cdot \tau^\infty$$

$$= 1.618034^{30} \cdot 0.400447 \cdot 4.730785$$

$$= 3524581$$

This estimate is only 4 more than the actual state space of the tree $S_{1}^{30} = 3524577$. Table 4.1 compares the $\tau$-estimates and actual state spaces of various lookahead trees derived from the example FSA in Figure 4.1(a). Note that states 2 and 3 have identical lookahead trees because they have identical rows in the adjacency matrix. Underestimation for trees from states 2 and 3 at higher depths is a result of the fact that $v_{2k}$ and $v_{3k}$ were rounded down to 6 decimal places, while $v_{1k}$ was rounded up. To ensure overestimation, values should be rounded up.
Table 4.1: Estimated and actual state spaces for Figure 4.1(a) FSA

<table>
<thead>
<tr>
<th>$n$</th>
<th>$x$</th>
<th>$\tilde{S}_x^n$</th>
<th>$S_x^n$</th>
<th>$\tilde{S}_x^n - S_x^n$</th>
<th>% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>14.28571</td>
</tr>
<tr>
<td></td>
<td>2,3</td>
<td>13</td>
<td>11</td>
<td>2</td>
<td>18.18182</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>21</td>
<td>20</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2,3</td>
<td>34</td>
<td>32</td>
<td>2</td>
<td>6.25</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>233</td>
<td>232</td>
<td>1</td>
<td>0.431034</td>
</tr>
<tr>
<td></td>
<td>2,3</td>
<td>377</td>
<td>375</td>
<td>2</td>
<td>0.533333</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>28657</td>
<td>28656</td>
<td>1</td>
<td>0.00349</td>
</tr>
<tr>
<td></td>
<td>2,3</td>
<td>46368</td>
<td>46366</td>
<td>2</td>
<td>0.004314</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>3524581</td>
<td>3524577</td>
<td>4</td>
<td>0.000113</td>
</tr>
<tr>
<td></td>
<td>2,3</td>
<td>5702885</td>
<td>5702885</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>433494894</td>
<td>433494436</td>
<td>458</td>
<td>0.000106</td>
</tr>
<tr>
<td></td>
<td>2,3</td>
<td>701408545</td>
<td>701408731</td>
<td>-186</td>
<td>0.000027</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>53316351077</td>
<td>53316291172</td>
<td>59905</td>
<td>0.000112</td>
</tr>
<tr>
<td></td>
<td>2,3</td>
<td>86267554136</td>
<td>86267571270</td>
<td>-17134</td>
<td>0.00002</td>
</tr>
</tbody>
</table>

Run Time Example

Recall the MACHINE example from Section 3.3. Imagine we are tasked with calculating running times of an implementation of RUBLL applied to this problem. This could be a standard task when initially attempting to determine which depths can be used for certain states when a time limit is imposed on each decision being made. Say we need to find the run time of this RUBLL implementation for trees unwound from the WORKING and BROKEN states for depths 20 through 40. An easy way to do this would be to simply time the algorithm on these trees, but this could take an incredibly long time depending on the implementation. Running the algorithm on the tree unwound from the WORKING state to depth 30 yields a run time of 14212 ms (milliseconds). Run times throughout this example were calculated using a laptop with a 2.53 GHz dual core and 2 GB of RAM and an implementation favouring usability over execution speed. This time could become much longer for larger trees because their state space typically increases exponentially with depth. Using $\tau$-estimation it is possible to obtain fairly accurate estimates of these run times in a short amount of time.

First we build the adjacency matrix $A$ for the MACHINE problem and calculate its largest eigenvalue $\lambda_k$ and corresponding eigenvector $v$, shown below.
CHAPTER 4. STATE-SPACE ANALYSIS

\[
\begin{bmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
0.372103 \\
0.684404 \\
0.443356 \\
0.443356 \\
\end{bmatrix} =
\begin{bmatrix}
1.839287 \\
0.684404 \\
0.443356 \\
0.443356 \\
\end{bmatrix}
\]

Rows and columns 1, 2, 3 and 4 correspond to states IDLE, WORKING, STILL WORKING and BROKEN respectively. Throughout these calculations values are rounded to six decimal places. The decomposition of \( A \) can be used to calculate the \( \tau^\infty \) value corresponding to this problem, which will be explained later in the chapter. Equation (4.6) can be used to calculate \( \tau^\infty \) from \( \lambda_k \) and the inverse modal matrix which is not shown here.

\[
\tau^\infty = \frac{\lambda_k}{\lambda_k - 1} \cdot \sum_{j=1}^{M} u_{kj}
\]

\[
\tau^\infty = \frac{1.839287}{0.839287} \cdot 2.153229
\]

\[
\tau^\infty = 4.718776
\]

Having obtained the \( \tau^\infty \) value for the problem, we can now use (4.5) to estimate the state space of its trees. For instance, we can estimate the state space of the tree unwound from the WORKING state to depth 30:

\[
\tilde{S}^{30}_2 = \lambda^{30}_2 \cdot v_{2k} \cdot \tau^\infty
\]

\[
= 1.839287^{30} \cdot 0.684404 \cdot 4.718776
\]

\[
= 280948643
\]

So if this RUBLL implementation is able to traverse the 280948643 state tree in 14212 ms, we can estimate that it can calculate about 19768.41 states of a tree every millisecond. Now we have a way to estimate the state space of a tree, and an estimate on how quickly this RUBLL implementation will traverse the tree. These tools can be used to easily calculate estimates for the run times for many trees without any further referral to the implementation.

Table 4.1 and 4.2 show the estimated and actual run times of RUBLL on trees unwound from the WORKING and BROKEN states, respectively. In these tables, \( n \) is the depth of the tree, \( \tilde{S}^n_x \) is the estimated state space of the tree and est. RT (ms) is the estimated run time in milliseconds, \( S^n_x \) is the actual state space of the tree and actual RT (ms) is the actual run time in milliseconds. State space estimates and run time estimates are rounded to the nearest whole number.
Table 4.2: Estimated and Actual state spaces and run times in milliseconds, and calculated % error of run times for RUBLL algorithm implemented on trees unwound from WORKING state in the MACHINE problem from Section 3.3

<table>
<thead>
<tr>
<th>$n$</th>
<th>$S^*_n$ est. RT (ms)</th>
<th>$S^*_n$ actual RT (ms)</th>
<th>% error (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>634062</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>1166222</td>
<td>59</td>
<td>8.47</td>
</tr>
<tr>
<td>22</td>
<td>2145017</td>
<td>109</td>
<td>0.92</td>
</tr>
<tr>
<td>23</td>
<td>3945301</td>
<td>200</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td>7256541</td>
<td>367</td>
<td>6.54</td>
</tr>
<tr>
<td>25</td>
<td>13346862</td>
<td>675</td>
<td>1.93</td>
</tr>
<tr>
<td>26</td>
<td>24548709</td>
<td>1242</td>
<td>0.56</td>
</tr>
<tr>
<td>27</td>
<td>45152121</td>
<td>2284</td>
<td>0.48</td>
</tr>
<tr>
<td>28</td>
<td>83047710</td>
<td>4201</td>
<td>1.07</td>
</tr>
<tr>
<td>29</td>
<td>152748573</td>
<td>7727</td>
<td>0.35</td>
</tr>
<tr>
<td>30</td>
<td>280948465</td>
<td>14212</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>516744859</td>
<td>26140</td>
<td>0.03</td>
</tr>
<tr>
<td>32</td>
<td>950442102</td>
<td>48079</td>
<td>0.07</td>
</tr>
<tr>
<td>33</td>
<td>1748135802</td>
<td>88431</td>
<td>0.77</td>
</tr>
<tr>
<td>34</td>
<td>3215323455</td>
<td>162650</td>
<td>0.30</td>
</tr>
<tr>
<td>35</td>
<td>5913902631</td>
<td>299159</td>
<td>1.24</td>
</tr>
<tr>
<td>36</td>
<td>10877364228</td>
<td>550240</td>
<td>2.02</td>
</tr>
<tr>
<td>37</td>
<td>20006594619</td>
<td>1012049</td>
<td>1.20</td>
</tr>
<tr>
<td>38</td>
<td>36797869397</td>
<td>1861448</td>
<td>0.56</td>
</tr>
<tr>
<td>39</td>
<td>67681842809</td>
<td>3423737</td>
<td>0.50</td>
</tr>
<tr>
<td>40</td>
<td>124486333615</td>
<td>6297236</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Equation (4.5) can be used to estimate the state space of the trees of depth 20 unwound from the WORKING state, $\tilde{S}^*_{20}$, and from the BROKEN state, $\tilde{S}^*_{40}$. At this point, incremental estimates can be calculated by simply multiplying the preceding estimate by $\lambda_k$, which corresponds to the asymptotic rate of growth of the state space of the trees. Run time estimates are determined by dividing these state space estimates by 19768.41. After obtaining this initial speed estimate for the implementation, and the decomposition of the underlying adjacency matrix, estimates for all depths and states can be calculated quite easily, which may be favourable over the 6 hours and 22 minutes required to calculate the actual run times for these 42 trees. With proper implementation, accurate run time estimates can be obtained for these 42 trees in less than one minute. Note that the estimated values of run times are based on the additional assumption that this implementation of RUBLL has a speed of 19768.41 states per millisecond, so the % error for state space estimates would typically be lower than the % error for run time estimates.
Table 4.3: Estimated and Actual state spaces and run times in milliseconds, and calculated % error of run times for RUBLL algorithm implemented on trees unwound from BROKEN state in the MACHINE problem from Section 3.3

<table>
<thead>
<tr>
<th>n</th>
<th>$S^n_x$</th>
<th>est. RT (ms)</th>
<th>$S^n_x$</th>
<th>actual RT (ms)</th>
<th>% error (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>410745</td>
<td>21</td>
<td>410743</td>
<td>16</td>
<td>23.81</td>
</tr>
<tr>
<td>21</td>
<td>755477</td>
<td>38</td>
<td>755475</td>
<td>48</td>
<td>26.32</td>
</tr>
<tr>
<td>22</td>
<td>1389540</td>
<td>70</td>
<td>1389536</td>
<td>63</td>
<td>10</td>
</tr>
<tr>
<td>23</td>
<td>2555762</td>
<td>129</td>
<td>2555756</td>
<td>126</td>
<td>2.33</td>
</tr>
<tr>
<td>24</td>
<td>4700781</td>
<td>238</td>
<td>4700769</td>
<td>251</td>
<td>5.46</td>
</tr>
<tr>
<td>25</td>
<td>8646085</td>
<td>437</td>
<td>8646063</td>
<td>438</td>
<td>0.23</td>
</tr>
<tr>
<td>26</td>
<td>1590263</td>
<td>804</td>
<td>1590259</td>
<td>827</td>
<td>2.86</td>
</tr>
<tr>
<td>27</td>
<td>2924950</td>
<td>1480</td>
<td>2924942</td>
<td>1483</td>
<td>0.20</td>
</tr>
<tr>
<td>28</td>
<td>5379823</td>
<td>2721</td>
<td>5379807</td>
<td>2793</td>
<td>2.65</td>
</tr>
<tr>
<td>29</td>
<td>9895038</td>
<td>5005</td>
<td>9895009</td>
<td>5102</td>
<td>1.94</td>
</tr>
<tr>
<td>30</td>
<td>1819981</td>
<td>9207</td>
<td>1819976</td>
<td>9423</td>
<td>2.35</td>
</tr>
<tr>
<td>31</td>
<td>3347468</td>
<td>16933</td>
<td>3347457</td>
<td>17255</td>
<td>1.90</td>
</tr>
<tr>
<td>32</td>
<td>6156955</td>
<td>31145</td>
<td>6156934</td>
<td>31763</td>
<td>1.98</td>
</tr>
<tr>
<td>33</td>
<td>1132440</td>
<td>57285</td>
<td>1132436</td>
<td>57970</td>
<td>1.20</td>
</tr>
<tr>
<td>34</td>
<td>2082883</td>
<td>105364</td>
<td>2082876</td>
<td>106877</td>
<td>1.44</td>
</tr>
<tr>
<td>35</td>
<td>3831020</td>
<td>193795</td>
<td>3831006</td>
<td>200618</td>
<td>3.52</td>
</tr>
<tr>
<td>36</td>
<td>7046346</td>
<td>356445</td>
<td>7046319</td>
<td>366211</td>
<td>2.74</td>
</tr>
<tr>
<td>37</td>
<td>1296025</td>
<td>655604</td>
<td>1296020</td>
<td>669850</td>
<td>2.17</td>
</tr>
<tr>
<td>38</td>
<td>2383762</td>
<td>1205844</td>
<td>2383752</td>
<td>1224619</td>
<td>1.56</td>
</tr>
<tr>
<td>39</td>
<td>4384423</td>
<td>2217894</td>
<td>4384404</td>
<td>2247013</td>
<td>1.31</td>
</tr>
<tr>
<td>40</td>
<td>8064213</td>
<td>4079343</td>
<td>8064177</td>
<td>4118034</td>
<td>0.95</td>
</tr>
</tbody>
</table>

4.2 Computing $\tau$ for General Case

In this section we explore cases where the assumptions in Section 4.1 may not hold. For some of these cases we do not have closed-form solutions, but provide some observations and simplifications.

4.2.1 Non-Mixing

What we call mixing is a characteristic of FSAs whose trees quickly settle into the proportions dictated by the eigenvector. The adjacency matrix of a mixing FSA corresponds to an aperiodic irreducible Markov Chain. When the dominant eigenvalue is distinct in magnitude, the proportions of states from the FSA at each depth of the lookahead tree will be roughly equal for all depths. The actual proportions for each state are defined by the same row in the inverse modal matrix used to calculate $\tau^\infty$, i.e., state $x$ from the FSA will make up approximately $u_{kx}/\sum_{i=1}^{M} u_{ki}$ of the states of
the lookahead tree at any depth. This mixing of states throughout depths is what allows a simplified estimate to be computed.

Not all FSAs will be mixing. Some FSAs are structured so that their trees will never settle into the specified proportions; we call these non-mixing. A non-mixing FSA’s adjacency matrix corresponds to a periodic Markov Chain. The structure of non-mixing FSAs restricts the number of certain states at certain depths of the tree. Non-mixing FSAs can be characterized by looking at the lengths of the paths from some state to itself. Say the lengths of the paths are \(l_1, l_2, \ldots, l_p\); if the greatest common denominator of these values is not 1 then the FSA is non-mixing. The length of the growth cycle (referred to as the period) of a non-mixing FSA is equal to this greatest common denominator, meaning that there will be \(\text{gcd}(l_1, l_2, \ldots, l_p)\) repeating stages of growth in the lookahead tree, and as many \(\tau\) values. This is because an irreducible matrix with period \(h\) has \(h\) dominant eigenvalues with the same magnitude and different phases [28]. Regarding this we have the following Theorem.

**Theorem 2.** Suppose that the adjacency matrix \(A\) of an FSA is irreducible with period \(h\) and the \(h\) dominant eigenvalues of \(A\) are

\[
\lambda_m = r e^{\frac{2\pi i (m-1)}{h}}, \quad m = 1 \ldots h
\]

If \(A\) is rearranged such that those \(h\) dominant eigenvalues are the top \(h\) diagonal elements of the Jordan canonical form of \(A\), then for large \(n\), \(\tau^\infty\) can be calculated by:

\[
\tau^\infty_x = \frac{1}{v_{x1}} \sum_{m=1}^{h} \left\{ v_{xm} \left( r e^{\frac{2\pi i (m-1)(n+1)}{h}} - r^{-n} \sum_{k=1}^{M} u_{mk} \right) \right\} - r^{-n} \sum_{k=1}^{M} u_{mk} \}
\]

(4.7)

**Proof.** After rearranging the matrix \(A\), \(\lambda_1\) would be a real dominant eigenvalue. As in the proof of Theorem 1, we have

\[
\lim_{n \to \infty} \left\{ \lambda_1^n v_{x1} \tau \right\} = \lim_{n \to \infty} \left\{ \sum_{m=1}^{M} (v_{xm} \cdot \delta_m^n \cdot \sum_{j=1}^{M} u_{mj}) \right\}
\]

\[
\tau^\infty = \lim_{n \to \infty} \left\{ \frac{1}{v_{x1}} \sum_{m=1}^{M} v_{xm} \cdot \frac{\delta_m^n}{\lambda_1^n} \cdot \sum_{j=1}^{M} u_{mj} \right\}
\]

As discussed in the proof of Theorem 1, for all \(|\lambda_m| < \lambda_1\), \(\frac{\delta_m^n}{\lambda_1^n} \to 0\) as \(n \to \infty\). Therefore, contributions from eigenvalues other than the \(h\) dominant ones are negligible. So we have

\[
\tau^\infty_x = \frac{1}{v_{x1}} \sum_{m=1}^{h} v_{xm} \left( \sum_{j=0}^{n} r^{-j} e^{\frac{2\pi i (m-1)j}{h}} \right) (\sum_{k=1}^{M} u_{mk}) \}
\]
which, using the geometric series formula, can be written in the form

\[
\tau^\infty_x = \frac{1}{v_{x1}} \sum_{m=1}^h \left\{ v_{xm} r^e \frac{2\pi n (m-1)(n+1)}{r^e} - r^{-n} \left( \sum_{k=1}^M u_{mk} \right) \right\}
\]

As a corollary when the period is two, the matrix will have two dominant eigenvalues \( \lambda_k, -\lambda_k \).

A simple example of a non-mixing FSA is shown in Figure 4.3(a) with a lookahead tree unwound from state 1 shown in Figure 4.3(b). This FSA has a period of 2, so in the tree state 1 is restricted to even depths and states 2 and 3 are restricted to odd depths.

![Figure 4.3](image)

Figure 4.3: (a) Non-mixing, bipartite, alternating growth FSA; (b) resulting lookahead tree unwound from state 1 to depth 4; and (c) \( \tau \) for depths 0 through 20

This FSA displays a pattern of alternating growth as depicted in Figure 4.3(c), so there will be two different steady state \( \tau \) values, the larger of which refers to depths where more growth has just taken place. It is also bipartite, meaning that it has two disjoint sets of states (\{1\} and \{2, 3\} in this example), and events from one set only lead to states in the other set. The adjacency matrix of bipartite FSAs will always have a dominant eigenvalue that is not distinct in magnitude, i.e., \( \lambda_k \) and \( -\lambda_k \) are both eigenvalues. It is possible to calculate the exact steady state \( \tau \) values for this and other bipartite alternating growth examples. Say \( k^- \) refers to the position of \( -\lambda_k \) in the eigenvalue matrix. The presence of \( \lambda_{k^-} \) changes the way in which the equation for \( \tau^\infty \) reduces.

