A Metaheuristic for Autonomous and Self-Organizing Consensus Formation Inspired by Honey Bee Nest Site Selection Behavior

by

Alexander Speros Mentis

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
December 13, 2014

Approved by

Levent Yilmaz, Chair, Professor of Computer Science
Saâd Biaz, Professor of Computer Science
N. Hari Narayanan, Professor of Computer Science
David Umphress, Professor of Computer Science
T. Randolph Beard, Professor of Economics

Abstract

This research presents a self-organizing system for engineering consensus among cooperating, distributed processes and studying consensus negotiation in natural systems that use the quorum sensing mechanism. The presented method combines metaheuristic techniques and a negotiation protocol based on honey bee nest site selection behavior in order to optimize the collective social utility of a group decision. Unlike existing negotiation and voting protocols, the proposed metaheuristic accommodates negotiation of consensus from among two or more decision values. It is decentralized, self-organizing, and does not require a fully-connected network in which each process can communicate directly with each other. Benefits of these attributes are that there is no central point of failure, and processes can make informed decisions using only information acquired from immediate neighbors, rather than requiring the consolidation and evaluation of global preferences. The proposed metaheuristic is modeled as an agent-based system using the Repast modeling and simulation framework, and this model is used to conduct simulation experiments in accordance with the Design of Experiments methodology to analyze the Honey Bee Consensus metaheuristic and its performance on multiple social network models. As the Honey Bee Consensus metaheuristic is an extension and new application of a recently-proposed agent-oriented Quorum Sensing pattern, the presented understanding of its parameter and topology influence is of value in steering the behavior of self-organizing systems based on the Quorum Sensing pattern language. Additionally, understanding how network topology influences consensus formation has applications spanning decision theory, social choice theory, network science, and control theory. Significant outcomes of this research include a description of the proposed metaheuristic and the identification of a quorum size to population ratio that provides optimal speed-accuracy tradeoff for a range of population sizes and social network models.

Acknowledgements

I have many family members, friends, and colleagues to thank for their support and encouragement over the past few years. I would like to first thank my wife, Beth, who shared every day of this amazing adventure with me. She cheered me on through my successes and lent me unconditional support during the challenging times. I thank my mother for giving me the foundation I needed for all of my achievements in life, including this one. I would also like the thank the United States Army for sponsoring my graduate education and my colleagues at the United States Military Academy who showed confidence in my abilities by selecting me for this fellowship and who gave me exceptionally good advice for achieving my academic goals.

Thank you to the entire Auburn family, but especially my advisor, Dr. Levent Yilmaz, and my dissertation committee who provided critical motivation, mentorship, and guidance, for which I am very grateful. Dr. Fadel Megahed saved me hundreds of hours of simulation time by providing me with access to his cloud computing resources. The Computer Science and Software Engineering department and the Graduate School provided financial support for conference attendance and JoAnn Lauraitis was kind enough to process all of the paperwork associated with that support. Finally, thanks to my friends, fellow graduate students, and Donald Johnston for your moral support and motivation during this journey.

Table of Contents

List of Tables

List of Figures

1. Introduction

It is often necessary for the various components of a system to agree on a single value, or set of values, that indicates an action each of the components should perform or that is a computational result the system should return; this is known as the consensus problem (Barborak, Dahbura, & Malek, 1993; Fischer, Lynch, & Paterson, 1985). More recently, the problem has also been formulated as the "Democratic Primary Problem" (DPP) (Kearns & Tan, 2008), subsequently renamed the "Networked Biased Voting Problem" (NBVP) (Tan, 2010), with respect to achieving agreement on a value among individuals in a social network, in this case a party candidate for an election. The study of consensus formation encompasses both uncovering the mechanisms and processes by which social groups form consensus to solve problems and creating the ability to engineer consensus in artificial applications. The thesis of this dissertation is that a metaheuristic based on the technique used by honey bees to select a new nest site is feasible for engineering distributed consensus negotiation in social networks and yields insight into the influence of the quorum sensing pattern on consensus negotiation in social groups.

Examples of the application of engineering consensus for automated tasks include vehicle control, in which the vehicles must agree on the center of the formation or other control variables (Fax & Murray, 2004; Ren, Beard, & Atkins, 2005); agreement among distributed databases on whether or not to commit a transaction (Gifford, 1979; Pandey & Tripathi, 2012; Thomas, 1979); agreement among energy producers and consumers in a smart grid on factors related to the amount of energy to produce or consume at a particular time (Ziang & Mo-Yuen, 2011; Ziang, Xichun, & Mo-Yuen, 2011); agreement among road segments in a smart highway on the best combination of speed limits to reduce congestion; agreement among distributed sensors on the correct value to report (Britton & Sacks, 2004; Wokoma, Sacks, & Marshall, 2003); and agreement among agents or processes on another agent or process's reputation or trustworthiness (Yanbin & Yang, 2003). There are many other possible examples, but, in all of these cases, when individual components disagree on the value to act upon or report they must use a mechanism to negotiate a consensus on a single value.

The mechanisms by which this consensus is reached, its optimality with respect to the individuals and the collective, the application of the consensus, and the role of the underlying communication network in the formation of consensus spans several fields of study. Decision theory is concerned with the factors relevant to reaching a rational or optimal decision, sometimes under uncertainty or in situations with complex relationships between the decision factors (Horvitz, Breese, & Henrion, 1988). This may involve multiple decision makers; however, in the case of competing agents with at least partially conflicting interests, the problem becomes one also related to game theory, and in the case of combining individual, possibly differing or erroneous, preferences to reach the best decision for the collective as a whole, it relates to social choice theory (Shoham & Leyton-Brown, 2009; Wooldridge, 2009). Control theory studies, in part, the application of consensus to directing the control of dynamic systems with respect to leader-following and system convergence to, and stability in, consensus. Meanwhile, the study of the influence of the underlying communication network topology in a system's ability to reach consensus and the influence of individual voters and their locations in the network is of interest in the field of network science, which studies network representations of physical, biological, and social phenomena (such as communication and social interaction) with the goal of creating predictive models of these phenomena (*Network Science*, 2005).

1.1. Motivation and Scope

This research approaches consensus negotiation as a decision optimization problem, primarily from the perspective of social choice theory. Thus, it is assumed that individuals may have differing preferences due to self-interest, uncertainty, or error, but the overall goal of each individual is to achieve cooperative social welfare and consensus upon a decision that yields the highest utility for the collective. Traditionally, the solution to this problem is frequently implemented through the use of various negotiation or voting protocols; however, a weakness of most of these protocols is that they require a centralized mediator or centralized consolidation and evaluation of individual votes. These centralized solutions fail to scale well as the centralized coordinator's workload increases with the

number of participants, they introduce the requirement for a way to select the coordinator, and the selected coordinator becomes a central point of potential failure. Solutions for consensus without centralized control have their own weaknesses. Chief among these is the need for a fully-connected communication network so that each participant can communicate directly with every other one. Mesh networks like this are simply not possible in all applications. Furthermore, the feasibility of pairwise negotiation or market-style solutions, which allow participants to directly negotiate individual agreements with others, decreases as the population size increases because each pairwise agreement must be reconciled with all the previously-negotiated agreements. Meanwhile, existing consensus negotiation techniques that work on non-fully-connected networks are not designed to handle formation of consensus from among more than two possible outcomes.

In addition to these weaknesses, both centralized solutions and decentralized solutions in which negotiating parties expect consistency in the underlying communication network are unsuitable for use in communication networks with switching topologies, where an individual's neighbors change over time. Under these conditions in centralized regimes, the path to the coordinator is constantly changing, so efficiently ensuring that all votes reach the coordinator and broadcasting the coordinator's final decision to all participants becomes challenging as the network's spanning tree is potentially changing with each topology change. In decentralized solutions, as a negotiator's neighbors change, the negotiator loses track of previously-negotiated agreements and, with that, awareness of the progress being made toward consensus.

A protocol that facilitates distributed, self-organizing consensus formation without the requirement for a central coordinator or a fully-connected communications network is desired. It is also desired that this protocol have the capability of allowing consensus negotiation on two or more possible outcomes. To this end, this research applies metaheuristic techniques, often used for hard optimization problems, to the described problem of optimizing distributed consensus negotiation, because the distributed, iterative, population-based characteristics of metaheuristics share many similarities with the distributed negotiation process. Commonly-used metaheuristic techniques are augmented

with a negotiation protocol inspired by nature that provides insight into possible ways to overcome the problems with propagation and combination of preferences.

The scope of this dissertation is to define and describe a proposed nature-inspired metaheuristic for distributed, cooperative, self-organizing consensus negotiation. The relationships between the parameters of the protocol and the underlying social network are studied in order to understand their effects on a system's ability to converge to consensus under the influence of the metaheuristic. This understanding allows the considerations important to applying a more broadly-applicable agent-oriented design pattern, the quorum-sensing design pattern, to be characterized.

1.2. Quorum Sensing: A Natural Mechanism for Negotiation and Compromise

In the search for solutions involving complex systems, it is common to look to ways in which nature solves similar problems from which inspiration can be derived. Indeed, it is the case that many natural systems exhibit the ability to form decentralized, distributed, self-organized consensus (Larissa Conradt & Roper, 2005). In all instances, it is obvious that an essential requirement for a successful negotiation is that there must exist at least one option that all individuals are willing to accept as the final decision, otherwise consensus is impossible. The crux of the problem, then, is to define under what conditions an individual should change its preferred outcome in order to prevent deadlock due to intransigence. A common natural mechanism for determining this condition is quorum sensing. A quorum is a group of individuals with the same preference, the size of which is sufficient to exceed a predetermined threshold value that defines the point at which another individual, with a preference different than the quorum members', is willing to accept the quorum's preference as its own without further deliberation.

Quorum sensing has been shown to be used for self-organization by bacteria to regulate gene expression (Bassler, 1999), by ants (Pratt, Mallon, Sumpter, & Franks, 2002) and bees (Seeley & Visscher, 2003) to select new nest sites, and by fish to coordinate their shoal movements (Ward, Sumpter, Couzin, Hart, & Krause, 2008). These natural systems often use iteration, stochasticity, and feedback loops (e.g., local activation/long-range

inhibition) to solve the consensus problem, resulting in decentralized, emergent, self-organizing, and self-adaptive collective behavior that yields acceptable results through many individual applications of simple rules. Composed of mobile individuals situated in the chaos of uncertain natural environments, the negotiation protocols used by these collectives must tolerate changing neighborhood topologies, individual errors, and possible loss of individuals from the collective. This robustness under uncertainty is also a highly desirable trait for artificial applications of distributed consensus negotiation. It has also been shown that the quorum sensing pattern can be used to tune the speed-accuracy tradeoff through manipulation of the quorum size (Chittka, Skorupski, & Raine, 2009; Larissa Conradt & Roper, 2005; Passino & Seeley, 2006; Sumpter & Pratt, 2009).

