ABSTRACT

WOLCZYNSKI, JUSTIN M. The Optimal Assignment of Spontaneous Volunteers. (Under the direction of Dr. Maria E. Mayorga.)

In the wake of a disaster, spontaneous volunteers are ordinary citizens who impulsively volunteer their time to assist disaster relief efforts. These volunteers are a tremendous resource but pose a difficult logistical challenge. Unlike classical labor assignment problems, those tasked with the management of spontaneous volunteers are faced with great uncertainty regarding their labor force. Volunteer managers cannot know how many spontaneous volunteers will arrive, when they will arrive, or for how long they will stay. We model this problem as a multi-server queueing system with two queues, and both stochastic server arrival and abandonment. Upon arrival, these servers (volunteers) are immediately assigned to one of two queues, each with its own amount of deterministic and nonincreasing work to be done. Volunteers at the same queue work in parallel. After a volunteer begins work at a particular queue, they then abandon the system according to a stochastic process. Interarrival, abandonment, and service times are all assumed Exponential. We argue that, because of the assumption of static and deterministic demand, this model would most appropriately be applied to the relatively stable work associated with long-term recovery efforts, such as debris-clearance.

We frame this system as a discrete time Markov Chain with the objective of maximizing reward by completing deterministic units of demand. We employ policy iteration to generate optimal policies for different cases of identical and dissimilar queues. We define common-sense heuristic policies, including a Join-the-Shortest-Queue analog called the Fewest Volunteers policy which assigns incoming volunteers to the queue with the fewest number of volunteers currently working. We show that the Optimal Policy is, for the most part, a mixture of the heuristic policies defined here, but would be impractical to implement. We show but do not prove that the Optimal Policy exhibits a logical switching-curve structure. We develop an Arena model to simulate this system, using time to complete all work as the performance metric. We show through paired-sample comparisons that, although the Fewest Volunteers policy is not optimal, it performs very well relative to the Optimal Policy in all cases considered. We perform distribution-fitting analysis on work completion times and find that a number of distributions make for remarkably good and consistent fits. We repeat the paired-sample comparisons and distribution-fitting analysis using Weibull interarrival times instead of Exponential and find that Weibull interarrivals do not make a significant difference in terms of either the performance of the optimal and heuristic policies, or in terms of the distribution fitting results. We argue that while the Optimal Policy is impractical, simulation suggests that the Fewest Volunteers heuristic policy makes a good and practical alternative for the spontaneous volunteer assignment problem when the only goal is to complete work efficiently.

The Optimal Assignment of Spontaneous Volunteers

by
Justin M. Wolczynski

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Operations Research

Raleigh, North Carolina

2015

APPROVED BY:

James R. Wilson

Michael G. Kay

Dr. Maria E. Mayorga
Chair of Advisory Committee

DEDICATION

This thesis is dedicated to
Walter “Papa” Wolczynski
and
Eva “Mima” Bauer,
for their endless love and support.

BIOGRAPHY

Justin Wolczynski was born and raised in New Jersey. After starting at Raritan Valley Community College in Branchburg, NJ, he transferred to Ithaca College in Ithaca, NY where he obtained his B.A. in Mathematics in 2012. In the Fall of 2013, Justin entered the MSc program in Operations Research at North Carolina State University. His research interests include dynamic programming, stochastic optimization, and queueing theory.

ACKNOWLEDGEMENTS

I would like to take this opportunity to show my appreciation for my wonderful advisor, Dr. Maria E. Mayorga, to whom I owe a tremendous amount. Since I met her she has been a consistent and dependable source of both guidance and contagious enthusiasm. Her support has been essential and I am forever grateful.

I am grateful to my committee members Dr. James R. Wilson and Dr. Michael G. Kay for agreeing to read and review this thesis and for all their help over the course of its construction.

I would like to offer a special thank you to Dr. James R. Wilson for his enormous assistance with the development of the Arena model.

I would also like to thank Dr. Emmett J. Lodree and Dr. Julie Casani for their invaluable contributions and motivation.

Lastly, I would like to thank my family. To my brother, Brent, for subsidizing my coffee needs and lifting my spirits with Simpsons stills; to my parents for always being supportive and understanding; and to Papa and Mima for making my education possible. Thank you.

TABLE OF CONTENTS

LIST OF TABLES	vii
LIST OF FIGURES	viii
Chapter 1 Introduction	1
Chapter 2 Literature Review	3
2.1 Operations Research work in Emergency Management	3
2.2 Volunteerism	4
2.2.1 Spontaneous Volunteerism	4
2.2.2 Optimal Management of Volunteers	5
2.3 The Join the Shortest Queue rule	7
Chapter 3 Solution Methods and Structure	8
3.1 Formulation	8
3.2 Solution Method	12
3.3 Optimal Policy Structure	14
3.4 Heuristic Policies	17
3.4.1 Comparison with Optimal Policies	18
Chapter 4 Simulation Results and Analysis	21
4.1 Arena Model Description	22
4.2 Paired-sample Comparisons	23
4.2.1 Exponential interarrival times	24
4.2.2 Non-exponential interarrival times	27
4.3 Policy Differences	29
4.3.1 Exponential interarrivals	30
4.3.2 Non-exponential interarrival times	32
4.4 Distribution Fitting	34
4.4.1 Exponential interarrivals	34
4.4.2 Non-exponential interarrivals	35
4.4.3 Summary	37
Chapter 5 Conclusion	46
5.1 Summary	46
5.2 Policy Implications	47
5.3 Limitations	48
5.4 Future Work	48
Bibliography	50
APPENDICES	51
Appendix A Policy Iteration Algorithm	52
Appendix B Arena Model Documentation	68
B.1 Upper Level	68

B.2 Lower Level 75

B.2.1 Work Completion Track 79

B.2.2 Departure Track 82

LIST OF TABLES

Table 3.1	Case parameters.	14
Table 3.2	Policy iterations and runtimes.	14
Table 3.3	Optimal policy vs. heuristic policy disagreements.	19
Table 4.1	Case parameters.	24
Table 4.2	Paired-sample project completion time results for Exponential interarrivals over 1,000 runs. Boldface indicates statistical significance.	25
Table 4.3	Paired-sample activity one completion time results for Exponential interarrivals over 1,000 runs. Boldface indicates statistical significance.	26
Table 4.4	Paired-sample activity two completion time results for Exponential interarrivals over 1,000 runs. Boldface indicates statistical significance.	27
Table 4.5	Case parameters for non-Exponential interarrival analysis.	29
Table 4.6	Paired-sample project completion time results for Weibull interarrivals over 1,000 runs. Boldface indicates statistical significance.	30
Table 4.7	Paired-sample activity one completion time results for Weibull interarrivals over 1,000 runs. Boldface indicates statistical significance.	31
Table 4.8	Paired-sample activity two completion time results for Weibull interarrivals over 1,000 runs. Boldface indicates statistical significance.	32
Table 4.9	Optimal policy vs. heuristic policy disagreements.	33
Table 4.10	Policy differences over 1,000 replications with exponential interarrivals.	33
Table 4.11	Policy differences.	34
Table 4.12	Policy differences over 1,000 replications with Weibull interarrivals.	34
Table 4.13	Distribution fitting results for project completion times with Exponential interarrivals. Sample size: 1,000.	39
Table 4.14	Distribution fitting results for activity one completion times with Exponential interarrivals. Sample size: 1,000.	40
Table 4.15	Distribution fitting results for activity two completion times with Exponential interarrivals. Sample size: 1,000.	41
Table 4.16	Distribution fitting results for project completion times with Weibull interarrivals. Sample size: 1,000.	42
Table 4.17	Distribution fitting results for activity one completion times with Weibull interarrivals. Sample size: 1,000.	43
Table 4.18	Distribution fitting results for activity two completion times with Weibull interarrivals. Sample size: 1,000.	44

LIST OF FIGURES

Figure 3.1	Graphical representation of the system.	9
Figure 3.2	Solution plots for Case 1.	16
Figure 3.3	Solution plots for Case 2.	17
Figure 4.1	Graphical representation of the system dynamics.	22
Figure 4.2	CDF plots for the Exponential and Weibull fits.	28
Figure 4.3	Project completion distribution fitting results for Case 1. (a) PDF, Optimal; (b) CDF, Optimal; (c) PDF, FV; (d) CDF, FV.	36
Figure 4.4	Project completion distribution fitting results for Case 2. (a) PDF, Optimal; (b) CDF, Optimal; (c) PDF, FV; (d) CDF, FV.	37
Figure 4.5	Project completion distribution fitting results for Case 10. (a) PDF, Optimal; (b) CDF, Optimal; (c) PDF, FV; (d) CDF, FV.	38
Figure 4.6	Project completion distribution fitting results for Case 13. (a) PDF, Optimal; (b) CDF, Optimal; (c) PDF, FV; (d) CDF, FV.	45
Figure B.1	Flow of volunteers, upper level model, overview.	70
Figure B.2	Flow of volunteers, upper level model, detail 1.	71
Figure B.3	Flow of volunteers, upper level model, detail 2.	72
Figure B.4	Read assignment policy.	73
Figure B.5	Track time until the end of run.	74
Figure B.6	Work and departure process, lower level model, overview.	76
Figure B.7	Work and departure process, lower level model, split between work and departure.	78
Figure B.8	Work and departure process, lower level model, Work Completion track.	81
Figure B.9	Work and departure process, lower level model, Departure track.	83

Chapter

1

Introduction

It is an unfortunate fact of life that disasters, of all kinds, are inevitable. They can occur naturally or as a result of human action. Whatever the cause, disasters are large, chaotic, and unpredictable events that can leave behind enormous costs in terms of life lost and both structural and environmental damages. The field of Disaster Operations Management (DOM), as described in Altay and Green (2006), can be divided into four phases: Mitigation, Preparedness, Response, and Recovery. While Mitigation and Preparedness refer to actions taken before a disaster strikes in order to mitigate damages, Response and Recovery refer to those actions taken in the wake of a disaster to address immediate and grave needs and then return the community to a state of normalcy. According to Galindo and Batta (2013), the Mitigation, Preparedness, and Response stages have all received varying but significant attention over the years in the OR/MS field, while the Recovery stage has received considerably less focus, representing just 3.4% of the 155 papers surveyed from 2005 to 2010. One part of the Recovery stage of DOM is the crucial work that spontaneous volunteers perform.

Spontaneous volunteers are people who, in the wake of a disaster, impulsively contribute to the disaster response and recovery without affiliations to recognized volunteer organizations (e.g., the Red Cross) or other typical first responders. These people constitute a labor source which is at once invaluable and hard to manage. The spontaneous volunteer assignment problem is distinct from labor assignment problems in that organizers cannot know how many such volunteers will arrive,

when they will arrive, or for how long they will stay. The optimal assignment of incoming spontaneous volunteers is the focus of this thesis.

In this thesis we will abstract the spontaneous volunteer assignment problem as a queueing system with two queues, fixed demand at each of the queues, and with stochastically arriving servers who stochastically abandon the system after beginning work. The work in this case can be thought of as work which requires no special training, for example debris clearance. The goal of this thesis is to develop a policy which assigns incoming volunteers to one of two queues in order to maximize benefit gained by completing units of work. It has been well-established over the years that in a multi-server $M/M/s$ queueing system with stochastically arriving customers and identical queues, the Join-the-Shortest-Queue (JSQ) rule is optimal in that it stochastically minimizes the number of customers in the system. An analogous rule for the spontaneous volunteer assignment problem would be one which assigns an incoming volunteer to the queue with the fewest number of volunteers currently working. One might expect that such an analogous rule would be optimal in the spontaneous volunteer assignment problem, but we will show through simulation and Policy Iteration that this is not the case.

In Chapter 2 we will review research relevant to this discussion, including work on DOM, volunteerism, and the optimality of the JSQ rule. In Chapter 3 we will provide a formal definition for the problem being examined, introduce notation, explain how optimal policies were generated, define heuristic policies to be tested against the optimal policies, and then examine the structure of the optimal policies. In Chapter 4, we will document the Arena model used to simulate the system, provide paired-sample comparison results between optimal policies and heuristic policies, and then examine the similarities between the optimal policies and the heuristic policies. Next we will provide distribution-fitting results on the completion times when following various policies. This analysis will be repeated for cases with non-Exponential interarrival times. Finally, in Chapter 5 we will conclude with a summary of our results and their limitations, along with a discussion of possible policy implications.

Chapter

2

Literature Review

2.1 Operations Research work in Emergency Management

The field of DOM has received a lot of attention in a number of disciplines, including operations research. Altay and Green (2006) produced a survey of OR/MS literature on DOM spanning from 1980 to 2004, with the intent of identifying both research gaps and possible research directions. In surveying the literature, the authors used several different classification categories, including the stage of the disaster life cycle with which the research was concerned. The authors describe four distinct phases of the disaster life cycle: Mitigation, Preparedness, Response, and Recovery. *Mitigation* refers to actions taken prior to a disastrous event in order avoid it or, failing that, lessen its effects on the community. Examples include barrier construction, the improvement of building codes, and risk analysis. *Preparedness* refers to actions taken to prepare a community so that it can better respond in the aftermath of a disaster. Examples include public education campaigns, maintenance of emergency supplies, and disaster training exercises. *Response* refers to the emergency procedures taken in the immediate wake of a disaster to serve dire needs and to limit damages of all kinds. Examples include search and rescue missions, fire fighting, and the evacuation of vulnerable populations. Lastly, *Recovery* refers to long-term actions initiated after the immediate wake of the disaster, with the goal of returning the affected community back to normal conditions. Examples

include debris clearance, the rebuilding of infrastructure, and financial assistance. Of the 109 articles surveyed, 44.0% were on Mitigation, 21.1% were on Preparedness, and 23.9% were on Response. Lastly, just 11.0% of articles surveyed were focused on Recovery, and only one was from a mainstream OR/MS outlet. The authors note that "the area in dire need for more research is, without question, disaster recovery."

Altay and Green (2006) served as inspiration for Galindo and Batta (2013), another survey of OR/MS literature which picks up where Altay and Green (2006) left off. The articles surveyed in Galindo and Batta (2013) span from 2005 to 2010, and the authors use a similar classification scheme as Altay and Green (2006), along with other novel additions, such as an examination of the common assumptions used and a judgment of their validity in a disaster relief context. According to Galindo and Batta (2013), of the 155 articles surveyed, 23.9% were on Mitigation, 28.4% were on Preparedness, 33.5% were on Response, and just 3.2% were on Recovery. Additionally, the authors include another category, *Multiple stages*, for articles which addressed multiple stages of the disaster life cycle. These articles represented 11.0% of those surveyed. It is the conclusion of the authors that "most of the gaps observed by Altay and Green (2006) still remain." Both Altay and Green (2006) and Galindo and Batta (2013) found a gap in the OR/MS literature focused on the Recovery phase of the disaster life cycle, a gap which has persisted from 1980 to 2010. It is our belief that the problem of spontaneous volunteer assignment examined in this thesis is most appropriately applied to the Recovery phase of the disaster life cycle.

In examining the common assumptions made in OR/MS articles on DOM, Galindo and Batta (2013) note the assumption of "static parameters (demand, travel times, costs, etc.)." The authors label this assumption as unrealistic because "costs, demand, travel times, etc. may change as disaster relief evolves." In the volunteer assignment problem examined here, we make the assumption of deterministic (i.e., static) demand at each of the queues. Therefore, we believe that the system examined here would most appropriately be applied to the relatively stable work associated with the Recovery phase of the disaster life cycle, an example of which is debris clearance.