**Corollary 3.** If the adjacency matrix \( A \) of an FSA has a dominant eigenvalue that is not distinct in magnitude, i.e., \( \lambda_k \) and \( -\lambda_k \) are both eigenvalues, then the asymptotic values of \( \tau \), denoted by \( \tau^\infty_{\text{alt}} \), are given by

\[
\tau^\infty_{\text{alt}} = \lambda_k \sum_{j=1}^M u_{kj} \pm \lambda_k \sum_{j=1}^M u_{k-j}.
\]
CHAPTER 4. STATE-SPACE ANALYSIS

Proof. We solve for \( \tau \) in \( \lim_{n \to \infty} \{ \tilde{S}_x^n \} = \lim_{n \to \infty} \{ S_x^n \} \). From (4.3) and (4.5) we have

\[
\tau^\infty = \lim_{n \to \infty} \left\{ \sum_{i=1}^{M} \left( \frac{v_{xi}}{v_{xk}} \cdot \frac{\delta^n_{i}}{\lambda^n_{k}} \cdot \sum_{j=1}^{M} u_{ij} \right) \right\}
\]

Cases when \( i \) is not equal to \( k \) or \( k^- \) can be ignored because \( \lambda_k > \lambda_i \), therefore as \( n \to \infty \), \( \frac{\delta^n_{i}}{\lambda^n_{k}} \to 0 \). Cases when \( i \) is equal to \( k \) will yield \((\frac{\lambda_k}{\lambda_{k^-}}) \cdot \sum_{i}^{M} u_{ki}\) as in the distinct eigenvalue case. When \( i = k^- \), \( \frac{v_{xi}}{v_{xk}} \) will be equal to 1 if \( x \) is in the set of states with smaller projected growth, and \(-1\) if \( x \) is in the set of states with larger projected growth. A non-mixing FSA having \( \frac{v_{xk^-}}{v_{xk}} \neq \pm 1 \) indicates that it has at least one siphoning component which must be handled in a manner discussed in Section 4.2.2. Also note that \( \lim_{n \to \infty} \left\{ \frac{\delta^n_{k-}}{\lambda^n_{k}} \right\} = \pm \frac{\lambda_k}{\lambda_k + 1} \). This results in

\[
\tau^\infty_{alt} = \frac{\lambda_k}{\lambda_k - 1} \cdot \sum_{j=1}^{M} u_{kj} \pm \frac{\lambda_k}{\lambda_k + 1} \cdot \sum_{j=1}^{M} u_{k^-j} \quad (4.8)
\]

We can use the result of Theorem 2 to calculate the \( \tau \) values for the FSA in Figure 4.3(a):

\[
\tau^\infty_{alt} = \frac{1.414214}{1.414214 - 1} \cdot (0.707107 + 0.5 + 0.5) \pm \frac{1.414214}{1.414214 + 1} \cdot (-0.707107 + 0.5 + 0.5)
\]

These steady state \( \tau \) values are consistent with \( \tau \) shown in Figure 4.3(c). This approach will work for any non-mixing FSA with alternating growth.

Not all bipartite FSAs will have alternating growth, and not all non-mixing FSAs are bipartite. In these cases an equation for steady state \( \tau \) must be calculated on a case by case basis. Failing this, estimates for steady state \( \tau \) values can be obtained by solving for \( \tau \) in \( \tilde{S}_x^n = S_x^n \) at reasonably high depths.

For computing \( \tau \), besides using Theorem 2 (which requires computing spectral decomposition of \( A \)), we can proceed as follows. The FSA shown in Figure 4.4(a) is bipartite but it has a period of four, so it will have four steady state \( \tau \) values as opposed to two. In this case an estimate can be substituted by applying (4.4) to the largest growth state:

\[
\tau^e = \frac{S_x^e}{v_{xk} \cdot \lambda_k^e} \quad (4.9)
\]
Figure 4.4: (a) Non-mixing, bipartite FSA with four $\tau$ values and (b) resulting lookahead tree unwound from state 1 to depth 5

where $\omega$ is the largest growth state and $e$ is the depth at which $\tau$ is estimated. The largest growth state will generally give better estimates than other states when compared at equal depths. With a period of four, $e$ should be a multiple of 4 to help keep track of things. The remaining three $\tau$ values can be estimated by $\tau_{e+1}$, $\tau_{e+2}$ and $\tau_{e+3}$. When calculating $\tilde{S}_n^\omega$, the $\tau$ value to use can be determined by looking at $n$ modulo 4, i.e., $\tau_{e+(n \mod 4)}$ should be used. When calculating $\tilde{S}_x^n$ with $x \neq \omega$, the $\tau$ value to use should be incremented by the distance from $\omega$ to $x$. For instance, $\tau_{e+1}$ would be used to calculate $\tilde{S}_4^{20}$. This approach of estimation can be adapted to any non-mixing FSA.

4.2.2 Component Analysis

Many FSAs will not be strongly connected (referred to as *reducible* when dealing with Markov Chains), and some will have adjacency matrices that are not diagonalizable. In these cases the problem can be broken down into components which are suitable for $\tau$-estimation.

**Siphoning Components**

When an FSA is not strongly connected it can be interpreted as a set of strongly connected components which are unidirectionally connected to each other. In the case that these strongly connected components consist of no more than one state each, there can be no exponential growth because unidirectionally connected states will not create branching in the lookahead tree. A *siphoning component* is a component with no outgoing transitions. The term *siphon* has been roughly adapted from Petri net theory [2]; in Markov chain terminology a siphoning component is called a *closed state*. A FSA with one or more siphoning components will not have a primitive adjacency matrix. In this case it is not always possible to calculate a single $\tau^\infty$ for the entire FSA, specifically when a siphoning component has a smaller rate of growth.
than the largest growth component. States in these siphoning components will have 0 entries in the eigenvector corresponding to the largest eigenvalue, so the $\tau_\infty$ value calculated for the full FSA will not be applicable to trees being unwound from these states. In order to estimate the state space of trees unwound from states in one of these components, it must be treated as an independent FSA with its own $\tau_\infty$.

A siphoning component may be made up of different components with different growth rates, so it may contain siphoning components with even smaller rates of growth. It is possible to compute the $\tau_\infty$ for all of the states in an FSA with siphoning components by recursively removing the states that can reach the largest growth component and recalculating. This is accomplished as follows:

1. Calculate $\tau_\infty$ for the FSA in question. This $\tau_\infty$ value will apply to all of the states that can reach the largest growth component, which are the states $x$ for which $v_{xk} \neq 0$.

2. Remove these states from the FSA and calculate the new $\tau_\infty$ values for the resulting components. These values can be used to $\tau$-estimate trees unwound from states in these components. If a component has a non-largest growth siphoning component within it, return to 1.

The example shown in Figure 4.5(a) requires some special attention because it has a siphoning component (states 3, 4 and 5) with a smaller rate of growth than the rest of the system. For estimating trees unwound from states 1 or 2, the method for steady state $\tau$ calculation obtained in Section 4.1.3 can be applied to the full adjacency matrix. However, for estimating trees unwound from states 3, 4 or 5, the method must be applied to this siphoning component treated as an independent 3-state FSA.

The FSA shown in Figure 4.5(b) differs from the FSA in Figure 4.5(a) in that the siphoning component (states 1 and 2) has the larger growth rate (since (b) transitions

![Figure 4.5](image-url)
from state 3 to state 2, while (a) transitions from state 2 to state 3). For the FSA in Figure 4.5(b), $\tau^\infty$ can be calculated using the full adjacency matrix, and this can be used to estimate trees unwound from any state.

**Non-Diagonalizable**

An adjacency matrix with two or more identical eigenvalues will often not be diagonalizable, in which case $\tau^\infty$ cannot be calculated as in the dominant eigenvalue case. This often occurs when an FSA has two or more separate components with identical rates of growth. These problems can sometimes be solved by analyzing the components individually, though for the most part this remains an open problem.

![Figure 4.6: FSA with identical components \{2, 3\} and \{4, 5\}](image)

The lookahead tree state space of the FSA shown in Figure 4.6 can be approximated by calculating steady state $\tau$ for one of the identical components, $\hat{S}_x^n$. When $x = 2, 3, 4, 5$, $\hat{S}_x^n$ can be used to estimate $S_x^n$. However, it is not possible to calculate an exact $\tau^\infty$ value for state 1 because the full FSA is not diagonalizable. In this case an estimate for $\tau$ can be calculated using (4.9). Not surprisingly, $\tau^\infty$ for state 1 is double the value obtained for the identical components: $\tau_1^\infty = 7.2068$, $\tau_{2,3,4,5}^\infty = 3.6034$. This example is problematic and must be analyzed as separate components, however, the problem becomes impossible to solve using $\tau$ when the identical components have the largest growth in the FSA and they are unidirectionally connected to each other through any set of other components. When two identical components are connected in some way, their collective growth rate becomes greater than the growth rate of each individual component, but the largest eigenvalue of this system will only reflect the largest growth of any component. In this case, (4.4) no longer captures the growth behaviour of the FSA and $\tau$ will never reach a steady state. These FSAs still each have a $\tau$ value; let us call it $\tau'$. Instead of (4.4), $\tau'$ is defined by

$$S^n_x = G^n \cdot v_{xk} \cdot \tau'$$

(4.10)
where $G$ is the actual growth rate of the system. We have been unable to determine a method for exactly calculating the growth rate in this case.

Figure 4.7: Non-invertible FSA with connected identical components {3, 4} and {5, 6}

The FSA shown in Figure 4.7 is an example for which it is difficult to simplify the method for estimating state space. It is not possible to calculate $\tau$ for states 1, 2, 3 or 4 without knowing the growth rate of these combined components. It is not possible to calculate $\tau$ for states 1 and 2 because the adjacency matrix is not diagonalizable. This also means that simplifications observed in (4.3) are not applicable. Finally, it is not possible to calculate exact state-space values using (4.2) because $A - I$ is not invertible. This means that in order to get exact state-space values, (4.1) must be used, which requires computation at every step in the tree. An alternative is to use (4.1) to calculate a fair estimate of the step-wise growth rate $G$, and use this value to estimate $\tau'$ in (4.10). We estimate the growth rate of this FSA to be 1.6463 at depth 50 by calculating $S_x^{50}/S_x^{49}$ for any $x$. The algorithm discussed in Section 4.4 will use this estimated growth rate to calculate a $\tau$ estimate for states 1, 2, 3 and 4, and will compute a separate estimate for states 5 and 6 treated as their own automaton.

4.3 Complexity Analysis

Here we present a step-by-step guide to estimating the state space of a lookahead tree while considering the computational complexity of the various methods.

1. **Determine the growth rate.** This can be accomplished by looking at the largest eigenvalue $\lambda_k$ of the adjacency matrix (where $\lambda_k > 1$ indicates exponential growth), or by using the method presented in [17]. The largest eigenvalue is determined by decomposing the adjacency matrix $A$ into the form $V D V^{-1}$, which for an $m$-by-$m$ matrix has a time complexity of $O(m^3)$. The method presented in [17] has a time complexity of $O(q + t)$, where $q$ is the number of states and $t$ is the number of transitions. If the tree has polynomial growth, its
order of growth can be determined in polynomial time [17]. Otherwise, proceed to 2.

2. **Determine if \( A \) is primitive.** An adjacency matrix will be primitive if its FSA is strongly connected. This can be determined by showing that every state in the FSA can reach every other state, which is the same as showing that every element in \( A^n \) is greater than 0 for some \( n \). If \( A \) is not primitive then the problem must be broken up using component analysis, or the growth rate must be estimated as discussed in Section 4.2.2. In the worst case, component partitioning will require as many \( \tau \) value computations as there are components, although \( m \) (and hence, the complexity of \( \tau \) computations) will be decreasing as states are removed each time. If \( A \) is primitive, proceed to 3.

3. **Determine if \( A \) is diagonalizable.** This is the case when \( A \) has \( M \) distinct eigenvectors, which are present in the decomposition of \( A \). It is also the case that \( A \) is diagonalizable when the algebraic multiplicity of each eigenvalue is equal to its geometric multiplicity [28]. If \( A \) is not diagonalizable then the state space can be calculated exactly using (4.2), assuming that \( A - I \) is invertible. Equation (4.2) requires the computation of \( A^{n+1} \), where \( n \) is the depth of the lookahead tree. This can be calculated using repeated squaring in \( \log_2(n+1) \) matrix multiplications, so the method has time complexity of \( O(m^3 \cdot \log_2 n) \). If \( A - I \) is not invertible, (4.2) will not work and so the problem must be broken up using component analysis as discussed in Section 4.2.2. If \( A \) is diagonalizable then the state space can be accurately estimated using the \( \tau \)-estimate. First it should be determined if the FSA is non-mixing, which is accomplished by looking at the greatest common denominator of the lengths of paths from some state to itself, \( l_1, l_2, \ldots, l_p \). The period \( h \) will be equal to \( \gcd(l_1, l_2, \ldots, l_p) \); if the period is greater than 1, then the FSA will be non-mixing. If \( h = 2 \) then the FSA has an alternating growth and its \( \tau_{\infty}^{\text{alt}} \) values can be calculated exactly using (4.8). If \( h > 2 \) then (4.7) can be used to calculate the exact \( \tau \) values. If \( A \) is not diagonalizable then we have no method for calculating exact \( \tau \) values, so estimates can be determined using (4.9). This requires the calculation of \( S^e \), which can be determined in \( O(m^3 \cdot \log_2 e) \) time using (4.2). After the first \( \tau \) value is found, the remaining \( h - 1 \) values are found by multiplying \( S^e \) by \( A \) each time. This evaluates to a time complexity of \( O(m^3 \cdot (\log_2 e + h)) \). The complexity of calculating \( \tau_{\infty}^{\text{mix}} \) for mixing cases, and \( \tau_{\infty}^{\text{alt}} \) for alternating growth cases is the same. First, steady state \( \tau \) must be calculated from the inverse modal matrix using (4.6) or (4.8); this requires the decomposition of \( A \) which takes \( O(m^3) \)
Once steady state $\tau$ has been found, the state space of lookahead trees unwound from any state in the FSA to any depth can be calculated in linear time using (4.5). If $A$ is not primitive then its exact state space can be calculated using (4.2) assuming that it is invertible.

Equation (4.2) allows the exact state space of a lookahead tree unwound from any state to depth $n$ to be calculated in $O(m^3 \cdot \log n)$ time, and requires $O(m^3)$ for each additional depth. On the other hand, using the $\tau$-estimate depicted in (4.5) takes $O(m^3)$ time to calculate $\tau$, at which point the state space of trees of all depths can be computed in linear time. For cases when either (4.2) or (4.5) can be applied, the approach can be selected based on the requirements of the problem; however, sometimes it is the case that neither can be used (as in Figure 4.7), or that only one of (4.2) or (4.5) will work.

4.4 Algorithm

This section presents two procedures, tauAnalysis and tauEstimate, which together constitute an algorithm for the method of $\tau$ state-space estimation presented throughout this chapter. In order to obtain a state-space estimate, tauAnalysis must be executed on the desired FSA in order to calculate the parameters required by tauEstimate to make the actual estimations. It should be noted that all of the variables assigned throughout the tauAnalysis procedure are associated with the specific automaton $M$ by which the procedure was called. So in the case that there are siphoning components and the state set must be partitioned and analyzed recursively, the parameters calculated in the recursive calls are associated with the specific partitions being called on. These associations are carried over to the tauEstimation procedure.

```
procedure tauAnalysis
  input: deterministic finite automaton $M$

  compute $A$, the adjacency matrix of $M$
  compute $D$, the diagonal matrix of eigenvalues of $A$
  compute $V$, the modal matrix of corresponding eigenvectors of $A$
  $\lambda_k \leftarrow$ largest eigenvalue of $A$

  if $A$ shows exponential growth then
    if $A$ is primitive then
      if $A$ is diagonalizable then

```

$\Rightarrow$ if $\lambda_k > 1$

$\Rightarrow$ if $M$ is strongly connected

$\Rightarrow$ if eigenvalues are unique
find $V^{-1}$ such that $A = VDV^{-1}$

if $A$ is mixing then $\triangleright$ if $gcd(l_1, l_2, \ldots, l_p) < 2$ (Section 4.2.1)

case $= 1$

$$\tau^\infty = \frac{\lambda_k}{\lambda_k - 1} \cdot \sum_{j=1}^{M} u_{kj}$$

else if $gcd(l_1, l_2, \ldots, l_p) = 2$ then $\triangleright$ alternating growth

case $= 2$

$$\tau^+_\infty = \frac{\lambda_k}{\lambda_k - 1} \cdot \sum_{j=1}^{M} u_{kj} + \frac{\lambda_k}{\lambda_k + 1} \cdot \sum_{j=1}^{M} u_{k-j}$$

$$\tau^-\infty = \frac{\lambda_k}{\lambda_k - 1} \cdot \sum_{j=1}^{M} u_{kj} - \frac{\lambda_k}{\lambda_k + 1} \cdot \sum_{j=1}^{M} u_{k-j}$$

else $\triangleright$ non-mixing

case $= 3$

period $\leftarrow gcd(l_1, l_2, \ldots, l_p)$

$\omega \leftarrow$ largest growth state \hspace{1cm} $\triangleright$ state $x$ with largest $v_{xk}$

$e \leftarrow$ user input \hspace{1cm} $\triangleright$ prompt user for depth of estimation

for int $i = 0 \rightarrow$ period do

$$\tau^e_i = S^e_\omega / (v_{\omega k} \cdot \lambda_k)$$

end for

end if

else $\triangleright A$ is not diagonalizable

case $= 4$

$e \leftarrow$ user input \hspace{1cm} $\triangleright$ prompt user for depth of estimation

$G = S^e_x / S^{e-1}_x$

$$\tau^e = S^e_x / (v_{xk} \cdot G)$$

end if

else $\triangleright$ There are siphoning components

case $= 5$

initialize $LGcomp$ and $SGcomp$

for all $x \in X$ do $\triangleright$ for each state

if $v_{xk} \neq 0$ then

add $x$ to $LGcomp$ \hspace{1cm} $\triangleright$ largest growth component

else

add $x$ to $SGcomp$ \hspace{1cm} $\triangleright$ smaller growth component

end if

end for

TAUANALYSIS($LGcomp$) \hspace{1cm} $\triangleright$ recursive calls

TAUANALYSIS($SGcomp$)

end if


else
  print out → no exponential growth detected
end if
end procedure

Once \texttt{tauAnalysis} has been run on an FSA, \texttt{tauEstimate} can be used repeatedly to estimate the state space of trees from any state to any depth. The category of FSA is stored by \texttt{tauAnalysis} in the \textit{case} variable to be referenced by \texttt{tauEstimate}.

\begin{verbatim}
procedure tauEstimate
  inputs: automaton $M$, root state $x$, estimation depth $n$
  output: state space estimate

  if case = 1 then
    return $\lambda_k \cdot v_{zk} \cdot \tau^\infty$
  else if case = 2 then
    $\omega$ ← largest growth state
    $d$ ← shortest path from $\omega$ to $x$
    if $d$ is even then
      return $\lambda_k \cdot v_{zk} \cdot \tau^+_d$
    else
      return $\lambda_k \cdot v_{zk} \cdot \tau^-_d$
    end if
  else if case = 3 then
    $d$ ← shortest path from $\omega$ to $x$
    return $\lambda_k \cdot v_{zk} \cdot \tau^e_d$
  else if case = 4 then
    return $G \cdot v_{zk} \cdot \tau^e$
  else if case = 5 then
    if $x \in LGcomp$ then
      return $\tauEstimate(LGcomp, x, n)$
    else
      return $\tauEstimate(SGcomp, x, n)$
    end if
  end if
end if
\end{verbatim}
end procedure

4.5 Discussion

We have presented a number of methods for simplification of the calculation of the state space of lookahead trees:

(4.1) Formula for exact state space, makes no assumptions but requires iterative computation.