Inspired by these natural applications, quorum sensing is increasingly being applied in distributed systems to guide self-organization. Examples include its application in sensor clustering (Wokoma et al., 2003) and server population management (Peysakhov & Regli, 2005). The prevalence of this technique in natural and artificial systems has led to its identification as an agent-oriented design pattern by Fernandez-Marquez et al. (2012). The metaheuristic presented in this research implements this design pattern using what is known about how honey bees negotiate consensus when selecting new nest site locations.

1.3. Overview of the Proposed Metaheuristic and Implementation Method

When a honey bee colony must select a new nest site, a swarm collects at an interim location near the old nest, and a subset of several hundred bees in the swarm (scouts) are responsible for searching for and evaluating candidate nest locations (Seeley & Buhrman, 1999). Initially, none of the scouts know any potential locations, so they are all uncommitted and start exploring. Some of the scouts will find candidate sites and become committed scouts. Others will be unsuccessful in finding viable sites and return to the swarm as uncommitted scouts. Committed scouts share their evaluations of the sites they found by performing dances that direct other scouts to the candidate locations. The intensity of a scout bee's dancing indicates the scout's perceived utility of the site it discovers (Seeley & Buhrman, 1999). In this way, uncommitted scouts are recruited to

confirm or reject the nominations and evaluations of the committed scouts. Uncommitted scouts are able to observe only the dances of the committed scouts within their sight. Given the choice between different dances, they choose a dance at random proportional to the number of observed dancers for each dance (Visscher & Camazine, 1999). After an independent evaluation of the site, the uncommitted bees return to the swarm and, if the site was found suitable, become committed scouts and dance for the site.

When a committed scout dances for a candidate site, the intensity of the dance decays (i.e., produces negative feedback) over time, even when the scout is dancing for a popular site (Seeley, 2003; Seeley & Buhrman, 1999). While this was previously thought to have been due to purely internal influences (Seeley & Buhrman, 1999), recent research has shown that bees produce a "stop signal" that causes other bees to stop dancing for a site (Seeley et al., 2012). This provides a mechanism for compromise and prevents stubbornness from causing deadlock. When a bee stops dancing for a site, it becomes an uncommitted scout; however, it is relatively uncommon for formerly committed scouts to dance for more than one site, and even more uncommon for them to dance for three or more sites (Seeley & Buhrman, 1999).

Committed scouts alternate between dancing for their chosen site at the swarm and visiting the nest site location for which they are dancing. Recent research has shown that once a certain number of bees (a quorum) are detected at a candidate nest site by a scout, the scout begins "piping" (producing a sound with its wings) at the swarm when it returns (Seeley & Visscher, 2003). This piping signals to the swarm that the end of deliberations is near and, as the piping increases as more bees join in the activity, eventually a level of piping is reached that triggers liftoff of the swarm for the selected nest site.

To translate this observed honey bee behavior into a generally-applicable algorithm for distributed consensus, it is useful to represent the behavior exhibited by individual bees during the negotiation process as a set of simple rules encapsulated by agents in an agent-based model. In agent-based modeling, agents are situated in a context that defines an environment, which agents sense and modify in accordance with their defined behaviors. In this particular application, the context is the underlying communication network that

connects the agents and the distribution of individual decision preferences at a given point in time. Peers to which an agent is directly connected in the network constitute that agent's neighbors. Agents sense and modify their environment by exchanging messages with their neighbors. The objective is to create a model in which the simple rules followed by the agents in response to neighbor interactions result in a context where the individual preferences have converged and stabilized at a single, uniform value. Due to this choice of representation, and for the sake of consistency, the terms "agent" or "individual" will be used throughout the rest of this dissertation to represent, in abstract terms, an individual member of a collective seeking consensus with the understanding that, in an actual application requiring consensus negotiation, an agent could actually be a process in software, a node in a distributed system, a single sensor, a vehicle, or any other autonomous entity.

The described agent-based model has been implemented using the Repast Simphony modeling and simulation framework (M. North et al., 2013). This has enabled simulation and study of the algorithm's behavior under different parameters and contexts and the formulation of conclusions about the factors that influence the successful formation of consensus.

1.4. Research Questions

In addition to describing and implementing a new metaheuristic algorithm for distributed, self-organizing consensus negotiation, this research tests a simulation model implementation of the algorithm to evaluate its performance and understand the fundamental conditions required for quorum-based consensus. This enables the identification of guidelines for the conditions under which quorum sensing is most appropriate and parameter values to use for the best performance in various applications. In consideration of these goals, this research addresses the following questions:

1. How can honey bee nest site selection behavior be implemented as a metaheuristic for reaching a decision that optimizes the social welfare of a group of agents?

2. How well does such a metaheuristic work, overall, for achieving socially-optimal consensus in commonly-studied social network models?
3. Does the quorum size parameter have a significant impact on the speed of distributed consensus formation and, if so, how does the choice of quorum size value affect the balance and tradeoffs between successful consensus formation, the speed of consensus, and the quality of consensus?
4. How does the social network model influence consensus dynamics?
5. What lessons can be generalized from this research for use in future applications of the agent-based quorum sensing pattern?

1.5. Contributions

The primary contribution made by this research is the specification of a novel metaheuristic for optimal, distributed, self-organizing consensus negotiation suitable for negotiation of consensus on multiple decision values. This metaheuristic is applicable to solving consensus problems in the distributed systems and self-organizing systems communities. As the Honey Bee Consensus metaheuristic uses the quorum sensing pattern, two important contributions in the agent-based patterns field are made regarding the quorum sensing pattern:

1. it is shown that a quorum size of 25% of the total population size is a near-optimal value to use for balancing the speed-accuracy tradeoff with Honey Bee Consensus
2. it is observed that Honey Bee Consensus performs similarly for quorums of intermediate size, regardless of the underlying social network model

1.6. Outline of the Dissertation

This dissertation is organized as follows. Chapter 2 provides the basis for the consensus problem, its variants, and relevant theoretical results. The state of the art in distributed consensus negotiation and current challenges are reviewed, exemplars of quorum sensing in nature and the current state of its codification as an agent-oriented

design pattern are described, and it is explained how existing metaheuristic optimization techniques contribute to the proposed metaheuristic. Chapter 3 presents the formal model for the agent-based implementation of the metaheuristic. Chapter 4 describes the evaluation methodology and experiment design. Chapter 5 presents the experiment results. Chapter 6 provides a discussion of the experiment results and their broader applications. Finally, Chapter 7 summarizes the research and proposes future work.

2. Literature Review

2.1. The Consensus Problem

2.1.1. Background

In its most basic form, the consensus problem is simply the problem of driving a system comprised of members preferring different outcomes, called decision values, to a state in which all system members agree on the same decision value. Thus, the correctness of result lies in reaching *any* valid agreement, rather than the comparison of the value agreed upon to an ideal value or in accordance with the majority opinion. A well-studied probabilistic stochastic process for opinion diffusion expected to reach unanimity in networks is the voter model (Clifford & Sudbury, 1973). In the voter model, an individual node *i* in the network is picked uniformly at random. Subsequently, one of this node's neighbors is selected uniformly at random and *i* sets its decision value in accord with the randomly selected neighbor's. This process continues until all nodes in the network share the same decision value.

The voter model's key weakness is its ambivalence about the final decision value obtained. More frequently, not only is unanimity desired, but there is a correct decision value that should be obtained. Traditional applications of consensus in computing include maintaining concurrency in distributed databases in which multiple copies of the database are kept in sync through agreement among the databases on which operations are committed (Thomas, 1979), clock synchronization, and sensor agreement (Pease, Shostak, & Lamport, 1980). In these types of applications, assuming the participation of only non-faulty processes (i.e., process that produce correct output), it is sufficient to consider a consensus algorithm correct if, at the end of the algorithm, all processes agree on the final value. Unfortunately, in most real-world cases it is unlikely that all participating processes and their communications links will be fault-free; therefore, there is a significant body of work exploring the limits of how much faultiness (and of what types) can exist in systems under various constraints and still admit consensus, e.g.: (Bracha & Toueg, 1983; Lamport, Shostak, & Pease, 1982; Pease et al., 1980). Even with correctly operating processes, distributed systems can comprise different states that need to be resolved. Some solutions

to these problems include ensuring that each process can communicate with a sufficient number of non-faulty processes (Barborak et al., 1993) and using a majority opinion as the decision value (Gifford, 1979; Thomas, 1979). Both of these are examples of using quorums to overcome faulty or diverse opinions in order to reach consensus on a decision value that is best for the system overall.

When the size threshold required to obtain a quorum is defined as a simple plurality, a new version of the consensus problem known as the majority consensus (or majority coordination) problem is obtained. In this problem, the correct decision value is that which is in accordance with the majority of the participants at the outset of negotiations. Mossel and Schoenebeck (2010) propose several models for solving this problem that do not use quorums; however, all of their proposed solutions are limited to achieving majority consensus on only two possible decision values. Furthermore, none of the solutions discussed thus far take into consideration preference weighting, or bias. The addition of this capability was one of the main contributions of the Networked Biased Voter Problem.

2.1.2. The Networked Biased Voter Problem

The version of the consensus problem known as the NBVP (Tan, 2010) is mentioned in the introduction. It deserves special attention here because it represents the version of the consensus problem most directly applicable to the solution proposed by this research.

In the NBVP, there are two competing choices. Each voter prefers each choice with a real-valued weight, which sum to 1. It is assumed that one choice is always collectively preferred to the other and that the preferred opinion is not known a priori to anyone. The objective is to determine this value through a distributed algorithm that is simple and local, and converges in time polynomial in the number of voters to the collectively preferred consensus (Kearns & Tan, 2008).

We revisit the NBVP in more detail later. Table 2.1 summarizes the differences between the methods described and the contributions of Honey Bee Consensus (HBC).

Table 2.1. Comparison of distributed consensus techniques.

Technique	Consensus Reached	Number of Decision Values Supported	Supports Weighted Preferences
Consensus Problem Techniques	Any unanimous	Many	No
Majority Consensus	Plurality	2	No
NBVP	Plurality	2	Yes
HBC	Highest social utility	Many	Yes

2.2. Distributed Artificial Intelligence and Agent Rationality

As described in the previous section, the standard consensus problem is only concerned with ensuring that all participants reach any valid consensus. In majority consensus or NBVP, however, different outcomes have different utility values when considered in relation to the desires of the collective as a whole, and the appropriate goal is to reach a consensus that matches the preference of the majority at the beginning of negotiations. When there are more than two possible outcomes, a majority consensus may be sought or, alternatively, one could seek a consensus that maximizes the collective utility. The latter situation can be further divided into two cases:

> Case 1: Each agent has the same utility function. All agents will view the consensus result equally favorably (or unfavorably) because they are all evaluating it with respect to the same metric. In this scenario, the agents are considered to be cooperating because they all seek the same goal (maximum collective utility) and they all agree on the utility values of the available choices due to using the same utility function.
>
> Case 2: Agents have different utility functions. Agents may disagree on the favorability of an outcome due to differences in their utility functions. In this scenario, agents may be in competition with respect to their individual preferences but still need to cooperate for the sake of reaching collective goals. This paradoxical situation has been called “co-opetition” (Nalebuff & Brandenburger, 1996).

