DISTRIBUTED REINFORCEMENT LEARNING IN EMERGENCY RESPONSE SIMULATION

by

Cesar Lopez

B.Sc., Cooperative University of Colombia, 2001
Ed.S., Cooperative University of Colombia, 2004
M.A.Sc., The University of British Columbia, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

March 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the dissertation entitled:

Distributed Reinforcement Learning in Emergency Response Simulation

submitted by Cesar Lopez in partial fulfillment of the requirements for

the degree of Doctor of Philosophy

in Electrical and Computer Engineering

Examining Committee:

Jose R. Marti
Supervisor

Sarbjit Sarkaria
Supervisory Committee Member

Christine Chen
Supervisory Committee Member

Shahriar Mirabbasi
University Examiner

Terje Haukaas
University Examiner

Additional Supervisory Committee Members:

Supervisory Committee Member

Supervisory Committee Member

Abstract

In this thesis we present the implementation of a coordinated decision-making agent for emergency response scenarios. The agent's implementation uses Reinforcement Learning (RL). RL is a machine learning technique that enables an agent to learn from experimenting. The agent's learning is based on rewards, feedback signals proportional to how good its actions are. The simulation platform used was i2Sim, the Infrastructure Interdependencies Simulator, in which, we have tested the suitability of the approach in previous studies. In this work, we have added new features, for increasing the speed of convergence and enabling distributed processing capabilities. These additions include enhanced reward and exploration schemes and a scheduler for orchestrating the distributed training.

We include two test cases. The first case is a compact model with 4 critical infrastructures. In this model, the agent's training required only 10% of the attempts as compared to references given by past studies done in our group. Improvements in convergence come from the enhanced shaping reward and exploration schemes. We trained the agent across 24 simultaneous configurations of our model (scenarios). The complete distributing training process needed 4 minutes.

The second case is an extended model, a more detailed representation of the first case. This extended case included additional infrastructures and a higher level of resolution. By adding more infrastructures, the dimensionality of the problem grew four thousand times. This dimensionality growth did not affect performance and the training had an even faster

convergence. We ran 96 parallel instances of the extended model and the process completed in 2.87 minutes. The results show a fast and stable convergence framework with a wide range of applicability. This agent could help during multiple stages of emergency response including real time situations.

Lay Summary

This thesis provides emergency responders with a tool that can help them make decisions during challenging situations. We use artificial intelligence to train an agent in simulated emergency response scenarios. With the help of robust computational equipment, we were able to arrange our implementation to be deployed with parallel processing capabilities. Parallel processing means that we can run simultaneous simulations spread across different computers. This technology reduces the time needed to find the answers emergency responders seek. We were able to complete the evaluation of about 100 simultaneous scenarios in less than 3 minutes. Hence, the responder using our tool would know what to do in less than 3 minutes, while considering 100 possibilities after an incident occurs. Better decisions can save thousands of lives during natural disasters or other hazardous events.

Preface

The work accomplished in this thesis has led to a number of accepted or under processing publications in journals and conferences. In some of those publications I have acted as coauthor, providing assistance in specialized technical tasks. As an author I have prepared the entirety of the documents. Common to all publications is the supervision of Dr. José Martí and the assistance of Dr. Sarbjit Sarkaria.

The reference models and motivation for this project come from past implementations that were presented at:

- 24th Canadian Conference on Electrical and Computer Engineering, 2011 CCECE. Appears in the proceedings as: *"Disaster Management in Real Time Simulation using Machine Learning"*.

- Eighth IFIP WG 11.10 Conference on Critical Infrastructure Protection, 2014 IFIP. Found in the proceedings as: *"Reinforcement Learning using Monte Carlo Policy for Disaster Mitigation"*.

For these two publications I was a coauthor. I provided assistance in modeling and specialized help in programming the learning agents and multiplatform interaction.

A part of Chapter 3 was accepted at the IEEE Global Humanitarian Technology Conference, GHTC 2018. This work was presented on October 19th 2018 and it carries the title: *"Distributed*

Reinforcement Learning Framework for Resource Allocation in Disaster Response". The final proceedings have already been published. I am the main author. I have conducted all the experimental work and written the manuscript.

A version of Chapter 4 was accepted at the 2018 IEEE International Systems Engineering Symposium (ISSE). This work carries the title: *"Multiagent Optimization in Disaster Response"*. I am the main author. I have performed all technical work and written the manuscript.

A preliminary version of the model described in Chapter 4 was included in Deliverable D 4.8 of the FP7 project of CIPRNet, The Critical Infrastructures Preparedness and Resilience Research Network. I put together the model and provided all the documentation of its functionality and configuration parameters.

Reduced versions of Chapters 3 and 4 were combined in a Journal paper accepted by IEEE Access on August 2018. The paper has been published as: *"Distributed Reinforcement Learning in Emergency Response Simulation"*. As the main author I designed the models, conducted simulation and analysis of results and wrote the manuscript.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

a	Action (MDPs)
AI	Artificial Intelligence
BC	British Columbia
CI	Critical Infrastructure
CII	Critical Infrastructure Interdependencies
CSQ	Cathedral Square Substation
DGR	Dal Grauer Substation
DP	Dynamic Programming
EMIS	Emergency Management Information Systems
ER	Emergency Room
ft^3	Cubic Feet
GB	Giga Bytes
GHz	Giga Hertz
HPC	High Performance Computing
HRT	Human Readable Table
i2Sim	Infrastructures' Interdependencies Simulator
IT	Information Technology
kL	Kilo Liter
LUT	Look Up Table
MATLAB	Matrix Laboratory
MB	Mega Bytes
MC	Monte Carlo
MDP	Markov Decision Process
MPI	Message Passing Interface)
MUR	Murrin Substation
MW	Mega Watt
OI	Operating Index
PC	Production Cell
PM	Physical Mode
RHEL	Red Hat Enterprise Linux
RL	Reinforcement Learning
RM	Resource Mode

s	State (MDPs)
SPG	Sperling Substation
SPH	Saint Paul's Hospital
t	Time
TD	Temporal Difference
UBC	The University of British Columbia
VGH	Vancouver General Hospital
WaterPS	Water Pumping Station
xCAT	Extreme Cluster/Cloud Administration Toolkit

Glossary

Crisis Management	Is the process of dealing with disruptive and unexpected events. Key elements are: a threat, surprise, and a short decision time. Crisis management deals with the threats prior, during, and after they strike.
Head Node	A management node, in a cluster, that acts as a single point of remote access. Interaction with computing nodes is always done through the head node.
HPC Cluster	A grouping of separate servers, called nodes. This group of servers offers combined memory and processing capacity to users. Users perceive a cluster as a single processing unit.
Linux	Linux is an open source operating system. Freely distributable and cross-platform. Linux is based on Unix.
Montecarlo	Optimization technique based on repeated random sampling. Generally, it provides approximate solutions. Used in cases where analytical or numerical solutions don't exist or are too difficult to implement.
Node	A server part of a computing cluster. A usual classification is to have computing nodes for performing calculations / processing; and management nodes for interfacing computing nodes, web services or storage units.
Q-Learning	A temporal difference off-policy method. Q-Learning follows a greedy approach and it can converge faster than Sarsa, but its convergence is not fully guaranteed.

Sarsa A temporal difference method in reinforcement learning. It takes its name from its definition: $\langle \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{r}_{t+1}, \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1} \rangle$, where $\boldsymbol{s}$ refers to state, $\boldsymbol{a}$ to action and $\boldsymbol{r}$ to reward.

Watershed Water reservoir, usually located in a protected area, used as a water intake for supplying tap water.

Acknowledgements

I want to extend my gratitude to Professor José R. Martí, my supervisor, for all the invaluable lessons shared during these years of working in a variety of multidisciplinary projects. Giving me the opportunity of joining his team has been an unforgettable experience that enabled me to grow as a professional in both the academic and technical dimensions. Moreover, it provided an opportunity for establishing long lasting friendships.

I want to thank Dr. Sarbjit Sarkaria for his insight and great ideas in formulating the framework that evolved to be the core of this dissertation. His willingness to accommodate time during weekends and night time for sharing his ideas with me is remarkable.

I also want to acknowledge the help provided by Dr.Paul Lusina in proof-reading my manuscripts. Likewise, I acknowledge the good ideas and recommendations received from Dr. Hao Ma, Dr. Benedito Bonato and Dr. Bojiang Ma in relation to publishing, styling and document organization.

My sincere gratitude to Wayne Tebb, Dr. Xing Liu and Dr. Mandeep Pannu for allowing me to become part of KPU's CSIT Department. Being part of a great team is a true blessing and it gave me the opportunity of sharing and polishing my ideas towards the final portion of my program.

Finally, a special thanks to all my family for their patience, understanding and support during the good and bad moments experienced in this process. Balancing family, work and research,

efficiently, could require a more complex framework than the one proposed in this dissertation.

The support of my family made it possible.

Dedication

This work is dedicated to my family.

Chapter 1: Introduction

1.1 Motivation

The frequency of natural disasters is increasing (Figure 1.1). Three main contributors to this trend are: growing population and infrastructures, vast population growth in coastal areas and urbanizing risk-prone areas [1] [2].

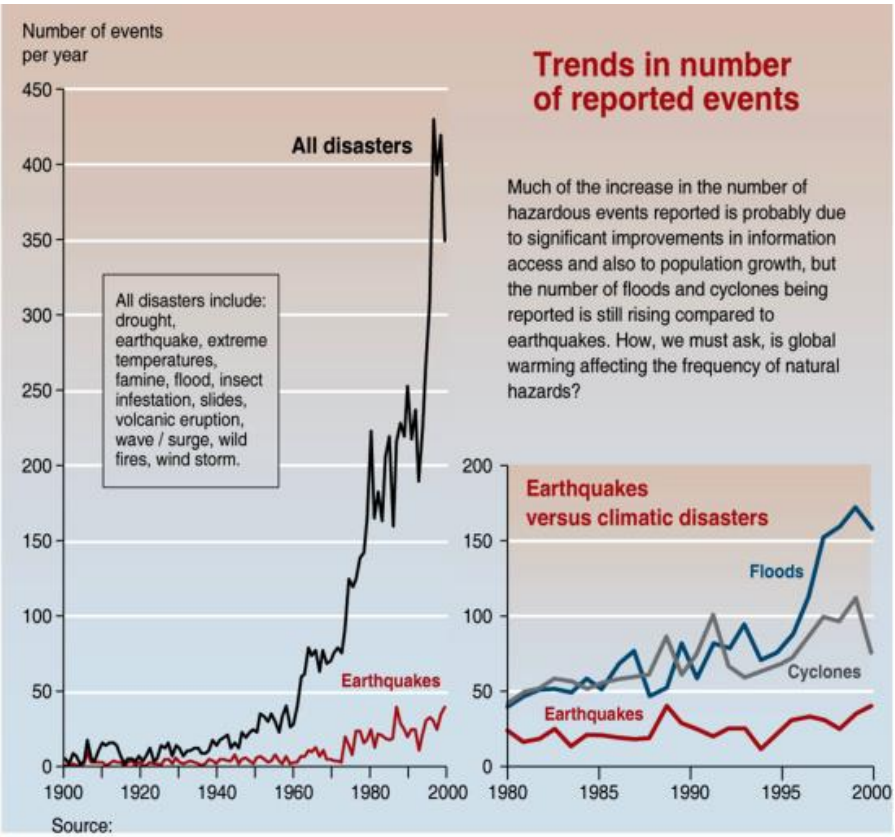


Figure 1.1 Trends in Natural Disasters [1]

Nowadays, people seem to be more aware of disaster occurrences and their negative impact on society. Due to the Internet and social networks [2], disaster news travels faster than before and spreads in no time.

Social networks have opened space for fake news through these channels of incoming data. But, when proper filters are in place, valuable input can be obtained in favour of decision makers. One key factor that has played a positive role in minimizing deaths, while the disasters become more frequent, is the evolution of Emergency Management as a discipline and its systematic contribution to increasing the effectiveness of first responders. For the last three decades Emergency Response has professionalized in higher education, from 1996 to the middle of 2006 the number of programs available has increased from 2 to over 100 [3].

Making decisions during a disaster is one of the most critical tasks responders perform, given that the timeframe is limited and lives and property are at stake. Governments keep enhancing their response plans and strategies, but one thing is common to all of them, the consideration of Critical Infrastructures (CIs) as foundational. The U.S. Department of Homeland Security in [4] defines CIs as the backbone of the country's economy, security and health. CIs are essential to the well-being of people and their failures can cascade with disastrous consequences as they are highly interdependent. Hence, a coordinated response is necessary to avoid poor decisions amplifying the impact of hazardous situations.

Our research group has developed a simulation platform (i2Sim) based on the analysis of Critical Infrastructure Interdependencies. i2Sim, the Infrastructures Interdependencies Simulator, includes the advantages of topological and agent-based simulation. Its flexible architecture enables a holistic analysis of emergency scenarios. To be of aid for decision makers, a separate layer for decision making allows testing of different optimization algorithms without modifying the representation of physical components.

i2Sim has been used for different applications in our own projects and in multiple collaborations. An important milestone in i2Sim's evolution happened in 2009 when Defence Research and Development Canada (DRDC) assigned our group a project prior to the Vancouver 2010 Winter Olympics. The outcomes of this project included an improved version of the simulator and a model of the City of Vancouver [5]. These deliverables were used by responders for evaluating potential disaster scenarios and their contingencies in planning stages of the Winter Olympics. The City of Vancouver (CoV) model became an important test model and it has been used as a baseline for evaluating different optimizations techniques in several projects. This work focuses on two of those projects [6] [7] that optimize decision making in disaster scenarios. We propose significant improvements over their achievements while adding what they proposed as future work. Details about the reference projects are provided below.

The first referential project, and most affine to this dissertation, is a decision-making study in the context of disaster response using reinforcement learning (Khouj et al.). The project used a reduced version of the CoV model and mapped a small set of variables from the environment due to limitations in memory, the "curse of dimensionality". Their initial solution converged, but slowly. In that project the authors used a Java-MATLAB joint implementation that introduced significant communication overhead due to inter-platform synchronization [8].

A new attempt at that initial solution moved the original approach towards Montecarlo techniques for fixing communication latency issues; but it sacrificed the quick convergence offered by temporal difference methods [9]. The third iteration of the referred effort was solely written in Java. An abstraction of the i2Sim model, used for the study, was coded in Java aiming

at improving communication efficiency [6]. While this approach trained the agent in a few minutes it had the downside of losing i2Sim's functionality in trade for an increase in performance. We believe that the original i2Sim's architecture encompasses an advantageous framework for the disaster response problem, hence it is better to adapt a method to work with i2Sim instead of the converse. A literature review in support of this claim is provided in the next chapter.

In the second referential project [7], Alsubaie discusses an optimization over a specific CI i2Sim scenario. The author proposes a solution via ordinal optimization. According to the author a global maximum is not guaranteed, instead a "good enough solution" is found. While the solution achieves improvement over a baseline, no references to the number of episodes or running times are provided. The author proposes a second solution with a linear programming adaptation but considers it more computationally expensive. As in his first solution, Alsubaie does not mention any efficiency parameters or time complexities for his models. Similar to the approach taken by Khouj et al., Alsubaie executes his optimization apart from the i2Sim framework and uses it merely for validation.

This thesis addresses the limitations of the two aforementioned studies. The starting point consists on analyzing and improving the decision-making process with efficiency metrics that allow comparison. The following step is a proposed implementation that interacts "online" with i2Sim. The final step is to add parallel processing capabilities as an efficiency/dimensionality booster. As mentioned earlier, these improvements respond to the weaknesses found in the referential cases listed previously [6] [7], as well as implementing their proposed future work.

What this thesis proposes is a *unified simulating platform*, i.e., the optimization agents are part of i2Sim's decision layer. By doing this, i2Sim's full functionality remains available to the users. With the optimization solution contained within the MATLAB/Simulink domain, intensive inter-platform message passing is avoided and communication overhead is minimized. We use the same machine learning approach used in [6] and [8], Reinforcement Learning (RL), but, we take it a step further by proposing an enhanced reward scheme that speeds up convergence by a very large margin. In addition to the improvements in the sequential solution (single thread running on a single PC), we propose a parallel distributed algorithm that helps with the dimensionality constraints while allowing simultaneous multi-scenario optimizations.

Reinforcement Learning (RL) is based on an agent interacting with the environment, making experimental decisions and learning by experience. RL has been chosen because it offers the possibility of working with limited information. Emergency scenarios, including critical infrastructures, are represented without a "perfect model" of the environment due to a variety of factors like confidentiality and complexity, among others.

This PhD dissertation encompasses four foundational concepts: Emergency Response, Critical Infrastructure Analysis and Simulation, Reinforcement Learning, and Distributed Processing. No formulations of this problem enclosing all these terms were found during the literature review. Relevant concepts and related work about these topics are described in the next chapter.

1.2 Problem Statement and Research Objectives

Decision making in emergency response is a non-trivial activity. Experience and historic records are important tools for dealing with hazardous events, but often they are not enough. Poor

decision making might amplify the initial impact of the incident/disaster and elevate its consequences to catastrophic. Evaluating new potential scenarios and discovering new policies for dealing with them can provide important help for managers and responders. Likewise, having supporting tools that can help discover optimal solutions for unknown situations is ideal. If the frameworks capable of helping with decision making are fast enough, they can be used in real time situations. In response to the aforementioned, this dissertation proposes the following objectives:

1. To re-engineer the City of Vancouver model as a larger model with multiple critical infrastructures. The representation must be done in a realistic manner, to the highest possible extent.

2. To implement a reinforcement learning (RL) agent(s) capable of finding the best decision for all the proposed model(s). This agent should improve over the referential past projects, in similar optimization contexts, by increasing the speed of convergence. The agent must solve the proposed scenarios in a few minutes. Moreover, the agent(s) are to be deployed in i2Sim's decision layer. Hence, i2Sim remains fully functional and orchestrates the simulation of scenarios.

3. To identify decoupling points in the model that enable parallelization of the "best" sequential version. With the aim of extending the solution to a parallel/distributed framework, find decoupling points in coordination with the RL implementation. This way, the agent can handle larger environments, while training in a parallel multi-scenario process.

4. To write a scheduler that handles the distributed version of the proposed model(s). This scheduler is expected to provide a good level of automation, with a strong focus on reducing communication overhead between computing nodes.

1.3 Thesis Contributions and Organization

The main contributions of this PhD work are:

1. An improved version of our baseline models of the City of Vancouver. Additions and changes relate to operational data and optimization of subsystems. This model is to be used as a baseline for future projects.

2. The addition of an integer distributor to i2Sim's toolbox. This component will keep the outputs as integers according to the distribution ratios. This element enables better mapping of entities in real life applications, e.g., avoiding situations where two halves of an ambulance are sent to two different places.

3. A significant increase in the speed of convergence of the reinforcement learning agent(s). This is achieved by adding an improved reward scheme using reward shaping. Likewise, using a decreasing exploratory scheme (variable $\in -greedy$ scheme) for the agent(s) to try new things less frequently as getting more experienced. These improvements remain in the context of emergency response scenarios.

4. Providing a methodology for decoupling state variables (environment descriptors) in tabular reinforcement learning based agents. This allows splitting the Look-Up Table

(LUT) into independent sub-matrices. Without locating the partitioning points, a parallel/distributed solution is not possible.

5. A distributed reinforcement learning framework, capable of decomposing the look up table (LUT) into independent submatrices. This methodology addresses dimensionality constraints while enabling simultaneous instances of a model to be optimized. Optimizing simultaneous instances of a model allows multi-scenario solutions.

6. The implementation of a *Scheduler* to manage the distributed training process of the agent(s). The scheduler provisions simulation in the loop for automating the iterative training process. Likewise, it minimizes communication overhead by using the clustering software as a trigger and passing the control to MATLAB/Simulink.

This document is structured as follows:

Chapter 1: describes the motivation behind this project and lists the objectives and contributions of this thesis. Chapter 2: provides definitions and a thorough literature review of the core topics encompassed by the solution described in this dissertation. Chapter 3: describes the simulation framework (i2Sim) and its architecture. Additionally, it provides a contextualized definition of reinforcement learning. Finally, it describes the computing infrastructure hosting the solution. Chapter 4: discusses the approach used to implement the aggregate test case, improvements to the sequential version and the scheduler architecture. Results are analyzed, and conclusions listed. Chapter 5: covers the process of growing the

compact model into a detailed one. Configurations and assumptions are explained, along with the agent's adaptation. The adequacy of the distributed environment is discussed with its corresponding results and conclusions. Chapter 6: summarizes all the work enclosed by this thesis and goes through recommendations for future related studies.

Chapter 2: Emergency Response, CI Interdependencies and Parallel Processing

This chapter collects the definitions and related work for the three topics on which this thesis is based on: emergency response (ER), critical infrastructures and their interdependencies, and parallel and distributed processing. The first section goes over emergency response (ER) as a process and shows how this work fits in all stages of the emergency continuum. This ER section also provides a list of relevant terms and a literature review on machine learning applications in the ER context. The second section of this chapter describes critical infrastructures (CI) and justifies their analysis within emergency response activities. Likewise, the CI section lists relevant references and collects, from literature, recommended practices for CI modeling and related work. The final portion of this chapter starts with an overview of parallel processing covering basic concepts, cluster architectures and big data frameworks. The remainder of this last section evaluates related work aiming at finding the best parallel/distributed approach for reinforcement learning (RL).

2.1 Emergency Management

Emergency Management is defined in [10] as: *"The discipline and profession of applying science, technology, planning and management to deal with extreme events that can injure or kill large numbers of people, do extensive damage to property, and disrupt community life"*.

With the intent of facilitating the understanding of emergency response's concepts and aiming at removing any ambiguation, we provide a set of definitions below:

- ***Hazard:*** is a situation that poses potential harm affecting people, property or the environment [11].

- ***Disaster:*** when the negative impact of hazard becomes a reality, it is called a Disaster [12]. In other words, hazard is the possibility of something bad happening, while a disaster is the result of that something bad happening.

- ***Emergency*** refers to the imminence of an event taking place rather than its consequences. The process requires attention for minimizing the effects and would certainly go beyond routine procedures [13].

- ***Risk*** is a probability assessment that quantifies the damage and/or the economic impact under the occurrence of a harmful event [12].

- ***Vulnerability*** reveals how much a hazard or disaster can affect human life and property.

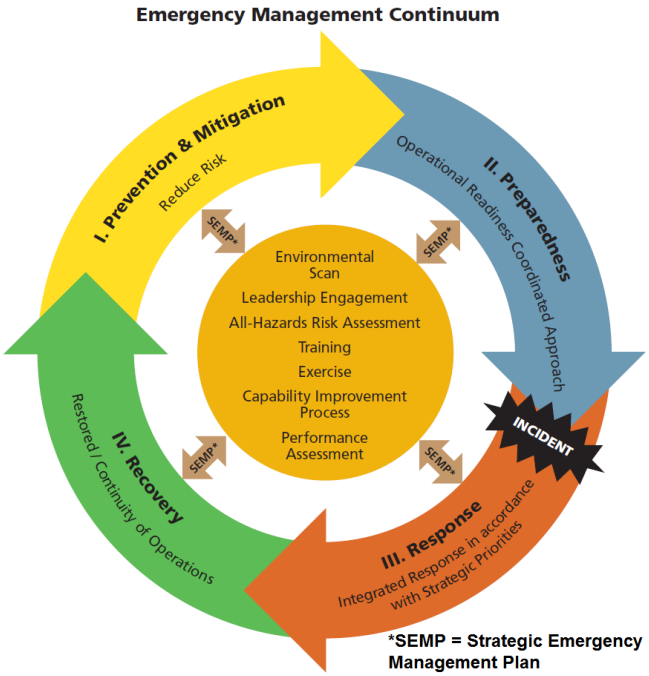


Figure 2.1 Emergency Management Cycle [14]

Emergency management, as a process, is normally seen as a linear sequence of steps. In reality, emergency management is a continuum that cycles through four interconnected stages [15]. The emergency management continuum is illustrated in Figure 2.1, above. The phases of the emergency management cycle are listed below, along with their corresponding definitions.

- ***Prevention/Mitigation:*** is an attempt to keep hazards from turning into disasters, or to reduce the consequences of disasters when they happen. Mitigation efforts pursue long-term actions to manage (reduce or remove) the risk [16].

- ***Preparedness:*** getting ready to respond to disasters and to manage their negative consequences by using contingencies taken prior to an event, i.e. emergency plans, training, etc. [15]. This first half of the emergency response continuum is carried out without an incident. Some literature joins preparedness and prevention as a single phase.

- ***Response:*** all emergency response activities start from the moment an incident strikes and end once the situation following impact has been stabilized [17].

- ***Recovery:*** comprises all those measures taken to bring back communities to pre-emergency conditions, or even a better status; after an emergency or disaster has been detected and response has ended [11].

Traditionally, emergency management in Canada has focused on preparedness and response [15]. As observed in Figure 2.1, the emergency management cycle is an unending process or interrelated steps [14] and all stages are important.

The implementation discussed in this dissertation includes advantages that make it usable at any of the stages of the emergency management cycle. The explanation for this claim follows:

During *prevention/mitigation*, tasks related to identifying the sensitivity of CIs and evaluating their resilience can be carried out for devising new ways of increasing robustness for those CIs with highest level of criticality. In relation to *preparedness*, simulation of past events enables managers and planners to evaluate the outcome of decisions made, while responding to those events. Likewise, they can test other actions that could potentially lead to better results. These lessons learned are a good insight towards improving action plans and procedures for first responders. For these two first stages, the time to produce results does not follow tight constraints as no disrupting event has yet happened.

During *response*, the time window to identify suitable actions is very limited and it does not allow more than a few minutes. Hence, a real-time simulation environment, capable of determining best course of action, is ideal. *Recovery* uses knowledge gathered during previous stages and evaluate a ***positive*** cascading effect aiming at prioritizing restoration of infrastructures in a way that minimizes the total restoration time. The features included in this project match the requirements listed in each stage. Moreover, some of our past projects have modeled similar activities, with an emphasis on disaster response.

2.1.1 Related work

As stated in [10] Emergency Management uses all technological means for reducing the impact of emergencies/disasters/catastrophes on people and property. Developing Emergency

Management Information Systems (EMIS) started with Murray Turoff in the late sixties [18], since then, advances in IT have allowed their continuous evolution.

With the evolution of artificial intelligence and, more precisely, machine learning, many efforts have been directed to bettering decision making, thus, improving the effectiveness of emergency responders.

2.1.1.1 Emergency Management and Machine Learning

Most of the work found on machine learning techniques, aiding emergency procedures, focus on very specific tasks and consider the environment in isolation. A distributed building evacuation simulator is presented in [19]. The tool developed, allows for a distributed environment where each floor can be processed independently from the others. This paper assumes independence of the processes taking place in each floor as in common areas like stairs. In this approach, agents (evacuees) are preprogrammed on what action to take, e.g., going to next exit. The simulators in charge of each level only communicate among themselves when an agent changes floors. For example, an agent from the third floor arrives to the second floor, then it is eliminated from the simulator of the third floor and added to the one in charge of the second floor. The agents' behaviour can be updated in runtime and their trajectories are specified using graphs. The intention of the proposed solution in [19] is to estimate the best routes for evacuation by interacting with sensor networks and informing the evacuees through panel indicators. Similarly, the topic under study in [20] is evacuation, it includes a broader view of the evacuation modelling, but the solution is restricted merely to egress simulation. The authors approached the problem by adapting the cognitive packet network concept used in networks. In this type of solution, every

node has a *random neural network* that gets updated by a reinforcement learning algorithm. For [20] to work, a graph-based representation and sensor networks are needed.

An approximate dynamic approach is used in [21] for implementing a dynamic ambulance dispatch and relocation. The proposed solution in [21] reduced the state space by means of spatial and temporal aggregation. The aggregation was used only for function approximation but not for modelling the dynamics of the model. After extensive tests on real data and comparison to historic data, the response time showed a 12.89% reduction. As mentioned previously for [19] and [20], the tasks performed by the author in [21] were done without considering an interdependent environment.

In [22] a concise background about EMIS is provided, highlighting their function in helping decision makers in emergency response. The paper recognizes that the systems analyzed in emergency management are interdependent and complex in nature. Despite the extensive referencing to complex interdependent systems, no framework is proposed to include these features. The authors focus on processing data with no regard for simulation. The authors refer to Data Mining (DM) and Machine Learning (ML), but they don't propose any implementation.

The study described in [23] introduces a methodology for environmental emergency decision support. The authors rely on an Artificial Neural Network (ANN) for solving an atmospheric accident in a district of Shanghai. The main motivation for [23] is the high frequency of occurrence of these type of events in China, one every two days. The limitations found by the authors include a difficulty to integrate ANNs in this type of applications and lack of training

samples. The work in [24] aims to provide forecasts of hurricane inundation with high resolution and in a short time. For achieving their goal, the authors have used pattern recognition via an Artificial Neural Network (ANN). The training data used in [24] comes from an extensive historic database of storms and storms responses. The approach taken in this project was tested over data from New Orleans and surroundings municipalities from the Gulf of Mexico.

Some of the previously mentioned efforts used supervised learning approaches like Artificial Neural Networks. Supervised learning is based on examples provided by a knowledgeable/experienced supervisor or historic data. Experience is important, but by exploring new options there is a chance to expand that knowledge and discover new alternatives. In uncharted territory interaction becomes key, an agent needs to learn from his own experience which is unsupervised learning [25].

Reinforcement learning (RL) in the context of disaster response is discussed in [8]. The authors implemented an agent that interacted with an experimental model built with the Infrastructures Interdependencies Simulator (i2Sim). The *Q-Learning* algorithm used a *Look up table (LUT)* with a limited representation of the full state space. This initial effort used a static exploration rate and provided experimental support for learning rate and discounted reward values. The convergence of the process occurred after 50 complete runs of the model. Further work presented in [9] introduced a change from the Q-Learning to a Monte-Carlo solution. This change in the RL technique was intended as a solution for communication the overhead created by the Java-MATLAB interaction. The Monte-Carlo policy estimation required 100 runs of the model under study. The model was an extension of the one used in [8] but the state space remained limited.

2.2 Critical Infrastructures

A Critical infrastructure (CI) is defined as: *"processes, systems, facilities, technologies, networks, assets and services essential to the health, safety, security or economic well-being of Canadians and the effective functioning of government"* [26] by Public Safety Canada in the National Strategy for Critical Infrastructure. This definition considers similar components as the ones provided by countries like US, Australia, New Zealand and UK. This can be found in a state of the art in critical infrastructure protection and resilience collected through a project supported by Defense Research and Development Canada (DRDC) [27]. The US Department of Homeland Security specifies 16 critical infrastructure sectors in [28]. Similarly, the Government of Canada defines 10 Critical Infrastructures (CIs): energy and utilities, information and communication technology, finance, health, food, water, transportation, safety, government and manufacturing. The Canadian critical infrastructure sectors are shown in Figure 2.2 below:

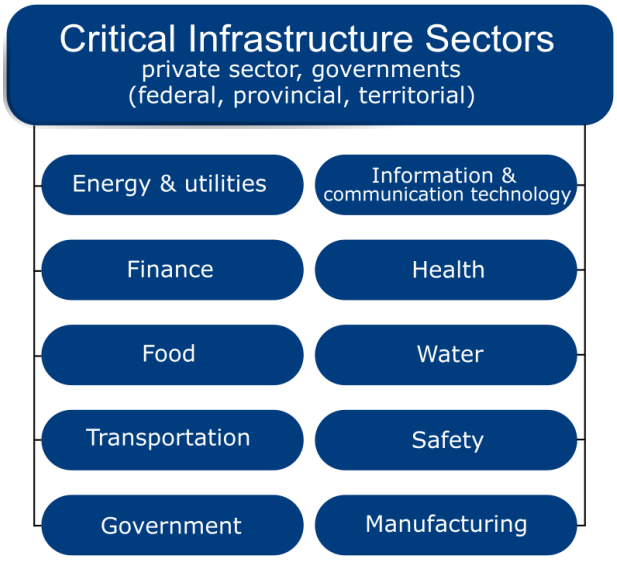


Figure 2.2 Critical Infrastructure Sectors in Canada [26]

As found in [27], all the listed nations (Canada, US, Australia, New Zealand and UK) included energy, water, transport, ICT, health, food supply, banking/finance, government services, safety/emergency services. The results of this report also list modeling & simulation, risk assessment, decision making/support, policy & directives, and monitoring & warning as mitigation strategies. In addition, the literature collected in [27] shows an increasing interest in disasters, particularly in natural disasters.

Critical Infrastructures (CIs), while defined as individual entities, don't operate in isolation. They are intrinsically connected at levels that sometimes are not observable at first glance. Those relationships can help identifying vulnerabilities in the systems as well as tracing cascading effects after incidents impact the system.

2.2.1 Related Work

Setola and Theocharidou in [29] provide a thorough review on dependencies among infrastructures. Their work characterizes these relationships based on their order; where order is determined by the proximity of the other entities affected by the incident as it propagates. The authors explain how dependencies might have loops and reciprocal influence. These dependencies are called interdependencies. Moreover, the authors describe 5 categories for interdependency analysis: physical, cyber, geographical, logical, and social.

Individual infrastructures are complex on their own. Considering that critical infrastructures are interconnected elevates the complexity of their analysis. As single infrastructures are systems and they are inherently interdependent, in conjunction, the study turns into system of

systems modeling with the aim of unveiling those hidden bidirectional relationships (interdependencies) [30].

We can see that when dealing with CIs, decisions are effective if they are not taken in isolation. Hence, for an effective coordination, an understanding on how the systems' components interact is required in order to minimize the consequences of hazardous events affecting complex scenarios [31].

2.2.1.1 Modeling of Critical Infrastructure Interdependencies

Modeling and Simulation are two important tools because they enable the planners and managing teams (practitioners) to evaluate potential risks and vulnerabilities. Many disciplines can run experiments and real-life tests. In emergency response, the recreation of events and real-life tests are not possible to the extent other professions take it. Instead, simulation tools come to aid and allow running modeled representations of the situations under study. These simulation tools need special characteristics for being suitable solutions during emergency response. They must include the interdependencies among infrastructures.

Ouyang in [32] provides a survey on Modeling and Simulation of Interdependent CIs. This overview provides a walkthrough of the inherent relationship of both terms in the context of Disaster Response. The author highlights the importance of interdependencies in disruptive scenarios, as some of the interrelations are hidden in normal conditions but made obvious under critical conditions. Likewise, the author lists the methodologies used for modeling CIs found in literature. These approaches include:

- ***Empirical:*** based on historical disaster data and expert experience. Works in identifying patterns and provides alternatives for mitigating risk.

- ***Agent based:*** models the behaviour of decision makers. Allows capturing all types of interdependencies and work with discrete-event simulations and what-if-scenarios. It can be integrated with other modeling techniques. Some weaknesses are: quality depends on assumptions made, hard to justify sometimes. Moreover, challenges may rise due to lack of data.

- ***Input/output based:*** based on a technical coefficient specifying the ratio of inputs needed to produce the necessary output. A key term, *inoperability*, is defined as the inability of the infrastructure to perform its function.

- ***Topological based:*** nodes/blocks represent CIs and links represent physical and/or relational connections. CIs are modeled with discrete states and analyzed through simulation.

Rinaldi in [30] proposes a set of important variables to be included in interdependencies modeling. These variables include: types of interdependencies, infrastructure environment and characteristics, coupling and response, types of failures, and state of operations. With these foundational variables, the author mentions agent-based modeling as a suitable technique for modeling interdependencies. Moreover, the author classifies this modeling technique as a complex adaptive systems study. Taking into account that agent-based modeling has weaknesses, the author recommends combining this approach with other simulation methods.

In [33], De Nicola, Vicoli and Villaini recognize and highlight the importance of simulation in crisis management and technical disasters. The authors introduce CEML (Crisis and Emergency Modeling Language), a behavioral modeling framework. CEML focuses on reactive behavioural descriptions where a set of actions are performed as something happens. Hence, the two key ideas are to receive "Abstract Service Stimuli" and to respond with "Abstract Service Response". While the idea is to connect the behavioural language CEML with agent-based simulators, the authors don't have a simulating platform of their own or specify a compatible one. The authors based their study on behavioural modeling but don't include topological considerations or other relevant input that allows proper mapping of the infrastructure interdependencies. Since no full cases have been solved with this methodology execution times are not available and suitability for disaster response or crisis management is hard to evaluate/verify.