(4.2) Simplified formula for state space, requires that $A - I$ be invertible.

(4.3) Simplified formula for state space, requires that $A$ be diagonalizable.

(4.4) Equation for state space based on $\tau$, requires a $\tau$ value to estimate $S^n_x$.

(4.5) Equation for the steady state $\tau$-estimate, requires $\tau^\infty$ to calculate $\tilde{S}^n_x$, an estimate for $S^n_x$.

(4.6) Equation for calculating $\tau^\infty$ derived from (4.3) and (4.5), requires that $A$ be diagonalizable.

(4.7) Equation for calculating $\tau^\infty_x$ for irreducible cases.

(4.8) Equation for calculating $\tau^\infty_{alt}$ for alternating growth cases derived from (4.3) and (4.5), requires that $A$ be diagonalizable.

(4.9) Equation for calculating an estimate for $\tau$, requires that the growth rate of the system is captured in $\lambda_k$.

(4.10) Equation for the $\tau$-estimate when the growth rate of the system is greater than $\lambda_k$.

In most cases the question is whether the accuracy of using (4.2) is worth the extra complexity over using (4.5), the $\tau$-estimate. In practice it is likely that the user will require many estimates of state space at different depths from different states, and this is where the greatest time savings come with using (4.5). However, we have made no claims as to the accuracy of $\tau$ estimation simply because it is a difficult problem. At very large depths it is true that any $\tau$ estimates calculated using $\tau^\infty$ will converge on the actual state space of the tree, but it is difficult to calculate the rate of this convergence in order to determine the accuracy of estimates made at smaller depths. This rate of convergence, or the rate of the FSA’s mixing, is likely related to the
Frobenius problem. The Frobenius problem is as follows: given the positive integers \( l_1, l_2, \ldots, l_p \) with \( \gcd(l_1, l_2, \ldots, l_p) = 1 \), find the largest integer not representable as a non-negative integer linear combination of these numbers, denoted \( g(l_1, l_2, \ldots, l_p) \) [34]. If \( l_1, l_2, \ldots, l_p \) each represent the length of a path from some state \( x \) in an FSA to itself, then the integers which cannot be represented in this fashion represent the depths in the lookahead tree unwound from \( x \) that do not contain \( x \). The rate of mixing is defined by how quickly the number of states at each depth settle into the proportions dictated by the dominant eigenvector, so depths that are missing some states will not satisfy these proportions. The rate of mixing will be slower the larger \( g(l_1, l_2, \ldots, l_p) \) is, and it will be slower the more depths \( < g(l_1, l_2, \ldots, l_p) \) there are that contain no \( x \). Precisely how the Frobenius problem can be used to determine a metric for rate of mixing remains to be seen.

It should also be noted that the complexity of using (4.5) is due to the complexity of the matrix multiplication algorithm used. The standard matrix multiplication algorithm with \( O(m^3) \) can be substituted with the Coppersmith-Winograd algorithm with \( O(m^{2.376}) \) [14], or the Strassen algorithm with \( O(m^{2.807}) \) [14]. Even with the reduced complexity resulting from use of these algorithms, it is usually cheaper to use (4.5) over (4.2) because in practice the need for estimating the state space of trees to many depths from many states is likely.

Other future research should be aimed at determination of a method for the calculation of the rate of growth \( G \) of an FSA with two or more identical-growth unidirectionally-connected largest growth components. With a method for calculating \( G \) in this case, the \( \tau \)-estimate could be extended to work in all cases with the aid of (4.10) and component analysis.

Still more research in this area could be aimed at the development of a method by which the lookahead tree state space of a composition of many identical modules can be predicted through the analysis of only one of those modules. This would prove especially useful for implementing a limited lookahead approach to decentralized or modular control of discrete-event systems.
Chapter 5

Outbreak Simulation and Results

In this chapter, RUBLL will be applied to a simulation of an outbreak of some fictitious disease designed by Thomas Brunsch in [8]. The simulation is composed of a set of separate modules, so it is not possible to apply RUBLL as it was intended to a single lookahead tree. The solution to this problem is to modify how RUBLL is applied so that it performs a comparable control task in a novel situation. This solution is an example of how RUBLL can be intelligently adapted to work in a wide variety of problems. Section 5.1 discusses the simulation as it was originally designed by Brunsch and replicates his results. Section 5.2 presents in detail the process of applying the RUBLL algorithm to this problem, and the remainder of the chapter discusses the results of this analysis.

5.1 Simulation Description

The simulation models the behaviour of a generic disease (similar to a respiratory disease such as influenza or bronchitis) as it affects a population, and the various actions that a health unit can take in response to the outbreak. It is assumed that the health unit has detected the outbreak at an early stage. The system is composed of many different modules, each representing an exposed individual. The model for each individual is represented as an FSA and discussed in Section 5.1.2. The overall system is represented by a vector of numbers indicating which state each of the modules are in. Because each module in the system is identical, they can all be represented by one single module. Therefore the complete system can be represented by the vector $M$ and a prototype FSA representing the behaviour of each module. The simulation starts with some exposed individuals and continues until there are no more in the system and the outbreak is considered to be contained, or until the number of exposed people becomes larger than some pre-specified number and the outbreak is considered to be uncontrollable. Each step forward through time, or time step, in the simulation will make one event transition in each of the modules present in the system. Which events happen in each module is determined by the probabilities assigned to the events, which can be augmented by the choices made by the health unit. Health unit actions are described in detail in Section 5.1.1, and event
probabilities and other parameters are discussed in Section 5.1.3. Certain events in the modules will result in the exposure of new individuals, so new modules can always enter the system, changing the size of its vector representation. As the simulation runs, it keeps track of the number of infected people who recover, are permanently disabled or die from the disease, and the number of people who resisted the disease but suffered adverse side effects from the treatment. These values can be used to analyze the behaviour of the simulation and evaluate the health unit actions that were taken.

5.1.1 Health Unit Responses

There are several actions that the Health Unit can take in response to an outbreak. The first is the *isolation* of symptomatic individuals from the general population. This allows for the infected people to be treated in a controlled environment, and it helps to contain the spread of the disease to other healthy individuals. This strategy of dealing with an outbreak is typically very effective, however, some diseases are more likely to spread in the predromal period (before the onset of symptoms), so it is necessary to consider other options. Another action that can be taking is called *contact tracing*, which is the action of identifying exposed individuals by determining who has been in close contact with individuals who have become symptomatic. This will usually include family members and co-workers. When an individual is traced as an exposed case there are two options in dealing with them. They may be placed under *targeted surveillance*, where the person goes on with their life normally but is regularly examined for signs of symptoms. This approach is far less invasive than the second option, which is to *quarantine* traced individuals. Quarantined individuals can be immediately treated for the disease, which is referred to as chemoprophylaxis. This approach is very effective in preventing spread of the disease, but it is costly and can also result in unnecessary side effects of the treatment in people who may not even be infected. The other option is to wait until people in quarantine become symptomatic before treating them. This ensures that fewer people are likely to suffer adverse drug effects but more easily allows for further spread of the disease. The purpose of this simulation is to determine the effectiveness of these strategies. In the simulation, each strategy is represented by a different set of event probabilities in the model of an individual.
5.1.2 Model of an Exposed Individual

The FSA representation of an exposed individual is shown in Figure 5.1 (taken from [7]). Events shown with a dotted line arrow are those that occur probabilistically, meaning that under some control strategy their occurrences are associated with probabilities other than 0 or 1. Events with a star (⋆) next to their event label are those that will result in the exposure of new individuals. The number of new exposures is determined by a random number generated within a Poisson distribution; this is discussed more in Section 5.1.3.

Each individual starts off as being exposed in state 1. The person can either resist the disease or become infected, which occurs with a probability specified in the simulation parameters. If the person is infected they will transition to state 2, at which point it is decided if they have been located by health officials through the tracing of an already infected individual’s contacts or not. If it was decided to contact trace, the person will transition to state 3 with a high probability, but there is still a small chance that they are not found using contact tracing, which transitions them to state 6 with a low probability. If it is decided not to trace infected individual’s contacts then they will transition to state 6. From here they probabilistically either show symptoms (state 7) or do not (state 24). From state 7 the choice is made whether to isolate the individual (state 11) or not (state 8) which results in different numbers of new exposures. Either way the person is now treated which places them in state 5. Once a person’s contacts have been traced in state 3 the choice can be made to quarantine them (state 4) or put them under surveillance (state 9). Under targeted surveillance, if the person becomes symptomatic (state 22) then they will be isolated (state 11) and then treated (state 5). If they don’t become symptomatic within the disease’s onset timeframe then they will transition to state 24. From states 5 and 24 the infected individual can either recover (state 27), become disabled by the disease (state 28), or die (state 29).

From state 1, if the individual resists the disease they will transition to state 12. It is unknown to the health officials that the person has resisted so many of the same actions will be taken, however, none of the events in the resisted half of the model will result in new exposures. If the person has been found using contact tracing (state 13), they can be placed in quarantine (state 14) or under targeted surveillance (state 19). Quarantined individuals can be treated immediately (state 15) or given a delayed treatment (state 20). These individuals are not very likely to exhibit symptoms of a disease they don’t have, but there is still a slim possibility that they can be considered symptomatic. Symptomatic individuals not already in quarantine can be
Figure 5.1: FSA representation of an exposed individual [7]
isolated (state 21) and given a treatment (state 15), or they can be treated but not isolated (state 18). If an individual does not become symptomatic then they are in the clear and transition to state 25. People who have been treated will transition through state 15 which means that they are probabilistically susceptible to suffering adverse side effects (state 26).

Modules that are in a marked state (states 25 to 29) will be removed from the system.

### 5.1.3 Simulation Parameters

The behaviour of the simulation is governed by the parameters that are being used. There are two types of parameters which are associated with events in the model, the probability that the event will occur ($\xi$), and the average number of people that become exposed as a result of the event ($\lambda$). The specific number of exposed is determined each time by a random number drawn from a Poisson distribution with mean $\lambda$. The probability parameters are explained below:

- $\xi_I$ - Infection rate.
  Determines the probability of an exposed individual becoming infected. This parameter is disease specific but can still be affected by circumstances, such as general health level of the affected populace and vaccination rate. This is associated with event infected from state 1.

- $\xi_c$ - Chance of contact tracing.
  Determines the probability that a person is located as an exposed case through contact tracing. This is associated with event ctrace from state 2.

- $\xi_{iso}$ - Possibility of isolation.
  Determines the probability that a person can be isolated. This is associated with event isolate from states 7 and 17.

- $\xi_s^i$ - Probability of infected showing symptoms.
  This is usually very likely. This is associated with event sympto from states 6, 9 and 10.

- $\xi_s^u$ - Probability of uninfected showing symptoms.
  This is usually very unlikely because it is purely coincidental. This is associated with event sympto from states 16, 19 and 20.

- $\xi_d^i$ - Probability of infected dying after treatment.
  This is associated with event die from state 5.
\( \xi_p^t \) - Probability of infected being permanently disabled after treatment.
This is associated with event \textit{disa} from state 5.

\( \xi_d^n \) - Probability of infected dying without treatment.
This is associated with event \textit{die} from state 24.

\( \xi_p^n \) - Probability of infected being permanently disabled without treatment.
This is associated with event \textit{disa} from state 24.

\( \xi_a \) - Probability of adverse drug reactions for uninfected individuals.
This is associated with event \textit{seffects} from state 15.

The exposure parameters determine the mean of the Poisson distribution from which the number of new exposed individuals is drawn. These parameters are explained below:

\( \lambda_{inf} \) - Average number of individuals exposed when a person becomes infected.
This is associated with event \textit{infected} from state 1.

\( \lambda_{ncsym} \) - Average number of people exposed when a person who has not been identified via contact tracing becomes symptomatic.
This is associated with event \textit{sympto} from state 6.

\( \lambda_{sym} \) - Average number of people exposed when a person under targeted surveillance becomes symptomatic.
This is associated with event \textit{sympto} from state 9.

\( \lambda_q \) - Average number of people exposed when a person who has been identified via contact tracing is placed in quarantine.
This is associated with event \textit{quarantine} from state 3.

\( \lambda_{qsym} \) - Average number of people exposed when an untreated person in quarantine becomes symptomatic.
This is associated with event \textit{sympto} from state 10.

\( \lambda_{nsym} \) - Average number of people exposed when an untreated person in quarantine does not become symptomatic.
This is associated with event \textit{no sympto} from state 10.

\( \lambda_{nsym}^{nsym} \) - Average number of people exposed when a person who has not been quarantined does not become symptomatic.
This is associated with event \textit{no sympto} from state 6, and event \textit{no sympto} from state 9.
\[ \xi_I = 0.185 \quad \xi_c = 0.9 \quad \xi_{iso} = 0.99 \quad \xi^t = 0.9999 \]
\[ \xi^u = 0.0001 \quad \xi^t = 0.2 \quad \xi^t = 0.02 \quad \xi^t = 0.2 \]
\[ \xi^u = 0.02 \quad \xi^u = 0.0001 \]

Table 5.1: Assumed parameters for first simulation (smallpox)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda_{inf} )</td>
<td>1</td>
</tr>
<tr>
<td>( \lambda_{sym} )</td>
<td>4</td>
</tr>
<tr>
<td>( \lambda^s )</td>
<td>3</td>
</tr>
<tr>
<td>( \lambda_q )</td>
<td>1</td>
</tr>
<tr>
<td>( \lambda_{sym}^q )</td>
<td>0</td>
</tr>
<tr>
<td>( \lambda_{nsym}^q )</td>
<td>0</td>
</tr>
<tr>
<td>( \lambda_{nSYM}^q )</td>
<td>1</td>
</tr>
<tr>
<td>( \lambda_{niso} )</td>
<td>16</td>
</tr>
</tbody>
</table>

\( \lambda_{niso} \) - Average number of people exposed when a symptomatic person who has not been identified via contact tracing is not isolated.

This is associated with event not iso from state 7.

Brunschen used two sets of parameter values to simulate two different outbreak scenarios.

**Scenario One - Smallpox**

The first scenario models a disease which is similar to smallpox. The parameters used are shown in Table 5.1. Smallpox is caused by the Variola virus, either Variola major or Variola minor, both of which are highly transmittable. The parameters used in this scenario are modeled after the more extreme case of Variola major, which has a mortality rate of between 3 and 30 percent. Smallpox has an incubation period of 7 to 17 days during which time the disease is not contagious. Symptoms usually start appearing 12 to 14 days after contraction and include fever, muscle pain, headache, general malaise, nausea and vomiting. These prodromal symptoms typically last about 2-4 days before the characteristic rash starts to appear, complete with small lesions called enanthem. These lesions will enlarge into pustules which release the virus into the saliva, and can last 16-20 days before scabbing and falling off. There is an efficient vaccine for smallpox but there is no cure once the disease has been contracted. Vaccination within 3 days of exposure can prevent or significantly reduce the severity of the disease. Up to 7 days after exposure, vaccination can still have a positive effect on the severity of symptoms. Beyond this only supportive treatments are available, such as fluid therapy, wound treatment, infection control and medication to relieve fever and discomfort. Males can suffer permanent infertility from smallpox and there is a chance of loss of eyesight due to corneal ulcerations.

The simulation of this scenario is initialized with 15 symptomatic individuals and 75 who have just been exposed. Luckily this outbreak was detected at an early stage.
Scenario Two - Influenza

The second scenario models a disease which is similar to influenza. The parameters used are shown in Table 5.2. There are many variations of influenza; serious cases include Spanish flu (1918-1919), Asian flu (1957-1958) and Hong Kong flu (1968-1969), while less severe variations will often spread as seasonal epidemics. Symptoms of influenza can start suddenly one or two days after contraction and usually include fever, cough, nasal congestion, general fatigue, headaches, body aches and muscle pains. Typically in the more common cases the mortality rate is only high in the very young and the very old, so vaccination is recommended for these high risk groups. The best treatment usually involves being well rested and hydrated, and possibly medication to relieve fever and aches and pains. On average the mortality rate and the chance of suffering permanent physical disablement is low.

The simulation of this scenario is initialized with 15 symptomatic individuals and 200 who have just been exposed. This outbreak was detected at a fairly early stage.

