Design and Implementation of an Agent-Based Model of Pertussis with Performance Considerations

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By

G. Wade McDonald

Permission to Use

Abstract

Pertussis, also known as Whooping Cough, is an airborne communicable disease caused by the *Bordetella pertussis* bacterium. Symptoms include fever, runny nose, and a cough that typically progresses to the point where it interferes with breathing, producing the characteristic whoop from which the common name is derived. Complications, which disproportionately affect infants, include bacterial pneumonia which can lead to death. Pertussis is vaccine-preventable and vaccination programs exist in most countries yet a recent resurgence has been observed in jurisdictions with high vaccine coverage, including Alberta and Canada.

Simulation modeling has a long history in the study of epidemiology, including that of pertussis, but most of such work has employed compartmental models. Agent-based models (ABMs) allow differentiation down to the individual level, which cannot be done in aggregate compartmental models, as well as simpler specification of heterogeneity and interaction patterns which can be tedious to implement in aggregate compartmental models. These benefits come at the cost of increased computational burden.

This thesis seeks to design and implement an ABM representing the epidemiology of pertussis in Alberta, Canada, and apply that model to evaluate vaccination during pregnancy as a potential intervention strategy to reduce pertussis incidence in infants. In support of this objective, data structures will be explored to improve performance for large ABMs developed using AnyLogic software.

Acknowledgements

There are a number of people I would like to thank for their assistance and support in my studies culminating in this thesis.

Firstly, I would like to thank Dr. Nathaniel D. Osgood, my supervisor, who I have enjoyed working with since 2016 when I began as an Undergraduate Student Research Assistant in his Computational Epidemiology and Public Health Informatics Lab (CEPHIL). Dr. Osgood boasts a wealth of knowledge and experience which he is happy to share with his students. Dr. Osgood has provided me with invaluable guidance both academically and in my interactions with colleagues and stakeholders of various backgrounds. Through his "Bootcamps" and "Simulathons," Dr. Osgood has exposed me and his other students to a wide variety of projects and collaborators beyond the classroom and lab. I am greatly appreciative of Dr. Osgood taking me on as a student and look forward to continuing to work with him in the future.

I would further like to thank Christine Hillis for keeping Dr. Osgood's lab organized and running efficiently and patiently guiding me through policies and procedures. I thank Drs. Ian McQuillan, Eric Neufeld, and Volker Gerdts for agreeing to be on my committee and investing time and effort in evaluating my work. Drs. Alexander Doroshenko and Karsten Hempel are thanked for their leadership, work, and domain expertise applied to the pertussis model. Dr. Mark Keil was invaluable for his advice on computational geometry and associated algorithms. I thank Reid McDonald, my brother, for help with R, GIS, editing, and personal support. Finally, I would like to thank my parents, Alfred and Charleen McDonald, for personal and logistical support in my return to post-secondary studies.

Contents

List of Tables

List of Figures

List of Abbreviations

ABM	Agent-Based Model
aP	Acellular Pertussis Vaccine
API	Application Programming Interface
CDC	Centers for Disease Control and Prevention
CDF	Cumulative Distribution Function
CEPHIL	Computational Epidemiology and Public Health Informatics Lab
CGAL	Computational Geometry Algorithms Library
CIRN	Canadian Immunization Research Network
CSV	Comma-separated Value
DES	Discrete Event Simulation
DTP	Diptheria, Tetanus, and Whole-Cell Pertussis Vaccine
DTaP	Diptheria, Tetanus, and Acellular Pertussis Vaccine
DTwP	Diptheria, Tetanus, and Whole-Cell Pertussis Vaccine
GIS	Geographic Information Systems
GUI	Graphical User Interface
IHDA	Alberta Interactive Health Data Application
JAR	Java Archive
JDK	Java Developer Kit
JNI	Java Native Interface
JVM	Java Virtual Machine
LHS	Latin Hypercube Sampling
MCMC	Markov Chain Monte Carlo
NNV	Number Needed to Vaccinate
ODE	Ordinary Differential Equation
OR	Odds Ratio
ORI	Outbreak Response Immunization
PFP	Preventable Fraction in the Population
PI	Principal Investigator
RAM	Random Access Memory
SD	System Dynamics
SEIR	Susceptible-Exposed-Infective-Recovered
SEIRS	Susceptible-Exposed-Infective-Recovered-Susceptible
SIR	Susceptible-Infective-Recovered
SIRS	Susceptible-Infective-Recovered-Susceptible
Tdap	Tetanus Diptheria, and Acellular Pertussis Vaccine
VC	Vaccine Coverage
VE	Vaccine Effectiveness
WHO	World Health Organization
wP	Whole-cell Pertussis Vaccine

1 Introduction

1.1 Motivation

Infectious disease is among the leading causes of death worldwide. Pertussis, in particular, affects 20 to 40 million people globally, leading to about 400,000 deaths annually. Approximately 1,000 to 3,000 people per year are known to be infected in Canada. Children under 1 year are at the greatest risk, especially if not current in their vaccinations [1].

1.2 Pertussis

Pertussis, also known as Whooping Cough, is an airborne communicable disease caused by the *Bordetella pertussis* bacterium. The disease is spread through droplets produced by coughing or sneezing. Symptoms, which generally appear seven to ten days after infection, include fever, runny nose, and a cough that typically progresses to the point where it interferes with breathing, producing the characteristic whoop from which the common name is derived. Untreated patients may be contagious for three weeks or more following the onset of symptoms [2].

Pertussis is vaccine-preventable and the World Health Organization (WHO) estimates that in 2008 global vaccination against pertussis prevented approximately 687,000 deaths [2]. Two types of vaccines exist: whole-cell and acellular. Since its development, some countries have moved towards the use of the acellular vaccine due to concerns about adverse reactions to the whole-cell vaccine, while others have continued to use the whole-cell vaccine due to its lower cost [3].

In Alberta, pertussis vaccines are given at ages 2, 4, 6, and 18 months, $4-6$ years, and $14-16$ years. All vaccines have been acellular since 1997, prior to which the whole-cell vaccine was used [4].

A resurgence of pertussis has been observed in recent years in some jurisdictions with high vaccine coverage, including Alberta and Canada [3, 4, 5].

1.3 Agent-Based Modeling

Agent-based modeling is a dynamic simulation modeling technique in which "a system is modeled as a collection of autonomous decision-making entities called *agents*" [6]. Agents have the properties of being autonomous and unique, and interact with their environment and other agents locally. Agents may change over time, have memory of previous states, and progress through life cycles [7].

Agent-based models (ABMs) allow us to examine emergent behaviour of a system. That is, “system dynamics that arise from how the system’s individual components interact with and respond to each other and their environment” [7].

1.4 Goal

This thesis seeks to design and implement an ABM representing the epidemiology of pertussis in Alberta, Canada, and apply that model to evaluate vaccination during pregnancy as a potential intervention strategy to reduce pertussis incidence. The model is required to be tractable for a nominal population of $500,000$, and represent vaccination and disease history of individual people as well as transfer of maternal immunity and waning of both vaccine- and disease-induced immunity. Heterogeneity in contact patterns is to be captured as well as clustering of populations with low vaccine coverage. As a means to this end, data structures, algorithms, and design patterns will be explored to improve performance for large ABMs developed using AnyLogic software [8].

2 Background

2.1 Pertussis

Pertussis, also known as Whooping Cough, is a communicable disease, spread through respiratory droplets, caused by the *Bordetella pertussis* bacterium.

2.1.1 Disease Progression

Pertussis is mainly transmitted through airborne respiratory droplets, produced by coughing or sneezing. It may also be transmitted through contact with contaminated clothing or other items [2, 9].

Pertussis is a toxin-mediated disease where bacteria attach to respiratory epithelial cells. Toxins produced by the bacteria paralyze the cilia of these cells and cause inflammation of the respiratory tract, thereby interfering with the clearing of pulmonary secretions [9].

The incubation period of the disease is generally $7 - 10$ days with a range of $4 - 21$ days. The onset—or first stage of infection—is described by the Centers for Disease Control and Prevention (CDC) as "insidious," in that the symptoms of runny nose, sneezing, low-grade fever, and mild occasional cough are similar to those of the common cold. This stage may last for $1 - 2$ weeks [9].

During the second, or *paroxysmal* stage, the infected person has paroxysms—or bursts—of rapid coughs in an effort to expel thick mucus. Following the paroxysm of coughs, "a long inspiratory effort is usually accompanied by a characteristic high-pitched whoop" [9]. This second stage lasts an average of $1 - 6$ weeks but may last up to 10 weeks. During this stage, the frequency of paroxysmal attacks average 15 per day. The first two weeks of this stage show an increase in paroxysmal attack frequency, followed by steady-state frequency for $2 - 3$ weeks, and ultimately a gradual decrease [9].

The third and final stage of the disease, called *convalescent*, is characterized by a gradual recovery in which the paroxysmal nature of the cough lessens and disappears within $2 - 3$ weeks [9]. Untreated patients are most contagious for three weeks following the onset of symptoms [2, 9].

Infants less than six months of age are at greatest risk of developing complications with pertussis, the typical one being bacterial pneumonia. CDC data indicate that 5.2% of the general population and 11.8% of infants under 6 months contract pneumonia as a complication. Most pertussis-related deaths are due to pneumonia [9].

2.1.2 Pre-Vaccination Era

Prior to introduction of vaccination against pertussis, the disease was a frequent cause of sickness and death, especially among infants and children [9]. In Canada, pertussis incidence averaged 156 cases per $100,000$ population in the five years prior to introduction of vaccination in 1943 [10]. Solid mortality figures for the pre-vaccination era are difficult to find. The CDC estimates that, prior to vaccination, the United States annually saw around $200,000$ childhood pertussis cases, resulting in about $9,000$ deaths [11]. In Canada, from 1934–1943, 35 deaths were reported per 1,000 cases notified [12].

2.1.3 Vaccines

Whole-cell pertussis vaccine (wP) contains complete *Bordetella pertussis* cells that have been *inactivated* to prevent them from reproducing and causing the full-blown disease. It is typically distributed as a combined vaccine which also inoculates against diptheria and tetanus, commonly abbreviated as DTP or DTwP. Adverse reactions to wP were common, and included redness, swelling, pain at the injection site, and fever [9]. Use of wP began in 1943 in Canada [10].

Acellular pertussis vaccines (aP) contain particular components of the *Bordetella pertussis* cell which have been purified and inactivated. Two commercial formulations exist that contain different components, or *antigens*. These are further combined with diptheria and tetanus vaccine formulations to produce a pediatric formulation, abbreviated DTaP, and an adolescent and adult formulation, abbreviated Tdap. Tdap differs from DTaP in having lower concentrations of diptheria and pertussis antigens in comparison with DTaP [9]. Use of aP vaccines began in Canada in 1997 [10].

The WHO "estimates that in 2008 global vaccination against pertussis prevented approximately 687,000 deaths" [2]. Since its development, some countries have moved towards aP use due to concerns about adverse reactions to wP, while others have continued to use wP due to its lower cost [3].

2.1.4 Epidemiological Trends

Following introduction of pertussis vaccination in 1943, overall Canadian incidence trended downwards to a mean annual minimum of 7 cases per $100,000$ population between 1984 and 1988. A resurgence occurred in the 1990s, peaking at 34.9 cases per $100,000$ in 1994, followed by a decline from 1998 reaching its nadir in 2011 with an incidence of 2 cases per $100,000$, the lowest ever recorded. 2012 featured another significant jump in incidence [10].

Smith et al. [10] further observe that local maxima in incidence consistently occur every $2-5$ years, in spite of being less visible in periods of low absolute incidence. Grenfell and Anderson [13] remark that the "inter-epidemic" period for pertussis was not altered by the introduction of vaccination in the UK context.

Domenech de Cellès et al. [14] point out that, worldwide, there is no consistent resurgence of pertussis. Of 63 countries examined in their analysis, they found that "pertussis incidence had increased in 16 countries but decreased in 32 countries, with no significant trend in 15 countries" [14]. Even among countries where the general trend was one of increase since 1980, most had one or more intervening periods of decrease. They report that "only four [countries]—Australia, Israel, the Netherlands and the USA—experienced no decrease in pertussis incidence over 1980–2012" [14].

2.1.5 Pertussis in Alberta

Pertussis is notifiable by law in Alberta. In order to be considered a case by Alberta Health, a suspected case of pertussis must be either confirmed by a laboratory test or have an epidemiological link to a laboratory-confirmed case along with demonstration of compatible symptoms [4].

In Alberta, pertussis vaccines are given at ages 2, 4, 6, and 18 months, $4-6$ years, and $14-16$ years. All vaccines have been aP since 1997, prior to which wP was used [4].

Liu et al. [4] wrote in 2017 that "There has been a resurgence of pertussis cases in Alberta in the last 15 years among all age groups, despite high vaccination coverage." Figure 2.1 shows Alberta-wide pertussis incidence since 2000. While rates are high overall, the general trend is downward towards a minimum of less than 2 cases per $100,000$ in 2010, followed by an upward trend, peaking in 2017. This aligns with the Canada-wide trends reported by Smith et al. [10], see Section 2.1.4. Vaccine coverage in Alberta has been consistently high since 2007, as seen in Figure 2.2.

2.2 Modeling

2.2.1 Overview of Simulation Modeling

Simulation Modeling

Use of models is pervasive in science, but what exactly is meant by the term "model" can vary between fields and disciplines. Some examples include "the billiard ball model of a gas, the Bohr model of the atom, the Lotka-Volterra model of predator-prey interaction, the Mundell-Fleming model of an open economy, and the scale model of a bridge" [16]. Many people primarily think of models as describing physical objects, however, a model could also describe "a social theory, an element of human behavior, or a relationship among statistical variables" [17], among other concepts.

Railsback and Grimm [7] cite Starfield [18] in describing a model as "a purposeful representation of some real system." Law and Kelton [19] suggest that a model constitutes the collection of assumptions made about how a system works in order to understand how it behaves. Simple models can often yield an exact *analytical* solution using the tools of mathematics and statistics. More complex models must be evaluated *numerically* using a computer in a process called *simulation*. Simulation involves calculating model outputs, numerically, for a given set of inputs and then evaluating the outcomes [19]. Dynamic simulation modeling, specifically, involves simulations that are calculated over time.

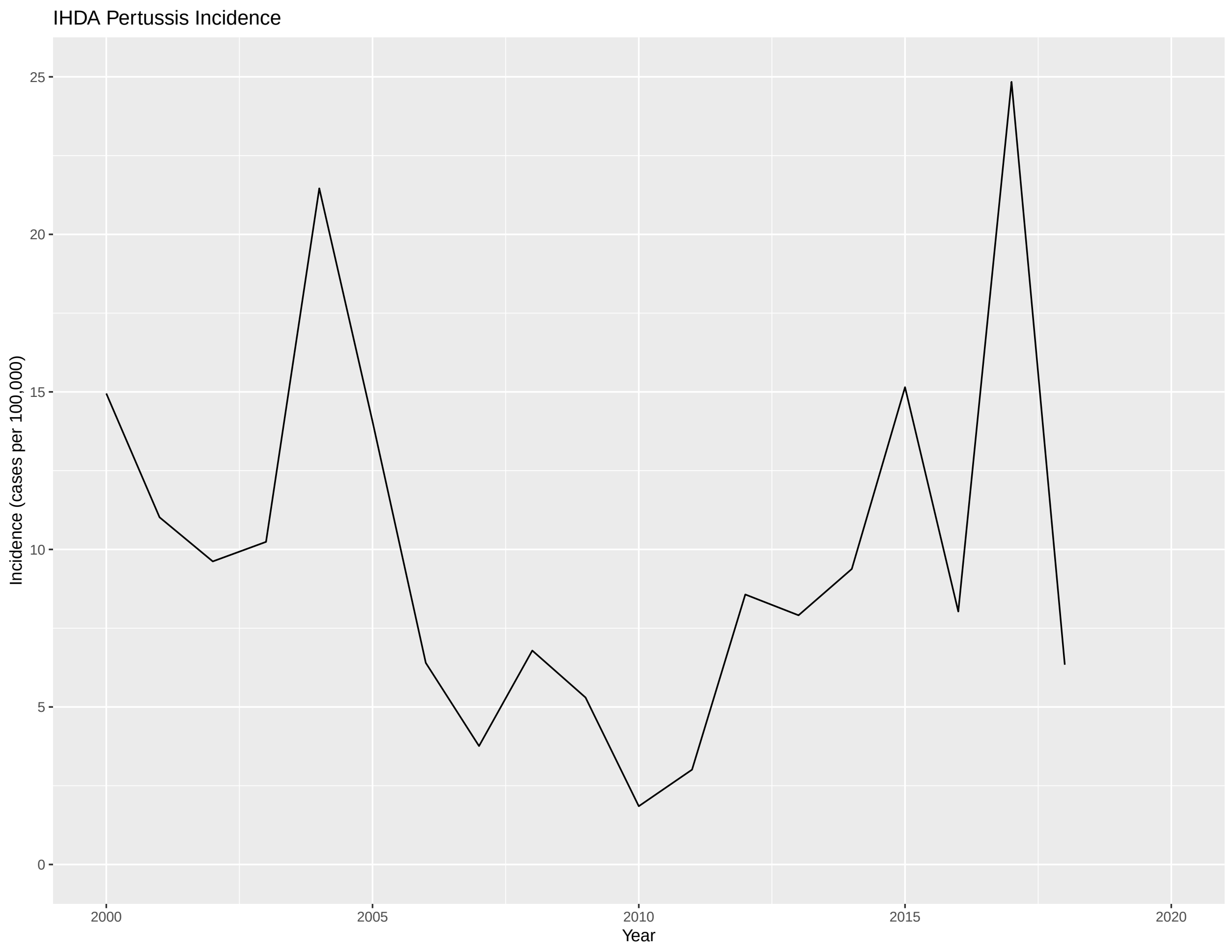


Figure 2.1: Pertussis Incidence in Alberta 2000–2018 [15]

Dynamic systems simulation modeling represented in the public health and health care space can be largely divided into three methodologies: System Dynamics (SD), Discrete Event Simulation (DES), and Agent-Based Modeling. The present work focuses exclusively on Agent-Based Modeling.

Compartmental Models

The term *compartmental model* generally refers to an Ordinary Differential Equation (ODE) or System Dynamics (SD) model. These models divide quantities into *compartments* and define rates of change of the size of those compartments using differential equations. Historically, most epidemiological models have been aggregate ODE models, going back as far as work in 1911 by Ross [20] and in 1927 by Kermack and McKendrick [21].

The System Dynamics branch of compartmental modeling techniques were developed by Jay W. Forrester at the Massachusetts Institute of Technology in the 1950s [22] and abstract ordinary differential equations into diagrams consisting of *stocks*, which represent a quantity, and *flows*, which represent change of a quantity. While SD and ODE techniques are essentially equivalent mathematically, SD has the advantages of being

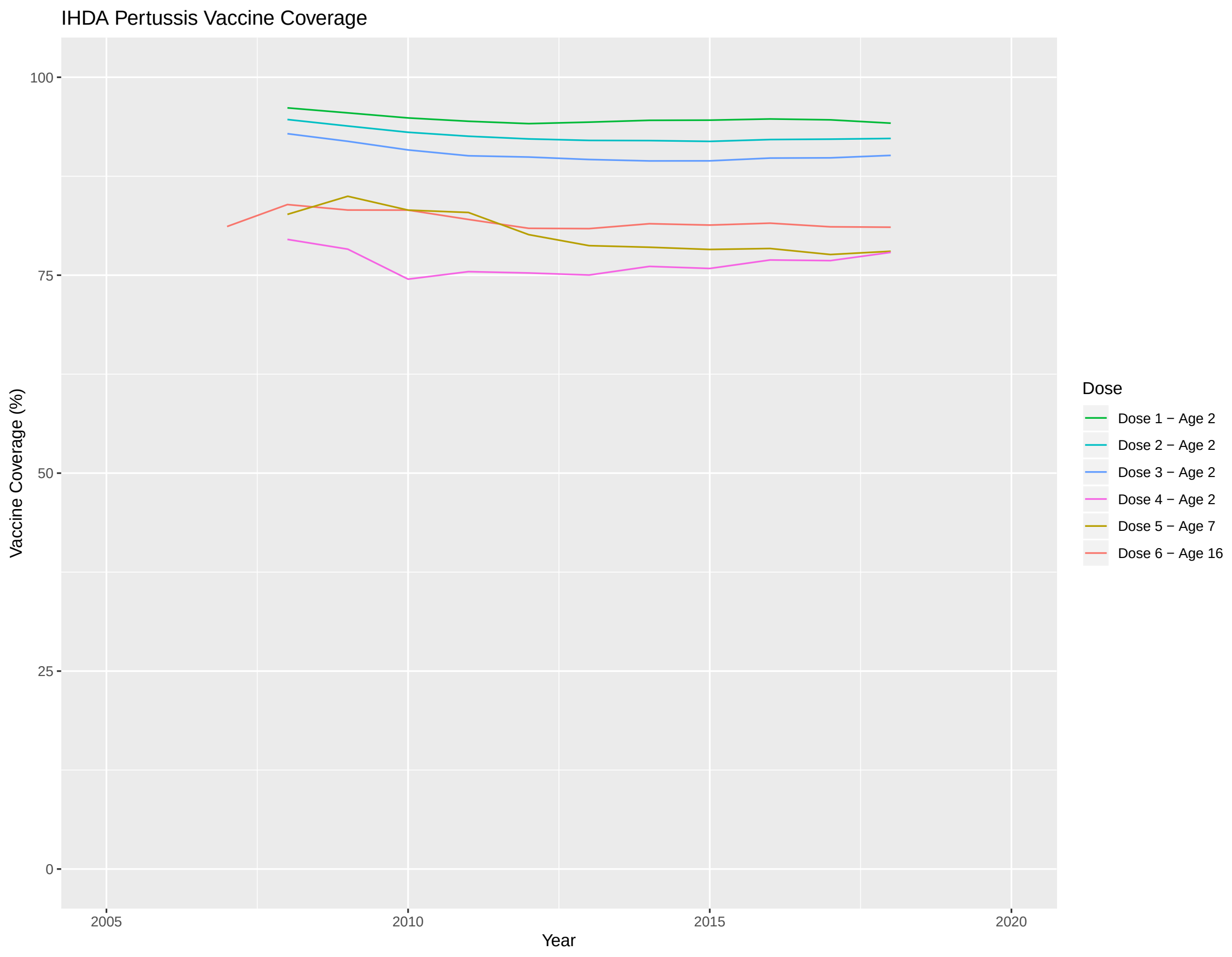


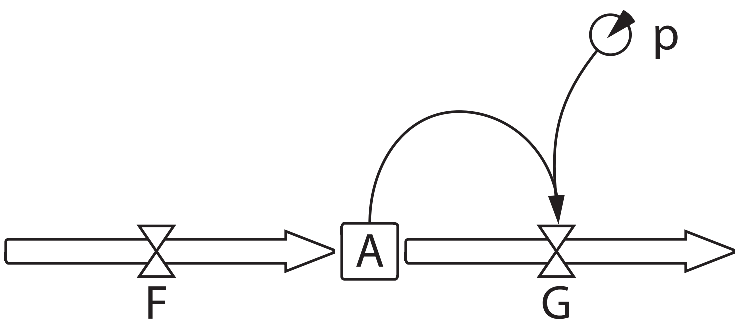
Figure 2.2: Pertussis Vaccine Coverage in Alberta 2007–2018 [15]

simple to understand, especially for non-mathematicians, and making explicit feedbacks and accumulations in a system that can be non-obvious when looking at a system of differential equations. ODE methods, on the other hand, have a long history pre-dating Forrester's work and continue to be preferred by many researchers because they do not abstract away from the mathematical equations.

Figure 2.3 shows a simple compartmental model expressed using SD stocks and flows and as an ODE. In the SD diagram, A represents a *stock*, which is a quantity. The double arrows, F and G, represent *flows*, which change the value of the stock A over time. The single arrows indicate instantaneous dependencies, so the value of G immediately and directly depends on the value of the stock A and the parameter p. This diagram translates to an ODE where the time-derivative of the stock, A, is equal to the sum of flows into A minus the sum of flows out of A.

Agent-Based Models

Agent-Based Models (ABMs), are simulation models defined by placing a number of *agents* within an environment; these models constitute a subset of Individual-Based Models, although the latter term is sometimes



$$\frac{\partial A}{\partial t} = F - pA$$

Figure 2.3: A Simple Compartmental Model in SD and ODE Notation

used interchangeably with “Agent-Based Model”. Railsback and Grimm [7] describe agents as “unique and autonomous entities that usually interact with each other and their environment locally.” Elucidating several terms in the previous definition:

- **Uniqueness** implies that agents have one or more characteristics or properties which differ from agent to agent [7].
- **Local Interaction** “means that agents do not interact with *all* other agents but only with their neighbors” [7].
- **Autonomy** “implies that agents act independently of each other and pursue their own objectives” [7].

Agents retain a state that can potentially include memory of the agent’s own history, allowing agents to move through “different ‘life cycles’ or stages... possibly including birth and death” [7]. Further, an agent’s behaviour can adapt in response to changes in its own state, other agents, and the environment [7].

Analyses applying ABMs focus on *emergence* by specifying agent properties and behaviours and then observing “system dynamics that arise from how the system’s individual components interact with and respond to each other and their environment” [7] while running the model. Railsback and Grimm [7] refer to ABMs as *across-level* models, to emphasize the fact that ABMs can represent changes in the *system* resulting from *individual* behaviour as well as changes in *individual* behaviour resulting from the state of the *system*. Additionally, ABMs often feature time- and space-variation of state variables of both individual agents and the environment [7].

Comparing Methodologies

Historically, aggregate ODE models have represented as a traditional, established form of dynamic modeling in epidemiology—most particularly mathematical epidemiology—with other methods occasionally being met with skepticism by scientists established in that field. Since the present work is an ABM, it is reasonable to address some strengths and limitations of the ABM approach when compared to the traditional aggregate ODE method. For simplicity in terminology, ODE/compartmental models referred to below will be of aggregate character unless otherwise stated.

- **Heterogeneity** of the population can be achieved simply in ABMs by adding properties to the agent definition and comes at a small incremental cost in terms of computation and memory. Compartmental models can also achieve heterogeneity via stratification, however it comes at the cost of adding compartments in a combinatorial way. For example, if a basic model has 3 compartments and we desire to stratify it in two sexes and five age groups, we end up with $3 \times 2 \times 5 = 30$ compartments in the final model; moreover, the changes required to support a new dimension of heterogeneity extend across the breadth of the areas of the model that must handle it. While such stratification—sometimes called "subscripting" or "arraying"—is well-supported by many SD modeling packages, the combinatorial expansion in model size is limiting [23, 24].

- **Individuality** is something that can not be captured in aggregate compartmental models, since they must divide the population into a finite number of compartments within which complete and random mixing is assumed. ABMs allow agents to have distinct, even unique, properties, including retaining a history of states, if required [23, 7].

- **Interaction** between agents in an ABM is said to be local [7]. Which other agents interact with a given agent can typically be specified with simple declarative rules that can vary at the individual level. Compartmental models can approximate this down to the compartment level. For example, in epidemiology, age-structured compartmental models typically specify contact rates between compartments in a *contact* or *transition* matrix. But such approaches cannot duplicate it to the individual level [24].

- **Scalability** in terms of computational resource requirements with growing population size favours compartmental models. A compartmental model can handle any population size without change in resource requirements provided that the number of compartments is fixed, i.e., $O(1)$, while ABM performance scales with population size, often super-linearly. Moreover, the constants involved strongly favour the run-time of compartmental models, which can often have run-times measured in seconds or minutes while large ABMs can take multiple hours to run [23].

- **Explainability** to stakeholders who may be non-technical or be experts in domains other than simulation modeling can be a valuable feature. Compartmental models summarized using the declarative diagramming language of SD have been found to be easily understood by people of varying backgrounds

[25]. ODE representations can be valuable for those with mathematical backgrounds—although elements of the formulation can be complicated by the need to reason about aggregation involved. ABMs, on the other hand, can contain a significant amount of computer source code, which are often indecipherable to non-programmers, and other components whose relationships may be non-obvious to people other than the model builder without strong documentation efforts [24].