2.2 Volunteerism

2.2.1 Spontaneous Volunteerism

The primary motivation for this work is spontaneous volunteerism in the wake of disasters. Lowe and Fothergill (2002) define spontaneous volunteers as civilians who impulsively contribute to the disaster response without previous affiliations to recognized volunteer organizations (e.g., the Red Cross) or other typical first responders, such as firefighters and EMS workers. A large number of

spontaneous volunteers are typically from the areas affected by the disaster, and are therefore in a unique position to provide assistance. These citizens are available to offer assistance in the immediate wake of a disaster, and often have intimate knowledge of the affected areas, making them invaluable for search-and-rescue missions. On the other hand, both EMS services and aid organizations may be delayed in responding due to the nature of the disaster and may not have the same familiarity with the affected communities.

Historically, spontaneous volunteers have served as a tremendous resource for disaster response operations. Glass (2001) notes that at the site of the Nimitz Freeway collapse, a direct result of the Loma Prieta Earthquake, the bulk of the people rescued were saved by civilian bystanders and not by EMS. According to Lowe and Fothergill (2002), in the wake of 9/11, spontaneous volunteers performed many crucial tasks: "they cheered on the Ground Zero workers, gave massages, prepared food, made beds, gave blood, worked as translators, vacuumed rooms, fed workers, counseled, did research, and helped with supplies for the Red Cross." During the first three weeks after 9/11, the Red Cross processed over 15,570 such volunteers. Barnett and Flint (2007) note that in the wake of Hurricane Katrina, there was a tremendous need for local volunteers to perform immediate post-disaster work. These local volunteers "were the rescuers, caretakers and, in many cases, the final comforting companions to the dying. They were the first, and often the only, line of response that would exist for weeks." In addition to comforting the victims of disaster, Fernandez et al. (2005) note the belief that the act of volunteering is helpful to the volunteers themselves. For community members who were affected by the disaster, volunteering their time can serve "as an avenue to reduce stress" and "as an outlet for rage."

While spontaneous volunteers make invaluable contributions to relief efforts, managing them is a harrowing task, and often a burden. The spontaneous volunteer assignment problem is distinct from labor assignment problems and has its own set of unique challenges. Falasca and Zobel (2012) and Sampson (2006) note that volunteers may prefer to work with certain groups or prefer some assignments over others, and their preferences must be given greater weight than in typical employee assignment problems. Volunteers may have skills conducive to one task but not another. Volunteers who are not put to work or are overworked are less likely to volunteer in the future, and this creates an indirect cost consideration. Additionally, relief organizers can never know how many spontaneous volunteers will arrive, when they will arrive, or for how long they will stay. This last set of complications is the focus of this thesis.

2.2.2 Optimal Management of Volunteers

Fernandez et al. (2005) note that volunteer efforts are sometimes ineffective or inefficient because systems do not exist to handle the influx of volunteers. Therefore, management of these volunteers is

often done ad hoc, and those tasked with their management are forced to remove focus from their primary duties. Spontaneous volunteers are at once a tremendous resource and a challenging burden, so it is important that systems are established to better manage them. A fair amount of work has been done regarding the optimal assignment of volunteers, but most do not allow for stochastic arrivals and departures, instead focusing on balancing objectives with the labor considerations which are unique to the volunteer assignment problem.

Fernandez et al. (2005) provide an overview of the challenges faced by emergency managers with respect to managing volunteers and provide a broad roadmap for effectively handling the influx of volunteers and using them efficiently. Sampson (2006) examines the differences between traditional labor assignment problems and the volunteer assignment problem. Sampson (2006) and Falasca and Zobel (2012) both provide models for efficiently assigning volunteers to tasks. However, both models are focused on balancing potentially conflicting objectives, like work completion, volunteer satisfaction, and volunteer retention. Additionally, both models assume full information about the volunteer labor pool.

The closest work to the problem being examined here is that of Lodree et al. (2015) which is an empirical study of volunteer convergence in the wake of the April 2011 tornado in Tuscaloosa, Alabama. The authors found the distribution of interarrival times of volunteers to not be homogenous over time. Using a clustering algorithm, the authors were able to divide interarrivals into disjoint groups such that, within each group, the interarrivals were homogeneous. The distribution-fitting work done in Lodree et al. (2015) will serve as the basis of simulations done in this thesis using non-Exponential interarrival times. In addition to the distribution-fitting analysis, Lodree et al. (2015) also models the spontaneous volunteer assignment problem as a queueing system in a disaster relief warehouse environment, with stochastic server arrivals and departures, and stochastic customer arrivals. Their model allows for group arrivals of volunteers as well as stochastically arriving customers. The authors test the performance of various heuristic policies for volunteer assignment via simulation. The problem examined here differs from that of Lodree et al. (2015) because we assume deterministic demand, which is more appropriate for situations like debris-clearance, and we do not allow for group arrivals of volunteers. The problem examined in this thesis is in some sense a more tractable version of the model in Lodree et al. (2015) thus allowing us to solve it as a Markov Decision Process (MDP).

Other research relevant to the spontaneous volunteer assignment problem examined here is work on the optimality of the JSQ rule, which will serve as inspiration for one of the heuristic policies defined in Chapter 3. The JSQ rule is concerned with assigning incoming customers in a queueing system to the queue with the fewest number of customers. An analogous rule for our purposes is one that assigns an incoming server to the queue with the fewest number of servers. In the next section we will examine literature on the optimality of the JSQ rule.

2.3 The Join the Shortest Queue rule

Consider an $M/M/s$ queueing system with multiple queues, and one identical server at each queue. Further, assume that the service rate at a particular queue is not dependent on the number of customers. The JSQ rule assigns incoming customers to the queue with the fewest number of customers. It has been well established over the years that in the situation described above, the JSQ rule is optimal in that it stochastically minimizes the total number of customers in the system at each point in time. The spontaneous volunteer assignment problem modeled here is one in which identical servers are entering and abandoning the system exponentially, and the "customers" are represented by deterministic demand at each of the queues. Instead of assigning customers to queues, we are assigning incoming servers to queues so as to maximize the benefit gained by completing work. The spontaneous volunteer assignment analog to the JSQ rule would be a rule which assigns incoming volunteers to the queue with the fewest number of volunteers currently working, a rule which we will denote by the Fewest Volunteers (FV) policy. One major difference between the $M/M/s$ queueing system described above and the spontaneous volunteer assignment problem examined here is that the total service rate at any given queue is proportional to the number of servers working at that queue, so we have state-dependent service rates. Johri (1989) proves that if we allow service rates which depend on the number of customers at queues in the $M/M/s$ queueing system described above, then the JSQ rule is optimal as long as: 1. the service rates are non-decreasing in the number of customers; and 2. the increase in service is nonincreasing in the number of customers. In Johri (1989), the state-dependent service rates depend on the number of customers and not on the number of servers, as only one server is allowed at each queue; therefore, it is not a perfect analogy to the system examined here.

However, Akgun et al. (2011) extend the optimality of the JSQ rule for flexible arrivals to the following distinct cases: arbitrary arrivals, multiple-server stations, and exponential customer abandonment. The system examined here is one in which there are two multiple-server stations and exponential server abandonment. The situations examined in Akgun et al. (2011) are closer analogies to the spontaneous volunteer assignment model developed here. One might expect that, based on the results of Akgun et al. (2011), the analogous FV rule would be optimal in the spontaneous volunteer assignment problem in that it would maximize benefit gained by completing work. We will show in Chapter 5 that, although the FV rule performs well relative to the optimal solution when tested via simulation, Policy Iteration shows that it is not optimal. Additionally, in light of the extension to arbitrary arrivals found in Akgun et al. (2011), we will perform the same simulation analysis but with Weibull-distributed interarrivals.

Chapter

3

Solution Methods and Structure

In this chapter we formally define the problem being examined, introduce the notation involved, discuss the methods used to generate optimal policies, define heuristic policies, and then examine the structure of the optimal policies.

3.1 Formulation

We frame this problem as a queueing system in which there are two queues, each with some deterministic amount of work to be done, starting at D_i , where i refers to one of the two queues, either 1 or 2. Each queue has a maximum capacity for the number of volunteers that can work there at a time, designated by V_i . Volunteers, the servers, arrive to the system randomly with their interarrival times being Exponentially-distributed with rate λ . Upon arrival, each volunteer is assigned according to the given control policy to either Queue 1 or Queue 2, where they then begin work with exponential service rate μ_1 or μ_2 , depending on the queue. Volunteers can only be assigned to queues which have nonzero demand. If a volunteer is assigned to a queue that is at maximum capacity, it is assumed that the volunteer will go elsewhere to assist with work that is unrelated to the two queues in this system, and so an assignment to a full queue amounts to a fictitious transition. These assignments are made instantly. Volunteers at a particular queue work in parallel with each other, and so the total service

rate at a particular queue is equal to the number of volunteers working at the queue multiplied by the queue's service rate. If we define $v_i(t)$ as the number of volunteers currently working at Queue i at time t , then the total service rate at that queue at time t is $v_i(t)\mu_i$. Once a volunteer begins work at one of the queues, the volunteer will abandon the system after a random amount of time. Each volunteer's abandonment time is determined according to an exponential distribution with rate γ . A holding cost H is incurred for unfinished units of work, and a reward R is gained for the completion of each unit of work. A graphical representation of this system is found in Figure 3.1.

Figure 3.1 Graphical representation of the system.

The state variables are as follows:

$$\begin{aligned} v_i(t) &= \text{The number of volunteers working at Queue } i \text{ at time } t, \\ &\text{where } v_i(t) \in [0, 1, \dots, V_i], \\ d_i(t) &= \text{The number of unfinished work units remaining at Queue } i \text{ at time } t, \\ &\text{where } d_i(t) \in [0, 1, \dots, D_i], \end{aligned}$$

Therefore, given a Markov deterministic policy u , the state of the system at time t while following policy u is given by $(v_1^u(t), v_2^u(t), d_1^u(t), d_2^u(t))$. The state transitions are all Markovian and therefore this system can be formulated as a continuous time Markov chain. In order to solve this problem we employ uniformization, as detailed in Bertsekas (2001), to create an equivalent discrete time system. We define the maximum transition rate as $\beta = \lambda + V_1\mu_1 + V_2\mu_2 + (V_1 + V_2)\gamma$. We use the following notation:

- λ : the volunteer arrival rate;
- γ : the abandonment rate of each volunteer;
- i : refers to a specific queue for $i \in \{1, 2\}$;
- μ_i : the individual service rate at Queue i ;
- d_i : the number of units of work remaining at Queue i ;
- D_i : initial demand at Queue i ;
- v_i : the number of volunteers currently working at Queue i ;
- V_i : the maximum allowable number of volunteers at Queue i ;
- $\mathcal{A}$: the set of allowable assignments, queues where $d_i > 0$, and $v_i < V_i$;
- $\mathcal{S}$: the set of possible states;
- R : the reward obtained for completing a unit of demand; and
- H : the holding cost applied to unfinished demand.

If we define k as the number of periods left-to-go, where $k \in [0, \dots, K]$, then the recursive optimality equation for the finite horizon problem is given by

$$\begin{aligned}
G_k(v_1, v_2, d_1, d_2) = & \frac{1}{\beta} \left[-H(d_1 + d_2) \right. \\
& + \lambda \max_{j \in \mathcal{A}} G_{k-1}(A_j(v_1, v_2, d_1, d_2)) \\
& + \sum_{i=1}^2 \mu_i v_i [G_{k-1}(W_i(v_1, v_2, d_1, d_2)) + \beta R] \\
& + \sum_{i=1}^2 \gamma v_i G_{k-1}(D_i(v_1, v_2, d_1, d_2)) \\
& \left. + (\beta - \lambda - \sum_{i=1}^2 v_i(\mu_i + \gamma)) G_{k-1}(v_1, v_2, d_1, d_2) \right] \tag{3.1}
\end{aligned}$$

where A_j , W_i and D_i are operators defined as

$$\begin{aligned}
A_j(v_1, v_2, d_1, d_2) &= \begin{cases} (v_1 + 1, v_2, d_1, d_2) & \text{if } j = 1, \\ (v_1, v_2 + 1, d_1, d_2) & \text{if } j = 2, \\ (v_1, v_2, d_1, d_2) & \text{if } \mathcal{A} = \emptyset, \end{cases} \\
W_i(v_1, v_2, d_1, d_2) &= \begin{cases} (v_1, v_2, d_1 - 1, d_2) & \text{if } i = 1, d_1 > 1, \\ (v_1, v_2, d_1, d_2 - 1) & \text{if } i = 2, d_2 > 1, \\ (0, v_2, d_1 - 1, d_2) & \text{if } i = 1, d_1 = 1, \\ (v_1, 0, d_1, d_2 - 1) & \text{if } i = 2, d_2 = 1, \end{cases} \\
D_i(v_1, v_2, d_1, d_2) &= \begin{cases} ((v_1 - 1)^+, v_2, d_1, d_2) & \text{if } i = 1, \\ (v_1, (v_2 - 1)^+, d_1, d_2) & \text{if } i = 2. \end{cases}
\end{aligned}$$

The first row of (3.1) accounts for the holding cost incurred at each stage, where the cost is proportional to remaining demand. The second row accounts for the arrival of a volunteer and the decision to send them to one queue or the other. The third row accounts for the completion of units of work, where the rate at which work is completed at a particular queue is proportional to the number of volunteers currently working at that queue. Additionally, an instantaneous reward is obtained every time a unit of work is completed. The fourth row accounts for the abandonment of volunteers. The rate of this abandonment is proportional to the number of volunteers currently working at each queue. And finally, the fifth row accounts for the fictitious transition.

We define the optimal action at stage k as

$$a_k(v_1, v_2, d_1, d_2) = \arg \max_{j \in \mathcal{A}} G_{k-1}(A_j(v_1, v_2, d_1, d_2)).$$

In order to solve this discrete time MDP via Policy Iteration, we must translate the finite-horizon problem described above to an infinite-horizon problem. To that end, let u refer to a stationary policy. For all states $s \in \mathcal{S}$, the action to be taken upon an arrival when in state s , dictated by policy u , is $u(s)$. The possible actions are: assign to Queue 1, assign to Queue 2, or assign to one of the two queues randomly with equal probability. These actions are represented by 1, 2, and 3, respectively. We define $G_k^u(s)$ to be the total expected reward when starting from state $s = (v_1, d_1, v_2, d_2)$ with k periods left-to-go when following policy u .

$$\begin{aligned} G_k^u(s) = & \frac{1}{\beta} \left[-H(d_1 + d_2) \right. \\ & + \lambda G_{k-1}^u(A_{u(s)}(s)) \\ & + \sum_{i=1}^2 \mu_i v_i [G_{k-1}^u(W_i(s)) + \beta R] \\ & + \sum_{i=1}^2 \gamma v_i G_{k-1}^u(D_i(s)) \\ & \left. + (\beta - \lambda - \sum_{i=1}^2 v_i (\mu_i + \gamma)) G_{k-1}^u(s) \right]. \end{aligned} \quad (3.2)$$

Given (3.2), the infinite-horizon expected reward when starting from state s would be $\lim_{k \rightarrow \infty} G_k^u(s)$. In most cases, infinite-horizon expected reward is in fact infinite. But in this case it converges. This is because both the state space and action space are finite, and there is an absorbing state, $(0, 0, 0, 0)$. Every state $s \in \mathcal{S}$ can reach $(0, 0, 0, 0)$ in a finite number of steps, and as demand cannot increase, the system cannot leave state $(0, 0, 0, 0)$ once it enters it. Additionally, $G_k(0, 0, 0, 0) = 0$ for all k . These conditions are sufficient to ensure that the infinite-horizon expected reward converges to a finite value. This allows us to use Policy Iteration as our solution method without averaging or discounting.