Foglietta, Panzieri and Pascucci introduce the concept of holistic and reductionist simulation models in [34]. Their work highlights the importance of mixing both approaches, however, while using a holistic approach all interactions among CIs are disregarded. For proper representations of interconnected infrastructure, they proposed a two-layer architecture where an upper layer hosts the infrastructures, in isolation, and the lower layer maps their interconnection. The authors present CISIA (Critical Infrastructure Simulation by Interdependent Agents), an agent-based simulator. CISIA is said to handle both the reductionist and holistic approaches, however, not many details about the implementation are provided. The authors specify that their framework is applicable for risk management, though, it is only suited for planning stages and not for real time situations.

In [35] Aung and Watanabe discuss emergency and disaster response as applied by the government of Japan. The authors define a methodology based on IIM (the inoperability input-output model) defined by Leontief and Bayesian networks for representing the dependencies. Availability of data is said to be high as the government discloses IIM like matrices with infrastructure coefficients. These coefficients reflect the dependency of some infrastructures over the rest. Infrastructures are classified as weak if they are highly dependent and strong if they are mostly independent. The study is limited to the more independent infrastructures and Bayesian networks are duplicated because they don't allow bidirectional connections. As the study uses IIM, representation of non-linear systems is difficult or approximated. Critical infrastructures are known to be highly non-linear, this is a limiting factor in this framework. Likewise, availability of data can be scarcer in other regions, hence, extending the use of this framework seems difficult.

Tofani, D'Agostino and Martí, introduce phenomenological simulation in [36]. Phenomenological simulation offers more abstract representations to be added to models as different components (modules) interact in a system of systems fashion. Among phenomenological analysis the authors list the most common approaches used in practice. These methodologies coincide with the ones described by Ouyang in [32] and include: topological analysis, where the emphasis goes on how elements connect more than their dynamics. A second method is input and output systems, having components, like infrastructures, represented by a block with a specific input/output mapping. Likewise, the authors include agent base modelling, where the agents can provide optimizations based on expert knowledge or acquired knowledge of the environment.

2.2.2 Conclusion

As stated in the references collected, critical infrastructures need to be modeled in a way that allows representation and analysis of their interdependencies. The government of the United States, in their National Infrastructure Protection Plan, considers the understanding and unveiling and addressing risks of interdependencies. This is considered essential to enhancing critical infrastructure security and resilience [37]. Modeling critical infrastructures requires the use of combined approaches. i2Sim uses topological modeling in its physical layer, where components reflect their links in the real world. Moreover, i2Sim's toolbox uses a set of blocks input/output mappings. i2Sim can model past events and use the experience acquired to search more optimum solutions, this qualifies as empirical modeling. As a layered architecture, i2Sim enables independence of tasks. Decision making, and damage assessment activities occur independently. Agent based decision makers can coexist in i2Sim's decision layer; this thesis describes machine learning optimizations at the decision layer. The chosen simulation platform (i2Sim) combines all the modeling techniques found in the literature review done for this project. It also includes positive features found in related work, like using human factors as in CEML, system of systems analysis, and suitability for crisis management. i2Sim is a tested platform and the optimizations done in this thesis can have their efficiency evaluated in terms of execution time.

2.3 Parallel and Distributed Computing

The simulation of scientific problems in engineering continues to grow. Larger problems require more memory space and computing power. Parallel programming has become a suitable solution due to availability of HPC (High Performance Computing) clusters. Using

HPC, the computations are partitioned and assigned to parallel resources for execution [38]. A brief introduction to parallel and distributed computing follows.

2.3.1 Parallel Processing Fundamentals

Parallel architectures can be classified based on two characteristics: their communication arrangement (Table 2.1) and their data-program characteristics, also known as Flynn's Taxonomy (Table 2.2). Both classification methods are shown below:

Name	Description	Implementations
Shared Memory	Completely independent processes have their address space (or a portion) mapped to a common physical location.	OpenMP, Pthreads
Message-Passing	Based on *Send* and *Receive* calls. Data is sent as messages and operations happen locally. Most often, all nodes execute identical copies of a program, with the same code and private variables.	MPI
Data-Parallel	In these machines, a scalar processor is integrated with a collection of function units that operate on vectors of data out of one memory in a pipelined fashion.	Vector Processors, GPUs

Table 2.1 Parallel communication taxonomy [39]

Name	Description	Examples
SISD	Single-Instruction, Single-Data	Conventional sequential computers
MISD	Multiple-Instruction, Single-Data	No commercial implementation
SIMD	Single-Instruction, Multiple-Data	Vector processors
MIMD	Multiple-Instruction, Multiple-Data	Any multicore system

Table 2.2 Flynn's taxonomy

The creation of a parallel program starts with its best sequential version. It is important to mention that not every program is parallelizable. A dependency analysis of the sequential version is required in order to determine the processes that can run in parallel. For clarification,

a *task* is the minimum unit of concurrency, tasks are grouped into *processes* (also called threads) and assigned to *physical processors* [39]. The steps followed in parallelization are shown in Table 2.3 and Figure 2.3, as follows:

Step	Architecture dependent?	Description	Major Performance Goals
Decomposition	Mostly no	-Breaking up the computation into a collection of tasks.	-Expose enough concurrency, but not too much.
Assignment	Mostly no	-Specify how tasks will be distributed among processes.	-Balance workload. -Reduce communication volume.
Orchestration	Yes	-How to organize data structures and schedule tasks locally. -Define the size of messages. -How to execute interprocess communication and synchronization. -The programming language has a strong influence on this stage due to available primitives and their costs.	-Reduce non-inherent communication via data locality. -Reduce cost of comm/synch as seen by processor. -Reduce serialization of shared resources. -Schedule tasks to satisfy dependences early.
Mapping	Yes	-Most of the times binding processes to processors is done by the operating sytem or programming language.	-Put related processes on the same processor if necessary. -Exploit locality in network topology.

Table 2.3 Steps in Parallelization [39]

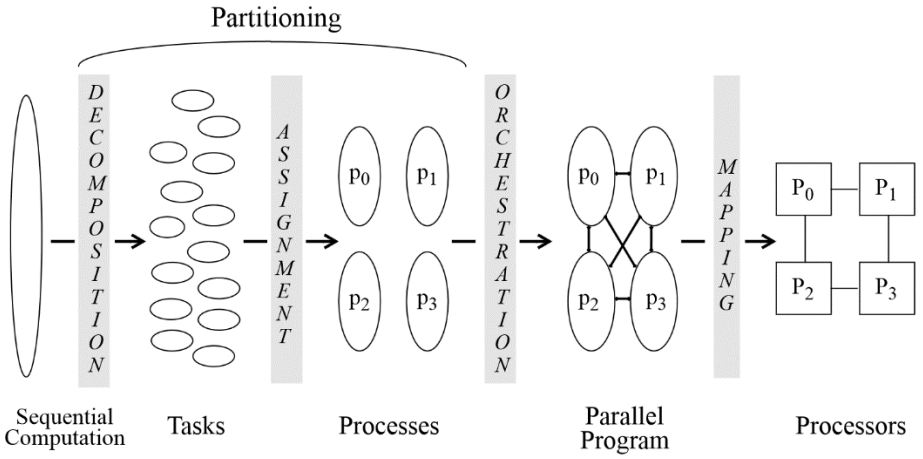


Figure 2.3 Steps in Parallelization [39]

The speed up is, ideally, measured as: $Speed\_Up = \frac{Time(Sequential)}{Time(Parallel)}$, according to Amdahl's Law [40].

2.3.2 Distributed Computing and Server Clusters

A distributed system is a network of autonomous computing nodes interacting to achieve a goal. The nodes in a distributed system are independent and do not physically share memory or processors [41]. The three fundamental characteristics of a distributed system are: *Concurrency*, *Synchronization in time* and *Handling failures* [42]. These fundamental characteristics are guaranteed, in most cases, by the operating system and the clustering software.

A *Cluster* is defined as a collection of interconnected stand-alone computers/servers working together as a single, integrated computing resource. Clusters are considered a distributed computing model for allowing their computing components to be dispersed over a large area [43]. When the nodes are geographically dispersed, and the arrangement can include different owners (probably not fully trusting one another), it is called a *Grid* instead of a Cluster.

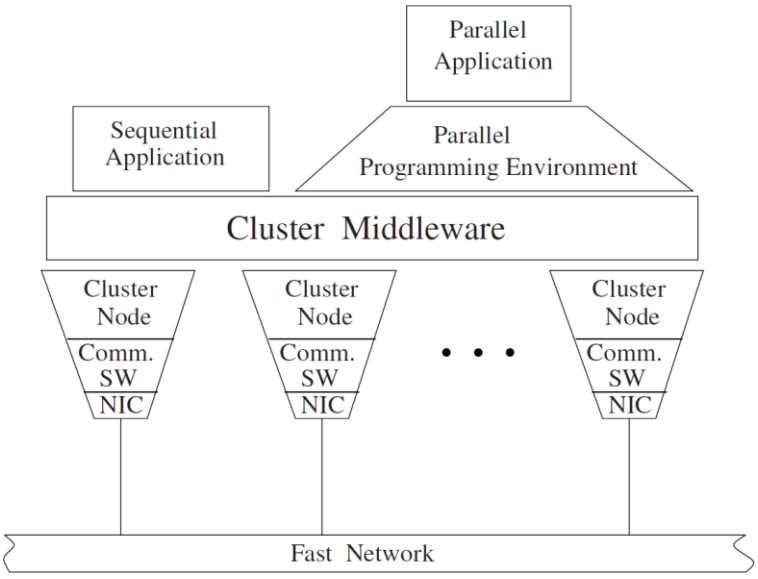


Figure 2.4 Cluster Computing Architecture [43]

2.3.3 Related Work

The related work in this section focuses on reinforcement learning (RL) implemented by means of parallel processing. Most publications on machine learning and parallel processing use classification and regression methods like neural networks. This thesis is based on RL; hence the emphasis goes on similar implementations.

2.3.3.1 Reinforcement Learning and Parallel Processing

Parallel reinforcement learning is advancing and, even though, some experimental work is available, most references remain under development and theoretically proposed work. Parallelizing RL in [44] decomposes the problem into a graph of connected subproblems that communicate via message passing. The scheduling architecture uses round robin. The implementation takes place in a multicore machine; thus, it uses shared memory and would only work for low-dimensional state spaces. Likewise, in [45] the authors provide experimental results of a parallel RL with different scheduling options. The test case is a maze solving and three approaches for scheduling are benchmarked: random scheduling, round robin and weighted priority scheduling. Both solutions are hard to generalize or to replicate.

2.4 Big Data Frameworks

2.4.1 Apache Hadoop

Hadoop is a well-known MapReduce tool created by Google [46]. Hadoop a is framework for handling large distributed tasks across clusters. Hadoop is scalable and uses the Hadoop Distributed File System (HDFS). MapReduce refers to the process of distributing the work

among nodes (Map) and collecting/consolidating results (Reduce) [47]. Hadoop's scheduling and cluster resource manager is called YARN.

2.4.1.1 Related Work

How RL can be implemented in conjunction with Hadoop is discussed in [48]. The authors show through large matrix multiplication examples, a proof of tabular forms of RL integrating with MapReduce. However, [48] does not take into consideration the random access to the look up table as opposed to well define symmetric block partitioning for matrix multiplication. The work presented in this reference is difficult to replicate. Moreover, Apache Spark, considered the successor of Hadoop, offers a better environment for machine learning applications. Therefore, no further references about Hadoop are included.

2.4.2 Apache Spark

Spark was created by the AMPLab at UC Berkeley as a data analytics cluster computing framework [46]. Spark is defined as an engine for large scale analytics. Works in Spark can be written in Java, Scala, Python, and R. Spark supports SQL and the structuring of the data is done with Datasets. In the past Spark was based on RDDs (Resilient distributed datasets) [49] but while their support remains, in newer versions (after version 2.0) it has been deprecated. Spark includes a machine learning library called Mlib. Mlib includes routines that allow classification, regression, clustering and distributed linear algebra among others. Mlib uses the linear algebra package Breeze. Spark outperforms Hadoop because it enables in-memory calculations instead of using intensive I/O as in the case of Hadoop. References on Spark's website [49] suggest than in the case of regressions Spark is 100 times faster than Hadoop.

Spark runs on multiple platforms like Hadoop's YARN, Amazon EC2 and in standalone cluster mode, among others.

2.4.2.1 Related Work

A couple of RL implementations were found using Apache Spark. The study done in [50] analyzed the effectiveness of online marketing strategies for turning website visitors into buyers. This project was implemented by *Nedim Lipka* from Adobe. The customers were represented as vectors including behavioural and contextual features (location, browser used, etc.). All the information came from recorded sessions. The results evaluated the performance of Spark compared to Hadoop, with significant outperform of Spark over Hadoop. The author explained this difference due to Hadoop not reporting intermediate results without going back to HDFS (the file system used by Hadoop), thus generating I/O overhead. Spark, on the other hand, allows in memory processing. This study uses a very small state space, 9,496 states, and the author focuses on comparing Spark with Hadoop. Moreover, the author uses Java Hashmaps for his implementation so changes in size can severely impact performance. Execution times are provided, and they will be evaluated against our first solution in Chapter 4.

The second implementation of RL over Spark simulates Electricity Market Bidding [51]. This "work in progress" highlights the virtues of Spark over Hadoop. In this study, a master agent interacts with electricity sellers, also agents, and decides the best strategy for electrical allocation one day in advance. One fact that does not seem accurate, mentioned during the presentation of the paper in the Spark Summit 2014, was having all rewards positive. If no negative rewards are included, the agent would not be able to determine which actions are bad

and, therefore, the convergence to a policy would be compromised. No size/performance data is provided by the authors. Hence, suitability for use in real-time applications is verified.

2.4.3 Conclusion

The solution this thesis proposes is a consolidated framework for emergency response decision making. Thus, critical infrastructure interdependencies (CII) analysis is vital. Likewise, the decision-making agent must be capable of finding the optimum outcomes in a matter of a few minutes; otherwise, the framework is not apt for helping in real-time situations. Adding parallel processing capabilities improves the solution in two fronts. First, dimensionality constraints can be eased or even removed (in some cases) if the formulation uses distributed architectures. Second, having multiple scenarios running, simultaneously, enables responder to evaluate multiple possibilities without increasing the timespan. Related work shows that when using machine learning in emergency response applications, most of the work uses classification/regression techniques by means of neural networks. Hence, the policy is predefined and new samples are placed accordingly. On the other hand, reinforcement learning (RL) offers learning by experience. Thus, new paths can be explored and new policies can be learned.

In consideration of ways to put RL to work in a parallel/distributed execution, based on the literature and related work review done, we narrowed down our options to three: using Apache Spark, using MATLAB's distributed computer server (annual license), or to write our own scheduler. Using Apache Spark seems reasonable due to its cluster optimization routines and machine learning library, Mlib. However, the library seems better suited for classification techniques and not as much for RL. The two applications mentioned earlier in this chapter

show the possibility of using configuring RL processes in Spark. However, their applications are lightweight and independent from any platform. As we aim at configuring an RL agent working in conjunction with i2Sim, but using i2Sim's clock, using Spark is not viable for two reasons. First, the agent is expected to exchange data with physical layers at every time step, producing a high-volume message passing leading to communication overhead. Second, to avoid the communication overhead, an alternative is to embed i2Sim as part of a Spark solution. Weaknesses in our past projects were introduced when they were handled using the criteria above. As this project seeks improvement over those past projects [9] [6] [7] [52], this alternative is discarded.

The use of MATLAB's distributed computing server is a way of unlocking the parallel toolbox and move MATLAB/Simulink projects to a cluster. An initial limitation is the cost. This service is offered as an annual subscription of about $2000 CAD. The second consideration is how it would work. The improvement would consist of an adapted MPI (message passing interface), this enables a distribution of the look up table but wouldn't help with running simultaneous instances. With the infeasibility of this second option, according to our judgment, we decided to pursue our own distributed scheduler implementation.

The Scheduler written for this thesis follows a similar approach to the one used by message passing systems. Hence, it falls under the Multiple-Instruction, Multiple-Data (MIMD) category of Flynn's Taxonomy.

Chapter 3: Modeling Framework Description

The simulation scenarios, part of this thesis, have been modeled using i2Sim, the Infrastructure Interdependencies Simulator. The decision-making is approached with reinforcement learning (RL). By using RL, an agent can learn the best outcomes for different configurations of the environment without previous knowledge. It is convenient to provide a concise review of i2Sim and RL, to create the necessary context for understanding how the simulation cases were assembled, configured and solved. The final part of the chapter will describe the computing infrastructure (hardware/software) used for running the test cases.

3.1 The Infrastructure Interdependencies Simulator (i2Sim)

The Infrastructure Interdependencies Simulator (i2Sim) is a simulation framework developed at the Complex Systems Integration Centre of the University of British Columbia. It is the continuation of The Joint Infrastructure Interdependencies Research Program (JIIRP), sponsored in 2005 by Public Safety and Emergency Preparedness Canada (now Public Safety Canada) and the Natural Sciences and Engineering Research Council (NSERC) [5].

I2Sim allows the representation of interconnected critical infrastructures and the modeling of their interdependencies. In addition, through the simulation process some hidden interdependencies can be revealed [53]. To understand how i2Sim works, we provide an overview of its most significant features. We start by describing i2Sim's layered architecture, its advantages, and member layers. We continue by describing the components' ontology, a functional grouping of the components in the physical layer. We close i2Sim's overview by discussing the two key components for this project: the production cell, and the distributor. A

new type of distributor has been added to the i2Sim's toolbox, as part of the contributions of this thesis, the integer distributor. The integer distributor reacts to splitting ratios in a less strict manner as it can only output integer values. A proper description of the integer distributor is provided later in this chapter.

3.1.1 Layered Architecture

i2Sim is a flexible environment with a stack of connected layers. This approach enables individual layers to have an independent functionality from the other layers. The foundational layers operate fully in the i2Sim environment while the supporting layers can be specified using internal routines or via APIs when connecting to external applications. This level of abstraction promotes independence of tasks, where each layer deals only with its specific role while treating the remaining layers as black boxes. Points of communication between layers are limited to gain control over their message exchange. The list of layers and their descriptions are provided below.

3.1.1.1 Low Level Layer (Specific Domain Simulators)

This layer refers to specialized tools outside i2Sim. The low level layer manages ways of interacting with specific domain simulators e.g., electrical simulators like MicroTran or PSCAD, among others. In previous projects we developed adaptors that enabled bidirectional communication between i2Sim and third-party applications.

3.1.1.2 Physical Layer

This layer operates fully within the i2Sim environment and provides the communication points for interacting with all the other layers. It provides the user with a series of blocks for representing the physical infrastructures and their topological connections. This layer performs

physical infrastructure interdependency evaluation and clock synchronization. The layer follows an ontology that uses a unified set of components capable of representing a diversity of physical disaster response scenarios.

3.1.1.3 Damage Assessment Layer

This layer is capable of interacting with external sensor data via APIs. The main job of this layer is to perform damage assessment over physical infrastructures. It connects to the physical layer, and pushes updates containing the status of the physical simulated components. The damage assessment layer can also act as a bridge between the physical layer and specialized damage assessment tools like BCSims [54].

3.1.1.4 Decision layer

This layer has a bidirectional channel of communication with the physical layer. These connection points allow this layer to capture the instantaneous state of the physical components. An agent or a group of agents analyze the obtained data and choose the best course of action, that aligns with the goals of the scenario under study. The main task of decision makers, in disrupted scenarios, is to allocate resources. The output signals, from this layer to the physical, are optimized distributions of shared resources. These combinations are treated as a set of ratios transferred to a component called the distributor. While other elements could be managed from the decision layer, the most common optimization relates to resource allocation and the distributor is the only point of contact.

An illustration of the layered architecture is displayed as follows:

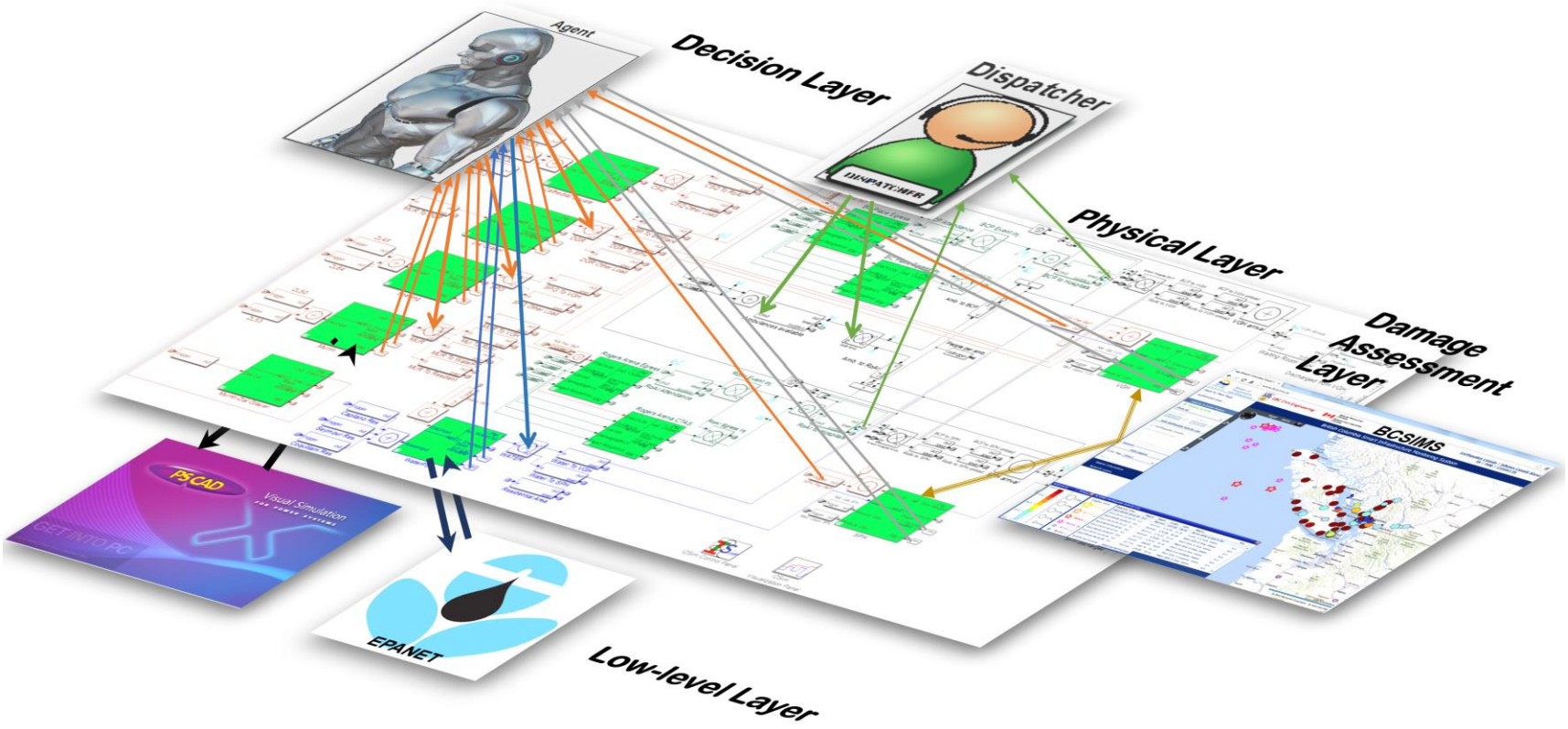


Figure 3.1. i2Sim's Layered Architecture

The decision layer uses different algorithms, among which, Artificial Intelligence (AI) agents are a preferred choice. Our past work has evaluated multiple approaches for the decision layer; including: Monte Carlo, reinforcement learning (RL), ordinal optimization and genetic algorithms. However, RL seems to adapt better to the simulator's implementation due to the high level of discretization natural to the representation of most components. RL, in its basic form, uses a matrix-based look-up table (LUT). The LUT maps states to actions and serves as the storage for the agent's knowledge. A large environment needs many state descriptors, for that reason, dimensionality is a big concern. As more infrastructures are added to the model, the dimensionality could go beyond the limits of possible representation. The Distributed RL agent developed for this thesis, is an alternative for mitigating this *curse of dimensionality*. Moreover, the agent's enhanced implementation addresses ways of achieving a faster training.

3.1.2 Ontology

i2Sim's ontology was created to provide the right level of generalization that different scenarios and infrastructures require. This ontology applies solely to the physical layer and its components. The ontology comprises a simplified set of universal components used to represent multiple real objects [5] [55]. The ontology groups the components, into categories, by their affinity.

3.1.2.1 Tokens

The tokens are the units that flow through the system. A token is a quantity that is transferred from one component to another. Hence, tokens are the inputs and outputs of every component and always travel through a channel. Tokens don't belong to a specific type, they are just values. For modelling purposes, tokens are labeled inside i2Sim's components. These labels are informational and may include units.

3.1.2.2 Cells

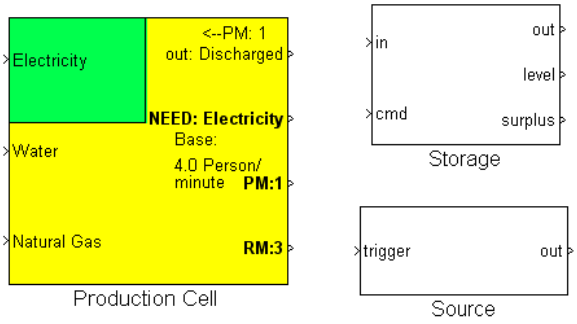


Figure 3.2 The i2Sim Cells

Cells are the production units. The cells have the power of creating, transforming and storing tokens. The three available cells are: sources, storages and productions cells (PC). Sources produce an output without inputs, they act as generators. Sources are commonly used to feed PCs and to provide values to simulation/element parameters. Storages are reservoirs, their input and output tokens are of the same type. Storages are frequently used to model hospital waiting, venue seating areas and other variable accumulators. Production cells are a key component, they are usually mapped to infrastructures. Their output tokens may be of the same nature of what goes in their inputs or a transformation of combined heterogeneous resources. The PC is covered in more detail ahead in this chapter.

3.1.2.3 Channels

Figure 3.3 The i2Sim Channel

Channels are means to transport tokens from element to element. Channels have a single parameter, a time delay. The time delay is the amount of time tokens remain inside the channel.

This way, complex interconnections like roads, pipes, etc., can be expressed based on the time it takes for tokens to travel from their origin to their destination. As a recommended practice, every link between two elements must be implemented using a channel. Otherwise, the simulation engine may stop the execution with an exception or error.

3.1.2.4 Control Elements

The control elements are those allowing the user to establish decision points (distributor), tuning the simulation parameters, and controlling the simulation engine (control panel). i2Sim's control panel is shown in Figure 3.4.

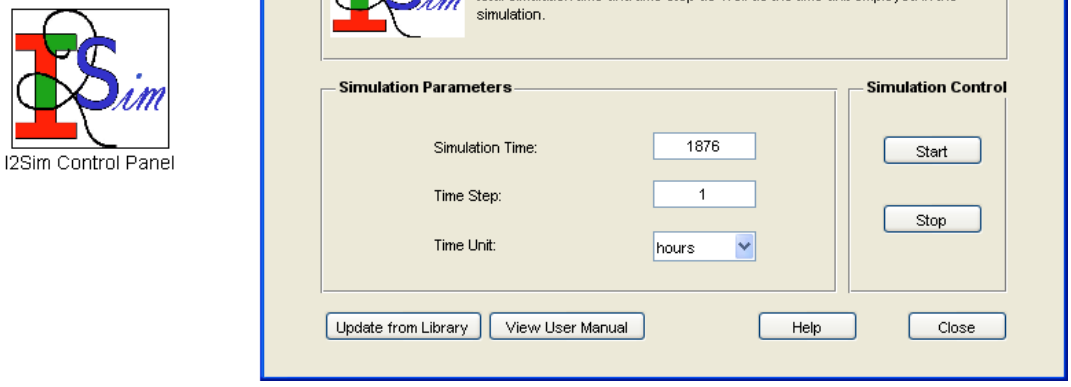


Figure 3.4 The i2Sim Control Panel. Element's Block and GUI

The control panel is mandatory in every simulation running with all graphical aids. This element is the clock of the engine and sets the right simulation parameters for i2Sim's execution. However, in GUI-less operation this functionality is provisioned by manipulating the engine's parameters directly. This is applicable to situations like simulation in the loop, where manual interaction is minimum.

Decision points relate to resource allocation and distribution, this functionality is provided by the “Distributor”. Distributors are the point of contact between the physical layer and the agents in the decision layer. A complete description of the Distributor is available later in this chapter.

3.1.2.5 Visualization Elements

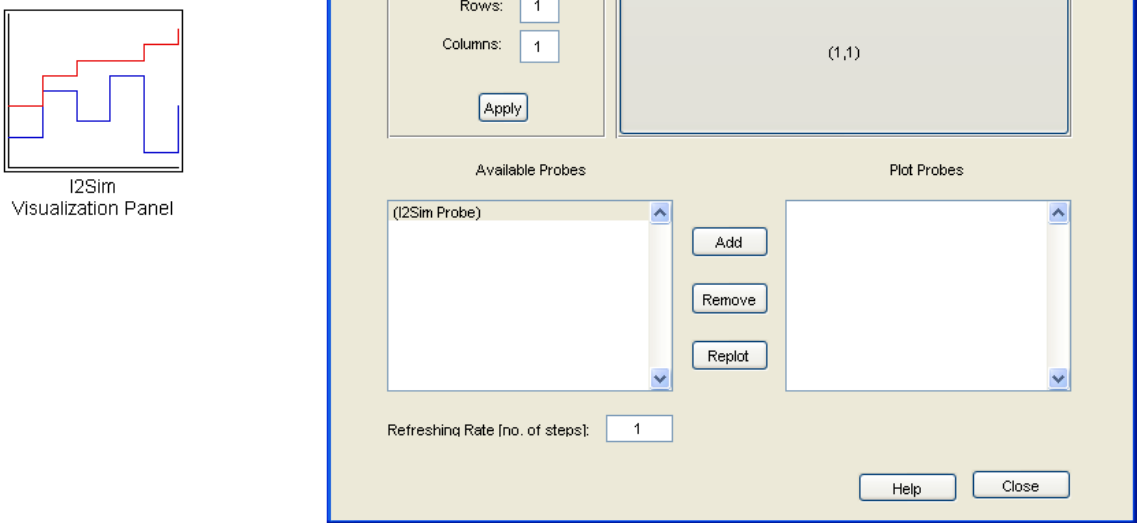


Figure 3.5 The i2Sim Visualization Panel’s Block and GUI

Visualization elements allow the user to probe specific system outputs and to present the results on two-dimensional axes against time. Results may be presented as a matrix of individual or overlapped plots. Figure 3.5 displays the visualization panel’s GUI. This intuitive interface enables the user to decide how to split the graphical space and how to assign signals to each

section. For signals to appear in the list of available probes they need to be attached to an i2Sim probe. The i2Sim probe element is shown in Figure 3.6.

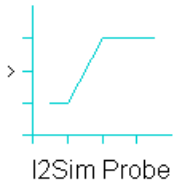


Figure 3.6 The i2Sim Probe

When the visualization panel is used, at least one i2Sim probe must be connected in the model and referenced in the visualization panel. Fail to comply with this requirement will trigger an error and will keep the simulation from running.

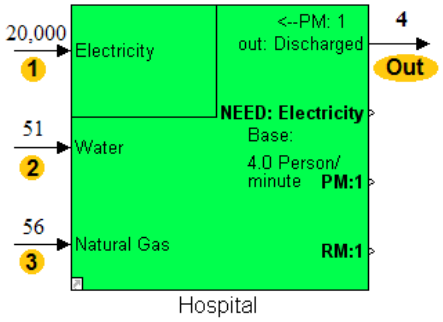
3.1.3 The Production Cell (PC)

The Production Cell is a component in the i2Sim library used to represent the CIs as input/output models. It is implemented as a block and the input/output mapping is commonly done via a Human Readable Table (HRT). The HRT table can be easily generated by the users and enables them to model the behaviour of complex infrastructures which are highly nonlinear. HRTs, as defined in i2Sim, have a number of inputs and a single output. The inputs may correspond to resources needed for the operation of the infrastructure, but they could also be *"modifiers"*. Modifiers are a special type of inputs that allow the consideration of human factors that affect the performance of the cell, e.g., medical personal fatigue.

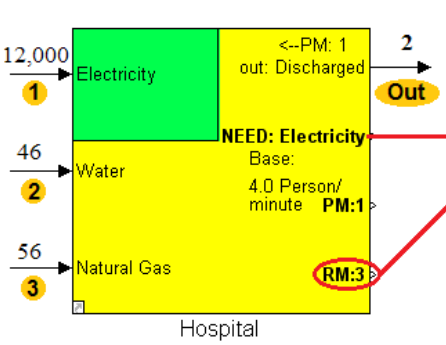
The PC's operability modes are defined by two variables: Physical Mode (PM) and Resource Mode (RM). The physical mode represents the physical integrity of the facilities, i.e., their

capability to operate. The RM is an instantaneous specification of the level of input resources available. The input (resource) with the lowest value is called the *limiting factor* as it determines the output. Figure 3.7 illustrates the concept of the limiting factor and the changes made to the infrastructure's output when the inputs are changed.

The production cell displays the PM as a coloured rectangle in the upper-left corner, and as a value in its third output port. The resource mode is reported as the solid color filling the cell itself, and its value is passed to the fourth output port. All these details are visible in Figure 3.7, details about colour mapping and combined PM/RM operability are treated later in this chapter.



	Output	Inputs		
	Discharged [Person/Minute]	Electricity [kW]	Water [kL/minute]	Natural Gas [ft^3/minute]
1	*4* Out	20,000 1	51 2	56 3
2	*3*	15,000	38	42
3	*2*	10,000	26	28
4	*1*	5,000	13	14
5	*0*	0	0	0



Limiting factor

	Discharged [Person/Minute]	Electricity [kW]	Water [kL/minute]	Natural Gas [ft^3/minute]
1	*4*	20,000	51	56 3
2	*3*	15,000	38 2	42
3	*2* Out	10,000 1	26	28
4	*1*	5,000	13	14
5	*0*	0	0	0

Figure 3.7 Operation of the Production Cell Based on a Human Readable Table (HRT)

The HRT concept incorporates a dimensional reduction analysis by using the limiting factor. The inputs/output mapping turns into a step function, a *floor function* [56], in this case.

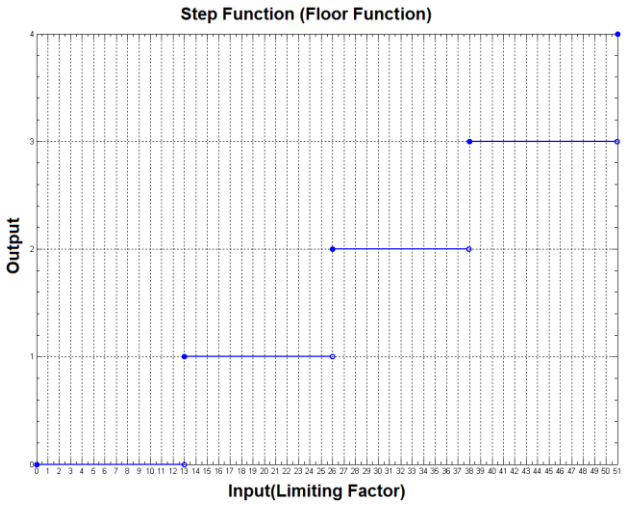


Figure 3.8 Production Cell's Output with an HRT

When having multiple input resources, equivalent to multivariable functions, the dependant variable (output), resides on the *hyperplane* delimited by the limiting factor. The discrete thresholds provided by the HRT are convenient and adequate for RL implementations, but the PC is not limited to HRTs. Different mappings are acceptable and can be configured as the application requires. An easy option for turning the HRT into a continuous function is interpolation. For a previous project in energy balancing [55], the PC's output was turned into a piecewise linear function, as seen in Figure 3.9.