2.2.2 Agent-Based Modeling in Health

One of the early contributions to mathematical modeling in health was Kermack and McKendrick's 1927 *A Contribution to the Mathematical Theory of Epidemics* [21], which proposed what is now known as the SIR model. SIR stands for Susceptible-Infectious-Recovered, which are the three compartments in the model. The model assumes a "closed", randomly mixing population with no births, deaths, or migration, and that a single infection confers permanent immunity. A large number of basic variations on this model have since been proposed and applied; these include the following:

- **SIRS** – Susceptible-Infective-Recovered-Susceptible allows for waning of immunity, whereupon people leave Recovered and return to Susceptible.
- **SEIR** – Susceptible-Exposed-Infective-Recovered adds a latent Exposed period, in which individuals are infected but not yet infectious.
- **SEIRS** – Susceptible-Exposed-Infective-Recovered-Susceptible provides a latent period as well as immunity waning.

These models can additionally be modified to characterize birth, death, migration, demographic stratification, and structured contact patterns, among other features.

Deterministic aggregate compartmental models have long been preferred in epidemiology "because it is easy to solve them numerically, interpret their results directly, and gradually increase their complexity by adding new compartments" [26]. These models, however, cannot fully account for some aspects of disease behaviour such as randomness and heterogeneity in contact patterns [26].

ABMs are gradually being adopted in health research. Badham et al. [27] state that there are now "calls for greater use of ABMs to understand public health issues and to formulate and evaluate plans to address them." They go on to state that ABMs can "effectively model systems governed predominantly by micro-level interactions or where there is substantial heterogeneity in agents' characteristics or their environment." Auchinloss and Diez Roux [28] encourage adoption of ABMs in epidemiology, suggesting that doing so may "promote thinking about the ways individuals interact with each other and with their environments." Chalabi and Lorenc [29] cautiously recommend use of ABMs while pointing out that rules governing agent behaviour must be evidence-based and that some may view them as black-box models. El-Sayed et al. [30] state that use of ABMs may enable researchers to "understand the etiologic implications of heterogeneity within the population, social interaction, and environmental influence simultaneously" but suggest that more work needs to be done in social network analysis in order to be able to draw solid conclusions from ABMs within that domain.

2.2.3 Dynamic Modeling of Pertussis

Compartmental Models

Mathematical modeling of pertussis began as early as 1989 in work by Grenfell and Anderson [13]. This study adapted previous ODE models of measles, mumps, and rubella to investigate the epidemiology of pertussis in the UK, specifically England and Wales. The model was stratified into 5 age groups, assumed that infants were born susceptible, and characterized primary vaccine failure and waning of vaccine-induced immunity. Analysis suggested that waning of natural disease-induced and vaccine-induced immunity, variability in case reporting efficiency, and seasonal variations all played a role in explaining observed trends.

A seminal contribution, upon which much further work was based, was made in 1997 by Hethcote [31]. This was an age-structured ODE model of pertussis featuring waning of both natural disease-induced and vaccine-induced immunity, as well as multiple degrees of infectivity. The model is a modified SIRS model with 12 compartments: four stages of natural disease-induced immunity (R_1, R_2, R_3, R_4), four stages of vaccine-induced immunity (V_1, V_2, V_3, V_4), as well as three levels of infectivity (I, I_m, I_w). The model replicated American incidence data from 1940 to 1995 and projected to 2040. The simulations suggested that vaccination of children alone as was, and continues to be, the practice, could not possibly produce a herd immunity effect for pertussis due to waning of immunity and persistence of low-level infections.

Hethcote and other authors went on to further develop the model to examine additional research questions. In one such study, Van Rie and Hethcote [32] applied the model to examine alternative vaccination strategies, including vaccination of adolescents, adults, and households with newborns (sometimes referred to as *cocooning*), in a US context. They found that, of the strategies examined, vaccinating adolescents had the lowest number needed to vaccinate (NNV) to prevent one case of pertussis in the general population, while cocooning had the lowest NNV to prevent one case in infants.

Hethcote, Horby, and McIntyre [33] performed a similar study in Australia with a modified version of the 1997 model to assess a recent change to vaccination scheduling in that country, and found that the change was likely to reduce pertussis incidence as intended.

Fabricius et al. [34] applied a modification of Hethcote's 1997 model to evaluate the impact of adding a vaccine dose at 11 years of age in an Argentinian context. They found that, while the additional dose reduced incidence overall, it did little to benefit infants, who are most vulnerable. Their recommendation was to instead improve first-dose coverage.

Pesco et al. (2014) [35] applied a modification of Hethcote's 1997 model to examine recent trends of increasing pertussis incidence in multiple jurisdictions within the United States. The study focused specifically on vaccine effectiveness and effective transmission rates. Findings were that reduction of vaccine effectiveness led to increased incidence in infants and children, while changes to transmission rates led to higher incidence in adolescents with large cyclic outbreaks.

Pesco et al. (2015) [36] applied a modification of Hethcote's 1997 model to examine reduction of delays in vaccine administration as a strategy to reduce incidence in an Argentinian context. Findings indicated that reducing delays improves vaccine coverage and protection of infants.

Gambhir et al. [37] fitted a compartmental model of pertussis designed by the researchers to incidence data from the United States using Markov Chain Monte Carlo (MCMC) methods. The goal was to examine potential causes for increasing pertussis incidence. Their results suggested a difference in efficacy between the whole-cell (wP) and acellular (aP) pertussis vaccines, that "adults and adolescents may be a significant reservoir of infection," and that the immunity induced by the wP vaccine may be "nearly equivalent to that of the natural infection."

Agent-Based Models

Sanstead et al. [38] developed an ABM representing pertussis epidemiology in Dakota County, Minnesota, USA using NetLogo software; the focus of the study lay in examinining possible causes for observed increases in pertussis incidence. People in the model were divided into four age groups. Some key assumptions were that vaccinated children received the full 5-dose series with "no delays and 85% vaccine efficacy," vaccine-induced immunity did not apply until the first three doses were received, and waning of immunity was "instantaneous and complete". Individuals were assumed to interact with each other "randomly each day without restrictions" with transmission probabilities that were derived from Hethcote's 1997 model. The latent period of infection was neglected, variable degrees of infectiousness were not represented, and infected people were assumed to be infectious for 21 days regardless of symptoms unless treated with antibiotics. People treated with antibiotics were assumed to have reduced infectiousness and be sequestered from interactions. The model successfully replicated age-structured pertussis incidence in Dakota County during three key outbreaks. The model was highly sensitive to adult immunity parameters. Findings suggested that current trends may be influenced by waning of vaccine-induced immunity, underreporting, and fluctuations in adult immunity.

Doroshenko et al. [5] developed an ABM representing pertussis epidemiology in Alberta, Canada using AnyLogic software, with a focus on examining outbreak response immunization (ORI) as a possible intervention strategy. Findings suggested that "outbreak response may yield modest benefits for protecting infants". This model was a key reference for the present work due to the involvement of co-authors Drs. Doroshenko and Osgood, the focus on pertussis in Alberta, and the use of AnyLogic as the development tool.

Key features of the Doroshenko et al. model are [5]:

- A nominal population size of $500,000$ people with dynamic births and deaths (an "open" population).

- The population was distributed across a stylized square urban area with a high-density core surrounded by a lower density periphery.
- Agents were connected within a *distance-based* contact network, meaning two agents were connected if and only if their Euclidean distance was below a certain threshold. The threshold itself was age-dependent, with a larger radius being used if both agents were under 16 years of age.
- Disease was represented using an ABM adaptation of Hethcote's 1997 compartmental model [31], the statechart for which is shown in Figure 2.4.
- An ORI module designed to detect incipient outbreaks within the model and apply the intervention.
- A Grid data structure which provided improved range search performance over AnyLogic's built-in methods.

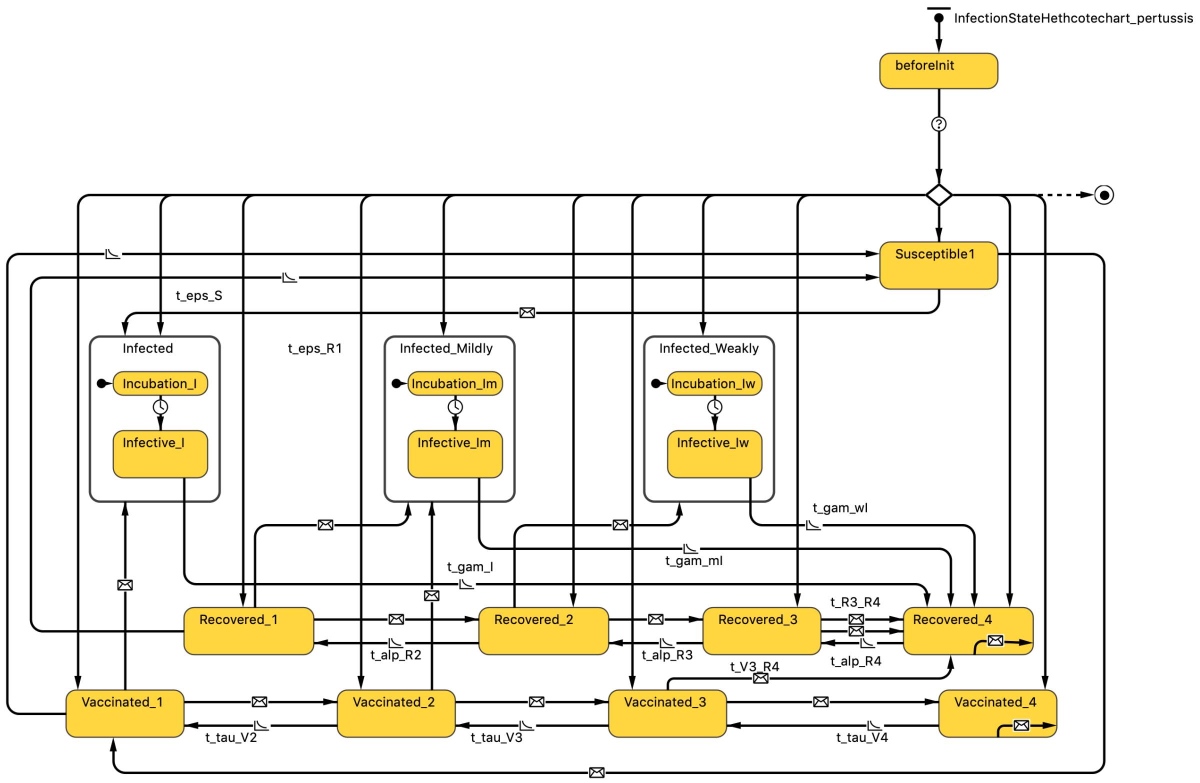


Figure 2.4: Statechart for ABM Adaptation of Hethcote's 1997 Model [31] by Doroshenko et al. [5]

This work has drawn materially on the Doroshenko et al. [5] work in three key ways. First, learning from the implementation of the stylized square urban area was applied in development of the population density map for this work. Second, the ABM adaptation of Hethcote's model [31] was used both as-is and in a modified form, see **Disease Module** in Section 3.2.1. Third, the Grid data structure was incorporated into this work, as detailed in Sections 2.3.2 and 4.2.1.

2.3 Data Structures

AnyLogic's built-in `agentsInRange()` method for an agent, A, returns a collection of other agents within a population lying within a specified range of A; to accomplish this, it searches the entire population and computes the distance to each other agent. For a given population size n, this procedure is $O(n)$ for each operation and is always worst-case. Thus it becomes advantageous to search for algorithms and data structures to improve this search time. Storing the network connections, on the other hand, requires $O(e)$ storage, where e is the number of edges in the graph. AnyLogic stores the network graph as an adjacency list in each agent, so this has an upper bound of $O(n^2)$ storage, where n is the number of agents, so enhancing computational efficiency in this area is also of interest.

2.3.1 Range Search

Range search is considered "a central problem in computational geometry" [39] and, stated plainly, involves determining which points lie within a specified subset of a space.

The specific case applicable to this project involves determining, in 2-dimensional space, which other points lie within a certain distance of a given point. Here, the *range*, in its geometric sense, delineates a circle centred at the coordinates of the given point. Further, we are not simply interested in reporting the geometric coordinates, but using this method as an index into a set of agents to speed searching. An additional simplifying consideration in this application is that the agents are not freely mobile; instead, their positions remain static throughout the majority of the model run.

Agarwal [39] indicates that a single range search query can be solved by brute force in $O(n)$ time, which is what AnyLogic's built-in `agentsInRange()` method does. Most applications, however, including this one, require many queries, so the goal becomes pre-processing the set of points into a data structure that speeds searching.

Orthogonal range searching [39] further constrains the problem by applying the restriction that the ranges be rectangular and axis-aligned. For this application, the range is a circle, so the strategy is to do an orthogonal range search within a square bounding box, and to then refine the search by computing distances. Figure 2.5 shows a general range search on the left and an orthogonal range search with inscribed circle on the right.

2.3.2 Grid Structure

The Grid structure was originally implemented in AnyLogic by Weicheng Qian [5], but is not particularly novel by computer science standards. Two-dimensional Cartesian space is divided into grid squares of a chosen size. A data structure, accessible by index, represents those grid squares. The record of each grid square contains a sequentially accessed data structure which stores references to every record whose position falls within the bounds of that square, geometrically.

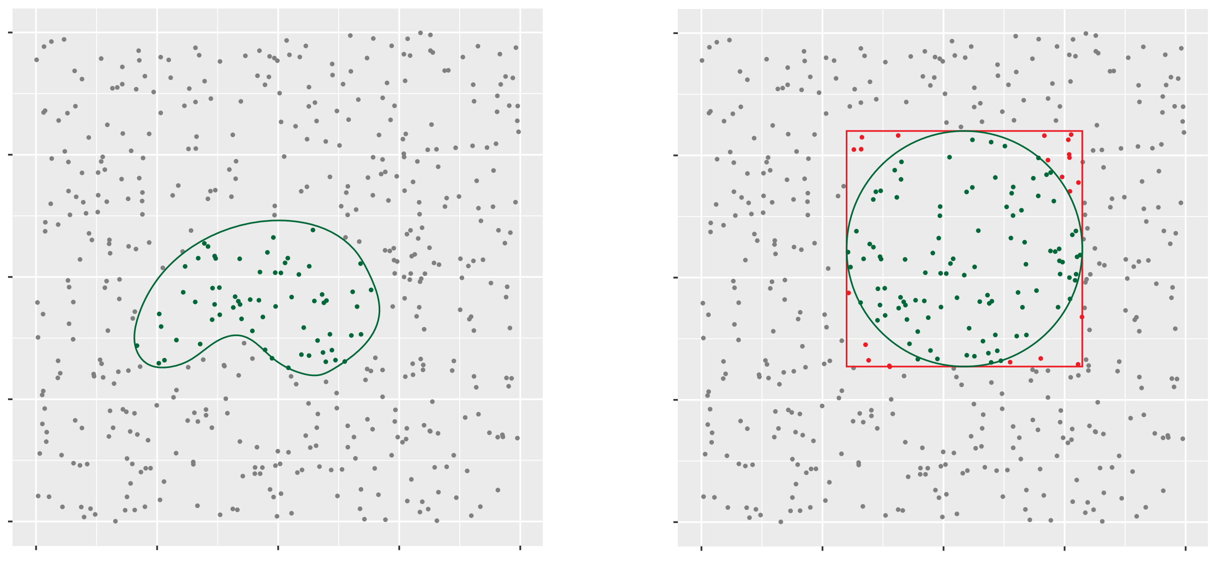


Figure 2.5: Range Searches: General (left), Orthogonal and Circular (right)

The benefit of the Grid structure is realized when performing a Range Search. To obtain a set of agents within a certain distance of a given agent, A, one need only search the square to which the A belongs and any adjacent squares that are enclosed by or intersect with a circle representing the maximum range. This reduces the search space, speeding the operation; however, the scaling remains $O(n)$. For large populations and relatively short ranges, this can result in a significant improvement, while in the worst case it is $O(n)$, where n is the entire population—no worse than the built-in algorithm.

The grid square size must be tuned for given space dimensions and agent density in order to secure the best average-case performance. The worst case would have a single square for the entire space, which reduces the problem to sequential search of the entire population, $O(n)$. In the opposite extreme, making squares arbitrarily small increases the memory overhead of the Grid as well as allocation time.

To perform a range search on agents in the Grid structure, first the indices of bins to search are computed from the coordinates of the index agent and a given range. This first step is key to the performance improvement sought, as it eliminates computation of distance for agents far out of range of the index agent. The agents contained in the selected bins are then further filtered based on their computed distance from the index agent and the final set of agents in range is returned. Geometrically, it would be possible to add an entire bin to the result, without computing distances, if it was known to be fully contained in the range circle, leading to additional gains; this is not the case in the current implementation, however.

2.3.3 k-d Tree

A k-d Tree, standing for k-dimensional tree, is a binary tree-based data structure for storing data with multidimensional keys. Branches of the tree are partitioned based on a single dimension of the key, cycling through the dimensions as the tree deepens. Comparing based on only one dimension at each level has the benefit of reducing computation as well as storage compared to the QuadTree, another method for storing point data. A disadvantage of the k-d Tree is that its structure depends on the order of insertion of data as well as the chosen order of alternation through the dimensions of the key. Another disadvantage is that a subtree of a k-d Tree is not a valid k-d Tree [40].

Samet [40] describes a specific variant of the k-d Tree—termed a Point k-d Tree—that "partitions the underlying space at the data points and cycles through the different axes in a predefined and constant order." The implementation in this work specifies an $x, y, x, y...$ order. This particular tree is summarized below and further implemented in Java as part of this work, as described in Section 4.1.

The discriminator indicates which dimension of the key is being considered for the current node. If a given node, N, is an x-discriminator, then all nodes in its left subtree have x-values less than that of N and all nodes in the right subtree have x-values greater than or equal to that of N [40].

To insert a node into a k-d Tree [40]:

- If the tree is empty, insert the new node as the root of the tree.
- Otherwise, search the tree for a match for the new node's key.
- If an exact match is found, then the record of the new node replaces that of the found node.
- Otherwise, create the new node at the found location, which is an empty leaf node.

Consider the example of inserting several records into a two-dimensional point k-d Tree, adapted from Samet [40], shown in Figure 2.6. On the left, the partition of 2-d space is shown, and on the right, a diagram of the tree structure. *(a)* shows insertion of the first two nodes; node A is the first record inserted, and is thus placed at the root of the tree. A becomes an x-discriminator, so to insert B we compare on the x-coordinate. Because $66 > 48$, we move to the "high" child of A, which is empty, so B is inserted there. *(b)* to insert node C, we first compare its x-coordinate to that of A; $10 < 48$, so we move to the "low" child of A, which is empty, so C is inserted. *(c)* inserting D, we again compare its x-coordinate to that of A; $75 > 48$, so we move to the "high" child, which is B. B is a y-discriminator, so we compare D with B on the basis of their y-coordinates; $12 < 58$, so we move to the "low" child of B, which is empty, so D is inserted there. *(d)* inserting E, we compare its x-coordinate to that of A; $20 < 48$, so we move "low" and compare with C based on the y-coordinate. $18 < 56$ so we move "low" again and insert E as the "low" child of C. *(e)* inserting F, we compare with A based on x. $53 > 48$ so we move "high" and compare with B based on y. $90 > 58$, so we move "high" again and insert F there.

In performing a range search, subtrees that are outside of the bounding box of the range can be excluded, reducing the number of nodes that must be traversed. The search is performed recursively, beginning at the root of the tree. If the coordinates of the node being examined lies within the bounds of the range, then the record(s) at that node are added to the result. If the coordinate of a child node, according to its

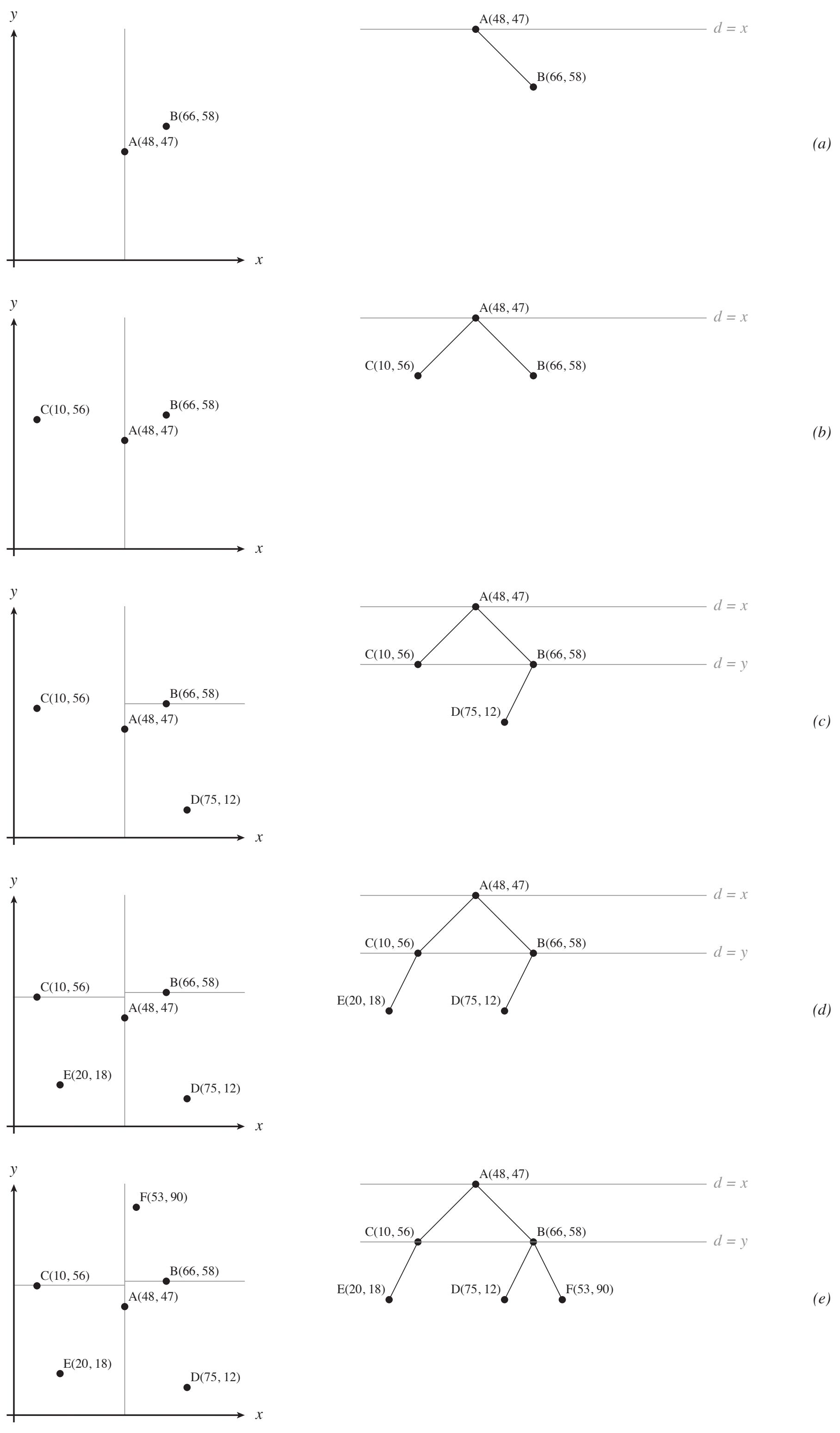


Figure 2.6: An Example of Inserting Records into a k-d Tree

discriminator, falls within the bounds, then it is examined recursively; otherwise, it can be "pruned", being excluded from the search [40, 41, 42].

Theoretical timing for the k-d Tree is $O(kn \log(n))$ for setup, where k is the key size or number of dimensions, and $O(\log(n))$ for access [41, 42]. In order to maintain this performance, the tree must remain in balance. In the worst case, where the tree is fully unbalanced, it becomes a list and access is sequential, $O(n)$. Tree balance is discussed further in Section 4.1.2.

2.4 Latin Hypercube Sampling

Latin Hypercube Sampling (LHS) is a technique developed by McKay et al. in 1979 [43] to sample from high dimensional spaces. In a Latin Square sample, the 2-dimensional sample space is divided into a square grid, and the sampling procedure ensures that there is one sample in each row and one in each column [43, 44]. Extending this concept and generalizing this invariant to three or more dimensions yields a Latin Hypercube Sample [43].

3 Pertussis Model

An Agent-Based Model (ABM) of pertussis disease was developed using AnyLogic software. This chapter discusses the development process and resulting model.

3.1 Group Model Building

An interdisciplinary group model building process was employed in development of the model, as outlined in this section.

3.1.1 Team Composition

This model was developed by an interdisciplinary team consisting of the following members:

- **Dr. Alexander Doroshenko, MD, MPH** is an Associate Professor in the Division of Preventive Medicine and Adjunct Professor in the School of Public Health at the University of Alberta, as well as a Medical Officer of Health with Alberta Health Services. Dr. Doroshenko served as the primary stakeholder, expert in epidemiology and medicine, and co-Principal Investigator on the project.
- **Dr. Karsten Hempel, PhD** is a Postdoctoral Fellow in the Department of Medicine at the University of Alberta. Dr. Hempel brings expertise in statistics and mathematical modeling and served as one of the primary model builders on the project.
- **Dr. Nathaniel Osgood, PhD** is a Professor in the Department of Computer Science and Associate Faculty in the Department of Community Health and Epidemiology and Division of Biomedical Engineering at the University of Saskatchewan. In addition to his capacity as supervisor for this thesis work, Dr. Osgood served as the primary expert on dynamic modeling and computer science and co-Principal Investigator on this project as well.
- **Wade McDonald, BE, BSc** is a Master's student in the Department of Computer Science at the University of Saskatchewan, and the author of this thesis. Mr. McDonald served as one of the primary model builders on the project.

3.1.2 Advisory Panel

The project was further advised by an Advisory Panel consisting of the following experts:

- **Dr. Natasha Crowcroft, MD, PhD** is a Professor in the Dalla Lana School of Public Health at the University of Toronto.
- **Dr. David Fisman, MD, MPH** is a Professor in the Dalla Lana School of Public Health at the University of Toronto as well as an Attending Physician at a number of Toronto-area hospitals.
- **Dr. Scott Halperin, MD** is a Professor in the Division of Infectious Diseases, Faculty of Medicine at Dalhousie University.
- **Dr. Nicola Klein, MD, PhD** is a Senior Research Scientist at the Kaiser Permanente Northern California Division of Research and director of the Kaiser Permanente Vaccine Study Center.
- **Dr. Pejman Rohani, PhD** is a Professor at the Odum School of Ecology and the Department of Infectious Diseases, College of Veterinary Medicine at the University of Georgia.

3.1.3 Modeling Process

The modeling process was generally conducted according to an iterative cycle of model development by the primary model developers, consultation with the principal investigators, and periodic review by the Advisory Panel.

This model was initially envisioned as an extension of the model developed by Doroshenko et al. in 2016 [5]. Dr. Doroshenko, as primary stakeholder, requested that a number of additional features be added to the model. After discussions on how features would be implemented, it was decided that a new model would be created, while incorporating key features from the 2016 model.

Plans were presented to the Advisory Panel and implementation of the model proceeded. Work was organized such that the primary model builders worked independently while having approximately weekly video meetings to coordinate their efforts. Each would meet with their local co-PI to discuss progress and issues. A number of times throughout the project, all four core team members would gather at the University of Alberta or University of Saskatchewan for intensive three to four day work sessions.

At the second Advisory Panel meeting, a working model was presented that implemented key features. With feedback from the advisory panel, model development continued, adding a number of additional features.

At the third Advisory Panel meeting, a substantially feature-complete model was presented. After including feedback from the panel, the process of calibration began. Calibration of the model was initially done in a "manual" manner, where parameter values were selected, the model was run, and outputs were judged for realism.

Leading up to the next Advisory Panel meeting, summary statistics and objective functions for automated calibrations were developed. A tentative parameter set was selected and the main experiment was run, along with sensitivity analyses. These changes and outputs reflecting them were discussed at the fourth Advisory Panel meeting.

With input from the Advisory Panel, work proceeded with a fulsome automated calibration, discussed further in Section 3.3. Following this, the main experiment and sensitivity analyses were to be run again to produce results; see Section 3.5.

3.2 Model Structure

The model was developed using AnyLogic software; its construction is outlined in this section.

3.2.1 Agents

The model was composed of the agent types outlined below.