A field in computer science that has focused on addressing these issues in agent-based systems is Distributed Artificial Intelligence (DAI). Historically, the first case described above was the focus of the distributed problem solving (DPS) research agenda, and the second case was the focus of Multi-Agent Systems (MAS) (Durfee & Rosenschein, 1994; Ephrati & Rosenschein, 1996). More recently, however, the term MAS has come to encompass all systems composed of multiple agents, whether cooperating, competing, or a mixture of the two (Wooldridge, 2009). Regardless of the underlying research agenda, though, it is not surprising that these scenarios in agent-based computing were preceded by their human-interactive counterparts in social science, economics, and game theory. These fields have contributed to the underlying assumption that artificial agents should behave rationally and to methods for defining and evaluating rationality (Durfee & Rosenschein, 1994). One particular difficulty in measuring rationality in order to judge actions or choices is that the "correct" or rational action to take may be different depending upon whether the agent has global or partial knowledge of the factors relevant to the decision. Economic rationality provides a way to evaluate decisions made in the presence of uncertainty and heuristics used in those decisions (Doyle, 1992).

Doyle (1992) summarizes concepts from economic rationality as they apply to artificially intelligent reasoning to evaluate the "goodness" of actions, used by both decision-theoretic and game-theoretic approaches, and provides references to the economic literature on which artificially intelligent rationality is based. Decision-theoretic approaches to problems in artificial intelligence are described in (Feldman & Sproull, 1977; Gmytrasiewicz, Durfee, & Wehe, 1991; Horvitz et al., 1988; Jacobs & Kiefer, 1973), and a game-theoretic approach to consensus is presented in (Wang et al., 2013). These approaches give various examples of how utility can be incorporated into an agent's reasoning process.

2.3. Social Choice Theory and Voting

Now the application of rational agents to the problem of determining socially-optimal consensus is considered. Since each rational agent will seek to maximize its

individual utility, reaching consensus in Case 1 from the previous section is trivial. In Case 2, however, the differences of opinion between agents as to which outcome is the best and the need to choose only one of several possible outcomes, as dictated by the consensus problem, necessarily introduces a conflict in which some agents must accept sub-optimal outcomes from their perspectives. Nevertheless, due to the paradox of co-opetition, it remains rational for some agents to acquiesce to the preferences of other agents since we are operating under the assumption that the utility for any non-consensual outcome is zero for all agents. The critical question is: which agents should change their preferences, and which alternative outcome should they prefer?

To answer this question, it must first be understood how the optimality of a final group decision can be defined. Social welfare theory distinguishes between assigning an ordering over the set of possible alternatives based on individual preferences, obtained through the application of a social welfare function (SWF), and settling on one of the possible alternatives, obtained by the application of a social choice function (SCF) (Ephrati & Rosenschein, 1996). In other words, SWFs aggregate individual preferences, and SCFs are formal representations of voting procedures. Shoham et al. provide formal definitions of these functions. (Shoham & Leyton-Brown, 2009) Taylor (2008) and, especially, Brams et al. (2002) give detailed descriptions of a wide range of voting procedures that have been developed within this field of study. A key observation here is that optimality is subject to the SWF and/or SCF applied; therefore, it is important to acknowledge that there is no perfect voting procedure. Indeed, it has been proven that there is no SWF that satisfies all reasonably desirable properties that SWFs should hold (Arrow, 1950), therefore SCFs must be chosen with consideration given to the SWF properties they can ensure. Nevertheless, it is reasonable to conclude that a potential answer to the critical question, above, is that a "good" consensus protocol should result in all rational agents aligning their preferences with the maximum preference from a chosen SWF or that matches the outcome that would result from applying a chosen SCF to all of the agents' preferences. To this end, the trivial solution is to either consolidate all of the agent preferences and apply the chosen SWF or hold an election; however, this introduces the problem of centralized control in the election

coordinator, resulting in a single point of potential failure. These methods also require that all agents publicly disclose their preferences, which may not be desirable in all cases (Heiskanen, 1999). Finally, depending on the voting procedure chosen, strongly opinionated minorities or poorly-informed majorities could unduly influence the results of the election, possibly leading to a sub-optimal final choice, and this, too, must be considered.

2.4. Automated Negotiation

In game theory terminology, voting procedures are a "one-shot game." That is, the agents make their choices, the outcome is evaluated, and the election result is binding. Since the election is not repeated, agents cannot use their observations of the other agents' past actions to improve their performance in future rounds. While there are some voting procedures that use iterative removal of candidate outcomes, all voters provide their complete preferences in advance of the election, so voters cannot benefit from new knowledge part-way through the election (Brams & Fishburn, 2002). As was seen in the previous section, one of the weaknesses of elections is that strongly-opinionated minorities or poorly-informed majorities can unduly influence elections and, in a one-shot scenario, the agents are committed to these results, even if they are sub-optimal. It may be more desirable to allow agents to influence each other's opinions by iteratively sharing information and coming to a more gradual consensus. This can be achieved with automated negotiation.

Jennings et al. (2001) provide a thorough introduction to automated negotiation issues and discuss game theoretic impacts on negotiation strategies. They characterize negotiation as an iterative process, requiring the minimal capabilities of proposal of a solution and response (accept or reject) (Jennings et al., 2001). While their paper focuses on game theoretic and heuristic aspects of negotiation, many actual implementations of negotiation protocols make extensive use of the aforementioned social utility foundations to enable agents to reason rationally about the values of deals and compromises. Chevaleyre et al. (2006) and (Endriss, Maudet, Sadri, & Toni, 2006) present the application

of social welfare theories to the multi-agent resource allocation problem, but many of the issues affecting that problem are the same as those in distributed consensus in that both problems seek to negotiate a socially-optimal solution in a distributed environment.

A common weakness of many negotiation protocols proposed thus far are that they focus on combinatorial auctions (Endriss et al., 2006), which requires a fully-connected communication network and becomes increasingly infeasible as the number of agents increases; they only allow for pairwise negotiation (Ehtamo, Verkama, & Hamalainen, 1999; R. G. Smith, 1980); or they require a trusted mediator (Ehtamo et al., 1999; Heiskanen, 1999; Heiskanen, Ehtamo, & Hämäläinen, 2001; Li, Vo, & Kowalczyk, 2009; Lopez-Carmona, Marsa-Maestre, & Klein, 2011; Vo, Padgham, & Cavedon, 2007; Ziang & Mo-Yuen, 2011; Ziang et al., 2011), which can be both prohibitive in uncontrolled environments and, again, introduces a potential single point of failure. Finally, many negotiation techniques require some sort of capability to transfer utility between agents which is typically thought of in terms of payments between agents. This may be used as a mechanism to encourage fair behavior as, for example, in the use of the Clarke tax mechanism in (Ephrati & Rosenschein, 1996), or simply as a mechanism for persuasion, which enables more efficient negotiation over protocols that simply allow agents to accept or reject proposals (Jennings et al., 2001).

In addition to these weaknesses, very little work has been done on automated negotiation of complex contracts, focusing instead on negotiation of contracts involving only one or a small number of independent issues. As a sample of representative work that has been done, Klein et al. (2003) propose "the first negotiation protocol specifically for complex contracts." They present two versions of their solution, one which requires a mediator, and one that does not. Ito et al. (2008) present a mediated multi-issue negotiation protocol. Both of these approaches use simulated annealing as a metaheuristic for finding Pareto optimal solutions to complex contracts.

2.5 Self-Organization

Unfortunately, automated negotiation in the sense of iterated proposals versus counterproposals is not going to be effective in forming consensus in large, decentralized,

autonomous systems. It has been identified that using voting procedures to make a collective choice is too centralized, requiring consolidation of global information in order to reach a decision. Automated negotiation is, in a sense, too decentralized—the number of pairwise agreements between agents that must be negotiated (and re-negotiated as other, new negotiations are made) quickly becomes infeasible. Additionally, without a mechanism for the transfer of utility, the system would require some awareness of the global utility of each option, which is exactly the unknown being sought. Instead of attempting to purposefully design a system to reach consensus by directed interactions between all agents, a system that arrives naturally at the consensus decision by following simple rules that govern each agent's interactions with only its local neighbors is considered. Recall from section 2.1.2 that one of the criteria of a NBVP solution is that it should be "simple and local" (Kearns & Tan, 2008). Another criterion imposed by the NBVP, as formulated by Kearns et al. (2008), is that agents update only their own preferences based on observation of their neighbors' choices; they should not attempt to encode detailed information or send "signals" to neighbors (Kearns & Tan, 2008).

These goals and criteria strongly suggest the incorporation of self-organization principles. Parunak et al. (2011) define a self-organizing system as a "special kind of self-adaptive system" characterized, primarily, by agents that change their interrelations. In particular, self-organizing systems should not require centralized management to reach their goals or any explicit representation of system goals or architecture (Parunak & Brueckner, 2011). Thus, agents that decrease the entropy of the system, in this case by organizing the alignment of agents to the socially-optimal consensus, self-organize. Their rule set governing purely local interactions does not require any knowledge of the system goal, per se.

In searching for self-organizing solutions, it is common to look to nature for inspiration and understanding of the underlying organizational principles of self-organization (Couzin & Krause, 2003). Knoester et al. (2013) and Knoester & McKinley (2009) use digital evolution to evolve an algorithm for forming consensus in a group; however, this research adapts a consensus protocol already found in nature to the NBVP.

2.6. Natural Consensus, Quorum Sensing, and Agent-Oriented Design Patterns

2.6.1. Consensus Formation in Nature

How groups of animals make decisions has been an interest of biologists for some time, although, even until recently, little has been known about the processes animals use (L. Conradt & Roper, 2003). Conradt et al. (2003) modeled and studied the differences between despotic and democratic decision making in animal groups in outcome fitness. They determined that the synchronization costs for democratic decision making were usually lower than those for despotic control (L. Conradt & Roper, 2003). Couzin et al. (2005) introduce a simple model for direction of travel consensus in animal groups in which the group settled on average preferences when the differences in preferences between individuals were small, but achieved consensus for the majority option in the presence of large differences. They also showed the impact of a weighted feedback mechanism to allow consensus, rather than averaging, in cases with small differences (Couzin et al., 2005). To categorize the many decision making techniques in use, (Larissa Conradt & Roper, 2005) present a classification scheme for animal decision making techniques and argue that better understanding of how animals achieve consensus could "yield insights into the evolution of cooperation, communication and group decision making in humans."