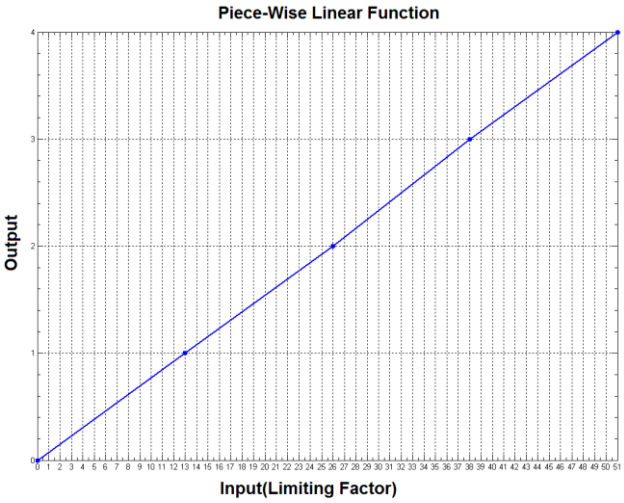


Figure 3.9 Piecewise Linear Interpolation of HRT

The PC belongs to the physical layer and it offers connectivity with upper layers. While input resources come from the same layer, the physical mode is provided by the damage assessment layer. Likewise, the current PM and RM values are sent to the decision layer, as state variables, for decision agents to determine the status of the environment.

3.1.4 The Distributor

The distributor is an element that splits an input signal into two or more output signals. The process is done by assigning a ratio to each one of the outputs. The sum of all ratios is 1 (100%) i.e., the value of the input. The distributor's behaviour can be fixed or controlled by an agent from the decision layer. With a fixed behaviour, the distributor has a permanent set of ratios throughout the simulation. When an agent controls a distributor, a message from the decision layer brings the instantaneous configuration of the distributor, the settings remain until new ones are provided.

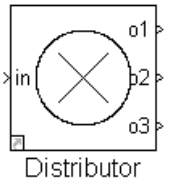


Figure 3.10 The i2Sim Distributor

3.1.4.1 The Integer Distributor

This new distributor was added to the i2Sim's toolbox for this project. The need for this element arose when implementing the emergency transportation subsystem. To create a realistic representation of an emergency scenario we could not allow fractions of ambulances traveling to different destinations. This distributor follows the split ratios as a proportional

estimation not as an exact numerical value. To illustrate the operation of the integer distributor let's assume an input 1 to be distributed between two outputs. Table 3.1 summarizes the results.

Input	Ratios		Regular Distributor		Integer Distributor	
	R1	R2	Output 1	Output 2	Output 1	Output 2
1	0%	100%	0.0	1.0	0	1
1	20%	80%	0.2	0.8	0	1
1	40%	60%	0.4	0.6	0	1
1	60%	40%	0.6	0.4	1	0
1	80%	20%	0.8	0.2	1	0
1	100%	0%	1.0	1.0	1	0

Table 3.1 Regular Distributor vs. Integer Distributor (Constant Input 1)

To achieve these outputs, the integer distributor calculates the regular values and applies a ceiling function to the output with the highest ratio and a floor function to the lowest. In case of a 50% - 50% distribution, the priority is given to the first output. While the previous example illustrates only one input token to be distributed, the combination of the ceiling and floor functions guarantees proper operation. This component was tested extensively over multiple setups. Table 3.2 shows a set of sample distributions with an input different from 1 and compares the outputs produced by both types of distributors:

Input	Ratios		Regular Distributor		Integer Distributor	
	R1	R2	Output 1	Output 2	Output 1	Output 2
5	75%	25%	3.8	1.3	4	1
4	75%	25%	3.0	1.0	3	1
15	75%	25%	11.3	3.8	12	3
52	22%	78%	11.4	40.6	11	41
23	75%	25%	17.3	5.8	18	5
9	75%	25%	6.8	2.3	7	2
2	51%	49%	1.02	0.98	2	0
2	35%	65%	0.7	1.3	0	2

Table 3.2 Regular Distributor vs. Integer Distributor (Input varies)

The most common case for the integer distributor in our modeling is the two output. However, having more than two outputs is possible. For more than two outputs, the distributor uses a ceiling function for the $N-1$ highest outputs and calculates the lowest output as the difference between the input and the already calculated outputs.

3.2 Reinforcement Learning

"Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence" [25]. When this idea is put in the context of artificial intelligence and a goal-oriented interactive learning is taken, it is called *Reinforcement Learning* (RL). Reinforcement Learning belongs to the Unsupervised Learning branch of Machine Learning.

"In Reinforcement learning, an agent wanders in an unknown environment and tries to maximize its long-term return by performing actions and receiving rewards" [57]. The basic idea of RL is to have a learner (agent) mapping situations (states) to actions by interacting with the environment. The challenge for the agent is to choose the action that generates the highest feedback signal for any specific state. Actions taken at any point may affect future situations. Therefore, future rewards (delayed rewards) are assigned. Delayed rewards and trial and error are two factors that differentiate RL from other Machine Learning approaches [25].

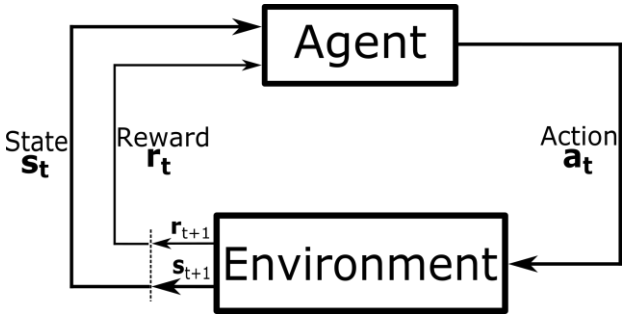


Figure 3.11 Agent-Environment Interaction in RL [25]

Reinforcement Learning is a Markov Decision Process (MDP). A formal definition of MDPs is provided next. For this thesis, only relevant RL topics are to be defined. Hence, all definitions will stay in the context on this research and all model formulations will follow the same structure.

3.2.1 Markov Decision Processes (MDPs)

A Markov Decision Process (MDP) is a tuple $\langle S, A, T, R \rangle$ in which S is a finite set of states, A a finite set of actions, T a transition function defined as $T: S \times A \times S \rightarrow [0,1]$ and R a reward function defined as $R: S \times A \times S \rightarrow \mathbb{R}$ [58].

3.2.1.1 States

A state is a unique characterization of the status of the environment. States are defined by a combination of significant features from the environment. States are a finite set $\{s_1, .., s_N\}$ where N is the size of the set, hence the size of the state space. Some Markov Decision Processes (MDPs) can have states that are invalid due to environmental constraints. For the work produced in this thesis, all states $s \in S$ are considered legal and S is defined as a discrete state set.

3.2.1.2 Actions

An action is what the agent is allowed to do at any point in time. Actions are a finite set $A = \{a_1, .., a_K\}$ where K is the size of the action space. The set of actions that can be performed in a specific state $s \in S$ is denoted $A(s)$. For this project $A(s) = A$, hence, all actions apply to all states.

3.2.1.3 The Transition Function

The transition function defines the probability on ending in state s' when action a is applied in the current state s. The transition function is defined as $T: S \times A \times S \rightarrow [0,1]$. In m*odel-free* implementations there is no knowledge about state transition. As transition representation is not possible in a mathematical form [59], transitions are led by exploration.

3.2.1.4 The Reward Function

The reward function implicitly specifies the goal of the learning process. The reward function calculates the numerical signal sent to the agent after performing an action a in a state s. For *model-free* algorithms the reward function is defined as $R: S \times A \times S \rightarrow \mathbb{R}$, or $R(s, a, s')$. In this case, the reward signal is calculated based on the transition from the initial state s to the resulting state s' after applying action a.

"This decision rule is said to be Markovian (memoryless) because it depends on previous system states and actions only through the current state of the system" [60]. This is called the Markov Property. In Reinforcement Learning, the *state signal* has the Markov Property if the environment's response at time $t + 1$ depends only on the state and action at step t. In simpler words, all information needed to make a decision is provided by the current state without the need of using history terms. This does not mean that the agent's experience is lost or disregarded, instead it means that it accumulates step by step. Thus, a look at the previous timestep provides the lessons learned in all the preceding interactions. This property has been addressed as *Independence of Path*, as well. With independence of path (Markov property), the dynamics of the environment is defined by $Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\}$ and the

environment is said to be one-step dynamics. Hence, the agent is able to predict the next state and reward by using only the current state and action [25].

As mentioned before, the goal of any MDP is to maximize the return. Therefore, the optimality of the model is based on collecting the best rewards along the timeline. The future can be considered in different ways, for this thesis, only the *discounted - finite horizon* model is reviewed. Following this approach, future rewards are considered but they are discounted depending on how far ahead they are. Thus, rewards obtained later are discounted more than the ones received earlier. The estimated return is discounted infinite horizon is given by:

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

The formula above introduces the discount factor, γ. The discount factor is a parameter ranging values from 0 to 1, as $0 \leq \gamma < 1$. When the agent does not care about future rewards, γ is set to 0 and the agent is said to be *myopic* [58].

3.2.2 Policies

A *Policy,* contingency plan, plan, or strategy specifies the decision rule to be used at every time step. *"A policy provides the decision maker with a prescription for choosing this action in any possible future state"* [60]. The policy is the mapping between states and actions, thus, it is defined as a function that takes state $s \in S$ as an input and outputs an action $a \in A$. There must exist an output value associated to each state. The policy is denoted as π. Deterministic policies are defined as: $\pi: S \rightarrow A$ *(Function* π *maps the set S into the set A)*. The scope of this study deals only with deterministic policies.

3.2.3 State-Action Value Function (Bellman's Equation)

"Value functions are the functions of a state, which determine how good the state is and how beneficial a particular action is in the current state" [59]. How good refers to the optimality criterion, which depends on the expected return. The value function is displayed below:

$$V^{\pi}(s) = E_{\pi}(R_t|s_t = s) = E_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s\right)$$

This value function estimates the expected return when starting in state s and following policy π. Similarly, we can define the value of taking an action a in a state s and following the policy π; this is denoted as $Q^{\pi}(s, a)$. In this case, it is called *Action-Value Function* or *Q Function*, as seen below [25]:

$$Q^{\pi}(s, a) = E_{\pi}(R_t|s_t = s, a_t = a) = E_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,|s_t = s, a_t = a\right)$$

Form the previous formula we can see how the expected return, while following the policy, is accumulated. This expected return is dependent on the current state and action, so that it determines the *Q-values* for each pair state/action.

The optimal Q function $Q^*(s, a)$ provides maximal values in all states and is determined by solving the following *Bellman's equation* [61]:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a')$$

Then, the optimal policy π^* is $\pi^*(s) = argmax_a Q^*(s, a)$. Hence, the optimal policy is based on choosing the action that yields the maximum Q-value for each state.

Q functions allow the agent to learn the optimal policy by exploring the environment rather than having to know its full dynamics. This case is called *model-free*, where the model represents the knowledge about state transitioning represented in mathematical form (Transition Function) [59]. The idea of exploring the environment opens one of the biggest discussions in Reinforcement Learning, the *exploration-exploitation dilemma*. Exploring refers to trying random actions, while exploiting means using the already known "good" actions. A good balance between exploration and exploitation is needed for achieving optimal solutions.

In the absence of exploration, the agent might get stuck in suboptimal solutions (local maxima) achieved in the developing policy. Exploration opens the opportunity of discovering new paths leading to a global maximum. A good rule to balance exploration and exploitation is the $\in -greedy$ approach, where $\in$ is the exploration factor, e.g. $\in = 0.1$, means that 10% of the movements are exploratory while 90% are greedy, i.e. they choose the highest value. It is recommended to diminish $\in$ as the training process progresses.

Reinforcement Learning can be solved with three different algorithms: Dynamic Programing (DP), Monte Carlo (MC) or Temporal Difference (TD) methods. DP bases new knowledge on previously learned estimates, but it requires a full model of the environment's dynamics. MC does not need a full model, but the updates happen only until the end of every episode. TD combines their strengths. Updates are done online and a full model of the environment (including all the dynamics) is not needed. As emergency response simulations include a diversity of infrastructures, a complete model of the environment is nearly impossible to construct. Likewise, making decisions under pressing conditions, with lives at risk, requires fast answers. Considering these two factors, we decide to use a TD method for all the models included in this thesis.

3.2.4 Temporal Difference Methods

Temporal Difference (TD) methods are implemented as *on-line, fully incremental* methods, i.e., updates take place at $t + 1$ (at every timestep) during the same episode. Every update is based on what happened during the previous point in time (t). This is known as *Bootstrapping* [25]. The most popular TD methods are *Sarsa-On Policy* and *Q-Learning*. We chose Sarsa-On policy due to his safe guaranteed convergence over Q-Learning.

3.2.5 Sarsa On-Policy

Sarsa On-Policy is a TD(0) estimation method. As a TD method, Sarsa updates as *sample backups*. Sample backups involve looking ahead for a sample successor state-action pair. With that future value and the reward Sarsa computes a *backed-up value* for the original state [25]. Sarsa is said to be a TD(0) because it considers only the next ($t + 1$) state-action pair. Thus, the update uses the quintuple $\langle \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{r}_{t+1}, \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1} \rangle$, which gives the name to the method. In this quintuple we observe the current state and action being used to determine the next state/reward. The recursive Q-value calculation for Sarsa on-policy looks like this:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

The previous equation introduces a new variable, the learning rate α. The learning rate determines how much of the arriving knowledge you add to what you already know. The learning rate takes values from 0 to 1, where 0 means no learning and 1 means keeps only new lessons. In [8] we did extensive testing over the learning parameters and found that $\alpha = 0.5$ and $\gamma = 0.7$ are an excellent combination, thus, they will be kept constant in all our cases. Sarsa is an On-Policy TD estimation method because the future sampling is done following the current policy. A demonstration of Sarsa(0) convergence is available in [62].

3.2.6 Shaping Rewards

For having a reasonable time for Reinforcement Algorithms, a well-chosen reward function is key. To achieve a faster optimal policy finding, additional rewards can be given to the agent as hints. These rewards are called *Shaping Rewards* and require a lot of trial and error. Poorly chosen rewards will lead the agent to learn poor solutions [63]. Poor choices for additional rewards could impair the agent. Shaping functions are more common to multi-criteria Reinforcement Learning, but equally applicable to single goaled implementations.

A *Shaping Reward Function* behaves in the same way as the reward function defined before does: $F: S \times A \times S \rightarrow \mathbb{R}$, and it is added to the original return. The new reward function will be: $R' = R + F = R(s, a, s') + F(s, a, s')$ [63]. In many cases this shaping is based on experimental work.

3.3 Computing Infrastructure

Our research lab is equipped with an IBM High Performance Computing (HPC) cluster. In this arrangement of servers with have 24 computing nodes. The computing nodes are used for processing distributed tasks. Besides, the cluster hosts 4 management servers. Management servers play a specific role, like database management or node administration. All interaction with the computing nodes is done only through the head node, one of the four management nodes. Every node (computing and management) in the cluster runs RedHat Enterprise Linux (RHEL). The head node, as a single point of remote access, needs an administration software for managing all the other nodes. This software is known as a clustering software. Our cluster is equipped with xCAT. More details about the cluster's specifications are available in Figure 3.12.

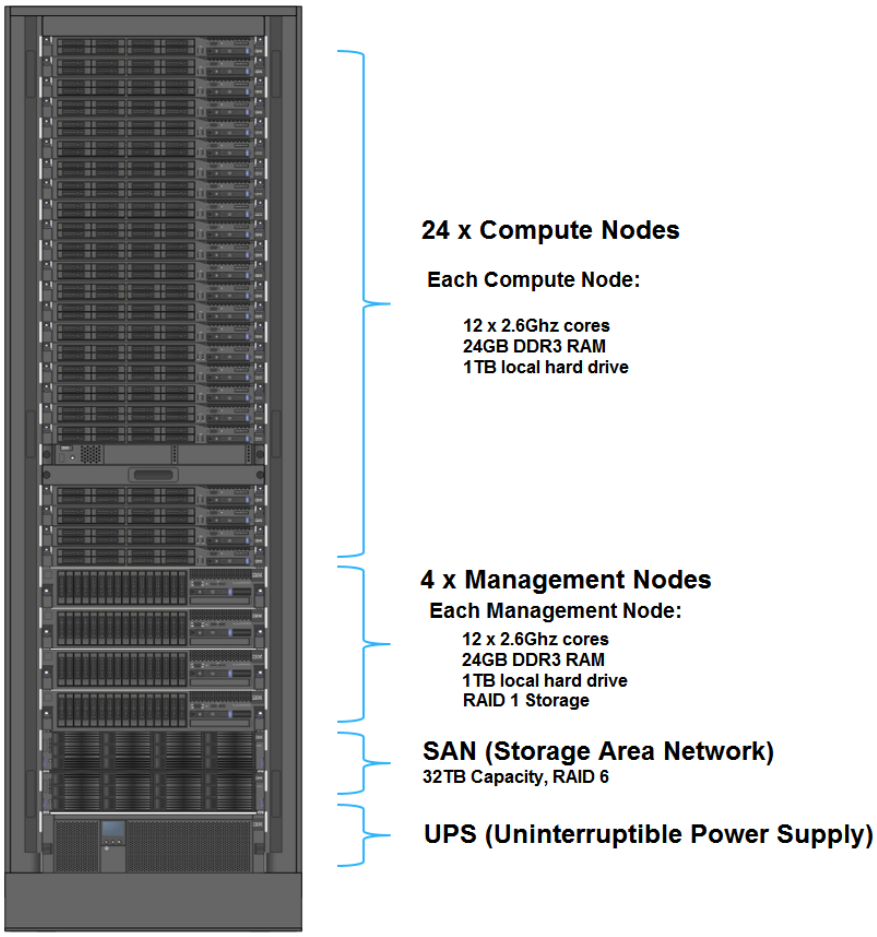


Figure 3.12 Cluster Specifications

3.3.1 Extreme Cluster/Cloud Administration Toolkit, xCAT

xCAT is a successful cluster administration package developed by IBM in 1999 and turned into open source in 2007 [64]. xCAT stands for Extreme Cluster/Cloud Administration Toolkit. xCAT is a framework that allows management of HPC clusters, datacenters, and clouds among others. It is based on best practices for systems administration. xCAT enables the system administrator to provision operating systems in both physical and virtual machines, remotely. Additional capabilities include remote systems' management, distributed shell support and quick configuration of node services: DNS, HTTP, NFS, etc. [65]. xCAT allows the systems'

administrator to aggregate nodes into groups for easy referencing. As an example of the available commands in xCAT it is worth to mention *xdcp* and *psh*. *"**xdcp** - Concurrently copies files to or from multiple nodes. In addition, provides an option to use rsync to update the files on the nodes, or to an installation image on the local node"* [66]. "***psh** is a utility used to run a command across a list of nodes in parallel"* [67].

For minimizing communication overhead, the distributed implementation, will be divided between the clustering software and the simulation platform.

Chapter 4: Aggregate Test Case

This compact model has been used in previous projects as a foundational test case. This model is an aggregate version of the City of Vancouver model used in [6] [5] [68]. For this test model, similar infrastructures have been grouped e.g. multiple electrical substations can be treated as a single electrical supply. This level of resolution is enough to represent the basic dynamics of the selected scenario. This model has been used in a similar study [8]. This past project will be used as reference for verifying the improvements achieved in this thesis.

When compacting the original model to an aggregate, several assumptions were made. The aim was to minimize the representation complexity while keeping the basic functionality. With i2Sim, modeling flexibility enables the user to add a specific level of detail to scenarios and/or subsystems. Certain subsystems can include a more detailed construction, proportional to their impact on the study's objective.

This model includes four interconnected critical infrastructures: an electrical substation, a water pumping station, one stadium (venue), and a hospital. Electricity is supplied to all other infrastructures, water is supplied to the stadium, the hospital, and residential areas. The model is shown in Figure 4.1.

This scenario assumes a disruptive event affecting the stadium and producing injuries to a total of 3000 people. These people require medical attention at a hospital. The simulation starts when the disruptive event has already ended. With this assumption, the optimization focuses on resource allocation, during the response stage of the emergency management process [15].

For this reason, most of the simulation span is devoted to finding the best resource allocation (electricity and water) for maximizing the output (treatment) at the hospital.

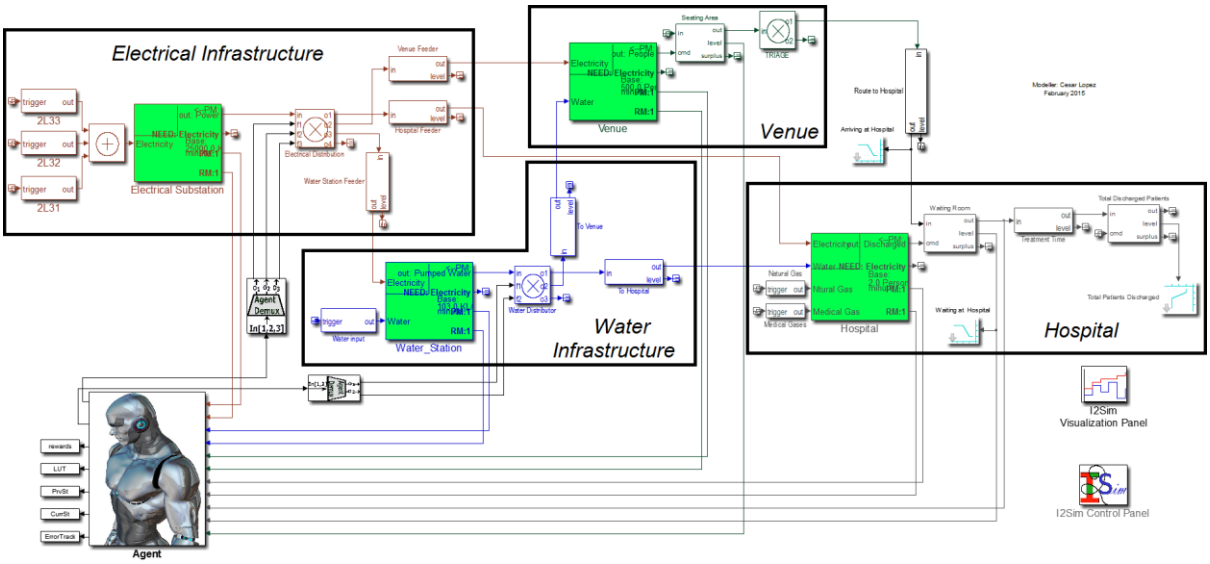


Figure 4.1 Aggregate Test Case

In some of our previous projects we focused on earthquakes [68] [5] [53]. For this study, we give no specification about the type of disaster. We feel that training the agent under multiple configurations, in parallel, opens the opportunity of learning multiple policies. If every different configuration could be understood as a different scenario, over the same model representation; the agent could undergo training, simultaneously, for a variety of incidents.

4.1 Structure of the Model

To better understand the functionality embodied by the model, the next section covers significant details of this implementation. Every infrastructure subsystem, from the four modeled, is described in regard to its functionality. Additional details about the model or the modeling framework will be provided, when necessary.

4.1.1 Electrical Infrastructure

As observed in Figure 4.2 (from left to right), this infrastructure is modeled with: three sources, one production cell (PC), one distributor and three channels. The three sources represent three lines coming from transmission into the substation, they aggregate as a single input to the PC. The production cell is the substation itself with an HRT as a driver. This HRT allows evaluation of the integrity of the substation. The channels mimic the outgoing feeders. The distributor simulates the switching gear. Hence, it has three outputs, one connected to each feeder.

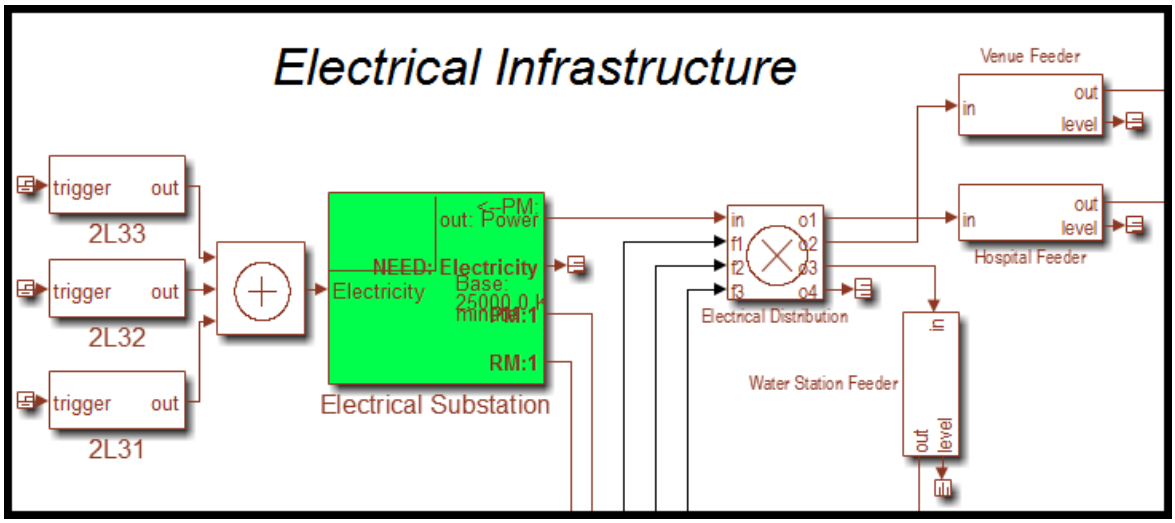


Figure 4.2 Electrical Infrastructure

The representation of an electrical substation, with a production cell, is simple. The input tokens don't change before exiting the infrastructure, as seen in its corresponding HRT below:

Output (kW)	**Input** (kW)
25000	25000
18750	18750
12500	12500
6250	6250
0	0

Table 4.1 Electrical HRT

The previous HRT does a straight mapping between the input and the output. The five levels indicate that the substation has four transformers. More details about this mapping are to be discussed in the next chapter. The availability of transformers provides a naturally discrete behavior for a substation. This is ideal for a reinforcement learning study.

4.1.2 Water Infrastructure

The water infrastructure includes a pumping station. This pumping station takes low-pressure water and electricity as inputs. The low-pressure water comes from a local watershed and is, nearly, always available. The electricity comes from the substation; thus, the functionality of the water station is dependent on the availability of electricity.

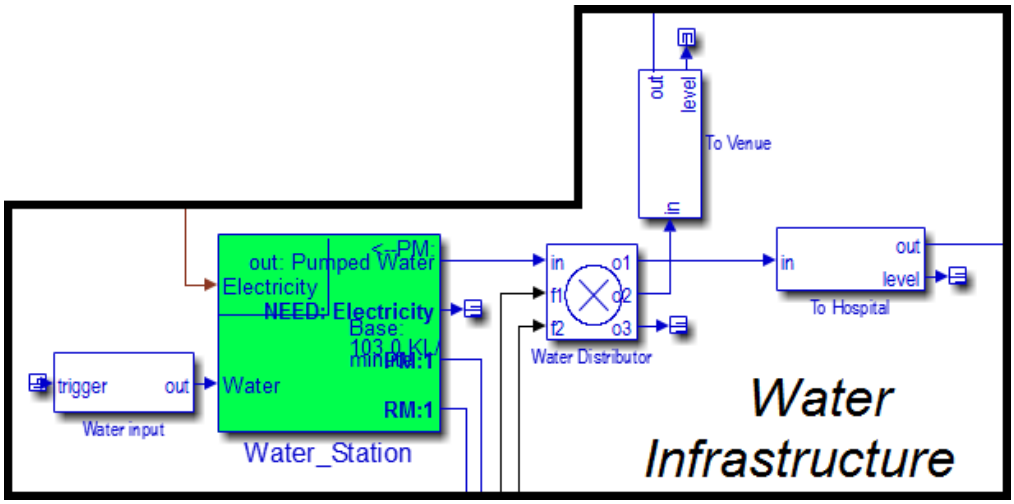


Figure 4.3 Water Infrastructure

Output (kL/h)	**Electricity** (kW)	**Water** (kL/h)
103	10	103
77	8	77
52	5	52
26	3	26
0	0	0

Table 4.2 Water Station's HRT

Availability of information related to local pumping stations is limited, all data used in the HRTs comes from previous projects [5] [68] [6].

4.1.3 Venue

The venue is assumed to be a stadium hosting an event with 10,000 people attending. The production cell used to represent the venue, has an HRT that provides the rate of egress. From all attendees, 30% are injured and in need of medical attention. This process implies an on-site medical team evaluating and classifying people outside the stadium. This classification follows the Canadian triage and acuity scale (CTAS) [69]. Injured people in need of clinical treatment are transported to the emergency facilities. No complex transportation subsystem is in place, so people get in-route to the hospital as soon as they exit the venue. This makes patient arrival to emergency very short. The transportation to the hospital is merely relying on a channel, time to arrival is set to 5 minutes.

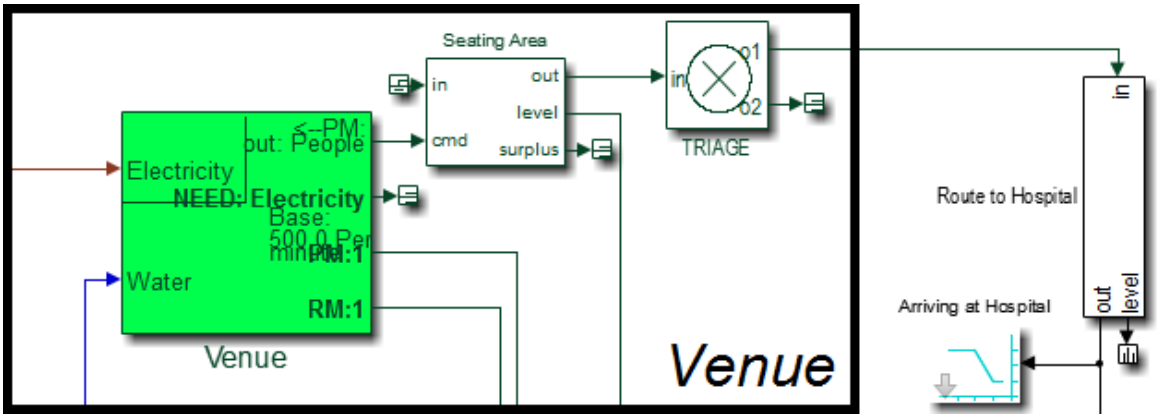


Figure 4.4 Venue Model + Transportation

The inputs to the venue are electricity and water. This assumption is slightly unrealistic, but it was introduced to create multiple interdependencies. The venue's HRT is shown in Table 4.3, below:

Output (People/min)	**Electricity** (kW)	**Water** (kL/h)
500	3000	41
250	2250	31
150	1500	21
50	750	10
0	0	0

Table 4.3 Venue's HRT

4.1.4 Hospital

In the hospital subsystem we have a PC with four inputs: electricity, water, natural gas and medical gases. The first two inputs come from other infrastructures previously described. Natural gas and medical gases are fed with sources. On the right of the PC a storage simulates a waiting area. The waiting area gets its input from the transportation channel, where patients traveled from the venue. A signal from the PC takes patients from the waiting area into treatment. Treatment is simulated with a channel where tokens (people) stay for 30 minutes before being discharged. The rightmost storage is used for the sole purpose of counting the total patients discharged. This accumulated value is used for graphical validation during execution.

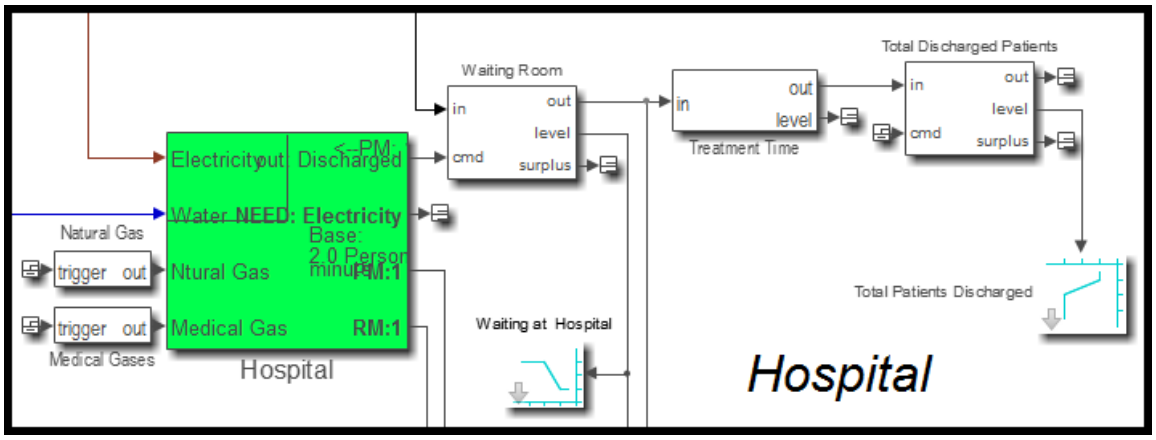


Figure 4.5 Hospital Subsystem

The HRT for the hospital collects data from past projects [5] [68] [6]. This data was collected via interviews with management personnel from local hospitals.

Output (People/h)	**Electricity** (kW)	**Water** (kL/h)	**Natural Gas** (ft^3/h)	**Medical Gas** (%)
10	2000	51	3333	10
7	1500	38	2500	75
5	1000	26	1667	50
2	500	13	833	25
0	0	0	0	0

Table 4.4 HRT Hospital

4.2 Applying Reinforcement Learning

To provide a clear formulation of the reinforcement learning (RL) agent, this chapter will follow the same structure used in the previous chapter for the RL overview. The agent has been implemented as level 2 Simulink block, similar to the ones used for i2Sim's toolbox. Inputs to the agent are variables that can describe the environment and outputs are the selected decisions in the form of distributor ratios.

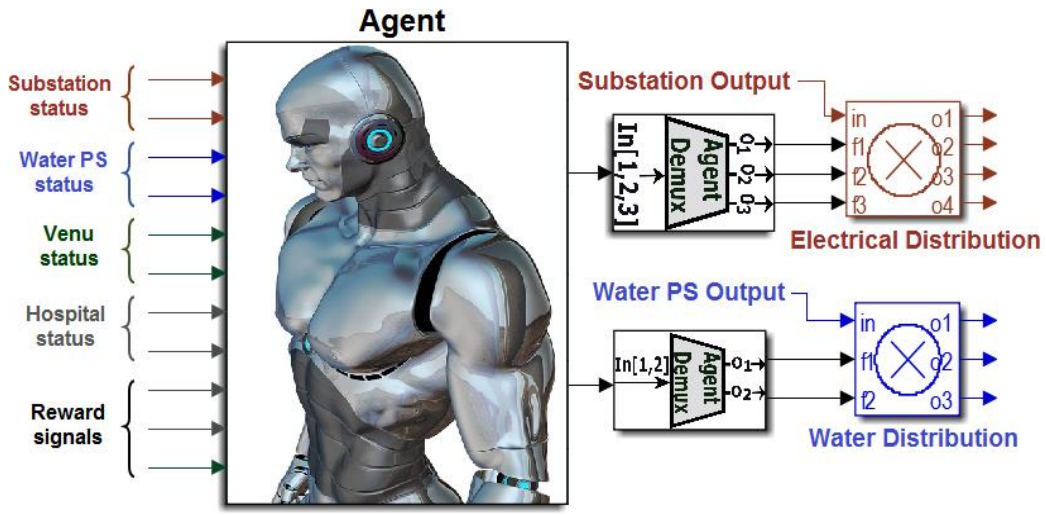


Figure 4.6 Reinforcement Learning Agent

4.2.1 State Space

The states are determined by features that configure an instantaneous snapshot of the environment. The variables used for identifying the states are the status descriptors of the infrastructures. These descriptors are the physical integrity of the facilities and equipment; and the amount of available

resources for those facilities to operate. An HRT is a table, as described in the previous section. Every row of the HRT represents a resource mode (RM). For all CIs, in the aggregate model, the availability of resources is discretized into 5 levels with a separation of 25%.

Following the resource mapping, the physical integrity (PM) of each production cell (infrastructure) goes from 0% to 100% in 25% steps. There are as many physical modes as rows in the HRT. The PM restricts operational capacity even when sufficient resources are in place. Thus, we could say that a physical mode contains resource modes. Proportionally, if the physical damage increases the number of resource modes decreases (See Figure 4.7).