3.2 Solution Method

In order to generate the optimal policy u^* we employ the Policy Iteration algorithm described below. The full Java code is made available in Appendix A.

Step 1 (Policy Definition): We begin by choosing an arbitrary stationary policy u which, for each

possible state, designates the assignment action to be taken upon the arrival of a volunteer. The possible actions are: assign to Queue 1, assign to Queue 2, or assign to one of the two queues randomly with equal probability. These actions are represented by 1, 2, and 3, respectively.

Step 2 (Initialization): Make the following assignment:

$$G_0^u(s) = 0, \text{ for all states } s \in \mathcal{S}.$$

Step 3 (Policy Evaluation): Use (3.2) to calculate the expected infinite-horizon reward under policy u by iteratively calculating $G_k^u(s)$ for all states s until

$$\max_{s \in \mathcal{S}} |G_k^u(s) - G_{k-1}^u(s)| < \epsilon, \quad (3.3)$$

where ϵ is the chosen error. Assign $K = k$, the first period for which (3.3) is true.

Step 4 (Policy Improvement): Once the infinite-horizon expected rewards have converged, we then check to see if policy u can be improved. For each state $s \in \mathcal{S}$, we find

$$a_{K+1}(s) = \arg \max_{j \in \mathcal{A}} G_K^u(A_j(s)).$$

If the optimal action matches that of policy u for all states s , then u is the optimal policy, u^* , and the algorithm ends. Otherwise, if any of the specified actions of u do not match the optimal action, then policy u is updated with those actions and the algorithm returns to Step 2.

The amount of time this algorithm takes to run is partly a function of the size of the state space, and the size of the state space is a function of V_1 , V_2 , D_1 , and D_2 . We assume when demand is reduced to zero at a particular queue that all of the volunteers currently working at that queue abandon the system. Therefore, states where there are volunteers but no demand at a particular queue are not included in the state space. Because of this, the size of the state space is not calculated as $(V_1 + 1)(V_2 + 1)(D_1 + 1)(D_2 + 1)$, as it would be if the state variables were allowed to vary freely across their respective ranges. The formula for the size of the state space is given by (3.4).

$$f(V_1, V_2, D_1, D_2) = D_2(D_1(V_1(V_2 + 1) + V_2 + 1) + V_2 + 1) + D_1(V_1 + 1) + 1. \quad (3.4)$$

The cases considered are found in Table 3.1. In every case, the arbitrary policy u chosen to begin the Policy Iteration algorithm was the Fewest Volunteers policy, to be defined later in Section 3.4. The Policy Iteration algorithm was written in Java, and was run in Eclipse Standard/SDK Version 4.4.0.20140612-0500, on a 2.83 GHz Intel(R) Core(TM)2 Quad CPU machine with 8.00 GB of RAM. The number of policy iterations and the runtimes of the Policy Iteration algorithm for each case are

presented in Table 3.2.

Table 3.1 Case parameters.

Case	V_1	V_2	D_1	D_2	λ	γ	μ_1	μ_2	H	R
1	5	5	10	10	1	0.5	2.4	2.4	1	10
2	5	5	10	10	1	0.5	2.4	2.0	1	10
3	5	5	10	10	1	0.5	2.4	1.6	1	10
4	3	5	7	10	1	0.5	2.4	2.4	1	10
5	3	5	7	10	1	0.5	2.4	2.0	1	10
6	3	5	7	10	1	0.5	2.0	2.4	1	10
7	3	5	7	10	1	0.5	2.4	1.6	1	10
8	3	5	7	10	1	0.5	1.6	2.4	1	10

Table 3.2 Policy iterations and runtimes.

Case	States	Iterations	Runtime
1	3721	3	5 m, 11 s
2	3721	3	4 m, 40 s
3	3721	4	6 m, 35 s
4	1769	4	1 m, 13 s
5	1769	3	0 m, 47 s
6	1769	3	0 m, 47 s
7	1769	3	0 m, 44 s
8	1769	4	0 m, 56 s

3.3 Optimal Policy Structure

In every case considered, the optimal policies generated by Policy Iteration have a logical switching-curve structure. This means that, for example, if the optimal action at state (v_1, v_2, d_1, d_2) is to assign an incoming server to the Queue 2, then the optimal action at $(v_1, v_2, d_1, d_2 + 1)$ is the same as at

(v_1, v_2, d_1, d_2) . This switching-curve structure is described by the following inequalities:

$$\begin{aligned} a_k(v_1, v_2, d_1, d_2) &\leq a_k(v_1, v_2, d_1, d_2 + 1), \\ a_k(v_1, v_2, d_1, d_2) &\leq a_k(v_1 + 1, v_2, d_1, d_2), \\ a_k(v_1, v_2, d_1, d_2) &\leq a_k(v_1, v_2, d_1 - 1, d_2), \\ a_k(v_1, v_2, d_1, d_2) &\leq a_k(v_1, v_2 - 1, d_1, d_2). \end{aligned}$$

This switching-curve structure can be viewed graphically by fixing d_1 and d_2 , and plotting the optimal decisions as v_1 and v_2 vary from 0 to their respective maxima V_1 and V_2 . We begin with the first case considered, where Queue 1 and Queue 2 are identical in every way.

$$V_1 = V_2 = 5, \quad D_1 = D_2 = 10, \quad \mu_1 = \mu_2 = 2.4, \quad \gamma = 0.5, \quad \lambda = 1.$$

The switching curve structure for this case is evidenced in Fig. 3.2. In these plots, a black square means the optimal decision is to assign an incoming volunteer to Queue 1, a white square means the optimal decision is to assign an incoming volunteer to Queue 2, and a grey square means it is a tie. It is clear from the plots that when $d_1 > d_2$, the policy favors sending incoming volunteers to Queue 1. When $d_1 = d_2$, it is optimal to assign an incoming volunteer to whichever queue has the fewest volunteers. Also note that when one queue is at maximum capacity and the other is not, for example when $v_1 = 5$ and $v_2 < 5$, it is always optimal to assign an incoming volunteer to the queue which is not full. This makes logical sense because assigning a volunteer to a queue which is at maximum capacity does not increase the number of volunteers at that queue. It is assumed that the volunteer will do work that is unrelated to d_1 and d_2 . In the context of the recursive optimality equation, an assignment to a full queue is essentially a fictitious transition.

Next we examine the second case. Here, the two queues are identical except for their service rates. That is,

$$V_1 = V_2 = 5, \quad D_1 = D_2 = 10, \quad \mu_1 = 2.4, \quad \mu_2 = 2.0, \quad \gamma = 0.5, \quad \lambda = 1.$$

The plots of the solutions, using the same values of d_1 and d_2 as in Fig. 3.2, are found in Fig. 3.3. Note that when $d_1 = d_2$ (c), the assignment rule is not as simple as whichever queue has fewer volunteers. Furthermore, it is no longer a tie along the full length of the $v_1 = v_2$ trajectory.

Although the optimal policies exhibit a logical switching-curve structure, these policies would prove difficult to implement for two reasons. First, the point at which the policy switches from assigning an incoming server to one queue instead of the other is difficult to predict. No equation has

Figure 3.2 Solution plots for Case 1.

been found to characterize that decision process. Second, the Optimal Policy is a function of both the number of servers at each queue as well as the number of units of work left at each queue. In the context of volunteer assignment, the number of servers (volunteers) at each queue would be simple to ascertain for an organized operation, as it would amount to a headcount. On the other hand, the number of units of work left at each queue is not a simple matter. "Units of work" is a necessary but imprecise abstraction. Even if one quantified what a unit of work meant (e.g. x tons of debris), getting a measure of the amount at a particular queue would be difficult, depending on the work being done. Additionally, according to Galindo and Batta (2013), our assumption of static, deterministic demand

Figure 3.3 Solution plots for Case 2.

is potentially unrealistic. For these reasons we are forced to conclude that the optimal policies for the problem described here are unfortunately impractical. This leads us to three common-sense heuristic policies which we used to test as alternatives to the Optimal Policy.

3.4 Heuristic Policies

We define here three common-sense heuristic policies to test as alternatives to the Optimal Policy which was shown in the previous section to be impractical. If one were in charge of assigning incoming

spontaneous volunteers in this scenario, one might assign incoming volunteers to wherever demand is largest, or one might assign them to wherever the number of currently working volunteers is the lowest. Otherwise one might ignore both of those criteria and just assign incoming volunteers randomly. We define the following heuristic policies:

Largest Demand (LD) In this policy, an arriving volunteer is assigned to the queue with the largest amount of remaining demand. In the event of a tie in demand, the volunteer is sent where there are fewer volunteers.

Fewest Volunteers (FV) In this policy, an arriving volunteer is assigned to the queue with the fewest number of volunteers. In the event of a tie in the number of volunteers, the volunteer is sent where demand is largest.

Random (R) In this policy, an arriving volunteer is assigned to one of the two queues with equal probability.

In each of these policies, the following rules take precedence over the overarching assignment policy:

1. If $j \notin \mathcal{A}$, the volunteer will be assigned to $i \neq j$.
2. If $v_j = V_j$ and $v_i < V_i$ with $i \neq j$, then the volunteer will be assigned to i .
3. If $v_1 = V_1$ and $v_2 = V_2$, then the volunteer will be assigned to work unrelated to queues 1 and 2, amounting to a fictitious transition.
4. If a state is tied both in the number of volunteers and the amount of demand at each location, then the volunteer is assigned randomly with equal probability to one of the two locations.

These rules ensure that the heuristic policies do not make irrational or infeasible assignments.

3.4.1 Comparison with Optimal Policies

Policy Iteration shows that for every case examined, neither the Largest Demand policy nor the Fewest Volunteers policy is optimal. The pitfalls of each of the policies can be seen by example. Assume that we are considering the first case, where the two queues are identical in every way. Consider the state $(0, 4, 9, 10)$. That is, the demand at Queue 1 has barely been started and no one is currently working there. Meanwhile, at Queue 2, no work has been completed but there are four volunteers working there already, nearly at that queue's capacity. In this situation, it seems logical to assign an incoming volunteer to Queue 1 because the disparity in the number of volunteers at each queue in some sense outweighs the small disparity in remaining demand. That is, the Fewest Volunteers policy makes sense. Now consider the state $(3, 4, 1, 10)$. In this situation, the work is nearly completed at Queue 1

and there are still three volunteers working there, which will probably be enough to finish the work. Meanwhile, the work at Queue 2 is still at its maximum starting point. In this case, it seems more logical to assign an incoming volunteer to Queue 2 since an assignment to Queue 1 might be a waste if work is about to be completed at that queue anyway. So in this case the Fewest Volunteers policy seems illogical. A similar examination of the Largest Demand policy shows that it would not always seem logical.

Although neither the Fewest Volunteers policy nor the Largest Demand policy is optimal, it is true that the Optimal Policy in each case agrees with either the Largest Demand policy or the Fewest Volunteers policy for the large majority of states. We now examine the similarities between the Optimal Policy and the heuristic policies for each of the cases presented in Table 3.1. In each case we tally the number of times the Optimal Policy disagrees with FV only, with LD only, and with both. These results are presented in Table 3.3. It is evident from this table that the Optimal Policy agrees more often with the Fewest Volunteers policy than with the Largest Demand policy. It is also evident that the Optimal Policy very rarely disagrees with both of these heuristic policies at the same time. In fact, this never occurs when the two queues have identical service rates (see Cases 1 and 4).

Table 3.3 Optimal policy vs. heuristic policy disagreements.

Case	States	FV	Percent	LD	Percent	Both	Percent
1	3721	380	10.2%	520	14.0%	0	0.0 %
2	3721	407	10.9%	513	13.8%	10	0.3 %
3	3721	446	12.0%	522	14.0%	34	0.9 %
4	1769	181	10.2%	260	14.7%	0	0.0 %
5	1769	204	11.5%	251	14.2%	7	0.4 %
6	1769	184	10.4%	261	14.8%	2	0.1 %
7	1769	192	10.85%	263	14.9%	7	0.4 %
8	1769	225	12.7%	248	14.0%	16	0.9%

The policy differences presented in Table 3.3 are only theoretical differences. It is true that some of the states where the Optimal Policy differs from either FV or LD are unlikely to ever occur in practice. For instance, going back to the first case where the two queues are identical, consider the state (4, 3, 10, 9). For this state the Optimal Policy would assign an incoming volunteer to Queue 2, whereas LD would assign an incoming volunteer to Queue 1. However, it is unlikely that, starting at state (0, 0, 10, 10), the system would have four volunteers at Queue 1 without having completed

any units of work at that queue. So, in practice, even though at this state the Optimal Policy differs from LD, it is unlikely that this situation would ever arise. To get a better sense of the real differences between the Optimal and heuristic policies, in Chapter 5 we will examine the states where there is disagreement and see how often they occur over the course of many simulation runs, thereby quantifying how different the policies are in practice.

Chapter

4

Simulation Results and Analysis

In order to evaluate the performance of the Optimal Policy compared with heuristic policies, an Arena model was developed to simulate the system at hand. The Optimal Policies were generated with the objective of maximizing benefit by completing units of work. As a proxy for benefit gained by completing work, the performance metric in the Arena model is the time to complete all units of work in the system when starting with maximum demand and zero volunteers at each queue. We refer to the completion of all units of work in a given run as project completion. Completion of work at an individual queue is termed activity completion. When comparing the Optimal Policy with heuristic policies, we will be comparing mean project completion times and mean individual activity completion times. We will also examine how often the Optimal Policy agrees with the heuristic policies over the course of simulation. In the final section of this chapter we will present distribution-fitting results for the project and individual activity completion times.

We will test the robustness of the Optimal Policy by allowing for non-Exponential interarrivals in the simulation and performing the same analyses under this scenario that were performed for Exponential interarrivals. Exponentially distributed interarrival times was a necessary assumption for generating Optimal Policies, but this assumption may be unrealistic. Therefore we will draw from the empirical study of Lodree et al. (2015) by using one of the fits the authors found for interarrival times of spontaneous volunteers in the wake of the April 2011 tornado in Tuscaloosa, Alabama. Further

motivation for this analysis is the work of Akgun et al. (2011), which generalized the optimality of the JSQ rule to the case with arbitrary arrivals.

4.1 Arena Model Description

In this section we present a description of the Arena model used to test the performance of the Optimal Policy and heuristic policies. Each run begins in state $(0, 0, D_1, D_2)$, that is: zero volunteers and maximum demand at each queue. Each run ends when demand at each queue is reduced to zero. Five different dedicated Random Number Streams are used in various capacities within the model to allow for paired-sample comparisons between the Optimal Policy and the heuristic policies. A logical flow diagram for this Arena model is found in Fig. 4.1.

Figure 4.1 Graphical representation of the system dynamics.

Volunteers that go through the system are generated randomly one at a time, according to the specified interarrival distribution using Random Number Stream 1 (RNS1). At the Assign station, the model first assigns the incoming volunteer its Delay To Departure attribute which specifies the time the volunteer will spend working before abandoning the system. This Delay To Departure is determined according to an Exponential distribution with rate γ , using RNS2. Next, the model checks the current state of the system, (v_1, v_2, d_1, d_2) , and then looks for that state in the Assignment Policy currently being used, whether it is the Optimal Policy or one of the heuristics. Once the state is found in the chosen policy, the model then follows the assignment decision specified by the Assignment Policy. The Assignment Policy will specify that the volunteer be sent to one of the two queues, or it will direct the system to choose randomly between the two queues with equal probability using RNS3. Once an assignment decision is reached, if the chosen queue is full, the volunteer is sent out of the system and is tallied as a volunteer who was assigned to unrelated work. If the chosen queue is not full, then the volunteer is sent to that queue and enters that queue's Activity Process. Each of the two Activity Processes is a submodel.