5.1.4 Simulation Results

Brunsch had no control algorithm for this simulation (because none were applicable) so he obtained results from it by running a series of simulations. Both scenarios were simulated 10000 times for each control strategy available to the health unit. In order to analyze the results of each simulation, the numbers of specific occurrences were recorded. These include the number of individuals who:

- were exposed during the outbreak
- got infected
- recovered from the disease
- died from the disease

Table 5.2: Assumed parameters for second simulation (influenza)
· suffered from long-term effects of the disease
· suffered side effects from the treatment
· were isolated
· received treatment
· were identified as a contact of a confirmed case
· were put under targeted surveillance
· were quarantined

These figures were recorded and used to evaluate the different health unit strategies. Brunsch’s original simulations were replicated here using a Java 1.6.0 implementation.

Scenario One: Smallpox

The parameters used in this simulation are given in Table 5.1. Results from the 10000 simulations for each strategy are shown in Table 5.3, where ISO is the isolation strategy, SURV is the contact tracing and targeted surveillance strategy, QUARA1 is quarantining with delayed treatment and QUARA2 is quarantining with immediate treatment. For each strategy the table shows the average, minimum and maximum number of exposed; the average, minimum and maximum number of infected; the average number of people who recovered, were killed or suffered permanent disablement from the disease; the average number of people who suffered side effects from the treatment; and the average number of people who were isolated, treated, found by contact tracing, placed under targeted surveillance, and quarantined. As Table 5.3 shows, the number of exposed decreases significantly from ISO to SURV and from SURV to QUARA1. However, not much difference is observed between QUARA1 and QUARA2. This is explained by the fact that $\lambda_{sym}^q$ is set to 0, which means that as people in quarantine become symptomatic they do not expose other people. This is in accordance with the characteristics of smallpox because early signs of symptoms will appear before the infected person becomes contagious with the onset of the rash. This result indicates that in the case of smallpox and other diseases with a low $\lambda_{sym}^q$ it is not beneficial to immediately treat people who have been quarantined. It is also notable how wide the variation is between outcomes of different simulations with identical initial configurations, for instance the number of people exposed under the isolation strategy varies from 276 to 33120.
CHAPTER 5. OUTBREAK SIMULATION AND RESULTS

<table>
<thead>
<tr>
<th></th>
<th>ISO</th>
<th>SURV</th>
<th>QAURA1</th>
<th>QUARA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>exposed</td>
<td>2984.2908</td>
<td>507.2568</td>
<td>165.9561</td>
<td>165.8171</td>
</tr>
<tr>
<td>min/max</td>
<td>276/33120</td>
<td>162/1720</td>
<td>91/353</td>
<td>95/345</td>
</tr>
<tr>
<td>infected</td>
<td>566.784</td>
<td>108.7041</td>
<td>45.7348</td>
<td>45.6992</td>
</tr>
<tr>
<td>min/max</td>
<td>47/6285</td>
<td>28/376</td>
<td>20/96</td>
<td>22/93</td>
</tr>
<tr>
<td>recovered</td>
<td>441.8689</td>
<td>84.8798</td>
<td>35.685</td>
<td>35.6826</td>
</tr>
<tr>
<td>killed</td>
<td>113.6249</td>
<td>21.6403</td>
<td>9.1445</td>
<td>9.1192</td>
</tr>
<tr>
<td>disabled</td>
<td>11.2902</td>
<td>2.184</td>
<td>0.9053</td>
<td>0.8974</td>
</tr>
<tr>
<td>side effects</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.0122</td>
</tr>
<tr>
<td>isolate</td>
<td>561.3107</td>
<td>108.6271</td>
<td>4.5289</td>
<td>4.5435</td>
</tr>
<tr>
<td>treat</td>
<td>561.3107</td>
<td>108.734</td>
<td>45.7443</td>
<td>167.3128</td>
</tr>
<tr>
<td>ctraced</td>
<td>0</td>
<td>942.9288</td>
<td>162.8656</td>
<td>162.72</td>
</tr>
<tr>
<td>surv</td>
<td>0</td>
<td>470.154</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>quara</td>
<td>0</td>
<td>0</td>
<td>162.8656</td>
<td>162.72</td>
</tr>
</tbody>
</table>

Table 5.3: Simulation results for scenario one (smallpox)

The isolation and surveillance strategies have a large variance in the number of infected, which is a result of the high \( \lambda \) values associated with them (\( \lambda_{nc}^{sym} = 4, \lambda_{sym}^{sym} = 3 \)). The number of new exposed individuals is taken from a Poisson distribution, and the variance of this distribution grows with its mean.

**Scenario Two: Influenza**

The parameters used in this simulation are given in Table 5.2. As in the first scenario each strategy was simulated 10000 times. For this scenario the isolation strategy resulted in an uncontrollable outbreak every time so results are not shown. The targeted surveillance strategy resulted in 7823 uncontrollable cases, and the remaining 2177 cases were only contained because some lucky occurrences early on ensured that the outbreak did not expand much beyond the initial number of exposures. The isolation and targeted surveillance strategies are not suitable for containing the outbreak in this scenario. The results for the two quarantine strategies are shown in Table 5.4, which is presented in the same way as Table 5.3.

As expected for a simulation with \( \lambda_{sym}^{sym} > 0 \), immediately treating the quarantined individuals will result in fewer exposures and infected cases than the delayed treatment strategy. However, with delayed treatment there are far fewer treatments
administered (74.9608 instead of 894.1958) which can cut down on health care costs significantly. Another advantage of delayed treatment is that with fewer treatments there are fewer people who suffer adverse side effects (0.0011 instead of 8.3335).

In accordance with Brunsch’s findings [8], the number of people infected by the disease over many simulations follows a log-normal distribution. Figure 5.2 shows the probability distribution of people infected over 10000 simulations using the quarantine strategy with immediate treatment for scenario two.

The simplicity of the simulation parameters allows it to be used to model different disease scenarios easily. It also allows for single parameters to be adjusted within a given scenario to explore their effect on the outcome. Brunsch ran additional simulations exploring the impact of adjusting $\xi_c$, the probability that an individual has been identified through contact tracing. Originally the scenario was run with $\xi_c = 0.99$, but it is likely that in a real-world scenario this value could be much lower. Simulations were run for the successful strategies with $\xi_c = 0.9$ and $\xi_c = 0.8$, and replicated results are shown below.

As can be seen, under the delayed treatment option a 9% decrease in contact tracing efficiency results in a 46% increase in the number of infected, and a 19% decrease in $\xi_c$ results in a 186% increase in the number of infected.
CHAPTER 5. OUTBREAK SIMULATION AND RESULTS

Figure 5.2: Distribution of infected individuals over 10000 runs using quarantine strategy with immediate treatment for scenario two

<table>
<thead>
<tr>
<th>QUARA1</th>
<th>$\xi_c = 0.99$</th>
<th>$\xi_c = 0.9$</th>
<th>$\xi_c = 0.8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>exposed</td>
<td>1198.9774</td>
<td>1880.8411</td>
<td>3980.2519</td>
</tr>
<tr>
<td>min/max</td>
<td>248/7675</td>
<td>235/12266</td>
<td>294/43163</td>
</tr>
<tr>
<td>infected</td>
<td>74.9265</td>
<td>109.0687</td>
<td>213.983</td>
</tr>
<tr>
<td>min/max</td>
<td>17/475</td>
<td>15/701</td>
<td>17/2279</td>
</tr>
<tr>
<td>recovered</td>
<td>74.7755</td>
<td>108.8514</td>
<td>213.5495</td>
</tr>
<tr>
<td>killed</td>
<td>0.0733</td>
<td>0.1099</td>
<td>0.2171</td>
</tr>
<tr>
<td>disabled</td>
<td>0.0777</td>
<td>0.1074</td>
<td>0.2164</td>
</tr>
<tr>
<td>side effects</td>
<td>0.0011</td>
<td>0.0018</td>
<td>0.0046</td>
</tr>
<tr>
<td>isolate</td>
<td>0.7485</td>
<td>10.8442</td>
<td>42.3236</td>
</tr>
<tr>
<td>treat</td>
<td>74.9608</td>
<td>109.1494</td>
<td>214.1562</td>
</tr>
<tr>
<td>ctraced</td>
<td>1201.8323</td>
<td>1706.0981</td>
<td>3196.921</td>
</tr>
<tr>
<td>surv</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>quara</td>
<td>1201.8323</td>
<td>1706.0981</td>
<td>3196.921</td>
</tr>
</tbody>
</table>

Table 5.5: Simulation results for scenario two comparing $\xi_c = 0.99$, $\xi_c = 0.9$ and $\xi_c = 0.8$ using the quarantine strategy with delayed treatment
CHAPTER 5. OUTBREAK SIMULATION AND RESULTS

<table>
<thead>
<tr>
<th>QUARA2</th>
<th>$\xi_c = 0.99$</th>
<th>$\xi_c = 0.9$</th>
<th>$\xi_c = 0.8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>exposed</td>
<td>887.6313</td>
<td>1313.5426</td>
<td>2400.288</td>
</tr>
<tr>
<td>min/max</td>
<td>248/4784</td>
<td>234/8448</td>
<td>270/22027</td>
</tr>
<tr>
<td>infected</td>
<td>59.2823</td>
<td>80.5442</td>
<td>134.9676</td>
</tr>
<tr>
<td>min/max</td>
<td>17/313</td>
<td>16/517</td>
<td>17/1193</td>
</tr>
<tr>
<td>recovered</td>
<td>59.1654</td>
<td>80.3764</td>
<td>134.6994</td>
</tr>
<tr>
<td>kill</td>
<td>0.0599</td>
<td>0.0852</td>
<td>0.1354</td>
</tr>
<tr>
<td>disabled</td>
<td>0.057</td>
<td>0.0826</td>
<td>0.1328</td>
</tr>
<tr>
<td>side effects</td>
<td>8.3335</td>
<td>11.2093</td>
<td>18.2488</td>
</tr>
<tr>
<td>isolate</td>
<td>0.5865</td>
<td>7.9554</td>
<td>26.7613</td>
</tr>
<tr>
<td>treat</td>
<td>894.1958</td>
<td>1203.7651</td>
<td>1959.2028</td>
</tr>
<tr>
<td>ctraced</td>
<td>893.6031</td>
<td>1195.728</td>
<td>1932.1775</td>
</tr>
<tr>
<td>surv</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>quar2</td>
<td>893.6031</td>
<td>1195.728</td>
<td>1932.1775</td>
</tr>
</tbody>
</table>

Table 5.6: Simulation results for scenario two comparing $\xi_c = 0.99$, $\xi_c = 0.9$ and $\xi_c = 0.8$ using the quarantine strategy with immediate treatment

With the immediate treatment option, a 9% decrease in $\xi_c$ results in a 36% increase in the number of infected, and a 19% decrease in $\xi_c$ yields a 128% increase in the number of infected. These results illustrate how important it is to conduct efficient contact tracing during an outbreak, and how sensitive the behaviour of the simulation is to small changes in key parameters. They also show that the immediate treatment strategy is less affected by a decrease in contact tracing efficiency, indicating that it is a more robust strategy.

5.2 RUBLL Implementation

In this section the RUBLL supervisory algorithm will be applied to Brunsch’s outbreak simulation. The issue that must be addressed is that RUBLL is designed to work on a model of a problem that is represented by a single FSA. The outbreak simulation is made up of separate modules that cannot be synchronously composed in any way that will maintain its operational behaviour. This means that RUBLL will have to be modified to analyze multiple lookahead trees corresponding to multiple modules. The spirit of RUBLL analysis can be maintained by applying it to each tree and summing the results to determine the favourable control choice for all of
the modules. Moreover, the longest path through any modules is seven steps, which means that after seven lookahead steps none of the current modules will still be in the system. The analysis needs to take future modules into account in order to capture the long-term behaviour of the simulation. Perhaps the best choice for the exposed individuals currently in the system is not the best choice when you consider the effects on all the people who will shortly become exposed. In order to analyze the predicted modules, the number of expected new exposures at each lookahead step must be kept track of. This expected number is equal to the mean of the Poisson distribution that the number of exposures is sampled from. All the predicted modules at a certain time step can be analyzed by looking at a single representative tree and multiplying its utility by the number of predicted modules. This approach can avoid an enormous number of redundant calculations. In fact, because the longest path through a module is seven steps, we know that all of the current modules will no longer be in play after at most seven steps. At this point, the analysis will only need to consider the predicted trees, and there can only be seven of these for the same reason. So after at most seven steps, the analysis will only have to consider at most seven modules.

![Figure 5.3: Lookahead window for adapted RUBLL implementation](image)
The modified analysis framework is detailed in Figure 5.3. The figure shows the lookahead window with the depth listed along the bottom. We adopt the term lookahead window here because since we are dealing with multiple modules at a time, it is not accurate to refer to their projected behaviour as a single lookahead tree. Each FSA block in the window represents a module in the system, where the blocks at the top represent current modules that are already in the system. Beyond the current modules, one module is introduced at each step in the window; these are the expected modules which each represent all of the new exposures entering the system at that time. The number of exposures at each step will affect the number of exposures at every subsequent step, so the utility of the window must be calculated from the current state in a forward manner. First, the utility of each current module \(U_1 \ldots U_m\) must be calculated while keeping track of the expected number of exposures resulting from each module \(l_i\) at each step \(i\), \(\lambda_i\). Next, the utility of the first expected module can be calculated. This is done by calculating the utility of the single module \(u_1\) and multiplying it by the number of exposures that the module represents, \(\Lambda_1 = \lambda_1 + \cdots + \lambda_1\). During this process, the number of exposures caused by each expected module \(n\) must also be multiplied by \(\Lambda_n\) and kept track of with \(\lambda^*_n\). This process continues until every module has been evaluated. Once the utility values of the current modules \((U_m)\) and the expected modules \((V_n)\) have been calculated up to the horizon of the lookahead window, they can be added together to calculate \(U_W\), the window utility for the employed control strategy. This task must be performed once for each control pattern; the one with the greatest window utility should be enforced. The pseudocode for this algorithm is shown below.
RUBLL Outbreak Simulation Implementation Pseudocode

MAIN:
1. Construct lookahead window \( W \) to depth \( N \)
2. Compute \( \text{BESTU}(depth) \) for current state \( x \) (as detailed in Algorithm 5.2.1)
3. Enforce \( \gamma_i \) with \( U_W = \text{BESTU}(depth) \)
4. Allow system to execute \( N \) steps probabilistically and return to 1

Algorithm 5.2.1: \( \text{BESTU}(depth) \)

1. Initialize array \( U[|\gamma|][|m|] \)
   \( \triangleright \) \( U \) stores utilities of current modules for each control pattern
2. Initialize array \( L[|\gamma|][\text{WindowSize}] \)
   \( \triangleright \) \( L \) stores expected number of exposed at each step (\( \Lambda \)) for each control pattern
3. Initialize array \( LU[|\gamma|][\text{WindowSize}] \)
   \( \triangleright \) \( LU \) stores utility for expected modules at each step for each control pattern
4. Initialize array \( \text{windowUtility}[|\gamma|] \)
   \( \triangleright \) \( \text{windowUtility} \) stores utility of entire lookahead window for each control pattern

for each module \( m \in M \)

\[
\begin{align*}
x & \leftarrow m.\text{currentState} \\
\text{for each} \; \gamma \in \Gamma \\
\quad \text{do} \; \begin{cases} 
U[|\gamma|][m] = \text{CALCULATEU}(\gamma, x, \text{WindowSize}) \\
\text{UPDATEL}(\gamma, x, \text{WindowSize}, 1, 1) 
\end{cases}
\end{align*}
\]

for each lookahead depth \( w \)

\[
\begin{align*}
\text{for each} \; \gamma \in \Gamma \\
\quad \text{do} \; \begin{cases} 
\text{calculate utility of expected modules:} \\
LU[|\gamma|][w] = L[|\gamma|][w] \cdot \text{CALCULATEU}(\gamma, x_0, \text{WindowSize} - w) \\
\text{UPDATEL}(\gamma, x_0, \text{WindowSize} - w, 1, L[|\gamma|][w]) 
\end{cases}
\end{align*}
\]

for each control pattern \( \gamma \in \Gamma \)

\[
\begin{align*}
\text{do} \; \begin{cases} 
\text{windowUtility}[\gamma] = \sum_{i=0}^{|m|} U[|\gamma|][i] + \sum_{j=0}^{\text{WindowSize}} LU[|\gamma|][j] 
\end{cases}
\end{align*}
\]

return (maximum \( \text{windowUtility}[\gamma] \) for all \( \gamma \))
Algorithm 5.2.2: calculateU(γi, x, depth)

if depth = 0
    then return (0)
U = Bi(ξ(x, σ))
for each outgoing event σ ∈ Σ
    do \{ U = U + Pi(x, σ) · (calculateU(γi, ξ(x, σ), depth − 1) + Be(σ)) \}
return (U)

Notice that at its core this implementation is very similar to B-RUBLL, Algorithm 3.4.1. Note that no redundant calculations are carried out here because the control pattern is being applied globally, as in the B-RUBLL algorithm. Implementing an algorithm similar to RUBLL 3.5.1 in which a new control decision is made at each state is not possible for this problem. This is because the utility values are calculated by back propagation while the number of new exposures must be calculated in a forward manner. Utility values are affected by the number of exposures, and in turn the number of exposures is dependent on the strategy being enforced which is defined by utility values. In order to get the functionality of RUBLL 3.5.1 as applied to the outbreak scenario, every permutation of control decisions over the lookahead window would need to be evaluated, which is computationally infeasible at useful depths. Luckily for the outbreak scenario problem it makes sense to analyze with control patterns that are applied uniformly over entire lookahead windows because in reality it will take some time to implement a change in plans of action.

Algorithm 5.2.3: updateL(γi, x, depth, probability, multiplier)

if depth > 0
    then for each outgoing event σ ∈ Σ
        do \{ L[γ][WindowSize − depth] = L[γ][WindowSize − depth] + multiplier · probability · Pi(x, σ) · λ(σ) \}
        do \{ updateL(γi, ξ(x, σ), depth − 1, Pi(x, σ), multiplier) \}

The updateL function (Algorithm 5.2.3) is in charge of updating the L array with the correct number of expected modules at each step. The expected number of exposed individuals associated with event σ is denoted λ(σ). The probability variable is used to keep track of the cumulative probability of the event string leading from the current state
of the module to the future event that is associated with new exposures. The \textit{multiplier}
variable is used to keep track of how many modules are represented by this lookahead tree.