HRT

		Output (kL/h)	Electricity (kW)	Water(kL/h)
PM 1	RM 1	103	10	103
	RM 2	77	8	77
	RM 3	52	5	52
	RM 4	26	3	26
	RM 5	0	0	0

		Output (kL/h)	Electricity (kW)	Water(kL/h)
		~~103~~	~~10~~	~~103~~
PM 2	RM 1	77	8	77
	RM 2	52	5	52
	RM 3	26	3	26
	RM 4	0	0	0

		Output (kL/h)	Electricity (kW)	Water(kL/h)
		~~103~~	~~10~~	~~103~~
		~~77~~	~~8~~	~~77~~
PM 3	RM 1	52	5	52
	RM 2	26	3	26
	RM 3	0	0	0

		Output (kL/h)	Electricity (kW)	Water(kL/h)
		~~103~~	~~10~~	~~103~~
		~~77~~	~~8~~	~~77~~
		~~52~~	~~5~~	~~52~~
PM 4	RM 1	26	3	26
	RM 2	0	0	0

		Output (kL/h)	Electricity (kW)	Water(kL/h)
		~~103~~	~~10~~	~~103~~
		~~77~~	~~8~~	~~77~~
		~~52~~	~~5~~	~~52~~
		~~26~~	~~3~~	~~26~~
PM 5	RM 1	0	0	0

Figure 4.7 PM & RM Relationship

Using two variables from each production cell (PC) for state mapping is not a problem. The complications can rise from the irregular number of resource modes (RM) in each PM. To better address this hierarchical relation, between the physical state and the input resources, we unified the PM and the RM into a single index. We called it the operating index (OI). Each production cell, in this model, has 15 possible PM/RM combinations. Hence, the operational index can take up to 15 values per PC. The operating index is illustrated in Figure 4.8.

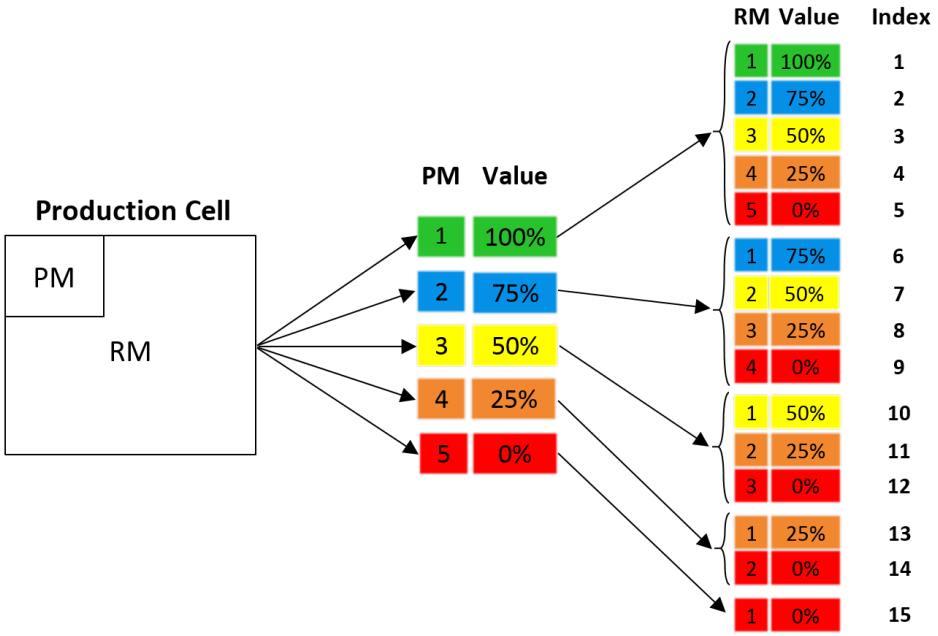


Figure 4.8 Production Cell (PC) Mapping to States Based on the Operating Index (OI)

As the four PCs follow the same setup, a combinatorial of their four operational indexes determines the size of the state space, 15^4. Where 15 is the number of operational indexes and 4 is the number of PCs. The environment can be sensed by the agent through 50,625 states. This number of states represent all the possible combinations of state variables. Up to a certain extent, different combinations could be understood as different scenarios derived from the same model. As expected, a group of the state mappings configures some scenarios with scarce

resource availability; hence, resource allocation is more challenging. These instances constitute the main goal of the agent's training. The *best* states are the ones running without physical damage, and with total resource availability. These best states are used as a baseline, for verifying the convergence of the solution before extending it to a distributed implementation.

4.2.2 Action Space

After sensing the status of the environment, the agent reacts by choosing an action from a list of physically and topologically possible actions. Since this problem focuses on coordinated response, the agent's actions are simultaneous electrical and water distribution settings. A total of 5 actions have been configured at the electrical distributor and 3 at the water distributor. A combinatorial of both sets yields a total of 15 actions. All actions are valid for all the states. The distributor combinations available to configure actions are shown below:

Hospital (%)	Venue (%)	Water PS (%)	Residential (%)
80	12	0.040	7.960
60	10	0.030	29.970
50	8	0.010	41.990
30	2	0.005	67.995
20	1	0.001	78.999

Table 4.5 Set of Actions for Electrical Distributor

Hospital (%)	Venue (%)	Residential (%)
50	50	0
30	20	50
10	5	85

Table 4.6 Set of Actions for Water Distributor

The two decision points (DP) and state variable descriptors have been highlighted in Figure 4.9.

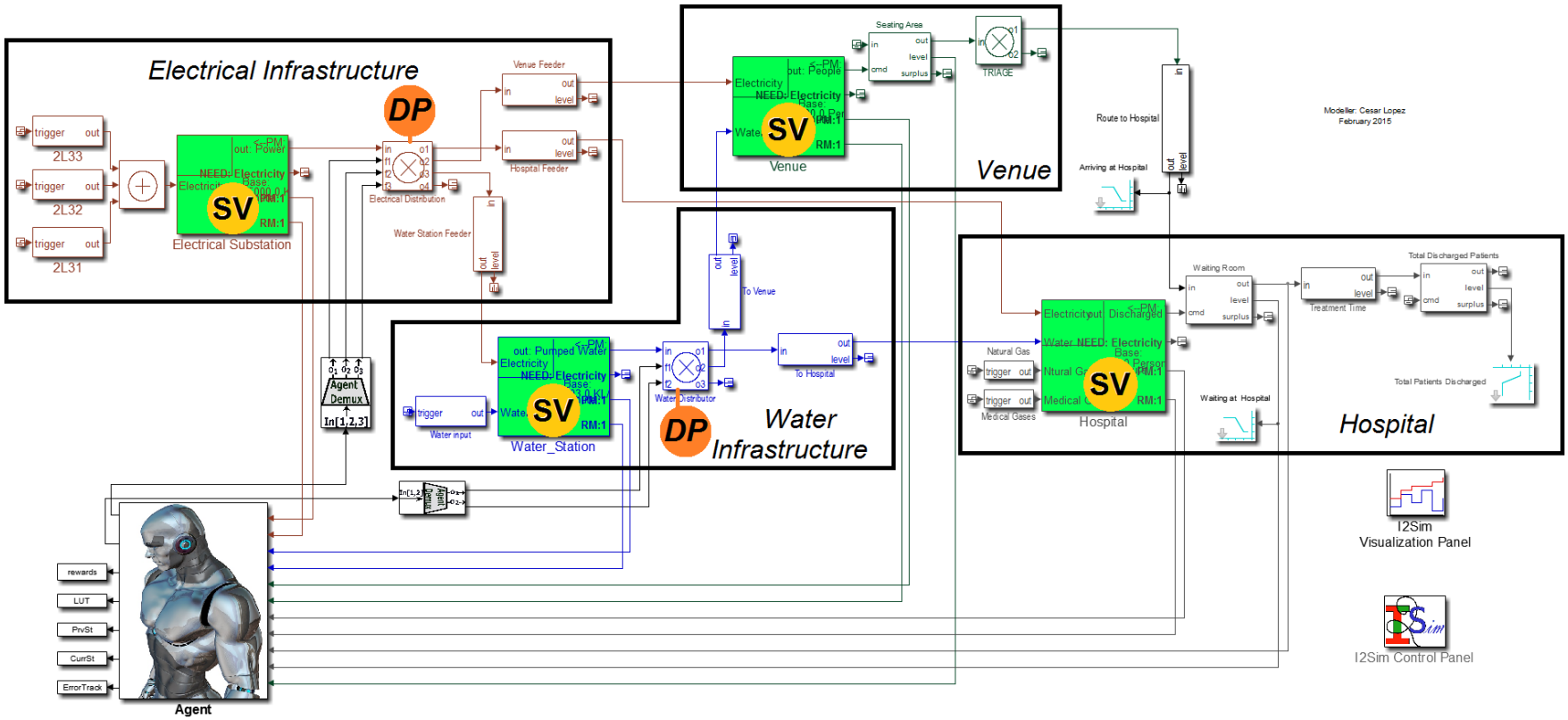


Figure 4.9 Aggregate Model with State Variables and Decision Points

This project uses a tabular state/action mapping, a look up table (LUT). The LUT is sized according to the, previously calculated, number of states (50,625) and actions (15). Hence, the LUT's size is given by $50{,}625 \times 15 = 759{,}375$. Each element in the LUT is indexed as $Q(s, a)$, in agreement with the *Q-Function* calculations covered in Chapter 2.

As mentioned earlier, the agent was implemented as a MATLAB/Simulink block and the default datatype is double (8 bytes). Therefore, the total memory space required for the LUT is $759{,}375 \times 8 = 6{,}075{,}000$B i.e., $\mathbf{5.79MB}$. The small amount of memory space, this LUT uses, does not represent a big requirement. But, growing the number of tracked PCs may push the memory requirements beyond representable bounds, especially in regular computing stations. Hence, a distributed representation is no longer optional.

Since memory is not an issue with this test case, dimensionality improvements won't be visible. However, dimensionality is not the only improvement this thesis focuses on. As a model can be configured to act as a multi-scenario representation, those multiple scenarios need be run one after another. Thus, enabling parallel training of multiple scenario configurations is a big improvement on our previous work.

4.2.3 Rewards

The main goal of the agent is to allocate the resources in a way that maximizes the number of patients treated at the hospital. The instantaneous reward is calculated by comparing the hospital's output at times t and $t - 1$. This rewards scheme was tested in [8] and [52]. Models with a similar configuration as the one described here, showed slow convergence.

A revised reward scheme was needed for speeding up convergence, and it is a goal of this research. We decided to add a shaping reward function. The successful addition of shaping rewards requires a lot of experimentation. We tested multiple options that involved transportation and egress tasks. However, this set of trial and errors did not help the agent through its learning process. Further analysis over the original scheme was needed for finding room for improvements.

As the simple reward function calculated the reward signal, merely, by subtracting the current hospital's output from its previous output, that yielded a small value. The first step towards an improved reward scheme was to amplify that difference. We added a penalty factor **5**, to make the feedback more impactful on the lessons acquired by the agent. This, however, did not help when there was a zero difference. A zero difference means that the latest action produced no change to the hospital's output. We used the current hospital's output as the reward signal for this case. One last possibility was to have a zero difference and a zero output, thus, a zero reward signal. We analyzed this option by asking: *if the hospital's waiting area is not empty, why no patients are treated?*. This last branch, of the possible reward options, sent a reward $-\mathbf{5}$ to the Q-function. Our revised reward scheme is depicted in Figure 4.10.

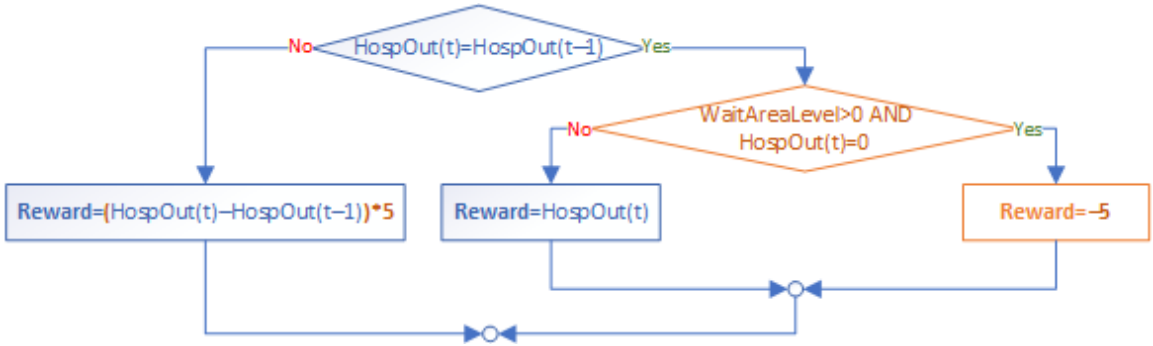


Figure 4.10 Improved Reward Scheme

In penalty function methods the recommended penalty factor ranges $\{1..10\}$. As the average output change of the hospital is 2.5, we decided to use twice that (5) as our penalty factor. Our new reward scheme increases speed of convergence by a great factor, as discussed further in this chapter. The increase is estimated based on the results of our reference past projects.

4.3 Sequential Setup and Results

The model is configured top span over 1560 discrete steps. This corresponds to 1560 minutes of real time (26 hours). This length is what will be referred to as an *episode* or a full run. The aim of this test is to have the best sequential solution. A parallel solution is valid only when compared to its best sequential version. We will discuss four items at this point: the look up table (LUT), model convergence, policy, and test results.

4.3.1 Look Up Table

As mentioned before, this implementation uses a tabular mapping of states and actions, a *Look Up Table* (LUT). The LUT is a matrix with actions as rows and states as columns.

Commonly, the LUT is initialized with zeros for all state/action intersections, i.e., every $Q(s, a)$. In some of our previous tests we discovered a condition we have called *convenient convergence*. What we call convenient/coincidental convergence is to find the right action for a state, due to its *"convenient"* location. As an example, consider a LUT having 0 as the initial Q-value for all positions. The priority for selecting any action, in a particular state, is the same. As the agent looks for the highest Q-value sequentially, action #1 would be selected. If action #1 happens to be the best one, for that state, it would be conveniently located. Thus, the solution would converge by a mere coincidence, leading to questionable results.

For proving strong convergence of our solution, instead of using zeros, we have used random values between *-0.5* and *0.5* as initial Q-values.

4.3.2 Convergence

For testing the convergence of the solution, we configured a model baseline where all the resources were available at a 100%. When all the resources are fully available, the utility providers are in normal operation, hence, the distribution of their supply is known. This model baseline is the reference for evaluating that the agent is learning the right policy. The reference baseline was executed without the agent for generating the target solution the agent is expected to match after being trained. The solution is shown below:

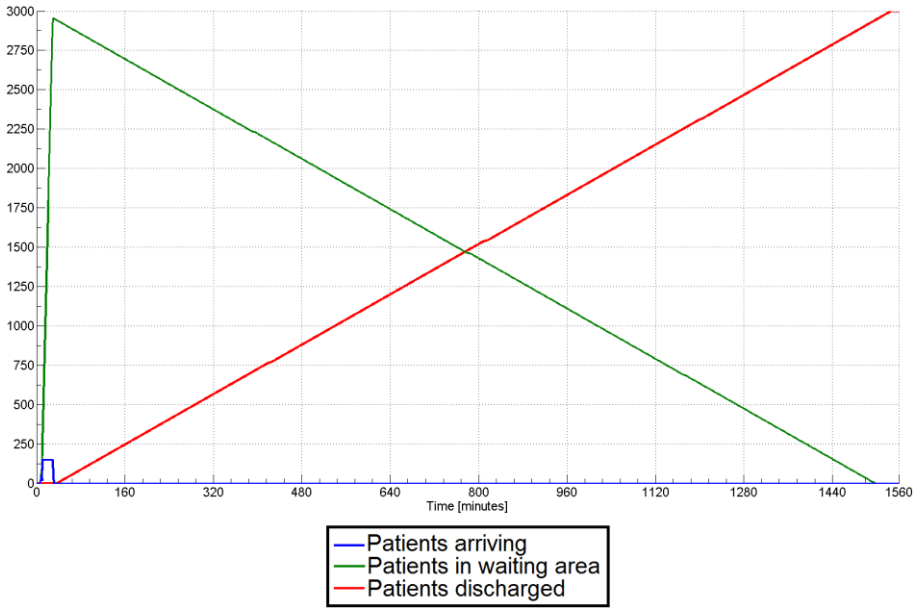


Figure 4.11 Referential Solution for Evaluating Convergence

Once the agent was added to the model, we tested the impact of the newly added shaping reward scheme. This improved feedback, helps the agent move towards the right solution much

faster than using the simple reward scheme. We sampled the first episode of the agent's training with each reward scheme and keeping all the other parameters constant. A look at the initial episode's results from both reward schemes shows the strong incidence of the shaping rewards over the speed of convergence. Being this the first run, a faster overall convergence is undoubtedly expected. A comparison of the two reward schemes is illustrated in Figure 4.12.

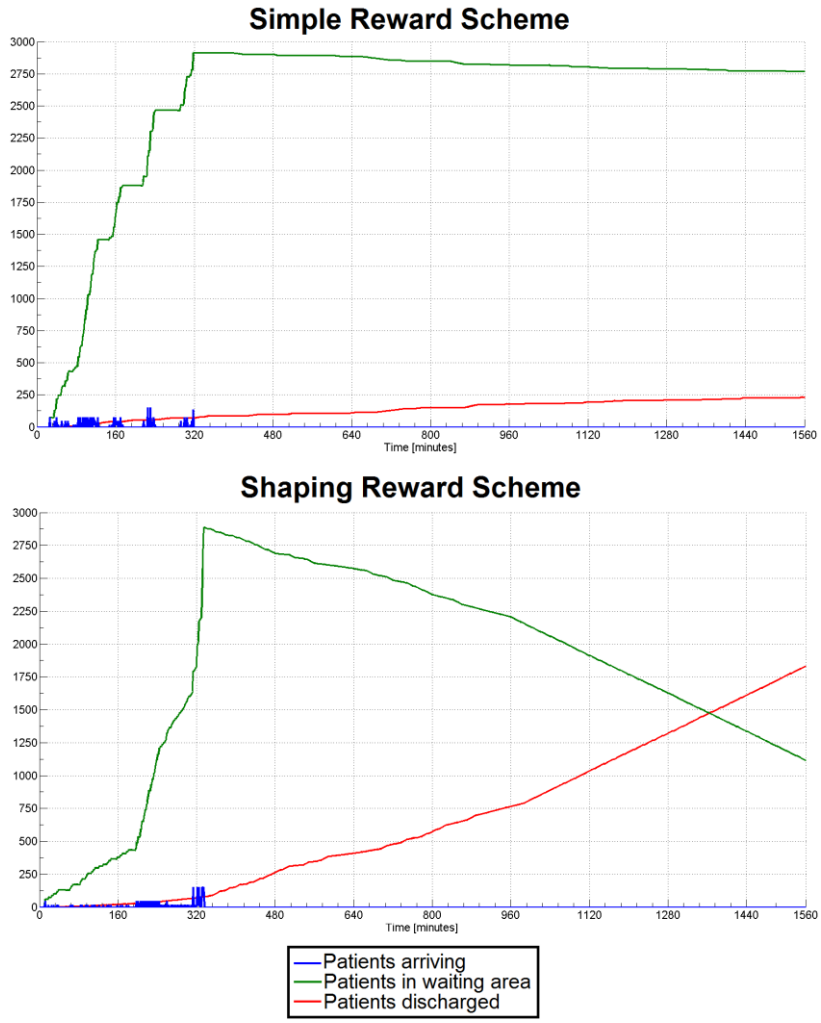


Figure 4.12 Simple vs. Enhanced Reward Schemes

The reference work we are seeking to improve [8] took *50* episodes, on average, to determine a policy over the same model. With the new approach, *5* runs are enough to converge to the desired

policy. However, we decided to allow this new version to have *6* runs. We believe this extra training episode helps increasing the robustness of the solution as it allows the agent to experience more. Additionally, we have used a baseline with all the resources available; scenarios with scarce resources could benefit from an extra training episode.

4.3.3 Policy

We let the agent explore new actions 5% of the times, at most, as a high level of exploitation is needed for SARSA to converge [62]. We decided to decrease exploration episodically, considering that the more the agent learns the less it needs to improvise (try random actions). We decide to use $\in = \{5, 3.7, 2.5, 1, 0.5, 0.2\}$ as our exploratory scheme. This scheme reduces exploration from 5% to almost 0% along the 6 episodes.

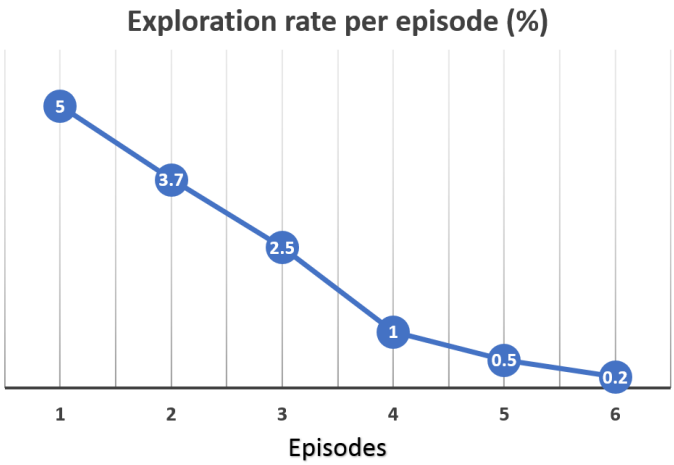


Figure 4.13 Exploration Rate Episodic Decrease

The decrease in exploration is more drastic during the first half of list. Since the agent learns fast, due to the improved reward scheme, exploration can also drop fast. Table 4.7 summarizes the details of the scheme. The values listed in Table 4.7 are calculated based on the total number of timesteps in an episode, 1560.

Run#	ϵ	Greedy	#Exploratory actions	Explores every ( ) steps
1	5%	95%	78	20
2	3.7%	96.3%	58	27
3	2.5%	97.5%	39	40
4	1%	99%	16	98
5	0.5%	99.5%	8	195
6	0.2%	99.8%	4	390

Table 4.7 ∈-Greedy Scheme

To summarize the reinforcement learning setup and training, we provide the algorithm running inside the agent:

Initialize LUT ($Q(s,a)$) with values between -0.5 and 0.5
REPEAT (for each step of the simulation)
IF timestep > 1 AND waiting area NOT empty THEN
Calculate s_t
Compute r_t *with reward function*
Choose a_t *policy* $\in -greedy$
$Q(s_{t-1}, a_{t-1}) \leftarrow Q(s_{t-1}, a_{t-1}) + \alpha[r_t + \gamma Q(s_t, a_t) - Q(s_{t-1}, a_{t-1})]$
$s_{t-1} = s_t$
$a_{t-1} = a_t$
ENDIF
UNTIL simulation is finished

As observed in the algorithm, the learning process starts from the second timestep as variables from $t - 1$ are used to generate feedback signals. Likewise, the learning is stopped when there are no more patients to treat (empty waiting area).

4.3.4 Results

On average, an episode took 50 seconds to complete. The training set was repeated over the baseline case 50 times, just to make sure it converged consistently. The agent learned the same policy across all tests. As specified before, the training set comprised 6 episodes. This extensive testing was validated visually through the graphical user interface of the simulator.

The baseline model was configured to complete treatment of all injured people, a few timesteps before completion. This was intentionally done for giving the agent just enough time to achieve the goal, while a graphical validation was in place. For the distributed implementation all graphical interfaces will be disabled. A new scheduler will be written to handle the parallel training. Figure 4.14 shows the first and last episode plots of a training sample.

Figure 4.14 First vs. Last Training Episodes

Below we provide the results of two full training samples (6 episodes). In some cases, the improvement is subtle but visible; in others the learning progression is slower but completed.

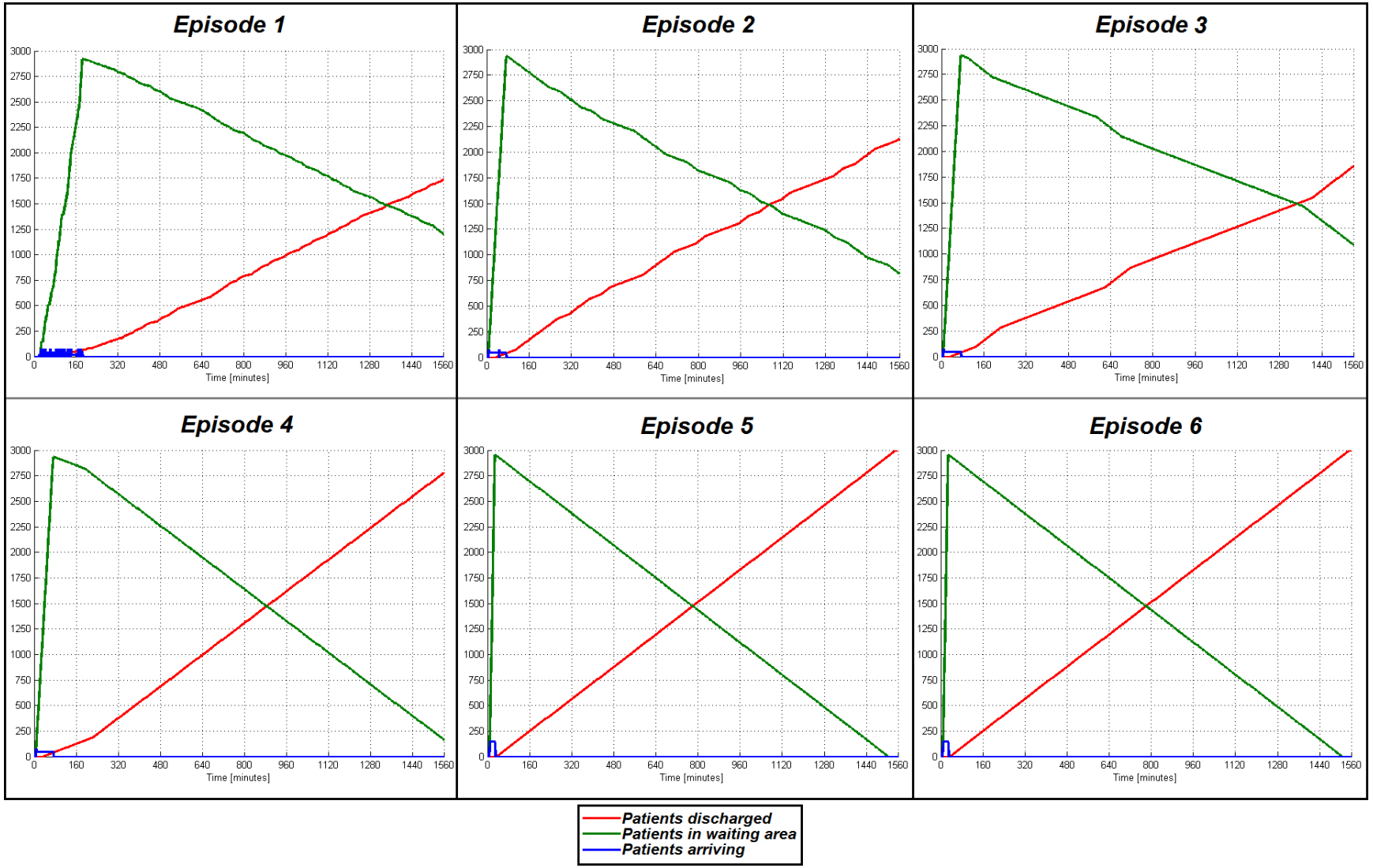


Figure 4.15 Sample 1 Full Sequential Training

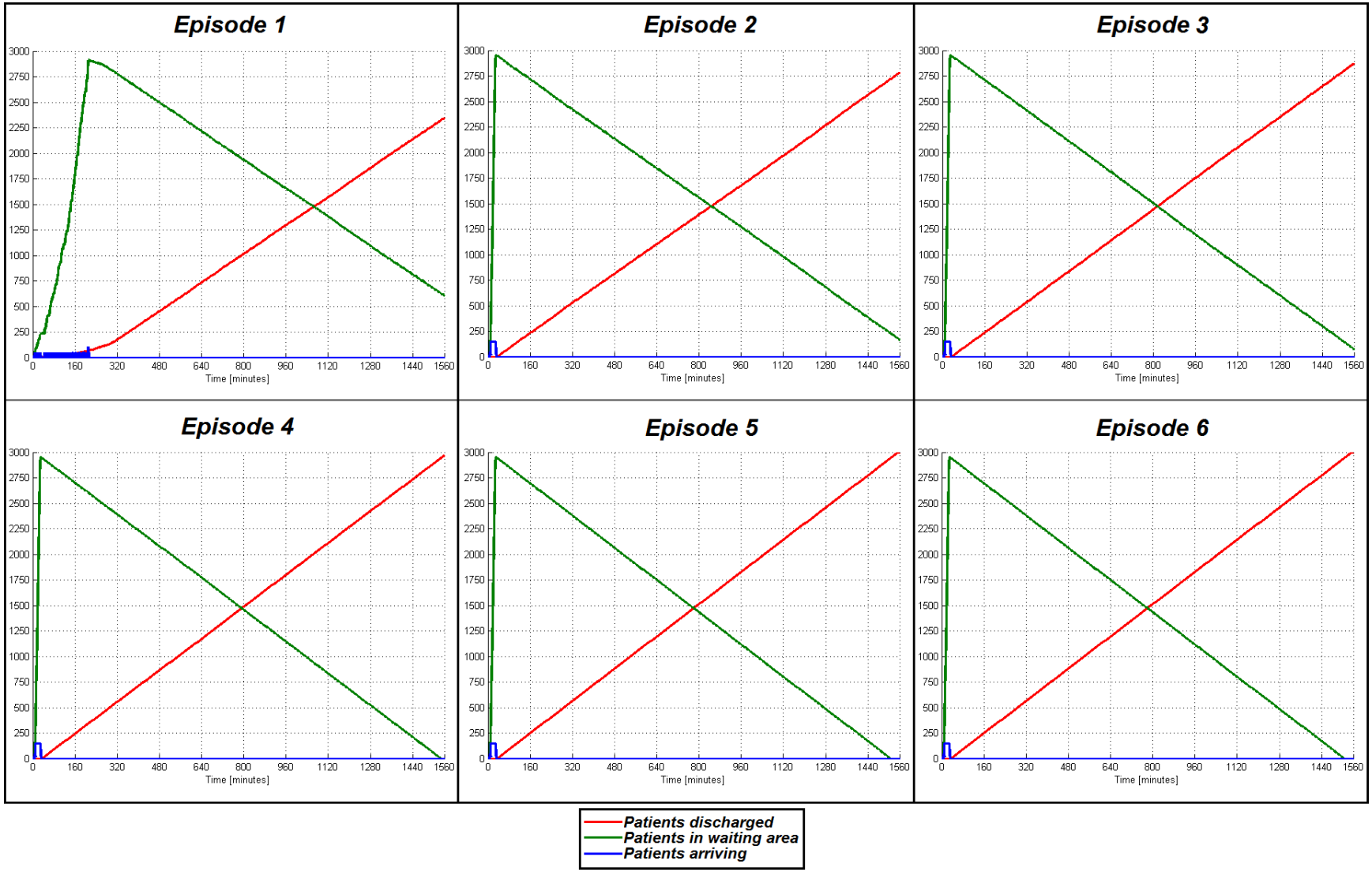


Figure 4.16 Sample 2 Full Sequential Training

As observed in the two previous figures, the agent converged to the desired policy after the 5th episode; the sixth episode, in both sets, reports identical results. We see both sample trainings converging to the same policy. With this sequential version finished and tested, we are ready to configure the distributed implementation.

4.4 Distributed Implementation

Finding decoupling points for the test case model was the first step in generating a distributed solution. An understanding of the agent/environment interaction shows that the actions taken by the agent change the status of the simulation. But, not all state generating variables are linked to a decision point in the model, hence, those are suitable partitioning points. Those variables are infrastructure variables that remain unaffected when agent allocates resources.

The agent manages the electrical and water supply distributions, by using a set of predefined actions, topologically correct. The states are determined by a combination of physical and resource modes from all the CIs. Since no action taken by the agent changes the physical integrity of any CI, the physical modes are state descriptors immune to the agent's acting. Likewise, the agent manages the split of electricity and water at the output of each corresponding facilities but does not control their inputs.

Based upon these conclusions, we chose the decoupling variables to be the physical modes of all infrastructures and the resource modes of electrical and water infrastructures. We generated a partitioning table to be passed to the running instances, as a parameter. This table was created by combining the decoupling state variables. We conducted a preliminary test, over different configurations of the model, and discovered that the agent cannot find a policy when the electrical availability is below 50%. The corresponding partitions were removed. It is important

to highlight that a small test case, like this, allows that simplification. Larger cases with hundreds to thousands of combinations would be impossible to filter. The final table includes 24 partitions and enables the same number of model instances to run independently. See Table 4.8, below:

Partition	PM1	RM1	**OI1**	PM2	RM2	**OI2**	PM3	RM3	**OI3**	PM4	RM4	**OI4**	**State**	**# States**
1	1	1	**1**	1	1	**1**	1	1	**1**	1	1	**1**	**1**	1125
	1	1	**1**	1	5	**5**	5	1	**15**	5	1	**15**	**1125**	
2	1	1	**1**	2	1	**6**	1	1	**1**	1	1	**1**	**1126**	900
	1	1	**1**	2	4	**9**	5	1	**15**	5	1	**15**	**2025**	
3	1	1	**1**	3	1	**10**	1	1	**1**	1	1	**1**	**2026**	675
	1	1	**1**	3	3	**12**	5	1	**15**	5	1	**15**	**2700**	
4	1	1	**1**	4	1	**13**	1	1	**1**	1	1	**1**	**2701**	450
	1	1	**1**	4	2	**14**	5	1	**15**	5	1	**15**	**3150**	
5	1	2	**2**	1	1	**1**	1	1	**1**	1	1	**1**	**3376**	1125
	1	2	**2**	1	5	**5**	5	1	**15**	5	1	**15**	**4500**	
6	1	2	**2**	2	1	**6**	1	1	**1**	1	1	**1**	**4501**	900
	1	2	**2**	2	4	**9**	5	1	**15**	5	1	**15**	**5400**	
7	1	2	**2**	3	1	**10**	1	1	**1**	1	1	**1**	**5401**	675
	1	2	**2**	3	3	**12**	5	1	**15**	5	1	**15**	**6075**	
8	1	2	**2**	4	1	**13**	1	1	**1**	1	1	**1**	**6076**	450
	1	2	**2**	4	2	**14**	5	1	**15**	5	1	**15**	**6525**	
9	1	3	**3**	1	1	**1**	1	1	**1**	1	1	**1**	**6751**	1125
	1	3	**3**	1	5	**5**	5	1	**15**	5	1	**15**	**7875**	
10	1	3	**3**	2	1	**6**	1	1	**1**	1	1	**1**	**7876**	900
	1	3	**3**	2	4	**9**	5	1	**15**	5	1	**15**	**8775**	
11	1	3	**3**	3	1	**10**	1	1	**1**	1	1	**1**	**8776**	675
	1	3	**3**	3	3	**12**	5	1	**15**	5	1	**15**	**9450**	
12	1	3	**3**	4	1	**13**	1	1	**1**	1	1	**1**	**9451**	450
	1	3	**3**	4	2	**14**	5	1	**15**	5	1	**15**	**9900**	
13	2	1	**6**	1	1	**1**	1	1	**1**	1	1	**1**	**16876**	1125
	2	1	**6**	1	5	**5**	5	1	**15**	5	1	**15**	**18000**	
14	2	1	**6**	2	1	**6**	1	1	**1**	1	1	**1**	**18001**	900
	2	1	**6**	2	4	**9**	5	1	**15**	5	1	**15**	**18900**	
15	2	1	**6**	3	1	**10**	1	1	**1**	1	1	**1**	**18901**	675
	2	1	**6**	3	3	**12**	5	1	**15**	5	1	**15**	**19575**	
16	2	1	**6**	4	1	**13**	1	1	**1**	1	1	**1**	**19576**	450
	2	1	**6**	4	2	**14**	5	1	**15**	5	1	**15**	**20025**	
17	2	2	**7**	1	1	**1**	1	1	**1**	1	1	**1**	**20251**	1125
	2	2	**7**	1	5	**5**	5	1	**15**	5	1	**15**	**21375**	
18	2	2	**7**	2	1	**6**	1	1	**1**	1	1	**1**	**21376**	900
	2	2	**7**	2	4	**9**	5	1	**15**	5	1	**15**	**22275**	
19	2	2	**7**	3	1	**10**	1	1	**1**	1	1	**1**	**22276**	675
	2	2	**7**	3	3	**12**	5	1	**15**	5	1	**15**	**22950**	
20	2	2	**7**	4	1	**13**	1	1	**1**	1	1	**1**	**22951**	450
	2	2	**7**	4	2	**14**	5	1	**15**	5	1	**15**	**23400**	
21	3	1	**10**	1	1	**1**	1	1	**1**	1	1	**1**	**30376**	1125
	3	1	**10**	1	5	**5**	5	1	**15**	5	1	**15**	**31500**	
22	3	1	**10**	2	1	**6**	1	1	**1**	1	1	**1**	**31501**	900
	3	1	**10**	2	4	**9**	5	1	**15**	5	1	**15**	**32400**	
23	3	1	**10**	3	1	**10**	1	1	**1**	1	1	**1**	**32401**	675
	3	1	**10**	3	3	**12**	5	1	**15**	5	1	**15**	**33075**	
24	3	1	**10**	4	1	**13**	1	1	**1**	1	1	**1**	**33076**	450
	3	1	**10**	4	2	**14**	5	1	**15**	5	1	**15**	**33525**	

Table 4.8 Look up table partitions

In Table 4.8, the columns (from left to right) correspond to: partition number (1-24), state space variables for electrical substation (1), water station (2), venue (3), and hospital (4). Next, we find the state number use to identify the upper and lower limits of the partition. Last, we have the number of states in each partition. The selected decoupling variables, for the distributed implementation of this model, were: PM1 (substation), RM1 (substation) and PM2 (waterPS). We decided to leave the resource mode of the water station (RM2) out of the partitioning variables. The rationale for this decision is: the agent can change RM2 when allocating electricity, hence, only water from the source is independent to the agent. But, lack of water coming from the watershed is improbable, thus, physical problems (PM2 related) are more realistic.