Person

The `Person` agent represented a human being in the model. People were parametrized as having an initial age, sex, and initial vaccine acceptance. At initialization, a `Person` was assigned a `Household`, and either a `Workplace` or `School`—depending on their age. The behaviour of a `Person` in the model was further governed by statecharts representing pregnancy and vaccinations depicted in Figure 3.1 as well as a `DiseaseModule`, described below. Definitions for statechart iconography can be found in the AnyLogic Online Help [45].

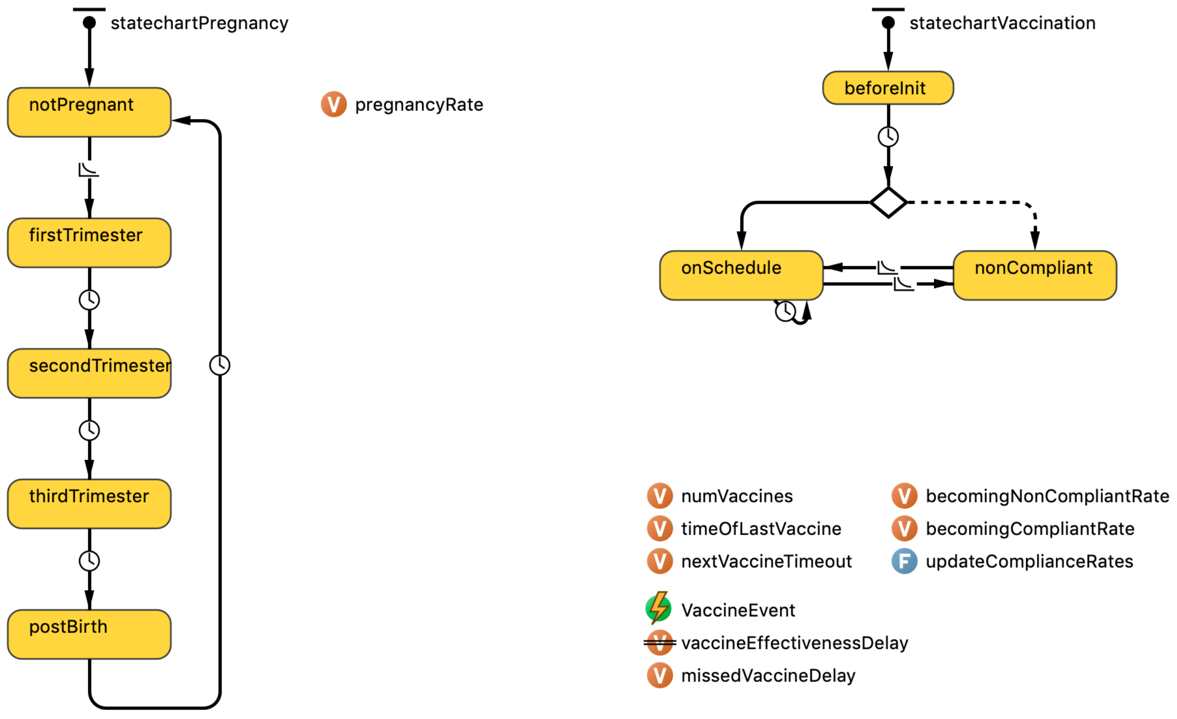


Figure 3.1: Statecharts for the Person Agent

A `Person` may be not pregnant, or may be in the first, second, or third trimester of pregnancy, or may be in a post-birth period. A male `Person` was prevented from entering pregnancy states by a *guard* statement on the pregnancy transition. A female `Person` would become pregnant with a fertility rate determined by their age and number of previous children. Upon entering the `thirdTrimester` state, the pregnant `Person` may receive a maternal vaccination if this intervention was enabled for the model run. Upon exiting the `thirdTrimester` state, a new `Person` of age 0 would be instantiated and added to the `Household` associated with the mother.

In the vaccination statechart, a `Person` would first enter the `beforeInit` state, where their vaccine acceptance was updated before placing them in either the `onSchedule` or `nonCompliant` state. A `Person` would transition between `onSchedule` and `nonCompliant` at a rate depending on their vaccine acceptance. While `onSchedule`, a `Person` will receive vaccines according the the schedule. If `nonCompliant`, they would not receive the vaccine but may catch up to the prescribed schedule if they return to `onSchedule` at a later time.

As a `Person` aged, pregnancy rate and death rate were updated annually. At age 18, a `Person` would leave their current `Household` and seek a new one, as well as leaving their `School` and joining a `Workplace`. At age 65, a `Person` would leave their `Workplace`, representing retirement.

Household, School, and Workplace

`Household`, `School`, and `Workplace` agents represented contact venues, or pools, in which people could spread disease if infectious. They were constructed using the `Agent` class within AnyLogic but did not have true agency within the model. Each of these venues kept track of the people that belonged to it as well as some conditions for membership, e.g., grade levels for Schools, and summary statistics.

`Schools` and `Households` had a `vaccineAcceptance` property, which influenced the vaccine acceptance of their members; this component was added to the model as a simple way to achieve clustering of people with similar vaccine attitudes, as is observed in Alberta [15].

`Households` persist; once the last `Person` residing there dies, the `Household` would be re-occupied by a new `Person` coming of age. If a `Person` came of age and there was no `Household` available, one would be instantiated.

Disease Module

The `DiseaseModule` was implemented following the Modular Design Pattern established by Kreuger [46]. This design pattern allowed the implementation of multiple representations of the disease mechanics of pertussis which could easily be swapped out one for another. In effect, this allowed parametrization of model structure.

The interaction of the `DiseaseModule` was defined by a Java interface called `DiseaseMechanicModule`, which specified the contract through which the module implementation interacted with the rest of the model

[46], see Listing 3.1. Each model run was parameterized by a specific class implementing the `Disease-MechanicModule` interface, with each `Person` agent in that run being associated with an instance of that class, which determines their disease behaviour.

```java
public interface DiseaseMechanicModule
{
  public boolean handleExposure(double forceOfInfection);
  public void handleVaccination(VaccineType type);

  public void initializeImmuneState(double protectionLevel, PrimingType primingType);

  public boolean isInfected();
  public boolean isInfective();
  public boolean isSusceptible();

  public double getForceOfInfection();
  public double getInfLevel();
  public double getProtectionLevel();
}
```

Listing 3.1: Disease Mechanic Module

Four Disease Modules were implemented:

- "Hethcote" – the agent-based adaptation of Hethcote's [31] compartmental model by Doroshenko et al. [5].
- "Hethcote Plus" – a modification of the above to add states for maternal antibody immunity in infants.
- "De Novo" – a new model of pertussis pending publication.
- "Null" – a minimal module to reduce computational burden for testing purposes.

The "Hethcote" module was implemented exactly as was done by Doroshenko et al. [5], see Figure 2.4. The "Hethcote Plus" module added five states for maternal immunity in infants, as shown in Figure 3.2. The methods of the `DiseaseModule` interface returned specified values based on the current state of the statechart.

The "De Novo" module represented disease states as continuous values, rather than discrete states. It also contained a statechart, shown in Figure 3.3, to indicate basic infection progression. `getForceOfInfection()`, `getInfLevel()`, and `getProtectionLevel()` returned continuous values representing disease state.

The "De Novo" model was developed by Drs. Doroshenko and Hempel. In it, protection, P, was measured as a number between 0 and 1. Exposure to pertussis caused protection to jump by an amount, specified in a parameter, depending on whether it was a disease exposure, or an administration of whole-cell pertussis vaccine (wP), or acellular pertussis vaccine (aP). Each disease module retained memory of the highest P it had attained due to each source of exposure. Immunity waned at a rate that depended on the source of the protection, with the rates of waning also being specified in parameters.

Maternal protection may be transferred to an infant upon birth, depending on a parameter setting. This *passive* protection, initially equal to the mother's *active* protection, would wane with a mean period of

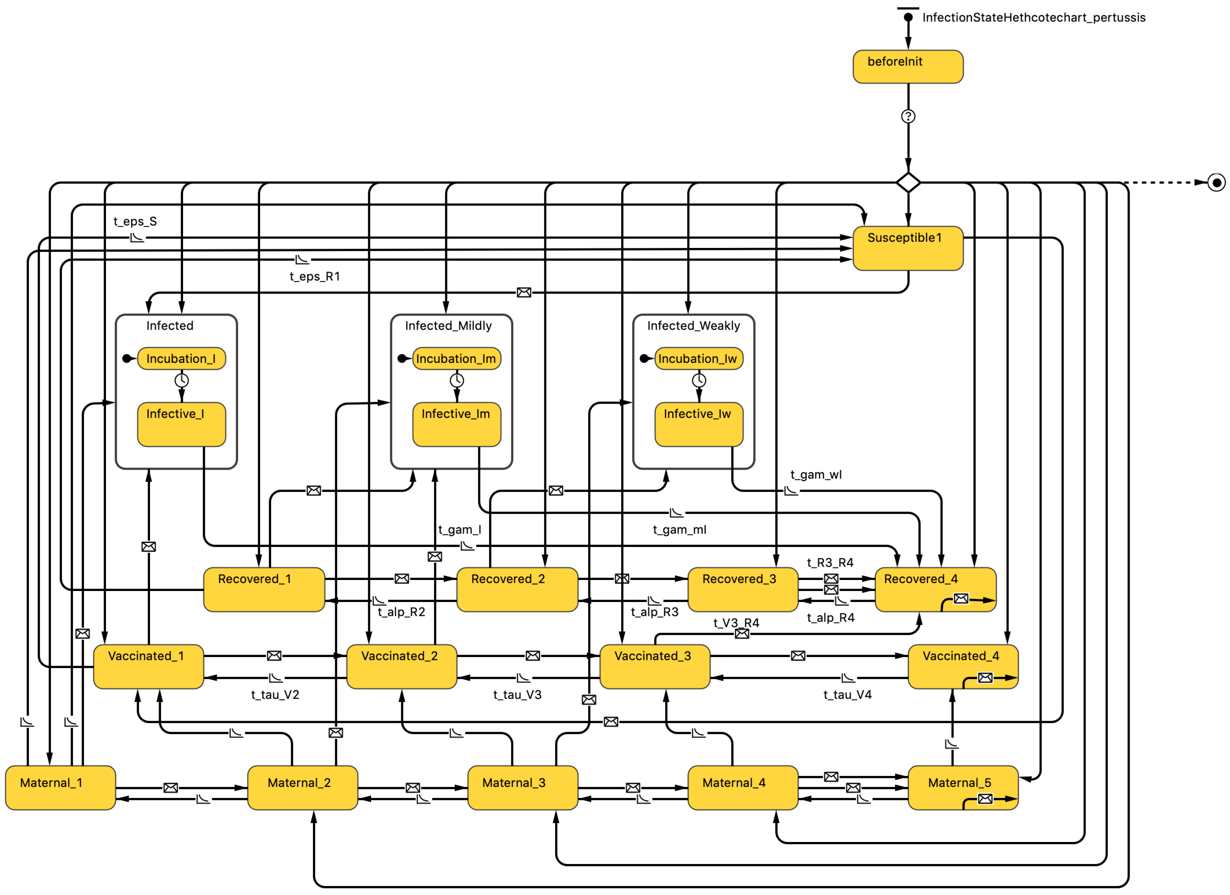


Figure 3.2: Statechart for the "Hethcote Plus" Disease Module

6 months. A blunting effect may optionally be enabled, wherein passive protection would interfere with protection conferred by vaccination of infants. The parameter `dose_blunting_proportion`, ranging from 0 to 1, specifies the factor by which the jump in protection is reduced if a child's mother was vaccinated during pregnancy.

Epi Curves

The `EpiCurves` agent is another construct within the model that used the `Agent` class but lacked true agency. Use of the `Agent` class allowed the `EpiCurves` to be implemented in the AnyLogic graphical user interface (GUI), reducing development time compared with a simple Java class. Its purpose was to log model output for and render visualizations of disease incidence over time, stratified according to specified age groups, sometimes called *Epi Curves*. The `EpiCurves` agent could log case counts and incidence in three categories for up to fifteen age groups in each of two time intervals. Age groups and time intervals were specified in parameters for easy modification. Figure 3.4 shows an example visualization in the `EpiCurves` agent during a model run.

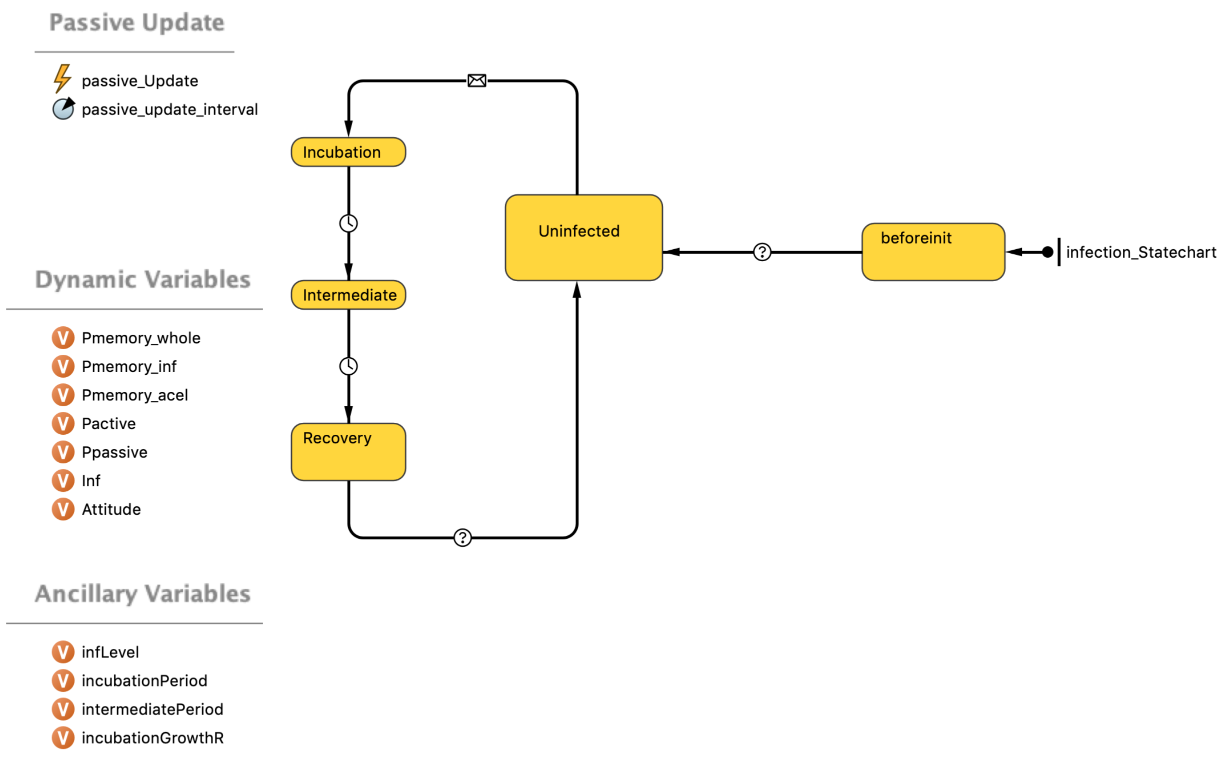


Figure 3.3: Statechart for the De Novo Disease Module

3.2.2 Networks

The model implemented multiple networks, each described below. Here, *network* refers to connections between agents through which they communicate or interact. In this model, the networks represented a contact network, through which an infected `Person` could expose other `Person` agents to the pathogen. For performance reasons, the built-in `Connections` construct in AnyLogic, which is typically used to implement networking, was eschewed in favour of a custom implementation, discussed in detail in Chapters 4 and 5.

Household, School, and Workplace

The `Household`, `School`, and `Workplace` contact venues were implemented as agents which contain a list of member `Person` agents. `Person` agents were assumed to be fully connected within the venue. By default, random mixing is assumed within all venues. The `Household` and `School` optionally allowed weighting of contact rates by age, and the `School` also allowed biasing contacts towards students in the same grade level.

A `School` had a nominal capacity and grade level range specified as parameters. Children were allocated to the nearest `School` of the appropriate grade level with free capacity. If nearby `Schools` were at or above capacity, a tradeoff between distance and capacity would occur.

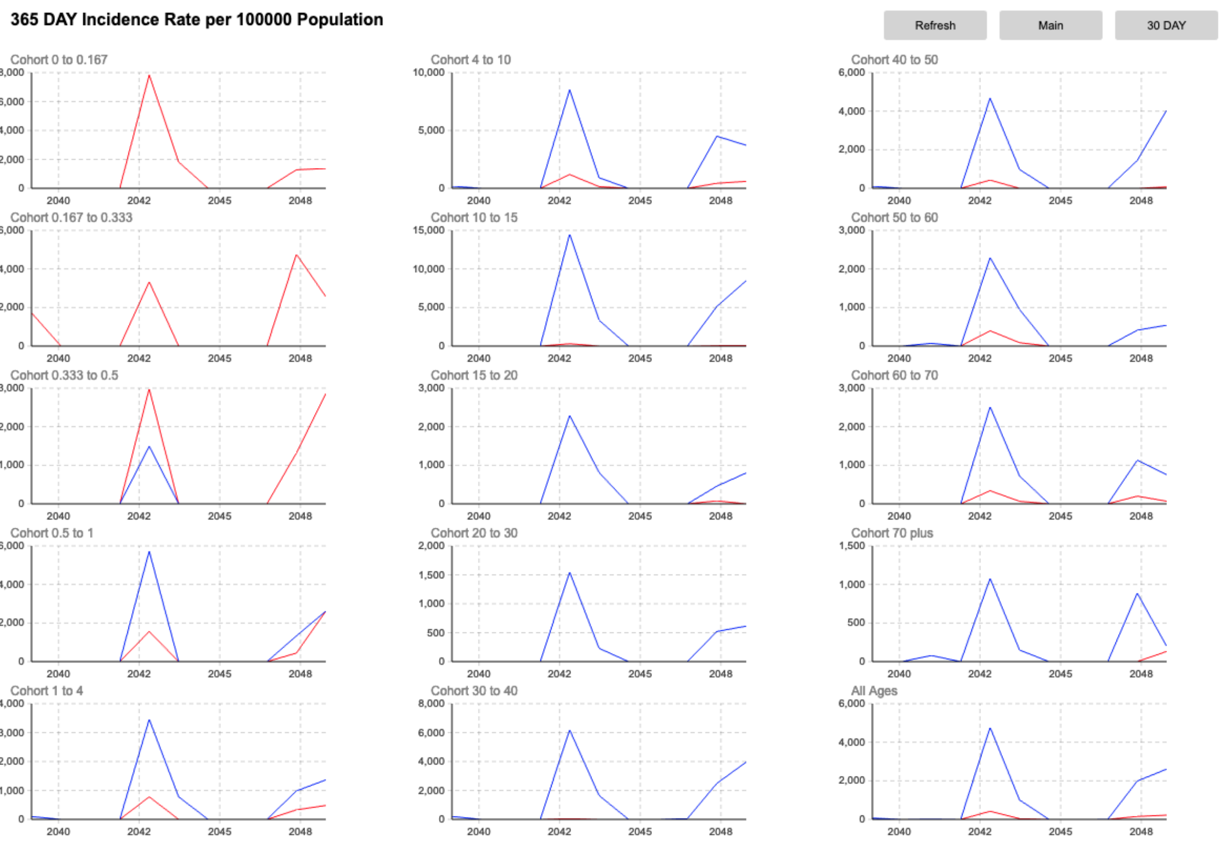


Figure 3.4: Visualization in Epi Curves Agent

Background Contacts

Background contacts represented any contacts that did not occur in the `Household`, `School`, or `Workplace` venues. Background contacts were implemented in two ways: Via fixed rates regardless of age, or via a contact matrix.

When using fixed rates, an infectious `Person` would send an exposure message to a random `Person`, selected with uniform probability, within `backgroundConnectionRange` at the rate—probability per unit time—of `backgroundContactRate`. The parameters `backgroundConnectionRange` and `backgroundContactRate` did not vary from one `Person` to the next, this differed from the 2016 Doroshenko et al. model [5] which varied the connection range.

A contact matrix is typically used in aggregate compartmental modeling to define mixing rates between compartments, often representing age groups, where random mixing is assumed within those compartments. Each element, a_{ij}, of a contact matrix, $\mathbf{A}$, represents the rate of contact between an infected person of age i and another person of age j [47]. In this model, when using the "contact matrix" setting, the probability of selecting a particular `Person` as a contact depended on the age of the index `Person` and that of the `Person` to be contacted, rather than the probability of selection being uniform across other individuals within range.

3.2.3 Parametrization

Directly Specified Parameters

Some parameters for the model were specified directly in the AnyLogic editor or, optionally, loaded from a comma-separated value (CSV) file listing their values. The `parameterSetFilePath` parameter specified the directory path to a parameter set file to be loaded. If `parameterSetFilePath` was `null` then values set in the AnyLogic editor or model GUI would be used.

Table 3.1 lists parameters that describe the epidemiology of the model. Table 3.2 lists parameters that describe the demography of the model, which also influences epidemiology.

Table 3.3 specifies configuration parameters that affect the epidemiology of the model. Changes to these parameter values can be considered changes to model assumptions, and may necessitate re-calibration.

Table 3.4 lists configuration parameters that do not affect the epidemiology of the model. These parameter values can be freely adjusted without influencing model outcomes.

Inputs

Specific inputs that included many subcomponent values were loaded from files to allow for flexibility without adding large numbers of parameters. These are:

- **Population Density Map** – This file defined the model space, divided into a square grid, and specified the population density of each square. The specific file to be loaded was specified in the parameter `densityFile`. This mechanism is discussed further in Section 3.4.
- **Live Birth Probabilities** (`LiveBirth_probs.csv`) – defined the rate (probability per unit time) of live births per day per fertile woman stratified by age of mother and number of children she already has. This is used to set the pregnancy rate of the `Person` agent. Derived from Alberta Open Data [55].
- **Household Types** (`household_type_AB.csv`) – defined the distribution of household types and was used in initialization of the population. Households were divided into *Couple*, *Single Male*, and *Single Female* types. Derived from Alberta Open Data [55].
- **Number of Children** (`num_children_AB.csv`) – defined the distribution of the number of children per household stratified by household type and was used in the initialization of the population. Derived from Alberta Open Data [55].
- **Age of Children** (`child_ages_AB.csv`) – defined the distribution of child ages within a household and was used in initialization of the population. Derived from Alberta Open Data [55].
- **Age of Parents** (`adult_ages_AB.csv`) – defined the distribution of adult ages stratified by number of children in the household and was used in initialization of the population. Derived from Alberta Open Data [55] and Canada 2016 Census [56].

Table 3.1: Epidemiological Parameters

Parameter	Value	Unit	Source
backgroundConnectionRange	23.1	distance	Calibration
exogenousInfectionRate	0.037117117	day^{-1}	Calibration
vaccineCatchupDelay	30	day	Expert
catchupVaccineCutoffAge	18	year	Expert
maxBackgroundConnections	100	person	Arbitrary
backgroundContactRate	0.036	day^{-1}	Calibration
householdContactRate	0.09	day^{-1}	Calibration
schoolContactRate	0.5	day^{-1}	Calibration
workplaceContactRate	0	day^{-1}	Arbitrary
schoolVaccineAcceptance_p	2.6	–	Calibration
schoolVaccineAcceptance_q	0.9	–	Calibration
primedProtectionVacc	0.65	–	Arbitrary
primedProtectionDisease	0.75	–	Arbitrary
probOfPrimaryVaccineFailure	0.15	–	[48]
epsilonAgePreferenceForSchoolContacts	0.9	–	Arbitrary
thresholdForTransmission	0.75	–	Arbitrary
thresholdForCaseCount	0.25	–	Arbitrary
pJump	0.25	–	Arbitrary
probOfPrimaryVaccineFailureDose6	0.15	–	[48]
Pactive_decay_acel	5.48×10^{-5}	day^{-1}	[49, 50]
Pactive_decay_inf	2.74×10^{-5}	day^{-1}	[51]
Pactive_decay_whole	2.74×10^{-5}	day^{-1}	[51]
probOfMissingDose5and6	0.33	–	Calibration
primingBetaP	10	–	Arbitrary
adultReportingEfficiency	0.01	–	Arbitraty
Ppassive_decay	0.022	day^{-1}	[52]

Table 3.2: Demographic Parameters

Parameter	Value	Unit	Source
childrenProportion	0.2167513	–	[53]
schoolCapacityRatio	0.8	–	[53]
postBirthPeriodLength	6	month	Expert
schoolHoursStart	9	hour	Arbitrary
schoolHoursEnd	15	hour	Arbitrary
workHoursStart	9	hour	Arbitrary
workHoursEnd	17	hour	Arbitrary

- **School List** (`school_list.csv`) – defined a sampling of Alberta schools with grade ranges and student capacity and was used in the initialization of `School` agents. Derived from Alberta Student Population Enrolment [53]

- **Death Rates** (`deathrates_AB.csv`) – defined the mortality rate (probability per unit time) stratified by age. This was used to set the death rate for `Person` agent. Derived from Canada 2016 Census [56].

- **Vaccination Schedule** (`vaccination_schedule.csv`) – defined the vaccination schedule, and indicated the age in years at which a child is due for a vaccine, and the type of vaccine administered.

- **Contact Matrix File** (`POLYMOD_Finland.csv`) – defined a contact matrix for background contacts [54].

- **Household Contact Matrix** (`household_contact.csv`) – defined a sought contact matrix within households for use in objective functions.

- **Polymods Probabilities** – defined a contact matrix to optionally weight within-household contact rates by age. The specific file to be loaded was specified in the parameter `probsFile`.

- **Exogenous Weights** (`exogenousInf_multipliers.csv`) – defined a set of weights to be optionally applied to the exogenous infection rate, modifying its age distribution.

- **School Age Weights** (`schoolcontact_multipliers.csv`) – defined a set of weights to be optionally applied to in-school contacts, modifying the age distribution.

- **Target Frequency Size** (`Target_freqSize.csv`) – defined a reference for outbreak size and duration for early objective function tests.

- **Target Vaccine Coverage** (`Target_vaccineCoverage.csv`) – defined a reference for vaccine coverage for early objective function tests.

Table 3.3: Disease Configuration Parameters

Parameter	Default Value	Description
diseaseMechanic	DE_NOVO	The disease module to be used, see **Disease Module** in Section 3.2.1
contactRegime	BACKGROUND_-MATRIX	The Contact Regime to be used, discussed under **Directly Specified Parameters** in Section 3.2.3
matrixContactEventInterval	0.041666667	Time interval in days for contact matrix contact events
probOfVaccinationDuring-Pregnancy	0	Probability that a pregnant woman will receive a pertussis vaccination.
cocooningFrac	0	Fraction of pregnant women to which a cocooning intervention is applied.
rateOfTetanus	0	Rate per day at which a risk of tetanus occurs, prompting a prophylactic adult Tdap dose
isUsingHouseholdPolymods	FALSE	Apply a POLYMOD-like [54] age-preference to household contact rates
probsFile	–	Probability input file for applying POLYMOD-like [54] age-dependent contact rates
isUsingExogenousPolymods	TRUE	Apply a POLYMOD-like [54] age-preference to exogenous exposure rates
isUsingSchoolEpsilon	TRUE	Apply a same-age-preference for school contacts
transferImmunityOnly-UponIntervention	TRUE	Transfer maternal immunity only if the mother received vaccination during pregnancy
narrowLL	FALSE	Use narrow log-likelihood for objective functions
dose_blunting_proportion	0	Degree to which vaccination in infants is blunted by their mother having vaccination during pregnancy

Table 3.4: General Configuration Parameters

Parameter	Default Value	Description
drawAgents	FALSE	Should agents be drawn on screen?
drawBins	FALSE	Should population bins be drawn on screen?
isExportingData	TRUE	Should model outputs be exported to files?
isLoggingExposures	FALSE	Should individual exposures be logged to a file (deprecated)?
vaccCoverageUpdateInterval	30	Days between updates of vaccine coverage stats
vaccCoverageAges	[2.0, 2.0, 2.0, 2.0, 7.0, 15.0, 21.0]	Ages for vaccine coverage stats
isHeadless	TRUE	Is the simulation running without a GUI?
cellSizeFactor	21	Ratio of `binWidth` to `cellWidth`
caseReportInfThreshold	0	Infection level threshold below which a case is subject to underreporting. Deprecated.
useUnderreporting	FALSE	Apply underreporting to outputs?
modelRunTimeInYears	50	Model run time
minAgeWaningStats	5	Minimum age for immunity waning stats output
maxAgeWaningStats	10	Maximum age for immunity waning stats output
isUsingSimpleGrid	TRUE	Use Grid data structure?
isUsingOnDemandBGContacts	FALSE	Calculate background contacts as-needed?
burnInPeriodInYears	0	Number of years at the beginning of the model run to exclude from outputs
viewSubClinicalInEpiCurves	TRUE	Display sub-clinical cases in the Epi Curves agent?
viewUnreportedInEpiCurves	TRUE	Display unreported cases in the Epi Curves agent?
useMemoryEfficientLoggers	TRUE	Log more outputs directly to disk?
isSuppressingLargeDatasets	TRUE	Minimize output data stored in memory?
discardThreshold	1.0×10^{-5}	Used in early objective function tests

- **Target Yearly Frequency Size** (`Target_yearFreqSize.csv`) – defined a reference for yearly incidence frequency size for early objective function tests.
- **Target Incidence Frequency CDF** (`Target_IncidFreqCDF.csv`) – defined a reference for incidence frequency cumulative distribution function (CDF) for early objective function tests.
- **Target Age Incidence** (`Target_ageIncidence.csv`) – defined a reference for age incidence distribution for early objective function tests.
- **Age Incidence Probability Weights** (`ageIncidProbWeights.csv`) – defined a set of weights used for early objective function tests.
- **Target Increase in Relative Risk** (`Target_incrRR.csv`) – defined a reference for increase in relative risk for early objective function tests.
- **School Contact Prior Distribution** (`prior_schoolContact.csv`) – defined a prior distribution for school contacts for computation of log-likelihood in early objective function tests.
- **Household Contact Prior Distribution** (`prior_householdContact.csv`) – defined a prior distribution for household contacts for computation of the log-likelihood in early objective function tests.
- **Exogenous Infection Prior Distribution** (`prior_exogenousInfection.csv`) – defined a prior distribution for exogenous infections for computation of the log-likelihood in early objective function tests.