Once a volunteer enters an Activity Process, they remain in that process until either their departure time arrives or until all of the work at that activity is completed. Within each Activity Process, discrete work completion is scheduled according to an Exponential distribution with a rate proportional to the number of volunteers currently working at that Activity Process. For example, work at Activity Process 1 is scheduled according to an Exponential distribution with rate $v_1\mu_1$. Each Activity Process uses its own RNS to schedule work completion, bringing the total of number of Random Number Streams to five. Great care must be taken within each Activity Process to ensure that work completion and individual volunteer abandonment occur in the correct order. For instance, if a volunteer's departure time occurs after the next work completion has been scheduled but before the work completion is scheduled to occur, then that work completion needs to be superceded and then rescheduled using the new number of volunteers at that Activity Process.

The model was developed in Arena version 14.70 and all simulations were run on a 2.83 GHz Intel(R) Core(TM)2 Quad CPU machine with 8.00 GB of RAM. Each simulation took around 90 seconds to complete. A more detailed explanation of this model is found in Appendix B.

4.2 Paired-sample Comparisons

In this section we present paired-sample comparisons evaluating the performance of the Optimal Policy compared to the heuristic policies. This analysis is performed for both Exponentially distributed interarrival times and Weibull-distributed interarrival times. Paired-sample comparisons of mean project completion time, mean activity one completion time, and mean activity two completion time

were performed using Arena's Output Analyzer.

4.2.1 Exponential interarrival times

The parameters for each of the cases examined here are found in Table 4.1. These cases were chosen to examine the effect of different work rates, maximum capacities for volunteers, and initial starting demand. The work rates take on three different values, while the maximum capacities for volunteers and starting demand each take two different values. After ignoring cases which are equivalent due to symmetry, we are left with eight cases. Paired-sample comparison results for mean project completion times are found in Table 4.2. For each case, Table 4.2 is composed of the mean when following the Optimal policy, the mean when following each of the heuristic policies, and then, for each of the heuristic policies, the difference from the Optimal mean, the standard deviation, the relative difference, and the relative 95% Confidence Interval.

Table 4.1 Case parameters.

Case	V_1	V_2	D_1	D_2	λ	γ	μ_1	μ_2	H	R
1	5	5	10	10	1	0.5	2.4	2.4	1	10
2	5	5	10	10	1	0.5	2.4	2.0	1	10
3	5	5	10	10	1	0.5	2.4	1.6	1	10
4	3	5	7	10	1	0.5	2.4	2.4	1	10
5	3	5	7	10	1	0.5	2.4	2.0	1	10
6	3	5	7	10	1	0.5	2.0	2.4	1	10
7	3	5	7	10	1	0.5	2.4	1.6	1	10
8	3	5	7	10	1	0.5	1.6	2.4	1	10

As expected, mean project completion times are lower for higher values of μ_1 and μ_2 . Additionally, mean project completion times are higher for higher values of demand. Statistically significant differences between the Optimal policy and the heuristic policies are boldfaced. The Optimal Policy outperforms both the LD policy and the Random policy in every case examined. In cases 2 and 8 the Optimal Policy outperforms the FV policy, while in Case 7 the FV policy outperforms the Optimal Policy. In every other case, the Optimal Policy and the FV policy are statistically tied in performance. Case 7, where the FV policy outperforms the Optimal Policy, shows that project completion time in the Arena model is not a perfect translation of the objective of maximizing benefit in the DTMC problem; however, the performance of the Optimal Policy in all other cases suggests project completion is a

Table 4.2 Paired-sample project completion time results for Exponential interarrivals over 1,000 runs. Boldface indicates statistical significance.

Case	Optimal	Heuristic	Mean	Difference	SD	Rel. Difference	Rel. 95% CI	
1	7.7211	FV	7.7274	0.0063	0.571	0.1%	-0.4%	0.5%
		LD	7.9181	0.197	1.07	2.6%	1.7%	3.4%
		R	8.268	0.5469	1.48	7.1%	5.9%	8.3%
2	8.0981	FV	8.1873	0.0892	1.28	1.1%	0.1%	2.1%
		LD	8.373	0.2749	1.36	3.4%	2.4%	4.4%
		R	8.7579	0.6598	1.74	8.1%	6.8%	9.5%
3	8.8677	FV	8.9303	0.0626	1.56	0.7%	-0.4%	1.8%
		LD	9.0779	0.2102	1.75	2.4%	1.1%	3.6%
		R	9.4588	0.5911	1.87	6.7%	5.4%	8.0%
4	6.9945	FV	7.0056	0.0111	0.58	0.2%	-0.4%	0.7%
		LD	7.2898	0.2953	1.24	4.2%	3.1%	5.3%
		R	7.541	0.5465	1.5	7.8%	6.5%	9.1%
5	7.4752	FV	7.4865	0.0113	1.32	0.2%	-0.9%	1.2%
		LD	7.6871	0.2119	1.43	2.8%	1.7%	4.0%
		R	8.0469	0.5717	1.64	7.6%	6.3%	9.0%
6	7.3843	FV	7.355	-0.0293	0.865	-0.4%	-1.1%	0.3%
		LD	7.6506	0.2663	1.34	3.6%	2.5%	4.7%
		R	7.9154	0.5311	1.61	7.2%	5.8%	8.5%
7	7.9478	FV	7.8645	-0.0833	1.23	-1.0%	-2.0%	-0.1%
		LD	8.131	0.1832	1.35	2.3%	1.3%	3.4%
		R	8.4696	0.5218	1.61	6.6%	5.3%	7.8%
8	8.1072	FV	8.2496	0.1424	1.7	1.8%	0.5%	3.1%
		LD	8.3907	0.2835	1.64	3.5%	2.2%	4.8%
		R	8.7586	0.6514	1.83	8.0%	6.6%	9.4%

good metric. The results in Table 4.2 also suggest that while the LD and Random policies perform very poorly relative to the Optimal Policy, the FV policy in general performs very well and that its performance is not significantly affected by changing work rates, maximum capacities, or starting levels of demand. This is especially noteworthy because the FV policy accounts for neither work rates nor starting levels of demand, and only indirectly considers maximum capacities.

It is also important to note that there is no scale on these project completion times. Within the Arena model these refer to hours, but in the real world these relative differences have a broader scope. With this in mind, consider the performance of the LD and Random policies, two policies which one

might implement when faced with the harrowing task of assigning unknown numbers of randomly arriving volunteers. These are both rational policy choices but in this model they have been shown to be suboptimal by between 1.1% and 9.5%. This suboptimality could translate in the real world to very poor performance on a large scale, on work that is vitally important to entire communities.

Paired-sample comparison results for mean activity one and activity two completion times are found in Tables 4.3 and 4.4, respectively. Note that in terms of activity one and activity two completion times, the differences between the Optimal Policy and the heuristic policies are much more pronounced. For instance, in Table 4.3, note that in Case 8 the 95% CI for the LD policy is (22.0%, 25.1%).

Table 4.3 Paired-sample activity one completion time results for Exponential interarrivals over 1,000 runs. Boldface indicates statistical significance.

Case	Optimal	Heuristic	Mean	Difference	SD	Rel. Difference	Rel. 95% CI	
1	7.0058	FV	6.953	-0.0528	0.469	-0.8%	-1.2%	-0.3%
		LD	7.0593	0.0535	0.793	0.8%	0.1%	1.5%
		R	6.8294	-0.1764	0.0867	-2.5%	-3.7%	-1.4%
2	6.9863	FV	7.091	0.1047	1.13	1.5%	0.5%	2.5%
		LD	7.3471	0.3608	1.24	5.2%	4.1%	6.3%
		R	6.967	-0.0193	1.68	-0.3%	-1.8%	1.2%
3	6.9643	FV	7.2164	0.2521	1.38	3.6%	2.4%	4.9%
		LD	7.7286	0.7643	1.41	11.0%	9.7%	12.2%
		R	7.1219	0.1576	1.82	2.3%	0.6%	3.9%
4	6.1924	FV	6.1933	0.0009	0.61	0.0%	-0.6%	0.6%
		LD	6.634	0.4416	1	7.1%	6.1%	8.1%
		R	5.683	-0.5094	1.38	-8.2%	-9.6%	-6.8%
5	6.0158	FV	6.3351	0.3193	1.26	5.3%	4.0%	6.6%
		LD	6.913	0.8972	1.35	14.9%	13.5%	16.3%
		R	5.7937	-0.2221	1.57	-3.7%	-5.3%	-2.1%
6	6.8741	FV	6.7325	-0.1416	0.942	-2.1%	-2.9%	-1.2%
		LD	7.0639	0.1898	0.9	2.8%	1.9%	3.6%
		R	6.2242	-0.6499	1.53	-9.5%	-10.8%	-8.1%
7	7.6566	FV	7.3993	-0.2573	1.06	-3.4%	-4.2%	-2.5%
		LD	7.6082	-0.0484	0.945	-0.6%	-1.4%	0.1%
		R	7.066	-0.5906	1.58	-7.7%	-9.0%	-6.4%
8	6.0012	FV	6.5009	0.4997	1.43	8.3%	6.8%	9.8%
		LD	7.4127	1.4115	1.51	23.5%	22.0%	25.1%
		R	5.894	-0.1072	1.75	-1.8%	-3.6%	0.0%

Table 4.4 Paired-sample activity two completion time results for Exponential interarrivals over 1,000 runs.
 Boldface indicates statistical significance.

Case	Optimal	Heuristic	Mean	Difference	SD	Rel. Difference	Rel. 95% CI	
1	6.9606	FV	6.9265	-0.0341	0.405	-0.5%	-0.9%	-0.1%
		LD	7.0103	0.0497	0.754	0.7%	0.0%	1.4%
		R	6.8897	-0.0709	1.31	-1.0%	-2.2%	0.1%
2	7.6137	FV	7.5902	-0.0235	0.854	-0.3%	-1.0%	0.4%
		LD	7.5907	-0.023	0.968	-0.3%	-1.1%	0.5%
		R	7.5762	-0.0375	1.4	-0.5%	-1.6%	0.6%
3	8.569	FV	8.5672	-0.0018	1.07	0.0%	-0.8%	0.8%
		LD	8.3844	-0.1846	1.15	-2.2%	-3.0%	-1.3%
		R	8.4786	-0.0904	1.45	-1.1%	-2.1%	0.0%
4	6.3304	FV	6.258	-0.0724	0.563	-1.1%	-1.7%	-0.6%
		LD	5.9769	-0.3535	1.01	-5.6%	-6.6%	-4.6%
		R	6.6283	0.2979	1.25	4.7%	3.5%	5.9%
5	7.0302	FV	6.9422	-0.088	0.997	-1.3%	-2.1%	-0.4%
		LD	6.5603	-0.4699	1.19	-6.7%	-7.7%	-5.6%
		R	7.2832	0.253	1.25	3.6%	2.5%	4.7%
6	6.2189	FV	6.3427	0.1238	0.907	2.0%	1.1%	2.9%
		LD	6.1025	-0.1164	0.925	-1.9%	-2.8%	-0.9%
		R	6.7703	0.5514	1.37	8.9%	7.5%	10.2%
7	6.1254	FV	6.4633	0.3379	1.04	5.5%	4.5%	6.6%
		LD	6.2913	0.1659	0.958	2.7%	1.7%	3.7%
		R	6.9184	0.793	1.54	12.9%	11.4%	14.5%
8	7.8233	FV	7.9291	0.1058	1.05	1.4%	0.5%	2.2%
		LD	7.4787	-0.3446	1.19	-4.4%	-5.3%	-3.5%
		R	8.1508	0.3275	1.32	4.2%	3.1%	5.2%

4.2.2 Non-exponential interarrival times

Here we perform the same analysis as in Section 4.2.1, but allow for non-Exponential interarrival times in the simulation. For this we turn to Lodree et al. (2015), an empirical study of spontaneous volunteer convergence in the wake of the April 2011 tornado in Tuscaloosa, Alabama. Interarrival rates were found to not be homogeneous over time and so the authors used a clustering algorithm to partition interarrivals into disjoint groups such that, within each group, the interarrival distribution was homogeneous. For one cluster, the best fit for the interarrival times was a Weibull distribution

with a p -value of 0.253. The authors provided us with the parameters for that fit. The scale parameter, α , is 36.2959 while the shape parameter, β , is 0.983179. The authors also found that an Exponential fit with $\lambda = 0.0273401$ had a similar p -value of 0.2118. These fits are in terms of minutes, and so for our purposes $\lambda = 1.640406$ and the Weibull interarrivals in the simulation are in terms of minutes. For each of the eight cases, the same analysis performed in Section 4.2.1 was performed using Weibull-distributed interarrival times with $\alpha = 36.2959$ and $\beta = 0.983179$. Since exponentially distributed interarrival times are a necessary assumption for this system to be framed and solved as a Markov decision process, Optimal Policies were generated using the Exponential fit for this same cluster, with $\lambda = 1.640406$. It turns out that in this case, the Weibull distribution provided is nearly identical to the Exponential distribution provided. This is evidenced by the plots of their cumulative distribution functions (CDFs), as shown in Fig. 4.2. Based on the similarity of the Weibull fit to the Exponential fit, we do not expect our results for the Weibull interarrival scenario to differ much from the results for the Exponential interarrival scenario. The mean interarrival time is lower in this scenario than in the Exponential scenario, and so we do expect shorter completion times in general. The eight cases examined here are the same as those examined in the previous section, but with a different value of λ . These cases are presented in Table 4.5.

Figure 4.2 CDF plots for the Exponential and Weibull fits.

Paired-sample comparison results for mean project completion times are found in Table 4.6. As with Exponential interarrivals, mean project completion times are lower for higher values of μ_1 and μ_2 , and mean project completion times are higher for higher values of demand. The cases examined here, 9-16, are equal to cases 1-8 except for λ , and the paired-sample comparison results are nearly

Table 4.5 Case parameters for non-Exponential interarrival analysis.

Case	V_1	V_2	D_1	D_2	λ	γ	μ_1	μ_2	H	R
9	5	5	10	10	1.640406	0.5	2.4	2.4	1	10
10	5	5	10	10	1.640406	0.5	2.4	2.0	1	10
11	5	5	10	10	1.640406	0.5	2.4	1.6	1	10
12	3	5	7	10	1.640406	0.5	2.4	2.4	1	10
13	3	5	7	10	1.640406	0.5	2.4	2.0	1	10
14	3	5	7	10	1.640406	0.5	2.0	2.4	1	10
15	3	5	7	10	1.640406	0.5	2.4	1.6	1	10
16	3	5	7	10	1.640406	0.5	1.6	2.4	1	10

the same. Just like with Exponential interarrivals, the Optimal Policy outperforms both the LD policy and the Random policy in every case. The performance of the FV policy relative to the Optimal policy is almost the same as with Exponential interarrivals. The Optimal Policy outperforms the FV policy in Case 16, while the FV policy outperforms the Optimal Policy in Case 15. These two cases are equal except in λ to cases 7 and 8 which had the same result. In Case 3 with Exponential interarrivals, the Optimal Policy outperformed the FV policy, but here in Case 11, which is equal to Case 3 except in λ , they are tied in performance. These results show that although the Optimal Policy is designed for Exponential interarrivals, it still performs well relative to the heuristic policies when simulated using Weibull interarrivals. Additionally we see that, like with Exponential interarrivals, the FV policy performs very well compared with the Optimal Policy, and in one case outperforms the Optimal Policy. In short, allowing for Weibull interarrivals changes very little.