A closer look at Table 4.8 shows that different partitions have different sizes. Those different sizes are sequentially repeated every 4 partitions. We called this repeating pattern a *band*. An illustration of then bands over the partitioning table is shown in Figure 4.17.

Partition	PM1	RM1	**OI1**	PM2	RM2	**OI2**	PM3	RM3	**OI3**	PM4	RM4	**OI4**	**State**	**# States**	
1	1	1	**1**	1	1	**1**	1	1	**1**	1	1	**1**	**1**	1125	**Band 1**
	1	1	**1**	1	5	**5**	5	1	**15**	5	1	**15**	**1125**		
2	1	1	**1**	2	1	**6**	1	1	**1**	1	1	**1**	**1126**	900	
	1	1	**1**	2	4	**9**	5	1	**15**	5	1	**15**	**2025**		
3	1	1	**1**	3	1	**10**	1	1	**1**	1	1	**1**	**2026**	675	
	1	1	**1**	3	3	**12**	5	1	**15**	5	1	**15**	**2700**		
4	1	1	**1**	4	1	**13**	1	1	**1**	1	1	**1**	**2701**	450	
	1	1	**1**	4	2	**14**	5	1	**15**	5	1	**15**	**3150**		
			. . .								. . .				
21	3	1	**10**	1	1	**1**	1	1	**1**	1	1	**1**	**30376**	1125	**Band 6**
	3	1	**10**	1	5	**5**	5	1	**15**	5	1	**15**	**31500**		
22	3	1	**10**	2	1	**6**	1	1	**1**	1	1	**1**	**31501**	900	
	3	1	**10**	2	4	**9**	5	1	**15**	5	1	**15**	**32400**		
23	3	1	**10**	3	1	**10**	1	1	**1**	1	1	**1**	**32401**	675	
	3	1	**10**	3	3	**12**	5	1	**15**	5	1	**15**	**33075**		
24	3	1	**10**	4	1	**13**	1	1	**1**	1	1	**1**	**33076**	450	
	3	1	**10**	4	2	**14**	5	1	**15**	5	1	**15**	**33525**		

Figure 4.17 Partition Bands

For this model the number of partitions matched the number of available computing nodes, thus, a one to one mapping was possible. In larger cases, the association between instances and nodes could require a matching by bands instead of by partitions.

Once the problem was decoupled in suitable partitions, we needed to find a distributed execution environment that prevented each local process from running out of bounds. We considered different options. Amongst these options, we evaluated Apache Spark. A problem we found was Spark's automatic partitioning that would not follow our decoupling points. While the implementation still works, shuffling data among nodes highly affects the performance. An alternative to this problem is the use of a customized mapping, but for our problem it could become overcomplex. Moreover, to fit our agent in Apache Spark we would need to reprogram the solution, this would lead to a problematic we are trying to solve: fit the optimization in the platform and not the platform to the optimization. In Chapter 2 we provided a literature review in support of i2Sim being an adequate modeling platform for emergency and disaster scenarios, as such we want to keep all functionality available in i2Sim while adding the optimization.

A second option, possibly better as our simulator an agent's implementations are in MATLAB, was a license of MATLAB's parallel processing library. The cluster capabilities are included in most MATLAB's distributions, but they are locked. Unlocking the full potential of this library, is possible through a license in the range of thousands of dollars. This license is not a onetime payment, instead it comes as an annual subscription.

As all the previous options were not feasible, we decide to write our own scheduler for handling the distributed agent's training. A first step towards writing this managing routine was to gain an

understanding of the cluster's architecture. A general cluster architecture taken from literature is shown in Figure 4.18:

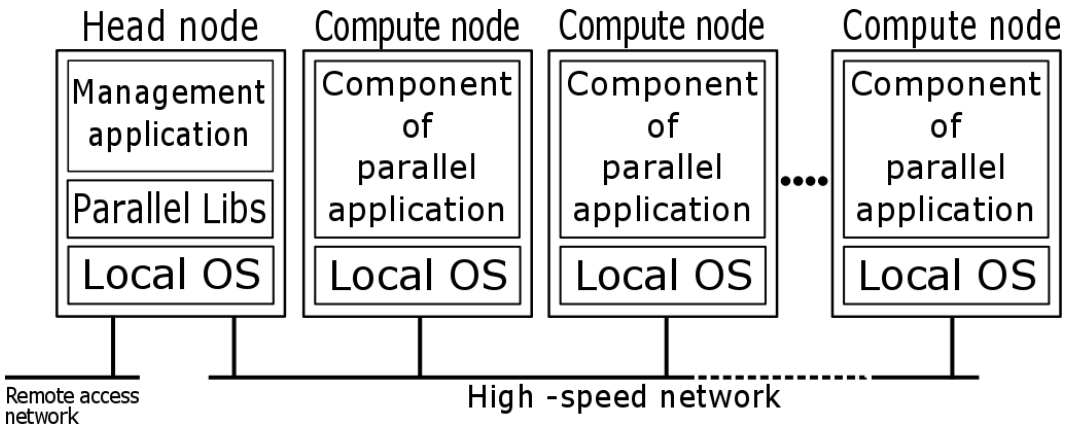


Figure 4.18 Cluster Architecture [70]

We took that basic cluster architecture (Figure 4.18) and adapted it to our IBM cluster's organization, including the software components of our distributed model's implementation:

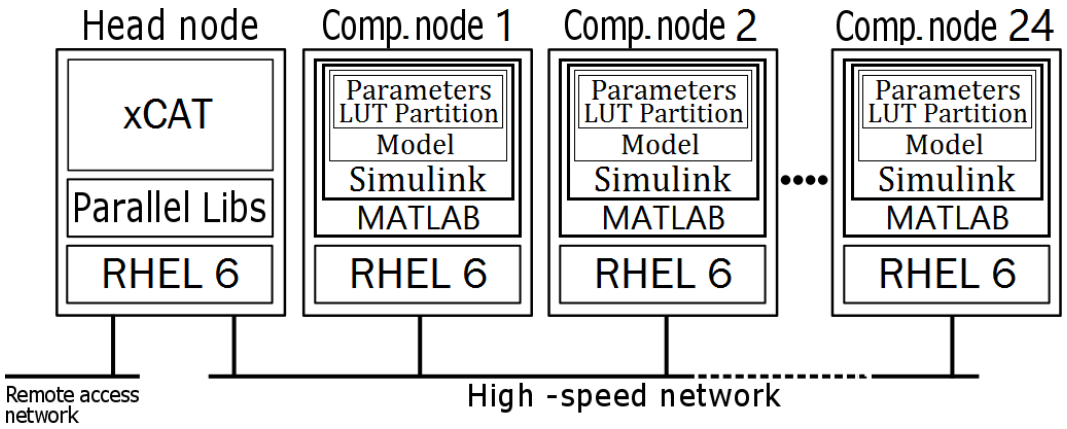


Figure 4.19 Adapted Cluster Architecture

The adapted cluster's organization provided enough insight for formulating our scheduling routine(s). The description of our scheduler, tailored to our distributed reinforcement learning agent, is described next.

4.4.1 The Scheduler

The scheduler is a set of routines written to automate the training of the RL agent across multiple partitions of the state space, simultaneously. Our HPC was the designated running platform. The idea was to use all 24 computing nodes, each one associated with a problem partition. In order to minimize communication overhead, we separated the scheduler implementation between the clustering software and the simulation platform. The clustering software, as mentioned before, is xCAT, Extreme Cluster/Cloud Administration Toolkit. i2Sim, at the software level, is a MATLAB/Simulink toolbox using level 2 blocks. The scheduler performs the following tasks: setup model and learning parameters, looping execution of instances and gathering results from all nodes. The scheduler architecture is depicted in Figure 4.20.

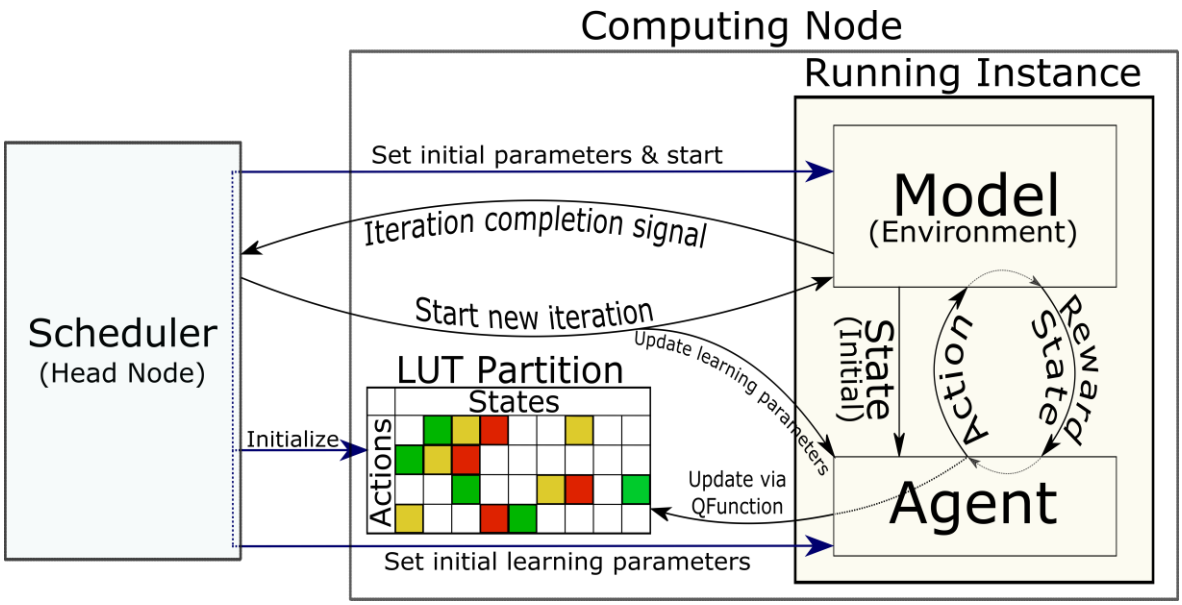


Figure 4.20 Scheduler Architecture

4.4.1.1 Setup of Model and Learning Parameters

A copy of the model is preloaded in each node along with the $\in -Greedy$ scheme, partition matrix, and initial look up table. As we had a one to one mapping between partitions and

computing nodes, the scheduler assigns each partition to the matching node, e.g., Node01 works the first partition. At that point xCAT passes the control to MATLAB by spawning all the threads via a parallel shell.

4.4.1.2 Looping Execution of Instances

The scheduler operates in MATLAB's scope, now, while xCAT awaits completion of the running threads. Each running instance is an independent sequential training. The scheduler iterates, each instance, 6 times. Model parameters do not change as partitions remain unaltered. At the start of each iteration the scheduler modifies the learning parameters following the chosen $\in -Greedy$ scheme, and each agent populates its corresponding LUT (partition).

4.4.1.3 Gathering of Results

Upon completion of the episodic iterative process, the control is passed back to xCAT. The scheduler then collects all partitions (local look up tables).

4.4.2 Results

The scheduler was initially tested on a regular PC, Core i7 with 16GB of RAM, running a single instance with a full training. This test enabled testing of the local scheduler tasks, where a single instance is iterated 6 times and the varying exploratory schema applied in every iteration. Based on the agent's performance during these tests, we could see the possibility of launching multiple simultaneous instances on a PC. However, as we intended to run 24 separates threads, the PC could not handle that load. An individual sample run is displayed in Figure 4.21, below:

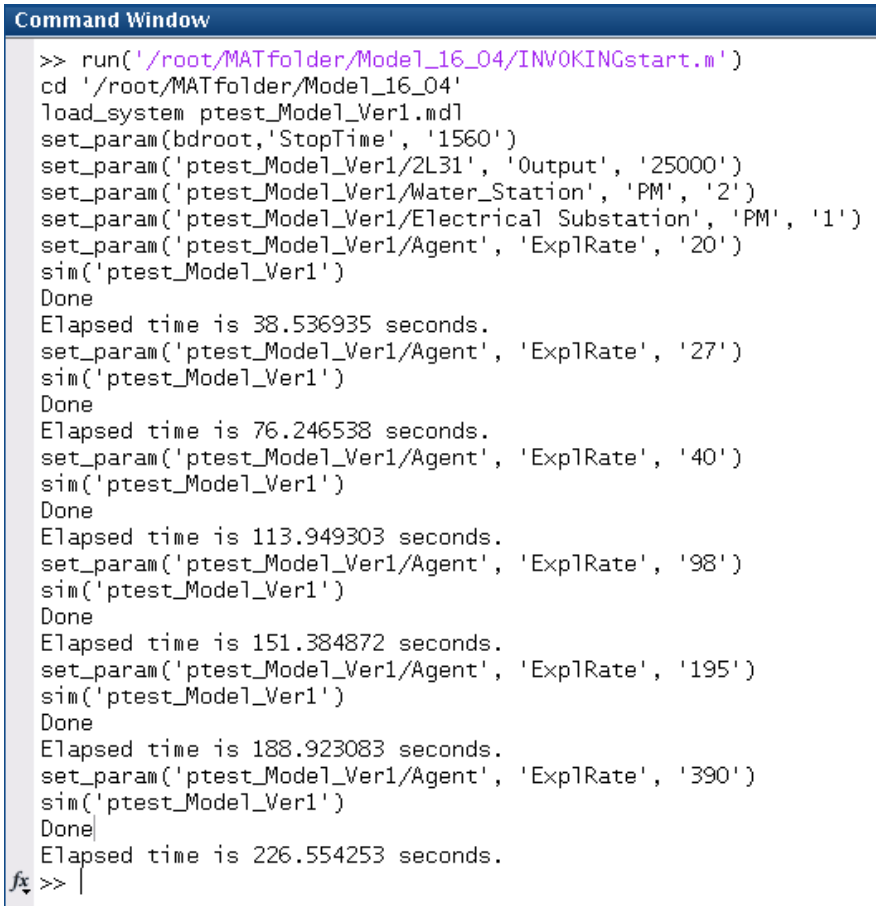


Figure 4.21 Individual Training Run Sample

We ran the training in the cluster multiple times for verifying consistency of the results. The execution was stable, and the results were retrieved correctly. A sample run of the complete distributed training is depicted in Figure 4.22. In this image we observe different workers being launched and their interaction happening in arbitrary order. As the graphical user interface is disabled during parallel execution we get only console output. As mentioned before, the order of messaging between the head node and the processing nodes is entirely random as partitions are not the same size and other factors, such as transport in the network, could affect the response of the nodes. Parameter management and episodic elapsed times are also noticeable. Completion times are very homogeneous and remain around *252 seconds* (*4.2 minutes*).

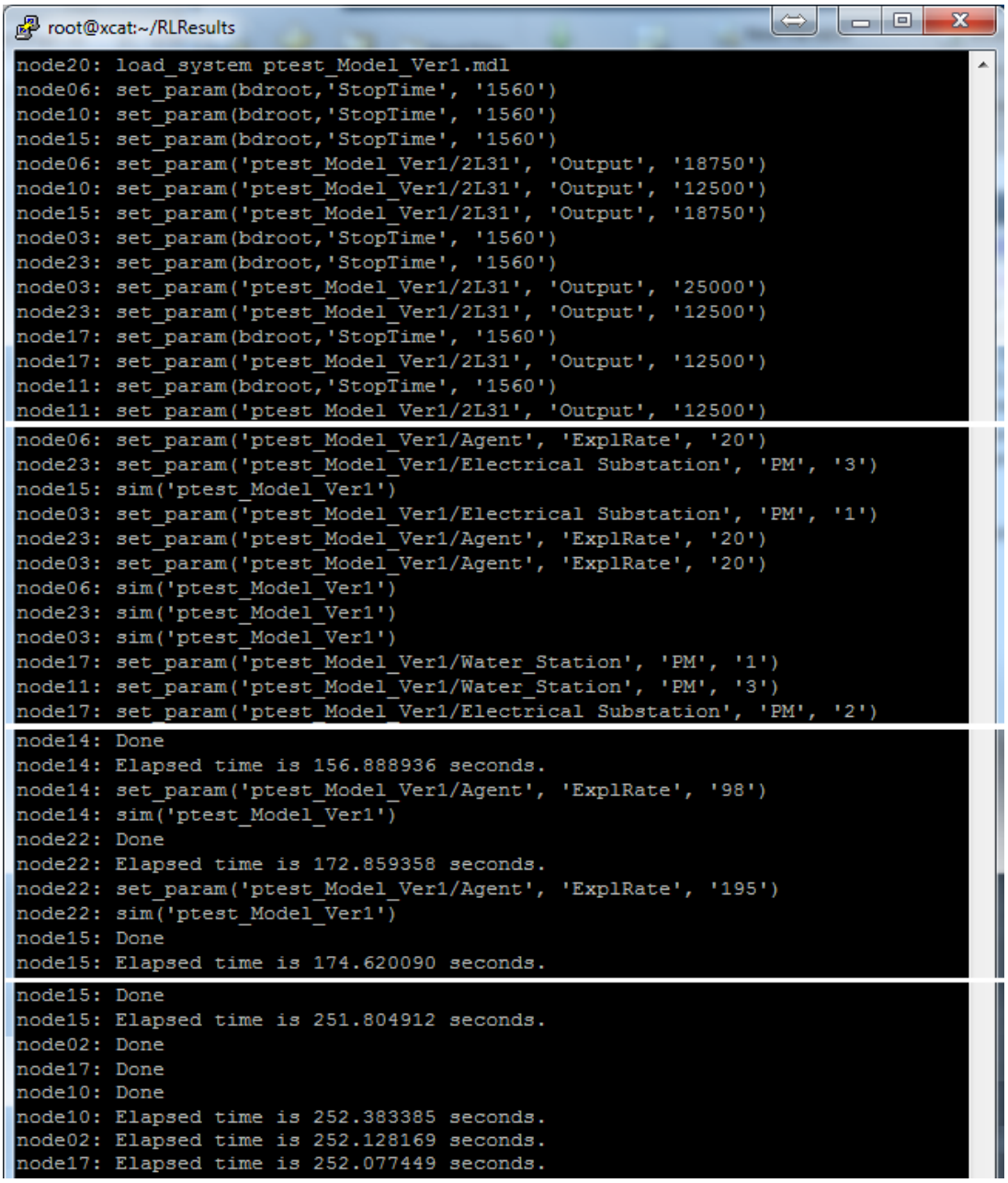


Figure 4.22 Sample Distributed Training Run

Individual node execution times are collected in Figure 4.23. The complete agent's training (all instances) is determined by the longest node's time. As highlighted, in Figure 4.23, this time is *252.757741 seconds* or *4.21 minutes.* Table 4.9 summarizes the time measurements for the distributed sample run discussed.

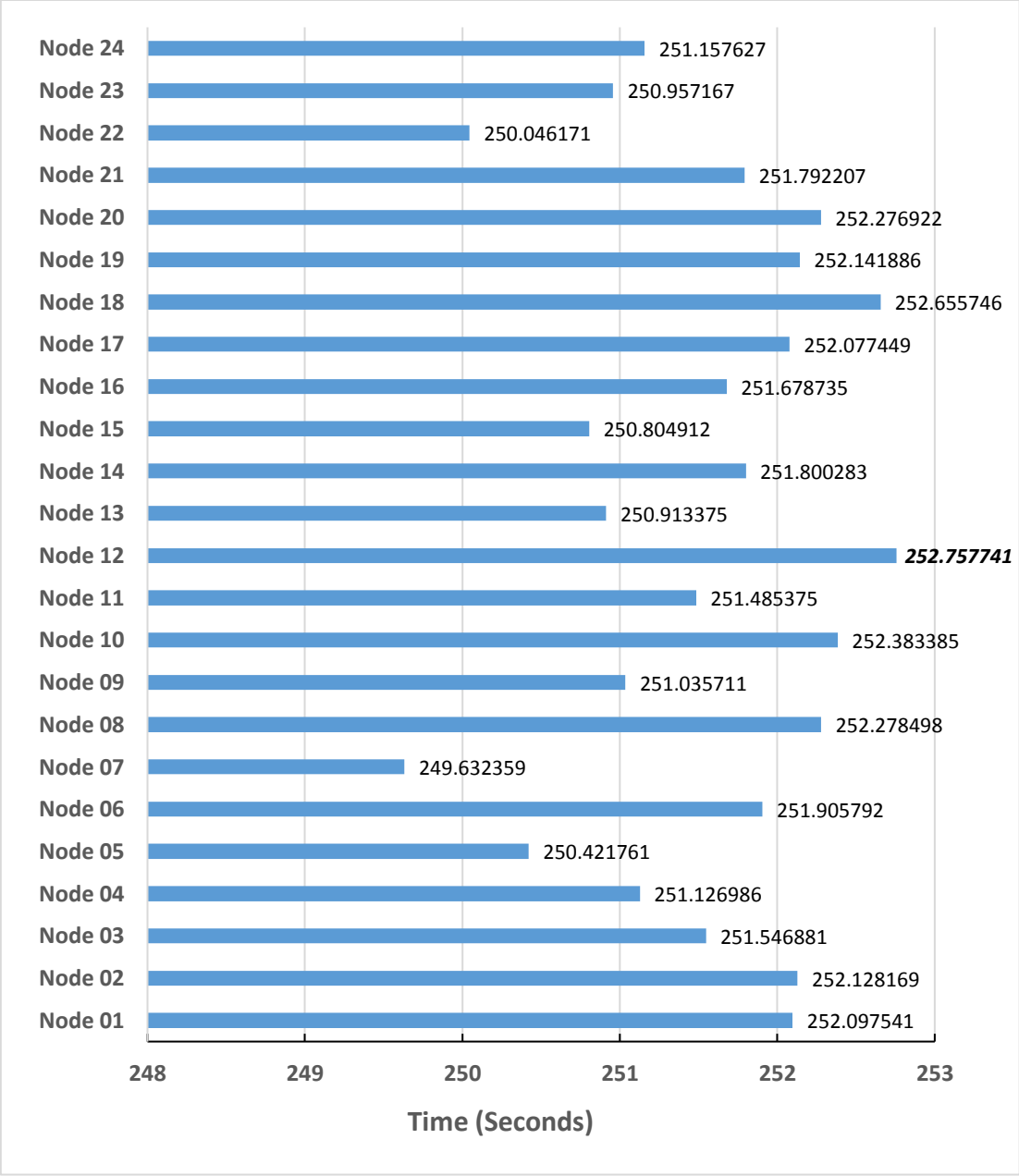


Figure 4.23 Distributed Training Execution Time (in Seconds)

	Elapsed Time	
	Seconds	**Minutes**
Maximum	252.757741	4.21
Average	251.545945	4.19
Minimum	249.632359	4.16

Table 4.9 Distributed Training Time Summary

In Chapter 2: we listed an implementation done for marketing of Adobe solutions, aiming at turning potential customers into active ones [50]. As the data included in the documentation of this solution has execution times we found a suitable comparison, as shown in Table 4.11 and Table 4.11 below:

	Adobe Digital Marketing [50]	**Distributed RL with i2Sim**
# States	9,496	50,625
Scheduling	Apache Spark	Customized
Software running	Custom solution	Simulink (i2Sim + RL Agent)

Table 4.10. Distributed RL vs Adobe Marketing Spark RL - Setup

	Execution Time (minutes)	
	Adobe Digital Marketing [50]	**Distributed RL with i2Sim**
1 instance	40.00	3.76
10 instances	6.00	4.21
20 instances	5.00	4.21
24 instances	N/A	4.21

Table 4.11. Distributed RL vs Adobe Marketing Spark RL – Execution

As seen in the comparison, our solution deals with a higher dimensionality (5.3 times) and runs MATLAB+Simulink in each instance with a full simulation looped 5 times. Our times remain unaffected as the solution scales because all the instances have similar running times and the total time is determined by the slowest thread.

Chapter 5: Expanded Test Case

This extended test case is a more detailed representation of the small test case use in the previous chapter. The extra level of detail enables a more accurate mapping of real infrastructures, without exceeding limits. The amount of detail in a scenario is not always a metric to evaluate the adequacy of the model. Having a very high resolution can have repercussions on the simulation's performance, and it is not a guarantee of obtaining better results. Another factor that can constrain the depth of the model's representation is the availability of data. Critical infrastructures are vital to a country; hence, some related data is not open for disclosure.

The original version of this model was designed during a project sponsored by Defence Research and Development Canada (DRDC) in 2009 [5]. This project's implementation was used as a training environment in preparation for the 2010 Winter Olympics. From that point, this model has evolved and its numerous versions have been a baseline in several projects, including [8] [9] [6] [68]. The version of the model described in this thesis, is the newest and it includes a variety of revisions, where it was possible. Some portions and subsystems of the model remain as in the original version.

In this extended version we simulated an incident affecting two venues hosting simultaneous events. Both stadia located in the same geographical area, in Downtown Vancouver, and at full capacity. The remaining of this chapter will go through the modeling details and it will address the reinforcement learning implementation in a similar way as it was discussed in the previous chapter. Figure 5.1 provides a snapshot of the model.

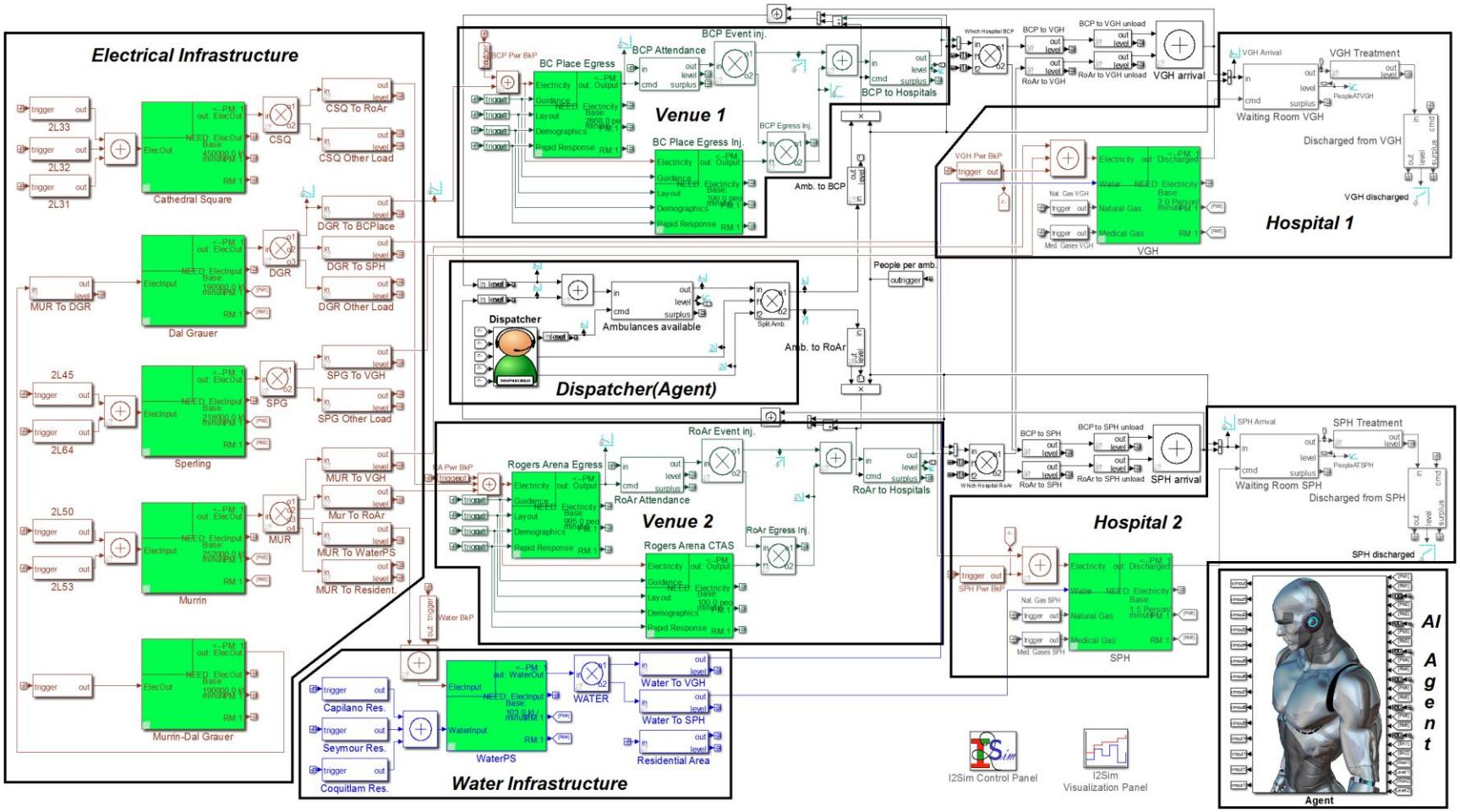


Figure 5.1 Extended Test Case

5.1 Components of the model

The model includes an electrical infrastructure, water infrastructure, two stadia and two hospitals. These infrastructures have been delimited in Figure 5.1.

5.1.1 Electrical system

The electrical supply is modeled with 4 substations and a bypass line. The substations part of the model are: Dal Grauer (DGR), Cathedral Square (CSQ), Murrin (MUR) and Sperling (SPG). The geographical location of the substations is shown below:

Figure 5.2 Geographical Location of Substations

In order to generate the HRTs we collected data about the substations. This data might not reflect the latest updates as access to these sensitive specifications is restricted. The HRTs were based on the number of transformers available at each location. This produces a natural discretization. As an example, if a substation has three transformers, the possible levels of operation are 4: 100%, 67%, 33%, or 0%. Figure 5.3 depicts the substation-HRT mapping used.

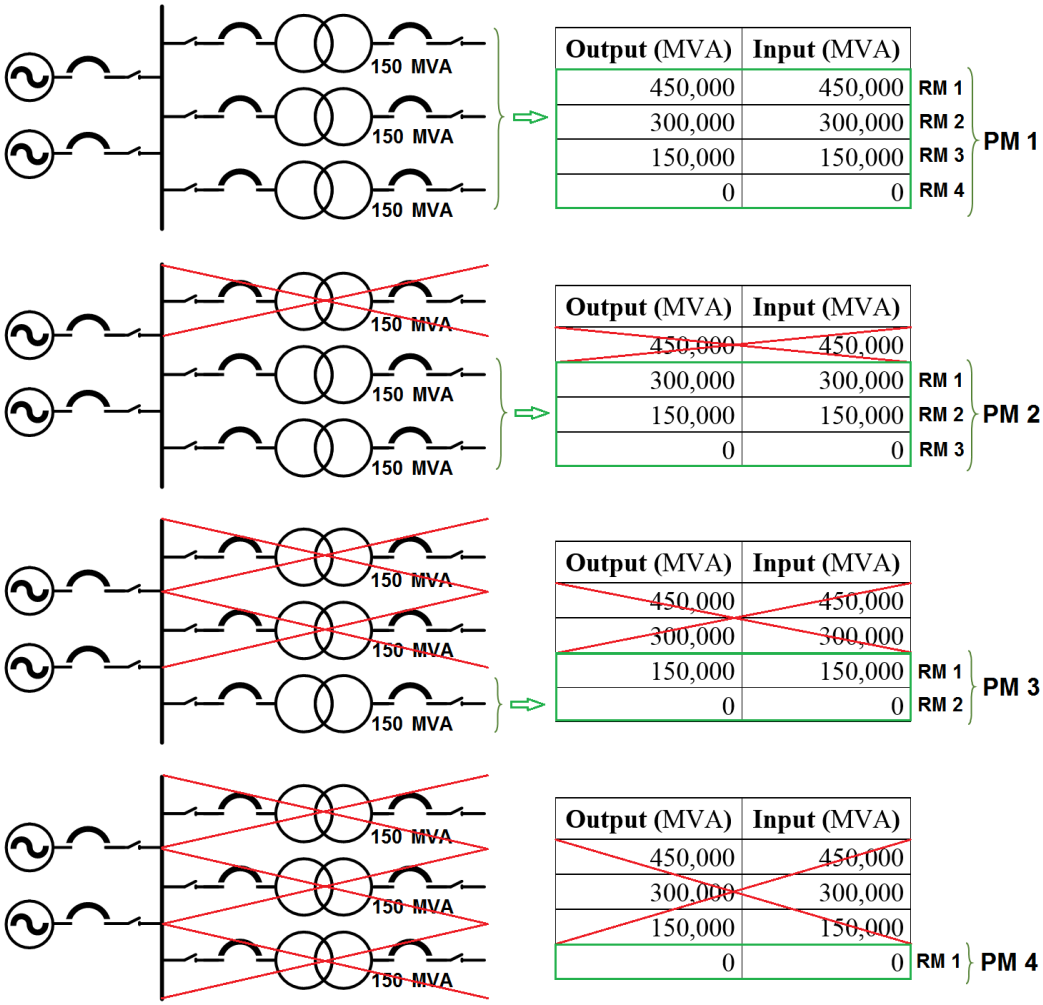


Figure 5.3 Electrical Substation and i2Sim Mapping

Next, we provide a snapshot of the electrical infrastructure as it was modeled in i2Sim, followed by the details and HRTs of every substation.

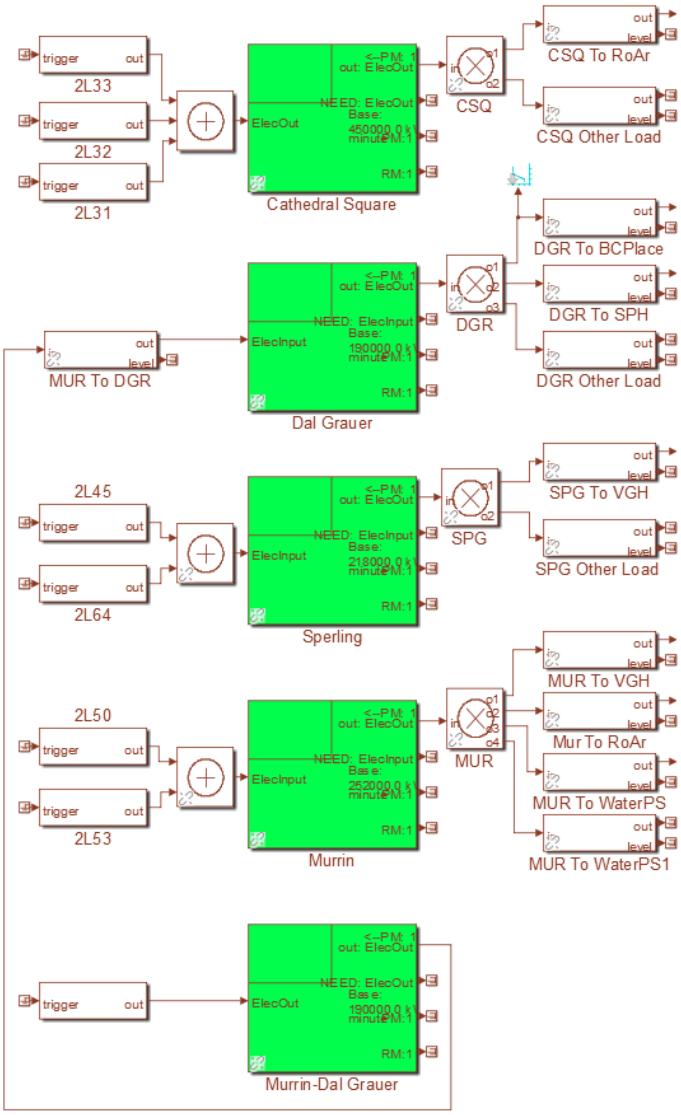


Figure 5.4 Electrical Infrastructure

5.1.1.1 Cathedral Square Substation (CSQ)

CSQ is an underground substation under the Cathedral Square Park at the intersection of Dunsmuir and Richards streets. CSQ is supplied by three 230 kV circuits: 2L33 and 2L32 coming from Horne Payne Substation, and 2L31 from Murrin. CSQ has three 150 MVA transformers, two in service since 1984 and one more added in 2009. For simplification, the loads were assumed as a resistive, thus, MVA will be directly converted to MW.