3.2.4 Scenarios

Scenarios represent runs of the model with particular parameters, inputs, outputs and goals. In AnyLogic, the `Experiment` is where the model is set up for a particular run or series of runs, i.e., a scenario. The design of the experiment in the study is discussed below in Section 3.4. A number of scenarios were included in the model, and are summarized in this section.

- `Simulation` represented a basic Graphical User Interface (GUI) run of the model used for testing. Parameters must be set within the AnyLogic editor.
- `Simulation_SimpleGUI` provided a simple setup GUI to select a parameter set file, and then would run the model in GUI mode.
- `TreeTest` was for testing the k-d Tree and Grid data structures, discussed in Chapter 4.
- `Calibration` was for calibration runs with or without a GUI. There are multiple calibration scenarios (using the `Optimization` experiment type in AnyLogic) because selection of which parameters are to be varied, their ranges, and the objective function is specified in the design-time editor associated with the AnyLogic `Experiment`.

- `NoGUI` represented a single run of the model, launched from the command-line without a GUI. Parameters were loaded from a specified file.
- `NoGUI_Calibration_Gen` would run a calibration from the command-line with no GUI. The `Experiment` was specified on the command line, which determined the calibrated parameters and objective function, and fixed parameters are loaded from a file.

3.2.5 Ancillary Structures

GUI Elements

The model could display icons for individual agents and tiles coloured to represent a chosen measure—with population density representing the default measure. In later stages of development, these features were disabled, however, to reduce computational and memory burden when running with large populations.

When running with a GUI, a number of dynamic plots would appear, summarizing select aspects of model state dynamically. As well, the visualizations of the Epi Curves agent are accessible interactively. Figures 3.5 and 3.6 show screen shots of dynamic graphs in the model GUI.

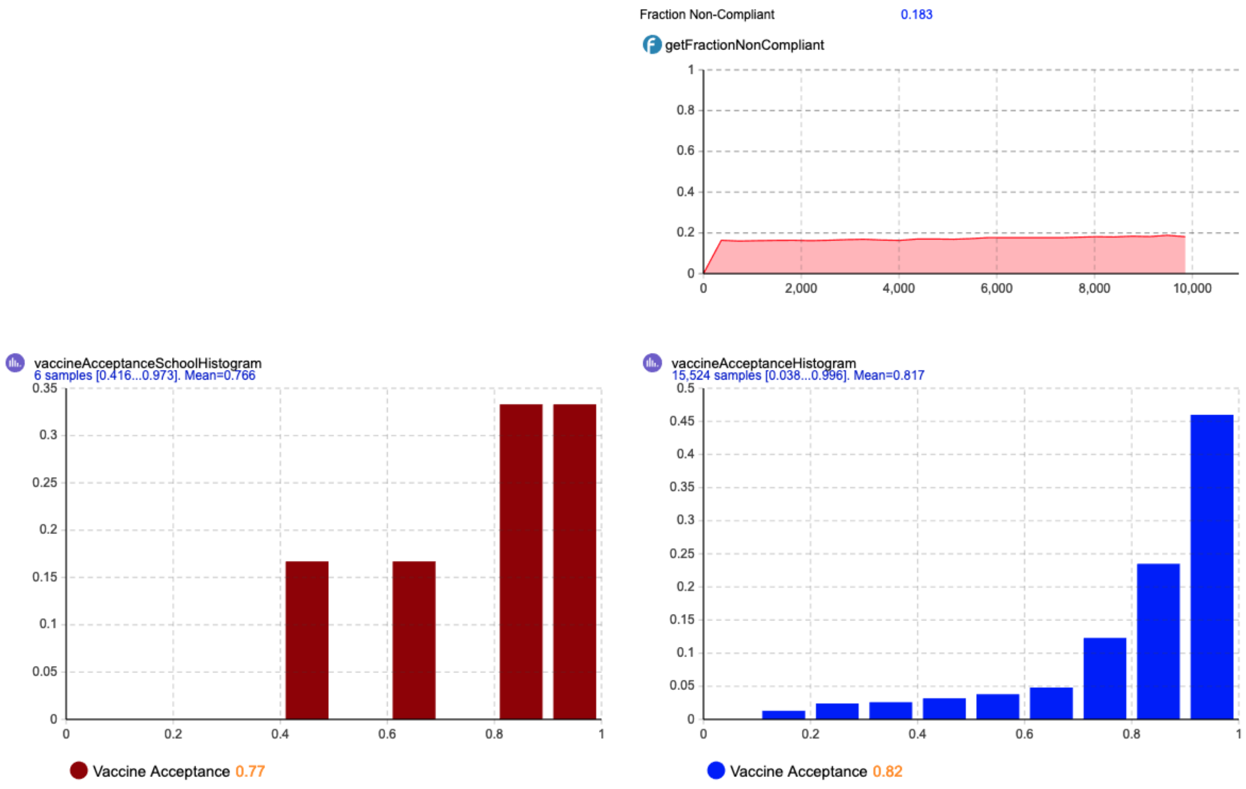
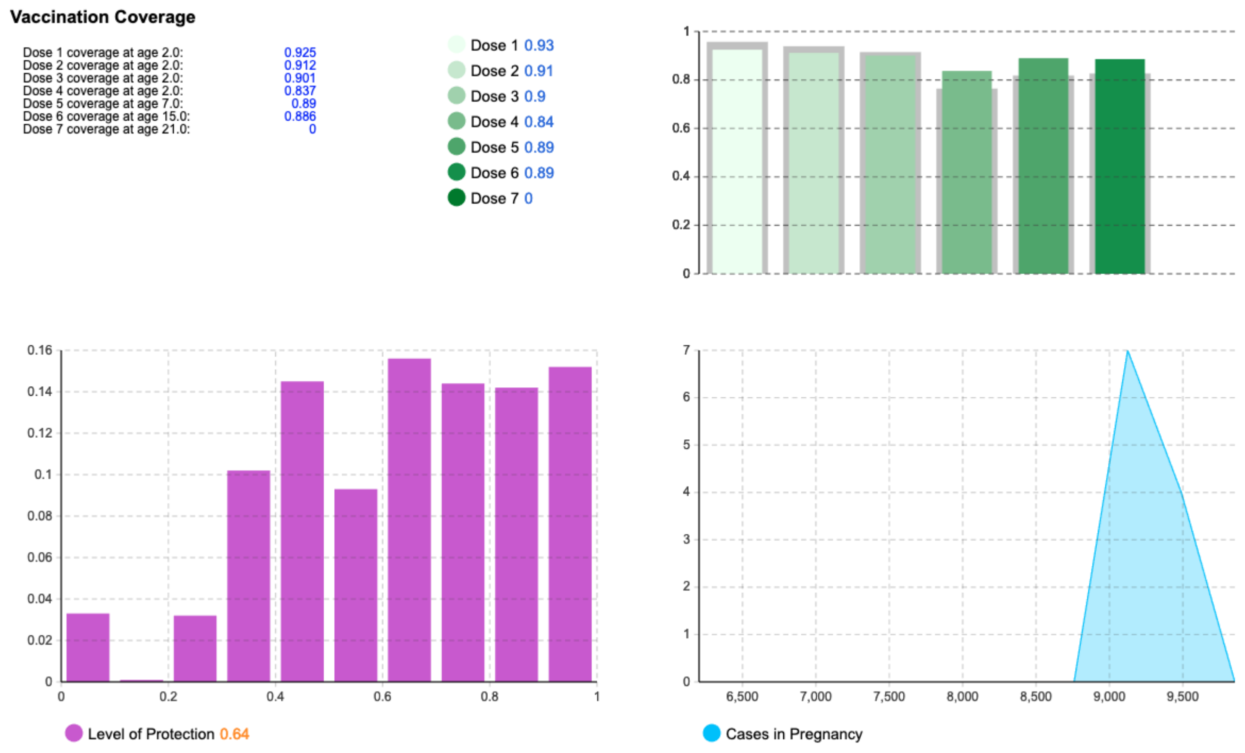

Fraction Non-Compliant 0.183
getFractionNonCompliant
vaccineAcceptanceSchoolHistogram
6 samples [0.416...0.973]. Mean=0.766
Vaccine Acceptance 0.77
vaccineAcceptanceHistogram
15,524 samples [0.038...0.996]. Mean=0.817
Vaccine Acceptance 0.82

Figure 3.5: Dynamic Graphs in Model GUI

To improve performance, the GUI could be suppressed. This was done for the main experiments and automated calibrations; see NoGUI in Section 3.2.4.


Vaccination Coverage

Dose 1 coverage at age 2.0:	0.925
Dose 2 coverage at age 2.0:	0.912
Dose 3 coverage at age 2.0:	0.901
Dose 4 coverage at age 2.0:	0.837
Dose 5 coverage at age 7.0:	0.89
Dose 6 coverage at age 15.0:	0.886
Dose 7 coverage at age 21.0:	0

Figure 3.6: Dynamic Graphs in Model GUI

File Outputs

Output describing model results was saved in a CSV format for post-processing in any data analysis package. This could be disabled, if desired, with appropriate parameter settings. Outputs were changed mid-project to reduce memory use by writing more data to disk without retaining it in memory. The following files represent the "preferred" output set in the current version of the model:

- `E_*_parameterset.csv` – replicated the actual parameter values used in the model run. This file could be fed back into the model as an input parameter set file.
- `E_*_Log_Exposures.csv` – contained a record of every exposure occurring in the model with the following data:
  - **Network** through which the person was exposed, i.e., *Exogenous*, *Household*, *School*, or *Background*
  - **Force of Infection** associated with the exposure
  - **Unique ID** of the exposed person
  - **Age** of the exposed person
  - **Mother's Age at Birth** of the exposed person
  - **Mother had Intervention** (TRUE / FALSE) for the exposed person
  - **Household Size** of the exposed person
  - **School Size** of the exposed person
  - **Protection Level** of the exposed person
  - **Pregnancy Status** of the exposed person
  - **Unique ID** of the exposing person
- `E_*_Log_TimeseriesAnnual.csv` – contains an annual timeseries including the following data:
  - **Population** stratified in one-year age intervals
  - **Population** stratified into age bins specified by the `incidenceAgeGroups` in the model
  - **Fraction Infected** stratified into one-year age intervals between `minAgeWaningStats` and `maxAgeWaningStats`
- `E_*_Log_TimeseriesMonthly.csv` – contains a monthly timeseries including the following data:
  - **Vaccine Coverage** for each of the 7 vaccine doses
  - **Population** stratified in one-year age intervals

  - **Population** stratified into age bins specified by the `incidenceAgeGroups` in the model

- `E_*_Log_Memory.csv` – contains diagnostic run time and memory information
- `E_*_Log_ObjectiveValues.csv` – contains objective function values, even if the run was not a calibration
- `E_*_immunitydistribution.csv` – immunity distribution of the population at the end of the model run; this is a raw table output of the `immunityDistribution` table in the model, a `HistogramData` object

Additional output files could be obtained from the model by setting the `isSuppressingLargeDatasets` parameter to `FALSE`. These can be considered deprecated as well as being redundant to the outputs already mentioned, but are retained for testing purposes. The files are:

- `E_*_agedistribution.csv` – age distribution of the population in one-year age intervals over the last 30 years of the model run in one-year time intervals, representing raw output from a `Histogram2DData` object.
- `E_*_ageofinfectiondistribution.csv` – distribution of age of infection in one-year age intervals over the last 30 years of the model run in one-year time intervals, representing raw output from a `Histogram2DData` object.
- `E_*_CasesByAge.csv` – count of cases in one-year age intervals cumulative over the entire model run excluding the burn-in period, representing raw output from a `HistogramData` object.
- `E_*_casesByAgeDistribution.csv` – distribution of cases in one-year age intervals over the last 30 years of the model run in one-year time intervals, representing raw output from a `Histogram2DData` object.
- `E_*_CasesByMotherAndChildAge.csv` – distribution of cases in one-year age intervals of mother- and child-age cumulative cases over the entire model run excluding the burn-in period, representing raw output from a `Histogram2DData` object.
- `E_*_contactmatrix_houshold.csv` – count of pertussis-spreading contacts in the household venue only in one-year age intervals of exposed person and exposing person cumulative over the entire model run, excluding the burn-in period. This represents raw output from a `Histogram2DData` object.
- `E_*_contactmatrix.csv` – count of pertussis-spreading contacts in one-year age intervals of exposed person and exposing person cumulative over the entire model run, excluding the burn-in period. This represents raw output from a `Histogram2DData` object.

- `E_*_contactRateMatrix_houshold_unmediated.csv` – rate of all contacts in the household venue only in one-year age intervals of exposed person and exposing person cumulative over the entire model run, excluding the burn-in period. This represents modified output from a `Histogram2DData` object.
- `E_*_contactRateMatrix_houshold.csv` – rate of pertussis-spreading contacts in the household venue only in one-year age intervals of exposed person and exposing person cumulative over the entire model run, excluding the burn-in period. This represents modified output from a `Histogram2DData` object.
- `E_*_contactRateMatrix_unmediated.csv` – rate of all contacts in one-year age intervals of exposed person and exposing person cumulative over the entire model run, excluding the burn-in period. This represents modified output from a `Histogram2DData` object.
- `E_*_contactRateMatrix.csv` – rate of pertussis-spreading contacts in one-year age intervals of exposed person and exposing person cumulative over the entire model run, excluding the burn-in period. This represents modified output from a `Histogram2DData` object.
- `E_*_contactsByAgeDistribution.csv` – distribution of contacts in one-year age intervals over the last 30 years of the model run, excluding the burn-in period, in one-year time intervals. This represents raw output from a `Histogram2DData` object.
- `E_*_cumulativeAllAgesIncidence.csv` – annual timeseries of cumulative incidence of pertussis across all ages.
- `E_*_epiCurveTimeInterval1.csv` – incidence timeseries at an interval specified by the `timeInterval1` parameter of the `EpiCurves` agent, stratified by age and reporting status.
- `E_*_epiCurveTimeInterval2.csv` – incidence timeseries at an interval specified by the `timeInterval2` parameter of the `EpiCurves` agent, stratified by age and reporting status.
- `E_*_InterventionCases.csv` – annual timeseries of cases stratified by age and intervention status.
- `E_*_lifeyearsbyage.csv` – sum of life-years lived in one-year age intervals cumulative over the entire model run, excluding the burn-in period. This represents raw output from a `HistogramData` object.
- `E_*_memoryusage.csv` – weekly (model time) timeseries of actual memory usage by the model executable.
- `E_*_transmissionByAgeDistribution.csv` – distribution of pertussis transmissions in one-year age intervals over the last 30 years of the model run, excluding the burn-in period, in one-year time intervals. This represents raw output from a `Histogram2DData` object.
- `E_*_vaccinecoverage.csv` – annual timeseries of vaccine coverage, stratified by dose.
- `E_*_waningstats.csv` – annual timeseries of fraction infected stratified into one-year age intervals between `minAgeWaningStats` and `maxAgeWaningStats`.

3.3 Calibration

The model was initially calibrated "manually" by selecting parameters, running the model, and observing outcomes. Once candidate parameter ranges were identified, two methods of automated calibration were employed, discussed in this section.

3.3.1 Manual Calibration

Calibration of the model was initially undertaken "manually" by iteratively selecting parameter values, running the model, and assessing the model outcomes. In particular, Dr. Doroshenko was able to draw on his experience as a Medical Officer of Health to identify which patterns in Epi Curves were representative of pertussis in Alberta or not.

3.3.2 Automatic Calibration

Vaccine Coverage

Vaccine coverage was calibrated using an AnyLogic `Optimization` experiment type. Parameters were varied in continuous ranges, as follows, for 500 realizations. The objective function was the sum square difference between the mean vaccine coverage over the model run and the reference value, across doses. Reference values were for Alberta, as obtained from the Interactive Health Data Application (IHDA) [15].

- `schoolVaccineAcceptance_p` from 1×10^{-6} to 3.0
- `schoolVaccineAcceptance_q` from 1×10^{-6} to 3.0
- `probOfMissingDose5and6` from 0.1 to 0.9

This calibration was undertaken separately because vaccine coverage was deemed to be independent of the disease dynamics.

Disease Behaviour

Automatic calibration of disease incidence was performed by taking a Latin Hypercube sample, discussed in Section 2.4, of the varied parameters over ranges that were informed by the manual calibration. Parameter set files were generated for each of the $1,000$ runs in the ensemble, and these were used to initialize the model in `NoGUI` mode. The following parameters were varied:

- `exogenousInfectionRate` from 0 to 0.3
- `backgroundContactRate` from 0 to 0.2
- `householdContactRate` from 0 to 1.0

- `schoolContactRate` from 0 to 2.0

Outputs were processed to generate likelihood profiles, which were used to estimate parameter values, shown in Figure 3.7. A likelihood profile plots the log-likelihood of the model over a range of a given parameter, allowing selection of the parameter value giving the maximum likelihood [57]. The likelihood value for a given parameter value was obtained by examining the likelihood for each run within a window around a given value of the focal parameter, taking the highest likelihood value, and repeating while sliding that window along the range of values for the focal parameter. Cubic spline smoothing was then applied to the likelihood profile. The overall log-likelihood for the model for a given run was a weighted sum of the individual likelihood measures for each output. Because of the way model outputs were done, the likelihood could be analyzed in post-processing without a need to re-run the model. The likelihood measure was based on comparing the following model outputs to reference data for Alberta [15].

- Mean all-ages incidence per year
- Cumulative Density Function (CDF) of the yearly all-ages incidence distribution
- Auto-correlation function of all-ages incidence
- Distribution of total cases by age

In examining the likelihood profiles in Figure 3.7, we can see clear maxima, within the tested ranges. The spike at 0 for `householdContactRate` and `backgroundContactRate` can be explained as an artifact of the cubic spline smoothing. It is proposed now to run a second ensemble of 100 simulations while fixing the `exogenousInfectionRate` and `schoolContactRate` at the peak values from Figure 3.7 and analyzing in the same manner; see Section 3.5.

3.4 Experiment Design

3.4.1 Focus

Infants, particularly those under 6 months of age, are the main risk group for pertussis [9]. Current vaccination strategies are known to be insufficient to establish herd immunity [13], and increasing vaccine coverage in older age groups appears to yield only marginal improvements in outcomes for infants [34]. It therefore becomes attractive to examine novel strategies that focus on the primary risk group.

The focus of this study was to examine the potential of administering pertussis vaccine to mothers in the third trimester of pregnancy as a potential means to increase protection for infants in the first 6 months after birth. This is motivated by the hypothesis that such vaccination will result in increased transfer of maternal antibodies to the infant, improving protection. A counter-hypothesis suggests that the additional antibodies may have a “blunting” effect on early doses of the vaccine administered to the child, by destroying

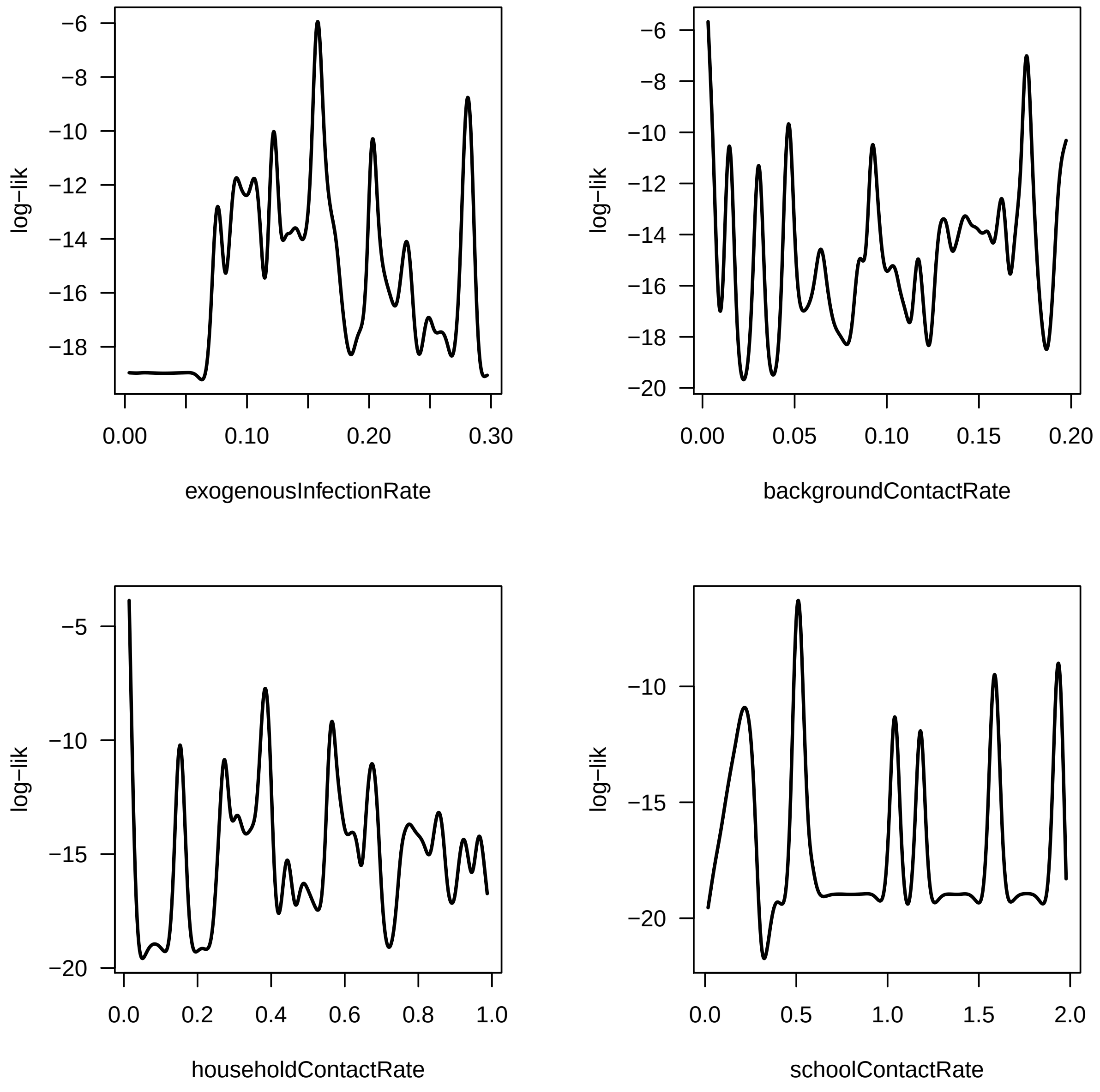


Figure 3.7: Likelihood Profiles

the vaccine antigens before the child's immune system can mount a response and establish its own immune memory [58].

3.4.2 Population

Size

A population size of $500,000$ was selected, as it was deemed to be representative of the health regions in Alberta, excepting Edmonton and Calgary, and was also known to be tractable in an ABM built with AnyLogic based on previous investigations undertaken in CEPHIL employing agent-based modeling. This represents a nominal population size, as the model implements an "open population" having birth and death rates informed by demographics, and thus the exact population within the model fluctuates over time.

Density Distribution

The population density distribution is loaded from a file, as mentioned under **Inputs** in Section 3.2.3. For initial work, the model used a series of files with uniform distributions having total populations ranging from $10,000$ to $1,000,000$. File-based input was undertaken to allow for testing of arbitrary population density distributions, so as to explore sensitivity to spatial effects, or to approximate the population distribution in a real-world region.

This particular feature was requested early in the group modeling process by Dr. Doroshenko, prompted by discussions that occurred during the peer review process for the Doroshenko et al. 2016 paper [5]. It was deemed that the ability to specify the population density distribution without altering model programming would open up possibilities for for sensitivity analyses and experiment variations that may help alleviate reviewer concerns that were raised in regards to the previous model.

The main experiment employed a uniform population distribution, however, leaving examination of differing population distributions for future work.

3.4.3 Setup

Two scenarios were considered as interventions in the main experiment: Mothers receiving Tdap vaccination during the third trimester of pregnancy at vaccine coverage (VC) levels of 50% and 75%. In the preliminary main experiment, a series of 10 simulations were run for each scenario and the scenarios were evaluated according to the preventable fraction in the population (PFP) and the vaccine effectiveness (VE). The full main experiment was proposed to consist of 100 simulations and will be run once calibration has been completed.

Vaccine effectiveness is a measure of how a vaccine actually reduces disease in a population. VE of 90% would indicate a 90% reduction of the number of cases of the disease in the vaccinated population as compared to the unvaccinated population [59]. VE is calculated as $1 - OR$, where OR represents "the odds ratio for developing the disease despite vaccination" [60]. Vaccine effectiveness refers to a measure undertaken in "field" conditions, i.e., non-ideal conditions. In contrast, vaccine *efficacy* represents a measure undertaken under ideal conditions such as a clinical trial. Both are calculated in the same manner [59] given these contexts.

An odds ratio (OR) "represents the odds of [an] outcome occurring given a particular exposure compared to the odds of [the same] outcome occurring in absence of that exposure" [61]. In this case, the *exposure* is the vaccination and the *outcome* is pertussis disease. An OR of less than 1 indicates that the exposure (vaccination) is associated with lower odds of the outcome (disease) [61].

The PFP estimates what proportion of cases could be prevented by applying the intervention to the entire population. PFP is calculated as in Equation 3.1, where I_p represents disease incidence in the population and I_e represents incidence in those "exposed" to the vaccination [62]. Letting I_u represent the disease incidence in the unvaccinated population, I_v represent the disease incidence in the vaccinated population, and vc represent the vaccine coverage we obtain the form shown in Equation 3.2, used in the analysis.

$$PFP = \frac{I_p - I_e}{I_p} \tag{3.1}$$

$$PFP = 1 - \left(\frac{I_u + I_v}{I_u}\right)\left(\frac{1}{1 - vc}\right) \tag{3.2}$$

3.5 Results

As of this writing, the final main experiment has not been run as the calibration has not been completed. Table 3.5 show the results for the preliminary main experiment assuming 50% vaccine coverage for the maternal immunization, Table 3.6 shows the results assuming 75% coverage for the intervention. Results suggest that the intervention can reduce the number of infant cases of pertussis, and that this reduction is greatest in the $0-2$ month age group. Higher maternal vaccine coverage results in greater reductions.