The RUBLL algorithm presented here has been adapted to an atypical environment. It
is designed to run efficiently in a computationally intensive scenario and the control scheme
matches the real-world constraints of the problem. The only thing left to do is define the
benefit values for the RUBLL framework to operate with.

5.2.1 Benefit Values

Benefit values define the relative worth of an action to the RUBLL controller. A greater
benefit value is preferable; costs are represented as negative benefits. All the benefit values
used in the outbreak simulation are negative; an infectious disease outbreak is not a very
positive situation to be in. In a scenario where the goal is to maximize profit there will
be more positive utility values, but in an outbreak scenario the goal is to minimize losses.
There are three basic rules to follow when determining what benefit values to use:

1. Generally use positive values to reward successes and negative values to punish fail-
   ures. Magnitudes of values should correlate to magnitudes of successes and failures.

2. Ensure that any rewards for recovering from a problem state are less substantial
   than the punishment for not avoiding it. This will avoid the risk of a controller that
   purposefully causes problems in order to reap the rewards for fixing them.

3. Do not attach values to occurrences that the controller has no control over. This will
   just introduce noise into the utility values.

Following these rules will generally result in benefit values which support preferable
control choices, but it is important to take the subtleties of the specific problem into account.

Occurrences associated with performing health unit actions (such as \textit{ctrace} and \textit{quarantine})
will have negative benefit values relative to the costs of the actions, while occurrences
with adverse consequences (\textit{s effects}) are associated with arbitrarily chosen negative benefits.
Employed benefit values are shown below and were obtained through consultations with
Adrienne Hansen-Taugher, Emergency Preparedness Coordinator for KFL&A (Kingston,

- Attempting to contact trace or isolate = $-1$
  
  \[ B_{c}(1, 16) = (0, 0, -1, -1, -1), \quad B_{c}(3, 18) = (0, -1, -1, -1, -1) \]

- Successfully contact tracing \((B_{c}(\textit{ctrace})) = -1\)

- Quarantining \((B_{c}(\text{s eeffects})) = -5\)

- Targeted surveillance \((B_{c}(\text{s urv})) = -4\)
Administering treatment \((B_e(treat)) = -1\)

Successfully Isolating \((B_e(isolate)) = -6\)

Individual suffering from side effects \((B_e(seffects)) = -2\)

Attempting to locate an individual via contact tracing has a cost of 1, and actually locating them has an additional cost of 1. If the person has not been contact traced then there is a chance they cannot be isolated; attempting to isolate has a cost of 1, while the successful act of isolation costs an additional 6. All costs are associated with events (i.e., they are defined by \(B_e\)) except for the cost of attempting to contact trace, which is associated with the strategies employing contact tracing (defined by \(B_c\)).

Intuition suggests that an event such as \(recover\) should have a positive benefit value associated with it, but this would be in violation of Rule 2 because there is no negative value assigned to new exposures (beyond the costs associated with dealing with exposed individuals). Assigning a positive value to \(recover\) would be detrimental to control decisions because the controller would try to expose as many people as possible to get the reward when they recover, which is the most probable outcome. To avoid this problem the positive value for \(recover\) would have to be smaller than the lowest-cost path for an exposed individual, and under the less intensive strategies this cost can be zero.

Intuition also suggests that events \(die\) and \(disa\) should have negative benefit values, but this would be in violation of Rule 3 because the chosen strategy does not affect the probabilities associated with these events. In fact, \(recover\) falls under this category as well. An easy way to see if event probabilities are affected by control decisions is to trace the probabilities leading to the event in question under each strategy and see if they are equal. The probability traces for \(recover\) are shown below for each strategy:

**No Strategy**

\[
P(recover) = (P(0, infected) \cdot P(1, not ctrace) \cdot P(2, sympto) \cdot P(3, not iso) \cdot P(4, treat) \cdot P(6, recover)) + (P(0, infected) \cdot P(1, not ctrace) \cdot P(2, no sympto) \cdot P(7, recover))
\]

\[
= \xi_I \cdot \xi_s \cdot (1 - \xi_d - \xi_p) + \xi_I \cdot (1 - \xi_d) \cdot (1 - \xi_d - \xi_p)
\]

\[
= \xi_I \cdot (1 - \xi_d - \xi_p)
\]

**Isolation Strategy**

\[
P(recover) = (P(0, infected) \cdot P(1, not ctrace) \cdot P(2, sympto) \cdot (P(3, not iso) \cdot P(4, treat) + (P(3, isolate) \cdot P(5, treat))) \cdot P(6, recover)) + (P(0, infected) \cdot P(1, not ctrace) \cdot P(2, no sympto) \cdot P(7, recover))
\]

\[
= \xi_I \cdot \xi_s \cdot (\xi_{iso} + (1 - \xi_{iso})) \cdot (1 - \xi_d - \xi_p) + \xi_I \cdot (1 - \xi_d) \cdot (1 - \xi_d - \xi_p)
\]

\[
= \xi_I \cdot (1 - \xi_d - \xi_p)
\]
Contact Tracing and Quarantine with Delayed Treatment

\[ P(\text{recover}) = P(0, \text{infected}) \cdot (P(1, \text{ctrace}) \cdot P(11, \text{surv}) \cdot (P(12, \text{sympt}) \\
\quad \cdot P(13, \text{isolate}) \cdot P(5, \text{treat}) \cdot P(6, \text{recover}) + P(12, \text{no sympto}) \\
\quad \cdot P(7, \text{recover})) + P(1, \text{not ctrace}) \cdot (P(2, \text{sympto}) \cdot (P(3, \text{isolate}) \\
\quad \cdot P(5, \text{treat}) + P(3, \text{not iso}) \cdot P(4, \text{treat})) + P(2, \text{no sympto}) \\
\quad \cdot P(7, \text{recover}) \\
\quad = \xi_t \cdot (\xi_c \cdot \xi_s^i \cdot ((1 - \xi_{iso}) + \xi_{iso}) + (1 - \xi_c) \cdot (\xi_s^i \cdot (\xi_{iso} + (1 - \xi_{iso}))) \\
\quad + (1 - \xi_s^i)) \cdot (1 - \xi_d^i - \xi_p^i) \]

Contact Tracing and Quarantine with Immediate Treatment

\[ P(\text{recover}) = P(0, \text{infected}) \cdot (P(1, \text{ctrace}) \cdot P(11, \text{quarantine}) \cdot P(14, \text{no treat}) \\
\quad \cdot (P(15, \text{sympto}) \cdot P(5, \text{treat}) \cdot P(6, \text{recover}) + P(15, \text{no sympto}) \\
\quad \cdot P(7, \text{recover})) + P(1, \text{not ctrace}) \cdot (P(2, \text{sympto}) \cdot (P(3, \text{isolate}) \\
\quad \cdot P(5, \text{treat}) + P(3, \text{not iso}) \cdot P(4, \text{treat})) + P(2, \text{no sympto}) \\
\quad \cdot P(7, \text{recover}) \\
\quad = \xi_t \cdot (\xi_c \cdot (\xi_s^i + (1 - \xi_{iso}))) + (1 - \xi_c) \cdot (\xi_s^i \cdot (\xi_{iso} + (1 - \xi_{iso}))) + (1 - \xi_s^i)) \\
\quad \cdot (1 - \xi_d^i - \xi_p^i) \]

The probability of an exposed individual recovering from the disease is \( \xi_t \cdot (1 - \xi_d^i - \xi_p^i) \) no matter what strategy is used. This is because \( \xi_d^i \) and \( \xi_p^i \) are used to determine death and disablement regardless of whether the individual has been treated or not. This may seem counterintuitive, but this simulation is more concerned with long-term behaviour of the outbreak based on the number of exposures than the effects of specific individuals being treated.
CHAPTER 5. OUTBREAK SIMULATION AND RESULTS

5.3 Offline Analysis

A number of techniques have been presented which allow the outbreak simulation to be analyzed without actually running it. These techniques include Offline RUBLL analysis, Markov chain analysis and state-space analysis. They will be applied to both outbreak scenarios in this section. At times throughout this section the control strategies are referred to as follows:

\[ \gamma_0 \] - No Strategy
\[ \gamma_1 \] - Isolation Strategy
\[ \gamma_2 \] - Contact Tracing and Targeted Surveillance
\[ \gamma_3 \] - Contact Tracing and Quarantine with Delayed Treatment
\[ \gamma_4 \] - Contact Tracing and Quarantine with Immediate Treatment.

The utility value resulting from applying strategy \( \gamma_i \) up to depth \( n \) is denoted by \( U_{\gamma_i n} \).

5.3.1 Scenario One: Smallpox

Utility Analysis

We can use the RUBLL 5.2.1 algorithm in conjunction with the benefit values defined in Section 5.2.1 to analyze the outbreak scenario from its initial state without having to run the simulation. Table 5.7 shows the utility values computed for all five control strategies up to depth 12. Results up to depth 50 are provided in Appendix A, Table A.1.

The first observation to make is that the cheaper strategies produce better utility values at lower depths. This is in accordance with the idea that spending less to control an outbreak will be cheaper in the short term, but over a long enough time it will become more expensive with the increased number of new exposures. Another observation to note is that the utility values are constantly increasing in magnitude with depth. As the event strings stretch to larger depths and more branches are investigated, the probabilities associated with each event become infinitesimally small. In a single FSA problem, these factors would balance out and the predicted expected values would converge on the actual expected values, as with the MACHINE example.

<table>
<thead>
<tr>
<th>depth</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Strategy</td>
<td>0</td>
<td>0</td>
<td>-15.00</td>
<td>-28.88</td>
<td>-31.45</td>
<td>-42.55</td>
<td>-161.07</td>
<td>-517.95</td>
<td>-1202.94</td>
<td>-4882.08</td>
<td>-14401.94</td>
<td>-46823.20</td>
</tr>
<tr>
<td>ISO</td>
<td>0</td>
<td>0</td>
<td>-104.09</td>
<td>-215.41</td>
<td>-247.11</td>
<td>-326.74</td>
<td>-855.26</td>
<td>-1598.69</td>
<td>-4008.07</td>
<td>-8651.57</td>
<td>-20637.45</td>
<td>-46823.20</td>
</tr>
<tr>
<td>SURV</td>
<td>-28.5</td>
<td>-225</td>
<td>-531.77</td>
<td>-956.06</td>
<td>-1820.17</td>
<td>-2957.90</td>
<td>-3980.58</td>
<td>-7380.67</td>
<td>-11424.18</td>
<td>-16015.54</td>
<td>-28376.40</td>
<td>-38376.40</td>
</tr>
<tr>
<td>QUARA1</td>
<td>-28.5</td>
<td>-238.5</td>
<td>-612.77</td>
<td>-723.39</td>
<td>-1051.88</td>
<td>-1673.02</td>
<td>-2166.11</td>
<td>-3014.14</td>
<td>-3769.38</td>
<td>-4952.68</td>
<td>-6070.13</td>
<td>-7729.21</td>
</tr>
<tr>
<td>QUARA2</td>
<td>-28.5</td>
<td>-238.5</td>
<td>-626.27</td>
<td>-804.39</td>
<td>-1131.88</td>
<td>-1758.08</td>
<td>-2355.00</td>
<td>-3235.41</td>
<td>-4111.80</td>
<td>-5351.81</td>
<td>-6631.94</td>
<td>-8384.11</td>
</tr>
</tbody>
</table>

Table 5.7: Utility values for depths 1 through 12 computed from the initial state of the simulation for Scenario One. Values are rounded to 2 decimal places.
in Section 3.3. This outbreak problem is not a single FSA so the utility values are subject to constant growth with the number of trees in the system. The state-space growth of the system is dependent on which strategy is being enforced, so each strategy will have its own exponential growth rate $G$. An estimate for $G$ at depth $n$ can be obtained by dividing the utility at depth $n$ by the utility at depth $n - 1$. Estimates for the growth rates at depth 50 are:

\[
\begin{align*}
\gamma_0 &\approx 3.16318448 \\
\gamma_1 &\approx 2.31603297 \\
\gamma_2 &\approx 1.55816042 \\
\gamma_3 &\approx 1.19732528 \\
\gamma_4 &\approx 1.19732528
\end{align*}
\]

The most important thing to note is which strategies are best at what depths (Appendix A, Table A.1):

- $\gamma_0$ beats $\gamma_1$ for $n < 16$
- $\gamma_1$ beats $\gamma_2$ for $n < 11$
- $\gamma_2$ beats $\gamma_3$ for $n < 6$
- $\gamma_3$ beats $\gamma_4$ for all $n$, so it is the best long-term strategy
- $\gamma_3$ is the best strategy for $n > 11$

It can be determined by looking at the utility values that $\gamma_3$ (Quarantine with Delayed Treatment) is the most beneficial long-term approach to controlling this outbreak. This is consistent with the occurrences in Table 5.3, which show that the results of using either of the quarantine strategies are almost identical, but immediate treatment results in more treatments administered and more side effects incurred. By setting all but one of the benefit values to zero, RUBLL will compute the utility that results from that particular benefit value. This technique is referred to as benefit isolation, and it can be used here to compare $\gamma_3$ and $\gamma_4$ because they have the same rate of growth. By isolating utilities resulting from just treatments and just side effects we can determine how responsible each is for the disparity between $\gamma_3$ and $\gamma_4$. The utility resulting from isolating benefit value $b_i$ using strategy $\gamma_i$ up to depth $n$ is denoted by $U_{\gamma_i}^n$.

**Benefit Isolation:** $\gamma_3$ & $\gamma_4$

\[
\begin{align*}
U_{\gamma_3}^{50} &= -20466955.9871801 \\
U_{\gamma_4}^{50} &= -22382001.3743016
\end{align*}
\]
This difference in utility values is attributable to the benefit values associated with treatments and side effects, denoted \( b_t \) and \( b_{se} \).

\[
\begin{align*}
U_{\gamma_{50}}^{b_t} &= -321550.563530 \\
U_{\gamma_{50}}^{b_t} &= -2236296.317281 \\
U_{\gamma_{50}}^{b_{se}} &= -0.023680 \\
U_{\gamma_{50}}^{b_{se}} &= -299.657051
\end{align*}
\]

With immediate treatment we observe a 595% increase in the amount spent on treatments, and a whopping 1265342% increase in the costs incurred from side effects caused by unnecessary treatment. The difference in this strategy’s utilities is attributable to just \( b_t \) and \( b_{se} \) thusly:

\[
U_{\gamma_{4n}} - U_{\gamma_{3n}} = U_{\gamma_{4n}}^{b_t} + U_{\gamma_{4n}}^{b_{se}} - U_{\gamma_{3n}}^{b_t} - U_{\gamma_{3n}}^{b_{se}}.
\]

For the other control strategies, the disparity in utilities is predominantly a result of different rates of growth, which renders simple benefit isolation less fruitful.

**Markov Chain Analysis**

Recall the Markov chain approach as it was applied in Section 3.3.2. This technique can be applied to the outbreak scenario by modifying the individual FSA (Figure 5.1) to include events for the exposure of new individuals. From any state with an event that causes new exposures a probabilistically weighted feedback arc is added which transitions back to the exposed state (state 1). This transitioning back to the exposed state mirrors the mechanic of introducing new modules of exposed individuals into the system. Including these arcs ensures that the growth behaviour of the system is represented within a single FSA model. Figure 5.4 shows where these arcs must be added to the model and the values that are associated with each. The arcs are labelled with their values in terms of the simulation parameters discussed in Section 5.1.

From Figure 5.4 it is possible to produce a transition matrix with weighted feedback arcs for any of the control strategies in either scenario. Figure 5.5 shows the transition matrix produced for scenario one using \( \gamma_3 \), and the normalized eigenvector corresponding to the largest eigenvalue.

The exposure arc from state 6 back to state 1 was rounded from 3.9997000000000003 to 3.9997, but all other matrix values are exact. Markov chain analysis can be performed on the transition matrix by calculating the largest eigenvalue \( \lambda_k \) and the corresponding normalized eigenvector \( v_k \). This matrix has a \( \lambda_k = 0.69367028 \), and \( v_k \) is shown in Figure 5.5, where values are rounded to 8 decimal places. The values in \( v_k \) correspond to the proportion of time that the system will be in each state. From
Figure 5.4: Probabilistically weighted exposure feedback arcs as they are appended to the model of an individual.

Figure 5.5: Transition matrix for quarantine strategy with delayed treatment ($\gamma_3$) with weighted exposure feedback arcs, and normalized eigenvector corresponding to the largest eigenvalue.
this it can be observed that with $\gamma_3$ the system will never enter states 9, 19, 22 or 23. In Section 3.3.2 the probability of being in each state is combined with the utility of each state to produce a global expected utility for the system. Because of the nature of the outbreak simulation, there are no depth-independent state utility values to compute an expected system utility with. However, it can still be beneficial to perform a Markov chain analysis here, particularly with regards to the $\lambda_k$ values. The dominant eigenvalue corresponds to the system’s growth rate over time, so strategies with $\lambda_k < 1$ reduce the number of modules over time and generally result in controllable outbreaks, where as strategies with $\lambda_k > 1$ increase the number of modules over time and will generally result in uncontrollable outbreaks. The dominant eigenvalues for each strategy are as follows.

- $\gamma_0: \lambda_k = 1.46679252$
- $\gamma_1: \lambda_k = 0.98259245$
- $\gamma_2: \lambda_k = 0.91928561$
- $\gamma_3: \lambda_k = 0.69367028$
- $\gamma_4: \lambda_k = 0.69367028$

These results correspond with the findings in Section 5.1.4 because the only strategy with $\lambda_k > 1$ is $\gamma_0$, and this was the only strategy that consistently produced uncontrollable outbreaks. So in order to determine if a strategy will be generally effective at controlling an outbreak, calculating $\lambda_k$ is a much more direct route than averaging over many trials.