Paired-sample comparison results for mean activity one and activity two completion times are found in Tables 4.7 and 4.8, respectively. Note that in terms of activity one and activity two completion times, the differences between the Optimal Policy and the heuristic policies are much more pronounced, as they were with Exponential interarrivals. For instance, in Table 4.8, note that in Case 16 the 95% CI for the LD policy is (17.3%, 21.7%).

4.3 Policy Differences

In order to put these comparisons of performance in perspective, we now move to an examination of the differences between the Optimal Policy and the heuristic policies over the course of simulation. For both Exponentially distributed interarrivals and Weibull-distributed interarrivals, we tally the number of times during a simulation of the Optimal Policy that a state occurs where the Optimal

Table 4.6 Paired-sample project completion time results for Weibull interarrivals over 1,000 runs. Boldface indicates statistical significance.

Case	Optimal	Heuristic	Mean	Difference	SD	Rel. Difference	Rel. 95% CI	
9	5.2131	FV	5.2213	0.0082	0.377	0.2%	-0.3%	0.6%
		LD	5.33	0.1169	0.709	2.2%	1.4%	3.1%
		R	5.6316	0.4185	1.02	8.0%	6.8%	9.2%
10	5.5095	FV	5.531	0.0215	0.814	0.4%	-0.5%	1.3%
		LD	5.6739	0.1644	0.917	3.0%	2.0%	4.0%
		R	5.9505	0.441	1.12	8.0%	6.7%	9.3%
11	6.0038	FV	5.9941	-0.0097	0.94	-0.2%	-1.1%	0.8%
		LD	6.1079	0.1041	1.01	1.7%	0.7%	2.8%
		R	6.419	0.4152	1.17	6.9%	5.7%	8.1%
12	4.7275	FV	4.7476	0.0201	0.584	0.4%	-0.3%	1.2%
		LD	4.9545	0.227	0.853	4.8%	3.7%	5.9%
		R	5.1716	0.4441	0.982	9.4%	8.1%	10.7%
13	5.0362	FV	5.0792	0.043	0.787	0.9%	-0.1%	1.8%
		LD	5.2365	0.2003	0.939	4.0%	2.8%	5.1%
		R	5.504	0.4678	1.05	9.3%	8.0%	10.6%
14	5.076	FV	4.9914	-0.0846	0.72	-1.7%	-2.5%	-0.8%
		LD	5.233	0.157	0.751	3.1%	2.2%	4.0%
		R	5.4201	0.3441	0.999	6.8%	5.6%	8.0%
15	5.4836	FV	5.3468	-0.1368	0.888	-2.5%	-3.5%	-1.5%
		LD	5.6231	0.1395	0.825	2.5%	1.6%	3.5%
		R	5.808	0.3244	1.13	5.9%	4.6%	7.2%
16	5.48	FV	5.6195	0.1395	0.888	2.5%	1.5%	3.6%
		LD	5.7035	0.2235	0.99	4.1%	3.0%	5.2%
		R	6.0082	0.5282	1.14	9.6%	8.3%	10.9%

Policy disagrees with either the FV or LD policies. This will give us a sense of the differences between the Optimal Policy and those heuristics in practice.

4.3.1 Exponential interarrivals

The theoretical policy differences examined in Chapter 3 are presented again here in Table 4.9. The policy differences over the course of simulation with Exponentially distributed interarrival times are presented in Table 4.10. Note that in Cases 1 and 4, where $\mu_1 = \mu_2$, there are very few differences between the Optimal Policy and the FV policy, amounting to 4.3% and 3.5% of states encountered over

Table 4.7 Paired-sample activity one completion time results for Weibull interarrivals over 1,000 runs. Boldface indicates statistical significance.

Case	Optimal	Heuristic	Mean	Difference	SD	Rel. Difference	Rel. 95% CI
9	4.7825	FV	4.7649	-0.0176	0.469	-0.4%	-1.0% 0.2%
		LD	4.8361	0.0536	0.793	1.1%	0.1% 2.1%
		R	4.7221	-0.0604	1.4	-1.3%	-3.1% 0.5%
10	4.8202	FV	4.8495	0.0293	1.13	0.6%	-0.9% 2.1%
		LD	5.0597	0.2395	1.24	5.0%	3.4% 6.6%
		R	4.8067	-0.0135	1.68	-0.3%	-2.4% 1.9%
11	4.6912	FV	4.9297	0.2385	1.138	5.1%	3.3% 6.9%
		LD	5.2939	0.6027	1.41	12.8%	11.0% 14.7%
		R	4.8921	0.2009	1.82	4.3%	1.9% 6.7%
12	4.2583	FV	4.1984	-0.0599	1.138	-1.4%	-3.4% 0.6%
		LD	4.6264	0.3681	1.41	8.6%	6.6% 10.7%
		R	3.951	-0.3073	1.82	-7.2%	-9.9% -4.6%
13	4.2271	FV	4.2617	0.0346	1.26	0.8%	-1.0% 2.7%
		LD	4.8098	0.5827	1.35	13.8%	11.8% 15.8%
		R	4.0157	-0.2114	1.57	-5.0%	-7.3% -2.7%
14	4.8021	FV	4.5848	-0.2173	0.942	-4.5%	-5.7% -3.3%
		LD	4.9516	0.1495	0.9	3.1%	1.9% 4.3%
		R	4.3595	-0.4426	1.53	-9.2%	-11.2% -7.2%
15	5.3008	FV	5.0634	-0.2374	1.06	-4.5%	-5.7% -3.2%
		LD	5.3977	0.0969	0.945	1.8%	0.7% 2.9%
		R	4.9249	-0.3759	1.58	-7.1%	-8.9% -5.2%
16	4.2784	FV	4.345	0.0666	1.43	1.6%	-0.5% 3.6%
		LD	5.1107	0.8323	1.51	19.5%	17.3% 21.7%
		R	4.0743	-0.2041	1.75	-4.8%	-7.3% -2.2%

the course of simulation, respectively. In cases 1, 4, and 6 the differences over the course of simulation for the FV policy are less than the theoretical differences between the Optimal Policy and the FV policy presented in Table 4.9. However, in all other cases, the differences between the Optimal Policy and the FV policy over the course of simulation are greater than the theoretical differences for those same cases. The lowest number of divergences from the LD policy amounts to 9.2%, in Case 1, where the two queues are identical. In cases 1, 6, and 7 the Optimal Policy and the LD policy are more similar in practice than in theory. In all other cases, the differences over the course of simulation are greater than the theoretical differences.

Table 4.8 Paired-sample activity two completion time results for Weibull interarrivals over 1,000 runs. Boldface indicates statistical significance.

Case	Optimal	Heuristic	Mean	Difference	SD	Rel. Difference	Rel. 95% CI
9	4.7437	FV	4.7132	-0.0305	0.405	-0.6%	-1.2% -0.1%
		LD	4.7877	0.044	0.754	0.9%	-0.1% 1.9%
		R	4.7557	0.012	1.31	0.3%	-1.5% 2.0%
10	5.196	FV	5.1829	-0.0131	0.854	-0.3%	-1.3% 0.8%
		LD	5.1722	-0.0238	0.968	-0.5%	-1.6% 0.7%
		R	5.1808	-0.0152	1.4	-0.3%	-2.0% 1.4%
11	5.8605	FV	5.7881	-0.0724	1.07	-1.2%	-2.4% -0.1%
		LD	5.701	-0.1595	1.15	-2.7%	-3.9% -1.5%
		R	5.8044	-0.0561	1.45	-1.0%	-2.5% 0.6%
12	4.3438	FV	4.3345	-0.0093	1.07	-0.2%	-1.7% 1.3%
		LD	4.0527	-0.2911	1.15	-6.7%	-8.3% -5.1%
		R	4.6068	0.263	1.45	6.1%	4.0% 8.1%
13	4.7678	FV	4.8133	0.0455	0.997	1.0%	-0.3% 2.3%
		LD	4.4877	-0.2801	1.19	-5.9%	-7.4% -4.3%
		R	5.0194	0.2516	1.25	5.3%	3.7% 6.9%
14	4.2557	FV	4.3919	0.1362	0.907	3.2%	1.9% 4.5%
		LD	4.125	-0.1307	0.925	-3.1%	-4.4% -1.7%
		R	4.666	0.4103	1.37	9.6%	7.7% 11.6%
15	4.2718	FV	4.4638	0.192	1.04	4.5%	3.0% 6.0%
		LD	4.2265	-0.0453	0.958	-1.1%	-2.5% 0.3%
		R	4.7843	0.5125	1.54	12.0%	9.8% 14.2%
16	5.3078	FV	5.4624	0.1546	1.05	2.9%	1.7% 4.1%
		LD	5.0819	-0.2259	1.19	-4.3%	-5.6% -2.9%
		R	5.6317	0.3239	1.32	6.1%	4.6% 7.7%

4.3.2 Non-exponential interarrival times

The theoretical policy differences for the cases designed for Weibull interarrival times are presented in Table 4.11. The policy differences over the course of simulation for these same cases are presented in Table 4.12. In Cases 9 and 12, where $\mu_1 = \mu_2$, the differences between the Optimal Policy and the FV policy over the course of simulation amount to just 6.1% and 6.0% respectively. This is less than the theoretical differences for those cases. In all other cases, the Optimal Policy and the FV policy are more similar in theory than in practice. In cases 9, 14, and 15 the Optimal Policy and the LD policy are more similar in practice than in theory. In all other cases, the Optimal Policy and the LD policy are

Table 4.9 Optimal policy vs. heuristic policy disagreements.

Case	States	FV	Percent	LD	Percent	Both	Percent
1	3721	380	10.2%	520	14.0%	0	0.0 %
2	3721	407	10.9%	513	13.8%	10	0.3 %
3	3721	446	12.0%	522	14.0%	34	0.9 %
4	1769	181	10.2%	260	14.7%	0	0.0 %
5	1769	204	11.5%	251	14.2%	7	0.4 %
6	1769	184	10.4%	261	14.8%	2	0.1 %
7	1769	192	10.85%	263	14.9%	7	0.4 %
8	1769	225	12.7%	248	14.0%	16	0.9%

Table 4.10 Policy differences over 1,000 replications with exponential interarrivals.

Case	States	FV	Percent	LD	Percent
1	7623	180	4.3%	701	9.2%
2	7985	330	15.3%	1116	14.0%
3	8692	1225	17.5%	1538	17.7%
4	6850	241	3.5%	1168	17.1%
5	7332	1801	24.6%	1908	26.0%
6	7232	612	8.5%	1006	13.9%
7	7776	1057	13.6%	907	11.7%
8	8003	2209	27.6%	2111	26.4%

more similar in theory than in practice.

Table 4.11 Policy differences.

Case	States	FV	Percent	LD	Percent	Both	Percent
9	3721	416	11.2%	484	13.0%	0	0.0%
10	3721	445	12.0%	487	13.1%	16	0.4%
11	3721	486	13.1%	498	13.4%	42	1.1%
12	1769	198	11.2%	243	13.7%	0	0.0%
13	1769	219	12.4%	244	13.8%	11	0.6%
14	1769	206	11.6%	243	13.7%	4	0.2%
15	1769	215	12.2%	240	13.6%	7	0.4%
16	1769	241	13.6%	246	13.9%	23	1.3%

Table 4.12 Policy differences over 1,000 replications with Weibull interarrivals.

Case	States	FV	Percent	LD	Percent
9	8439	513	6.1%	856	10.1%
10	8915	1157	13.0%	1254	14.1%
11	9707	2215	22.8%	1920	19.8%
12	7631	458	6.0%	1510	19.8%
13	8154	2231	27.4%	2092	25.7%
14	8189	1140	13.9%	1086	13.3%
15	8870	1413	15.9%	939	10.6%
16	8866	2606	29.4%	2242	25.3%

4.4 Distribution Fitting

We now examine the project completion, activity one completion, and activity two completion times over 1,000 replications for each case considered and perform distribution fitting analysis. All distribution fitting work was done with Stat::Fit 2. In light of the poor performance of both the LD and Random policies, as shown in Section 4.2, distribution fitting analysis is only performed here for the Optimal Policy and the FV policy.

4.4.1 Exponential interarrivals

The top three fits according to Stat::Fit rank of project completion times with Exponential interarrivals are presented in Table 4.13, along with the corresponding p -values for both the Kolmogorov-Smirnov and Anderson-Darling tests. Particularly good fits were found for Beta, Chi Squared, Erlang, Gamma,

Inverse Gaussian, Johnson SB, and Lognormal. The Gamma and Johnson SB distributions are of particular note as they appear in the top three by Stat::Fit rank a total of 14 out of 16 times each. Plots of the probability density function (PDF) and CDF fits under both the Optimal Policy and the FV policy for Case 1 are found in Fig. 4.3. In this case, great fits were found under both policies. For comparison, PDF and CDF fits under both the Optimal Policy and the FV policy for Case 2 are found in Fig. 4.4. Here, the fits under the FV policy are noticeably better than the fits under the Optimal Policy. The difference is particularly evident upon examining the CDF plots.

When following the Optimal Policy, the following distributions were not rejected for any case: Beta, Chi Squared, Erlang, Gamma, Inverse Gaussian, Johnson SB, Lognormal, Pearson 5, Pearson 6, and Weibull. The following distributions were rejected in every case under the Optimal Policy: Exponential, Inverse Weibull, Pareto, Rayleigh, Triangular, and Uniform. When following the FV policy, the following distributions were not rejected for any case: Beta, Chi Squared, Erlang, Gamma, Inverse Gaussian, LogLogistic, Lognormal, Pearson 5, Pearson 6, and Weibull. The following distributions were rejected in every case under the FV policy: Exponential, Inverse Weibull, Pareto, Triangular, and Uniform.

The top three fits according to Stat::Fit rank of activity one completion times with Exponential interarrivals are presented in Table 4.13, along with the corresponding p -values for both the Kolmogorov-Smirnov and Anderson-Darling tests. Here we see the addition of Pearson 6 in the top three in two cases, but besides that the top contenders are the same. When following the Optimal Policy, the following distributions were rejected in every case: Exponential, Pareto, Power Function, Triangular, and Uniform. When following the FV policy, the following distributions were rejected in every case: Exponential, Pareto, Power Function, Rayleigh, Triangular, and Uniform.

The top three fits according to Stat::Fit rank of activity two completion times with Exponential interarrivals are presented in Table 4.15, along with the corresponding p -values for both the Kolmogorov-Smirnov and Anderson-Darling tests. The results here are similar to those for activity one. When following the Optimal Policy, the following distributions were rejected in every case: Exponential, Inverse Weibull, Pareto, Power Function, Rayleigh, Triangular, and Uniform. When following the FV policy, the following distributions were rejected in every case: Exponential, Pareto, Power Function, Triangular, and Uniform.

4.4.2 Non-exponential interarrivals

The top three fits according to Stat::Fit rank of project completion, activity one completion, and activity two completion times with Weibull interarrivals are presented in Table 4.16, Table 4.17 and Table 4.18 respectively, along with the corresponding p -values for both the Kolmogorov-Smirnov and

Figure 4.3 Project completion distribution fitting results for Case 1.
 (a) PDF, Optimal; (b) CDF, Optimal; (c) PDF, FV; (d) CDF, FV.