Table 3.5: Preliminary Results for Main Experiment with Vaccine Coverage of 50%

Age Group	VC	PFP	VE
$0-2$ months	0.50	0.5	1
$2-4$ months	0.50	0.39	0.78
$4-6$ months	0.50	0.45	0.91
$6-12$ months	0.50	0.38	0.76
Overall	0.50	0.42	0.85

Table 3.6: Preliminary Results for Main Experiment with Vaccine Coverage of 75%

Age Group	VC	PFP	VE
$0-2$ months	0.75	0.74	0.99
$2-4$ months	0.75	0.64	0.85
$4-6$ months	0.75	0.67	0.89
$6-12$ months	0.75	0.64	0.86
Overall	0.75	0.67	0.89

3.6 Sensitivity Analyses

As of this writing, the final sensitivity analysis has not been run, pending calibration, preliminary results are presented here. One-way sensitivity analyses were conducted to examine the influence on model outcomes of the rate at which immunity wanes and immunological blunting. Results of these analyses are shown in Table 3.7. This suggests that model outputs are sensitive to values of these parameters.

Table 3.7: Preliminary Results for Sensitivity Analyses

Scenario	PFP	VE
75% Coverage	0.67	0.89
75% Coverage, 50% passive immunity waning	0.67	0.89
75% Coverage, 25% passive immunity waning	0.73	0.98
75% Coverage, 10% immune blunting	0.62	0.83
75% Coverage, 50% immune blunting	0.55	0.74

3.7 Discussion

Preliminary results suggest that, given model assumptions, an intervention providing vaccination to mothers during pregnancy would result in a reduction in pertussis incidence in infants, with the greatest benefit being conferred to the $0-2$ month age group. Results are sensitive to assumptions about waning of immunity and immunological blunting. Whether the Government of Alberta, or other jurisdictions, would be recommended to proceed with such an intervention would be the subject of further analysis. While many of the considerations to be weighed in such a decision lie outside the scope of this thesis, this model, with modifications, could be a valuable tool in informing such decisions.

3.8 Division of Work

This section will outline the share of work on this model between Dr. Karsten Hempel (KH) and Mr. Wade McDonald (WM). It can be understood that Drs. Doroshenko and Osgood acted in both supervisory and expert capacity across the entirety of this project.

WM was fully responsible for implementation of basic model structure, data structures, modular disease mechanic interface, model GUI, `EpiCurve` agent, `NoGUI` setup, file output, and batch runs. WM and KH shared responsibility for file input implementation, model initialization, `Person` agent, post-processing of outputs, and vaccine coverage calibration. KH was fully responsible for pre-processing of inputs, disease modules, disease behaviour calibration, main experiment design, and sensitivity analyses.

4 Data Structures

To supplement AnyLogic's built-in range search function, which has been found to be problematic when scaling to large numbers of agents, a Grid structure discussed in Section 2.3.2, and k-d Tree discussed in Section 2.3.3 were implemented in Java as means to improve range-search performance.

4.1 k-d Tree

4.1.1 Existing Libraries

The k-d Tree, outlined in Section 2.3.3, is an established data structure. In such a situation, one would typically look to established and robust libraries for an implementation. The first avenue of investigation led to the Computational Geometry Algorithms Library (CGAL) [63]. This library, developed in C++, is well-established and implements many geometric algorithms and supporting data structures, including a k-d Tree.

The pertussis model outlined in Chapter 3 is built using AnyLogic software, which is Java-based. In order to make use of CGAL from within AnyLogic, use of the Java Native Interface (JNI) [64] was considered. JNI allows Java programs to call native libraries, including those written in C++.

Two major disadvantages are associated with use of JNI. Native libraries are platform-specific, so must be compiled for the platform being used. Further, those libraries typically have dependencies, which are also platform-specific. This opens up a situation where it may be reasonably straightforward to get a program incorporating JNI running on a specific machine, but unforeseen problems may crop up when moving to another machine.

In developing the pertussis model, machines running all major operating systems were in use, which is no issue when working in pure Java. Adding JNI to the project would require native libraries (i.e., CGAL) and all of their dependencies be compiled and bundled for all operating systems. It was decided to search for a Java implementation of a k-d Tree, instead.

Searching online revealed that there are no established libraries in Java implementing a k-d Tree. There exist some unverified implementations online, however. It was decided that implementing a k-d Tree in Java would be a valuable learning experience and produce a beneficial asset for use in future projects within CEPHIL.

4.1.2 Implementation

A 2-dimensional k-d Tree data structure was implemented in Java. Additionally, the Grid structure, described in Section 2.3.2, was re-implemented in a way that it could easily be used interchangeably with the k-d Tree. A number of support classes and interfaces were also implemented, described below.

KDPoint and DoublePoint

A Java interface, `KDPoint`, is defined to represent a k-dimensional coordinate key of a node in the k-d Tree. For generality, the coordinates can be represented by any `Comparable` type. The `DoublePoint` class is then defined, implementing the `KDPoint` interface for coordinates of type `Double`. Refer to code listings A.1 and A.2 in Appendix A.

HasDoublePointCoordinate

A Java interface, `HasDoublePointCoordinate`, is defined that specifies an object to have a method, `get-Coordinates()`, that returns the object's coordinates as a `DoublePoint`.

RangeSearchGrid2D

A Java interface, `RangeSearchGrid2D`, is defined that specifies *add* and *remove* operations and several iterators. `rangeSearchIterator` represents a rectangular, axis-aligned range search and `radiusSearchIterator` represents a circular range search specified by a centre point and radius. Specifying this interface allows the `SimpleRangeSearchTree2D` and `SimpleRangeSearchGrid2D`, outlined below, to operate interchangeably.

The search space is divided into square bins, mainly for the benefit of the Grid structure. `binIterator` iterates through the records associated with a single bin within the space.

SimpleRangeSearchTree2D

`SimpleRangeSearchTree2D` is an implementation of the `RangeSearchGrid2D` interface using a 2-dimensional k-d Tree.

A `Simple2DNode` is defined that stores the discriminator, coordinates, a list of values, and references to child nodes. The traditional k-d Tree, as described by Samet [40], stores a single value in each node and, on insertion of a new value with coordinates that already exist in the tree, replaces the old value with the new. In this application, however, there exist multiple records with identical coordinates. To accommodate this, each node can store multiple values, in a list, all sharing the same coordinates.

The `SimpleRangeSearchTree2D` provides an `init()` method to initialize the tree in balance from a collection of records. Since the structure of a k-d Tree depends on the order of insertion of records, it is possible, if random insertion is used [41], for the tree to be initialized out-of-balance, in which case access will approach the worst case of $O(n)$. Initialization is done, as described by Bentley [41], by sorting the data at each level of the tree and inserting the median element (according to the appropriate discriminator) at the current node. All points below the median are recursively inserted in the same manner at the low child and all points above the median are recursively inserted at the high child. This method could also be used to re-create the tree, should it become unbalanced.

Individual values may be added or removed. Removal of the last item in a node does not remove the node itself; deletion of nodes is expensive [40] and, in this particular application, it is likely that new records will be inserted at the same coordinates.

In this implementation, the tree does not monitor the degree to which it is balanced or automatically re-balance itself; it is up to the programmer to re-balance the tree by calling `init()`. It should further be pointed out that the "OPTIMIZE" algorithm, as Bentley [41] named it, does not modify the tree in-place to restore balance, it simply generates a new optimal tree.

A `RangeSearchIterator` is defined to perform a rectangular, axis-aligned range search and a `Radius-SearchIterator` performs a circular range search. See code listing A.5 in Appendix A.

4.2 Grid Structure

4.2.1 Implementation

The Java class `SimpleRangeSearchGrid2D` implements the Grid structure described in Section 2.3.2 in compliance with the `RangeSearchGrid2D` interface. This allows it to be used interchangeably with the `SimpleRangeSearchTree2D`. See code listing A.6 in Appendix A.

4.3 Library

The implementations were bundled as a JAR file, which can be included as a dependency in an AnyLogic model or added to the `classpath` of a general-purpose Java application. Any object that is to be stored in the `SimpleRangeSearchGrid2D` or `SimpleRangeSearchTree2D` must implement the `HasDoublePointCoordinate` interface.

4.4 Parallelism and Concurrency Considerations

Since the emergence of diverse commercial microprocessors in the 1970s, exponential growth in the number of transistors per chip, dubbed "Moore's Law," has allowed rapid growth in computing power. These gains came mainly as a result of increased clock-speeds until shortly after the turn of the millennium, when heat dissipation challenges put an end to this line of development. Clock-speeds have since stagnated, and performance increases have subsequently come primarily through adding multiple cores to microprocessors. This change addresses a problem at the level of hardware design, but creates new challenges for software programmers [65]. In light of this reality, this section will discuss parallelism and concurrency as they relate to this work.

4.4.1 Definitions

Parallelism and concurrency are distinct problems but often occur together, and both have become greater concerns in programming with the near-ubiquity of multi-core processors.

Parallelism

Parallelism is achieved by "delegating different parts of [a] computation to different processors that execute at the same time" [66]. Some tasks are more amenable to parallelization than others; Amdahl [67] argued that there is a maximum possible speedup based on the fraction of the task that could be broken into subtasks. Gustafson [68] suggested that Amdahl was needlessly pessimistic and proposed an alternative formula. These formulations came to be known as *Amdahl's Law* and *Gustafson's Law*, respectively. Regardless, overhead in setting up parallel subtasks and the fact that some subtasks must be performed serially combine to place limits on performance gains from arbitrary degrees of parallelization. Purely parallel applications can often be straightforward to specify and debug, when compared to those involving concurrency.

Concurrency

Concurrency, on the other hand, can be difficult to define formally in a way that is also easy to understand. Raynal [69] describes a concurrent program as being "made up of several entities (processes, peers, sensors, nodes, etc.) that cooperate to a common goal. This cooperation is made possible thanks to objects shared by the entities. These objects are called *concurrent objects*." Alternatively, Marlow [66] describes a concurrent program as having "multiple *threads of control*" that "execute 'at the same time'; that is, the user sees their effects interleaved." Marlow goes on to state that "concurrent programming is concerned with structuring a program that needs to interact with multiple independent external agents (for example, the user, a database server, and some external clients)."

Concurrency can be considered a problem of coordinating access to shared resources. If multiple threads are allowed to access an object, often a data structure, without restriction, it eventually leads to data corruption [70], possibly due to *race conditions*. This type of problem can be eliminated through *mutual exclusion*. A number of programming methods have been developed over the years to mitigate this pitfall [71]; major techniques available in Java are discussed in Section 4.4.2, but misapplication of these methods can lead to other difficult-to-diagnose problems, including *deadlocks*, and *resource starvation*. These terms are defined as:

- **Race Condition** – occurs when "multiple threads are accessing a shared resource and the results depend on thread scheduling" [72]. Results of a program suffering from a race condition "may vary from run to run" [72].

- **Mutual Exclusion** – controls access to a shared object by allowing one thread exclusive access to that object and waiting until that thread completes its operations before allowing the next thread access [72].

- **Deadlock** – "occurs when a group of two or more threads cannot make any progress, since each is waiting for a resource held by one of the others" [72]. This is a result of incorrect implementation of mutual exclusion.

- **Resource Starvation** – refers to situations where a thread cannot gain sufficient access to a shared resource in order to complete its job [70, 73]. This could mean that the starved thread is shut out completely, or that it gets some access but not enough and ends up taking "too long" to complete its work. Starvation may either occur consistently or only when the system is under heavy load.

Debugging concurrency problems is difficult because the symptoms can be as vague as the program seeming to take too long to run or appearing to hang. Furthermore, symptoms may occur spuriously and be difficult to reproduce.

4.4.2 Java

Parallelism

The basic Java programming language and its standard library feature a number of parallel computing Application Programming Interfaces (APIs). Threading has been available since version 1.0 [74]. The `Thread` class provides the necessary functionality but can require the programmer to concern themselves with low-level management of the threads. The `Executor` interface was introduced in Java version 1.5 [75] and the `ForkJoinPool` was introduced in Java 7 (version 1.7) [76]. The `ForkJoinPool` implements the `Executor` interface. `ForkJoinPool` reduces the amount of low-level thread management that needs to be done on the part of the programmer. Tasks must still, however, be set up as a `ForkJoinTask` [77, 78]. Parallel streams were introduced in Java 8 (version 1.8). Parallel streams operate in parallel without the programmer having to do any low-level management of threads, and with syntax identical to that of sequential streams [79].

Concurrency

Java provides a number of facilities for managing concurrency issues [70]:

- **Synchronized Methods** – applying the `synchronized` keyword to a method definition establishes the restriction that only one `synchronized` method within the enclosing object can be active at a time.

- **Synchronized Statements** – define a block of code for which a lock must be acquired before proceeding and released after completion. The object providing the intrinsic lock is specified.

- **Atomic Access** – using the `volatile` keyword for variable declaration enforces atomic access for that variable. This is more efficient than synchronization but may still allow some errors due to concurrency issues.

- **Guarded Blocks** – allow threads to coordinate their activity around particular conditions by employing the `wait()` and `notifyAll()` methods.

- **Immutable Objects** – immutable objects cannot be changed after construction and therefore "cannot be corrupted by thread interference or observed in an inconsistent state" [70].

- **High Level Concurrency Objects** – Java also provides a number of high-level objects supporting concurrency, including `Executor` and `ForkJoinPool`, discussed under **Parallelism**, above, and Concurrent Collections, which implement data structures supporting concurrent access.

4.4.3 AnyLogic

Built-In Features

This work is implemented using AnyLogic 8.4.0, which is based on Java 9. AnyLogic is parallel between realizations but does not support concurrency at the level of a single realization. Model logic in a single-run `Experiment` runs in a single thread. A multi-run `Experiment`, such as an `Optimization`, can distribute the realizations to be run across multiple threads. In this case, each thread runs a separate instance of the model, and there are no resources shared between these instances. The basic unit of parallelization is one realization (instance) of the model and, at this level, it would qualify as "embarassingly parallel".

For multi-run experiments, AnyLogic has a setting to "Allow parallel evaluations". If enabled—the default—the experiment will run in a number of threads equal to `Runtime.availableProcessors()`; if the setting is disabled, it will run in a single thread. `Runtime.availableProcessors()` returns the number of logical processors available to the Java Virtual Machine (JVM); normally, this is equal to the number of virtual cores on the system, so a 4-core processor with hyperthreading enabled would have 8 logical processors. This number can be reduced at runtime by setting processor affinity in the OS, if supported. On Windows, processor affinity can be set on the command-line using the `start` command with the `/affinity` switch [80]. On GNU/Linux, the `taskset` command sets processor affinity [81]. macOS exposes affinity settings to the user neither through the command-line nor GUI [82].

Possible User Additions

AnyLogic allows arbitrary Java code within models, so one could conceivably add additional parallelism to a model through use of the features discussed in Section 4.4.2. Parallel streams seem particularly amenable to such an endeavour, as many AnyLogic objects, e.g., populations of agents, can be operated on as a stream.

This may benefit single-run experiments, but if the model is to be further parallelized using a multi-run experiment the performance gains may be moot.

Another potential application of parallel programming lies in developing a Custom Experiment. The Custom Experiment feature in AnyLogic allows the programmer to create their own experiment type using free-form Java code. The example shown in Listing 4.1 demonstrates use of a `ForkJoinPool` to create a multi-run Custom Experiment in AnyLogic [83]. While this may seem redundant to existing features accessible in the AnyLogic editor, it would allow additional control, such as specifying a number of threads to be used other than 1 or `Runtime.availableProcessors()`.

```
ForkJoinPool pool = new ForkJoinPool();

List<Callable<ParameterSet>> tasks = new ArrayList<>();
tasks.add( () -> new CalibrationPable(this).simulate(1D, 2D, 0.0, 0.5, "opt1") );
tasks.add( () -> new CalibrationPable(this).simulate(2D, 3D, 0.0, 0.5, "opt2") );
tasks.add( () -> new CalibrationPable(this).simulate(3D, 4D, 0.0, 0.5, "opt3") );

List<Future<ParameterSet>> futureResults = pool.invokeAll(tasks);

List<ParameterSet> res = futureResults.stream().parallel().map(
  f -> {
    try {
      return f.get();
    } catch (InterruptedException e) {
      e.printStackTrace();
    } catch (ExecutionException e) {
      e.printStackTrace();
    }
    return null;
  }).collect(Collectors.toList());


System.out.println("----- Best Parameter Sets in each Subspace -------");

res.forEach(
  r -> {
    System.out.println(r);
  }
);

System.out.println("----- Parallel Optimization Finished -------");
```

Listing 4.1: Use of ForkJoinPool in an AnyLogic Custom Experiment [83]

4.4.4 Implementations in this Work

As-Submitted

The AnyLogic model developed for this work uses only built-in parallelism. The data structures implemented for this work do not include any parallel or concurrent features; this was a design decision, given that the immediate intended use of these data structures was to enhance an AnyLogic model.

Potential Additions

Notwithstanding the previous statement, the data structures implemented here are not bound to AnyLogic by any dependencies and could potentially be used in general-purpose Java programs. If this is to be done, it may be advantageous to implement some of the features discussed in Section 4.4.2. Following are some comments in that vein.

Performing a range search of both the `SimpleRangeSearchGrid2D` and `SimpleRangeSearchTree2D` require making many pair-wise distance computations, the results of which are independent. No modifications are made to the data structures in this operation. This process could reasonably be parallelized, possibly using parallel streams, without introducing concurrency issues. Additionally, the basic search of the binary tree structure underpinning the `SimpleRangeSearchTree2D` is recursive and is amenable to parallelization using a `ForkJoinPool`, again without introducing concurrency issues.

Implementing concurrency support, or *thread safety*, as it is often called, in the `SimpleRangeSearchGrid-2D` and `SimpleRangeSearchTree2D` would allow them to be utilized in multithreaded Java programs. This could be accomplished, broadly, in one of two ways: Immutability or applying the other basic tools discussed in Section 4.4.2 to implement appropriate mutual exclusion that would allow concurrent access to a mutable data structure.

Use of immutable objects is a strategy that is well-known to ensure thread safety without the need to rely on potentially difficult-to-implement mutual exclusion techniques. There are performance tradeoffs involved, since some operations on the object may create a duplicate that reflects the requested change, leaving the original unmodified, but Oracle suggests that, in Java, these are less severe than many programmers tend to think [70]. The second alternative, implementing thread-safe, mutable versions of the data structures, is more daunting in terms of its scope. It would be worthwhile, if pursuing this route, to fully implement the interfaces necessary to make these data structures peers of the Concurrent Collections data structures that form part of the Java standard library [84].

5 Performance Evaluation

5.1 Objectives

The default AnyLogic method for storing a network involves storing an adjacency list for each agent in the network. In general, $O(e)$, where e is the number of edges in the network graph, is the best one can achieve for storing a network. For the current modeling project, however, a particular network structure is being employed—a distance-based network. In a distance-based network, each node is connected to every other node within a specified radius of itself. Furthermore, the timing of the built-in AnyLogic method for initializing a distance-based network is an always-worst-case $O(n)$, where n is the number of agents in the network. As part of this project, data structures were developed with a goal of improving performance of AnyLogic models with large populations using distance-based networks. This chapter discusses performance testing applied to new and existing data structures.

It was desired that one instance of the pertussis model with 500,000 agents be operable on a 16 GB system and that run-time be minimized subject to that constraint. Ideally, a run-time less than 8 hours in duration would allow for overnight runs.

5.2 Testing

5.2.1 Test Harness

A simple SEIRS ABM was created in AnyLogic to serve as the test bed for evaluating the computational burden for network implementions that were considered. The model implemented a distance-based network within a space that scaled with population size to maintain a specified population density. The network implementations were as follows:

- `BUILTIN_DISTANCE_BASED` - AnyLogic's built-in distance-based network
- `SIMPLE_GRID` - the Grid structure described in Sections 2.3.2 and 4.2.1
- `ON_DEMAND_BUILTIN` - AnyLogic's built-in distance-based network but only computing connections as needed
- `ON_DEMAND_SIMPLE_GRID` - Using AnyLogic's built-in `connections` object, but indexing agents using the Grid; connections are populated as needed

- KD_TREE - Using the k-d Tree to compute range searches

AnyLogic Parameter Variation Experiments were set up to compare run time and memory usage for the model while varying population size, connection range, and the range search algorithm used. NoGUI versions of the experiments were created to run remotely.

Deliberately omitted were many features of the main pertussis model, including multiple contact venues, complex disease logic, GUI elements, and data logging. The purpose was to establish an application benchmark for the testing of range search algorithms within an ABM while minimizing model structures that may impact run time and memory usage orthogonally to the choice of algorithm.

5.2.2 Test Systems

The following systems were used in testing:

- Dell-Desk
  - Dell XPS 8930
  - 3.2 GHz Intel Core i7 (6-core / 12-thread)
  - 32 GB RAM
  - Intel UHD Graphics 630
- Homebuild
  - Home-Built PC
  - 3.7 GHz Intel Core i7 (6-core / 12-thread)
  - 16 GB RAM
  - NVIDIA GeForce GTX 1080 Graphics

5.2.3 Grid Bin Width

Since the bin width of the Grid structure needs to be optimized for best performance, an AnyLogic Optimization Experiment was created to determine the optimum bin width. The objective function was set to minimize total run time of the model. Figure 5.1 shows run times for varying bin widths with a connection range $r = 50$ units. Local minima can be seen at bin width $b = r = 50$, $b = \frac{1}{2}r = 25$, $b = \frac{1}{3}r \approx 16.7$, and $b = \frac{1}{4}r \approx 12.5$.

Figure 5.1 suggests an optimal bin width, b, of 25 units given a connection range, r, of 50 units. It makes sense that the optimal bin width would be an integer fraction of the connection range if we consider an agent, A, with $r << b$ and gradually grow r. Initially, a range search will only need to consider the bin containing A. As r becomes larger and approaches b, additional bins will be added to the search but most of their agents

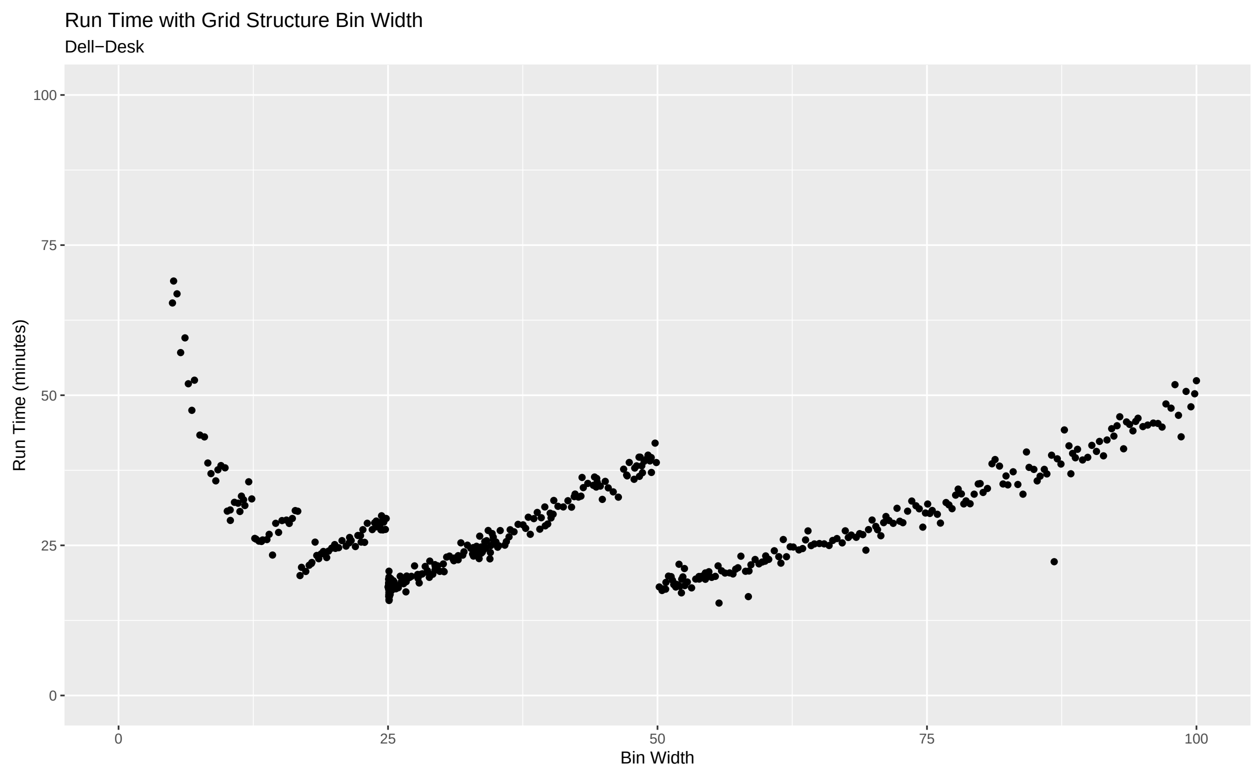


Figure 5.1: Optimization of Bin Width for Grid Structure

will not initially be search hits. As r continues to grow, more of the agents in the newly added bins will be positive search results until r approaches $2b$ and another step change occurs. So, with the caveats mentioned in Section 5.3, a bin width of $b = \frac{1}{2}r$ will be employed for the remainder of this discussion.

5.2.4 Testing Procedure

The model serving as the test harness, described in Section 5.2.1 above, was run with varying parameters in single-threaded mode. Elapsed real time and memory usage, as obtained from the Java runtime with `Runtime.totalMemory() - Runtime.freeMemory()`, were logged from within the model. Competing user processes were closed, where possible.

5.3 Results

It must be emphasized that the results discussed in this section are empirical and cannot be construed as proofs of correctness or extrapolated beyond the ranges of parameters tested. Nevertheless, they can be viewed as representing the relative performance scaling of the tested data structures in a typical use case.

5.3.1 Run Time Scaling with Agent Count

Figure 5.2 shows, on a linear scale, the run time scaling of the test model with agent count for the range search algorithm implementations outlined in Section 5.2.1. Figure 5.3 shows the same on a log-log scale. Here, the run time represents the time to run the actual simulation, minus any initialization time for the model executable. From Figure 5.2, we can see that the `ON_DEMAND_BUILTIN` has the worst absolute time for large populations as well as the worst scaling of the methods tested. Using a log-log scale, however, in Figure 5.3, we can see that all methods except `ON_DEMAND_BUILTIN` exhibit similar scaling with the `BUILTIN_DISTANCE_BASED` and `KD_TREE` having worse absolute times for all agent counts tested. `ON_DEMAND_BUILTIN` has the worst scaling of all methods tested.

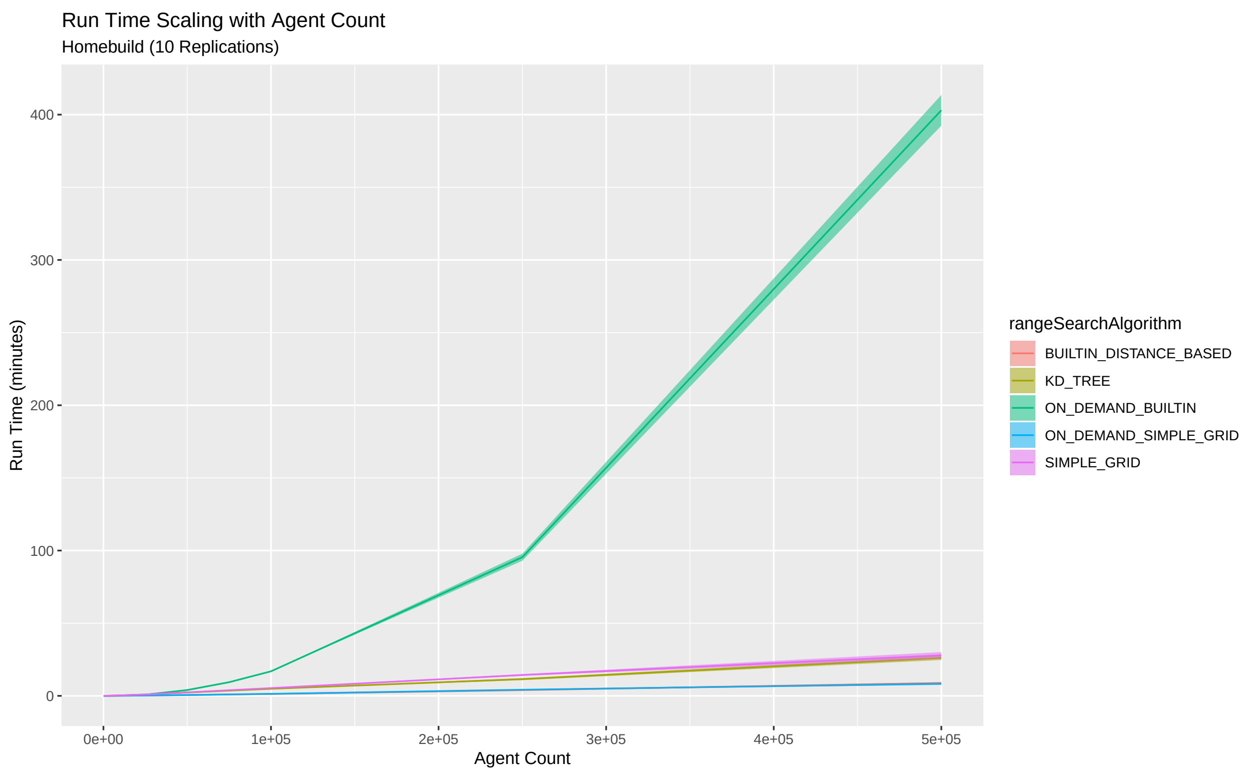


Figure 5.2: Run Time Scaling with Agent Count

Considering Figure 5.4, on a log-log scale, we can see that the `BUILTIN_DISTANCE_BASED` stands out from the other methods in having a substantial initialization time that scales super-linearly. Theoretically, this is $O(n^2)$, as it must compute the distance from each agent to each other agent, as discussed in Section 2.3. The initialization time is nearly 200 minutes for the case of 500,000 agents. In addition to being time-consuming, this poses a user experience issue, as the model appears to do nothing during the initialization, so a person monitoring the model may be confused as to whether the model is operating, or has crashed or hung. This is one of the issues that initially prompted exploration of alternative range search algorithms and data structures.