**State-Space Analysis**

The $\tau$ analysis technique discussed in Chapter 4 is not readily applicable to this simulation. However, the technique can be adapted to work here and produce accurate state-space estimates for each control strategy. There are two different state-space values that are important in this problem; the full number of states that are represented in the projected system, and the number of states that must actually be traversed by RUBLL 5.2.1. These are referred to as *represented* state space and *traversed* state space, respectively. To determine the represented state space a special adjacency matrix must be used. This matrix (referred to as the *representative adjacency matrix*) has probabilistically weighted exposure feedback arcs (as illustrated in Figure 5.4), but otherwise only contains 0 and 1 entries to indicate state adjacency within the module. Only state adjacency will affect tree state-space within a single module, but new exposures must be weighted by their expected values in order to accurately estimate the full number of represented states.
Figure 5.6: Representative adjacency matrix for quarantine with delayed treatment strategy ($\gamma_3$).

Figure 5.6 shows the representative adjacency matrix for $\gamma_3$. We can use this matrix to obtain a $\tau$ estimate for state space by following the steps outlined in Section 4.4. This matrix, like all of the representative adjacency matrices, is not primitive because it has a large siphoning component with no growth. The siphoning components are made up of all of the states that cannot eventually reach an event that causes new exposures. The largest growth component is made up of all of the states that can reach an event that causes new exposures, which in this scenario includes states 1, 2, 3, 6, 7 and 9. This largest growth component could be analyzed to produce a $\tau^\infty$ value if it were diagonalizable, but because it is not we must obtain an estimate for the exponential growth rate of the system and use it to calculate a $\tau$ value. Luckily the exponential growth rates have already been obtained by analyzing RUBLL's utility outputs. For $\gamma_3$ the growth rate $G$ is 1.197325 (estimated at depth 50, i.e., $U_{\gamma_3}^{50}/U_{\gamma_3}^{49}$). This can be used to compute a $\tau$ value using equation (4.10):

\[
S^n_x = G^n \cdot v_{xk} \cdot \tau \\
\tau = \frac{S_1^{50}}{v_{1k} \cdot G^{50}} \\
\tau = \frac{5929992.895660}{0.289149 \cdot 8139.685074} = 2519.562187
\]

This $\tau$ value can be used to create an equation for estimating represented window
state-space for $\gamma_3$:

$$
\tilde{S}_M^n = 1.197325^n \cdot 2519.562187 \cdot \begin{bmatrix} m_1 & m_2 & m_3 & m_6 & m_7 & m_9 \end{bmatrix} \begin{bmatrix} 0.289149 \\ 0.462359 \\ 0.162076 \\ 0.662790 \\ 0.025932 \\ 0.486195 \end{bmatrix}
$$

where $m_i$ represents the number of modules currently in state $i$. Table 5.8 shows state space estimates calculated using this equation for two different initial vectors and compares them to the actual state space values. Note that for the outbreak simulation the state of the system is represented by a vector so instead of unwinding the tree from a single state $x$, it is now unwound from a vector of states $M$. The starting state for scenario one (with 75 exposed and 15 infected individuals) is $[75,15]$, and a vector containing only one exposed individual is $[1]$.

Table 5.8: Estimated and actual state spaces for Scenario One with $\gamma_3$ rounded to 2 decimal places.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>$S_m^n$</th>
<th>$S_m^n - S_m^n$</th>
<th>% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>$[1]$</td>
<td>4411.32</td>
<td>-1953.77</td>
<td>44.29</td>
</tr>
<tr>
<td></td>
<td>$[75,15]$</td>
<td>436656.40</td>
<td>-210680.02</td>
<td>48.25</td>
</tr>
<tr>
<td>20</td>
<td>$[1]$</td>
<td>26710.97</td>
<td>-1983.55</td>
<td>7.43</td>
</tr>
<tr>
<td></td>
<td>$[75,15]$</td>
<td>2643999.38</td>
<td>-409127.90</td>
<td>15.47</td>
</tr>
<tr>
<td></td>
<td>$[75,15]$</td>
<td>16009688.04</td>
<td>-1600980.68</td>
<td>10.00</td>
</tr>
<tr>
<td>50</td>
<td>$[1]$</td>
<td>5929992.90</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$[75,15]$</td>
<td>586983522.84</td>
<td>-52337224.51</td>
<td>8.92</td>
</tr>
<tr>
<td>80</td>
<td>$[1]$</td>
<td>1316493546.21</td>
<td>561003.95</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>$[75,15]$</td>
<td>130313818777.56</td>
<td>-11573272252.32</td>
<td>8.88</td>
</tr>
</tbody>
</table>

The actual state space values are not whole numbers because the predicted modules are representing the expected number of exposures which is subject to probabilistic events. Error rates for $[75,15]$ are higher than for $[1]$ because this $\tau$ estimator was constructed using $S_{50}^{[1]}$, which explains the lack of error for $\tilde{S}_{50}^{[1]}$. State space is dependent on the distribution of states in $M$, which affects $\tau$ values and makes it hard to calculate a good $\tau$ estimator for every case in this problem. Estimates that were established using a state vector similar to the initial vector of the scenario being estimated will produce the lowest error rates. Even estimates established using similar vectors here will result in higher error rates than those observed in Section 4.1.4 because the outbreak simulation is far more complex.
As Table 5.8 shows, the represented state space can grow very quickly for an initial vector with 90 states, not to mention the possibility of vectors with close to 10000 states. Luckily, RUBLL 5.2.1 does not have to traverse all of these states to calculate what it needs to make a control decision. All of the predicted modules at a certain time step are represented by a single module, which means that only 7 predicted modules will need to be explored at any given time. This means that once the current modules in the system have exhausted after 7 lookahead steps, only the state-space equivalent of a single tree must be traversed at each step. Calculation of the exact traversed state-space of a lookahead window can be broken into two parts: the current modules and the predicted modules. The state space of the current modules in the system can be calculated (similarly to equation (4.1)) from the current system vector $M$ and a simple adjacency matrix $A$ (with no feedback arcs). Recall that since the number of states in Figure 5.1 is 29, the number of states at depth $n$ in a lookahead tree unwound from state $i$ is $\sum_{j=1}^{29} a_{ij}^{(n)}$, where $a_{ij}^{(n)}$ denotes the $(i,j)$-entry in the matrix resulting from $A$ being raised to the power of $n$. Multiplying $\sum_{k=0}^{n} A^k$ by the current system vector $M$ will describe the exact number of states in the lookahead window from the current modules:

\[
\text{State Space for Current Modules} = \sum_{j=1}^{29} \left( \sum_{i=1}^{29} m_i \cdot \sum_{k=0}^{n} a_{ij}^{(k)} \right)
\]

Note that there is no point computing the current module state space for $n > 6$ because $a_{ij}^{(k)} = 0$ for all $k > 6$. The individual FSA has a tree state-space of 75, so an easy worst-case estimate for the predicted modules is $75n$. However, the traversed state space of the individual FSA is dependent on the strategy being enforced. With $\gamma_0$ the tree has 21 states, with $\gamma_1$ it has 30 states, $\gamma_2$ and $\gamma_3$ both yield 50 state trees and $\gamma_4$ yields a 41 state tree. Also, as the first predicted modules are entering the system they are not fully traversed right away. For instance, the predicted module introduced at depth 1 will only contain a single state within the window horizon. At depth 2 the same module will have 3 states and at depth 3 it will have 7, and these numbers are again dependent on the strategy being enforced. It is possible to calculate the exact predicted module state-space using $A$:

\[
\text{State Space for Predicted Modules} = \sum_{j=1}^{29} \sum_{k=0}^{n-1} \sum_{h=0}^{k} a_{ij}^{(h)}
\]

Again note that there is no point in computing past $h = 6$ as $\sum_{h=0}^{k} a_{ij}^{(h)}$ will remain constant for $h > 6$ at the number of states in the individual lookahead tree.
Table 5.9: Utility values for depths 1 through 11 computed from the initial state of the simulation for Scenario Two. Values are rounded to 2 decimal places.

for the employed strategy. Each additional depth after \( n = 7 \) will add another full lookahead tree to the state space. Adding the state space of the current modules and the predicted modules yields an equation for traversed state-space:

\[
\hat{S}_n^M = \sum_{j=1}^{29} \left( \sum_{i=1}^{29} m_i \cdot \sum_{k=0}^{n} a_{ij}^{(k)} + \sum_{h=0}^{n-1} \sum_{k=0}^{h} a_{ij}^{(h)} \right)
\]  

The usefulness of this equation will be investigated by applying it to estimate execution times of the RUBLL 5.2.1 implementation in Section 5.3.2.

5.3.2 Scenario Two: Influenza

Now the same analysis techniques will be applied to Scenario Two. This section retains the same organization as the previous one, so more detailed descriptions of each technique are given in Section 5.3.1.

Utility Analysis

Table 5.9 shows the utility values computed for all five control strategies up to depth 11. Results up to depth 50 are provided in Appendix A, Table A.2.

These utility values exhibit many of the same properties as those in Table 5.7 and A.1. An important difference here is that \( \gamma_4 \) is the best long-term strategy (Appendix A, Table A.2).

\[
\begin{align*}
\gamma_0 & \text{ beats } \gamma_1 \text{ for } n < 30 \\
\gamma_1 & \text{ beats } \gamma_2 \text{ for } n < 11 \\
\gamma_2 & \text{ beats } \gamma_3 \text{ for } n < 6 \\
\gamma_3 & \text{ beats } \gamma_4 \text{ for } n < 7 \\
\gamma_4 & \text{ is the best strategy for } n > 11
\end{align*}
\]

This is consistent with the occurrences in Table 5.4, which show that using the quarantine strategy with immediate treatment results in fewer exposures and other negative occurrences, but results in far more treatments administered and side effects
incurred. According to the benefit values implemented with RUBLL, the costs associated with treatments and side effects for $\gamma_4$ will be trumped by $\gamma_3$’s larger rate of growth in just 7 lookahead steps from the initial state of the scenario. In fact, no matter how high the cost associated with side effects, immediate treatment will always be preferable at large enough depths. After looking a sufficient depth ahead, the costs associated with the additional exposures will trump the savings resulting from avoiding side effects. This is because the costs increase exponentially with depth while the savings increase linearly. Benefit isolation can be used here to account for the discrepancy between $\gamma_3$ and $\gamma_4$, but it must be performed at a specific depth of interest because their rates of growth are not equal. It makes sense to compare them at depths 6 and 7 because $\gamma_4$ surpasses $\gamma_3$ at depth 7.

**Benefit Isolation: $\gamma_3$ & $\gamma_4$**

Depth 6:

$U^{b_t}_{\gamma_46} - U^{b_t}_{\gamma_36} = -85.0466925$

$U^{b_{se}}_{\gamma_46} - U^{b_{se}}_{\gamma_36} = -0.01303796$

$U^{b_{a,b,c,b_q,b_{ts},b_i}}_{\gamma_46} - U^{b_{a,b,c,b_q,b_{ts},b_i}}_{\gamma_36} = 29.85$

At depth 6 we can expect that $\gamma_4$ will produce 85.0466925 more in the cost of treatments and 0.01303796 more in the cost of side effects than $\gamma_3$. However, using $\gamma_4$ we are already expecting 29.85 less spent on the cost of everything else, which is attributable to the larger growth rate of $\gamma_3$.

Depth 7:

$U^{b_t}_{\gamma_47} - U^{b_t}_{\gamma_37} = -188.87685266$

$U^{b_{se}}_{\gamma_47} - U^{b_{se}}_{\gamma_37} = -0.01589744$

$U^{b_{a,b,c,b_q,b_{ts},b_i}}_{\gamma_47} - U^{b_{a,b,c,b_q,b_{ts},b_i}}_{\gamma_37} = 780.7$

At depth 7, $\gamma_4$ is expecting 188.87685266 more in the cost of treatments and 0.01589744 more in the cost of side effects, but enjoys a savings of 780.7 with regards to other costs. As these strategies continue to greater depths it is easy to see how $\gamma_4$ is the best long-term strategy. Estimates for the growth rates at depth 50 are:

$\gamma_0 = 4.77235295$

$\gamma_1 = 4.10166655$

$\gamma_2 = 2.00769180$

$\gamma_3 = 1.37394618$

$\gamma_4 = 1.18056026$
### 5. OUTBREAK SIMULATION AND RESULTS

**Table 5.10:** Utility values for depths 1 through 10 computed from the initial state of the simulation and growth rates for Scenario Two with $\xi_c$ set to 0.99, 0.9 and 0.8. Utility values are rounded to 2 decimal places and growth rates are rounded to 8.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Depth</th>
<th>Utility Values</th>
<th>Growth Rates</th>
</tr>
</thead>
<tbody>
<tr>
<td>SURA</td>
<td>0.99</td>
<td>-29.85</td>
<td>-2.00769180</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>-28.5</td>
<td>-2.22957628</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>-27</td>
<td>-2.47209361</td>
</tr>
<tr>
<td>QUARA1</td>
<td>0.99</td>
<td>-29.85</td>
<td>-1.37394618</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>-28.5</td>
<td>-1.71801273</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>-27</td>
<td>-2.05815293</td>
</tr>
<tr>
<td>QUARA2</td>
<td>0.99</td>
<td>-29.85</td>
<td>-1.18056026</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>-28.5</td>
<td>-1.61000506</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>-27</td>
<td>-1.98957975</td>
</tr>
</tbody>
</table>

Parameter Adjustment: Contact Tracing

In Section 5.1.4, Scenario Two was run with three different contact tracing parameters ($\xi_c = 0.99$, $\xi_c = 0.9$ and $\xi_c = 0.8$) and their effects on simulation results were interpreted. Now we can observe the effects of these parameters on RUBLL’s utility values. Table 5.10 shows the utility values calculated for the three control strategies that use contact tracing; the utility values for using no strategy and the isolation strategy are unaffected by the contact tracing parameter.

The effect of adjusting the probability of contact tracing on utility values is dependent on the depth of lookahead. This is because changes in contact tracing efficiency affect the growth rate of the system, which correlates with the number of exposures at each step.

For the targeted surveillance strategy, a 9% decrease in contact tracing efficiency results in a 16.74% increase in exposure rates, and a 19% decrease in $\xi_c$ increases exposures by 29.44%. For the quarantine with delayed treatment strategy a 9% decrease in $\xi_c$ results in a 24.80% increase in exposure rates, and a 19% decrease in $\xi_c$ increases exposures by 29.44%. For the quarantine with immediate treatment strategy a 9% decrease in $\xi_c$ results in a 36.09% increase in exposure rates, and a 19% decrease in $\xi_c$ increases exposures by 68.18%. These percentages indicate how many more exposures will be observed at each step in the simulation as a result of adjusting $\xi_c$. An interesting result here is that in Section 5.1.4 it was found that adjusting $\xi_c$ had more of an effect on the number of infected with $\gamma_3$ than with $\gamma_4$. Here we observe that the rate of exposure is more affected by adjusting $\xi_c$ for the long-term strategies like $\gamma_4$. However, this is not a contradictory finding because an increased rate of exposure will not always produce more infected over the course of an outbreak. It depends on how successful the strategy is at dealing with these exposures as they come, which is best described by the dominant eigenvalue.
CHAPTER 5. OUTBREAK SIMULATION AND RESULTS

Markov Chain Analysis

Markov chain analysis is performed here as it was for the Scenario One, using transition matrices with weighted exposure feedback arcs as defined in Figure 5.4. The dominant eigenvalues for each strategy are as follows.

\[
\begin{align*}
\gamma_0 &: \lambda_k = 1.49021442 \\
\gamma_1 &: \lambda_k = 1.21550285 \\
\gamma_2 &: \lambda_k = 1.02692397 \\
\gamma_3 &: \lambda_k = 0.86488681 \\
\gamma_4 &: \lambda_k = 0.79131862
\end{align*}
\]

These are consistent with the findings for Scenario One that \( \lambda_k > 1 \) will result in an uncontrollable outbreak. Targeted surveillance has \( \lambda_k = 1.02692397 \); because this is so close to 1 it makes sense that in Section 5.1.4 \( \gamma_2 \) yielded a controllable outbreak rate of 21.77%. These controllable cases represent the trials in which the simulation had some lucky dice rolls early on in the outbreak. Markov chain analysis can also be used to investigate the effects of parameter adjustment on \( \lambda_k \). The results for \( \xi_c = 0.99, \xi_c = 0.9 \) and \( \xi_c = 0.8 \) for the three contact tracing strategies are:

Contact Tracing and Targeted Surveillance:

\[
\begin{align*}
\xi_c &= 0.99 : \lambda_k = 1.02692397 \\
\xi_c &= 0.9 : \lambda_k = 1.04537670 \\
\xi_c &= 0.8 : \lambda_k = 1.06563236
\end{align*}
\]

Contact Tracing and Quarantine with Delayed Treatment:

\[
\begin{align*}
\xi_c &= 0.99 : \lambda_k = 0.86488681 \\
\xi_c &= 0.9 : \lambda_k = 0.91151162 \\
\xi_c &= 0.8 : \lambda_k = 0.95786021
\end{align*}
\]

Contact Tracing and Quarantine with Immediate Treatment

\[
\begin{align*}
\xi_c &= 0.99 : \lambda_k = 0.79131862 \\
\xi_c &= 0.9 : \lambda_k = 0.86326155 \\
\xi_c &= 0.8 : \lambda_k = 0.92545786
\end{align*}
\]

For the targeted surveillance strategy a 9% decrease in contact tracing efficiency results in a 1.80% increase in \( \lambda_k \), and a 19% decrease in \( \xi_c \) increases \( \lambda_k \) by 3.77%. For the quarantine with delayed treatment strategy a 9% decrease in \( \xi_c \) increases \( \lambda_k \) by 5.39%, and a 19% decrease in \( \xi_c \) increases \( \lambda_k \) by 10.75%. For the quarantine with immediate treatment strategy a 9% decrease in \( \xi_c \) increases \( \lambda_k \) by 9.09%, and a 19% decrease in \( \xi_c \) increases \( \lambda_k \) by 16.95%. The \( \lambda_k \) values indicate generally how good a strategy is at dealing with exposed individuals. Even though the long-term strategies
seem to have $\lambda_k$ values that are more susceptible to adjustments in contact tracing efficiency, their $\lambda_k$ values are still lower for equal $\xi_c$ values. For instance, even though adjusting $\xi_c$ from 0.99 to 0.8 for has the greatest effect on $\gamma_4$ (exposure rate increase of 68.18%, $\lambda_k$ increase of 16.95%), it is still the best strategy in this case because its $\lambda_k$ says it is best at dealing with new exposures.