Output (kW)	Input (kW)
450000	450000
300000	300000
150000	150000
0	0

Table 5.1 CSQ HRT

5.1.1.2 Dal Grauer Substation (DGR)

DGR is an indoor substation, located in Downtown Vancouver at 944 Burrard Street. DGR has a firm capacity of 190 MVA. Since no details about the transformers were found, it was assumed to have three equal transformers, as seen in Table 5.2.

Output (kW)	Input (kW)
190000	190000
127000	127000
64000	64000
0	0

Table 5.2 DGR HRT

Dal Grauer is supplied by four cable circuits and two transformers at Murrin.

5.1.1.3 Sperling Substation (SPG)

Sperling Substation is located at the intersection of Arbutus street and King Edward. SPG is supplied by two circuits: 2L45 from Camosun Substation and 2L64 from Kidd# 2 Substation. Sperling's firm capacity is 218 MVA and its load is served by two transformers.

Output (kW)	Input (kW)
218000	218000
109000	109000
0	0

Table 5.3 SPG HRT

5.1.1.4 Murrin Substation (MUR)

MUR is located at 697-781 Main street. Murrin supplies approximately 60% of Downtown's load. MUR is equipped with three 84 MVA transformers to supply its load. Additionally, MUR has two transformers that supply DGR, as seen in Table 5.5.

Output (kW)	**Input** (kW)
252000	252000
168000	168000
84000	84000
0	0

Table 5.4 MUR HRT

Output (kW)	**Input** (kW)
190000	190000
95000	95000
0	0

Table 5.5 MUR Bypass HRT

5.1.2 Water Pumping Station

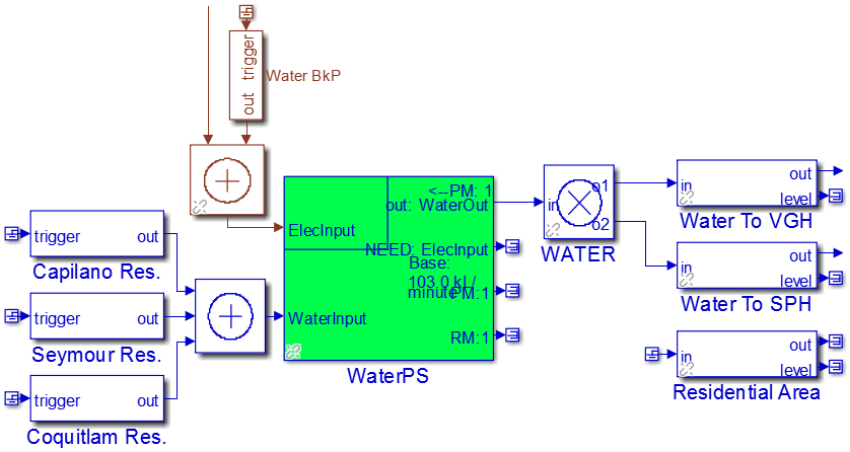


Figure 5.5 Water Infrastructure

This subsystem considers the pumping station as a single (aggregate) cell with an input coming from the three watersheds supplying the Greater Vancouver Area. This infrastructure could be represented with a continuous function, but as Sarsa(0) uses a discrete set of states; a five level HRT: 100%, 75%, 50%, 25% and 0% was used instead. Water allocation focuses on supplying the hospitals, hence residential customers are out of scope. We don't have a geographical location for this infrastructure.

Output (kL/h**)**	**Electricity (**kW**)**	**Water (**kL/h**)**
103	10	103
77	8	77
52	5	52
26	3	26
0	0	0

Table 5.6 Water PS HRT

5.1.3 Venues (Egress)

We consider two stadia at full capacity. A disruptive event forces an evacuation (egress) process. Each venue is represented with two PCs sharing the same inputs. The first PC determines the rate of egress (time to leave the facilities). The second one calculates the amount of people injured due to the event.

The inputs for a venue's HRT are resources and modifiers. Resources relate to physical supplies, i.e., electricity; while modifiers account for human factors. In this work, egress modifiers are: guidance, layout, demographics and rapid response. We have tested these modifiers in [68]. Egress happens even in the absence of resources. At egress, people are TRIAGED on-site. Those in need of medical attention will await transportation to emergency facilities.

The geographical location of the stadia follows:

Figure 5.6 Geographical Location of the Stadia

5.1.3.1 BC Place

BC Place is a stadium located at 777 Pacific Boulevard in Vancouver, BC. BC Place is considered a multi-purpose venue and has the largest retractable roof in the world. This venue has a maximum capacity of 54,500 [71].

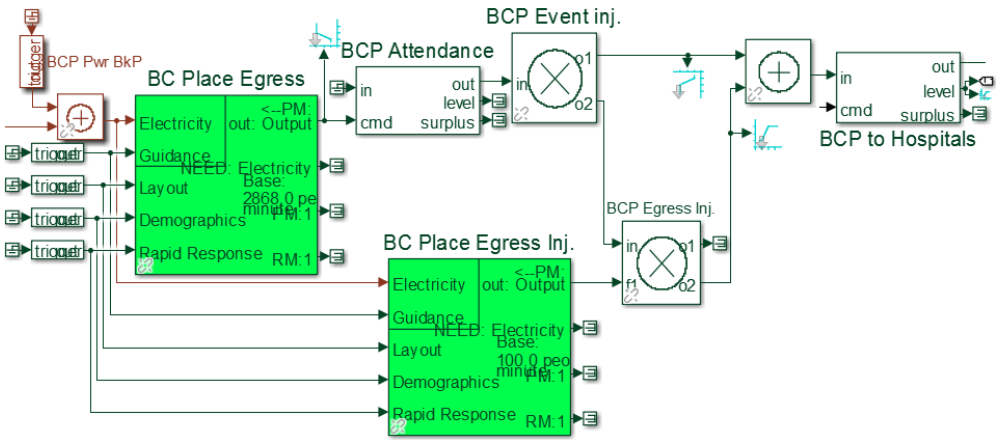


Figure 5.7 BC Place Egress Subsystem

The modeling of BC Place includes two production cells (PCs). While the two PCs represent the same stadium, they serve different purposes. The first PC (upper-left) determines the egress rate. This egress rate is a signal that takes people out of the seating area. The seating area is represented by a storage with a level (current amount of held tokens) equal to the BC Place's maximum capacity, 54,500. Once out, a fixed percentage (2%) of the egressed people are labeled as injured, by a distributor. These people's injuries are a direct consequence of the disruptive event. Following the egress and TRIAGE, the injured people (tokens) are transferred to a second storage, where they will wait for ambulance arrival. Table 5.7 is the HRT used for the BC Place Egress production cell.

Output (People/min)	**Electricity** (kW)	**Guidance**	**Layout**	**Demographics**	**Rapid Response**
2868	2840	1.50	1.63	1.50	1.50
2477	2130	1.25	1.38	1.38	1.38
2180	1420	1.00	1.00	1.00	1.00
2019	710	0.88	0.95	0.95	0.75
1817	0	0.75	0.90	0.90	0.63

Table 5.7 BC Place Egress HRT

The maximum egress output rate (first row) was taken from a reference study where evacuation of a similar venue took 19 minutes [72]. The following rows in the BC Place's egress HRT were calibrated to last 2/3 minutes more, consecutively. Therefore, the maximum time to complete egress is 30/31 minutes. The main assumption is that the process continues even with zero electricity.

Peak power ratings for this venue were not available. A peak of 10 MW is found in a reference for a stadium with of 72,000 capacity [73]. We used this rating and other documents related to

BC Place [74] for calculating the electrical requirements. The maximum electrical value is reduced on equal intervals along the 5 rows in the table, until reaching zero.

The remaining columns in the HRT are modifiers. These modifiers were defined in [75] and they were also used in [68]. The list of modifiers includes factors that have incidence over the egress process. Prior to inclusion in the models, these egress subsystems were tested in parallel with Simwalk [76], a specialized crowd simulator.

The second portion of the egress subsystem is carried out by the second PC. This production cell uses identical inputs to the one just described, but it generates a different output. As opposed to the first PC that outputs an amount of people, this second PC outputs a distributor ratio. This distributor ratio is a parameter input for the rightmost (second) distributor. The second distributor takes the remaining people from the first process (uninjured due to event) and separates them into safe people and injured during evacuation. As seen in the second HRT (Table 5.8 BC Place Egress Inj. HRT), injuries during the egress process happen when lacking some of the input in any column. Thus, no people get injured on full availability of resources and a maximum 4% do when fully lacking any of the inputs. Injured people during egress are combined with injured due to the event in the same storage, where they will be picked up by the ambulances.

Output (% safe People)	**Electricity** (kW)	**Guidance**	**Layout**	**Demographics**	**Rapid Response**
100	2840	1.50	1.63	1.50	1.50
99	2130	1.25	1.38	1.38	1.38
98	1420	1.00	1.00	1.00	1.00
97	710	0.88	0.95	0.95	0.75
96	0	0.75	0.90	0.90	0.63

Table 5.8 BC Place Egress Inj. HRT

5.1.3.2 Rogers Arena

Rogers Arena is in indoor venue property of Canucks Sports & Entertainment. It is located at 800 Griffiths Way in Vancouver, BC. Rogers Arena has a maximum capacity of 18,910 [77].

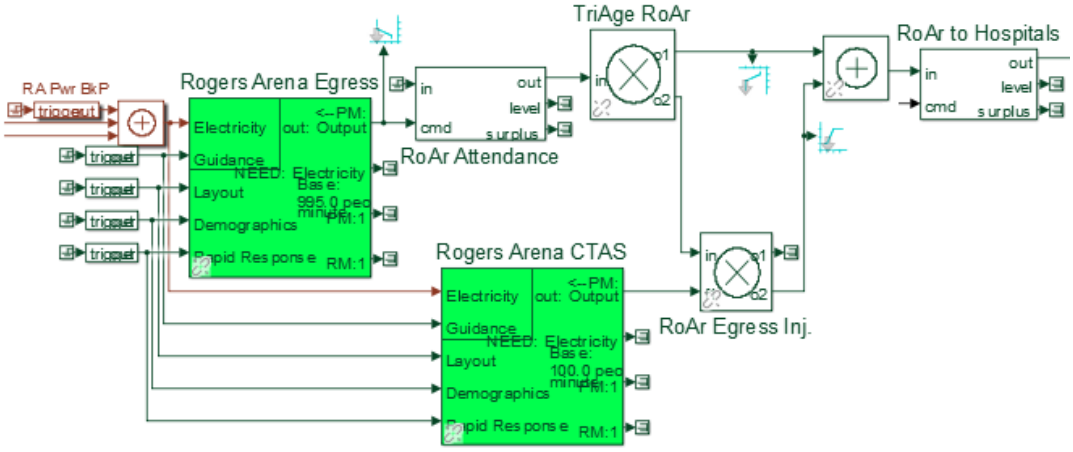


Figure 5.8 Rogers Arena Egress Subsystem

The modeling of this facility follows the same process described for BC Place. For simplicity of modeling, the evacuation process will last the same as in BC Place. People injured due to the event are 3% of all attendees for this venue. Finding electrical consumption ratings was challenging. A study in [78], collects daily energy ratings of venues. A value of 10 MWh is found as the peak. If we consider an event duration of 4 hours, we can assume 2.5 MW for the leading row of our HRT. Following rows will have equal reductions until reaching zero. See Table 5.9:

Output (People/min)	**Electricity** (kW)	**Guidance**	**Layout**	**Demographics**	**Rapid Response**
995	2500	1.50	1.63	1.50	1.50
860	1875	1.25	1.38	1.38	1.38
756	1250	1.00	1.00	1.00	1.00
700	625	0.88	0.95	0.95	0.75
630	0	0.75	0.90	0.90	0.63

Table 5.9 Rogers Arena Egress HRT

The HRT for the second component of the egress is almost identical to the one used for BC Place. The only difference goes in the electricity column, matching Rogers Arena's selected ratings. See Table 5.10:

Output (% safe People)	Electricity (kW)	Guidance	Layout	Demographics	Rapid Response
100	2500	1.50	1.63	1.50	1.50
99	1875	1.25	1.38	1.38	1.38
98	1250	1.00	1.00	1.00	1.00
97	625	0.88	0.95	0.95	0.75
96	0	0.75	0.90	0.90	0.63

Table 5.10 Rogers Arena CTAS HRT

5.1.4 Transportation

The transportation subsystem consists of two distributors, eight channels and one source. An illustration of the transportation model is available in Figure 5.9:

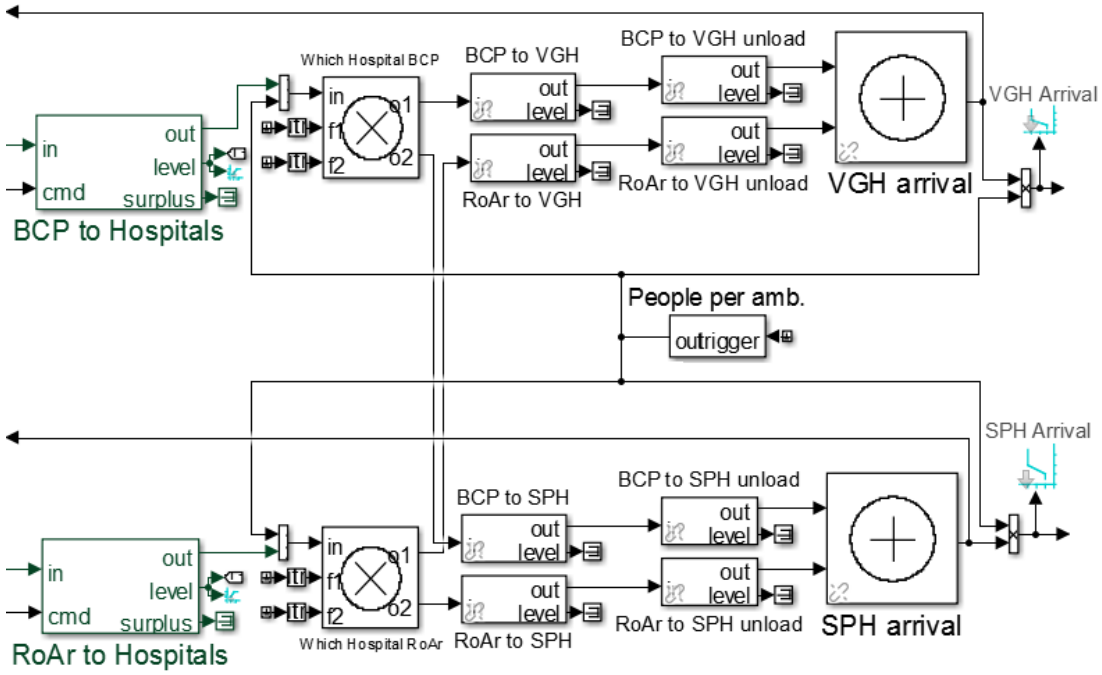
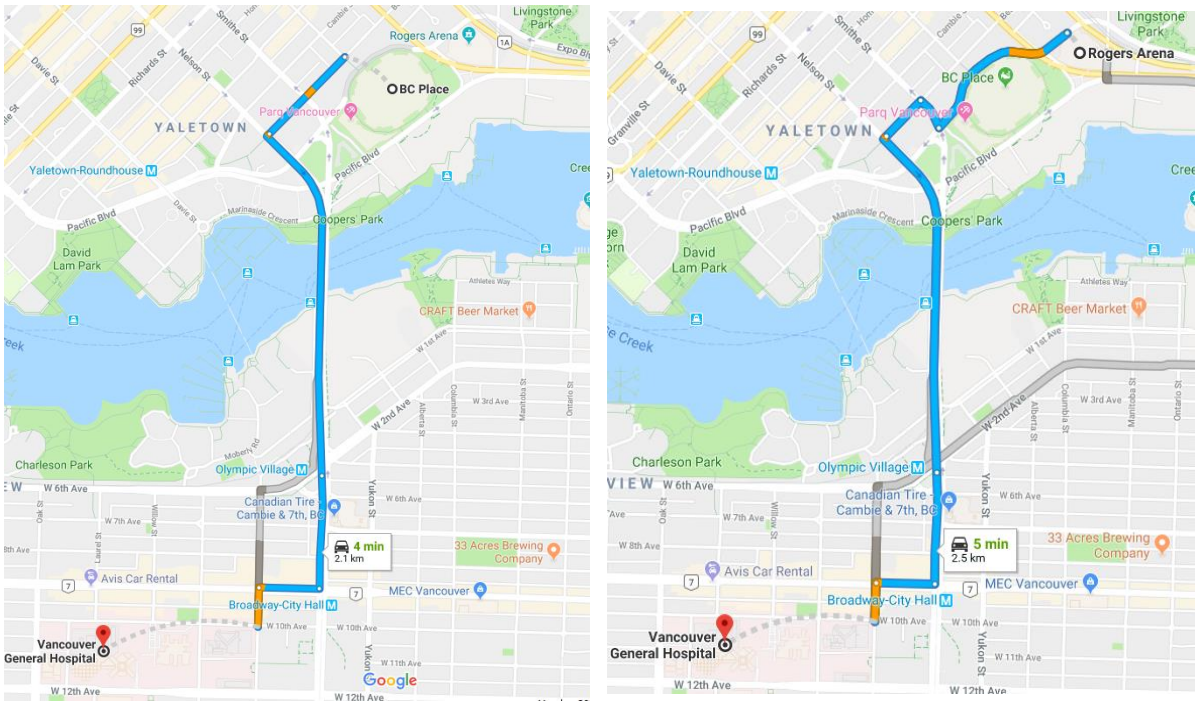


Figure 5.9 Transportation from Venues to Hospitals

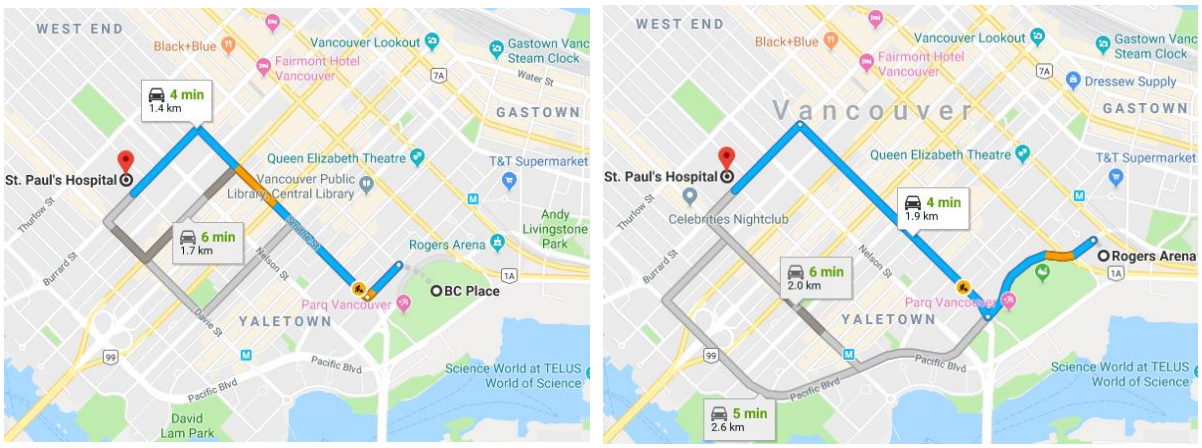
The two distributors, one at each venue, were used for choosing a destination hospital. Four channels simulate the routes from each venue to each hospital. Likewise, four additional channels account for unloading time and reading the ambulance to go back for more patients. The number of people per ambulance, can be adjusted by changing the value of the source *People per amb*.

The travelling times were determined using Google maps, with the real location of venues and hospitals. We assume low traffic as the roads modeled are designated disaster response routes. Traveling time, for ambulances going back to venues, was assumed to be 5 minutes, for matching the initial arrival time. In most cases this time matched Google maps estimations. More details about the transportation route times can be found in Figure 5.10, Figure 5.11 and Table 5.11.



BC Place to VGH Rogers Arena to VGH

Figure 5.10 Time to Arrival to VGH



BC Place to SPH **Rogers Arena to SPH**

Figure 5.11 Time Arrival to SPH

Start	Destination	Time (minutes)
BC Place	Vancouver General Hospital	4
BC Place	Saint Paul's Hospital	4
Rogers Arena	Vancouver General Hospital	5
Rogers Arena	Saint Paul's Hospital	4
Vancouver General Hospital	BC Place	5
Vancouver General Hospital	Rogers Arena	5
Saint Paul's Hospital	BC Place	5
Saint Paul's Hospital	Rogers Arena	5

Table 5.11 Summary of Transportation Route Times

5.1.5 Hospitals

The representation of the hospitals follows the same subsystem implementation as used in the aggregate case. Electrical and water ratings, as well as other values, were acquired via interviews with managers and technical personnel from both hospitals. As both hospitals are modeled in the same way, we display only one of them in the illustrating figures. HRTs have the same structure but different values, as VGH has a larger capacity than SPH. Geographical location and overview of the hospitals, along with their corresponding HRTs are provided below:

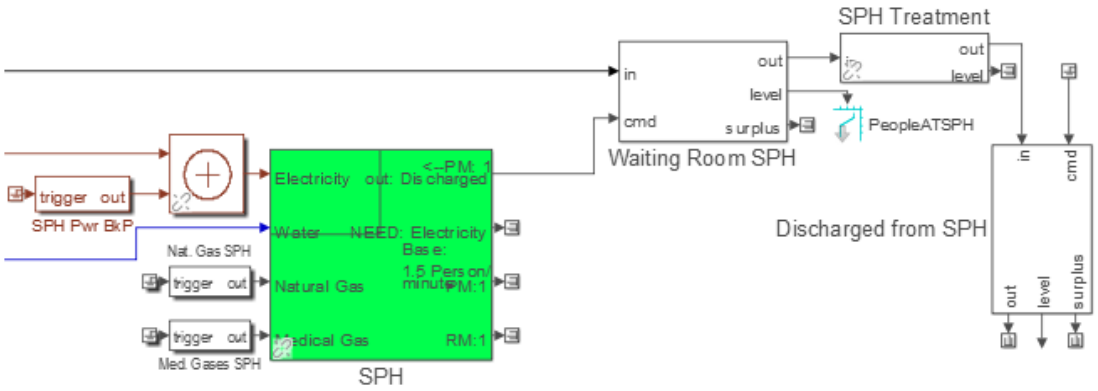


Figure 5.12 Hospital Model

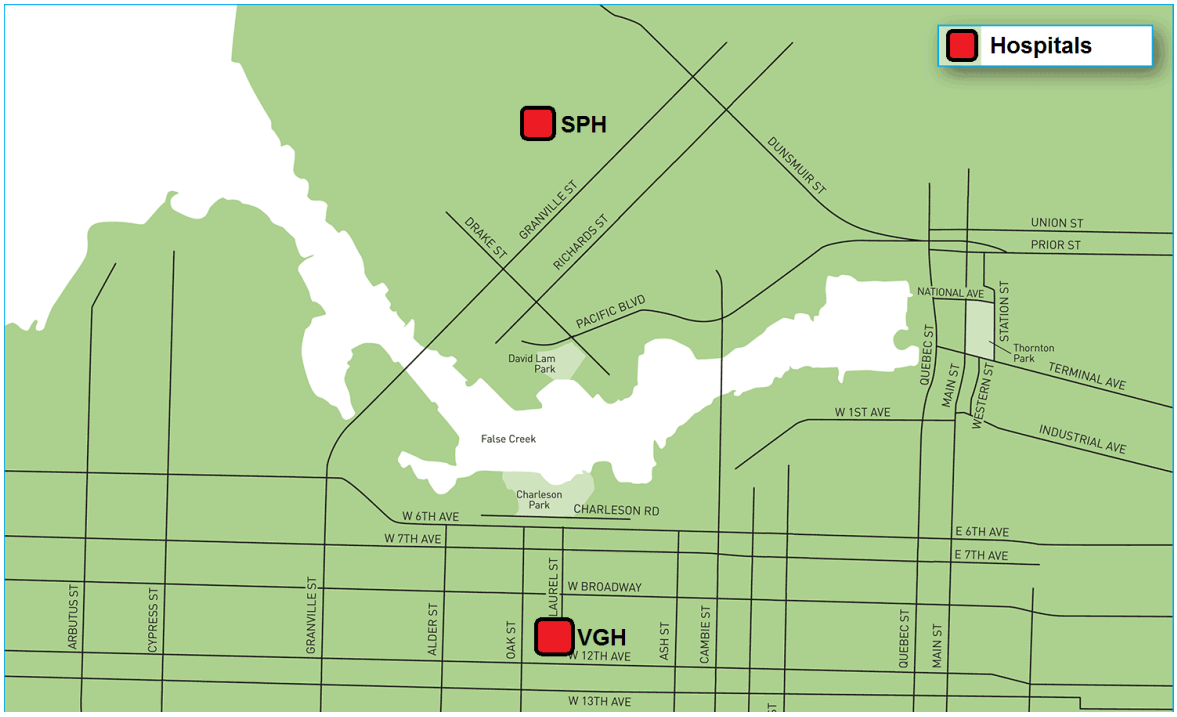


Figure 5.13 Geographical Location of the Hospitals

5.1.5.1 Vancouver General Hospital (VGH)

Vancouver General Hospital is the second largest hospital in Canada. VGH is the largest hospital in BC and receives patients from other regions due to its specialized services. It is located at 899 West 12th Avenue in Vancouver, BC [79]. HRT taken from [5] and [68].

Output (People to treatment/h)	**Electricity (**kW**)**	**Water (**kL/h**)**	**Natural Gas (**ft^3/h**)**	**Med. Gas (**%**)**
10	2000	51	3333	100
7	1500	38	2500	75
5	1000	26	1667	50
2	500	13	833	25
0	0	0	0	0

Table 5.12 VGH HRT

5.1.5.2 Saint Paul's Hospital (SPH)

Saint Paul's Hospital is located at 1081 Burrard Street in Vancouver, BC. SPH is an acute care, teaching and research hospital and has one of the busiest emergency departments in the province [80]. HRT taken from [5] and [68].

Output (People to treatment/h)	**Electricity (**kW**)**	**Water (**kL/h**)**	**Natural Gas (**ft^3/h**)**	**Med. Gas (**%**)**
10	1000	38	2499	100
7	750	26	1875	75
5	500	13	1250	50
2	250	7	624	25
0	0	0	0	0

Table 5.13 SPH HRT

5.2 The Decision Layer (Agents)

The decision layer has three independent, non-interfering agents. The agents are assigned to egress (dispatcher), transportation (On-site supervisor), and resource allocation (RL agent). The egress and transportation processes use a shorter time frame as compared to resource allocation. The dispatcher and the on-site supervisor have fixed behaviours, i.e., they know how to proceed beforehand. The RL agent experiments actions with the aim of learning the best possible outcomes. This RL agent extends the one implemented for the aggregate case. As the RL agent is a key component to this thesis, it will be covered in more detail.

5.2.1 The Dispatcher

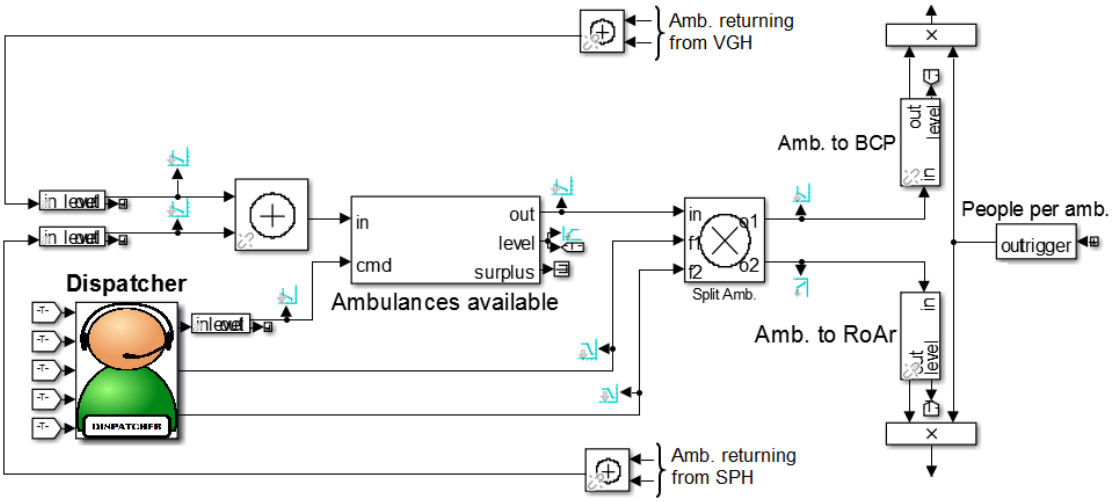


Figure 5.14 Dispatcher Agent

The Dispatcher knows the number of people in need of emergency transportation, at every venue. The ambulance dispatching model uses a storage cell to represent the pool of available ambulances. It outputs all the ambulances at every time step. A distributor right out of the storage allocates the ambulances to each hospital. In case of not needing as many ambulances as available, a third stream sends the remaining ambulances back to availability as if they never left their standby location. Once the ambulances have delivered the patients to the corresponding hospitals, they become immediately available as the traveling time has been already accounted for at the dispatch point. The ambulance allocation is proportional to the number of people waiting at each venue. Thus, if the number of injured people is the same, at both venues, a 50-50 percent split is expected.

The BC Emergency Health Services (BCEHS), manages ambulances in the province of BC [81]. It is the largest emergency services provider in Canada. BCEHS' fleet include:

Amount	Description
463	Basic life support ambulances staffed by primary care paramedics and emergency medical responders.
28	Advanced life support ambulances staffed by advanced care paramedics.
7	Ambulances dedicated to the Critical Care Transport Program staffed by critical care paramedics.
5	Ambulances outfitted with specialized neo-natal, pediatric and obstetric equipment for the paramedic Infant Transport Team.
34	Modified ambulances used as medical support units, decontamination units and integrated communications units for large-scale responses.
68	Support vehicles for the community paramedicine program.
73	Support vehicles for duty supervisors, primary response units, telecommunications, BCEHS learning and special operations/events.

Table 5.14 BC Emergency Health Services Ground Fleet [81]

We assume a total of 35 ambulances available. All ambulances are considered to be at a 5 minute radius from the venues. Without this assumption, the dispatching model would be more complex than the entire test case. Future studies could include more complex subsystems, or even external applications for modeling transportation. The capacity of the ambulances (number of people) can be adjusted with the same parameter used in the transportation subsystem. We used ambulances with 2 people capacity in this case.

During the construction of this model we added a new component to the i2Sim toolbox, an integer distributor. This component was described in the i2Sim overview and it was needed for ambulance allocation. The integer distributor provides a more realistic approach. Without this element, the simulation could have fractions of people traveling to different hospitals. With the

integer distributor, split ratios are sometimes more indicators than values, e.g., one ambulance available would go to the venue with a ratio higher than 50%.

5.2.2 The On-site Supervisor

This agent is the simplest of the three. Its behaviour is rigid i.e., it makes decisions that are not situation dependent. This on-site supervisor is an agent directing ambulance traffic to a specific hospital. These tasks could also use a machine learning agent in the future.

The on-site supervisor is modeled with two fixed ratio distributors. From all ambulances departing from BC Place, 60% go to VGH, the larger hospital; the remaining (40%) are directed to SPH. All ambulances departing from Rogers Arena are evenly distributed (50-50) to both hospitals. This fixed policy is assumed to be predetermined by emergency plans but is not based on real policies used by the municipality. This agent acts in the transportation subsystem as seen in Figure 5.15:

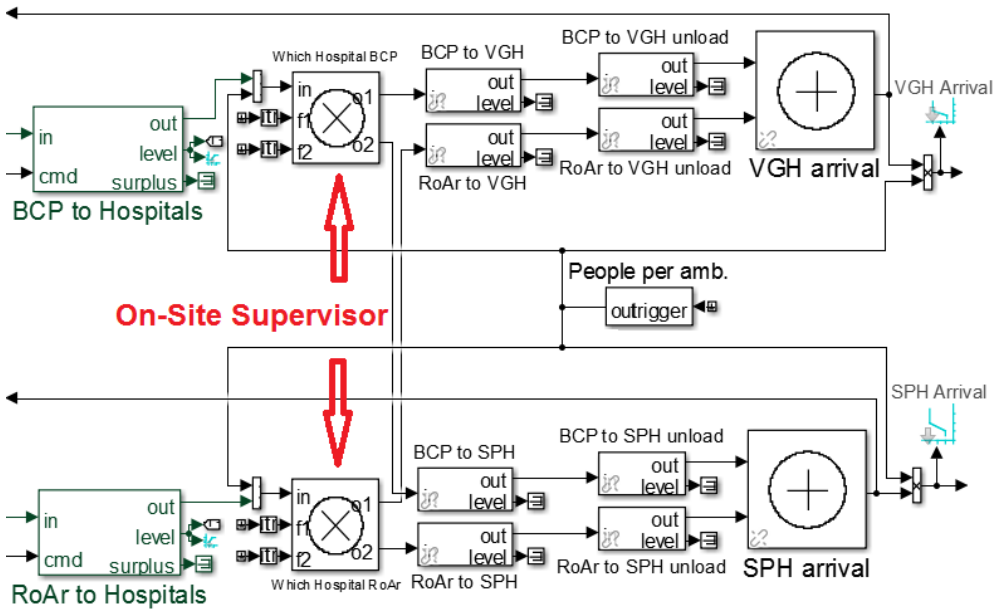


Figure 5.15 The On-Site Supervisor

5.2.3 The RL Agent

The RL agent is an expanded version of the formulation described in Chapter 3. By handling more CIs, the state descriptors have multiplied. Likewise, the set of actions is larger. The dimensionality of the problem increased by a factor of 4,169 compared to the proof of concept case. Details about the scaled RL setup follow.

5.3 The Sequential RL Agent

Following the same procedure used for the aggregate test case, we approached the agent representation as an MDP. Many of the features and settings used in the previous case are applicable and remain unchanged. Hence, we emphasize on the changes more than the similarities.

5.3.1 State Space

The state space is formed with the operational indexes (OIs) of the following infrastructures: electrical substations controlled by the agent, water pumping station and the hospitals. Additionally, we added an availability flag for electrical backup at each hospital. We selected three out of four substations based on which ones supply the hospitals and the water station.

We don't consider the venues sensitive to the agent's acting as the egress process is not interrupted by the lack of electricity. Table 5.15 shows the list of state variables selected and their corresponding number of operational indexes (OIs). A combinatorial of the individual OIs yields *8,100,000* states. The value is significantly higher than the 50,625 possible states in the previous case.