While Conradt et al. (2005) focus on the properties of consensus formation in many different species, other researchers have worked on determining the consensus mechanisms used by specific species. Of particular inspiration to this proposed research is the regulation of gene expression in bacteria (Bassler, 1999), transfer of information in fish shoals (Ward et al., 2008), ant nest site selection (Nigel R. Franks, Mallon, Bray, Hamilton, & Mischler, 2003), and honey bee nest site selection (Niven, 2012; Seeley, 2003; Seeley & Buhrman, 1999; Seeley & Visscher, 2003; Seeley et al., 2012; Visscher & Camazine, 1999).

In these studies, many factors are suggested as contributing to a system's ability to reach consensus. In (Kearns, Judd, Tan, & Wortman, 2009; Kearns & Tan, 2008; Seeley, 2003), the authors cite the importance of undecided voters in reaching consensus, and this

is further supported by (Couzin et al., 2011). One commonality among these techniques that stands out, however, is their use of quorum sensing.

2.6.2. Quorum Sensing

In (Baarslag & Jonker, 2011), the authors discuss in detail the conditions under which negotiated conditions should be accepted, but these are all based on endogenous factors. Quorum sensing is an exogenous acceptance condition requiring an individual to transition to one of its less-preferred choices when a threshold number of neighbors are aligned with that choice. (Sumpter & Pratt, 2009) examine quorum responses in detail and argue its central importance in decision making; its use has several benefits. In contrast to (Jennings et al., 2001), which advocates the use of a critique (counterproposal) capability, quorum sensing allows consensus without this mechanism. It also allows a system to make a trade-off between time to reach consensus and accuracy of consensus by tuning the quorum threshold (Chittka et al., 2009; Larissa Conradt & Roper, 2005; Passino & Seeley, 2006; Sumpter & Pratt, 2009).

2.6.3. Agent-Oriented Design Patterns

The identification, categorization, and formalization of recurring patterns in software design was pioneered by (Gamma, Helm, Johnson, & Vlissides, 1994) with the intent of facilitating the reuse of best practices in software construction and creating a common language for use in discussion of object oriented software designs. Since then, the use of design patterns has spread to other, more specific, application areas. Within agent-based modeling, simulation, and engineering, alone, there have been multiple areas of focus including general applications of patterns (Aridor & Lange, 1998; Ferreira de Araújo Lima, Duarte de Lima Machado, Abrantes de Figueiredo, & Sampaio, 2003; Kendall, Krishna, Pathak, & Suresh, 1998), self-organization and coordination (Tom De Wolf & Holvoet, 2006; Tom De Wolf & Holvoet, 2007; Deugo, Weiss, & Kendall, 2001; Gardelli, Viroli, & Omicini, 2007; Gatti, de Lucena, & Garcia, 2009; T. Holvoet, Weyns, & Valckenaers, 2009; Tom Holvoet, Weyns, & Valckenaers, 2010; Kasinger, Bauer, &

Denzinger, 2009; Snyder, Valetta, Fernandez-Marquez, & Serugendo, 2012), and bio-inspired patterns (Babaoglu et al., 2006; Fernandez-Marquez et al., 2012). North et al. (2011) make the key distinction between agent-based and agent-oriented design patterns. The former is focused on the software design patterns used to implement agent-based models, whereas the latter is focused on the patterns of agent behaviors and interactions.

Quorum sensing, as described above, has been used in multi-agent applications including sensor networks (Britton & Sacks, 2004; Wokoma et al., 2003) and server population management (Peysakhov & Regli, 2005). These applications and quorum sensing's frequent recurrence in nature have led to the identification of quorum sensing as an agent-oriented design pattern by (Fernandez-Marquez et al., 2012); however, these authors were not able to describe the implementation of the pattern in a generalized manner, stating instead that "there is no specific implementation for the Quorum Sensing Pattern" (Fernandez-Marquez et al., 2012). Not only does this feel unsatisfactory in the sense of being able to use the pattern in a modular way to solve problems, it also results in an inability to identify the considerations that should be used in the application of the pattern, as is done for patterns in the style of (Gamma et al., 1994). Therefore, although there is significant agreement in the prevalence of quorum sensing and how it works in general, there is still significant work to be done in understanding the dynamics of quorum sensing systems, especially if it is intended to use quorum sensing as a generalizable pattern for engineering self-organizing consensus.

2.7. Metaheuristic Influences and Eusocial Insect Models

In addition to using the quorum sensing pattern as a basis for a self-organized solution to the problem of forming distributed consensus, the solution proposed in this research also leverages ideas from metaheuristic optimization techniques and lessons learned from existing models of eusocial insects. Negotiation and consensus formation both have similarities to hard optimization, a major application area for metaheuristics, and eusocial insect models provide insight into how nature has evolved the values for parameters related to quorum-based consensus.

2.7.1. Metaheuristics

Consensus formation can be thought of as a search for the combination of choice and preference pairs for a set of agents that maximizes the utility of the collective, when all preferences for a single choice are considered together. According to (Jennings et al., 2001), "negotiation can be viewed as a distributed search through a space of potential agreements" with the goal of optimizing the utility of the final agreement. It is highly likely that these search spaces are non-linear such that greedy hill-climbing algorithms searching for the optimal configuration are susceptible to stalling at local maxima. These sorts of non-linear search applications are exactly the types of situations in which metaheuristics excel at finding solutions, so long as there is some way to compare the "goodness" of possible solutions (e.g., using their utilities) and there is a heuristic for how to modify a possible solution in such a way that it is likely to be closer to the final solution. Thus, metaheuristics iteratively sample the solution space, using heuristics and stochasticity to incrementally adjust the search to random, but hopefully better, locations in the search space and updating the solution with better solutions as they are found. There is usually a parameter that balances exploration and exploitation such that during the exploration phase bigger moves through the search space are allowed, and during exploitation searches are constrained to remain close to the best solutions found so far. Since these techniques are randomly sampling the search space and do not perform an exhaustive search they are not guaranteed to find the optimal solution. Instead, they are techniques that usually give a good solution, that is, they are heuristics that use heuristics in their processing, giving rise to the term "metaheuristic" (Dréo, Pétrowski, Siarry, & Taillard, 2006).

The iterative nature of the honey bee consensus protocol from section 1.3, the presence of utility values, and the ability to heuristically monitor the global formation of consensus through local observations suggest that the Honey Bee Consensus protocol would be a suitable candidate for a metaheuristic implementation. The stochasticity of natural systems is believed to be an important aspect to self-organization patterns as well (Couzin & Krause, 2003), and highly compatible with metaheuristic techniques. For

example, simulated annealing (Kirkpatrick, Jr., & Vecchi, 1983) has been applied to contract negotiation in (Ito et al., 2008; Klein et al., 2003). In addition to simulated annealing techniques, striking similarities are also seen between natural honey bee consensus and particle swarm optimization (Kennedy & Eberhart, 1995).

2.7.2. Eusocial Insect Models

Several models of eusocial insect colonies have been developed for ants (Pratt, Sumpter, Mallon, & Franks, 2005) and bees (List, Elsholtz, & Seeley, 2009; Passino & Seeley, 2006). These models are not directly applicable to general consensus formation because they seek to reproduce ant and bee behavior that involves actual insect movement in a spatial context. Nevertheless, choices made in these models regarding interaction protocols and parameter value choices are valuable in providing a general idea of where to start with metaheuristic variables and values. In the next chapter, the model proposed in this research is formalized and described in detail where it differs from natural systems.

2.8. Chapter Summary

The previous sections of this chapter situates this research precisely in the literature. It is desired to develop a metaheuristic that solves a modified form of the NBVP that, unlike the canonical NBVP, allows for tied preference weights among outcomes and more than two possible outcomes. The problem is further relaxed to allow a limited amount of information encoding and signal passing, specifically, information related to quorum size and membership.

Significant use is made of the concepts of preference, utility, and tools from decision theory as described in (Doyle, 1992) in order to evaluate the goodness of the consensus reached by the Honey Bee Consensus protocol. Unlike traditional DPS solutions, the presented technique does not require that each agent use the same utility function in its computation.

The implementation and analysis of the model allows inferences to be drawn about relationships between parameters related to quorum sensing and network topologies

influencing consensus. This will have applications in behavioral sciences and network sciences. It will also contribute to the agent-oriented pattern literature by providing a generalized quorum sensing pattern and associated considerations for its application. The next chapter provides the formal model upon which the implementation will be based.

3. Formal Model of Honey Bee Consensus

In this section, following the Overview-Design-Details (ODD) model specification protocol (Grimm et al., 2006), the adaptation of agent-based honey bee models to a general-purpose metaheuristic called Honey Bee Consensus (HBC) and the rationale for corresponding design decisions is described in detail. In this description, the model actors in HBC are referred to as "agents" and the term "bees" is used when the agent-based model origins that inform the design decisions are referred to. Much of the content of this chapter is an updated version of work previously published in (Mentis & Yilmaz, 2013).

3.1. Purpose

The purpose of the model is to create a multi-agent system of voting agents that can solve the NBVP, modified to accommodate more than two decision values, in a manner similar to the way in which honey bees form consensus on a new nest site location.

3.2. State Variables and Scales

The model consists of three hierarchical levels: agent, population, and social network. Agents encapsulate an identifier; a set of neighboring agents in the social network; a set of preferences for the possible outcomes; a currently-preferred decision value (choice); a commitment duration remaining for the currently-preferred outcome; a commitment decay (evaporation) rate; an evaporation threshold, below which the commitment duration triggers an agent to become uncommitted to any choice; and a quorum size threshold that is used by uncommitted agents to determine a new choice to which to commit. Agents also contain a *QuorumData* object that contains their current state of belief about the global consensus progress as learned from their local neighbors.

The population provides the global aggregation of individual preference data for the purposes of calculating the preferred outcome if all agents had perfect, global information about the preferences of their peers. The population also plays a role in determining the global distribution of preferences during model initialization, as discussed in section 3.5.

The social network is defined by the set of edges created by each agent's *neighbors* set. In other words, there is an edge in the social network between agents *A* and *B* if agent *B* is a member of the set of agents in *A.neighbors* or vice-versa. In the model, the network is an undirected graph so, for this example, agent *A* would be in *B.neighbors* and agent *B* would be in *A.neighbors*. The network structure will be maintained in a data structure provided by the simulation framework.

These state variables and their relationships are depicted in the UML diagram of Figure 3.1.

3.3. Process Overview and Scheduling

The model proceeds in simulation ticks. A simulation tick does not represent any real-world time, it is simply a synchronization barrier for agent activation. Agents are activated in an asynchronous random manner, and agent actions are broken into two phases at each simulation tick: a neighbor polling phase and a decay phase. At the beginning of every simulation tick, an agent is committed to one of the possible decision values.