State-Space Analysis

A $\tau$ state-space estimator can be created here as it was in Scenario One, using representative adjacency matrices with weighted exposure feedback arcs as defined in Figure 5.4. The largest growth component for $\gamma_4$ in Scenario Two includes another state (1, 2, 3, 6, 7, 9 and 10) because it has $\lambda^q_{sym} = 1$ and $\lambda^q_{nsym} = 1$ where Scenario One had $\lambda^q_{sym} = 0$ and $\lambda^q_{nsym} = 0$. We will compute an estimator for $\gamma_4$ using equation (4.10), and this time it will be computed from the initial state of the system $[200,15]$ instead of a single exposed state $[1]$, as it was in Section 5.3.1.

$$S^n_M = G^n \cdot v_{mk} \cdot \tau$$

$$\tau = \frac{S^{50}_{[200,15]}}{(200v_{1k} + 15v_{2k}) \cdot G^{50}}$$

$$\tau = \frac{861538244.941339}{35.941760 \cdot 4021.683845}$$

$$\tau = 5960.288654$$

This $\tau$ value can be used to create an equation for estimating represented window state-space for the targeted surveillance strategy:

$$\tilde{S}^n_m = 1.180560^n \cdot 5960.288654 \cdot \begin{bmatrix} m_1 & m_2 & m_3 & m_6 & m_7 & m_9 & m_{10} \end{bmatrix} \begin{bmatrix} 0.155448 \\ 0.323482 \\ 0.055897 \\ 0.843700 \\ 0.016769 \\ 0.390942 \\ 0.055897 \end{bmatrix}$$

where $m_i$ represents the number of modules currently in state $i$. Table 5.11 shows state-space estimates calculated using this equation for two different vectors, [1] and [200,15]. Error rates are higher for [1] than [200,15] because this estimator was constructed using $S^{50}_{[200,15]}$ which explains the lack of error for $\tilde{S}^{50}_{[200,15]}$. 
Table 5.11: Estimated and actual state spaces for Scenario Two with $\gamma_4$ rounded to 2 decimal places.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>$S^m_n$</th>
<th>$S^m_m$</th>
<th>$S^m_m - S^m_n$</th>
<th>% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>[1]</td>
<td>4872.29</td>
<td>3175.11</td>
<td>-1697.18</td>
<td>34.83</td>
</tr>
<tr>
<td></td>
<td>[200,15]</td>
<td>1126544.16</td>
<td>669226.63</td>
<td>-457317.53</td>
<td>40.59</td>
</tr>
<tr>
<td>20</td>
<td>[1]</td>
<td>25622.09</td>
<td>26005.84</td>
<td>383.75</td>
<td>1.50</td>
</tr>
<tr>
<td></td>
<td>[200,15]</td>
<td>5924201.27</td>
<td>5472864.27</td>
<td>-451337.00</td>
<td>7.62</td>
</tr>
<tr>
<td>30</td>
<td>[1]</td>
<td>134739.87</td>
<td>145978.14</td>
<td>11238.28</td>
<td>8.34</td>
</tr>
<tr>
<td></td>
<td>[200,15]</td>
<td>31153826.04</td>
<td>30716928.49</td>
<td>-436897.55</td>
<td>1.40</td>
</tr>
<tr>
<td>50</td>
<td>[1]</td>
<td>3726140.97</td>
<td>4094435.35</td>
<td>368294.39</td>
<td>9.88</td>
</tr>
<tr>
<td></td>
<td>[200,15]</td>
<td>861538244.94</td>
<td>861538244.94</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>[1]</td>
<td>541881141.42</td>
<td>59572237.57</td>
<td>53871096.15</td>
<td>9.94</td>
</tr>
<tr>
<td></td>
<td>[200,15]</td>
<td>125290838900.60</td>
<td>125356225328.85</td>
<td>65386428.25</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Just like in Table 5.8, the estimates for vectors that are less similar to the estimator vector are less accurate. For more accurate state-space estimates, the estimator should be constructed using the initial vector of the system being estimated. For estimating the state-space at different points during the execution of the simulation, the best estimator on average will be one created using a vector in the same proportions as the dominant eigenvector of the scenario’s representative adjacency matrix. Knowing the represented state-space of the system may not be very useful in this situation because it does not correlate with RUBLL’s execution times. For this we need to look at traversed state-space, which is defined exactly by equation (4.1). This equation can be used to estimate run times of RUBLL 5.2.1. Run times throughout this example were calculated using a laptop with a 2.53GHz dual core and 6GB of RAM and an implementation favouring usability over execution speed. First we obtain an estimate of the execution time per state in the tree by unwinding from the initial vector to depth 50. Calculating $U_{\gamma,50}$ takes this implementation 518462 nanoseconds, and we know it must traverse $\hat{S}_{50}^{[200,15]} = 10426$ states. This yields a per-state execution estimate of 49.727796 ns. With this it is possible to estimate the execution time of this implementation for all depths and initial vectors. Table 5.12 shows the estimated and actual run times for three starting vectors.

The per-state execution time estimate was calculated using vector $[200,15]$ at depth 50, but then the estimates were tested against another trial which accounts for the 1.68% error in estimating the same vector to the same depth. The execution times observed from this Java implementation are somewhat sporadic, especially for lower depths before it has the chance to settle into a smoother progression. This explains the higher error rate for initial vector [1]; a system with fewer initial modules will have fewer states at equal depths, which means that the implementation will have
Table 5.12: Estimated and actual run times for Scenario Two with $\gamma_4$, rounded to 2 decimal places.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>$S_m^n$</th>
<th>actual RT (ns)</th>
<th>est. RT (ns)</th>
<th>% error</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>[200,15]</td>
<td>9606</td>
<td>492737</td>
<td>477685.21</td>
<td>3.15</td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>1102</td>
<td>68075</td>
<td>54800.03</td>
<td>24.22</td>
</tr>
<tr>
<td></td>
<td>[1000]</td>
<td>42061</td>
<td>2045100</td>
<td>2091600.82</td>
<td>2.22</td>
</tr>
<tr>
<td>50</td>
<td>[200,15]</td>
<td>10426</td>
<td>527180</td>
<td>518462</td>
<td>1.68</td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>1922</td>
<td>95224</td>
<td>95576.82</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td>[1000]</td>
<td>42881</td>
<td>2083594</td>
<td>2132377.62</td>
<td>2.29</td>
</tr>
<tr>
<td>80</td>
<td>[200,15]</td>
<td>11656</td>
<td>585935</td>
<td>579627.19</td>
<td>1.09</td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>3152</td>
<td>162084</td>
<td>156742.01</td>
<td>3.42</td>
</tr>
<tr>
<td></td>
<td>[1000]</td>
<td>44111</td>
<td>2171120</td>
<td>2193542.81</td>
<td>1.02</td>
</tr>
</tbody>
</table>

less time to settle into the expected per-state execution time.

5.4 Online Analysis

Up to this point the analysis techniques applied to the outbreak simulation have been offline in nature, meaning that at no point during the analysis does the simulation probabilistically select events in the system to transition the states of the modules. The RUBLL algorithm is designed to function alongside the online simulation and make desirable control decisions as the system progresses. Unlike in the original simulation, RUBLL allows for the most desirable control decision to be employed at any time throughout the outbreak. The characteristics of the simulated outbreak at points where RUBLL changes the employed control strategy will be of interest here because they allude to the characteristics of real-world outbreaks that require special attention from the health unit. In order to find any of these points of interest the simulation should be run at the lowest depths resulting in each of the control strategies during the offline utility analysis. For the first scenario only $\gamma_0$ and $\gamma_3$ are ever selected by RUBLL, because at depths where $\gamma_1$ and $\gamma_2$ are preferred over $\gamma_3$, $\gamma_0$ still has the lowest utility (Table 5.7). Also recall that $\gamma_4$ is never more desirable than $\gamma_3$ for Scenario One. The utility values for Scenario Two behave similarly and so only $\gamma_0$ and $\gamma_4$ are attainable (Table 5.9). The depths used are as follows:

- Scenario One
  - $\gamma_0$ - depth 3
  - $\gamma_1$ - unattainable
  - $\gamma_2$ - unattainable
  - $\gamma_3$ - depth 12
  - $\gamma_4$ - unattainable
Scenario Two

- $\gamma_0$ - depth 3
- $\gamma_1$ - unattainable
- $\gamma_2$ - unattainable
- $\gamma_3$ - unattainable
- $\gamma_4$ - depth 11

Each scenario was run 10000 times for each of the attainable control strategies. Each selected control strategy was only enforced for one step in the system before recalculating so as not to miss any points of interest. Unfortunately, no changes in control enforcement were observed. Scenario One at depth 12 will always select $\gamma_3$, and at depth 11 will always select $\gamma_0$. Scenario Two at depth 11 will always select $\gamma_4$, and at depth 10 will always select $\gamma_0$. The number of occurrences were recorded over 10000 trials for each, but these results were not significantly different from those in Section 5.1.4 because in both cases a single control strategy was enforced for the entire simulation. A more complex simulation in which parameters may change during the course of the outbreak to more accurately reflect its current conditions would result in a more illuminating online analysis. Control changes in this more dynamic setting might indicate key conditions under which a health unit would want to consider changing strategies in a real-world outbreak. Online analysis of Brunsch’s simulation does not give any new information about controlling real-world outbreaks, but it does show that RUBLL 5.2.1 performs as intended because with a sufficient lookahead depth it will always select the strategy that produces the most desirable set of occurrences.

### 5.5 Analysis Comparison

This chapter presents two basic approaches to analyzing the outbreak simulation. The first was originally performed by Brunsch [8] which involves running the simulation multiple times and recording the number of specific occurrences in each run. The total number of observed occurrences is then averaged over the number of runs to produce the expected number of occurrences for that trial. The effectiveness of each strategy is evaluated based on these expected numbers of occurrences. This technique is online in nature and its results are shown in Section 5.1.4. The second approach to analyzing the outbreak simulation is offline in nature and stems from the origins of the RUBLL algorithm. Offline utility analysis can be used to establish which strategies are favoured at different lookahead depths, which gives insight into the simulation behaviour. The exponential growth rates of utility values indicate the relative rates
of exposures associated with each strategy. The isolation or adjustment of benefit values allows utility values to be broken down and explained in terms of their factors. Markov Chain analysis can be used to determine how successful each strategy is; if the largest eigenvalue of the transition matrix with weighted feedback arcs is less than one, then the strategy will typically result in a contained outbreak. The values in the corresponding eigenvector also reveal the proportion of time the system spends in each state. State-space analysis can be used to estimate execution times for RUBLL.

There are strengths and weaknesses associated with each approach. Brunsch’s online approach is easy to implement and understand, and it does not require any new parameters to be defined. However, it is more computationally time consuming because the simulation must be run multiple times in order to get stable results. It also does not produce any comparable results from uncontrollable trials, and observed occurrence values can only be intuitively explained based on the simulation parameters. The offline analysis using RUBLL is more difficult to implement and debug. Utility analysis requires the definition of benefit values in order to produce any results, and these results are subject to the accuracy of the benefit values. However, once these values have been defined the offline approach affords a variety of techniques for explaining different aspects of the simulation. Utility values indicate the effectiveness of strategies at specific lookahead depths and benefit isolation allows them to be explained. Exponential growth rates and dominant eigenvalues indicate the exposure rates and the success rates of each strategy. Each technique only needs to be applied to each scenario once in order to get stable results, and $\tau$-analysis can be used to estimate the execution times of each application. Finally, RUBLL is designed to automatically make desirable decisions based on the utility values, whereas the results from an online analysis still need to be interpreted by the user.

No single approach is best as they both offer results that the other can not. In the future it may be possible to calculate the expected occurrence values produced by online analysis in an offline fashion. RUBLL can be altered to compute the expected values for specific occurrences in much the same way as it calculates utility. The number of each occurrence is multiplied by the probability of the event string leading to it. Occurrence values calculated in this manner are subject to the same step-wise exponential increase as the utility values, because they observe the same state-space growth in the lookahead window. Controlling for this exponential growth (by dividing by $G^n$ at depth $n$) causes the expected occurrences to eventually settle, but the resulting values are not congruent with the values obtained through online analysis. It is speculated that the correct expected values could be calculated in this fashion by incorporating meaningful values obtained through Markov chain analysis, but this
requires further investigation to be determined.
Chapter 6

Conclusion and Future Directions

This work presents an approach to limited lookahead supervision which combines and extends the algorithms already present in the literature. The goal to produce a supervisory algorithm for Discrete-Event Systems with probabilistic events and a string desirability function was primarily motivated by Brunsch’s outbreak simulation. The resulting algorithm (RUBLL) is discussed in a variety of forms and applied to a variety of problems, including an extensive offline analysis of the outbreak simulation. Somewhere over the course of dealing with countless lookahead trees, some simple observations were made on the relationship between tree state-space and the spectral decomposition of the FSA’s underlying adjacency matrix. From these simple observations sprung a lengthy investigation into the state-space behaviour of general case FSAs, which resulted in a variety of state-space estimation methods, jointly referred to as $\tau$ analysis. The principle of these methods, $\tau$-estimation, is applied to a few different examples here throughout.

Major contributions of this work include the RUBLL algorithm, the $\tau$ analysis methods, and the analysis of the outbreak simulation. The modification of RUBLL as it is applied to the outbreak simulation is also important because it provides an example of how small modifications can allow the algorithm to be useful for a wide range of problems. The $\tau$ analysis methods are useful for anybody working with limited lookahead algorithms or other applications involving the state-space of lookahead trees. The analysis of the outbreak simulation was primarily presented as an example of RUBLL in application, but the results may provide some useful information to health unit officials about the general behaviour of an outbreak. One thing that can be taken from this analysis is the importance of controlling new exposures in an outbreak scenario, because it is the rate of exposures that determine the growth of the outbreak, which is the principle factor in its controllability. Only in cases where it is certain that catering to the interests of an individual can have no possibility of increasing exposures should it be done (i.e., waiting for symptoms to appear before treating a quarantined individual in a scenario with $\lambda_{sym}^q = 0$, as in Scenario One).

The RUBLL algorithm represents the ability to automatically make decisions in a set of problems which are more closely related to the real-world problems we face every day. It is interesting to think that this limited lookahead approach may be related to
how our minds actually solve these real-world problems. An interesting psychological study on this topic would be to have subjects perform a decision-making task which requires some foresight into the future repercussions of their choice. The state space of the lookahead tree required by RUBLL to solve the problem could be tested for correlations with the subject’s recorded reaction times.

Aside from being a possible model for human problem solving, RUBLL’s ability to solve real-world problems means that it has potential application to Artificial Intelligence. Unfortunately, it still requires a user to translate the problem into the correct framework so that RUBLL can understand it. In the distant future it may be possible to design an intelligent agent with sensors and perceptual algorithms complex enough to translate a physical problem into RUBLL’s framework, allowing it to compute the most beneficial course of action. Today’s perceptual algorithms are likely not anywhere near being able to translate a physical environment with the accuracy that is required to make correct decisions. This excludes RUBLL from being autonomously useful for solving real-world problems any time soon, but not all decision-making problems exist solely in the real world any more. Over recent decades more and more of the problems we face exist in some virtualized form, which makes them much easier to translate into a problem that RUBLL can understand. The parameters of these problems are already defined in some framework so their accuracy cannot be disputed. This thought warrants further investigation into the usefulness of RUBLL as an autonomous decision-making algorithm for virtual environments. There are many possibilities for continued development of the algorithm, but in its present form, user-assisted real-world problem solving remains RUBLL’s prominent avenue of application.
Bibliography


BIBLIOGRAPHY


Appendix A

Table A.1: Utility values for depths 1 through 50 computed from the initial state of the simulation for Scenario One. Values are rounded to 2 decimal places.