DGR	SPG	MUR	BKP VGH	BKP SPH	WPS	VGH	SPH
10	6	10	2	2	15	15	15

Table 5.15 State variables and Number of States

5.3.2 Action Space

The agent has control over four decision points, three electrical and one water distributors. The set of actions used obey to physical constraints or the best-known way of splitting resources. Electrical actions use ratios that match the peak rating stated in the hospitals HRTs, all other loads are grouped as *Other*. Electrical actions are taken at the distribution points part of DGR, SPG and MUR substations. The available actions to the agent are summarized below:

	BCP	SPH	Other
1	1.49%	5.26%	93.25%
2	2.23%	7.87%	89.90%
3	4.42%	15.63%	79.95%

Table 5.16 Electrical Actions at DGR

	VGH	OTHER
1	9.17%	90.83%
2	18.35%	81.65%

Table 5.17 Electrical Actions at SPG

	VGH	RoAr	WPS	OTHER
1	7.94%	1.01%	0.010%	91.04%
2	11.90%	1.52%	0.015%	86.57%
3	30.77%	3.92%	0.039%	65.28%
4	11.90%	3.92%	0.039%	84.14%
5	7.94%	0.00%	0.015%	92.05%
6	7.94%	0.00%	0.039%	92.02%

Table 5.18 Electrical Actions at MUR

The combination of these three tables into unified electrical actions yields 36. All state variables (SV) and decision points (DP) are marked in Figure 5.16.

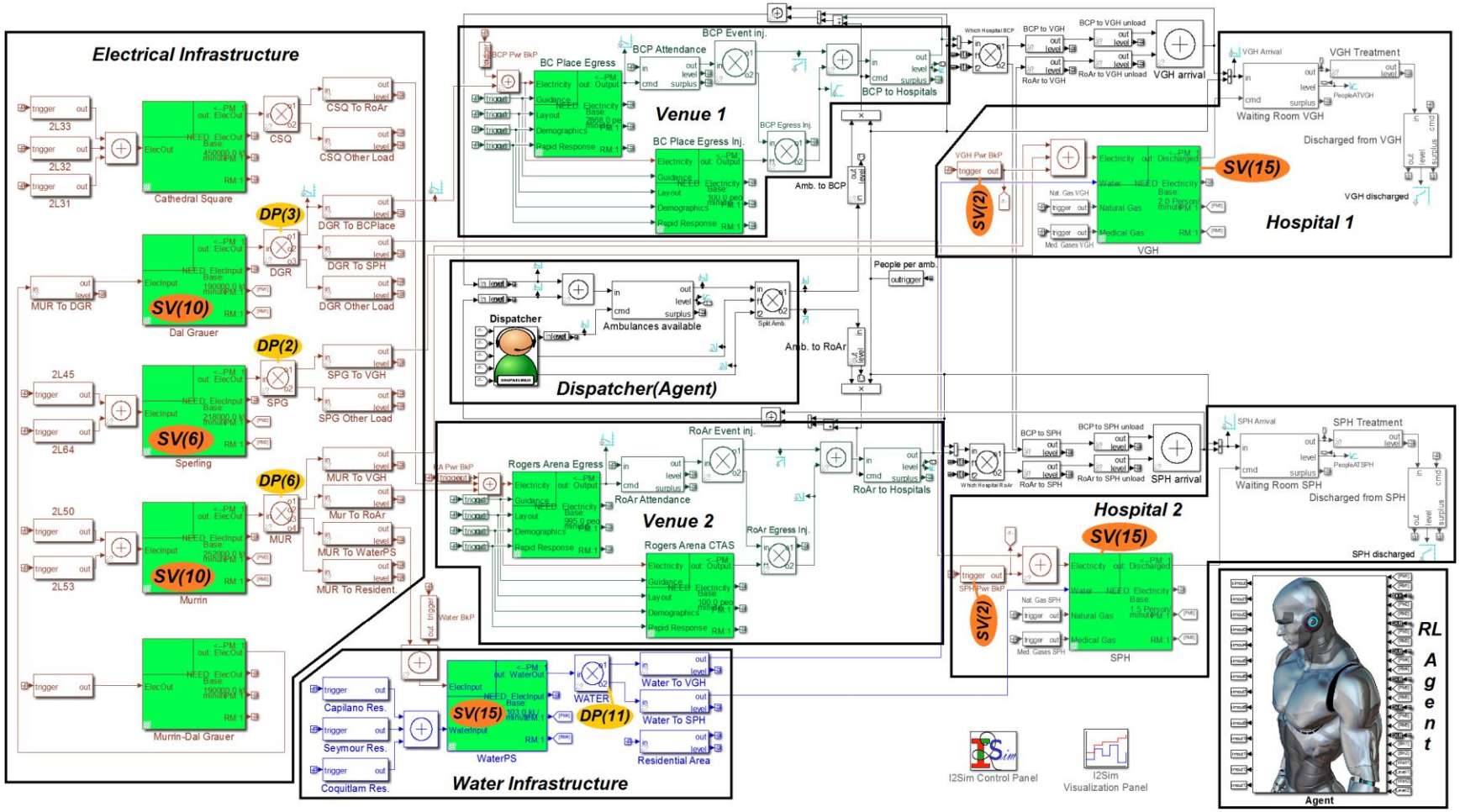


Figure 5.16 Extended Model with State Variables and Decision Points

For the water pumping station, the distribution offers more flexibility. We decided to use 10% intervals to generate *11* discrete combinations as follows:

VGH	SPH
100%	0%
90%	10%
80%	20%
70%	30%
60%	40%
50%	50%
40%	60%
30%	70%
20%	80%
10%	90%
0%	100%

Table 5.19 Water Distribution Ratios

Water distribution only happens between hospitals. Like in the aggregate test case, a coordinated response considers one action as a combination of settings at all distributors. By combining all decision points, the action space has a total of *396* feasible actions.

5.3.3 Policy and Rewards

Reward generation and policy remain the same. The only difference, in the reward scheme, is the use of a composite signal with the feedback from both hospitals.

5.3.4 Setup and Results

The size of the look up table (LUT) is calculated as $state\ space * action\ space$. This calculation reports *3,207,600,000* elements in the matrix based LUT. As mentioned before, MATLAB uses double as its default datatype. Thus, the size of the LUT is 23.89 GB. Due to

similarities in formulation with the small test case, the LUT partitions and bands have the same size and repeating patterns listed in Table 4.8.

The Model was configured to run for 720 timesteps, an equivalent to 12 hours. Four simultaneous training instances were launched in a regular PC. While the resource utilization was topped, all instances finished at the same time and no impact on running time was detected. In execution, episodes averaged 85.94 seconds with full graphical aids and 48.03 seconds without the visualization panel (results' plots).

We decided to test two different configurations in sequential scope. The first, a baseline that matches the criteria used when testing the aggregate case, a fully operational scenario. With no damage included, this baseline allows a validation of the agent choosing the right policy when all infrastructures achieve a steady 100%. Results for this case can be seen in Figure 5.17.

The second setup uses an LUT partition located 135,000 states apart from the zero and features electrical limitations. The corresponding results are illustrated in Figure 5.18. Although, this is not yet a distributed test, we had to use LUT strips to exercise different sections of the state space. Furthermore, running tests over LUT partitions enabled us to verify the extra parameters needed by the agent for handling the large distributed test.

As seen in the results, we let the training run for four episodes. The plots from episodes 3 and 4 are identical at each hospital. This indicates that the process converged. The agent learned the policy after 3 runs. Thus, we achieved a fast and guaranteed convergence with better times than in the aggregate test case.

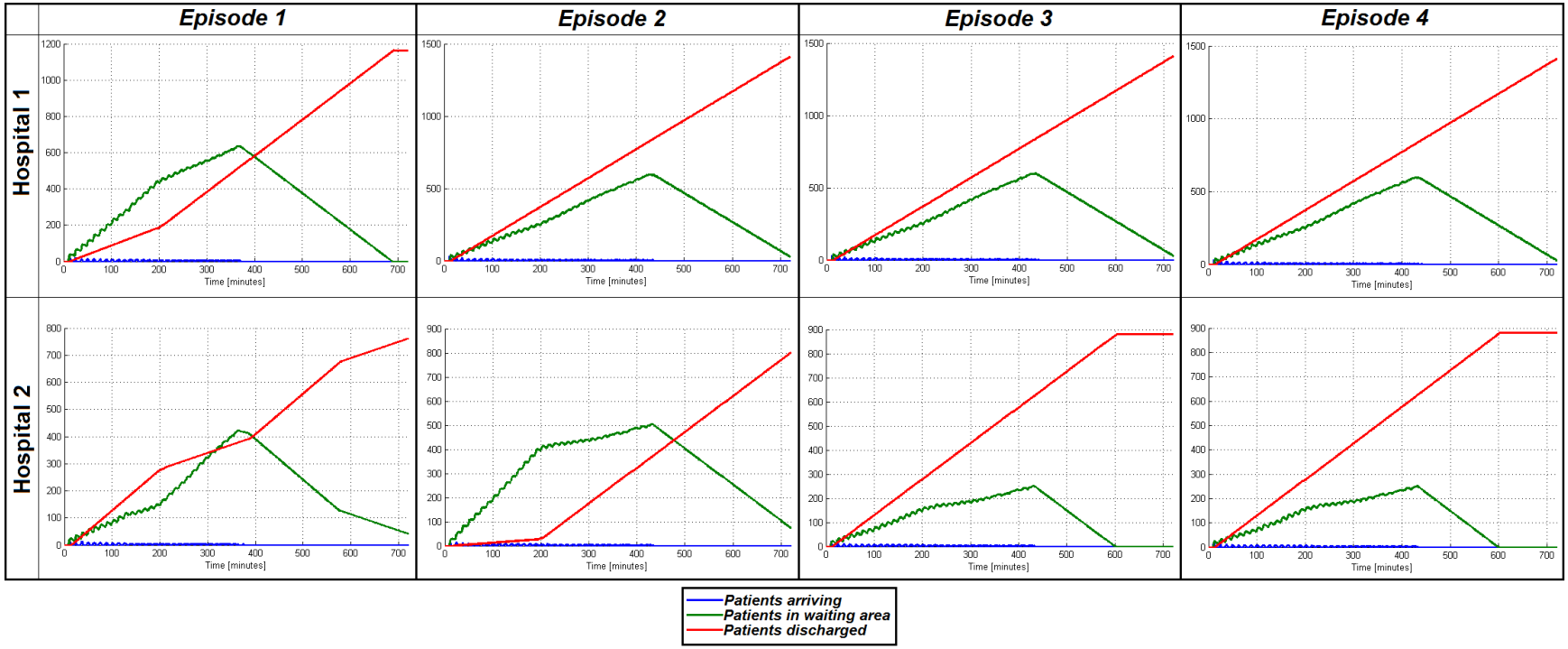


Figure 5.17 Baseline results. Model running with full operational status

As seen in the plots, the red line indicates the number of patients treated. This red line has an ascending slope and then it flattens, at this point all patients have been treated.

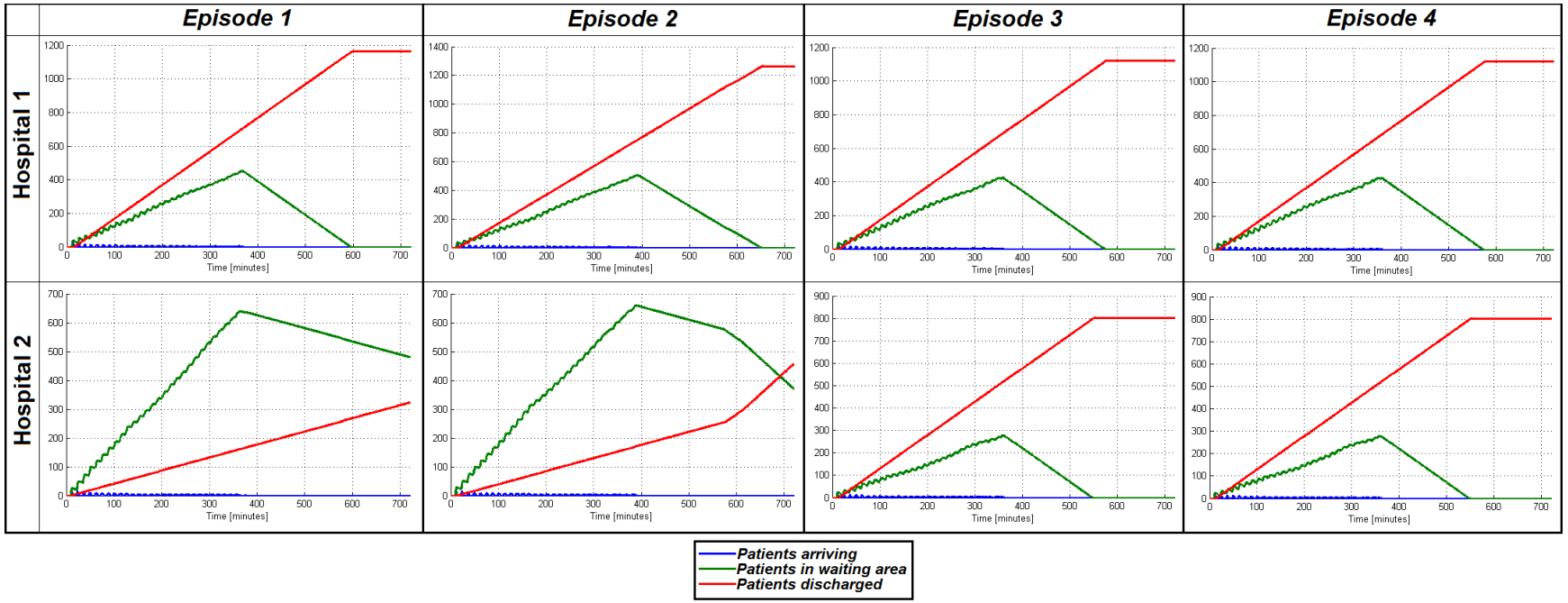


Figure 5.18 Results of baseline model with limited electricity at one substation

To verify an achieved improvement over our past projects, we collected the results and characteristics of the closest solutions using very similar configurations. The three solutions used as a reference for comparing our current results were labelled by the author in [6] as: DAARTS (Decision Assistance Agent in Real-Time Simulation), IDS (Intelligent Decision System) and UOP (Ultimate Optimization Package). We will use the same acronyms during the comparison, more details to be provided as follows:

	DAARTS	**IDS**	**UOP**	**This Thesis**
# States (Total)	225	3,375	3,375	8,100,000
# Actions	110	550	550	396
LUT Size (Total)	24,750	1,856,250	1,856,250	3,207,600,000
Time simulated	10 hours	10 hours	10 hours	12 hours
Timestep	5 minutes	5 minutes	5 minutes	1 minute
RL Method	TD Q-learning	Montecarlo	Montecarlo	TD Sarsa
Software running	Simulink (i2Sim) Java (RL Agent)	Simulink (i2Sim) Java (RL Agent)	Java (Model) Java (RL Agent)	Simulink (i2Sim + RL Agent)
Time per episode	6.21 minutes	3.25 minutes	4 seconds	48 seconds
# Episodes run	100	100	100	4
Training Time	621 minutes	325 minutes	6.67 minutes	3.20 minutes

Table 5.20. Sequential Implementation Results vs Past Related Projects

As seen in Table 5.20, the evolution in the problem's formulation goes from left to right. The reference approaches are subsequent improvements over versions of the same case. Listed values for state space and LUT size are based on the entire state space targeted for optimization in each implementation. Thus, *#States* corresponds to all the states that can be mapped, in each case, with the chosen state variables. The formulation of this thesis overcame the latency problems found in DAARTS and IDS and improved the convergence speed. A mere look at the times the full training required to complete, shows the current solution being faster than

DAARTS and IDS but the difference is higher than it looks. This works samples more timesteps during training. A resolution of 1 minute in a 12 hour simulation yields 720 timesteps. The two reference solutions (DAARTS and IDS) ran for only 120 timesteps (10 hours with a 5 minute resolution). The same is applicable to UOP in regard to time, but a further significant difference is present. UOP doesn't run i2Sim, instead it optimizes over a manual model extraction coded in Java. With these two considerations the analysis of UOP as a solution is easier to understand. This work samples 6 times more interactions with the environment. If times are adjusted UOP would take 24 seconds instead of 4. That makes UOP only twice faster, per episode, than the current project. As this thesis kept a fully functional i2Sim that difference is not significant in the time scale the solution are processed, 48 seconds vs 24 seconds. Moreover, the total training time require by the agent part of this dissertation is 50% of what UOP requires.

The results obtained and evaluation against our previous projects suggest that this solution is our best sequential and it is ready to be parallelized.

5.4 Distributed Training and The Scheduler

If the totality of the state space was to be explored, we would have over 10,000 partitions to exercise. By following the same tactic used in the previous case, we would have 10,000 simultaneous running instances. Even with the available cluster, this partitioning scheme would not offer a good balance between memory and CPU usage. Moreover, the goal of this implementation is to ease the work of decision makers; having results from all possible situations is overwhelming. We decided to run the training over 24 bands of the LUT. As

explained before, a band is a grouping of 4 partitions with sizes: 1125, 900, 675 and 450 respectively. Therefore, the number of instances run, with their corresponding LUT partitions, was 96, in total.

The architecture of the scheduler is the same described in previous sections. One improvement done to the agent was the addition of an offset factor. This enabled the agent to map states from 1 to *maxnumber* (1125, 900, 675 or 450) regardless of the value of the state. This parameter was not added before because the LUT was small and it could be entirely stored in every node. A second addition involved improvements in the management of parametric tables and LUT partitions. While these modifications made the setup, easier there is still some effort required to configure the environment.

As the distributed training happens in the absence of any graphical user interface, running times shrink. Additionally, the iterations (episodes) per partition were reduced to 4. The $\in -greedy$ scheme remain unmodified, but when running 4 episodes we used the first 4 exploration values, only (see Table 4.7). The training of all instances (96) was completed in ***2.86 minutes***.

Chapter 6: Conclusion

In this thesis we discussed an approach for optimizing multi-scenario models in the context of emergency response. The work parallelizes the training process of reinforcement learning (RL) agents to considerably speed up the performance and size of disaster response scenarios in critical infrastructure modeling. The bridge between critical infrastructures and emergency management is provided by the Infrastructure Interdependencies Simulator i2Sim. With i2Sim, the optimization process for resource allocation takes into account the cascading effects of every decision made. Speed of convergence and parallel processing capabilities are needed for this framework to prove robustness when used in real-time critical applications. The contributions made by this work are specially focused on reducing the convergence time of RL optimizations over i2Sim models, as well as minimizing the impact of dimensionality constraints. Details about the contributions enclosed in this thesis, are covered in the next section of this chapter. To close this chapter, we discussed a series of enhancements that can be added to the achievements of this work in the search of continuous improvement.

6.1 Contributions

A revised version of the City of Vancouver i2Sim model part of our case library. Improved speed of convergence for RL scenarios by adding shaping rewards and a variable exploration/exploitation scheme. A parallel distributed reinforcement learning experimental framework. The previous list of contributions is expanded in the subsequent items of this section, as follows:

1. ***Revised Version of the City of Vancouver Model:*** Since the completion of the 2010 Olympics project, sponsored by DRDC [5], the City of Vancouver (COV) model has been foundational to multiple projects. The revisions added in this thesis include: validation of the data used in the HRTs and distributors, up to what the availability of data allowed. Remodeling of subsystems with the aim of reducing complexity of implementation. The reduction in complexity enables faster scenario solving without losing functionality. Likewise, the addition of the integer distributor to i2Sim's library permits a more realistic split of purely integer tokens. Finally, the identification of multiple decision points delivers an opportunity of including multi-objective optimizations, with either cooperative or independent agents. This rework over the model intends to provide a polished baseline for future projects.

2. ***Convergence and Robustness:*** The identification of shaping rewards was key in reducing the episodic convergence of the solution. Initially, the enhancements done to the original reward scheme felt rather simplistic. Once the new reward scheme was tested, the gain in speed was of a large scale. The solution went from converging after 50 episodes to needing only 5. Likewise, by adding an $\in -greedy$ scheme with decreasing exploration made the agent's training smoother. Having a high and constant rate of exploration is not aligned with real life learning. When learning, the agent acquires knowledge based on experience; with more knowledge the need to try random actions lowers. Both additions created the right solution to be put in parallel, the best sequential implementation. The distributed approach proposed, runs multiple sequential problems in parallel. Thus, the best sequential solution is foundational.

3. ***Parallel Distributed Reinforcement Learning Framework:*** The starting point of the distributed implementation is the identification of decoupling points. This independence of variables is achieved by locating state variables that remain unaffected after the agent performs an action. The selection of exogenous variables is to be taken with care. While many independent points can be located, only the ones with high significance to the optimization goals are to be used. The methodology used in this thesis has work in splitting the Look-Up Table (LUT) into independent sub-matrices. The advantages obtained, include the capacity of handling large dimensionalities by means of partitioning. Likewise, the need of queuing sequential scenarios, for exploring different model configurations, is replaced by launching them in parallel. Hence, the execution time goes from $t_T(N) = N * sqt$ to just $t_T(N) = 1 * sqt$; where $t_T(N)$ is the total time to train for N scenario configurations and sqt is the time needed for a sequential training to complete.

A scheduler including a set of routines was developed to manage the distributed training of the decision-making agent. This scheduler triggers the optimization by launching multiple instances of the same model, via parallel shell. Each instance receives a customized set of parameters and a local look up table. The scheduler starts running in xCAT, the clustering software, and passes the control to MATLAB where i2Sim runs natively. Splitting the scheduler's implementation reduces communication overhead while provisioning simulation in the loop. This architecture handled the training of 96 simultaneous scenarios with a total training time of 2.86 minutes. The results are promising, and the methodology is worth of a continued development that allows extensive testing of larger cases. The scheduler was tested on a regular PC as well as in an HPC cluster, with positive results in both.

In the beginning, we believed that the LUT size was our biggest constraint because of its matrix format. Specially, when simulating large models; but moving from the small test model to the large case did not impact the running time. Our results point to the number of parallel instances as more impactful, due to its significant demand of CPU resources. Hence, CPU becomes the bottleneck as opposed to memory usage.

6.2 Proposed Future Work

On the practical side, one of the most challenging parts during the course of this project was the acquisition of data. A continuous collaboration with emergency practitioners and utility companies can help into establishing permanent links with the possibility of increasing accuracy and efficiency of projects like this.

On the technical side, the expansion of artificial intelligence techniques opens a number of opportunities for expanding this work. Trending topics like deep learning have open space for hybrid techniques like deep reinforcement learning. This thesis verified the suitability of distributed reinforcement learning applied to i2Sim models. Considering emerging techniques like deep reinforcement learning for extending the framework presented in this thesis is a step in moving towards very large model representations.

On the other hand, reduced-order implementations capable of running in regular PCs expand the sequential versions capabilities with no need of high-performance computing equipment. While the number of scenarios would be limited, in some cases, the evaluation of a few model configurations could be enough for small scale incidents.

On the modeling part, we would like to see multiple RL agents in the same model for testing cooperative roles and a composite goaled optimization. This approach could yield better results in complex cases as it would grow memory-wise while keeping the number of instances unchanged in the most part. Moreover, this suggestion would make our model optimization more cohesive with the agents communicating one another.

Bibliography

[1] E. Bournay, "Trends in natural disasters," Grid Arendal, 2009. [Online]. Available: http://www.grida.no/resources/7795. [Accessed 10 05 2018].

[2] S. A. Nelson, "Natural Disasters & Assessing Hazards and Risk," 09 01 2018. [Online]. Available: http://www.tulane.edu/~sanelson/Natural_Disasters/introduction.pdf. [Accessed 10 05 2018].

[3] T. E. Drabek and J. Evans, "Emergency Management and Homeland Security Curricula: Contexts, Cultures, and Constraints," Department of Sociology and Criminology, University of Denver, Denver, Colorado, 2007.

[4] U.S. Department of Homeland Security, "What Is Critical Infrastructure?," 8 12 2017. [Online]. Available: https://www.dhs.gov/what-critical-infrastructure. [Accessed 27 05 2018].

[5] M. Kough and J. Martí, "Modeling Critical Infrastructure Interdependencies in Support of the Security Operations for the Vancouver 2010 Olympics," 2010.

[6] M. Khouj, *Resource allocation optimization of a disrupted interdependent system using machine learning*, Vancouver, BC: PhD dissertation, 2014.

[7] A. Alsubaie, *Improving Critical Infrastructure Resilience with Application to Power Distribution Networks*, Vancouver, BC: Doctoral Dissertation, 2016.

[8] M. Khouj, C. López, S. Sarkaria and J. Marti, "Disaster management in real time simulation using machine learning," in *24th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, Niagara Falls, 2011.

[9] M. . T. Khouj, S. Sarkaria, C. Lopez and J. Marti, "Reinforcement Learning using Monte-Carlo Policy Estimation for Disaster Mitigation," in *8th IFIPWG11.10 International Conference, ICCIP 2014*, Arlington, VA, USA, 2014.

[10] T. E. Drabek and G. J. Hoetmer, Emergency management : principles and practice for local government, Washington, D.C: International City Management Association, 1991.

[11] Province of British Columbia, "The All-Hazard Plan," Emergency Management British Columbia, Victoria, BC, 4/11/2012.

[12] S. A. Nelson, "Natural Disasters & Assessing Hazards and Risk," 19 Aug 2014. [Online]. Available: http://www.tulane.edu/~sanelson/Natural_Disasters/introduction.pdf. [Accessed 31 August 2017].

[13] Federal Emergency Management Agency (FEMA), "Introduction to Emergency Management," 21 Nov 2006. [Online]. Available: https://training.fema.gov/hiedu/docs /fem/chapter%201%20-%20intro%20to%20em.doc. [Accessed 23 September 2017].

[14] Public Safety Canada, "Emergency Management Planning Guide 2010–2011," Government of Canada, Ottawa, 2010.

[15] Public Safety Canada, "An Emergency Management Framework for Canada," Emergency Management Policy Directorate, Ottawa, 2011.

[16] Environment Canada, "Emergency Management Basics," Government of Canada, 23 Jul 2013. [Online]. Available: https://www.ec.gc.ca/ouragans-hurricanes/default.asp?lang=En&n=31DADDF5-1#archived. [Accessed 11 May 2015].

[17] Federal Emergency Management Agency (FEMA), "Introduction to Emergency Management," 21 Nov 2006. [Online]. Available: https://training.fema.gov/hiedu/docs/fem/chapter%201%20-%20intro%20to%20em.doc. [Accessed 23 Feb 2015].

[18] C. White, Social Media, Crisis Communication, and Emergency Management: Leveraging Web ., Boca Raton, FL: CRC Press, 2011.

[19] N. Dimakis, A. Filippoupolitis and E. Gelenbe, "Distributed Building Evacuation Simulator for Smart Emergency Management," *The Computer Journal*, vol. 53, no. 9, pp. 1384-1400, 2010.

[20] H. Bi, A. Desmet and E. Gelenbe, "Routing Emergency Evacuees with Cognitive Packet Networks," in *Information Sciences and Systems 2013*, Switzerland, Springer International Publishing, 2013, pp. 295-303.

[21] V. Schmid, "Solving the dynamic ambulance relocation and dispatching problem using approximate dynamic programming," *European Journal of Operational Research*, vol. 219, no. 3, p. 611–621, 2012.

[22] A. T. Zagorecki, D. E. Johnson and J. Ristvej, "Data mining and machine learning in the context of disaster and crisis management," *International Journal of Emergency Management*, vol. 9, no. 4, pp. 351-365, 2013.

[23] Z. Liao, B. Wang, X. Xia and P. M. Hannam, "Environmental emergency decision support system based on Artificial Neural Network," *Safety Science*, vol. 50, no. 1, pp. 150-163, 2012.

[24] B. Hsieh and J. Ratcliff, "Coastal Hurricane Inundation Prediction for Emergency Response Using Artificial Neural Networks," in *14th International Conference, EANN 2013*, Halkidiki, Greece, September 2013.

[25] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning), Cambridge, Massachusetts: The MIT Press, 1998.

[26] Public Safety Canada, "National Strategy for Critical Infrastructure," 31 01 2018. [Online]. Available: https://www.publicsafety.gc.ca/cnt/rsrcs/pblctns/srtg-crtcl-nfrstrctr/index-en.aspx. [Accessed 03 05 2018].

[27] E. Wiseman, T. McLaughlin and N. Mrad, "Critical Infrastructure Protection and Resilience Literature Survey: State of the Art," DRDC Project #CSSP-2013-TI-1039, Ottawa, 2014.

[28] U.S. Department of Homeland Security, "Critical Infrastructure Sectors," 11 07 2017. [Online]. Available: https://www.dhs.gov/critical-infrastructure-sectors. [Accessed 27 05 2018].

[29] R. Setola and M. Theocharidou, "Modelling Dependencies Between Critical Infrastructures," in *Managing the Complexity of Critical Infrastructures, A Modelling and Simulation Approach*, Springer Open, 2016, pp. 19-41.

[30] S. M. Rinaldi, "Modeling and Simulating Critical Infrastructures and Their Interdependencies," in *37th Hawaii International Conference on System Sciences*, Hawaii, 2004.

[31] J. R. Martí, "Multisystem Simulation: Analysis of Critical Infrastructures for Disaster Response," in *The Last Frontier of Complexity. Understanding Complex Systems*, Switzerland, Springer, 2014, pp. 255-277 .

[32] M. Ouyang, "Review on modeling and simulation of interdependent critical infrastructure systems," *Reliability Engineering and System Safety,* vol. 121, no. 2014, pp. 43-60, 2013.

[33] A. De Nicola, G. Vicoli and M. L. Villani, "A Rule-based Approach for Modelling Behaviour in Crisis and Emergency Scenarios," in *I-ESA Conferences, vol 5*, London, 2012.

[34] C. Foglietta, S. Panzieri and F. Pascucci, "Algorithms and Tools for Risk/Impact Evaluation in Critical Infrastructures," in *Intelligent Monitoring,Control, and Security of Critical Infrastructure Systems*, Berlin, Springer, Berlin, Heidelberg, 2014, pp. 227-238.

[35] Z. Z. Aung and K. Watanabe, in *Third Annual IFIP WG 11.10 International Conference on Critical Infrastructure Protection*, Hanover, New Hampshire, USA, 2009.

[36] A. Tofani, G. D'Agostino and J. Martí, "Phenomenological Simulators of Critical Infrastructures," in *Managing the Complexity of Critical Infrastructures, A Modelling and Simulation Approach*, Springer Open, 2016, pp. 85-107.

[37] Argonne National Laboratory, "Analysis of Critical Infrastructure Dependencies and Interdependencies," U.S. Department of Energy (DOE), Chicago, 2015.

[38] T. Rauber and G. Rünger, Parallel Programming for Multicore and Cluster Systems, Berlin: Springer, 2010.

[39] D. Culler, J. P. Singh and A. Gupta, Parallel Computer Architecture A Hardware / Software Approach, California: Morgan Kaufmann, 1998.

[40] F. Willmore, "Introduction to Parallel Computing," 6 February 2012. [Online]. Available: https://portal.tacc.utexas.edu/c/document_library/get_file?uuid=e05d457a-0fbf-424b-87ce-c96fc0077099&groupId=13601. [Accessed 14 June 2015].

[41] J. DeNero, "Chapter 4: Distributed and Parallel Computing," Berkeley University, May 2015. [Online]. Available: http://wla.berkeley.edu/~cs61a/fa11/lectures/communication.html. [Accessed 31 May 2015].

[42] P. J. Brooke and R. F. Paige, Practical Distributed Processing, London: Springer, 2008.

[43] R. Trobec, M. Vajtersic and P. Zinterhof, Parallel Computing - Numerics, Applications, and Trends, London: Springer, 2009.

[44] J. T. Barron, D. S. Golland and N. J. Hay, "Parallelizing Reinforcement Learning".

[45] Q. Liu, X. Yang, L. Jing, J. Li and J. Li, "A parallel scheduling algorithm for reinforcement learning in large state space," *Frontiers of Computer Science,* vol. 6, no. 6, pp. 631-646, December 2012.

[46] J. Bell, Machine Learning Hands-On for Developers and Technical Professionals, Indianapolis, Indiana: WILEY, 2015.

[47] Apache Hadoop, "Apache Hadoop," 2018. [Online]. Available: http://hadoop.apache.org/. [Accessed 25 09 2018].

[48] Y. Li and D. Schuurmans , "MapReduce for Parallel Reinforcement Learning," in *Recent Advances in Reinforcement Learning*, Athens, Springer Berlin Heidelberg, 2012, pp. 309-320.

[49] Apache Spark, "Apache Spark," 2018. [Online]. Available: http://spark.apache.org/. [Accessed 25 09 2018].

[50] N. Lipka, "Towards Distributed Reinforcement Learning for Digital Marketing with Spark," in *Spark Summit 2013*, San Francisco, 2013.

[51] V. S. Agneeswaran and V. Nanduri, "Distributed Reinforcement Learning for Electricity Market Bidding with Spark," in *Spark Summit 2014*, San Francisco, 2014.

[52] M. T. Khouj, S. Sarkaria, C. Lopez and J. Marti, "Reinforcement Learning using Monte-Carlo Policy Estimation for Disaster Mitigation," in *8th IFIPWG11.10 International Conference, ICCIP 2014*, Arlington, VA, USA, 2014.

[53] J. R. Martí and JIIRP-UBC Team, "The I2Sim Simulator for Disaster Response Coordination in Interdependent Infrastructure Systems," Institute for Computing, Information, and Cognitive Systems (ICICS), Vancouver, 2006.

[54] Ministry of Transportation and Infrastructure, British Columbia Smart Infrastructure Monitoring System (BCSIMS), Victoria, BC: Government of British Columbia, 2015.

[55] C. Lopez, *Multi-energy systems simulator for hourly management and optimization of GHG emissions and fuel costs*, Vancouver: University of British Columbia, MASc Thesis, 2011.

[56] J. Stewart, "Functions and Models, Limits and Derivatives (Continuity)," in *Calculus 7E Early Trascendentals*, Cengage Learning, 2008.

[57] E. Even-Dar and M.-Y. Kao, "Reinforcement Learning," in *Encyclopedia of Algorithms*, Springer, 2008, pp. 771-774.

[58] M. Wiering and M. v. Otterlo, Reinforcement Learning - State-of-the-Art, Heidelberg: Springer, 2012.

[59] P. Kulkarni, Reinforcement and Systemic Machine Learning for Decision Making, New Jersey: IEEE Press, WILEY, 2012.

[60] M. L. Puterman, Markov Decision Processes - Discrete Stochastic Dynamic Programming, New Jersey: Jhon Wiley & Sons, Inc., 1995.

[61] T. Hester, TEXPLORE: Temporal Difference Reinforcement Learning for Robots and Time-Constrained Domains, Switzerland: Springer, 2013.

[62] S. Singh, T. Jaakkola, M. L. Littma and C. Szepesvári , "Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms," *Machine Learning*, vol. 38, no. 3, pp. 287-308, 2000.

[63] A. Y. Ng, "Shaping and Policy search in Reinforcement learning," University of California, Berkeley, California, 2003.

[64] Sumavi, "xCAT Administrators Guide - Introduction," 2010. [Online]. Available: http://sumavi.com/chapters/introduction. [Accessed 5 June 2015].

[65] SourceForge, "xCAT Wiki," 2015. [Online]. Available: http://sourceforge.net/p/xcat/wiki/Main_Page/. [Accessed 13 May 2015].

[66] SourceForge, "xCAT commands xdcp," [Online]. Available: http://xcat.sourceforge.net/man1/xdcp.1.html. [Accessed 06 June 2015].

[67] SourceForge, "xCAT commands psh," [Online]. Available: http://xcat.sourceforge.net/man1/psh.1.html. [Accessed 06 June 2015].

[68] H. Juárez García, *Multi-hazard risk assessment: an interdependency approach*, Vancouver, BC: PhD Dissertation, 2010.

[69] Canadian Association of Emergency Physicians, "The Canadian Triage and Acuity Scale: Education Manual," 2012. [Online]. Available: https://caep.ca/wp-content/uploads/2017/06/module_1_slides_v2.5_2012.pdf. [Accessed 31 July 2018].

[70] M. van Steen and A. S. Tanenbaum, "Types of Distributed Systems," in *Distributed Systems Version 3.01*, Maarten van Steen, 2017, pp. 27-29.

[71] BC Place, "The stadium," [Online]. Available: https://www.bcplace.com/the-stadium. [Accessed 6 09 2018].

[72] P. Giachini, J. M. Gonsoulin, K. W. Hart, P. G. Yeung, N. V. Revenko and K. G. Crowther, "Risk-Informed Assessment of Scott Stadium Evacuation through Agent-Based Simulation," in *Systems and Information Engineering Design Symposium*, Charlottesville, VA, 2010.