Anderson-Darling tests. The results do not differ greatly from the results for Exponential interarrivals; however, Inverse Gaussian, Pearson 5, and Pearson 6 are all more prevalent under Weibull interarrivals. Plots of the PDF and CDF fits under both the Optimal Policy and the FV policy for Case 10 are found in Fig. 4.5. This case is particularly notable as the p -values for the top three fits under the FV policy are all equal to 1, for both tests. Now consider Case 13. This case is notable because the fits under the FV policy are noticeably better than the fits under the Optimal Policy; furthermore, under the FV policy, both p -values for the Johnson SB distribution are equal to 1, and the A-D p -value for the Gamma distribution is equal to 1. The PDF and CDF fits under both the Optimal Policy and the FV policy for Case 13 are found in Fig. 4.6.

Figure 4.4 Project completion distribution fitting results for Case 2.
 (a) PDF, Optimal; (b) CDF, Optimal; (c) PDF, FV; (d) CDF, FV.

4.4.3 Summary

To conclude this section we note that the following distributions were never rejected through the course of this distribution fitting analysis, whether with Exponential or Weibull interarrivals: Erlang, Gamma, Inverse Gaussian, Lognormal, and Pearson 5. These distributions were consistently found to be acceptable (and in some cases perfect) fits for both project completion times and individual activity completion times. Meanwhile, the following distributions were rejected in every case under both Exponential and Weibull interarrivals: Exponential and Uniform.

Figure 4.5 Project completion distribution fitting results for Case 10.
 (a) PDF, Optimal; (b) CDF, Optimal; (c) PDF, FV; (d) CDF, FV.

Table 4.13 Distribution fitting results for project completion times with Exponential interarrivals.
Sample size: 1,000.

Case	Optimal	p(K-S)	p(A-D)	FV	p(K-S)	p(A-D)
1	Gamma	0.997	0.996	Gamma	0.987	0.996
	Johnson SB	0.996	0.997	Johnson SB	0.977	0.997
	Erlang	0.992	0.995	Inv. Gaussian	0.93	0.967
2	Johnson SB	0.846	0.877	Johnson SB	0.985	0.999
	Gamma	0.686	0.876	Gamma	0.979	0.999
	Beta	0.772	0.703	Beta	0.952	0.998
3	Johnson SB	0.897	0.873	Johnson SB	0.937	0.99
	Beta	0.853	0.855	Gamma	0.912	0.986
	Gamma	0.807	0.882	Erlang	0.9	0.988
4	Inv. Gaussian	0.993	0.983	Johnson SB	0.995	0.998
	Johnson SB	0.957	0.985	Inv. Gaussian	0.988	0.994
	Lognormal	0.975	0.953	Gamma	0.987	0.992
5	Gamma	0.882	0.934	Gamma	0.938	0.983
	Johnson SB	0.83	0.937	Johnson SB	0.917	0.984
	Beta	0.917	0.833	Inv. Gaussian	0.76	0.946
6	Johnson SB	0.99	0.984	Johnson SB	0.996	0.998
	Gamma	0.967	0.988	Gamma	0.993	0.997
	Inv. Gaussian	0.894	0.932	Inv. Gaussian	0.912	0.989
7	Inv. Gaussian	0.968	0.987	Johnson SB	0.88	0.965
	Gamma	0.924	0.994	Beta	0.822	0.903
	Lognormal	0.913	0.962	Gamma	0.738	0.921
8	Inv. Gaussian	0.925	0.977	Beta	0.814	0.782
	Johnson SB	0.88	0.974	Chi Squared	0.77	0.819
	Lognormal	0.822	0.962	Gamma	0.645	0.874

Table 4.14 Distribution fitting results for activity one completion times with Exponential interarrivals.
Sample size: 1,000.

Case	Optimal	p(K-S)	p(A-D)	FV	p(K-S)	p(A-D)
1	Johnson SB	0.997	0.998	Gamma	0.996	1
	Gamma	0.981	0.994	Beta	0.995	1
	Beta	0.933	0.994	Johnson SB	0.979	1
2	Beta	0.973	0.993	Gamma	0.986	1
	Johnson SB	0.961	0.998	Johnson SB	0.955	0.999
	Chi Squared	0.969	0.937	Beta	0.96	0.979
3	Gamma	0.951	0.986	Johnson SB	0.98	0.994
	Johnson SB	0.934	0.99	Gamma	0.927	0.987
	Chi Squared	0.92	0.977	Beta	0.892	0.99
4	Beta	0.95	0.984	Gamma	0.985	0.989
	Gamma	0.937	0.989	Johnson SB	0.964	0.978
	Pearson 6	0.896	0.889	Beta	0.96	0.976
5	Erlang	0.761	0.655	Gamma	0.994	0.994
	Gamma	0.707	0.703	Erlang	0.994	0.993
	Inv. Gaussian	0.59	0.487	Johnson SB	0.991	0.991
6	Gamma	0.994	1	Johnson SB	0.999	0.998
	Johnson SB	0.98	0.998	Gamma	0.986	0.997
	Beta	0.975	0.989	Beta	0.916	0.953
7	Inv. Gaussian	0.996	0.977	Johnson SB	0.96	0.989
	Lognormal	0.989	0.959	Beta	0.944	0.966
	Gamma	0.952	0.988	Gamma	0.721	0.917
8	Pearson 6	0.815	0.884	Johnson SB	0.908	0.946
	Inv. Gaussian	0.73	0.814	Inv. Gaussian	0.85	0.938
	Gamma	0.762	0.679	Gamma	0.768	0.924

Table 4.15 Distribution fitting results for activity two completion times with Exponential interarrivals.
Sample size: 1,000.

Case	Optimal	p(K-S)	p(A-D)	FV	p(K-S)	p(A-D)
1	Beta	0.957	0.974	Johnson SB	0.941	0.968
	Gamma	0.926	0.975	Gamma	0.847	0.95
	Pearson 6	0.803	0.848	Beta	0.797	0.928
2	Johnson SB	0.908	0.944	Gamma	0.995	0.999
	Beta	0.89	0.938	Johnson SB	0.997	0.995
	Gamma	0.771	0.96	Beta	0.957	0.989
3	Chi Squared	0.992	0.975	Beta	0.8	0.931
	Gamma	0.982	0.959	Gamma	0.803	0.916
	Johnson SB	0.986	0.938	Johnson SB	0.755	0.9
4	Johnson SB	0.97	0.997	Johnson SB	0.987	0.997
	Inv. Gaussian	0.967	0.987	Gamma	0.936	0.959
	Gamma	0.953	0.995	Inv. Gaussian	0.884	0.994
5	Gamma	0.903	0.975	Inv. Gaussian	0.993	0.997
	Erlang	0.945	0.926	Johnson SB	0.97	0.998
	Pearson 6	0.855	0.737	Lognormal	0.977	0.985
6	Inv. Gaussian	0.92	0.933	Johnson SB	0.983	0.997
	Lognormal	0.904	0.851	Inv. Gaussian	0.862	0.997
	Gamma	0.794	0.968	Gamma	0.838	0.963
7	Inv. Gaussian	0.978	0.991	Beta	0.938	0.956
	Johnson SB	0.972	0.99	Weibull	0.793	0.783
	Lognormal	0.992	0.967	Erlang	0.527	0.681
8	Johnson SB	0.985	0.978	Beta	0.858	0.919
	Inv. Gaussian	0.974	0.976	Gamma	0.819	0.931
	Lognormal	0.93	0.951	Chi Squared	0.718	0.833

Table 4.16 Distribution fitting results for project completion times with Weibull interarrivals.
Sample size: 1,000.

Case	Optimal	p(K-S)	p(A-D)	FV	p(K-S)	p(A-D)
9	Lognormal	0.979	0.997	Pearson 5	0.999	0.998
	Pearson 5	0.987	0.988	Lognormal	0.993	0.999
	Inv. Gaussian	0.946	0.999	Inv. Gaussian	0.975	0.999
10	Inv. Gaussian	0.909	0.945	Johnson SB	1	1
	Johnson SB	0.865	0.921	Gamma	1	1
	Lognormal	0.846	0.933	Inv. Gaussian	1	1
11	Gamma	0.747	0.872	Inv. Gaussian	0.865	0.907
	Inv. Gaussian	0.708	0.863	Pearson 6	0.887	0.849
	Beta	0.682	0.813	Erlang	0.828	0.886
12	Inv. Gaussian	1	1	Inv. Gaussian	0.999	1
	Lognormal	0.998	1	Johnson SB	0.996	0.999
	Gamma	0.999	0.99	Gamma	0.989	0.996
13	Gamma	0.705	0.849	Johnson SB	1	1
	Johnson SB	0.672	0.806	Gamma	0.999	1
	Erlang	0.636	0.81	Erlang	0.991	0.996
14	Beta	0.915	0.865	Gamma	0.949	0.971
	Gamma	0.839	0.916	Inv. Gaussian	0.935	0.98
	Erlang	0.861	0.831	Lognormal	0.912	0.967
15	Johnson SB	0.916	0.848	Johnson SB	0.898	0.984
	Gamma	0.865	0.843	Gamma	0.854	0.948
	Inv. Gaussian	0.854	0.85	Beta	0.733	0.959
16	Beta	0.994	0.987	Johnson SB	0.934	0.983
	Johnson SB	0.991	0.975	Gamma	0.904	0.982
	Gamma	0.98	0.98	Inv. Gaussian	0.889	0.98

Table 4.17 Distribution fitting results for activity one completion times with Weibull interarrivals.
Sample size: 1,000.

Case	Optimal	p(K-S)	p(A-D)	FV	p(K-S)	p(A-D)
9	Pearson 5	0.907	0.885	Pearson 5	0.761	0.924
	Lognormal	0.839	0.911	Lognormal	0.722	0.947
	Inv. Gaussian	0.784	0.916	Gamma	0.717	0.896
10	Inv. Gaussian	0.823	0.73	Lognormal	0.931	0.964
	Lognormal	0.858	0.686	Pearson 5	0.958	0.933
	Gamma	0.686	0.824	Inv. Gaussian	0.878	0.963
11	Gamma	0.931	0.969	Pearson 5	0.977	0.91
	Johnson SB	0.856	0.945	Lognormal	0.931	0.937
	Beta	0.693	0.861	Inv. Gaussian	0.872	0.939
12	Pearson 6	0.963	0.956	Johnson SB	0.983	0.996
	Johnson SB	0.886	0.984	Gamma	0.973	0.982
	Inv. Gaussian	0.873	0.974	Pearson 6	0.97	0.945
13	Pearson 6	0.988	0.974	Johnson SB	0.998	0.999
	Johnson SB	0.951	0.983	Gamma	0.995	0.999
	Inv. Gaussian	0.933	0.982	Inv. Gaussian	0.981	0.999
14	Inv. Gaussian	0.978	0.935	Gamma	0.98	0.995
	Lognormal	0.969	0.922	Inv. Gaussian	0.905	0.988
	Pearson 5	0.931	0.88	Lognormal	0.838	0.968
15	Inv. Gaussian	0.974	0.935	Gamma	0.986	0.994
	Lognormal	0.946	0.919	Erlang	0.99	0.975
	Pearson 5	0.872	0.873	Johnson SB	0.913	1
16	Pearson 6	0.974	0.872	Johnson SB	0.999	1
	Inv. Gaussian	0.903	0.891	Gamma	0.977	0.998
	Lognormal	0.833	0.837	Inv. Gaussian	0.974	1

Table 4.18 Distribution fitting results for activity two completion times with Weibull interarrivals.
Sample size: 1,000.

Case	Optimal	p(K-S)	p(A-D)	FV	p(K-S)	p(A-D)
9	Johnson SB	0.986	0.995	Gamma	0.962	0.931
	Inv. Gaussian	0.986	0.995	Inv. Gaussian	0.856	0.913
	Gamma	0.995	0.975	Pearson 6	0.787	0.866
10	Beta	0.943	0.982	Gamma	0.97	0.986
	Gamma	0.886	0.966	Johnson SB	0.918	0.975
	Inv. Gaussian	0.893	0.957	Beta	0.903	0.974
11	Johnson SB	0.725	0.852	Johnson SB	0.998	0.998
	Gamma	0.709	0.836	Gamma	0.993	0.998
	Inv. Gaussian	0.691	0.824	Erlang	0.99	0.997
12	Lognormal	1	1	Lognormal	0.995	0.999
	Pearson 5	0.999	0.997	Inv. Gaussian	0.99	1
	Inv. Gaussian	0.994	1	Gamma	0.975	0.983
13	Johnson SB	0.961	0.956	Inv. Gaussian	0.996	0.999
	Gamma	0.922	0.943	Lognormal	0.994	0.998
	Inv. Gaussian	0.838	0.969	Johnson SB	0.976	0.999
14	Johnson SB	0.99	1	Inv. Gaussian	0.975	0.996
	Pearson 6	0.989	0.999	Johnson SB	0.941	0.992
	Inv. Gaussian	0.973	1	Lognormal	0.931	0.993
15	Inv. Gaussian	1	1	Inv. Gaussian	0.926	0.991
	Lognormal	0.986	1	Gamma	0.931	0.969
	Pearson 6	0.99	0.945	Pearson 6	0.894	0.97
16	Johnson SB	0.99	0.997	Gamma	0.994	0.997
	Gamma	0.967	0.996	Inv. Gaussian	0.989	0.999
	Beta	0.981	0.977	Erlang	0.989	0.993

Figure 4.6 Project completion distribution fitting results for Case 13.

(a) PDF, Optimal; (b) CDF, Optimal; (c) PDF, FV; (d) CDF, FV.

Chapter

5

Conclusion

5.1 Summary

In this paper we have modeled the spontaneous volunteer assignment problem as a queueing system with randomly arriving and departing servers working, in parallel, on fixed and deterministic demand. Through an examination of literature surveys on emergency management research in Operations Research, as well as relevant research in volunteer assignment, we have shown that the model developed here is most appropriately applied to work associated with the Recovery phase of post-disaster relief efforts, the phase of Disaster Operations Management which has received the least amount of attention over the years, as evidenced by Altay and Green (2006) and Galindo and Batta (2013). We framed the problem as a continuous time Markov Chain (CTMC) with the objective of maximizing reward by completing units of work. We then created an equivalent DTMC representation using uniformization. The finite nature of both the state space and the action space, as well as the presence of an absorbing state with zero reward, allowed for the use of Policy Iteration to generate optimal policies without resorting to averaging or discounting to ensure convergence. We initially developed eight different cases by varying the work rates at each of the locations in addition to varying the maximum capacity and starting demand at one of the locations.

Using common-sense practices as motivation, we developed three different heuristic policies:

the Fewest Volunteers policy, the Largest Demand policy, and the Random policy. We showed that, although the optimal policies are impractical to implement, they are for the most part a mixture of the Fewest Volunteers policy and the Largest Demand policy. Additionally we noted, but did not prove, that the Optimal Policy exhibits a logical switching-curve structure.

In order to test the performance of the Optimal and heuristic policies, an Arena model was developed to simulate the system, using time to complete all work as a proxy for the DTMC objective of maximizing reward. The Arena model used Common Random Number Streams across all simulations, allowing for paired-sample comparisons between the Optimal and heuristic policies. Through our simulations we have found that both the Largest Demand and Random heuristic policies perform very poorly when compared with the Optimal Policy and the Fewest Volunteers policy. They were found to be significantly suboptimal in all cases. Meanwhile, the Fewest Volunteers policy performed very well when compared to the Optimal Policy. In most cases the difference in performance was not statistically significant. This is notable for two reasons. First, the Fewest Volunteers policy is much more practical to implement than the Largest Demand Policy. Second, the Fewest Volunteers policy performed well even when work rates were dissimilar, a factor which the heuristic policy does not take into account.