<table>
<thead>
<tr>
<th>Depth</th>
<th>No Strategy</th>
<th>ISO</th>
<th>SURV</th>
<th>QUARA1</th>
<th>QUARA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>-28.50</td>
<td>-28.50</td>
<td>-28.50</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>-225.00</td>
<td>-238.50</td>
<td>-238.50</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>-104.09</td>
<td>-685.24</td>
<td>-723.39</td>
<td>-804.39</td>
</tr>
<tr>
<td>4</td>
<td>-15.00</td>
<td>-215.41</td>
<td>-956.06</td>
<td>-1051.88</td>
<td>-1131.88</td>
</tr>
<tr>
<td>5</td>
<td>-28.88</td>
<td>-247.11</td>
<td>-2957.90</td>
<td>-3161.11</td>
<td>-3355.00</td>
</tr>
<tr>
<td>6</td>
<td>-31.45</td>
<td>-326.74</td>
<td>-1820.17</td>
<td>-1963.02</td>
<td>-2058.08</td>
</tr>
<tr>
<td>7</td>
<td>-161.07</td>
<td>-1598.69</td>
<td>-3980.58</td>
<td>-4014.14</td>
<td>-4235.41</td>
</tr>
<tr>
<td>8</td>
<td>-517.95</td>
<td>-4008.07</td>
<td>-7380.67</td>
<td>-7479.38</td>
<td>-7811.80</td>
</tr>
<tr>
<td>9</td>
<td>-1202.94</td>
<td>-8651.57</td>
<td>-11424.18</td>
<td>-11952.68</td>
<td>-12351.81</td>
</tr>
<tr>
<td>10</td>
<td>-4882.08</td>
<td>-20637.45</td>
<td>-16015.54</td>
<td>-16607.13</td>
<td>-17631.94</td>
</tr>
<tr>
<td>11</td>
<td>-14401.94</td>
<td>-46823.20</td>
<td>-28376.40</td>
<td>-29729.21</td>
<td>-3284.11</td>
</tr>
<tr>
<td>12</td>
<td>-42131.01</td>
<td>-109435.58</td>
<td>-42989.40</td>
<td>-9371.18</td>
<td>-10246.63</td>
</tr>
<tr>
<td>13</td>
<td>-150104.59</td>
<td>-252043.50</td>
<td>-62411.10</td>
<td>-11706.71</td>
<td>-12729.20</td>
</tr>
<tr>
<td>14</td>
<td>-441872.70</td>
<td>-585215.54</td>
<td>-107343.50</td>
<td>-14106.65</td>
<td>-15430.73</td>
</tr>
<tr>
<td>15</td>
<td>-1400537.04</td>
<td>-1353446.04</td>
<td>-161297.53</td>
<td>-17405.45</td>
<td>-18955.98</td>
</tr>
<tr>
<td>16</td>
<td>-4618825.70</td>
<td>-3136644.48</td>
<td>-240720.30</td>
<td>-20898.94</td>
<td>-22865.06</td>
</tr>
<tr>
<td>17</td>
<td>-13996264.38</td>
<td>-7262087.70</td>
<td>-404182.19</td>
<td>-25571.24</td>
<td>-27879.98</td>
</tr>
<tr>
<td>18</td>
<td>-45081350.39</td>
<td>-16821788.95</td>
<td>-606431.99</td>
<td>-30640.43</td>
<td>-33525.65</td>
</tr>
<tr>
<td>19</td>
<td>-143729983.45</td>
<td>-38956846.10</td>
<td>-922770.63</td>
<td>-37273.18</td>
<td>-40670.29</td>
</tr>
<tr>
<td>20</td>
<td>-446215457.45</td>
<td>-90228292.68</td>
<td>-1520168.76</td>
<td>-44610.47</td>
<td>-48811.80</td>
</tr>
<tr>
<td>21</td>
<td>-1431341076.18</td>
<td>-208968471.82</td>
<td>-2285760.39</td>
<td>-54043.84</td>
<td>-59002.82</td>
</tr>
<tr>
<td>22</td>
<td>-4515338687.16</td>
<td>-483980876.13</td>
<td>-3524263.52</td>
<td>-64643.23</td>
<td>-70729.57</td>
</tr>
<tr>
<td>23</td>
<td>-14202225156.46</td>
<td>-1120912667.19</td>
<td>-5718126.24</td>
<td>-78080.17</td>
<td>-85280.11</td>
</tr>
<tr>
<td>24</td>
<td>-45252903930.87</td>
<td>-2596073098.89</td>
<td>-8632594.54</td>
<td>-93368.25</td>
<td>-102154.93</td>
</tr>
<tr>
<td>25</td>
<td>-142587947155.05</td>
<td>-6012588978.53</td>
<td>-13423476.16</td>
<td>-112531.61</td>
<td>-122946.29</td>
</tr>
<tr>
<td>26</td>
<td>-450814753570.16</td>
<td>-13925355039.68</td>
<td>-21521308.36</td>
<td>-134555.35</td>
<td>-147210.90</td>
</tr>
<tr>
<td>27</td>
<td>-1430032731445.90</td>
<td>-32251581932.98</td>
<td>-32646607.58</td>
<td>-161912.98</td>
<td>-176938.74</td>
</tr>
<tr>
<td>28</td>
<td>-4512038107246.06</td>
<td>-74695724560.00</td>
<td>-51023648.63</td>
<td>-193609.01</td>
<td>-211808.29</td>
</tr>
</tbody>
</table>
Table A.1: Utility values for depths 1 through 50 computed from the initial state of the simulation for Scenario One. Values are rounded to 2 decimal places.

<table>
<thead>
<tr>
<th>Depth</th>
<th>No Strategy</th>
<th>ISO</th>
<th>SURV</th>
<th>QUARA1</th>
<th>QUARA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>-14284370766991.10</td>
<td>-172997765860.92</td>
<td>-8106835.20</td>
<td>-232696.56</td>
<td>-254335.55</td>
</tr>
<tr>
<td>31</td>
<td>-45214846655857.90</td>
<td>-400668521644.05</td>
<td>-123564642.77</td>
<td>-278277.24</td>
<td>-304420.90</td>
</tr>
<tr>
<td>32</td>
<td>-1.43E+14</td>
<td>-927961519001.89</td>
<td>-305557366.00</td>
<td>-399667.63</td>
<td>-437196.75</td>
</tr>
<tr>
<td>33</td>
<td>-4.52E+14</td>
<td>-2149189458177.64</td>
<td>-305557366.00</td>
<td>-399667.63</td>
<td>-437196.75</td>
</tr>
<tr>
<td>34</td>
<td>-1.43E+15</td>
<td>-4977593671101.25</td>
<td>-467890696.17</td>
<td>-479606.68</td>
<td>-524329.33</td>
</tr>
<tr>
<td>35</td>
<td>-4.52E+15</td>
<td>-11528271035785.80</td>
<td>-734094102.92</td>
<td>-573704.14</td>
<td>-627551.30</td>
</tr>
<tr>
<td>36</td>
<td>-1.43E+16</td>
<td>-2669985586776.90</td>
<td>-1152638003.64</td>
<td>-688102.56</td>
<td>-752325.98</td>
</tr>
<tr>
<td>37</td>
<td>-4.53E+16</td>
<td>-61837746501834.60</td>
<td>-1772049639.60</td>
<td>-823215.41</td>
<td>-900451.27</td>
</tr>
<tr>
<td>38</td>
<td>-1.43E+17</td>
<td>-1.43E+14</td>
<td>-2780599908.95</td>
<td>-986984.01</td>
<td>-1079168.48</td>
</tr>
<tr>
<td>39</td>
<td>-4.53E+17</td>
<td>-3.32E+14</td>
<td>-4350854301.78</td>
<td>-1180928.57</td>
<td>-1291688.86</td>
</tr>
<tr>
<td>40</td>
<td>-1.43E+18</td>
<td>-7.68E+14</td>
<td>-6711451129.73</td>
<td>-1415438.74</td>
<td>-1547713.55</td>
</tr>
<tr>
<td>41</td>
<td>-4.53E+18</td>
<td>-1.78E+15</td>
<td>-10526306783.51</td>
<td>-1693761.21</td>
<td>-1852575.57</td>
</tr>
<tr>
<td>42</td>
<td>-1.43E+19</td>
<td>-4.12E+15</td>
<td>-16432139403.20</td>
<td>-2029645.42</td>
<td>-2219399.83</td>
</tr>
<tr>
<td>43</td>
<td>-4.53E+19</td>
<td>-9.54E+15</td>
<td>-25416943306.43</td>
<td>-2428974.15</td>
<td>-2656671.17</td>
</tr>
<tr>
<td>44</td>
<td>-1.43E+20</td>
<td>-2.21E+16</td>
<td>-39833219036.47</td>
<td>-2910140.69</td>
<td>-3182304.75</td>
</tr>
<tr>
<td>45</td>
<td>-4.54E+20</td>
<td>-5.12E+16</td>
<td>-62088026472.00</td>
<td>-3482992.16</td>
<td>-3809430.38</td>
</tr>
<tr>
<td>46</td>
<td>-1.44E+21</td>
<td>-1.19E+17</td>
<td>-96244143892.33</td>
<td>-4172380.43</td>
<td>-4562694.70</td>
</tr>
<tr>
<td>47</td>
<td>-4.54E+21</td>
<td>-2.75E+17</td>
<td>-150697716613.68</td>
<td>-4994049.11</td>
<td>-5462032.16</td>
</tr>
<tr>
<td>48</td>
<td>-1.44E+22</td>
<td>-6.36E+17</td>
<td>-234679523356.15</td>
<td>-5981880.25</td>
<td>-6541583.24</td>
</tr>
<tr>
<td>49</td>
<td>-4.54E+22</td>
<td>-1.47E+18</td>
<td>-364385272164.38</td>
<td>-7160315.98</td>
<td>-7831205.20</td>
</tr>
<tr>
<td>50</td>
<td>-1.44E+23</td>
<td>-3.41E+18</td>
<td>-570037003234.93</td>
<td>-8575920.67</td>
<td>-9378468.82</td>
</tr>
</tbody>
</table>

Table A.2: Utility values for depths 1 through 50 computed from the initial state of the simulation for Scenario Two. Values are rounded to 2 decimal places.

<table>
<thead>
<tr>
<th>Depth</th>
<th>No Strategy</th>
<th>ISO</th>
<th>SURV</th>
<th>QUARA1</th>
<th>QUARA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>-29.85</td>
<td>-29.85</td>
<td>-29.85</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>-487.25</td>
<td>-502.10</td>
<td>-502.10</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>-104.00</td>
<td>-1558.89</td>
<td>-1771.74</td>
<td>-1786.59</td>
</tr>
<tr>
<td>4</td>
<td>-14.99</td>
<td>-188.44</td>
<td>-2207.62</td>
<td>-2499.61</td>
<td>-2712.46</td>
</tr>
<tr>
<td>5</td>
<td>-24.99</td>
<td>-247.08</td>
<td>-2610.51</td>
<td>-3379.84</td>
<td>-3720.22</td>
</tr>
</tbody>
</table>
### Table A.2: Utility values for depths 1 through 50 computed from the initial state of the simulation for Scenario Two. Values are rounded to 2 decimal places.

<table>
<thead>
<tr>
<th>Depth</th>
<th>No Strategy</th>
<th>ISO</th>
<th>SURV</th>
<th>QUARA1</th>
<th>QUARA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>-32.00</td>
<td>-332.16</td>
<td>-8262.57</td>
<td>-5601.93</td>
<td>-5922.10</td>
</tr>
<tr>
<td>7</td>
<td>-43.25</td>
<td>-2151.04</td>
<td>-18908.75</td>
<td>-7861.44</td>
<td>-7812.28</td>
</tr>
<tr>
<td>8</td>
<td>-325.99</td>
<td>-4448.00</td>
<td>-23547.87</td>
<td>-12360.59</td>
<td>-10687.03</td>
</tr>
<tr>
<td>9</td>
<td>-1124.29</td>
<td>-32518.22</td>
<td>-67070.39</td>
<td>-16476.65</td>
<td>-13502.62</td>
</tr>
<tr>
<td>10</td>
<td>-5794.30</td>
<td>-80301.88</td>
<td>-151190.40</td>
<td>-24498.22</td>
<td>-17345.92</td>
</tr>
<tr>
<td>11</td>
<td>-28220.09</td>
<td>-519534.50</td>
<td>-202205.13</td>
<td>-32807.77</td>
<td>-21416.87</td>
</tr>
<tr>
<td>12</td>
<td>-127995.97</td>
<td>-1442653.17</td>
<td>-536890.69</td>
<td>-91144.54</td>
<td>-39609.11</td>
</tr>
<tr>
<td>13</td>
<td>-637489.26</td>
<td>-8401583.09</td>
<td>-63675.81</td>
<td>-32333.67</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>-2966347.48</td>
<td>-25486854.06</td>
<td>-1707966.47</td>
<td>-91144.54</td>
<td>-39609.11</td>
</tr>
<tr>
<td>15</td>
<td>-14312155.45</td>
<td>-136945561.11</td>
<td>-4299691.26</td>
<td>-122140.95</td>
<td>-47777.17</td>
</tr>
<tr>
<td>16</td>
<td>-68145110.67</td>
<td>-44464897.67</td>
<td>-9558482.42</td>
<td>-173120.40</td>
<td>-57692.07</td>
</tr>
<tr>
<td>17</td>
<td>-324766797.48</td>
<td>-224655852.33</td>
<td>-14304845.25</td>
<td>-232676.20</td>
<td>-69153.00</td>
</tr>
<tr>
<td>18</td>
<td>-1553222768.27</td>
<td>-7688027472.61</td>
<td>-3450525.82</td>
<td>-327543.98</td>
<td>-82902.57</td>
</tr>
<tr>
<td>19</td>
<td>-7400182531.76</td>
<td>-3704712609.19</td>
<td>-76161612.70</td>
<td>-441704.21</td>
<td>-98397.92</td>
</tr>
<tr>
<td>20</td>
<td>-35349420696.32</td>
<td>-13204384296.35</td>
<td>-119065778.90</td>
<td>-618668.53</td>
<td>-118045.35</td>
</tr>
<tr>
<td>21</td>
<td>-168637227902.93</td>
<td>-613509288535.50</td>
<td>-277371188.67</td>
<td>-836782.88</td>
<td>-140444.15</td>
</tr>
<tr>
<td>22</td>
<td>-804839987705.50</td>
<td>-2256707726360.41</td>
<td>-607847199.18</td>
<td>-1167579.04</td>
<td>-167029.73</td>
</tr>
<tr>
<td>23</td>
<td>-3841253665782.17</td>
<td>-10194185499058.30</td>
<td>-986160024.73</td>
<td>-1583395.29</td>
<td>-198287.72</td>
</tr>
<tr>
<td>24</td>
<td>-18330187967932.90</td>
<td>-3842518525547.90</td>
<td>-2233221493.11</td>
<td>-2207289.79</td>
<td>-235304.48</td>
</tr>
<tr>
<td>25</td>
<td>-87483769968138.90</td>
<td>-1.70E+14</td>
<td>-4858613740.74</td>
<td>-2994016.06</td>
<td>-279091.91</td>
</tr>
<tr>
<td>26</td>
<td>-4.17E+14</td>
<td>-6.52E+14</td>
<td>-8135274638.60</td>
<td>-415545.35</td>
<td>-330463.86</td>
</tr>
<tr>
<td>27</td>
<td>-1.99E+15</td>
<td>-2.84E+15</td>
<td>-18002720978.30</td>
<td>-5658791.31</td>
<td>-391252.76</td>
</tr>
<tr>
<td>28</td>
<td>-9.51E+15</td>
<td>-1.11E+16</td>
<td>-38889646285.39</td>
<td>-7839194.52</td>
<td>-463092.31</td>
</tr>
<tr>
<td>29</td>
<td>-4.54E+16</td>
<td>-4.74E+16</td>
<td>-66892755632.22</td>
<td>-10692096.68</td>
<td>-547836.25</td>
</tr>
<tr>
<td>30</td>
<td>-2.17E+17</td>
<td>-1.87E+17</td>
<td>-145269724524.44</td>
<td>-14789440.17</td>
<td>-647941.71</td>
</tr>
<tr>
<td>31</td>
<td>-1.03E+18</td>
<td>-7.94E+17</td>
<td>-311676683943.17</td>
<td>-20198152.02</td>
<td>-766068.26</td>
</tr>
<tr>
<td>32</td>
<td>-4.93E+18</td>
<td>-3.16E+18</td>
<td>-548555460451.43</td>
<td>-27903967.96</td>
<td>-905572.23</td>
</tr>
<tr>
<td>33</td>
<td>-2.35E+19</td>
<td>-1.33E+19</td>
<td>-1173154414745.96</td>
<td>-38150018.24</td>
<td>-1070221.56</td>
</tr>
<tr>
<td>34</td>
<td>-1.12E+20</td>
<td>-5.33E+19</td>
<td>-2500752476998.09</td>
<td>-52651904.03</td>
<td>-1264639.08</td>
</tr>
<tr>
<td>35</td>
<td>-5.36E+20</td>
<td>-2.23E+20</td>
<td>-4488484567925.70</td>
<td>-72049111.80</td>
<td>-1494125.61</td>
</tr>
<tr>
<td>36</td>
<td>-2.56E+21</td>
<td>-8.99E+20</td>
<td>-9479954582981.13</td>
<td>-99355788.53</td>
<td>-1765079.77</td>
</tr>
<tr>
<td>37</td>
<td>-1.22E+22</td>
<td>-3.75E+21</td>
<td>-20085537055070.40</td>
<td>-136058155.24</td>
<td>-2084929.22</td>
</tr>
</tbody>
</table>
Table A.2: Utility values for depths 1 through 50 computed from the initial state of the simulation for Scenario Two. Values are rounded to 2 decimal places.

<table>
<thead>
<tr>
<th>Depth</th>
<th>No Strategy</th>
<th>ISO</th>
<th>SURV</th>
<th>QUARA1</th>
<th>QUARA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>-5.83E+22</td>
<td>-1.52E+22</td>
<td>-36659078365377.40</td>
<td>-187499172.74</td>
<td>-2462556.14</td>
</tr>
<tr>
<td>39</td>
<td>-2.78E+23</td>
<td>-6.30E+22</td>
<td>-76642447894382.10</td>
<td>-256915751.24</td>
<td>-2908344.69</td>
</tr>
<tr>
<td>40</td>
<td>-1.33E+24</td>
<td>-2.55E+23</td>
<td>-1.61E+14</td>
<td>-353857685.49</td>
<td>-3434645.40</td>
</tr>
<tr>
<td>41</td>
<td>-6.33E+24</td>
<td>-1.06E+24</td>
<td>-2.99E+14</td>
<td>-485102106.30</td>
<td>-4055956.72</td>
</tr>
<tr>
<td>42</td>
<td>-3.02E+25</td>
<td>-4.30E+24</td>
<td>-6.20E+14</td>
<td>-667847505.55</td>
<td>-4789468.67</td>
</tr>
<tr>
<td>43</td>
<td>-1.44E+26</td>
<td>-1.78E+25</td>
<td>-1.30E+15</td>
<td>-915917847.25</td>
<td>-5655408.92</td>
</tr>
<tr>
<td>44</td>
<td>-6.88E+26</td>
<td>-7.25E+25</td>
<td>-2.43E+15</td>
<td>-1260497905.20</td>
<td>-6677716.86</td>
</tr>
<tr>
<td>45</td>
<td>-3.29E+27</td>
<td>-2.99E+26</td>
<td>-5.01E+15</td>
<td>-1729276581.86</td>
<td>-7884601.04</td>
</tr>
<tr>
<td>46</td>
<td>-1.57E+28</td>
<td>-1.22E+27</td>
<td>-1.05E+16</td>
<td>-2379141687.38</td>
<td>-9309411.25</td>
</tr>
<tr>
<td>47</td>
<td>-7.48E+28</td>
<td>-5.03E+27</td>
<td>-1.98E+16</td>
<td>-3264824734.82</td>
<td>-10991475.95</td>
</tr>
<tr>
<td>48</td>
<td>-3.57E+29</td>
<td>-2.05E+28</td>
<td>-4.06E+16</td>
<td>-4490653001.35</td>
<td>-12977263.31</td>
</tr>
<tr>
<td>49</td>
<td>-1.70E+30</td>
<td>-8.45E+28</td>
<td>-8.43E+16</td>
<td>-6163750928.59</td>
<td>-15321597.19</td>
</tr>
<tr>
<td>50</td>
<td>-8.13E+30</td>
<td>-3.46E+29</td>
<td>-1.61E+17</td>
<td>-8476327851.31</td>
<td>-18089231.56</td>
</tr>
</tbody>
</table>