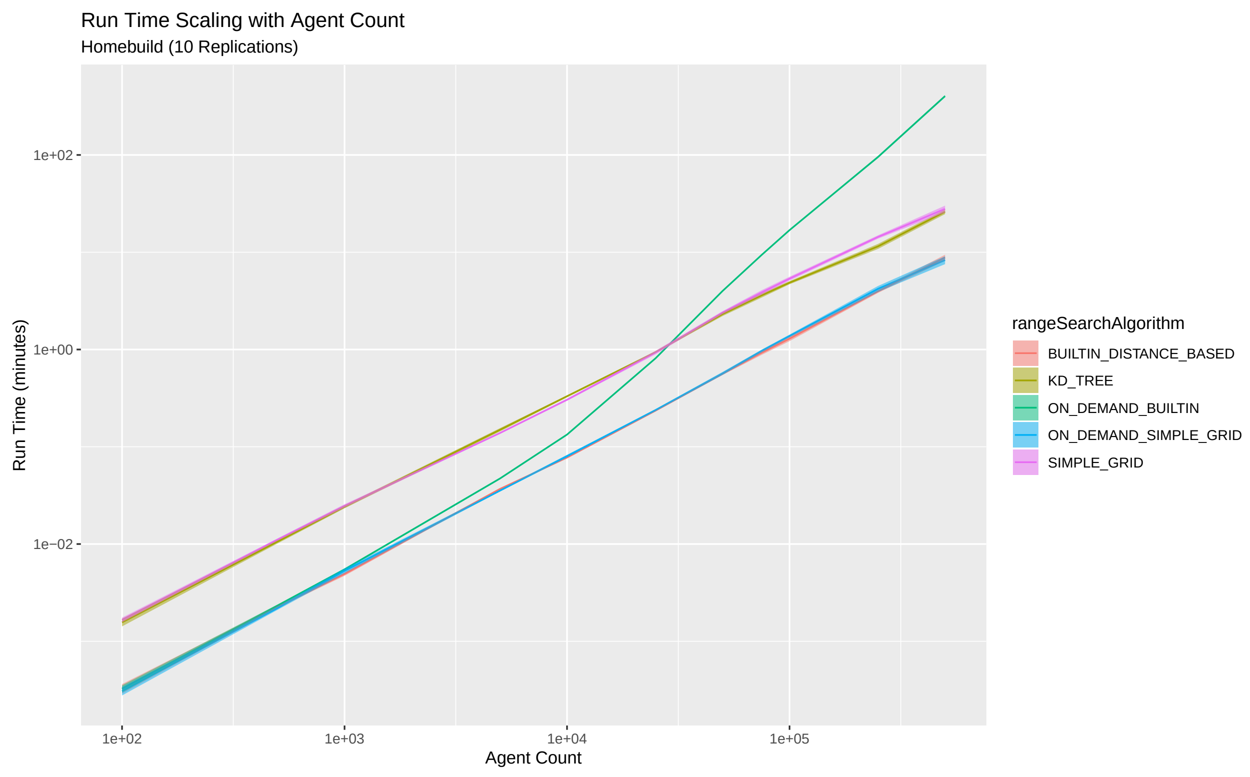


Figure 5.3: Run Time Scaling with Agent Count (Log-Log Scale)

Adding the timings from Figures 5.2 and 5.4 gives us the total model execution time, as is shown in Figure 5.5 on a linear scale and in Figure 5.6 on a log-log scale. This exposes the true time cost of `BUILTIN_DISTANCE_BASED`, which appeared competitive in Figure 5.2. Although the `ON_DEMAND_BUILTIN` avoids the large initialization time of the `BUILTIN_DISTANCE_BASED` method, it ends up with a visibly worse total time. `ON_DEMAND_BUILTIN` and `BUILTIN_DISTANCE_BASED`, the methods making use of the built-in range search, exhibit the worst total time and scaling.

5.3.2 Run Time Scaling with Connection Range

Figure 5.7 shows scaling of total time with connection range for a population of $500,000$. Given the other settings, connection ranges less that 25 units result in a mostly disconnected (i.e., extremely low-density) network. Above 25, the `SIMPLE_GRID` appears to scale linearly and the `KD_TREE` slightly sub-linearly. The methods making use of the built-in range search have significantly worse total time but appear to have competitive scaling above a connection range of 50 units.

5.3.3 Memory Scaling with Agent Count

Figure 5.8 shows scaling of memory use with agent count, on a log-log scale. The `SIMPLE_GRID` and `KD_TREE` have a large memory overhead but then appear to scale more slowly than the other methods. All tests

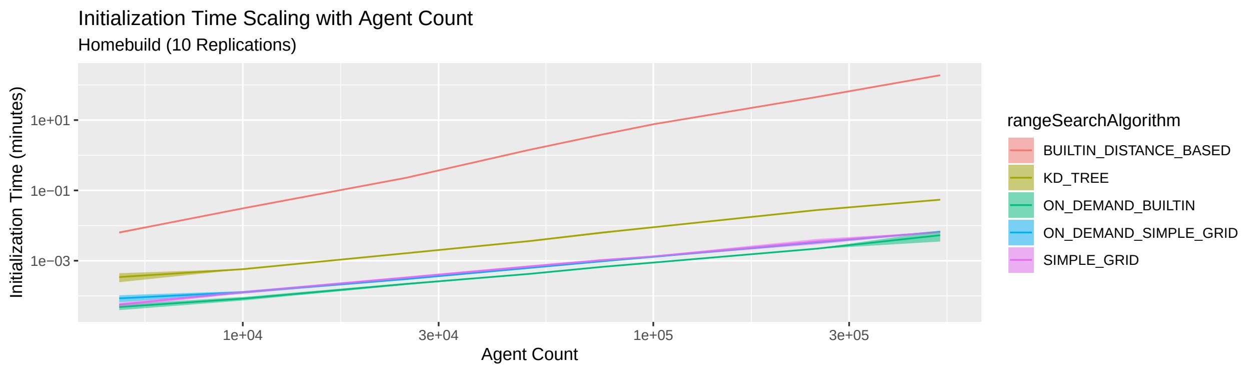


Figure 5.4: Initialization Time Scaling with Agent Count (Log-Log Scale)

remained well within the 16 GB physical RAM of the test system, suggesting that there is no issue with excessive paging to disk. The jagged shape of the curves suggests that perhaps more replications were in order for the test.

5.3.4 Memory Scaling with Connection Range

Figure 5.9 shows scaling of memory use with connection range for a population of $500,000$. Patterns are inconsistent below a range of 25 units, after which all methods appear to scale linearly. It is unclear why the `SIMPLE_GRID` and `ON_DEMAND_SIMPLE_GRID` have such large usage at low connection ranges.

5.3.5 Use with Full Model

Figure 5.10 shows the total time scaling and Figure 5.11 shows memory scaling with population size when used with the full pertussis model. The `SIMPLE_GRID` appears to scale linearly for both total time and memory use. The `KD_TREE`, on the other hand, exhibits super-linear scaling for total time, which constitutes a worsening, rather than an improvement, in performance. The reason for this is that the pertussis model calculates population counts and densities for bins in its operation; with the `SIMPLE_GRID` this can be done with an access by index and traversal of a relatively small list, while with the `KD_TREE` this requires a range search. In other words, more range searches are performed when using the `KD_TREE` than when using the `SIMPLE_GRID`.

The memory use of the `KD_TREE` shows a larger overhead but sub-linear scaling as compared with the `SIMPLE_GRID`. This is as expected and could represent a reasonable tradeoff if the timing gains had materialized.

5.4 Discussion

In a simplified setting, the `KD_TREE` implementation shows improved timing performance over the `SIMPLE_GRID` at the expense of increased memory use. This benefit does not hold when applying the `KD_TREE` to the full

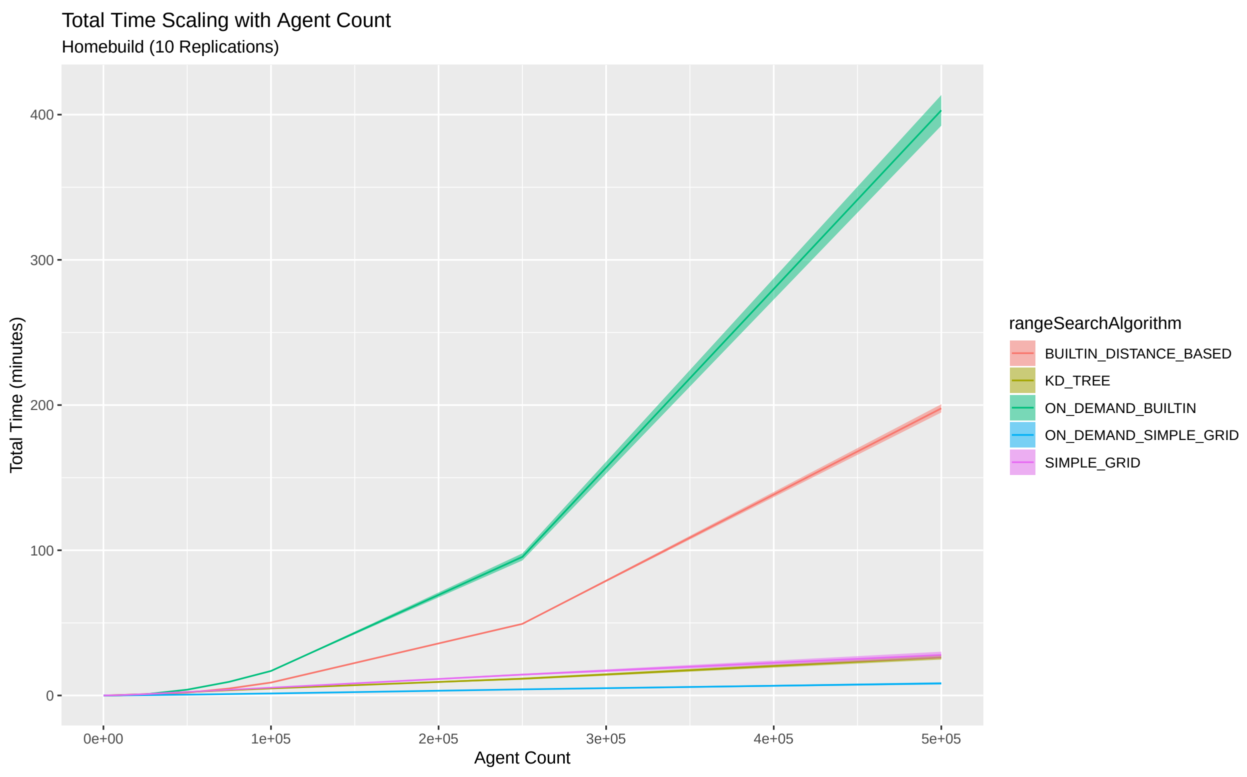


Figure 5.5: Total Time Scaling with Agent Count

pertussis model, however, as this model ends up performing more range searches in place of simpler direct access allowed by the `SIMPLE_GRID`.

Of the configurations tested, however, the stand-out performer is the `ON_DEMAND_SIMPLE_GRID`. This method maintains a Grid data structure for range search purposes and populates the built-in `Connections` object only if required by the agent.

Stochastics can interfere with these empirical tests in the sense that, if there is little disease spread in the test model, there will be fewer messages sent through the contact network, and thus lower computational and memory demands. In retrospect, it may have been better to create a test harness that accessed the network in a more deterministic way. Calculating network centralities may be one way to achieve this. For example, if we were to calculate degree centrality for each of n agents, we could be confident that n range searches had been performed.

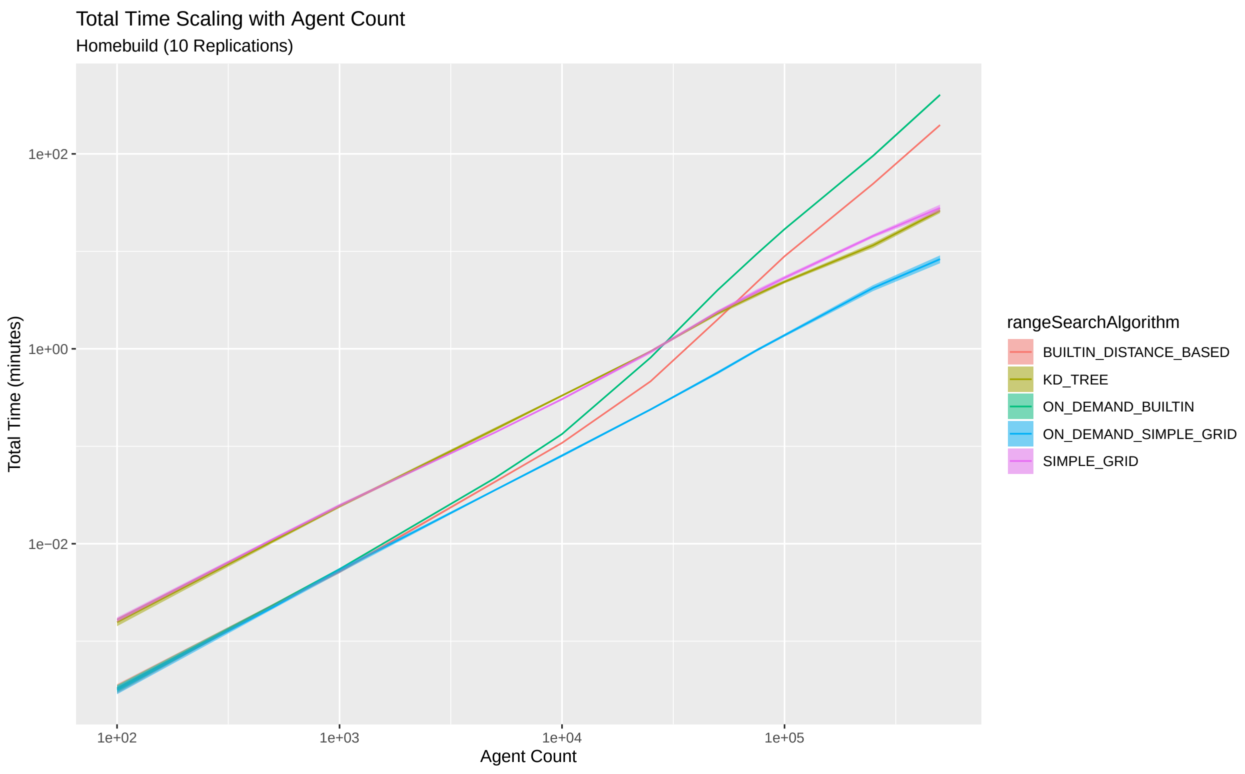


Figure 5.6: Total Time Scaling with Agent Count (Log-Log Scale)

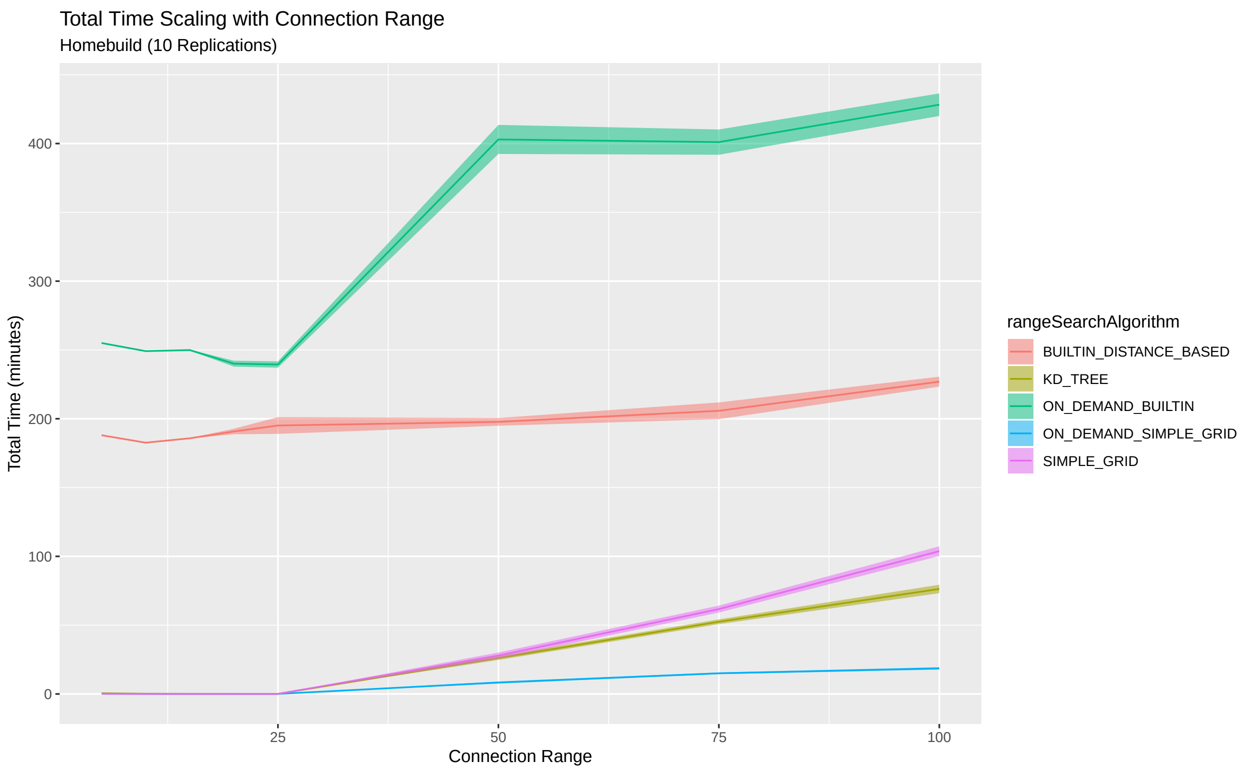


Figure 5.7: Total Time Scaling with Connection Range

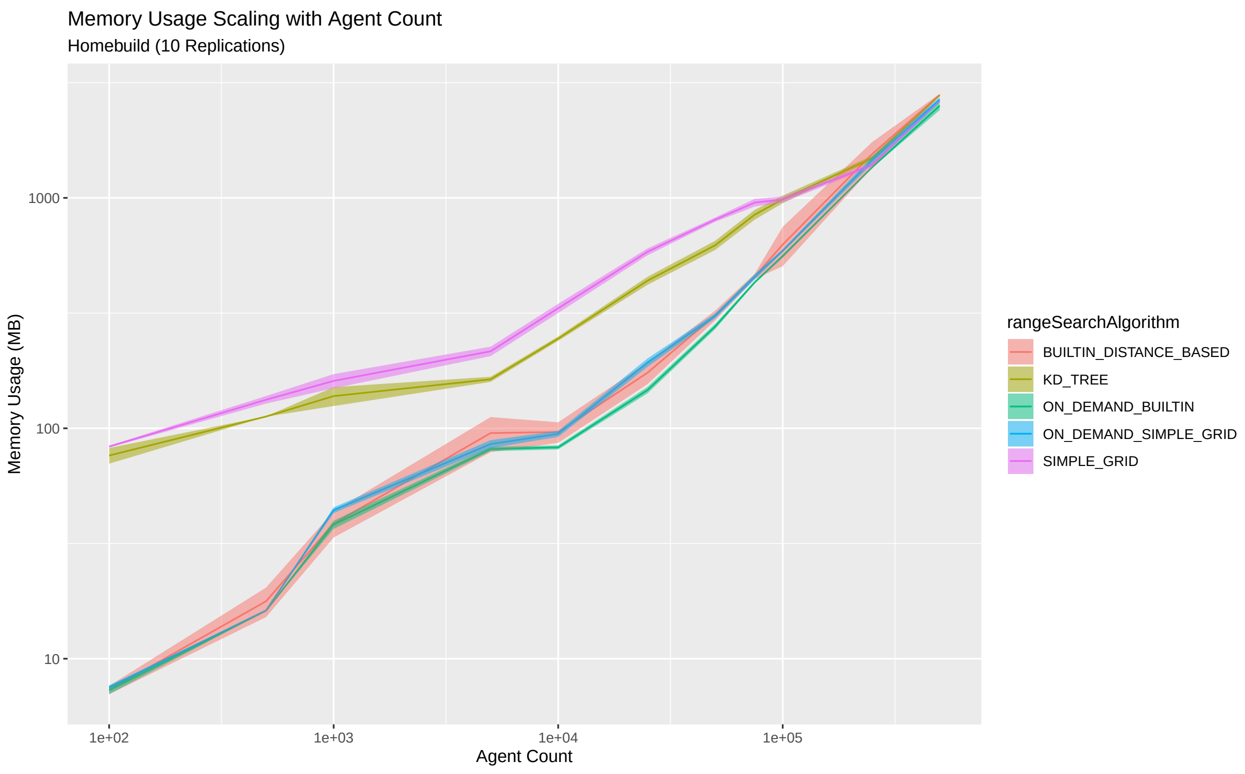


Figure 5.8: Memory Usage Scaling with Agent Count

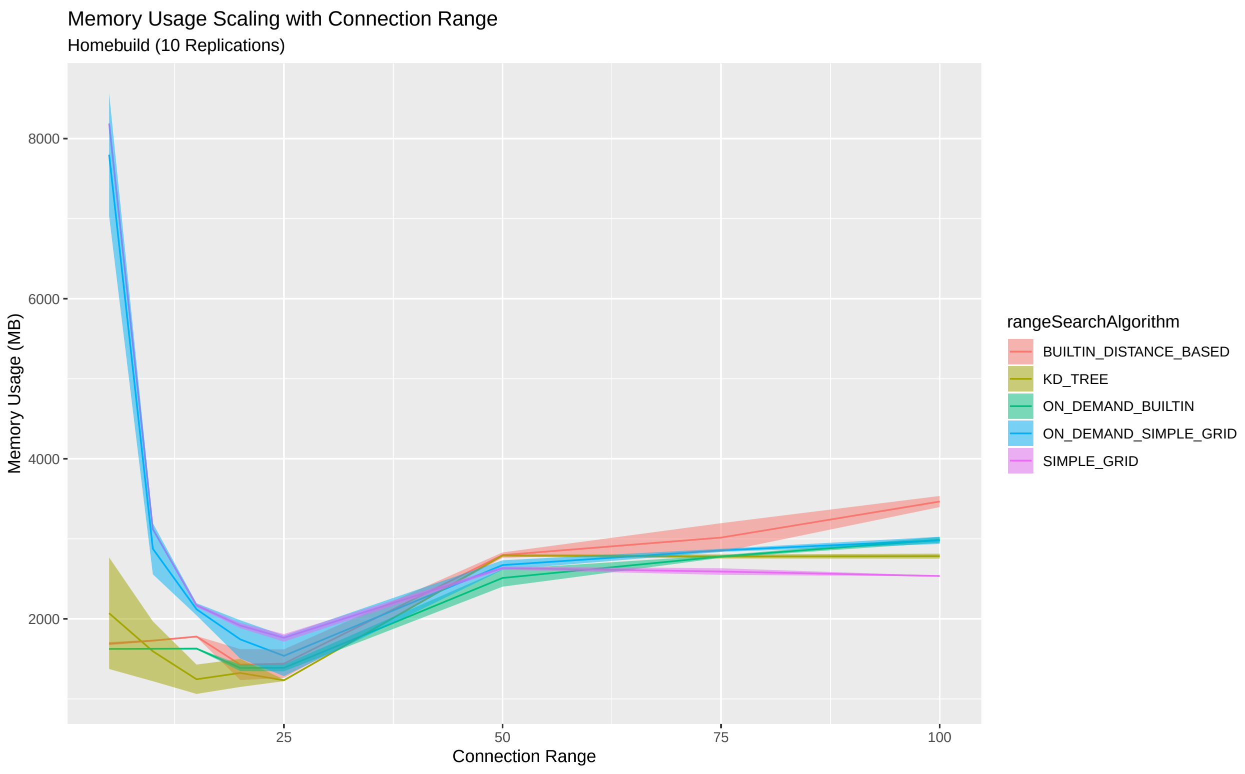


Figure 5.9: Memory Scaling with Connection Range

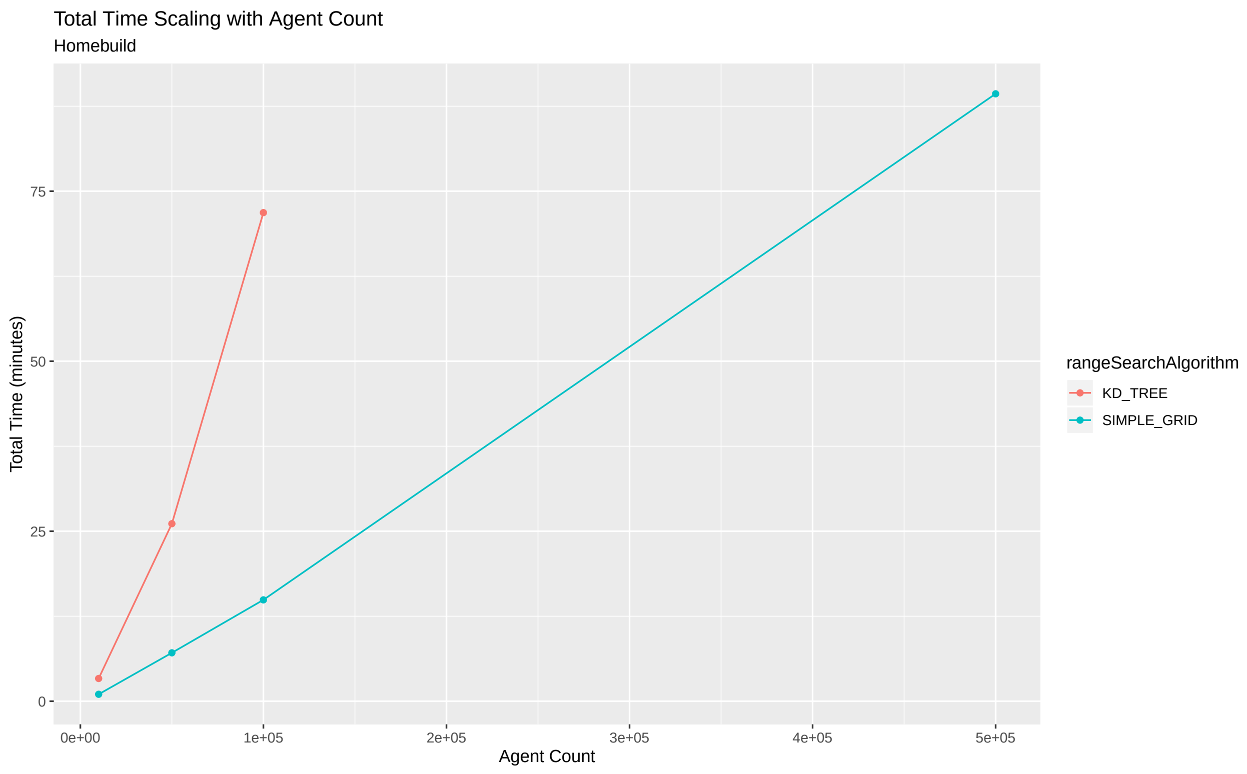


Figure 5.10: Total Time Scaling in Full model

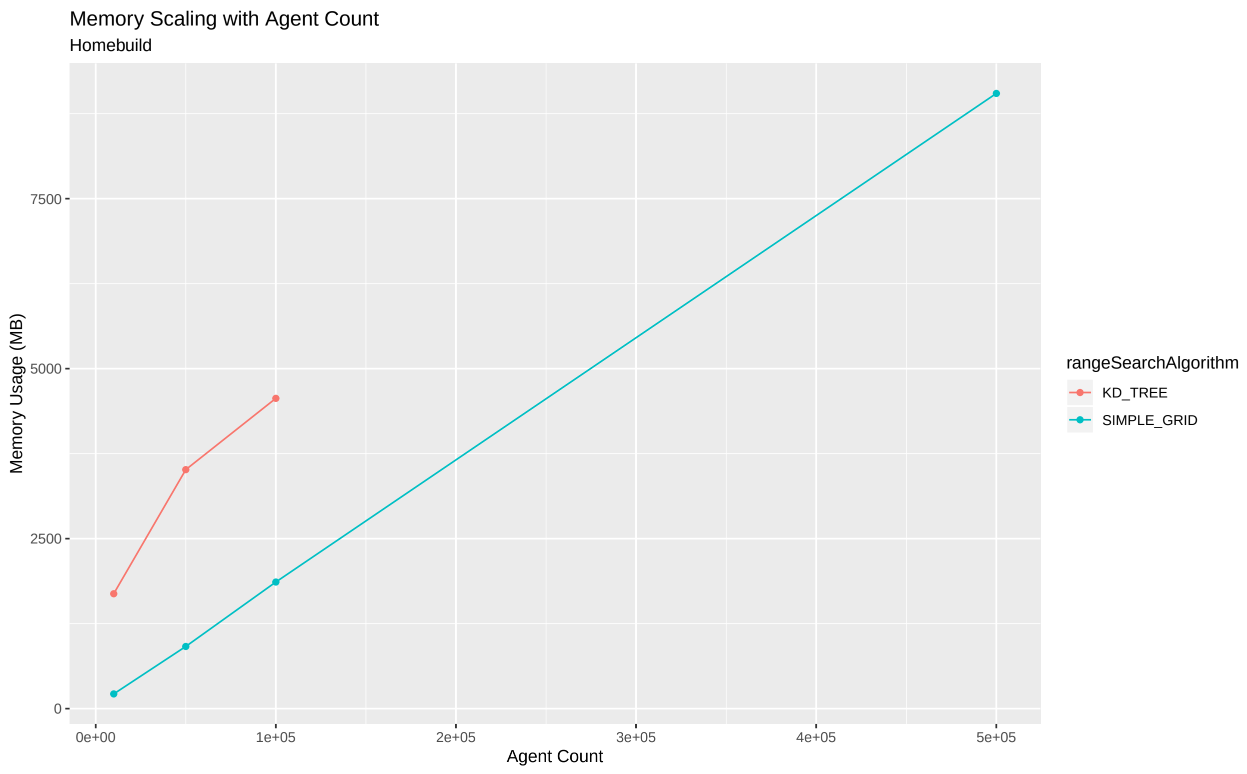


Figure 5.11: Memory Scaling in Full model

6 Conclusion

6.1 Deliverables

There are two main deliverables for this project. The first is an ABM built with AnyLogic that represents pertussis transmission in Alberta and is set up for evaluating the degree to which immunization of mothers during pregnancy reduces pertussis incidence in infants. The second is a Java library implementing a k-d Tree data structure intended to improve range search performance within AnyLogic models.