In the neighbor polling phase, an agent asks for the quorum data from all of its neighbors, but it only incorporates information from neighbors that it agrees with. This is a positive feedback mechanism similar to a honey bee seeing other bees when it visits the site it prefers, as described in section 3.7.4.1 for committed agents.

In the second phase, the decay phase, the agents apply the decay function to their remaining commitment duration as an application of evaporation from section 3.7.4.3. This is the system's negative feedback mechanism replicating a bee's eventual loss of interest for a site over time. When the result of this evaporation falls below the evaporation threshold and results in an agent becoming undecided, the agent follows the aggregation rules for neutral agents, as described in section 3.7.4.1. This triggers a re-polling of all neighbors, but unlike in the first phase, the newly uncommitted agent takes into consideration all of its neighbors quorums. Which neighbors the agent aligns its choices with depend on if there is a unique quorum among its neighbors, in which case it aligns

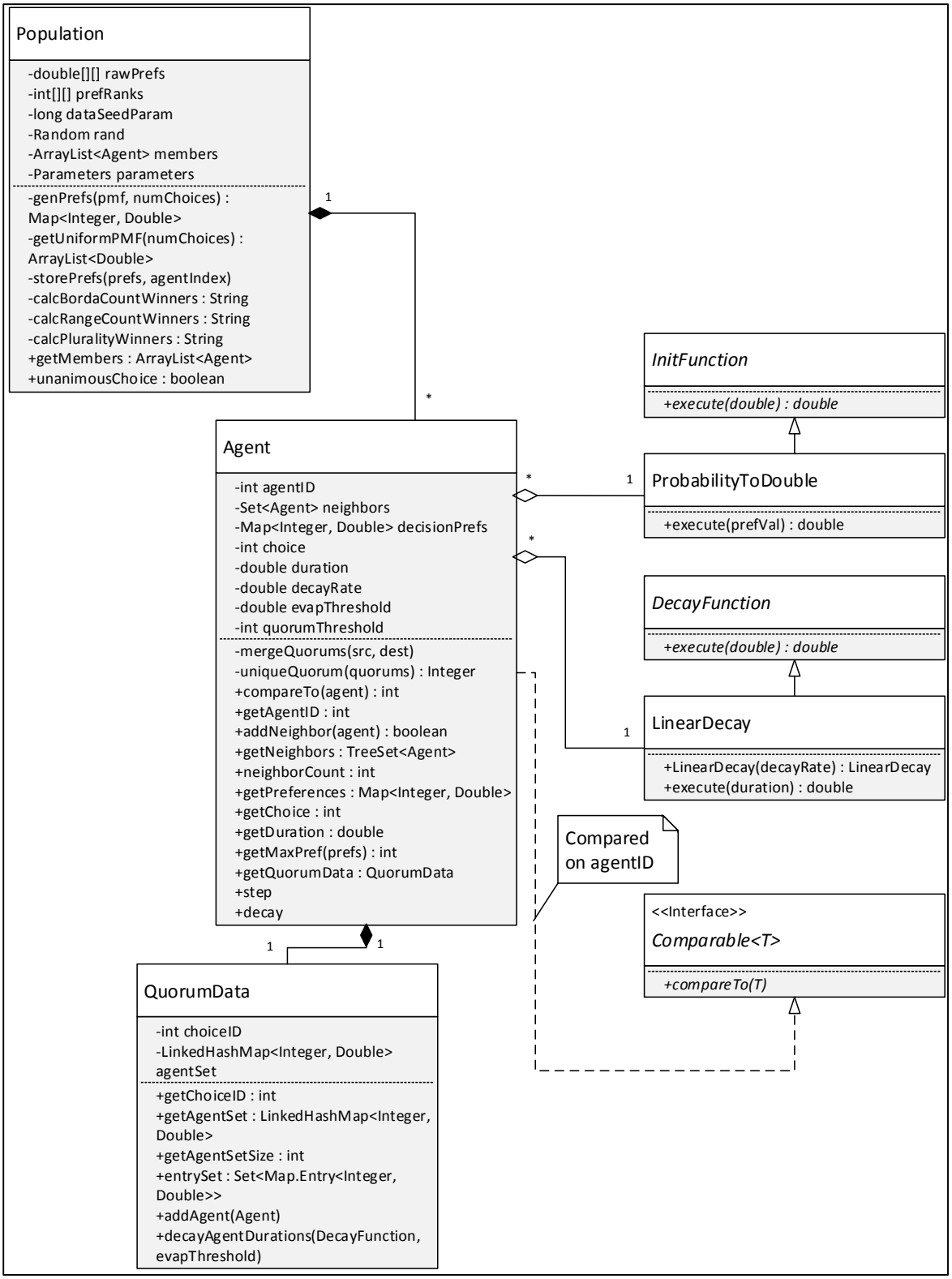


Figure 3.1. UML class diagram of implementation.

with the choice of that quorum, or else it is probabilistic based on fitness proportionate selection. In either case, the agent is guaranteed to become committed to one of the possible outcomes, maintaining the loop invariant for the beginning of every simulation tick. To clarify these interactions, UML activity diagrams and pseudocode for them are provided in the following sections. Further details of the submodels are provided in section 3.7.4.

3.3.1. UML Activity Diagrams

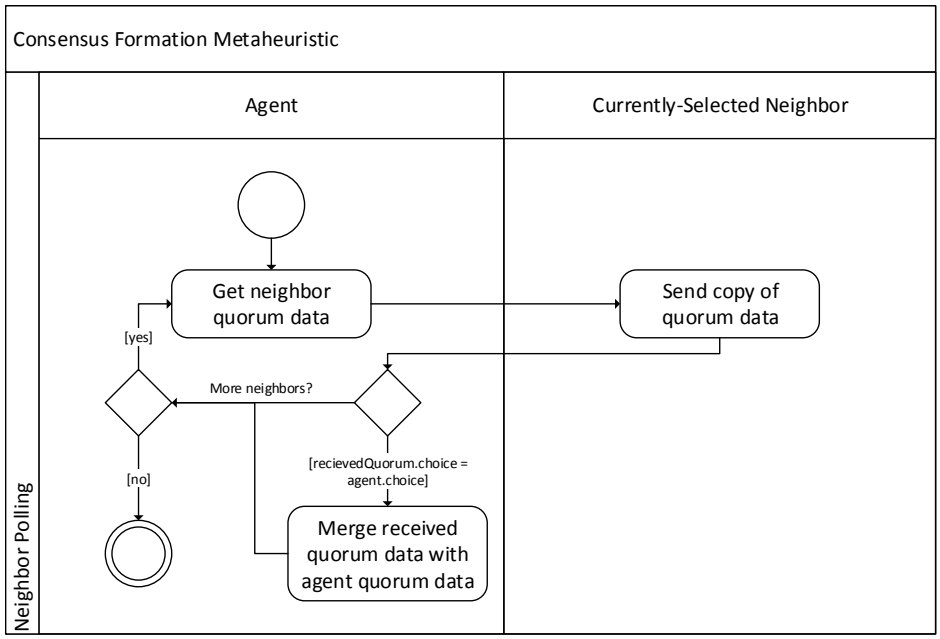


Figure 3.2. UML activity diagram for neighbor polling (positive feedback) phase.

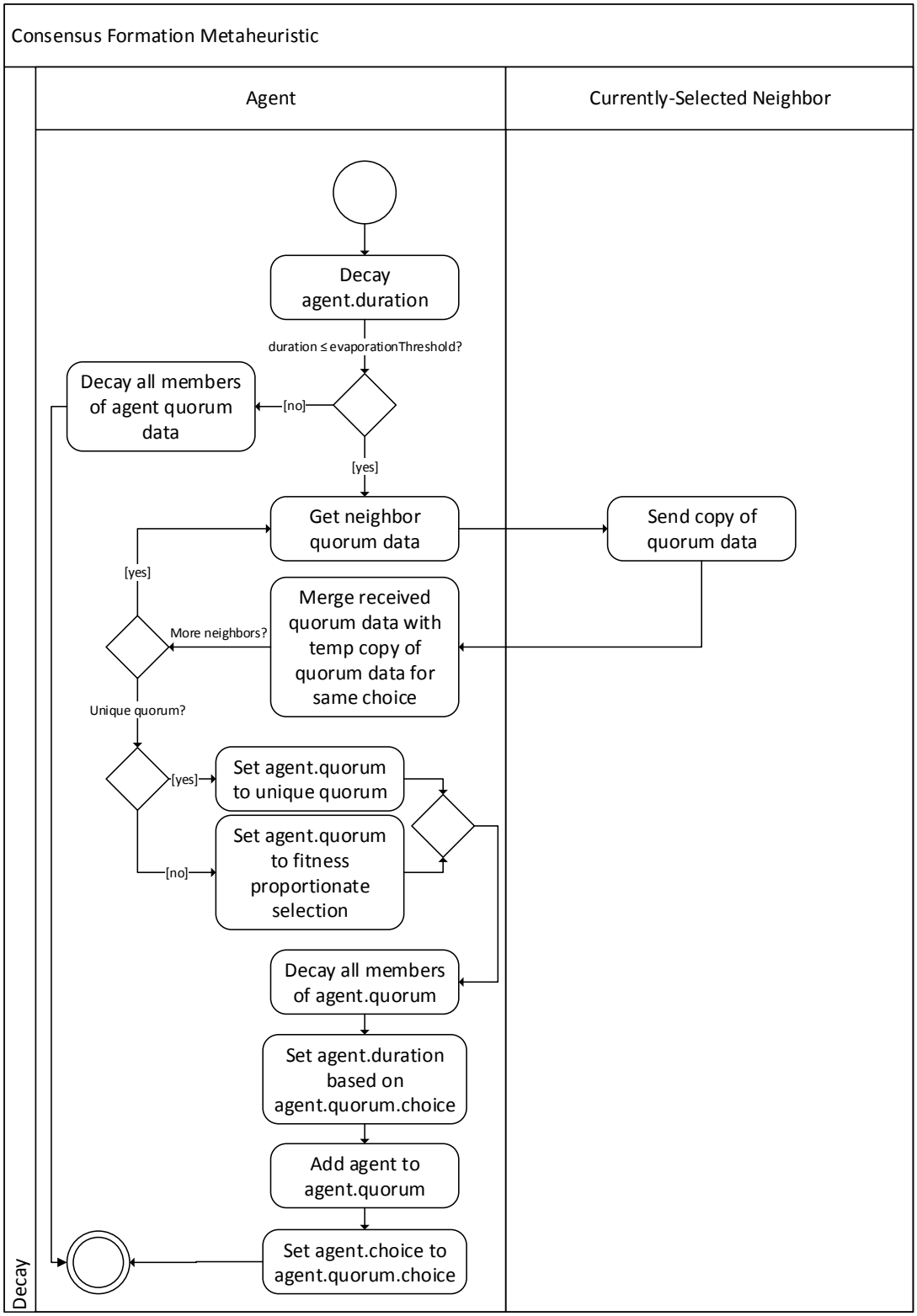


Figure 3.3. UML activity diagram for decay/evaporation (negative feedback) phase.