We then performed distribution-fitting analysis on project completion times and individual activity completion times under both the Optimal Policy and the Fewest Volunteers policy. The other two heuristic policies were ignored in this analysis because they were shown to be significantly suboptimal. We found that a number of distributions turned out to be tremendous fits for both project completion times and individual activity completion times. These include Beta, Gamma, Inverse Gaussian, Johnson SB, and Lognormal.

Next, we used empirical data for volunteer interarrival times provided by Lodree et al. (2015) in order to simulate the system using Weibull-distributed interarrivals. We used the Exponential fit to the same interarrivals in order to generate optimal policies, then used the Weibull distribution to simulate the arrivals within the Arena model. We repeated both the paired-sample comparison and distribution-fitting analysis in this scenario. The Weibull distribution turned out to be nearly identical to the Exponential fit and so we did not expect the results in this scenario to differ much from the scenario with Exponential interarrivals. The paired-sample comparison results were nearly the same, as were the distribution-fitting results.

5.2 Policy Implications

We have shown via Policy Iteration that neither the Fewest Volunteers policy nor the Largest Demand policy is optimal. However, we have also shown that the Optimal Policy is impractical for various

reasons. Thankfully, despite being strictly suboptimal in the context of the DTMC formulation, the Fewest Volunteers policy has been shown to be nearly optimal in the simulation model. The nearly optimal performance of the Fewest Volunteers policy, even in cases with dissimilar work rates at the queues, speaks strongly for the value of this heuristic policy. In addition, in a organized volunteer environment the Fewest Volunteers policy would require just a headcount, making it both efficient and easy to implement. Therefore, we believe that in the context of the Recovery phase of disaster relief operations, when faced with the harrowing task of assigning unknown numbers of randomly arriving volunteers, the Fewest Volunteers policy is a good heuristic to follow.

5.3 Limitations

Framing the spontaneous volunteer assignment problem as a DTMC requires assuming that all time delays involved are Exponential. This makes the model an idealized abstraction, like with most models which are tractable enough to permit solutions. Courtesy of Lodree et al. (2015), we know that in at least one instance the Exponential distribution was a decent fit for interarrival times. However, that same empirical study shows us that this is not always the case.

In using Weibull-distributed interarrival times in the simulation, the goal was to test the performance of an Optimal Policy designed for Exponential interarrivals in a situation for which it was not designed. We showed that using Weibull-distributed interarrival times had no significant effects on performance; however, the Weibull fit being nearly identical to the Exponential fit greatly reduces the significance of the result.

There are many considerations related to the volunteer assignment problem which we did not address. We did not allow for group arrivals of volunteers, we did not allow for reassignment of volunteers, and we did not address volunteer preferences or abilities. These are all real concerns with effects on productivity, efficiency, and each individual volunteer's propensity to volunteer in the future. Our model strictly concerned the assignment of volunteers with the sole objective of maximizing work completion, while treating all volunteers as identical. Given that the context of the problem is disaster relief, maximizing work completion is of course a noble goal; however, it is not the only consideration.

5.4 Future Work

One primary interest for us is continuing the examination of the effect of non-Exponential interarrivals. This could also be extended to non-Exponential abandonment or service times. We would like to incorporate volunteer preferences and abilities, as well as group arrivals, to make the model more

realistic. Additionally, one could incorporate a "fairness" factor in the form of costs incurred when there is a large disparity between individual activity completion times.

Another extension would be the addition of more queues. Here we only considered two, but adding more would be easy to implement in terms of both the model formulation and the Arena model. Adding more queues would increase the state space and thus the complexity of the solution and the runtimes of the policy iteration algorithm. However, given the low runtimes of both the policy iteration algorithm and the Arena model, we believe there is room for expansion before the problem becomes intractable. It is unclear whether or not the switching-curve structure of the Optimal Policy and the nearly-optimal performance of the Fewest Volunteers policy would hold, but we see no reason to believe that they would not.

In volunteer management scenarios, it is common for organizers to assign volunteers to do what could be called background or support work. Examples of this are arranging housing or packing lunches for incoming volunteers, or helping with transportation — i.e., anything that is not directly related to the work at hand but is necessary to support the volunteers who are directly doing field work. This practice alleviates some of the burden that would otherwise be placed on emergency managers, allowing them to focus more on their primary duties. Within the model, this could be treated as a third queue whose number of working servers limits the number of volunteers able to work at the other queues. In effect, no volunteers could be assigned to the working queues without there being volunteers already working in a support capacity.

In this paper we were able to abstract the spontaneous volunteer assignment problem as a queuing system and solve it as a DTMC. Although the Optimal Policy is impractical, we have shown that the common-sense Fewest Volunteers heuristic is an implementable solution which yields good results and seems to be robust; however, this remains to be fully tested.

BIBLIOGRAPHY

- [1] Akgun, O. T., Richter, R., Woff, R. (2011) Multiple-server system with flexible arrivals. *Advanced Applied Probability*, 43: 985-1004.
- [2] Altay, N., Green, W. (2006) OR/MS research in disaster operations management. *European Journal of Operational Research*, 175: 475-493.
- [3] Barnett, R. V., Flint, C. G. (2007) Community Volunteers: The Front Line of Disaster Response. *The International Journal of Volunteer Administration*, 23(4): 52-56.
- [4] Bertsekas, D. P. (2001) Dynamic programming and optimal control, Vol. 2 (2nd edn.) Nashua: Athena Scientific.
- [5] Falasca, M., Zobel, C. (2012) An optimization model for volunteer assignments in humanitarian organizations. *Socio-Economic Planning Sciences*, 46: 250-260.
- [6] Fernandez, L. S., Barbera, J. A., van Dorp, J. R. (2005) Strategies for Managing Volunteers during Incident Response: A Systems Approach. George Washington University.
- [7] Galindo, G., Batta, R. (2013) Review of recent developments in OR/MS research in disaster operations management. *European Journal of Operational Research*, 230(2): 201-211.
- [8] Glass, T. (2001) Understanding Public Response to Disasters. *Public Health Reports*, 116(2): 69-73.
- [9] Johri, P. K. (1989) Optimality of the shortest line discipline with state-dependent service rates. *European Journal of Operations Research*, 41: 157-161.
- [10] Lodree, E.J., Davis, L.B., Cook, R.A (2015) Managing relief center convergence following large-scale disaster events. Unpublished manuscript.
- [11] Lowe, S., Fothergill, A. (2002) A Need to Help: Emergent Volunteer Behavior after September 11th.
- [12] Sampson, S. (2006) Optimization of volunteer labor assignments. *Journal of Operations Management*, 24: 363-377.

APPENDICES

Appendix

A

Policy Iteration Algorithm

```
package policyIteration;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util. ;
import java.util.concurrent.TimeUnit;

import policyIteration.ReadData;

public class PolicyIteration {

    public static ArrayList<int[]> allstates = new ArrayList<>();

    public static void main(String[] args)
        throws FileNotFoundException , UnsupportedEncodingException {

        final long startTime = System.currentTimeMillis();

        boolean RewardEveryTime = true;
```

```
//First we need to read the data.txt file to get the parameters/info
ReadData Y = new ReadData();
Y.openFile();
Vector data = Y.readFile();
Y.closeFile();

String filename = String.valueOf(data.get(0));
String current = String.valueOf(data.get(0));

int a = 0;

double[] mu = new double[2];
double gamma = (double) data.get(1);
mu[0] = (double) data.get(2);
mu[1] = (double) data.get(3);
double lambda = (double) data.get(4);
double reward = (double) data.get(5);
double eps = (double) data.get(6);
double holding1 = (double) data.get(7);
int NOL = (int) data.get(8);
int[] MaxV = new int[NOL];
int[] MaxD = new int[NOL];
for(int i=0; i<NOL; i++){
    MaxV[i] = (int) data.get(i+9);
    System.out.println("MaxV_" + MaxV[i]);
    MaxD[i] = (int) data.get(i+9+NOL);
    System.out.println("MaxD_" + MaxD[i]);
}
int holding = (int) holding1;

//Generating all possible states
getStates(MaxV, MaxD);
System.out.println("States_have_been_generated!");
int b = allstates.size();

//Utility arrays
double[] OldUtil = new double[b];
double[] NewUtil = new double[b];
for(int i=0; i<b; i++){
    OldUtil[i]=0;
    NewUtil[i]=0;
}
```

```

//Solution array
double[] Opt = new double[b];

int[] OldPolicy = new int[b];
int[] NewPolicy = new int[b];

//Setting up model parameters
int MV=0;
for(int i=0; i<MaxV.length; i++){
    MV+=MaxV[i];
}

double beta = lambda + MaxV[0] mu[0] + MaxV[1] mu[1] + MV gamma;

double[] params = new double[7];
double error = eps;
params[0] = beta;
params[1] = gamma;
params[2] = mu[0];
params[3] = mu[1];
params[4] = lambda;
params[5] = reward;
params[6] = holding;

System.out.println("");
System.out.println(":: Parameters::");
System.out.println("Gamma:\t" + gamma);
System.out.println("Mu_1:\t" + mu[0]);
System.out.println("Mu_2:\t" + mu[1]);
System.out.println("Lambda:\t" + lambda);
System.out.println("Beta:\t" + beta);
System.out.println("Reward:\t" + reward);
System.out.println("Error:\t" + error);
System.out.println("Holding:\t" + holding);
System.out.println("");

//Initial values
boolean TryFV = true;
int[] RandomActions = new int[3];
RandomActions[0]=0; RandomActions[1]=1; RandomActions[2] = 3;
Random RandChoice = new Random();

```

```
if (TryFV){
for(int i = 0; i<b; i++){
    int[] state = allstates.get(i);
    if(state[2]==0){
        if(state[3]==0){
            OldPolicy[i] = 3;
            continue;
        }else{
            OldPolicy[i] = 1;
            continue;
        }
    }else{
        if(state[3]==0){
            OldPolicy[i] = 0;
            continue;
        }
    }

    //So now we know both demands are nonzero.
    if((state[0]==MaxV[0])&&(state[1]==MaxV[1])){
        OldPolicy[i]=3;
    }
    if(state[0]<state[1]){
        OldPolicy[i] = 0;
        continue;
    }else if(state[0]>state[1]){
        OldPolicy[i] = 1;
        continue;
    }else if(state[0] == state[1]){ //Volunteers equal
        if(state[2]==state[3]){ //Demand equal
            if(state[0]==MaxV[0]){
                OldPolicy[i] = 1;
                continue;
            }
            if(state[1] == MaxV[1]){
                OldPolicy[i] = 0;
                continue;
            }else{
                OldPolicy[i] = 3;
                continue;
            }
        }else{

```



```
        if (state[2]<state[3]){
            OldPolicy[i]=1;
            continue;
        }
        if (state[2]>state[3]){
            OldPolicy[0]=0;
        }
    }
}
} else {
    for (int i = 0; i<b; i++){
        int[] state = allstates.get(i);
        if (state[2]==0){
            if (state[3]==0){
                OldPolicy[i] = 3;
                continue;
            } else {
                OldPolicy[i] = 1;
                continue;
            }
        } else {
            if (state[3]==0){
                OldPolicy[i] = 0;
                continue;
            }
        }
    }

    int RandNum = RandChoice.nextInt(2);

    OldPolicy[i] = RandNum;
}
}

int n=1;
double newsum=0;
double newavg=0;

//C_bar and C_floor for the value iteration algorithm
double cmax=Double.NEGATIVE_INFINITY;
double cmin=Double.POSITIVE_INFINITY;
double difference=0;
```

```

for(int i=a; i<b; i++){
    NewUtil[i] -= newavg;
}

/ Policy Iteration Algorithm
Begin with arbitrary policy PI
STEP ONE : INITIALIZATION
    get initial values following arbitrary policy PI
STEP TWO : EVALUATION
    while difference > error
        for each state s
            g = G(s)
            find G(s) following PI
            difference = max( difference | abs(g - G(s)))
        end
    end
STEP THREE : IMPROVEMENT
    noChange = true
    for each state s
        a = PI(s), the prescribed action
        find optimal action then set
            PI(s) = optimal action
        if a != PI(s)
            noChange = false
            a = PI(s)
    if noChange, you're done
    else , GOTO STEP TWO
/

boolean Finished = false;

System.out.println(" :: Begin_Iteration :: ");
System.out.println();

double[] result = new double[2];
int n3 = 0;
while(Finished == false){
    Finished = true;
    n3++;
    for(int i=0; i<b; i++){
        OldUtil[i] = 0;
    }
}

```

```

    }

//STEP ONE: INTIALIZATION
for(int i=1; i<b; i++){
    result = determineUtility(i, allstates.get(i), OldUtil,
        params, MaxV, n, OldPolicy, RewardEveryTime);
    NewUtil[i]=result[1];
}
for(int i=1; i<b; i++){
    OldUtil[i] = NewUtil[i];
}

//STEP TWO: EVALUATION
int n2 = 0;
difference = Double.POSITIVE_INFINITY;
while(difference > error){
    n2++;
    difference = 0;

    for(int i=1; i<b; i++){
        result = determineUtility(i, allstates.get(i), OldUtil,
            params, MaxV, n, OldPolicy, RewardEveryTime);
        NewUtil[i] = result[1];
        difference = Math.max(difference, Math.abs(OldUtil[i]-NewUtil[i]));
    }
    for(int i=1; i<b; i++){
        OldUtil[i] = NewUtil[i];
    }

    System.out.println("Maximum_error_is_" + difference + ".");
    System.out.println("The_step_is_" + n2 + ".
    Policy_try_number:_ " + n3 + ".");
}

System.out.println("The_benefit_function
    has_converged_after_" + n2 + "_steps.");

//STEP THREE : IMPROVEMENT
for(int i=1; i<b; i++){
    int[] state = allstates.get(i);

    boolean NotSpecial = true;

```

```
//First, take care of the special cases.
if (state[2]+state[3]==0){
    Opt[i]=3;
    NotSpecial = false;
} else if (state[3]==0){
    Opt[i]=0;
    NotSpecial = false;
} else if (state[2]==0){
    Opt[i]=1;
    NotSpecial = false;
}
double[] result_old = new double[2];
double[] result_opt = new double[2];

if (NotSpecial){
    int[] choice1 = state.clone();
    choice1[0]=Math.min(choice1[0]+1,MaxV[0]);
    int[] choice2 = state.clone();
    choice2[1]=Math.min(choice2[1]+1,MaxV[1]);

    double benefit1 = findUtility(choice1,NewUtil);
    double benefit2 = findUtility(choice2,NewUtil);

    int comparison = Double.compare(benefit1,benefit2);

    if (comparison>0){
        Opt[i]=0;
    } else if (comparison<0){
        Opt[i]=1;
    } else{
        Opt[i]=3;
    }
}

    int OptAction = (int) Opt[i];
    //int OptAction = (int) result_opt[0];
    if (OldPolicy[i]==OptAction){
        continue;
    } else{
        Finished = false;
        OldPolicy[i]=OptAction;
    }
}
```

```

    }
    System.out.println("The_optimal_policy_was_found_after_" + n3 + "_tries.");

    //Find the optimal decisions now that the solution has converged
    for(int i=0; i<b; i++){
        Opt[i] = OldPolicy[i];
    }

    final long endTime = System.currentTimeMillis();
    final long elapsedTime = endTime - startTime;
    String time = String.format("%d_min, %d_sec",
        TimeUnit.MILLISECONDS.toMinutes(elapsedTime),
        TimeUnit.MILLISECONDS.toSeconds(elapsedTime) -
        TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes(elapsedTime))
    );
    System.out.println("Elapsed_time:_" + time);

    //::::::::::::::::::::: START OF THE METHODS :::::::::::::::::::::::

    //Determine Utility using fixed policy
    public static double[] determineUtility(int j, int[] state,
        double[] OldUtil, double[] params, int[] MaxV,
        int n, int[] Policy, boolean RewardEveryTime){
        double[] result = new double[2];
        double beta = params[0];
        double gamma = params[1];
        double[] mu = new double[2];
        mu[0] = params[2];
        mu[1] = params[3];
        double lambda = params[4];
        double reward = params[5];
        double holding = params[6];

        //"a" divides the state into two halves: volunteer half and demand half
        int a = (state.length)/2;

        //Gamma Portion