6.2 Contributions and Findings

6.2.1 Pertussis Model

An agent-based model of pertussis in Alberta was created through an interdisciplinary group model building process. This model met the key requirements of having a population of $500,000$ agents, representing vaccination, disease history, maternal immunity transfer, waning of immunity, and variations in vaccine coverage.

A continuous representation of immunity was developed for this model in the "De Novo" disease module. This representation was capable of representing pertussis behaviour but whether this level of fidelity was required or is a benefit to the health findings is left for future work.

Preliminary results from the pertussis model suggest that providing vaccination during the third trimester of pregnancy with coverage of 50% or 75% could result in reduction of pertussis incidence in infants, especially in the $0 - 2$ month age group. These results are sensitive to parameter assumptions for waning of immunity and immunological blunting.

6.2.2 Data Structures

A k-d Tree data structure was implemented in Java for use with AnyLogic models. In a simplified setting, this data structure showed improved timing and memory scaling compared with the Grid data structure developed by Doroshenko et al. [5] in 2016. These savings did not hold when combining the k-d Tree implementation with the full pertussis model. This was due to an increased number of range searches being performed as compared with the use of the Grid data structure. Combining the Grid structure with the built-in `Connections` object in AnyLogic appears to be the most performant method of maintaining a distance-based network where agents change position infrequently.

6.3 Limitations

6.3.1 Pertussis Model

Some limitations of the pertussis ABM are:

- A particular intervention, immunization during pregnancy, is programmed into the model, so testing of new interventions will require additional programming.

- Tdap was assumed to be the vaccine used for the intervention. With appropriate data to inform parameters, differences between different vaccine formulations could be examined.

- The model implements an abstracted representation of immune response that does not include cellular- or molecular-level interactions between the pathogen and human body. As such, conclusions cannot be drawn from this work regarding immunogenicity or transfer of immunity at those levels.

- Model studies such as this work can help identify whether proposed strategies are worthy of further investigation without incurring a high cost or putting human health at risk. Additional questions, however, must be answered before such an intervention can be deployed as public health policy. One key question among these that is not amenable to a simulation model study is whether it is safe for mother and child to administer a vaccine during pregnancy.

- Scaling results to larger population centres could be problematic as some effects represented in the model, such as density and network effects, may not scale as expected. Some other methods of analysis may allow us to consider a certain population size as “good enough” but that is not the case in a model such as this one without further work.

- The model is parameterized for Alberta specifically. Although attention was paid during development to ensure that the model could be adapted to other jurisdictions by changing input files and parameters, this would nonetheless require significant time, data inputs, and re-calibration.

- GUI elements in the model impose substantial computation and memory burden, such that it is not feasible to run the model at population sizes approaching those used in the main experiment with all GUI elements enabled.

- Pertussis was modelled explicitly, and significant changes would be required to adapt this work to other pathogens.

- Contact networks within the model impose specific mixing assumptions. Altering these assumptions or adding additional contact venues requires modifications to the model structure and subsequent re-calibration.

- Population size is limited to $1,000,000$ agents, nominally. This is an improvement over previous ABMs developed in AnyLogic in CEPHIL, but is not sufficient to allow simulation of the urban Edmonton or Calgary health zones with current populations of approximately 1.3 million each.

6.3.2 Data Structures

Some limitations of the implemented data structures are:

- The `SimpleRangeSearchTree2D` is specifically two-dimensional despite the fact that the k-d Tree can handle an arbitrary number of dimensions. This was a simplifying consideration specifically because the sphere of intended application of this implementation lay within AnyLogic population models.
- The data structures contributed are not parallelized and do not support concurrent access; see Section 4.4.4. This, again, was a simplifying consideration specifically because the intended application was within AnyLogic.
- The data structures are not in feature-parity with implementations of other data structures in the Java Collections Framework [85].
- Use of the `SimpleRangeSearchTree2D` with the full pertussis model resulted in worsened performance. This was due to more range searches being performed when compared with the `SimpleRangeSearchGrid2D`. An alternative data structure, such as a QuadTree, may improve performance while simultaneously allowing direct access to bins.
- The data structures cannot accommodate mobile agents. An agent's position in the `SimpleRangeSearchTree2D` and `SimpleRangeSearchGrid2D` data structures depends on its position in 2D space. Changing the position of an agent therefore requires modification of the data structure, which can be computationally expensive.

6.4 Future Work

In the immediate future, the goal will be to complete calibration and proceed with the main experiment and sensitivity analyses for the pertussis model.

The present work has inspired or elucidated the need for the following improvements going forward:

- Generalization of EpiCurves implementation to allow it to be incorporated into other epidemiological models in AnyLogic with minimal programming.
- Application of the model to test additional research questions regarding pertussis in Alberta. One such question is whether *cocooning*—that is, vaccinating people in the same household as a newborn—would be a viable strategy. While the model does not implement cocooning as an intervention, it could be

added with a small amount of work: During the birth event, vaccination messages could be sent to the other `Person` agents within the newborn's household.

- A health economic analysis of the cost effectiveness of this intervention could be readily conducted with a modified version of this model.
- Comparison of pertussis model outcomes with the different disease module implementations. This could help elucidate differences between a continuous representation of immunity and a discrete one and whether they affect model outcomes.
- Enhancement of the k-d Tree implementation to address concurrency and parallelization concerns; see Section 4.4.4.
- Further exploration of memory demands of the k-d Tree implementation under various JVM configurations.
- Consideration should be given to the degree to which the k-d Tree remains in balance throughout a model run and how that affects overall timing. While a uniform population density distribution was used in the main experiment, a non-uniform distribution has the potential to move the tree out of balance as the population grows. If tree balance has the potential to be problematic, a self-balancing tree-based structure may be worth considering.
- Investigation of the performance efficiency that may be obtained through use of a QuadTree range-search data structure as opposed to the k-d Tree implementation used in the present work.
- Consideration should be given to combining the k-d Tree with the built-in `Connections` object as was done with the Grid structure in the `ON_DEMAND_SIMPLE_GRID` configuration to compare performance.
- Establishment of guidelines and design patterns for use with large models built in AnyLogic, so to minimize computation and memory burden.
- Consideration should be given to caching as a means to improve performance in a model such as the one developed for this work. Since queries are frequent, and only issue from infectious agents, but the population changes infrequently this could be an ideal situation for caching.

References

[1] Government of Canada. (2019) Pertussis (whooping cough). [Online]. Available: https://www.canada.ca/en/public-health/services/immunization/vaccine-preventable-diseases/pertussis-whooping-cough.html

[2] World Health Organization. (2019) Pertussis. [Online]. Available: https://www.who.int/immunization/diseases/pertussis/en/

[3] T. Tan, T. Dalby, K. Forsyth, S. A. Halperin, U. Heininger, D. Hozbor, S. Plotkin, R. Ulloa-Gutierrez, and C. H. W. Von König, "Pertussis across the globe: recent epidemiologic trends from 2000 to 2013," *The Pediatric Infectious Disease Journal*, vol. 34, no. 9, pp. e222–e232, 2015.

[4] X. C. Liu, C. A. Bell, K. A. Simmonds, L. W. Svenson, S. Fathima, S. J. Drews, D. P. Schopflocher, and M. L. Russell, "Epidemiology of pertussis in Alberta, Canada 2004–2015," *BMC public health*, vol. 17, no. 1, p. 539, 2017.

[5] A. Doroshenko, W. Qian, and N. D. Osgood, "Evaluation of outbreak response immunization in the control of pertussis using agent-based modeling," *PeerJ*, vol. 4, p. e2337, 2016.

[6] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," *Proceedings of the National Academy of Sciences*, vol. 99, no. suppl 3, pp. 7280–7287, 2002.

[7] S. F. Railsback and V. Grimm, *Agent-Based and Individual-Based Modeling: A Practical introduction.* Princeton University Press, 2012.

[8] The AnyLogic Company. AnyLogic software. [Online]. Available: https://www.anylogic.com

[9] Centers for Disease Control and Prevention, *Epidemiology and Prevention of Vaccine-Preventable Diseases*, 13th ed., J. Hamborsky, A. Kroger, and S. Wolfe, Eds. Washington, D.C.: Public Health Foundation, 2015.

[10] T. Smith, J. Rotondo, S. Desai, and H. Deehan, "Pertussis: Pertussis surveillance in canada: Trends to 2012," *Canada Communicable Disease Report*, vol. 40, no. 3, p. 21, 2014.

[11] Centers for Disease Control and Prevention. (2019) Pertussis frequently asked questions. [Online]. Available: https://www.cdc.gov/pertussis/about/faqs.html

[12] P. Varughese, "Incidence of pertussis in Canada." *Canadian Medical Association Journal*, vol. 132, no. 9, pp. 1041–1042, 1985.

[13] B. T. Grenfell and R. M. Anderson, "Pertussis in England and Wales: an investigation of transmission dynamics and control by mass vaccination," *Proceedings of the Royal Society of London. B. Biological Sciences*, vol. 236, no. 1284, pp. 213–252, 1989.

[14] M. Domenech de Cellès, F. M. Magpantay, A. A. King, and P. Rohani, "The pertussis enigma: reconciling epidemiology, immunology and evolution," *Proceedings of the Royal Society B: Biological Sciences*, vol. 283, no. 1822, p. 20152309, 2016.

[15] Government of Alberta. Interactive Health Data Application. [Online]. Available: http://www.ahw.gov.ab.ca/IHDA_Retrieval/

[16] R. Frigg and S. Hartmann, "Models in science," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2018.

[17] J. Sokolowski and C. Banks, Eds., *Modeling and Simulation in the Medical and Health Sciences.* Hoboken: Wiley, 2011.

[18] A. M. Starfield, "Qualitative, rule-based modeling," *BioScience*, vol. 40, no. 8, pp. 601–604, 1990.

[19] A. M. Law, *Simulation Modeling and Analysis*, 2nd ed., ser. McGraw-Hill series in industrial engineering and management science. New york: McGraw-Hill, 1991.

[20] R. Ross, *The Prevention of Malaria*, 2nd ed. London: John Murray, 1911.

[21] W. O. Kermack and A. G. McKendrick, "A contribution to the mathematical theory of epidemics," *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 115, no. 772, pp. 700–721, 1927.

[22] J. D. Sterman, "System dynamics modeling: Tools for learning in a complex world," *California Management Review*, vol. 43, no. 4, pp. 8–25, 2001.

[23] N. Osgood, "Using traditional and agent based toolsets for system dynamics: present tradeoffs and future evolution," *System Dynamics*, 2007.

[24] ——, "Representing progression and interactions of comorbidities in aggregate and individual-based systems models," in *Proceedings of the 27th International Conference of the System Dynamics Society. Albuquerque, New Mexico*, 2009.

[25] P. S. Hovmand, *Community-Based System Dynamics.* Springer, 2014.

[26] A. Maltz and G. Fabricius, "SIR model with local and global infective contacts: A deterministic approach and applications," *Theoretical Population Biology*, vol. 112, pp. 70–79, 2016.

[27] J. Badham, E. Chattoe-Brown, N. Gilbert, Z. Chalabi, F. Kee, and R. F. Hunter, "Developing agent-based models of complex health behaviour," *Health & Place*, vol. 54, pp. 170–177, 2018.

[28] A. H. Auchincloss and A. V. Diez Roux, "A new tool for epidemiology: the usefulness of dynamic-agent models in understanding place effects on health," *American Journal of Epidemiology*, vol. 168, no. 1, pp. 1–8, 2008.

[29] Z. Chalabi and T. Lorenc, "Using agent-based models to inform evaluation of complex interventions: examples from the built environment," *Preventive Medicine*, vol. 5, no. 57, pp. 434–435, 2013.

[30] A. M. El-Sayed, P. Scarborough, L. Seemann, and S. Galea, "Social network analysis and agent-based modeling in social epidemiology," *Epidemiologic Perspectives & Innovations*, vol. 9, no. 1, p. 1, 2012.

[31] H. W. Hethcote, "An age-structured model for pertussis transmission," *Mathematical Biosciences*, vol. 145, no. 2, pp. 89–136, 1997.

[32] A. Van Rie and H. W. Hethcote, "Adolescent and adult pertussis vaccination: computer simulations of five new strategies," *Vaccine*, vol. 22, no. 23-24, pp. 3154–3165, 2004.

[33] H. W. Hethcote, P. Horby, and P. McIntyre, "Using computer simulations to compare pertussis vaccination strategies in Australia," *Vaccine*, vol. 22, no. 17-18, pp. 2181–2191, 2004.

[34] G. Fabricius, P. E. Bergero, M. E. Ormazabal, A. Maltz, and D. F. Hozbor, "Modelling pertussis transmission to evaluate the effectiveness of an adolescent booster in argentina," *Epidemiology & Infection*, vol. 141, no. 4, pp. 718–734, 2013.

[35] P. Pesco, P. Bergero, G. Fabricius, and D. Hozbor, "Modelling the effect of changes in vaccine effectiveness and transmission contact rates on pertussis epidemiology," *Epidemics*, vol. 7, pp. 13–21, 2014.

[36] ——, "Mathematical modeling of delayed pertussis vaccination in infants," *Vaccine*, vol. 33, no. 41, pp. 5475–5480, 2015.

[37] M. Gambhir, T. A. Clark, S. Cauchemez, S. Y. Tartof, D. L. Swerdlow, and N. M. Ferguson, "A change in vaccine efficacy and duration of protection explains recent rises in pertussis incidence in the united states," *PLoS Computational Biology*, vol. 11, no. 4, p. e1004138, 2015.

[38] E. Sanstead, C. Kenyon, S. Rowley, E. Enns, C. Miller, K. Ehresmann, and S. Kulasingam, "Understanding trends in pertussis incidence: an agent-based model approach," *American Journal of Public Health*, vol. 105, no. 9, pp. e42–e47, 2015.

[39] P. K. Agarwal, *Handbook of Discrete and Computational Geometry*, 3rd ed. Chapman and Hall/CRC, 2017, ch. 40, pp. 1057–1092.

[40] H. Samet, *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann, 2006.

[41] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[42] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," in *ACM Transactions on Mathematical Software*, 1977.

[43] M. D. Mckay, R. J. Beckman, and W. J. Conover, "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/00401706.1979.10489755

[44] D. Raghavarao, *Repeated Measurements and Cross-Over Designs.* Wiley Online Library, 2013.

[45] The AnyLogic Company. (2020) AnyLogic help. [Online]. Available: https://help.anylogic.com/index.jsp?topic=%2Fcom.anylogic.help%2Fhtml%2Fstatecharts%2FStatecharts.html

[46] L. K. Kreuger, "Data and design: Advancing theory for complex adaptive systems," PhD dissertation, University of Saskatchewan, 2018.

[47] L. Fumanelli, M. Ajelli, P. Manfredi, A. Vespignani, and S. Merler, "Inferring the structure of social contacts from demographic data in the analysis of infectious diseases spread," *PLoS Computational Biology*, vol. 8, no. 9, p. e1002673, 2012.

[48] A. McGirr and D. N. Fisman, "Duration of pertussis immunity after DTaP immunization: a meta-analysis," *Pediatrics*, vol. 135, no. 2, p. 331, 2015.

[49] N. S. Crowcroft, C. Johnson, C. Chen, Y. Li, A. Marchand-Austin, S. Bolotin, K. Schwartz, S. L. Deeks, F. Jamieson, S. Drews, M. L. Russell, L. W. Svenson, K. Simmonds, S. M. Mahmud, and J. C. Kwong, "Under-reporting of pertussis in Ontario: A Canadian Immunization Research Network (CIRN) study using capture-recapture." *PLoS ONE*, vol. 13, no. 5, p. e0195984, 2018.

[50] K. L. Schwartz, J. C. Kwong, S. L. Deeks, M. A. Campitelli, F. B. Jamieson, A. Marchand-Austin, T. A. Stukel, L. Rosella, N. Daneman, S. Bolotin, S. J. Drews, H. Rilkoff, and N. S. Crowcroft, "Effectiveness of pertussis vaccination and duration of immunity," *CMAJ : Canadian Medical Association Journal = Journal de l'Association Medicale Canadienne*, vol. 188, no. 16, p. E399, 2016.

[51] M. Domenech de Cellès, F. M. Magpantay, A. A. King, and P. Rohani, "The impact of past vaccination coverage and immunity on pertussis resurgence," *Science Translational Medicine*, vol. 10, no. 434, p. eaaj1748, 2018.

[52] S. A. Halperin, J. M. Langley, L. Ye, D. MacKinnon-Cameron, M. Elsherif, V. M. Allen, B. Smith, B. A. Halperin, S. A. McNeil, O. G. Vanderkooi, S. Dwinnell, R. D. Wilson, B. Tapiero, M. Boucher, N. Le Saux, A. Gruslin, W. Vaudry, S. Chandra, S. Dobson, and D. Money, "A randomized controlled trial of the safety and immunogenicity of tetanus, diphtheria, and acellular pertussis vaccine immunization during pregnancy and subsequent infant immune response," *Clinical Infectious Diseases*, vol. 67, no. 7, pp. 1063–1071, 2018.

[53] Government of Alberta. (2019) Student population statistics. [Online]. Available: https://www.alberta.ca/student-population-statistics.aspx

[54] J. Mossong, N. Hens, M. Jit, P. Beutels, K. Auranen, R. Mikolajczyk, M. Massari, S. Salmaso, G. S. Tomba, J. Wallinga *et al.*, "Social contacts and mixing patterns relevant to the spread of infectious diseases," *PLoS Medicine*, vol. 5, no. 3, p. e74, 2008.

[55] Government of Alberta. (2019) Open government program. [Online]. Available: https://open.alberta.ca/opendata

[56] Statistics Canada. (2016) Data products, 2016 census. [Online]. Available: https://www12.statcan.gc.ca/census-recensement/2016/dp-pd/index-eng.cfm

[57] S. R. Cole, H. Chu, and S. Greenland, "Maximum likelihood, profile likelihood, and penalized likelihood: a primer," *American Journal of Epidemiology*, vol. 179, no. 2, pp. 252–260, 2013.

[58] P. Zimmermann, K. Perrett, N. Messina, S. Donath, N. Ritz, F. van der Klis, and N. Curtis, "The effect of maternal immunisation during pregnancy on infant vaccine responses," *EClinicalMedicine*, vol. 13, pp. 21–30, 2019.

[59] *Principles of Epidemiology in Public Health Practice – An Introduction to Applied Epidemiology and Biostatistics*, 3rd ed. Atlanta, Georgia, USA: US Department of Health and Human Services, 2013, vol. 8. [Online]. Available: http://www.cdc.gov/ophss/csels/dsepd/SS197

[60] G. A. Weinberg and P. G. Szilagyi, "Vaccine epidemiology: efficacy, effectiveness, and the translational research roadmap.(editorial commentary)(editorial)," *Journal of Infectious Diseases*, vol. 201, no. 11, p. 1607, 2010.

[61] M. Szumilas, "Explaining odds ratios," *Journal of the Canadian Academy of Child and Adolescent Psychiatry*, vol. 19, no. 3, p. 227, 2010.

[62] J. M. Last, Ed., *A Dictionary of Epidemiology.* Oxford University Press, 1983.

[63] The CGAL Project, *CGAL User and Reference Manual.* CGAL Editorial Board, 2019. [Online]. Available: https://doc.cgal.org/4.14/Manual/packages.html

[64] Oracle, *Java Native Interface Specification*, 8th ed., 2014. [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/jniTOC.html

[65] S. Furber, "Microprocessors: The engines of the digital age," *Proceedings. Mathematical, Physical, and Engineering Sciences*, vol. 473, no. 2199, p. 20160893, 2017.

[66] S. Marlow, *Parallel and Concurrent Programming in Haskell*, 1st ed. O'Reilly, 2013.

[67] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference.* ACM, 1967, pp. 483–485.

[68] J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[69] M. M. Raynal, *Concurrent Programming : Algorithms, Principles, and Foundations.* Springer, 2013.

[70] Oracle. (2014) Concurrency - The Java Tutorials. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

[71] M. Ben-Ari, *Principles of Concurrent and Distributed Programming.* Prentice Hall, 1990.

[72] D. Eager, "CMPT 332 course notes," October 2015, unpublished.

[73] A. Downey, *The Little Book of Semaphores.* Green Tea Press, 2008.

[74] Oracle. (2017) Class Thread - Java SE 9 & JDK 9. [Online]. Available: https://docs.oracle.com/javase/9/docs/api/java/lang/Thread.html

[75] ——. (2017) Interface Executor - Java SE 9 & JDK 9. [Online]. Available: https://docs.oracle.com/javase/9/docs/api/java/util/concurrent/Executor.html

[76] ——. (2017) Class ForkJoinPool - Java SE 9 & JDK 9. [Online]. Available: https://docs.oracle.com/javase/9/docs/api/java/util/concurrent/ForkJoinPool.html

[77] E. Herrera. (2019) Introduction to the fork/join framework. [Online]. Available: https://www.pluralsight.com/guides/introduction-to-the-fork-join-framework

[78] Oracle. (2017) Class ForkJoinTask - Java SE 9 & JDK 9. [Online]. Available: https://docs.oracle.com/javase/9/docs/api/java/util/concurrent/ForkJoinTask.html

[79] ——. (2017) Package java.util.stream - Java SE 9 & JDK 9. [Online]. Available: https://docs.oracle.com/javase/9/docs/api/java/util/stream/package-summary.html

[80] Microsoft. (2011) How to launch a process with CPU affinity set. [Online]. Available: https://docs.microsoft.com/en-ca/archive/blogs/santhoshonline/how-to-launch-a-process-with-cpu-affinity-set

[81] Canonical. (2004) Man page for taskset (1). [Online]. Available: http://manpages.ubuntu.com/manpages/trusty/man1/taskset.1.html

[82] Apple. (2007) Thread affinity API release notes. [Online]. Available: https://developer.apple.com/library/archive/releasenotes/Performance/RN-AffinityAPI/

[83] W. Qian, "An example of parallel calibration in AnyLogic," August 2018, personal correspondence.