3.3.2. Pseudocode

Pseudocode for the agent actions at each simulation tick is as follows:

```
while tick ≠ maxIterations:
   for each agent a: // update with positive feedback
      for each neighbor n in a.neighbors:
         receivedQuorum = n.copyQuorumData()
         if receivedQuorum.choice = a.choice:
            merge receivedQuorum and a.quorum as follows:
               perform the set union of their agent sets;
                  if an agent is in more than one Q.A, keep the
                  largest d value for that agent

   for each agent a: // apply decay function (negative feedback)
      apply f_decay to a.duration
      if a.duration ≤ evaporationThreshold: // become undecided
         collect quorum objects from all neighbors, merging quorum
            objects sharing the same choice, as above
         if a unique merged quorum object meets the quorum threshold:
            a.quorum = unique merged quorum
         else:
            a.quorum = result of fitness proportionate selection
         apply f_decay to all members of a.quorum
         a.duration = f_init(a.p_a.quorum.choice)
         a.quorum.add(a)
         a.choice = a.quorum.choice
      else:
         apply f_decay to all members of a.quorum
tick++
```

3.4. Design Concepts

3.4.1. Emergence

Consensus on one of the possible outcomes emerges as the agents modify their commitments based on interactions with their neighbors and evaluation of propagated information. Individual response to local information is entirely represented by probabilistic rules. Fitness-seeking is therefore not modeled explicitly, since an individual cannot accurately evaluate the fitness of its own choice without global knowledge. Instead, the probabilistic responses to local interactions drive the agent toward the most-fit choice.

3.4.2. Sensing

Individuals are assumed to know their own preferences for each of the possible outcomes, to which outcomes they are currently committed, and which agents are their neighbors.

3.4.3. Interaction

Explicit interaction consists of polling neighbors and receiving the quorum data in response to the polls. Persuasion of neighbors is modeled implicitly by an agent refusing to incorporate the preferences of a neighbor until the agent becomes uncommitted. The agent whose commitment decays first, due to lower preference, becomes subject to the persuasion of its neighbors whose commitments have not yet decayed below the evaporation threshold.

3.4.4. Stochasticity

As is typical in metaheuristic algorithms, stochastic behavior is an integral part of this solution. It is used to determine the order of agent activation; the order in which the decay function is applied to agents and their quorum data; and the uncommitted agent choice alignment in the absence of a unique quorum, as described in section 3.7.4.1. All of these random values are drawn from the uniform distribution; however, uncommitted agent

preference alignment applies the random value probabilistically to perform a fitness proportionate selection from among neighbor preferences.

3.4.5. Observation

For model analysis, each run's model parameter settings, each agent's preference weights for the possible outcomes, and the population-level aggregation of preferences in accordance with the reference voting protocols described in Chapter 4 are recorded. At the end of the run, the number of simulation ticks executed, whether or not a consensus was reached, and the consensus value agreed upon (if applicable) are also recorded.

3.5. Initialization

The pseudorandom generation of agent preferences is performed in two stages. First, a target probability distribution (p_{target}) for the outcomes is generated from the uniform distribution $\mathcal{U}(0, 1)$. This determines what probability each outcome has of being the preferred choice of an agent. For example, a p_{target} of ⟨0.16, 0.04, 0.09, 0.68, 0.03⟩ would indicate that approximately 68% of the agents should assign Outcome 4 their highest preference, whereas approximately 3% should assign Outcome 5 their highest preference.

In general, n random values in the interval (0, 1) that sum to 1 are desired, where n is the number of possible outcomes. As Smith & Tromble show in (2004), however, it is not possible to just choose n values and normalize them by dividing by their sum, since that introduces bias. Instead, following (N. A. Smith & Tromble, 2004), probability distribution are constructed as follows:

Sample $x_1, \dots, x_{n-1}$ uniformly at random from $\{1, \dots, 100\}$ without replacement. Let $x_0 = 0$ and $x_n = 100$. Define p^i_{target} be the ith value in the target probability distribution. Then $p^i_{target} = \frac{x_i - x_{i-1}}{100}, \forall i \in \{1, 2, \dots, n\}$. Division by 100 transforms the integer preferences into percentages. If desired, the precision of the preferences could be increased by using a larger value, e.g. 1,000 to get three decimal places of precision. This algorithm has the limitation of not allowing any index of the distribution function to equal 0.00, but in the target problem space, this is exactly what is necessary. If any of the

outcomes were to have 0% chance of being preferred by an agent, then it should not be under consideration by the group.

Second, having obtained the target probability distribution, a roulette wheel selection based on p_{target} is used to assign each agent its preferences for each of the n outcomes. Using the same method as the first step, a new pseudorandom probability distribution is created representing all of the possible preferences an agent could have for an outcome. These values represent the weight of an agent's preference for each outcome versus the others, so they sum to 1, since the sum of these values captures all possible outcome preferences. Again, zero values are not allowed since that would be indicative of a stubborn agent that refused to consider one of the outcomes, and that could lead to a failure in consensus. These preference weights are sorted and iterated through from largest to smallest, assigning them, in turn, to the outcome returned by the roulette wheel selection. Roulette wheel selection will return unassigned outcomes proportionate to their probabilities in p_{target}, so it is most probable that the highest preference weights will be assigned to the outcomes that are supposed to be the most popular.

This methodology allows the pseudorandom generation of preferences across all agents that result in collective preference sets that can yield winners, losers, and ties, depending on the social rules by which they are evaluated. It also allows that multiple agents can prefer the same outcome despite different preference weights for that outcome. Were the p_{target} distribution for preference assignment not generated, the uniformly random agent preferences would be distributed so uniformly that each of the options would receive approximately equal total utility. Essentially, all choices would be equally good and thus the efficacy of the algorithm for finding optimal solutions would not be evaluable.

3.6. Input

3.6.1. Social Network Topology

There are several classes of graphs that have characteristics of particular interest in the study of social networks. The NetworkX Python package (Hagberg, Schult, & Swart, 2008) is used to generate random graphs from several classic graph classes, such as Erdős-

Rényi graphs and graphs that exhibit small world and power law characteristics. For this research, it is the intent to test and compare the algorithm's performance on each of these types of graphs in order to derive generalizations about consensus formation in networks that have these characteristics. It is expected that this information will be of interest to researchers who focus on problems involving these topology classes. The social network models studied are explained in detail in Chapter 4.

3.6.2. Quorum Size

The central mechanism of HBC is quorum-sensing; therefore, the impact of the quorum size parameter on algorithm performance is of particular interest in this research, both for developing guidelines for this algorithm's use in consensus formation problems as well as for general applicability to solutions incorporating the quorum-sensing pattern. For the classes of graphs just mentioned, it is of interest to determine if quorum size affects successful consensus formation and, if so, how the quorum size should be adjusted to successfully achieve consensus.

3.7. Submodels

3.7.1. Model Parameters

The parameters listed in Table 3.1 have been incorporated into the model.

Table 3.1. Model parameters.

Parameter	Description	Impact
Agent Count	Number of agents that must reach consensus.	Higher numbers are expected to make consensus harder to reach.
Quorum Size Threshold	The number of agents that must agree in order to reach a quorum.	Once a quorum is formed, it has the potential to force other agents to align themselves with its choice if it is the only quorum, otherwise, alignment is fitness proportionate.
Decay Rate	A value by which agent commitment is decreased at each round. Can be modified to select a decay function, e.g., one that produces a geometric decay rate, if desired.	Agent preference must decay to allow agents to become uncommitted and open to compromise, either by being influenced by their neighbors or, if a unique quorum has formed, by committing to the quorum's choice.
Evaporation Threshold	The value below which an agent's preference indicates non-commitment.	When an agent's decay rate has lowered its commitment below this value, the agent becomes uncommitted and open to persuasion to different outcomes.
Network Topology Type	Allows the use of a various social network models.	This allows the use of different network topologies created externally to the model.
Random Seed	The seed for the PRNG controlling agent activation order and stochastic actions.	Randomizes initial states and behavior on multiple runs.

3.7.2. Metaheuristic Foundation

Fundamentally, distributed decision making is viewed as an optimization problem in which the aim is to decide on a value that maximizes the collective utility of all the agents in the group. In the case of honey bees, they are seeking the nest site that is perceived by the collective to be the best of several possible choices.

As described previously, honey bees arrive at their nest site decision through a process that comprises a mix of independent site evaluation and influence from peer opinions. Several features of the Particle Swarm Optimization (PSO) metaheuristic (Kennedy & Eberhart, 1995) are recognized that are similar to the honey bee decision making process or that provide a solution to some of the problems that must be solved by HBC. Although the PSO metaheuristic is designed for optimizing continuous non-linear functions, which is different from the consensus problem at hand, PSO's use of particle interaction within a neighborhood of peers in a connected network; stochasticity, and social (external) and cognitive (internal) influences; and iteration-based termination conditions are useful for consensus purposes.

In PSO each particle has a current location in the solution space (analogous to a decision in the consensus problem) and it moves to new locations (decisions) by combining information it knows about its preferences with those of its neighbors. Similarly, uncommitted bees obtain information from their dancing neighbors that push the uncommitted bees toward a decision. This implies that HBC must define a neighborhood of peers for each agent and provide a way to share preferences. The original PSO description called for all of the particles to be fully interconnected in a mesh topology, and it is also common in the PSO literature for the particles to be connected in a ring topology; however, neither of these topologies is particularly realistic for bee colonies or the potential application areas. Instead, the idea of connecting agents to a neighborhood of peers is retained, but the network topology is made to be a configurable parameter and allows more natural topologies such as random, scale-free, and hierarchical network topologies. As in PSO, however, at each iteration of HBC, each agent polls all of its neighbors for influence.

The influence of stochasticity, and social (external) and cognitive (internal) factors drives how each particle in PSO responds to the influences it receives from its neighbors. Similar forces are used, explained in the Agent Attributes and Stigmergic Quorum Detection sections, to direct how agents with expired commitments determine a decision for recommitment.

Finally, the common technique of iteration-based termination, in which particles continuously refine their best values up to a certain number of iterations, is also a feature for adaptation. Bees use a piping signal that, upon reaching a certain threshold, causes the bees to take flight, effectively terminating the decision making process. Unfortunately, implementing this requires some sort of globally-accessible variable, which is undesirable in applications for distributed systems, or a broadcast propagation that would be time and message intensive. Instead, a configurable parameter is used to limit the time the system is allowed to attempt to reach consensus. The factors that influence the number of iterations required to routinely reach a consensus is then determined.