[73] Solar Power Authority, "How Much Energy Will Super Bowl LI Use? The Answer May Surprise You," 02 02 2017. [Online]. Available: https://www.solarpowerauthority.com/much-energy-will-super-bowl-li-use-answer-may-surprise/. [Accessed 03 09 2018].

[74] B. Mackin, "Bright New BC Place Is 'Power Smart'?," The Tyee, 19 10 2012. [Online]. Available: https://thetyee.ca/News/2012/10/19/BC-Place-Power-Smart/. [Accessed 03 09 2018].

[75] C. A. Marti, "Cultural Factors as Modifiers of an Egress Model for the Vancouver 2010 Olympics," The University of British Columbia, Vancouver, BC, 2009.

[76] Simwalk, "Actionalbe Crowd solutions," Simwalk, [Online]. Available: http://www.simwalk.com/projects/index.html. [Accessed 04 09 2018].

[77] S. Omondi, "The Largest Hockey Arenas in the World," 01 08 2017. [Online]. Available: https://www.worldatlas.com/articles/the-largest-hockey-arenas-in-the-world.html. [Accessed 04 09 2018].

[78] K. Grolinger, A. L'Heureux, M. A. Capretz and L. Seewald, "Energy Forecasting for Event Venues: Big Data and Prediction Accuracy," *Energy and Buildings*, no. 112, pp. 222-233, 2016.

[79] Vancouver Coastal Health, "Vancouver General Hospital (VGH)," 2017. [Online]. Available: http://www.vch.ca/Locations-Services/result?res_id=644. [Accessed 05 09 2018].

[80] Providence Health Care, "St. Paul's Hospital," 2017. [Online]. Available: http://www.providencehealthcare.org/hospitals-residences/st-paul%27s-hospital. [Accessed 05 09 2018].

[81] BC Emergency Health Services, "Ground Fleet Fact Sheet," 03 2018. [Online]. Available: http://www.bcehs.ca/about-site/Documents/factsheets/Fact%20Sheet%20GROUND%20FLEET.pdf. [Accessed 05 09 2018].

Appendices

Appendix A - MATLAB Block Driver RL Agent

```
function agent_level_2(block)
% Level-2 M file S-Function for times two demo.
%   Copyright 1990-2004 The MathWorks, Inc.
%   $Revision: 1.1.6.1 $
%   Programmed by: Cesar Lopez Castellanos clopez@ece.ubc.ca
%   Jan 15-30 2018 Model City of Vancouver REV. 03
% % dist variable created in PostPropag, InitCond and
% % Terminate, IT MUSTMATCH

  setup(block);

%endfunction
function setup(block)

  %% Register number of input and output ports
  block.NumInputPorts  = 24;
  block.NumOutputPorts = 14;

  %% Setup functional port properties to dynamically
  %% inherited.
  block.SetPreCompInpPortInfoToDynamic;
  block.SetPreCompOutPortInfoToDynamic;

 % Allow multidimensional signals
  block.AllowSignalsWithMoreThan2D = true;

 % Register parameters
  block.NumDialogPrms = 4;
  block.DialogPrmsTunable = {'Tunable', 'Tunable', 'Tunable', 'Tunable'};
  %Parameters are ={Nbr.of.states, Nbr.of.actions, Offset, Exploration.rate}
  %
 % Override input port properties
  for i=1:block.NumInputPorts
    block.InputPort(i).Dimensions  = 1;
    block.InputPort(i).DatatypeID  = 0;  % double
    block.InputPort(i).Complexity  = 'Real';
    block.InputPort(i).DirectFeedthrough = true;
  end

% % Override output port properties
  for i=1:block.NumOutputPorts
    block.OutputPort(i).Dimensions  = 1;
    block.OutputPort(i).DatatypeID  = 0; % double
    block.OutputPort(i).Complexity  = 'Real';
  end
```

```
%% Set block sample time to inherited
  block.SampleTimes = [-1 0];

  %% Run accelerator on TLC
  block.SetAccelRunOnTLC(false);

  %% Register methods
  block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
  block.RegBlockMethod('InitializeConditions',    @InitConditions);
  block.RegBlockMethod('SetInputPortSamplingMode',@SetInputPortSamplingMode);
  %block.RegBlockMethod('SetInputPortDimensions', @SetInpPortDims);
  block.RegBlockMethod('Terminate', @Terminate);
  block.RegBlockMethod('Outputs', @Output);
%endfunction

function DoPostPropSetup(block)
  %% Setup Dworks
  dist=35;%Nbr of Dworks preceding the LUT NEEDED in InitConditions
         %IF CHANGED HERE IT HAS TO BE CHANGED IN InitConditions
         % and in Terminate
  block.NumDworks = dist+block.DialogPrm(2).Data;

  block.Dwork(1).Name = 'disch_Hist_VGH';%T-1 # of discharged patients VGH
  block.Dwork(1).Dimensions      = 1;
  block.Dwork(1).DatatypeID      = 0;
  block.Dwork(1).Complexity      = 'Real';
  block.Dwork(1).UsedAsDiscState = true;

  block.Dwork(2).Name = 'disch_Hist_SPH';%T-1 # of discharged patients SPH
  block.Dwork(2).Dimensions      = 1;
  block.Dwork(2).DatatypeID      = 0;
  block.Dwork(2).Complexity      = 'Real';
  block.Dwork(2).UsedAsDiscState = true;

  block.Dwork(3).Name = 'PreviousSt';%Previous visited state to update value
                                    %Recall this structure is a vector
                                    %lenght 2 [state|action]
  block.Dwork(3).Dimensions      = 2;
  block.Dwork(3).DatatypeID      = 0;
  block.Dwork(3).Complexity      = 'Real';
  block.Dwork(3).UsedAsDiscState = true;

  block.Dwork(4).Name = 'learning';%Learning rate
  block.Dwork(4).Dimensions      = 1;
  block.Dwork(4).DatatypeID      = 0;
  block.Dwork(4).Complexity      = 'Real';
  block.Dwork(4).UsedAsDiscState = true;

  block.Dwork(5).Name = 'discount';%Discount factor
  block.Dwork(5).Dimensions      = 1;
  block.Dwork(5).DatatypeID      = 0;
  block.Dwork(5).Complexity      = 'Real';
  block.Dwork(5).UsedAsDiscState = true;
```

```
 %%Distributors' # of combinations
%%------------------------------------------------------------------------
 block.Dwork(6).Name = 'Dis1Comb';%Number of combinations for distr 1-DGR
 block.Dwork(6).Dimensions      = 1;
 block.Dwork(6).DatatypeID      = 0;
 block.Dwork(6).Complexity      = 'Real';
 block.Dwork(6).UsedAsDiscState = true;

 block.Dwork(7).Name = 'Dis2Comb';%Number of combinations for distr 2-SPG
 block.Dwork(7).Dimensions      = 1;
 block.Dwork(7).DatatypeID      = 0;
 block.Dwork(7).Complexity      = 'Real';
 block.Dwork(7).UsedAsDiscState = true;

 block.Dwork(8).Name = 'Dis3Comb';%Number of combinations for distr 3-MUR
 block.Dwork(8).Dimensions      = 1;
 block.Dwork(8).DatatypeID      = 0;
 block.Dwork(8).Complexity      = 'Real';
 block.Dwork(8).UsedAsDiscState = true;

 block.Dwork(9).Name = 'Dis4Comb';%Number of combinations for distr 4-WPS
 block.Dwork(9).Dimensions      = 1;
 block.Dwork(9).DatatypeID      = 0;
 block.Dwork(9).Complexity      = 'Real';
 block.Dwork(9).UsedAsDiscState = true;

%%------------------------------------------------------------------------
  %%Distributors' starting points, Dwork with their firt combination
%%------------------------------------------------------------------------
 block.Dwork(10).Name = 'Dis1Start';%1st comb. distr 1-DGR is in Dwork?
 block.Dwork(10).Dimensions      = 1;
 block.Dwork(10).DatatypeID      = 0;
 block.Dwork(10).Complexity      = 'Real';
 block.Dwork(10).UsedAsDiscState = true;

 block.Dwork(11).Name = 'Dis2Start';%1st comb. distr 2-SPG is in Dwork?
 block.Dwork(11).Dimensions      = 1;
 block.Dwork(11).DatatypeID      = 0;
 block.Dwork(11).Complexity      = 'Real';
 block.Dwork(11).UsedAsDiscState = true;

 block.Dwork(12).Name = 'Dis3Start';%1st comb. distr 3-MUR is in Dwork?
 block.Dwork(12).Dimensions      = 1;
 block.Dwork(12).DatatypeID      = 0;
 block.Dwork(12).Complexity      = 'Real';
 block.Dwork(12).UsedAsDiscState = true;

 block.Dwork(13).Name = 'Dis4Start';%1st comb. distr 4-WPS is in Dwork?
 block.Dwork(13).Dimensions      = 1;
 block.Dwork(13).DatatypeID      = 0;
 block.Dwork(13).Complexity      = 'Real';
 block.Dwork(13).UsedAsDiscState = true;
```

```
%%-------------------------------------------------------------------------
  %%Distributor combinations DISTR 1 - DGR
%%-------------------------------------------------------------------------
 block.Dwork(14).Name = 'Dis1Comb1';%1st comb. for distr 1-DGR
 block.Dwork(14).Dimensions      = 3;
 block.Dwork(14).DatatypeID      = 0;
 block.Dwork(14).Complexity      = 'Real';
 block.Dwork(14).UsedAsDiscState = true;

 block.Dwork(15).Name = 'Dis1Comb2';%2nd comb. for distr 1-DGR
 block.Dwork(15).Dimensions      = 3;
 block.Dwork(15).DatatypeID      = 0;
 block.Dwork(15).Complexity      = 'Real';
 block.Dwork(15).UsedAsDiscState = true;

 block.Dwork(16).Name = 'Dis1Comb3';%3rd comb. for distr 1-DGR
 block.Dwork(16).Dimensions      = 3;
 block.Dwork(16).DatatypeID      = 0;
 block.Dwork(16).Complexity      = 'Real';
 block.Dwork(16).UsedAsDiscState = true;
%%-------------------------------------------------------------------------
  %%Distributor combinations DISTR 2 - SPG
%%-------------------------------------------------------------------------
 block.Dwork(17).Name = 'Dis2Comb1';%1st comb. for distr 2-SPG
 block.Dwork(17).Dimensions      = 2;
 block.Dwork(17).DatatypeID      = 0;
 block.Dwork(17).Complexity      = 'Real';
 block.Dwork(17).UsedAsDiscState = true;

 block.Dwork(18).Name = 'Dis2Comb2';%2nd comb. for distr 2-SPG
 block.Dwork(18).Dimensions      = 2;
 block.Dwork(18).DatatypeID      = 0;
 block.Dwork(18).Complexity      = 'Real';
 block.Dwork(18).UsedAsDiscState = true;
%%-------------------------------------------------------------------------
  %%Distributor combinations DISTR 3 - MUR
%%-------------------------------------------------------------------------
 block.Dwork(19).Name = 'Dis3Comb1';%1st comb. for distr 3-MUR
 block.Dwork(19).Dimensions      = 4;
 block.Dwork(19).DatatypeID      = 0;
 block.Dwork(19).Complexity      = 'Real';
 block.Dwork(19).UsedAsDiscState = true;

 block.Dwork(20).Name = 'Dis3Comb2';%2nd comb. for distr 3-MUR
 block.Dwork(20).Dimensions      = 4;
 block.Dwork(20).DatatypeID      = 0;
 block.Dwork(20).Complexity      = 'Real';
 block.Dwork(20).UsedAsDiscState = true;

 block.Dwork(21).Name = 'Dis3Comb3';%3rd comb. for distr 3-MUR
 block.Dwork(21).Dimensions      = 4;
 block.Dwork(21).DatatypeID      = 0;
 block.Dwork(21).Complexity      = 'Real';
 block.Dwork(21).UsedAsDiscState = true;
```

```
block.Dwork(22).Name = 'Dis3Comb4';%4th comb. for distr 3-MUR
block.Dwork(22).Dimensions      = 4;
block.Dwork(22).DatatypeID      = 0;
block.Dwork(22).Complexity      = 'Real';
block.Dwork(22).UsedAsDiscState = true;

block.Dwork(23).Name = 'Dis3Comb5';%5th comb. for distr 3-MUR
block.Dwork(23).Dimensions      = 4;
block.Dwork(23).DatatypeID      = 0;
block.Dwork(23).Complexity      = 'Real';
block.Dwork(23).UsedAsDiscState = true;

block.Dwork(24).Name = 'Dis3Comb6';%6th comb. for distr 3-MUR
block.Dwork(24).Dimensions      = 4;
block.Dwork(24).DatatypeID      = 0;
block.Dwork(24).Complexity      = 'Real';
block.Dwork(24).UsedAsDiscState = true;

%%-------------------------------------------------------------------------
   %%Distributor combinations DISTR 4 - WPS
%%-------------------------------------------------------------------------
block.Dwork(25).Name = 'Dis4Comb1';%1st comb. for distr 4-WPS
block.Dwork(25).Dimensions      = 2;
block.Dwork(25).DatatypeID      = 0;
block.Dwork(25).Complexity      = 'Real';
block.Dwork(25).UsedAsDiscState = true;

block.Dwork(26).Name = 'Dis4Comb2';%2nd comb. for distr 4-WPS
block.Dwork(26).Dimensions      = 2;
block.Dwork(26).DatatypeID      = 0;
block.Dwork(26).Complexity      = 'Real';
block.Dwork(26).UsedAsDiscState = true;

block.Dwork(27).Name = 'Dis4Comb3';%3rd comb. for distr 4-WPS
block.Dwork(27).Dimensions      = 2;
block.Dwork(27).DatatypeID      = 0;
block.Dwork(27).Complexity      = 'Real';
block.Dwork(27).UsedAsDiscState = true;

block.Dwork(28).Name = 'Dis4Comb4';%4th comb. for distr 4-WPS
block.Dwork(28).Dimensions      = 2;
block.Dwork(28).DatatypeID      = 0;
block.Dwork(28).Complexity      = 'Real';
block.Dwork(28).UsedAsDiscState = true;

block.Dwork(29).Name = 'Dis4Comb5';%5th comb. for distr 4-WPS
block.Dwork(29).Dimensions      = 2;
block.Dwork(29).DatatypeID      = 0;
block.Dwork(29).Complexity      = 'Real';
block.Dwork(29).UsedAsDiscState = true;

block.Dwork(30).Name = 'Dis4Comb6';%6th comb. for distr 4-WPS
block.Dwork(30).Dimensions      = 2;
```

```
  block.Dwork(30).DatatypeID      = 0;
  block.Dwork(30).Complexity      = 'Real';
  block.Dwork(30).UsedAsDiscState = true;

  block.Dwork(31).Name = 'Dis4Comb7';%7th comb. for distr 4-WPS
  block.Dwork(31).Dimensions      = 2;
  block.Dwork(31).DatatypeID      = 0;
  block.Dwork(31).Complexity      = 'Real';
  block.Dwork(31).UsedAsDiscState = true;

  block.Dwork(32).Name = 'Dis4Comb8';%8th comb. for distr 4-WPS
  block.Dwork(32).Dimensions      = 2;
  block.Dwork(32).DatatypeID      = 0;
  block.Dwork(32).Complexity      = 'Real';
  block.Dwork(32).UsedAsDiscState = true;

  block.Dwork(33).Name = 'Dis4Comb9';%9th comb. for distr 4-WPS
  block.Dwork(33).Dimensions      = 2;
  block.Dwork(33).DatatypeID      = 0;
  block.Dwork(33).Complexity      = 'Real';
  block.Dwork(33).UsedAsDiscState = true;

  block.Dwork(34).Name = 'Dis4Comb10';%10th comb. for distr 4-WPS
  block.Dwork(34).Dimensions      = 2;
  block.Dwork(34).DatatypeID      = 0;
  block.Dwork(34).Complexity      = 'Real';
  block.Dwork(34).UsedAsDiscState = true;

  block.Dwork(35).Name = 'Dis4Comb11';%11th comb. for distr 4-WPS
  block.Dwork(35).Dimensions      = 2;
  block.Dwork(35).DatatypeID      = 0;
  block.Dwork(35).Complexity      = 'Real';
  block.Dwork(35).UsedAsDiscState = true;

 %%-----------------------------------------------------------------------
  %%Create Dworks for LUT
 %%-----------------------------------------------------------------------
  for gh=1:block.DialogPrm(2).Data
   if gh<10
     block.Dwork(gh+dist).Name = strcat('LUT_0',num2str(gh));%Look up table
   else
     block.Dwork(gh+dist).Name = strcat('LUT_',num2str(gh)); %Look up table
   end
   block.Dwork(gh+dist).Dimensions      = block.DialogPrm(1).Data;
   block.Dwork(gh+dist).DatatypeID      = 0;
   block.Dwork(gh+dist).Complexity      = 'Real';
   block.Dwork(gh+dist).UsedAsDiscState = true;
  end
%endfunction

function Terminate(block)
%% Saving LUT to file
dist=35; %This value must match "dist" in PostPropagationSetup
 %Improved save to text file LUT
```

```
  fid=fopen('LUT.txt','wt+');
   for act=1+dist:block.DialogPrm(2).Data+dist
     fprintf(fid,'%f\t', block.Dwork(act).Data);
     fprintf(fid,'\n');
   end
  fclose(fid);
 %end of improved routine
 disp('Done');
 toc;
%endfunction

function InitConditions(block)
%% Initilize Mapping
 dist=35; %This value must match "dist" in PostPropagationSetup

 %Initialize (LUT)
  mat=zeros(block.DialogPrm(1).Data,1);
  for act=1+dist:block.DialogPrm(2).Data+dist
    mat= block.Dwork(act).Data;
    for stat=1:block.DialogPrm(1).Data
      mat(stat) = rand(1,1)-0.5;
    end;
    block.Dwork(act).Data=mat;
  end
%end of initialize LUT

%%Load (LUT)
 fid=fopen('LUT.txt','r');
  for act=1+dist:block.DialogPrm(2).Data+dist
    block.Dwork(act).Data=fscanf(fid,'%f',[1,block.DialogPrm(1).Data]);
  end
 fclose(fid);
%end of load LUT

tic;

%Pass the information to the Dworks vectors to use them as global variables
  block.Dwork(1).Data=0; %Start VGH discharged patients with zero
  block.Dwork(2).Data=0; %Start SPH discharged patients with zero

    %%Previous state vector [state|action] below
  block.Dwork(3).Data(1)=0; % sense the state; %Initialize state n-1 with
                            % invalid state to be skipped Recall this
                            % structure is a vector lenght 2 [state|action]

  block.Dwork(3).Data(2)=0; %Initialize action n-1 with 0
    %%Previous state vector [state|action] above

  block.Dwork(4).Data=0.5;%Establishing learning rate
  block.Dwork(5).Data=0.7;%Establishing discount factor

  block.Dwork(6).Data=3; % # combts. Distr 1
  block.Dwork(7).Data=2; % # combts. Distr 2
```

```
  block.Dwork(8).Data=6; % # combts. Distr 3
  block.Dwork(9).Data=11;% # combts. Distr 4

  block.Dwork(10).Data=14; % Start point for Distr 1
  block.Dwork(11).Data=17; % Start point for Distr 2
  block.Dwork(12).Data=19; % Start point for Distr 3
  block.Dwork(13).Data=25; % Start point for Distr 4

  block.Dwork(14).Data=[0.014900, 0.052632, 0.932468]; % comb. 1 - Distr 1
  block.Dwork(15).Data=[0.022291, 0.078740, 0.898969]; % comb. 2 - Distr 1
  block.Dwork(16).Data=[0.044234, 0.156250, 0.799516]; % comb. 3 - Distr 1

  block.Dwork(17).Data=[0.091743, 0.908257]; % comb. 1 - Distr 2
  block.Dwork(18).Data=[0.183486, 0.816514]; % comb. 2 - Distr 2

  block.Dwork(19).Data=[0.079365, 0.010100, 0.000100, 0.910435]; % comb.1 Distr 3
  block.Dwork(20).Data=[0.119048, 0.015150, 0.000150, 0.865652]; % comb.2 Distr 3
  block.Dwork(21).Data=[0.307692, 0.039157, 0.000388, 0.652763]; % comb.3 Distr 3
  block.Dwork(22).Data=[0.119048, 0.039157, 0.000388, 0.841408]; % comb.4 Distr 3
  block.Dwork(23).Data=[0.079365, 0.000000, 0.000150, 0.920485]; % comb.5 Distr 3
  block.Dwork(24).Data=[0.079365, 0.000000, 0.000388, 0.920247]; % comb.6 Distr 3

  block.Dwork(25).Data=[1,   0  ]; % comb. 1 - Distr 4
  block.Dwork(26).Data=[0.9, 0.1]; % comb. 2 - Distr 4
  block.Dwork(27).Data=[0.8, 0.2]; % comb. 3 - Distr 4
  block.Dwork(28).Data=[0.7, 0.3]; % comb. 4 - Distr 4
  block.Dwork(29).Data=[0.6, 0.4]; % comb. 5 - Distr 4
  block.Dwork(30).Data=[0.5, 0.5]; % comb. 6 - Distr 4
  block.Dwork(31).Data=[0.4, 0.6]; % comb. 7 - Distr 4
  block.Dwork(32).Data=[0.3, 0.7]; % comb. 8 - Distr 4
  block.Dwork(33).Data=[0.2, 0.8]; % comb. 9 - Distr 4
  block.Dwork(34).Data=[0.1, 0.9]; % comb.10 - Distr 4
  block.Dwork(35).Data=[0,   1  ]; % comb.11 - Distr 4
%endfunction

function SetInputPortSamplingMode(block, idx, fd)
 block.InputPort(idx).SamplingMode = fd;
 for i=1:block.NumOutputPorts
   block.OutputPort(i).SamplingMode = fd;
 end
%endfunction

function Output(block)
 dist=35; %This value must match "dist" in PostPropagationSetup
%------------------------------------------------------------------------
 %Collect Inputs
%------------------------------------------------------------------------
  PM1     = block.InputPort(1).Data;
  RM1     = block.InputPort(2).Data;
  NbrPMs1 = block.InputPort(3).Data;
  [PROD1] = Product_index(PM1,RM1,NbrPMs1);
%------------------------------------------------------------------------
  PM2     = block.InputPort(4).Data;
  RM2     = block.InputPort(5).Data;
```

```
  NbrPMs2 = block.InputPort(6).Data;
  [PROD2] = Product_index(PM2,RM2,NbrPMs2);
%-------------------------------------------------------------------------
  PM3     = block.InputPort(7).Data;
  RM3     = block.InputPort(8).Data;
  NbrPMs3 = block.InputPort(9).Data;
  [PROD3] = Product_index(PM3,RM3,NbrPMs3);
%-------------------------------------------------------------------------
  PM4     = block.InputPort(10).Data;
  RM4     = block.InputPort(11).Data;
  NbrPMs4 = block.InputPort(12).Data;
  [PROD4] = Product_index(PM4,RM4,NbrPMs4);
%-------------------------------------------------------------------------
  PM5     = block.InputPort(13).Data;
  RM5     = block.InputPort(14).Data;
  NbrPMs5 = block.InputPort(15).Data;
  [PROD5] = Product_index(PM5,RM5,NbrPMs5);
%-------------------------------------------------------------------------
  PM6     = block.InputPort(16).Data;
  RM6     = block.InputPort(17).Data;
  NbrPMs6 = block.InputPort(18).Data;
  [PROD6] = Product_index(PM6,RM6,NbrPMs6);
%-------------------------------------------------------------------------
  PROD7   = block.InputPort(19).Data; %Values are 1(off) or 2(on)
    if PROD7 > 0
      PROD7=2;
    else
      PROD7=1;
    end
  PROD8   = block.InputPort(20).Data;
    if PROD8 > 0
      PROD8=2;
    else
      PROD8=1;
    end
%-------------------------------------------------------------------------
 %Get the counter ready
  counter=get_param(bdroot,'SimulationTime')+1;
%-------------------------------------------------------------------------
%sense the state
  %The order of the arguments matches precalculations on Excel
  %DGR(10)  SPG(6)  MUR(10) BKPVGH(2) BKPSPH(2) WPS(15) VGH(15) SPH(15)
  % PROD1   PROD2    PROD3   PROD7     PROD8     PROD4   PROD5   PROD6

  if counter<=1
     state=1;%+block.DialogPrm(3).Data;
  else
    [state]=Index_state(PROD1,PROD2,PROD3,PROD7,PROD8,PROD4,PROD5,PROD6)-
block.DialogPrm(3).Data;
  end
%-------------------------------------------------------------------------
  PrvSt=block.Dwork(3).Data; %Retrieving the previous visited state
%
  CurreSt=[state,0]; %Retrieving the current chosen state
%                     %the action is not needed because it uses PrvSt's
```

```
%                     %action in the Value function
%
  learning=block.Dwork(4).Data;%retrieving learning rate
  discount=block.Dwork(5).Data;%retrieving discount factor
%
% %%Reward Generation
%  VGH Reward
 if block.InputPort(21).Data == block.Dwork(1).Data %Values are equal
   if block.InputPort(22).Data>0 && block.InputPort(21).Data==0 %Patients waiting
but no discharged
      reward1 = -5;
   else
     reward1 = block.InputPort(21).Data;
   end
 else
      reward1 = (block.InputPort(21).Data - block.Dwork(1).Data)*5;
 end

%  SPH Reward
 if block.InputPort(23).Data == block.Dwork(2).Data %Values are equal
   if block.InputPort(24).Data>0 && block.InputPort(23).Data==0 %Patients waiting
but no discharged
      reward2 = -4;
   else
     reward2 = block.InputPort(23).Data;
   end
 else
     reward2 = (block.InputPort(23).Data - block.Dwork(2).Data)*4;
 end

reward = reward1 + reward2;
% %%End of Reward Generation
%
 if (PrvSt(1)>=1 && CurreSt(1)>=1) %just compute q-fuction if both n-1 and n-2
states are valid and waiting area is not empty
  if  (block.InputPort(22).Data>0 && block.InputPort(24).Data>0)

     block.Dwork(PrvSt(2)+dist).Data(PrvSt(1))=
block.Dwork(PrvSt(2)+dist).Data(PrvSt(1))+learning*(reward+discount*block.Dwork(Pr
vSt(2)+dist).Data(CurreSt(1))-block.Dwork(PrvSt(2)+dist).Data(PrvSt(1)));
  end
 end

% Calculate bounds for potential states based in static ProdMs
 %DGR(10)  SPG(6)   MUR(10) BKPVGH(2) BKPSPH(2) WPS(15) VGH(15) SPH(15)
 % PROD1   PROD2    PROD3   PROD7     PROD8     PROD4   PROD5   PROD6
 %if state is valid %[to cover the whole possible states ----> (for all PROD?'s]
 if(PROD1>=1 && PROD1<=10 && PROD2>=1 && PROD2<=6 && PROD3>=1 && PROD3<=10 &&
PROD4>=1 && PROD4<=15 && PROD5>=1 && PROD5<=15 && PROD6>=1 && PROD6<=15 &&
PROD7>=1 && PROD7<=2 && PROD8>=1 && PROD8<=2)
  Maxm=-1E400;
  Distr1index=1; %DGR
  Distr2index=1; %SPG
  Distr3index=1; %MUR
```

```
  Distr4index=1; %WAT

  %explrate=block.DialogPrm(4).Data
  if  rem(counter,block.DialogPrm(4).Data)==0  %going for explore movement every x
timesteps
     %Generate 100 values from the uniform distribution on the interval [a, b].
     %r = a + (b-a).*rand(100,1);
     Distr1index= round(1 + ( 3-1).*rand(1,1));  %random value between 1 &  3
     Distr2index= round(1 + ( 2-1).*rand(1,1));  %random value between 1 &  2
     Distr3index= round(1 + ( 6-1).*rand(1,1));  %random value between 1 &  6
     Distr4index= round(1 + (11-1).*rand(1,1));  %random value between 1 & 11
  else %Greedy movement %look for the highest Q-Value
      for hh=1:block.Dwork(6).Data
        for ii=1:block.Dwork(7).Data
           for jj=1:block.Dwork(8).Data
             for kk=1:block.Dwork(9).Data
                [action_ind] = Index_Action(hh,ii,jj,kk);
                if block.Dwork(dist+action_ind).Data(state) > Maxm   %Keep in mind
the offset with the rest of the Dworks is in dist
                   Maxm=block.Dwork(dist+action_ind).Data(state);
                   Distr1index=hh;
                   Distr2index=ii;
                   Distr3index=jj;
                   Distr4index=kk;
                end
              end
           end
         end
      end
  end
 else
  Distr1index=1; %To force the agent to choose 1st state if PMs are outta bounds
  Distr2index=1; %To force the agent to choose 1st state if PMs are outta bounds
  Distr3index=1; %To force the agent to choose 1st state if PMs are outta bounds
  Distr4index=1; %To force the agent to choose 1st state if PMs are outta bounds
  state=1; %To force the agent to choose 1st state if PMs are outta bounds
 end;
%
  [action_ind] = Index_Action(Distr1index,Distr2index,Distr3index,Distr4index);
%
%  block.OutputPort(4).Data =PrvSt;   %PrvSt;
%  block.OutputPort(5).Data =CurreSt;
%
 CurreSt(1)=state;       %updating current  state  for next time step
 CurreSt(2)=action_ind;  %updating current  action for next time step
 PrvSt=CurreSt;          %updating previous state  for next time step
%
%  block.OutputPort(3).Data=reward;

 block.Dwork(1).Data = block.InputPort(21).Data; %storing VGH discharged for
hystorical values in order to generate Rewards
 block.Dwork(2).Data = block.InputPort(23).Data; %storing SPH discharged for
hystorical values in order to generate Rewards
```

```
%%SET THE OUTPUTS VIA SETPARAM
 %Distributor 1
  blockname=strcat(bdroot,'/','DGR');
  [factors]=ParseRatios(block.Dwork(block.Dwork(10).Data-1+Distr1index).Data);
  set_param(blockname,'factors',factors);

 %Distributor 2
  blockname=strcat(bdroot,'/','SPG');
  [factors]=ParseRatios(block.Dwork(block.Dwork(11).Data-1+Distr2index).Data);
  set_param(blockname,'factors',factors);

 %Distributor 3
  blockname=strcat(bdroot,'/','MUR');
  [factors]=ParseRatios(block.Dwork(block.Dwork(12).Data-1+Distr3index).Data);
  set_param(blockname,'factors',factors);

 %Distributor 4
  blockname=strcat(bdroot,'/','WATER');
  [factors]=ParseRatios(block.Dwork(block.Dwork(13).Data-1+Distr4index).Data);
  set_param(blockname,'factors',factors);

  block.Dwork(3).Data=PrvSt; %Storing the previous visited state
 %endfunction

function [factors]=ParseRatios(vector)
  [length,col]=size(vector);
  cad='[';
  for ind=1:length-1
    cad=strcat(cad,num2str(vector(ind)),';');
  end
  factors=strcat(cad,num2str(vector(length)),']');
%endfunction

function [ind] = Product_index(PM,RM,NbrPM)
 ind=(PM-1)*NbrPM+RM;
 switch PM
    case 3
      ind=ind-1;
    case 4
      ind=ind-3;
    case 5
      ind=ind-6;
 end
%End function productivity index

 %DGR(10)  SPG(6)   MUR(10) BKPVGH(2) BKPSPH(2) WPS(15) VGH(15) SPH(15)
 % PROD1   PROD2    PROD3   PROD4     PROD5     PROD6   PROD7   PROD8
function [state_ind] = Index_state(Pro1,Pro2,Pro3,Pro4,Pro5,Pro6,Pro7,Pro8)
   state_ind=(Pro1-1)*(6*10*4*15^3)+(Pro2-1)*(10*4*15^3)+(Pro3-1)*(4*15^3)+(Pro4-
1)*(2*15^3)+(Pro5-1)*15^3+(Pro6-1)*15^2+(Pro7-1)*15+Pro8;
%End function state indexing
```

```
function [action_ind] = Index_Action(Act1,Act2,Act3,Act4)
 action_ind=(Act1-1)*(2*6*11)+(Act2-1)*(6*11)+(Act3-1)*11+Act4; %Observe the three
as it is the nbr of options of lowest rank combinatorial distr.
%End function action indexing

%Programmed by: Cesar Lopez Castellanos clopez@ece.ubc.ca
```

Appendix B - Basic Scheduler, MATLAB Side

```
function result = i2SimInterface(modelPath, modelName,...
       paramSimulationTime, paramTimeStep, paramTimeUnits)
   % add inputFileName,
   % the interface will output some results
   setDirectory(modelPath);
   openModel(modelName);
   concatNexec({'set_param(bdroot,''StopTime'', '''; ...
                num2str(paramSimulationTime); ...
               ''')''});
  %Set solver settings
     set_param(bdroot,'Solver','FixedStepDiscrete');
     set_param(bdroot,'SolverType','Fixed-step');
     set_param(bdroot,'StartTime','0');
  %End of Set solver settings
  %Load Partitions table
     load Partitions;
     name = evalc('system(''hostname'')');
     set_partition_settings(modelName,str2num(name(5:6)),Partitions);
  %End of Load Partitions Table
  %openControlPanel(modelName, controlPanelName);
  %startModel(modelName);
    load ExpVector;
    for lp=1:6
        %disp('Iteration: '); disp(lp);
        concatNexec({'set_param(''';modelName;'/Agent'', ...
                     ''ExplRate'', ''';num2str(ExpVector(lp));''')'});
        concatNexec({'sim('''; modelName; ''')'});
    end
   result = 1;
  function setDirectory(directory)
   %Sets the directory for a given simulation, be careful to add to the
   %set path all the variables and readables that are going to be used
      concatNexec({'cd ''' ; directory; ''''});
  %end function

   function openModel(modelName)
       concatNexec({'load_system '; modelName; '.mdl'});
   %end function

   function set_partition_settings(modelName,row,Partitions)
       vector=Partitions(row,:);
       WaterStationName='Water_Station';
       ElectStationName='Electrical Substation';
       FeederName='2L31';
       load ElecRM;
       fila=vector(1)-1+vector(2);
       Valor=ElecRM(fila);
       concatNexec({'set_param(''';modelName;'/';FeederName;''', ...
                    ''Output'', ''';num2str(Valor);''')'});
       concatNexec({'set_param(''';modelName;'/';WaterStationName;''', ...
                    ''PM'', ''';num2str(vector(3));''')'});
```

```
    concatNexec({'set_param(''';modelName;'/';ElectStationName;''', ...
                 ''PM'', ''';num2str(vector(1));''')'});
%end function set_partition_settings

function openControlPanel(modelName, controlPanelName)
   concatNexec({'open_system (''; modelName; '/' ; controlPanelName; ...
                ''', ''OpenFcn'')'});
%end function
function startModel(modelName)
   concatNexec({'set_param (''' ; modelName ; ''', ...
                ''SimulationCommand'', ''start'')'});
%end function
function pauseModel(modelName)
   concatNexec({'set_param (''' ; modelName ; ''', ...
                ''SimulationCommand'', ''pause'')'});
%end function
function continueModel(modelName)
   concatNexec({'set_param (''' ; modelName ; ''', ...
                ''SimulationCommand'', ''continue'')'});
%end function
function setInputFile(modelName, bloque, ruta)
   concatNexec({'set_param (''; modelName; '/' ; bloque; ...
                ''', ''route'',''' ; ruta; ''')'});
%end function
function setOutputFile(modelName, bloque, ruta)
   concatNexec({'set_param (''; modelName; '/' ; bloque; ...
                ''', ''route'',''' ; ruta; ''')'});
%end function
function setDeltaT(modelName, bloque, value)
   concatNexec({'set_param (''; modelName; '/' ; bloque; ...
                ''', ''DeltaT'',''' ; num2str(value); ''')'});
%end function
function dispNEval(cmd)
%displays and executes a cmd
   disp(cmd);
   eval(cmd);
%end function execute(cmd)

function concatNexec(celda)
    cmd='';
    [cols rows]=size(celda);
    for i=1:cols
        aux = char(celda(i)); %going from cell to string
        cmd = [cmd,aux];
    end
    dispNEval(cmd);
%end function concatNexec(cell)

function setTimeUnits(modelName, controlPanelName, TimeUnits)
   concatNexec({'set_param(''; modelName; '/'; controlPanelName; ...
                ''', ''TimeUnit'', '''; num2str(TimeUnits); ''')'});
%end function
```