[84] Oracle. (2017) Package java.util.concurrent - Java SE 9 & JDK 9. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html

[85] ——. (2017) Interface Collection - Java SE 9 & JDK 9. [Online]. Available: https://docs.oracle.com/javase/9/docs/api/java/util/Collection.html

Appendix A

Code Listings

A.1 KDPoint

```java
package ca.usask.gwm762.kdtree;

/**
 * The KDPoint interface defines a general n-dimensional point to be used as the
 * coordinate for a KDTree.
 *
 * @param <T> the type of the dimension (e.g., Double, but could be any
 *            Comparable type)
 *
 * @author  G. Wade McDonald
 * @version 1.0
 * @since   2018
 */
public interface KDPoint<T extends Comparable<T>> {
    /**
     *
     * @param dimension the given dimension (0-indexed).
     * @return the value of the coordinate at the given dimension.
     * @exception ArrayIndexOutOfBoundsException if the point doesn't contain
     * the dimension.
     */
    T get(int dimension);

    /**
     *
     * @return the number of dimensions in the Point.
     */
    int getDimensions();

    /**
     * Compares this point to another on the basis of the given dimension
     *
     * @param otherPoint the point to compare to
     * @param dimension the dimension to compare on
     * @return {@code this.get(dimension).compareTo(otherPoint.get(dimension))}
     * if the points are compatible
     * @exception ArrayIndexOutOfBoundsException if dimension is out of bounds
     * @exception IllegalArgumentException if the points have different dimensions
     */
    int compareOnDimension(KDPoint<?> otherPoint, int dimension);

    /**
     *
     * @param other the point to compute the distance to.
     * @return the squared distance between this point and other.
     * @exception IllegalArgumentException if the points have different dimensions
     */
    T distanceSquared(KDPoint<T> other);

    /**
     * Checks if the point resides within the rectangular range bounded by two
     * given points
     * Allows {@code null} for upper or lower to indicate no bound
     *
     * @param upper the upper bounding point of the range
     * @param lower the lower bounding point of the range
     * @return true if the point is in the range bounded by upper and lower
     * @exception IllegalArgumentException if the points have different dimensions
     */
    boolean isInRange(KDPoint<?> upper, KDPoint<?> lower);

    /**
     * Checks if the point resides within the specified Euclidean radius of the
     * specified point
     *
     * @param center the center point of the range
     * @param radius the radius of the range
     * @return true if the point is within radius of center
     * @exception IllegalArgumentException if the points have different dimensions
     */
    boolean isInEuclideanRange(KDPoint<T> center, T radius);
}
```

A.2 DoublePoint

```java
package ca.usask.gwm762.kdtree;

import static java.lang.Math.pow;

/**
 * Implements the KDPoint interface for type Double for use with the KDTree
 */
public class DoublePoint implements KDPoint<Double>, Cloneable {
    private Double[] coordinates;

    /**
     * Instantiates a DoublePoint with given dimensions and all coordinates set
     * to 0.0
     *
     * @param dimensions the number of dimensions for the point
     */
    public DoublePoint(int dimensions) {
        if(dimensions < 1) throw new IllegalArgumentException("Dimensions must be strictly
    positive.");

        this.coordinates = new Double[dimensions];

        for(int i = 0; i < coordinates.length; i++) {
            this.set(i, 0.0);
        }
    }

    /**
     * Instantiates a DoublePoint with coordinates specified as Double[]
     *
     * @param coordinates the coordinates of the point
     */
    public DoublePoint(Double[] coordinates) {
        this(coordinates.length);
        this.coordinates = coordinates;
    }

    /**
     * Instantiates a DoublePoint with coordinates specified as double[]
     *
     * @param coordinates the coordinates of the point
     */
    public DoublePoint(double[] coordinates) {
        this(coordinates.length);

        for(int i = 0; i < coordinates.length; i++) {
            this.set(i, coordinates[i]);
        }
    }

    /**
     * Sets the specified coordinate of the DoublePoint to the specified value
     *
     * @param dimension the coordinate to set
     * @param value the new value of the coordinate
     */
    public void set(int dimension, double value) {
        this.set(dimension, Double.valueOf(value));
    }

    /**
     * Sets the specified coordinate of the DoublePoint to the specified value
     *
     * @param dimension the coordinate to set
     * @param value the new value of the coordinate
     */
    public void set(int dimension, Double value) {
        checkDimension(dimension);

        this.coordinates[dimension] = value;
    }

    /**
     * gets the values of the specified coordinate of the point
     *
     * @param dimension the given dimension (0-indexed).
     * @return the value of the coordinate at the given dimension
     */
    @Override
    public Double get(int dimension) {
        checkDimension(dimension);

        return coordinates[dimension];
    }

    /**
     * Gets the number of dimensions
     *
     * @return the number of dimensions
     */
    @Override
    public int getDimensions() {
        return coordinates.length;
    }

    /**
     * Compares to another DoublePoint based on the given dimension
     *
     * @param otherPoint the point to compare to
     * @param dimension the dimension to compare on
     * @return {@code this.get(dimension).compareTo(otherPoint.get(dimension))}
     * if the points are compatible
     * @exception ArrayIndexOutOfBoundsException if dimension is out of bounds
     * @exception IllegalArgumentException if the points have different dimensions
     */
    @Override
    public int compareOnDimension(KDPoint<?> otherPoint, int dimension) {
        if(this.getDimensions() != otherPoint.getDimensions())
            throw new IllegalArgumentException("Dimensions of points must be equal.");

        checkDimension(dimension);

        return this.get(dimension).compareTo((Double)otherPoint.get(dimension));
    }

    /**
     * Returns the squared distance between this point and another DoublePoint
     *
     * @param other the point to compute the distance to.
     * @return the squared distance between the points
     * @exception IllegalArgumentException if the points have different dimensions
     */
    @Override
    public Double distanceSquared(KDPoint<Double> other) {
        double sumSq = 0.0;

        if(this.getDimensions() != other.getDimensions())
            throw new IllegalArgumentException("Dimensions of points must be equal.");

        for(int i = 0; i < this.getDimensions(); i++) {
            Double x1 = this.get(i);
            Double x2 = other.get(i);
            sumSq += pow((x1 - x2), 2.0);
        }

        return sumSq;
    }

    /**
     * Checks if the point resides within the rectangular, axis-aligned range
     * bounded by two given points
     *
     * Allows {@code null} for upper or lower to indicate no bound
     * @param upper the upper bounding point of the range
     * @param lower the lower bounding point of the range
     * @return true if the point is in the range bounded by upper and lower
     */
    @Override
    public boolean isInRange(KDPoint<?> upper, KDPoint<?> lower) {
        if (lower != null && this.getDimensions() != lower.getDimensions())
            throw new IllegalArgumentException("Dimensions of points must be equal.");
        if (upper != null && this.getDimensions() != upper.getDimensions())
            throw new IllegalArgumentException("Dimensions of points must be equal.");

        if(lower != null || upper != null) {

            for(int i = 0; i < this.getDimensions(); i++) {
                if((lower != null) && (this.compareOnDimension(lower, i) < 0)) {
                    return false;
                }
                if((upper != null) && (this.compareOnDimension(upper, i) > 0)) {
                    return false;
                }
            }

        }

        return true;
    }

    /**
     * Checks if the point resides within the specified Euclidean radius of the
     * specified point
     *
     * @param center the center point of the range
     * @param radius the radius of the range
     * @return true if the point is within radius of center
     */
    @Override
    public boolean isInEuclideanRange(KDPoint<Double> center, Double radius) {
        if (center == null || this.getDimensions() != center.getDimensions())
            throw new IllegalArgumentException("Dimensions of points must be equal.");

        Double radiusSquared = pow(radius, 2.0);

        return (this.distanceSquared(center) <= radiusSquared);
    }

    /**
     * Checks for dimensional compatibility
     *
     * @param dimension the required dimension
     * @throws ArrayIndexOutOfBoundsException if the dimension of this point is
     * not equal to the given dimension
     */
    private void checkDimension(int dimension) throws ArrayIndexOutOfBoundsException {
        int dim = this.coordinates.length;
        if(dimension < 0 || dimension > dim - 1)
            throw new ArrayIndexOutOfBoundsException("Dimension out of bounds: " + dimension
     + " for " + dim);
    }
```

```java

    /**
     * Checks for equality
     *
     * @param obj the Object to compare to
     * @return true of obj is a DoublePoint with identical coordinates
     */
    @Override
    public boolean equals(Object obj) {
        if(obj == null) return false;

        if(!(obj instanceof DoublePoint)) return false;

        DoublePoint other = (DoublePoint)obj;

        if(other == this) return true;

        if(this.getDimensions() != other.getDimensions()) return false;

        for(int i = 0; i < this.coordinates.length; i++) {
            if(!this.get(i).equals(other.get(i)))
                return false;
        }

        return true;
    }

    /**
     * Computes a hascode for this object
     *
     * @return the hashcode
     */
    @Override
    public int hashCode() {
        //TODO: Consider a more advanced hashing algorithm here?
        //A simple way to hash a multi-dimensional coordinate and ensure that
        //equal coordinates get the same hash.
        return this.toString().hashCode();
    }

    /**
     * Clones this Object
     *
     * @return the clone
     * @throws CloneNotSupportedException if clone is not supported
     */
    @Override
    public Object clone() throws CloneNotSupportedException {
        DoublePoint newClone = (DoublePoint)super.clone();

        for(int i = 0; i < this.coordinates.length; i++) {
            newClone.set(i, this.coordinates[i].doubleValue());
        }

        return newClone;
    }

    /**
     * Returns a string representation of the point
     *
     * @return the string representation
     */
    @Override
    public String toString() {
        StringBuilder s = new StringBuilder();
        s.append("(");

        for(int i = 0; i < this.coordinates.length; i++) {
            if(i > 0) s.append(", ");
            s.append(this.coordinates[i].toString());
        }

        s.append(")");

        return s.toString();
    }
}
```

A.3 HasDoublePointCoordinate

```java
package ca.usask.gwm762.rangesearch2d;

import ca.usask.gwm762.kdtree.DoublePoint;

public interface HasDoublePointCoordinate {
    /**
     * Returns the 2D coordinates of the object as a DoublePoint
     *
     * @return the 2D coordinates of the object
     */
    DoublePoint getCoordinates();
}
```

A.4 RangeSearchGrid2D

```java
package ca.usask.gwm762.rangesearch2d;

import ca.usask.gwm762.kdtree.DoublePoint;

import java.util.Iterator;

public interface RangeSearchGrid2D<T extends HasDoublePointCoordinate> {

    int getBinDimX();
    int getBinDimY();

    void add(T item);

    boolean remove(T item);

    Iterator<T> rangeSearchIterator(DoublePoint upper, DoublePoint lower);

    Iterator<T> binIterator(int grid_x, int grid_y);

    Iterator<T> radiusSearchIterator(DoublePoint center, Double radius);

}
```

A.5 SimpleRangeSearchTree2D

```
package ca.usask.gwm762.rangesearch2d;

import ca.usask.gwm762.kdtree.*;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Comparator;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.function.Consumer;

public class SimpleRangeSearchTree2D<V extends HasDoublePointCoordinate> implements
    RangeSearchGrid2D<V> {

    private static class Simple2DNode<V>{
        int discriminator;
        DoublePoint location;
        List<V> values;
        Simple2DNode<V> low, high;

        Simple2DNode(int discriminator, double x, double y, V value) {
            this.discriminator = discriminator;
            this.location = new DoublePoint(new double[] {x, y});
            this.values = new LinkedList<>();
            this.values.add(value);
            this.low = null;
            this.high = null;
        }

        double getX() {
            return this.location.get(0);
        }

        double getY() {
            return this.location.get(1);
        }

        DoublePoint getKey() {
            return this.location;
        }

        List<V> getValues() {
            return this.values;
        }

        void addValue(V item) {
            this.values.add(item);
        }

        boolean removeValue(V item) {
            return this.values.remove(item);
        }

        int getDiscriminator() {
            return this.discriminator;
        }

        int nextDiscriminator() {
            return (this.discriminator + 1) % 2;
        }

        int depthFromHere() {
            int lowDepth = low == null ? 0 : low.depthFromHere();
```

```java
            int highDepth = high == null ? 0 : high.depthFromHere();

            return 1 + Math.max(lowDepth, highDepth);
        }
    }

    private Simple2DNode<V> root;
    private double spaceWidth;
    private double spaceHeight;
    private double binWidth;
    private DoublePoint spaceLowerBound = new DoublePoint(new Double[] {0.0, 0.0});
    private DoublePoint spaceUpperBound;
    private int binDimX;
    private int binDimY;
    private int size = 0;

    public SimpleRangeSearchTree2D(double spaceWidth, double spaceHeight, double binWidth) {
        this.root = null;

        this.spaceWidth = spaceWidth;
        this.spaceHeight = spaceHeight;
        this.spaceUpperBound = new DoublePoint(new Double[] {spaceWidth, spaceHeight});
        this.binWidth = binWidth;

        this.binDimX = ( (int) Math.ceil( spaceWidth / binWidth ) );
        this.binDimY = ( (int) Math.ceil( spaceHeight / binWidth ) );
    }

    public boolean isEmpty() {
        return root == null;
    }

    public void init(Collection<V> data) {
        if(!this.isEmpty()) throw new RuntimeException("Tree has already been initialized.");

        LinkedHashMap<DoublePoint, LinkedList<V>> tempHashMap = new LinkedHashMap<>(data.size() / 2);

        data.forEach(p -> {
            DoublePoint coord = p.getCoordinates();

            LinkedList<V> entry = tempHashMap.computeIfAbsent(coord, k -> new LinkedList<>());

            entry.add(p);
        });

        ArrayList<DoublePoint> tempIndex = new ArrayList<>(tempHashMap.keySet());

        init(tempHashMap, tempIndex, 0, 0, tempIndex.size() - 1);
    }

    private void init(LinkedHashMap<DoublePoint, LinkedList<V>> data, ArrayList<DoublePoint> index,
                      int depth, int beginIdx, int endIdx) {
        int discriminator = depth % 2;
        int size = endIdx - beginIdx + 1;

        if(size > 1) {
            int cur = beginIdx + size / 2;
            int prev = cur - 1;

            index.subList(beginIdx, endIdx).sort(Comparator.comparingDouble(o -> o.get(discriminator)));

            while(cur > beginIdx && index.get(cur).compareOnDimension(index.get(prev), discriminator) == 0){
                cur--;
                prev--;
            }

            data.get(index.get(cur)).forEach(this::add);

            init(data, index,depth + 1, beginIdx, cur - 1);

            init(data, index,depth + 1, cur + 1, endIdx);

        } else if(size == 1) {
            data.get(index.get(beginIdx)).forEach(this::add);
        }
    }

    @Override
    public int getBinDimX() {
        return binDimX;
    }

    @Override
    public int getBinDimY() {
        return binDimY;
    }

    @Override
    public void add(V item) {
        DoublePoint location = item.getCoordinates();

        if(!location.isInRange(spaceUpperBound, spaceLowerBound))
            throw new IllegalArgumentException(item + " out of bounds: " + location);

        double itemX = location.get(0);
        double itemY = location.get(1);

        if(root == null) {
            //tree is empty, new root node
            this.root = new Simple2DNode<>(0, itemX, itemY, item);
            this.size++;
            return;
        }

        Simple2DNode<V> cur;
        Simple2DNode<V> next = root;

        //TODO: Review This
        //TODO: Check that size is incremented and decremented correctly
        while(next != null) {
            cur = next;

            //if (cur.getX() == itemX && cur.getY() == itemY) {
            if (cur.getKey().equals(location)) {
                //Coordinates are equal put the new value here and return
                cur.addValue(item);
                return;
            }

            if(cur.getDiscriminator() == 0 ? (itemX < cur.getX()) : (itemY < cur.getY())) {
                //point is less than current coordinate, move to low child
                next = cur.low;

                //this is a leaf node, new child with new value
                if(next == null) {
                    cur.low = new Simple2DNode<>(cur.nextDiscriminator(), itemX, itemY, item
    );
                    return;
                }
            } else {
                //move to high child
                next = cur.high;

                //this is a leaf node, new child with new value
                if(next == null) {
                    cur.high = new Simple2DNode<>(cur.nextDiscriminator(), itemX, itemY,
    item);
                    return;
                }
            }
        }
    }

    @Override
    public boolean remove(V item) {
        DoublePoint location = item.getCoordinates();

        if(!location.isInRange(spaceUpperBound, spaceLowerBound))
            throw new IllegalArgumentException(item + " out of bounds: " + location);

        if(root == null) {
            //tree is empty
            return false;
        }

        double itemX = location.get(0);
        double itemY = location.get(1);

        Simple2DNode<V> cur;
        Simple2DNode<V> next = root;

        while(next != null) {
            cur = next;

            if ((cur.getX() == itemX) && (cur.getY() == itemY)) {
                //Coordinates are equal attempt removal here
                boolean check = cur.removeValue(item);
                if(check) this.size--;
                return check;
            }

            if(cur.getDiscriminator() == 0 ? (itemX < cur.getX()) : (itemY < cur.getY())) {
                //point is less than current coordinate, move to low child
                next = cur.low;
            } else {
                //move to high child
                next = cur.high;
            }
        }
        return false;
    }

    final class RangeSearchIterator implements Iterator<V> {
        DoublePoint upperBound;
        DoublePoint lowerBound;
        LinkedList<Simple2DNode<V>> nodeStack = new LinkedList<>();
        LinkedList<V> valueStack = new LinkedList<>();
        V next;

        RangeSearchIterator(DoublePoint upperBound, DoublePoint lowerBound) {
            this.upperBound = upperBound;
            this.lowerBound = lowerBound;

            if(root != null) {
                nodeStack.push(root);
            }

            next = nextInRange();
```

```java
    }

    private V nextInRange() {
        while(valueStack.isEmpty() && !nodeStack.isEmpty()) {
            Simple2DNode<V> node = nodeStack.pop();

            int disc = node.getDiscriminator();

            if(upperBound == null || node.getKey().compareOnDimension(upperBound, disc) <= 0) {
                if(node.high != null)
                    nodeStack.push(node.high);
            }

            if(lowerBound == null || node.getKey().compareOnDimension(lowerBound, disc) > 0) {
                if(node.low != null)
                    nodeStack.push(node.low);
            }

            if(node.getKey().isInRange(upperBound, lowerBound)) {
                node.getValues().forEach(v -> valueStack.push(v));
            }
        }

        if(!valueStack.isEmpty()) {
            return(valueStack.pop());
        } else {
            return null;
        }
    }

    @Override
    public boolean hasNext() {
        return next != null;
    }

    @Override
    public V next() {
        V last = next;
        next = nextInRange();
        return last;
    }

    @Override
    public void remove() {
        //TODO: possibly implement this
        throw new UnsupportedOperationException();
    }

    @Override
    public void forEachRemaining(Consumer<? super V> action) {
        //TODO: implement this
    }
}

@Override
public Iterator<V> rangeSearchIterator(DoublePoint upper, DoublePoint lower) {
    return new RangeSearchIterator(upper, lower);
}

@Override
public Iterator<V> binIterator(int grid_x, int grid_y) {
    if (grid_x < 0 || grid_x >= binDimX || grid_y < 0 || grid_y >= binDimY)
        throw new IndexOutOfBoundsException("Coordinates specified are outside of grid bounds.");

    double maxX = binWidth * (grid_x + 1.0);
```

```
        double maxY = binWidth * (grid_y + 1.0);
        double minX = binWidth * grid_x;
        double minY = binWidth * grid_y;

        DoublePoint lowerBound = new DoublePoint(new Double[] {minX, minY});
        DoublePoint upperBound = new DoublePoint(new Double[] {maxX, maxY});

        return rangeSearchIterator(upperBound, lowerBound);
    }

    final class RadiusSearchIterator implements Iterator<V> {
        DoublePoint upperBound;
        DoublePoint lowerBound;
        DoublePoint center;
        Double radius;
        LinkedList<Simple2DNode<V>> nodeStack = new LinkedList<>();
        LinkedList<V> valueStack = new LinkedList<>();
        V next;

        RadiusSearchIterator(DoublePoint center, Double radius) {
            this.center = center;
            this.radius = radius;

            //Determine bounds for square search from center and radius
            this.upperBound = new DoublePoint(new double[] {center.get(0) + radius, center.
    get(1) + radius});
            this.lowerBound = new DoublePoint(new double[] {center.get(0) - radius, center.
    get(1) - radius});

            if(root != null) {
                nodeStack.push(root);
            }

            next = nextInRange();
        }

        private V nextInRange() {
            while(valueStack.isEmpty() && !nodeStack.isEmpty()) {
                Simple2DNode<V> node = nodeStack.pop();

                int disc = node.getDiscriminator();

                if(upperBound == null || node.getKey().compareOnDimension(upperBound, disc)
    <= 0) {
                    if(node.high != null)
                        nodeStack.push(node.high);
                }

                if(lowerBound == null || node.getKey().compareOnDimension(lowerBound, disc)
    > 0) {
                    if(node.low != null)
                        nodeStack.push(node.low);
                }

                if(node.getKey().isInEuclideanRange(center, radius)) {
                    node.getValues().forEach(v -> valueStack.push(v));
                }
            }

            if(!valueStack.isEmpty()) {
                return(valueStack.pop());
            } else {
                return null;
            }
        }

        @Override
        public boolean hasNext() {
```

```java
            return next != null;
        }

        @Override
        public V next() {
            V last = next;
            next = nextInRange();
            return last;
        }

        @Override
        public void remove() {
            //TODO: possibly implement this
            throw new UnsupportedOperationException();
        }

        @Override
        public void forEachRemaining(Consumer<? super V> action) {
            //TODO: implement this
        }
    }

    @Override
    public Iterator<V> radiusSearchIterator(DoublePoint center, Double radius) {
        return new RadiusSearchIterator(center, radius);
    }

    public int size() {
        return this.size;
    }

    public int depth() {
        return this.root == null ? 0 : root.depthFromHere();
    }
}
```

A.6 SimpleRangeSearchGrid2D

```java
package ca.usask.gwm762.rangesearch2d;

import java.util.Iterator;
import java.util.function.Consumer;
import java.util.List;
import java.util.LinkedList;
import java.util.ArrayList;
import java.io.Serializable;
import ca.usask.gwm762.kdtree.*;

public class SimpleRangeSearchGrid2D<T extends HasDoublePointCoordinate> implements
    RangeSearchGrid2D<T>, Serializable {

    private double spaceWidth;
    private double spaceHeight;
    private double binWidth;
    private int binDimX;
    private int binDimY;
    private int cellSizeFactor;
    private double cellWidth;
    private int cellDimX;
    private int cellDimY;
    private ArrayList<ArrayList<Cell<T>>> grid;

    /**
     * Default constructor
     */
    public SimpleRangeSearchGrid2D(double spaceWidth, double spaceHeight, double binWidth,
    int cellSizeFactor) {
        this.spaceHeight = spaceHeight;
        this.spaceWidth = spaceWidth;
        this.binWidth = binWidth;

        this.binDimX = ( (int) Math.ceil( spaceWidth / binWidth ) );
        this.binDimY = ( (int) Math.ceil( spaceHeight / binWidth ) );

        this.cellSizeFactor = cellSizeFactor;
        this.cellDimX = this.binDimX * this.cellSizeFactor;
        this.cellDimY = this.binDimY * this.cellSizeFactor;
        this.cellWidth = this.binWidth / (double)this.cellSizeFactor;

        this.grid = new ArrayList<>(cellDimX);

        for (int i = 0; i < cellDimX; i++) {
            ArrayList<Cell<T>> gridCol = new ArrayList<>(cellDimY);
            for (int j = 0; j < cellDimY; j++) {
                gridCol.add(j, new Cell<>());
            }
            grid.add(i, gridCol);
        }
    }

    @Override
    public int getBinDimX() {
        return binDimX;
    }

    @Override
    public int getBinDimY() {
        return binDimY;
    }

    @Override
    public void add(T item) {
        DoublePoint pos = item.getCoordinates();

        Cell<T> cell = getGridCellForPos(pos.get(0), pos.get(1));
        cell.items.add(item);
    }

    public boolean remove(T item) {
        DoublePoint pos = item.getCoordinates();

        Cell<T> cell = getGridCellForPos(pos.get(0), pos.get(1));
        return cell.items.remove(item);
    }

    public List<T> binSearch(int grid_x, int grid_y) {
        List<T> foundItems = new ArrayList<>();

        int base_cell_grid_x = grid_x * cellSizeFactor;
        int base_cell_grid_y = grid_y * cellSizeFactor;

        for(int i = base_cell_grid_x; i < base_cell_grid_x + cellSizeFactor; i++) {
            for(int j = base_cell_grid_y; j < base_cell_grid_y + cellSizeFactor; j++) {
                Cell<T> cell = getGridCellForIndex(i, j);
                foundItems.addAll(cell.items);
            }
        }

        return foundItems;
    }

/*
    public List<T> rangeSearch(T agent, double radius) {
        List<T> foundAgents = new ArrayList<>();

        Double radiusSquared = Math.pow(radius, 2.0);

        int offsetOfNumOfBinsToChk = (int) Math.ceil(radius / binWidth);

        DoublePoint pos = agent.getCoordinates();
        int grid_x = (int) Math.floor(pos.get(0) / binWidth);
        int grid_y = (int) Math.floor(pos.get(1) / binWidth);

        for (int i = grid_x - offsetOfNumOfBinsToChk; i <= grid_x + offsetOfNumOfBinsToChk; i++) {
            for (int j = grid_y - offsetOfNumOfBinsToChk; j <= grid_y + offsetOfNumOfBinsToChk; j++) {
                if (i >= 0 && i < binDimX && j >= 0 && j < binDimY) {
                    Cell<T> cell = getGridCellForIndex(i, j);

                    for (T other: cell.items) {
                        if (agent != other && pos.distanceSquared(other.getCoordinates()) <= radiusSquared)
                            foundAgents.add(other);
                    }
                }
            }
        }

        return foundAgents;
    }
*/

    public int getBinPopulation(int grid_x, int grid_y) {
        int size = 0;

        int base_cell_grid_x = grid_x * cellSizeFactor;
        int base_cell_grid_y = grid_y * cellSizeFactor;

        for(int i = base_cell_grid_x; i < base_cell_grid_x + cellSizeFactor; i++) {
            for(int j = base_cell_grid_y; j < base_cell_grid_y + cellSizeFactor; j++) {
                Cell<T> cell = getGridCellForIndex(i, j);
                size += cell.items.size();
            }
        }

        return size;
    }

    private Cell<T> getGridCellForIndex(int grid_x, int grid_y) {
        if (grid_x < 0 || grid_x >= this.cellDimX || grid_y < 0 || grid_y >= this.cellDimY)
            throw new IndexOutOfBoundsException("Coordinates specified are outside of grid
    bounds.");

        return grid.get(grid_x).get(grid_y);
    }

    private Cell<T> getGridCellForPos(double x, double y) {
        if (x < 0.0 || x >= this.spaceWidth || y < 0 || y >= this.spaceHeight)
            throw new IllegalArgumentException("Coordinates specified are outside of grid
    bounds.");

        int grid_x = (int) Math.floor(x / cellWidth);
        int grid_y = (int) Math.floor(y / cellWidth);

        return grid.get(grid_x).get(grid_y);
    }

    @Override
    public Iterator<T> binIterator(int grid_x, int grid_y) {
        List<T> results = binSearch(grid_x, grid_y);
        return results.iterator();
    }

    @Override
    public Iterator<T> rangeSearchIterator(DoublePoint upper, DoublePoint lower) {
        return null;
    }

    final class RadiusSearchIterator implements Iterator<T> {
        LinkedList<Cell<T>> cellStack = new LinkedList<>();
        LinkedList<T> agentStack = new LinkedList<>();
        T next;
        DoublePoint center;
        Double radiusSquared;
        int offsetOfNumOfBinsToChk;
        int grid_x;
        int grid_y;

        RadiusSearchIterator(DoublePoint center, Double radius) {
            this.center = center;
            this.radiusSquared = Math.pow(radius, 2.0);
            this.offsetOfNumOfBinsToChk = (int) Math.ceil(radius / binWidth);
            this.grid_x = (int) Math.floor(center.get(0) / binWidth);
            this.grid_y = (int) Math.floor(center.get(1) / binWidth);

            for (int i = grid_x - offsetOfNumOfBinsToChk; i <= grid_x +
    offsetOfNumOfBinsToChk; i++) {
                for (int j = grid_y - offsetOfNumOfBinsToChk; j <= grid_y +
    offsetOfNumOfBinsToChk; j++) {
                    if (i >= 0 && i < binDimX && j >= 0 && j < binDimY) {
                        cellStack.push(getGridCellForIndex(i, j));
                    }
                }
            }

            next = nextInRange();
        }

        private T nextInRange() {
            while(agentStack.isEmpty()) {
                if(cellStack.isEmpty()) {
                    return null;
                } else {
                    Cell<T> cell = cellStack.pop();

                    for (T other: cell.items) {
                        if (center.distanceSquared(other.getCoordinates()) <= radiusSquared)
                            agentStack.push(other);
                    }
                }
            }

            return agentStack.pop();
        }

        @Override
        public boolean hasNext() {
            return next != null;
        }

        @Override
        public T next() {
            T last = next;
            next = nextInRange();
            return last;
        }

        @Override
        public void remove() {
            //TODO: possibly implement this
            throw new UnsupportedOperationException();
        }

        @Override
        public void forEachRemaining(Consumer<? super T> action) {
            //TODO: implement this
        }
    }

    @Override
    public Iterator<T> radiusSearchIterator(DoublePoint center, Double radius) {
        return new RadiusSearchIterator(center, radius);
    }

    @Override
    public String toString() {
        return super.toString();
    }

    private static class Cell<S> implements Serializable {
        ArrayList<S> items;

        Cell() {
            this.items = new ArrayList<>();
        }

        @Override
        public String toString() {
            return super.toString();
        }

        /**
         * This number is here for model snapshot storing purpose<br>
         * It needs to be changed when this class gets changed
         */
        private static final long serialVersionUID = 2L;
    }

    /**
     * This number is here for model snapshot storing purpose<br>
     * It needs to be changed when this class gets changed
     */
    private static final long serialVersionUID = 2L;

}
```

A.7 Logger

```java
/**
 * Logger
 */

import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.IOException;

public class Logger implements Serializable {
    private PrintWriter printWriter;

    /**
     * Default constructors
     */
    public Logger(String fileName) {
        this(new File(fileName));
    }

    public Logger(File file) {
        try {
            printWriter = new PrintWriter(new FileWriter(file), true);
        } catch(IOException e) {
            System.err.println("Logger: Could not create PrintWriter: " + e);
            System.exit(1);
        }
    }

    public void log(String text) {
        printWriter.println(text);
    }

    public void close() {
        printWriter.close();
    }

    @Override
    public String toString() {
        return super.toString();
    }

    /**
     * This number is here for model snapshot storing purpose<br>
     * It needs to be changed when this class gets changed
     */
    private static final long serialVersionUID = 2L;

}
```