3.7.3. Agent Attributes

As described, when bees in nature start their search, all of the scouts are uncommitted; only some of them find potential nest sites and become committed scouts. The adaptation to Honey Bee Consensus must differ here, since it operates under the assumption that all agents in the system have a preferred course of action at the start of deliberations. The first attribute of the agents is defined as a set of fixed values representing the preference weights for each of the m possible outcomes:

$$\boldsymbol{P} = \{p_1, p_2, \ldots, p_m\}$$

where p_i is the preference weight for the i th decision option.

The weight of each preference is used to calculate the duration for which the agent will remain committed to a decision before reconsidering its position. This is done by applying a function f_{init} that translates a preference weight into an integer representing the number of iterations to maintain commitment. In the simplest case, the preference weights are limited to integer values and used directly, but using a function allows the weight representation and translation operations to be easily as desired.

Agent *i*'s initial commitment duration, d_0^i, is then set by applying f_{init} to the largest value in $\boldsymbol{P}$:

$$d_0^i = f_{init}\left(\max_{p \in \boldsymbol{P}}(p)\right)$$

The commitment of a scout to its chosen site wanes over time, so each agent needs an evaporation parameter, ϵ, which is applied to d at time t by some decay function f_{decay} such that:

$$d_t^i = f_{decay}(d_{t-1}^i, \epsilon)$$

The details of this decay function in HBC are subject to future experimentation, but observed bee behavior suggests that it should be linear, on average (Seeley, 2003; Seeley & Buhrman, 1999). The following is an example for the decay function:

$$d_t^i = d_{t-1}^i \cdot (1 - \alpha)$$

where $\epsilon = 1 - \alpha$ and $0 < \alpha < 1$.

When $d_i < 0$, the scout becomes uncommitted and seeks a new decision to which to commit, as described later in the section on Stigmergic Quorum Detection. For now, it is only important to note that the agent could recommit to the same preference for which its commitment just expired, or it could change to a different preference because, unlike the initial decision value, recommitments are influenced by neighbor preferences and aggregated quorum information. This is different than the way natural bees act, since most bees will not change allegiances once their enthusiasm for their chosen sites expires; however, the process retains similarity to the way bees that have never found a site base their exploration of new sites upon neighbor (social) information, but evaluate the quality of the new site upon internal (cognitive) information as emphasized in (List et al., 2009).

This repetitive process of commitment decaying to non-commitment provides the mechanism required for the expiration of dissent that is important in preventing deadlock in the decision making process (Passino & Seeley, 2006). It also provides a way to

artificially create "uninformed" agents in a population that starts out partially informed with personal preferences for each decision. This injection of uninformedness is desirable because recent research has suggested that the presence of uninformed agents is an important component to swarm decision making in natural systems (Couzin et al., 2011).

3.7.4. Stigmergic Quorum Detection

Bees not only advertise a preferred site, but they visit it periodically. This is a form of stigmergy in that the number of bees present at a candidate site informs a visiting bee about how widespread the preference for the site is throughout the swarm. That is to say, while one bee might not have seen a second bee dancing at the swarm, the first bee can infer the second bee's commitment to a candidate site by its presence at the site.

The stigmergic data is important for an agent to be able to detect a quorum among distributed agents throughout the hive and detect the convergence to a consensus, so HBC uses the stigmergy design pattern (Babaoglu et al., 2006) to adapt bee behavior to agent behavior. In the absence of a globally accessible variable or mobile agents, this information must be propagated through the network somehow. In engineered systems this is usually achieved through message passing between network nodes (Babaoglu et al., 2006). Various agent-based design patterns for system communication and distributed coordination agree that this communication typically consists of aggregation, propagation, and evaporation of the relevant data (Babaoglu et al., 2006; Tom De Wolf & Holvoet, 2006; Gardelli et al., 2007; Gatti et al., 2009; Kasinger et al., 2009).

The relevant data for quorum detection is encapsulated in a *Quorum Data* object, Q, that contains the identifier of the candidate choice it encapsulates, c, and a set of agents that are known by this *Quorum Data* object to prefer its choice along with each agent's remaining commitment duration for the choice. Q can be represented as

$$Q = \langle c, A \rangle$$

and A as

$$A = \{(a_1, d^1), \ldots, (a_k, d^k)\}$$

This object is the message to be passed in the implementation of the stigmergy pattern and supports aggregation, propagation, and evaporation as follows:

3.7.4.1. Aggregation

Aggregation occurs when an agent adds itself to the set of agents contained in the object. There are two possibilities here: the receiving agent is either committed to a choice or it is neutral. It will be explain how an agent becomes neutral shortly.

If a committed agent receives any *Quorum Data* objects from its neighbors for the candidate choice it prefers, it performs a set union on the agent member sets and adds itself to the resultant set of agents in the object along with the remaining duration of its commitment, d^i. In the process of the set union the commitment durations of all agents are updated to the most current value. Thus, when an agent receives a *Quorum Data* object for a site it prefers and views the set of agents it contains, it is analogous to a bee visiting its preferred site and observing the other bees that are visiting it. If the agent was already a member of the set, it updates its associated commitment duration with its current value for that choice. Committed scout bees do not visit sites to which they are not committed, so they do not have visibility of quorum data for non-preferred sites; therefore, committed agents simply ignore *Quorum Data* objects for sites they do not prefer.

A neutral agent, on the other hand, is uncommitted. An uncommitted agent considers the quorum data of all of its neighbors. If, after combining all of the received quorum data, it detects a unique *Quorum Data* object that has accumulated a quorum (i.e. sufficient number) of agents in its membership set, defined by a model parameter, the agent will commit to that option, c^*, at the level determined by $f_{init}(p_{c^*})$. This behavior is meant to simulate the receipt of a "stop signal," (Seeley et al., 2012) essentially forcing an agent to join the quorum. On the other hand, if there is no unique quorum, the agent must become committed to one of the decision options before joining a *Quorum Data* object. Following

the findings concerning dance selection in (Visscher & Camazine, 1999) and the relationship between independence and interdependence in (List et al., 2009), the agent makes this commitment probabilistically based on the number of neighboring agents that prefer a decision using fitness proportionate selection, for example, tournament or roulette wheel selection. The agent then commits to that option, $\overline{c}$, at the level determined by f_{init} ($p_{\overline{c}}$) and behaves as committed to the selected decision until its enthusiasm expires and the process repeats.

3.7.4.2. Propagation

Propagation of quorum data occurs when an agent is polled for its opinion by a neighboring agent. In response to the polling, an agent will return a single *Quorum Data* object appropriate to its preferences, created from the aggregation of the last set of *Quorum Data* objects it created from the polling of its neighbors.

3.7.4.3. Evaporation

Evaporation in a *Quorum Data* object is related to the aforementioned commitment duration associated with each agent in the *Quorum Data* structure. The length of time a bee remains committed to and dances for a preferred site is based upon the perceived quality of the site (Seeley & Buhrman, 1999). Because bees periodically revisit the site to which they are committed, this value also factors into the number of times the bee returns to the preferred site and its likelihood of being perceived by other agents as a member of the quorum for it. In other words, an agent should remain a member of the *Quorum Data* objects it has joined and that are being propagated through the network only about as long as the agent is committed to the sites contained by those *Quorum Data* objects. This can be approximated by applying f_{decay} to the value d associated with each agent $a \in Q.A$ at each iteration and refreshing d with the current value every time an agent receives a *Quorum Data* object in which it is already a member.

3.8. Chapter Summary

In this chapter, the Honey Bee Consensus model has been described following the ODD template. The observed behavior of honey bees has been mapped to Honey Bee Consensus rules, especially with regard to information propagation between neighbors and reaction to quorum formation. It has also been described how the aggregation, propagation, and evaporation patterns are applied to replicate the stigmergic effects achieved through the mobility of bees in natural environments. In the next chapter, the methods to be used to implement and evaluate the performance of Honey Bee Consensus is explained.

4. Evaluation Methodology and Experiment Design

4.1. Overview

In this chapter, the experiments conducted with HBC to determine its feasibility for use in distributed negotiation and consensus formation are explained. The experiments were designed to answer the following research questions:

1. How well does HBC work, overall, for achieving socially-optimal consensus in commonly-studied social network models? The performance metrics in this regard are whether or not a consensus is reached in the allotted number of negotiation rounds, the number of negotiation rounds required to reach consensus, and the social utility of the resulting consensus.
2. Does the quorum size parameter have a significant impact on the speed of distributed consensus formation in HBC, and, if so, how does the choice of quorum size value affect the balance and tradeoffs between successful consensus formation, the number of negotiation rounds required to reach consensus, and the quality of the final consensus as measured by its social utility?
3. How does the social network model and topology affect the performance of HBC?

4.2. Methodology

4.2.1. Evaluating the Overall Performance of Honey Bee Consensus

To evaluate HBC's performance an algorithmic analysis of its time complexity is presented and compared to the time complexity analyses of established techniques for distributed consensus negotiation over two possible choices. This provides HBC's worst-case performance with respect to negotiation rounds required to attempt to reach consensus and allows the practicality of using the compared methods for solving the Networked Biased Voter and Majority Coordination problems to be determined.

As discussed in Chapter 2, however, one of the key benefits of HBC over other techniques is its ability to negotiate consensus from among more than two possible choices in addition to considering the agents' weighted biases; therefore, the predominance of this research has focused not on competing with two-outcome distributed consensus techniques, but on gathering empirical performance data for HBC on problems with more than two possible choices on a variety of social network topologies.

General trends in network topology effects on HBC's performance were examined with respect to speed, accuracy, and failure rates. Performance metrics were measured for all trials performed on a given social network model at the same population size and combined, regardless of the parameters used to construct the individual random graphs. Using this data, the average number of ticks to consensus (discarding the outlier cases where the metaheuristic failed to achieve consensus in the allotted time); the percentage of times a given topology resulted in the desired outcome, given that a consensus was reached in the allotted time; and the percentage of times the topology failed, either because it resulted in an undesired consensus or failed to reach consensus at all, were plotted, each with 95% confidence intervals.

Since all of the underlying social networks for these plots were created using different model parameters, this data view provided insight about how HBC performed on a given social network model in general, regardless of the specific parameters chosen for the model. In particular, trends in the plots of the performance metrics versus the quorum size were examined, since one of the research objectives was to be able to provide quorum size parameter guidance for the quorum sensing design pattern. The confidence intervals allowed determination of the statistical significance of performance differences at the tested quorum size thresholds and examine the variance in performance for different social network topologies. As HBC yields heuristic results, the confidence intervals also allowed inference of the expected performance of the technique for a given class of social networks.