```

```

double gtotal = 0;
double gammatotal = 0;
double util = 0;
for(int i=0; i<a; i++){
    int[] newstate = state.clone();
    if(newstate[i]==0){
        util = OldUtil[j];
    } else{
        newstate[i] -= 1;
        util = findUtility(newstate, OldUtil);
    }
    gtotal+= gamma state[i];
    gammatotal+=gamma state[i] util;
}

//Mu Portion
double mtotal = 0;
double mutotal = 0;
util=0;
for(int i=0; i<a; i++){
    int[] newstate = state.clone();
    if(newstate[i+a]>0){
        newstate[i+a] = Math.max(newstate[i+a] - 1, 0);
        newstate = findNewState(newstate);
        util = findUtility(newstate, OldUtil);
        mtotal += mu[i] state[i];
        if(RewardEveryTime){
            mutotal += mu[i] state[i] ( util + beta reward);
        } else{
            if(newstate[2]+newstate[3]==0){
                mutotal += mu[i] state[i] ( util + 100 beta reward);
            } else{
                mutotal += mu[i] state[i] ( util);
            }
        }
    }
}

//Lambda Portion
double max = Double.NEGATIVE_INFINITY;
double ind = 0;

```

```

    if ((state[0]==MaxV[0])&&(state[1]==MaxV[1])){
        max=OldUtil[j];
    } else{
        int choice = Policy[j];
        int[] newstate = state.clone();
        //System.out.println("Choice is " + choice);
        if (choice==3){
            int[] newstate0 = state.clone();
            int[] newstate1 = state.clone();
            newstate0[0] = Math.min(newstate[0]+1, MaxV[0]);
            newstate1[1] = Math.min(newstate[1]+1, MaxV[1]);
            double ben0 = findUtility(newstate0, OldUtil);
            double ben1 = findUtility(newstate1, OldUtil);
            max = (ben0+ben1)/2;
        } else{
            newstate[choice] = Math.min(newstate[choice]+1, MaxV[choice]);
            max = findUtility(newstate, OldUtil);
        }
    }
    double ltotal = lambda max;

    //Fictitious Transition Portion

    double fict = (beta - lambda - gtotal - mtotal) OldUtil[j];

    //Total
    double total = (holding (state[2]+state[3])
        + gammatotal + mutotal + ltotal + fict)/(beta);

    result[0] = (double) Policy[j];
    result[1] = total;
    return result;
}

//Determine Utility
public static double[] getUtility(int j, int[] state, double[] OldUtil,
    double[] params, int[] MaxV, int n,
    boolean RewardEveryTime){
    double[] result = new double[2];
    double beta = params[0];

```

```
double gamma = params[1];
double[] mu = new double[2];
mu[0] = params[2];
mu[1] = params[3];
double lambda = params[4];
double reward = params[5];
double holding = params[6];

// "a" divides the state into two halves: volunteer half and demand half
int a = (state.length)/2;

//Gamma Portion
double gtotal = 0;
double gammatotal = 0;
double util = 0;
for(int i=0; i<a; i++){
    int[] newstate = state.clone();
    if(newstate[i]==0){
        util = OldUtil[j];
    } else{
        newstate[i] -= 1;
        util = findUtility(newstate, OldUtil);
    }
    gtotal+= gamma state[i];
    gammatotal+=gamma state[i] util;
}

//Mu Portion
double mtotal = 0;
double mutotal = 0;
util=0;
for(int i=0; i<a; i++){
    int[] newstate = state.clone();
    if(newstate[i+a]>0){
        newstate[i+a] = Math.max(newstate[i+a] - 1, 0);
        newstate = findNewState(newstate);
        util = findUtility(newstate, OldUtil);
        mtotal += mu[i] state[i];
        if(RewardEveryTime){
            mutotal += mu[i] state[i] ( util + beta reward);
        } else{
```



```

        if (newstate[2]+newstate[3]==0){
            mutotal += mu[i] state[i] ( util + 100 beta reward);
        }else{
            mutotal += mu[i] state[i] ( util);
        }
    }
}
}

```

//Lambda Portion

```

double max = Double.NEGATIVE_INFINITY;
double ind = 0;

if (( state[0]==MaxV[0])&&( state[1]==MaxV[1])){
    max=OldUtil[j];
} else{
    int[] options = getOptions(state , MaxV);
    double[] benefits = new double[options.length];
    double[] feasible = new double[options.length];
    for(int i=0; i<options.length; i++){
        if(options[i]==99){
            benefits[i] = OldUtil[j];
            feasible[i]=0;
        } else{
            int[] newstate = state.clone();
            if(newstate[i]<MaxV[i]){
                newstate[i] = Math.min(newstate[i] + 1, MaxV[i]);
                benefits[i] = findUtility(newstate , OldUtil);
            } else{
                benefits[i]=OldUtil[j];
            }
            feasible[i]=1;
        }
    }
}

//both assignments are feasible
if(feasible[0]+feasible[1]==2){
    for(int i=0; i<benefits.length; i++){
        if(benefits[i]>max){
            max = benefits[i];
            ind = i;
        }
    }
}

```

```

        if (benefits[0]==benefits[1]){
            ind = 3;
        }
    }

    } else if (feasible[0]==0){
        //both infeasible
        if (feasible[1]==0){
            max=0;
            //only 1 feasible
        } else {
            max=benefits[1];
            ind=1;
        }
    } else {
        //only 0 feasible
        if (feasible[1]==0){
            max=benefits[0];
            ind=0;
        }
    }
}

double ltotal = lambda max;

//Fictitious Transition Portion

double fict = (beta - lambda - gtotal - mtotal) OldUtil[j];

//Total
double total = (holding (state[2]+state[3])
+ gammatotal + mutotal + ltotal + fict)/(beta);

result[0] = ind;
result[1] = total;
return result;
}

//Determine Stored Utility

```

```
public static double findUtility(int[] state , double[] OldUtil){  
    int ind=0;  
    for(int i=0; i<allstates.size(); i++){  
        if(Arrays.equals(state , allstates.get(i))){  
            ind = i;  
            break;  
        }  
    }  
  
    double util = OldUtil[ind];  
    return util;  
}
```

//Determining New State

```
public static int[] findNewState(int[] test_state){  
    int[] state = new int[test_state.length];  
    int a = (state.length)/2;  
    for(int i=a; i<state.length; i++){  
        state[i] = test_state[i];  
    }  
    for(int i=0; i<a; i++){  
        if(test_state[i+a] == 0){  
            state[i]=0;  
        } else {  
            state[i]=test_state[i];  
        }  
    }  
  
    return state;  
}
```

//Finding Possible Assignments

```
public static int[] getOptions(int[] state , int[] MaxV){  
    int a = (state.length)/2;  
    int[] options = new int[a];  
    for(int i=0; i<options.length; i++){  
        if(state[i+a]<=0){  
            options[i] = 99;  
        } else {  
            options[i] = i;  
        }  
    }  
    return options;  
}
```

```
}

```

```
//Generating States

```

```
public static void getStates(int[] MaxV, int[] MaxD){
    //ArrayList<int[]> states = new ArrayList<>();
    int[] x = new int[MaxV.length+MaxD.length];
    for(int i=0; i<MaxV.length; i++){
        x[i]=MaxV[i];
    }
    for(int i=MaxV.length; i<(MaxV.length+MaxD.length); i++){
        x[i]=MaxD[i-MaxV.length];
    }

    int[] n = new int[x.length];

    printPermutations(n,x,0);

}

```

```
public static void printPermutations(int[] n, int[] Nr, int idx) {
    if (idx == n.length) { //stop condition for the recursion [base clause]
        boolean test = true;
        int[] copy = n.clone();
        int a = copy.length;
        for(int i=a/2; i<a; i++){
            if ((copy[i]==0)&&(copy[i-a/2]>0)){
                test = false;
                break;
            }
        }
        if(test){
            allstates.add(copy);
        }

        return;
    }
    for (int i = 0; i <= Nr[idx]; i++) {
        n[idx] = i;

        printPermutations(n, Nr, idx+1); } } }

```

Appendix

B

Arena Model Documentation

The Arena model is composed of three parts: the Upper Level, corresponding to the assignment of incoming volunteers to the queue specified by the chosen assignment policy, and the two Lower Level submodels which control volunteer abandonment and work completion. There is one such submodel for each queue. The two submodels are essentially identical and so we will only examine one in this appendix.

B.1 Upper Level

The Upper Level model governs the assignment process. It is straightforward and does not require much explanation. Volunteers arrive using Random Number Stream 1 (RNS1). Every time an arrival occurs, the system checks the current state of the system, then reads the statelist / assignment policy file to find the prescribed action. Once the action is found, the incoming volunteer is assigned its attributes **VolunteerID**, **DelayToDeparture** (using RNS2) and **DepartureTime**. **VolunteerID** is simply the number corresponding to this volunteer's arrival, e.g. two if it is the second arrival. **VolunteerID** is each volunteer's unique identifier and is used in the Activity Process submodels to correctly schedule

and execute work completion.

$$\mathbf{DelayToDeparture} = EXPO(1/\gamma, 2),$$

$$\mathbf{DepartureTime} = \mathbf{DelayToDeparture} + TNOW.$$

Once those attributes are assigned, the volunteer is assigned to whichever activity process submodel the assignment policy prescribed. An overview of the Upper Level assignment model is in Fig. B.1. Two detail images are found in Fig. B.2 and Fig. B.3.

Also included in the Upper Level are processes for reading the statelist / assignment policy and for tracking the time until the end of each run. These are shown in Fig. B.4 and Fig. B.5, respectively.

Figure B.1 Flow of volunteers, upper level model, overview.

Figure B.2 Flow of volunteers, upper level model, detail 1.

Figure B.3 Flow of volunteers, upper level model, detail 2.

Figure B.4 Read assignment policy.

Figure B.5 Track time until the end of run.

B.2 Lower Level

The Lower Level activity process submodels require more explanation. Great care must be taken to ensure that work completion and volunteer departures supercede each other correctly. An overview of the submodel for Activity Process 1 is found in Fig. B.6.

Figure B.6 Work and departure process, lower level model, overview.

Correctly scheduling unit work completions and departures requires the following:

- VolunteerID**(*attr.*): each volunteer's unique identifier,
- LatestVolunteerID**(*var.*): VolunteerID governing the scheduling of work completion,
- LatestUnitWorkCompletionTime**(*var.*): the most recently scheduled unit work completion time,
- FinishingVID**(*var.*): the VolunteerID associated with a given unit work completion,
- DepartingVID**(*var.*): the VolunteerID associated with a given departure.

Copies of each volunteer in Activity Process 1 are held in the Activity 1 Volunteer pool queue, where they remain until either their departure time arrives or all work is completed at Activity 1, prompting all volunteers to depart immediately. This is shown in Fig. B.7. The ranking scheme used in the Activity 1 Volunteer pool queue is the LVF rule, using **VolunteerID**. That is, the rank of 1 goes to the volunteer in the queue with the highest VolunteerID: the most recent arrival to Activity Process 1.

Figure B.7 Work and departure process, lower level model, split between work and departure.

To show the importance of this queue and this ranking, consider the following scenario. Say there are three volunteers currently in the Activity 1 Volunteer pool queue, with VolunteerIDs 1, 2, and 5, respectively. Then the first rank volunteer in the queue is volunteer 5. When a unit of work is scheduled to be completed, the time until the work is completed and the work completion time are defined by

$$\text{DelayToUnitWorkCompletion} = \text{EXPO}(1/(v_1\mu_1), 4),$$

$$\text{UnitWorkCompletionTime} = \text{TNOW} + \text{DelayToUnitWorkCompletion}.$$

If an arrival occurs after this unit work completion is scheduled but before the unit work completion is set to occur, then the work completion should not occur because now the queue is completing units of work according to $\text{EXPO}(1/((v_1 + 1)\mu_1), 4)$. Similarly, if any of the volunteers in Activity Process 1 departs the system during this time, then the work also needs to be rescheduled. Thus we require two checks to ensure that work is performed correctly and superceded when it should be.

B.2.1 Work Completion Track

Please refer to Fig. B.8. If a volunteer is heading through this Work Completion track, the first thing that occurs is three assignments are made: **DelayToUnitWorkCompletion**, **UnitWorkCompletionTime**, and **LatestUnitWorkCompletionTime**. Next, the **DelayToUnitWorkCompletion** is incurred. Then we assign **FinishingVID** = **VolunteerID**. Next we have the first work completion check, where the volunteer is searched for in the Activity 1 Volunteer pool queue. If the volunteer is not found, this means a volunteer departure occurred after the unit work completion was scheduled but before it was set to occur. If this occurs then the work completion is tallied as a superceded work completion and is not allowed to occur. If the volunteer is in the queue, then we move to the second check. Here, we check the following conditions:

$$\text{LatestVolunteerID} == \text{VolunteerID},$$

$$\text{LatestUnitWorkCompletionTime} == \text{UnitWorkCompletionTime}.$$

Remember that **LatestVolunteerID** is determined by the most recent volunteer to arrive to Activity Process 1 who is still in the Activity 1 Volunteer pool queue. Thus, this is a check to see whether or not there has been another arrival after the unit work completion was scheduled but before it was set to take place. If both conditions are false, then the unit work completion is superceded by the more recent one. Otherwise, the work completion is allowed to be completed. Next, if there is still work left to be done at Activity Process 1, the volunteer is routed back to the module which scheduled the unit work completion. Else, the activity is ended and all volunteers in the Activity 1 Volunteer pool queue

are instructed to depart.

Figure B.8 Work and departure process, lower level model, Work Completion track.

B.2.2 Departure Track

Please refer to Fig. B.9. This is an overview of the Departure Track. First, there is a delay equal to the volunteer's **DelayToDeparture**. Then there is an assignment, variable **DepartingVID = VolunteerID**. Next, we check to see if the volunteer is in the Activity 1 Volunteer pool queue. It is possible that work has been completed at Activity 1, which would force all volunteers in the Activity 1 Volunteer pool queue to leave. If the volunteer is there, the volunteer is removed from the Activity 1 Volunteer pool queue and is tallied as a departing volunteer. Then the number of volunteers working at Activity 1 is updated to reflect the departure. Next, the system checks whether or not there is work to be scheduled. If there is work left to be done at Activity 1 and there are volunteers still in the Activity 1 Volunteer pool queue, then work needs to be scheduled. To do this, we first update **LatestVolunteerID** using the volunteer in the Activity 1 Volunteer pool queue who is now the most recent volunteer after considering the recent departure, that is, the volunteer with rank 1. A copy of this volunteer is made from the Activity 1 Volunteer pool queue and is sent towards the Work Completion track to begin scheduling work.

Figure B.9 Work and departure process, lower level model, Departure track.