These evaluations necessitated that an appropriate set of models be selected to construct the social networks and that a way to determine the correct, or desired, consensus result, given the initial preferences and weighted biases of the social network members, be

defined. The decisions and rationale for these choices are explained in the next two sections.

4.2.1.1. Social Network Models

The social network models upon which the experiments were ran were the Watts-Strogatz (Watts & Strogatz, 1998), Barabási-Albert (Barabási & Albert, 1999), and Erdős-Rényi (Erdos & Rényi, 1959; Gilbert, 1959) models due to their common use in the study of social and complex networks (Strogatz, 2001). Each of these models provides an algorithm for randomly generating networks with short average path lengths, also known as "small-world" networks; however, they result in different connectivity structures with respect to the degree distribution of the nodes and/or the clustering coefficient (Amaral, Scala, Barthélémy, & Stanley, 2000; Porter, 2012). Specifically, the Watts-Strogatz model yields networks with normal degree distribution and high clustering coefficients, the Barabási-Albert model yields scale-free networks (networks in which the degree distribution follows a power law), and the Erdős-Rényi model yields random networks with clustering coefficients that are related to the number of edges in the network. By comparing HBC's performance on each of these topologies, the ways in which these unique network characteristics that appear in naturally-occurring social networks do or do not influence consensus formation can be determined.

The Python NetworkX library (Hagberg et al., 2008) was used to create the random networks with a variety of parameter values, as described in the Experiment Design section. The library was also used to ensure that each randomly-generated network was connected (i.e., that there was at least one path between any two nodes of the network), since this property is required to ensure consensus. Each of the social network models uses different parameters to guide its random construction of a social network with the desired characteristics. Generally speaking, the models are configured by parameters and rules that determine the number of nodes in the network, the number of edges in the network, and the probability of establishing an edge between two nodes. Here, a brief overview of these well-known models is provided.

Erdős-Rényi Model

The Erdős-Rényi model may refer to either of two model variants for constructing random graphs: the $G(n, M)$ model or the $G(n, p)$ model (West, 2001). In both models, the parameter n indicates the number of nodes in the graph. The $G(n, M)$ model yields a random graph from the set of all graphs with n nodes and M edges. The $G(n, p)$ model, on the other hand, connects every pair of nodes with a fixed probability p. For the purposes of this research, the $G(n, M)$ model was primarily used because the ability to specify the number of edges in the network allowed for a more direct comparison between the Erdős-Rényi random graphs and those created with the Watts-Strogatz model, which also produces social networks with a fixed number of edges. As described in the section on the Watts-Strogatz model, it can easily be used to yield $G(n, M)$-style Erdős-Rényi random networks by fixing the rewiring parameter. The only exception to this was the usage of the $G(n, p)$ model for the random networks of 200 agents seeking consensus from two outcomes. This variation allowed testing HBC's performance on a set of Erdős-Rényi random graphs with varying edge counts in a single trial group.

Watts-Strogatz Model

The Watts-Strogatz model provides a way to algorithmically create a family of social networks that have a small average path length and high clustering coefficient. It is constructed by connecting each node on the circumference of a ring to its k nearest neighbors. Then, each edge in the network is removed with uniform, independent probability p and "rewired" to connect a pair of nodes chosen uniformly at random (Porter, 2012). Figure 4.1 depicts an example starting network constructed with $k = 6$ and what the network might look like after four random re-wirings.

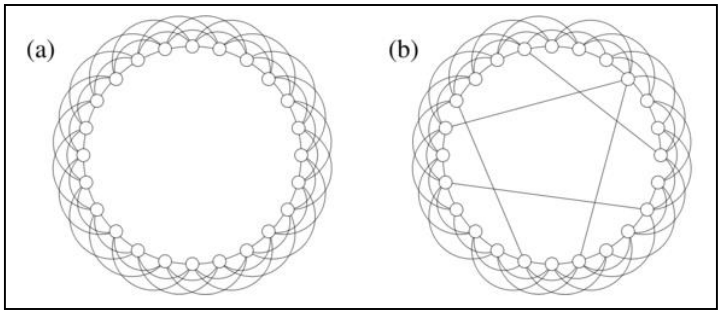


Figure 4.1. Starting graph for Watts-Strogatz model with k = 6 (a) and after 4 random re-wirings (b) (Porter, 2012).

For this model, as $p \to 1$ the clustering coefficient approaches zero as the number of nodes $n \to \infty$. At $p = 1$, the resulting structure is the equivalent of a network constructed with the $G(n, M)$ variant of the Erdős-Rényi model, where $M = \frac{n \cdot k}{2}$ (Porter, 2012).

Barabási-Albert Model

The Barabási-Albert model decides the existing nodes to which new nodes should connect using preferential attachment—nodes with a high number of neighbors are more likely to be connected to a new node than those with low numbers of neighbors. The parameter that affects this choice is *m*, which dictates the initial number of (unconnected) nodes in the starting graph and also the number of new, preferentially attached edges added with each additional node, up to the *n*th node. The result of this model is a network in which the nodal degree distribution follows a power law where there are a small number of nodes with high degree and a small number with very low degree. This approximates the condition in social networks where there are a small number of very popular or influential individuals, but most individuals only have a fairly small local community.

4.2.1.2. Evaluation Metrics

As discussed in the literature review, the rationality of an outcome can be judged in different ways. Following (Endriss et al., 2006), in this research outcomes are evaluated with respect to the resultant social welfare of the artificial multi-agent society. A number

of possible social welfare metrics are presented in (Brams & Fishburn, 2002; Chevaleyre et al., 2006; Shoham & Leyton-Brown, 2009; Taylor, 2008; Wooldridge, 2009); plurality voting, Borda voting, and range voting are the three used in the presentation of these results.

Since each of these voting protocols can yield a different winning result under the same conditions, there is no definitive way to establish one method as more correct than any of the others. In light of this, the desired consensus, with respect to social utility, is defined to be that consensus which would agree with the results of the majority of these three voting protocols for a given preference initialization. The remainder of this section consists of a description of each voting protocol used as evaluation metrics and an example illustrating how they can yield different winners under the same conditions. One difference is noted in the application of these voting protocols in that in HBC the agents are allowed to rank options equally if they both have the same utility. Allowing ties departs from what is traditionally allowed in Borda and range voting where each outcome must be assigned a distinct ranking; however, this rule is not enforced due to the differences in the goal of voting, in which one is trying to determine a definitive winner, and consensus formation, in which one is concerned with determining any acceptable outcome of high social utility and, therefore, where ties are not a problem.

Plurality Voting

In plurality voting, each decision maker casts a vote for their top choice. The choice that receives the most votes wins. In this data collection, tied choices are considered equally good; therefore, if HBC picks any of the tied choices we consider it to have picked an acceptable outcome with respect to plurality voting.

An interesting case with plurality voting is that the result may not yield the highest utility for the collective because it does not take into account any of the individual preference weights for the outcomes, it gives full weight only to the top choice. Consider the weighted voter preferences in Table 4.1. In this example, Candidate A would win the election in plurality voting because it is the first choice of Voters A and B. By some

measures, however, Candidate B would yield a higher collective utility because that candidate is relatively acceptable to Voters A and B while being much more acceptable to Voter C than Candidate A. The next two methods attempt to address this apparent paradox.

Table 4.1. Example of voters with weighted preferences.

	Candidate A	Candidate B	Candidate C
Voter A	0.50	0.40	0.10
Voter B	0.40	0.35	0.25
Voter C	0.05	0.60	0.35

Borda Voting

In Borda voting, each voter rank orders all of the candidates. Each rank is given a fixed weight. The accumulated weights for each candidate are summed, and the candidate with the highest weighting wins. Using Borda voting with the preference weights in Table would result in the weightings shown in Table 4.2.

Table 4.2. The Borda weights resulting from the preference weights in Table 4.1.

	Candidate A	Candidate B	Candidate C
Voter A	3	2	1
Voter B	3	2	1
Voter C	1	3	2
Sum	7	7	4

In this particular case, it can be seen that Candidates A and B would be tied. This results in better representation of Voter C's preferences, but it does not take into account Voter C's large relative preference for Candidate B over Candidate A. This information can be incorporated with the next voting protocol.

Range Voting

The reason Borda voting results in a tie between Candidates A and B is because, even though the ranks are weighted, the weighted difference in preferences between outcomes is ignored. With range voting, all candidates are rank ordered, just as in Borda voting, but each candidate receives a weight, chosen by the voter, within a specified range. In HBC, weights between 1 and 99 are assigned, corresponding to the preference value, so the preference weights in Table 4.1 result in the range voting results in Table 4.3.

Table 4.3. The range voting weights resulting from the preference weights in Table 4.1.

	Candidate A	Candidate B	Candidate C
Voter A	50	40	10
Voter B	40	35	25
Voter C	5	60	35
Average	31.67	45	21.67

As shown, the weights are averaged to get the final result, showing that Candidate B is the most acceptable to all voters when considering global preferences and relative weights.

4.2.2. Determining the Impact of the Quorum Size Parameter

To determine whether or not the quorum size has a significant effect on HBC's time to consensus metric, and to discover any confounding effects on consensus speed from the social network model parameters, the Minitab Statistical Software package ("Minitab 17 Statistical Software," 2010) was used to perform analyses of variance (ANOVA) setting *simulation ticks* (i.e., negotiation rounds) as the response variable and *quorum size* and the applicable social network model parameters as the independent variables defined by factors in the ANOVA model. The ANOVA analysis was performed for each class of social network, combined across all tested network sizes to allow the analysis to be valid for describing the significance of factor impacts on social networks of the class in general. Since the possible values for quorum size and model parameters are limited in range by the size of the social network, the factor levels were normalized across different sized networks

of the same class by considering the factor levels in ranges determined by percentage of the social network population. These ranges were labeled low, medium, and high and these ranges are defined in Appendix A.

4.2.3. Determining the Impact of the Social Network Model

The previously described ANOVAs examine factor impacts within a particular social network model. To determine the impact of the individual social network models on HBC's consensus formation, additional ANOVAs were performed on consolidated trial data for different network models of similar population sizes. Since different social network models use different parameters in network construction, the only factors that could be examined in these ANOVAs were model type and quorum size.

4.2.4. Verification and Validation

The correctness of simulations based on a model is usually described in terms of verification and validation. Verification is the process of ensuring the model is implemented correctly with respect to its specification (Yilmaz, 2006), in other words, whether or not one is "building the model right" (Balci, 1986). Validation is the process of ensuring the model produces results accurate enough to serve its intended purpose (Robert G. Sargent, 1996). In this case, ensuring one has "built the right model" (Balci, 1986).
A model's validity is considered relative to the context of its experimental conditions and its response accuracy. For the proposed research, the intended purpose of the model is to allow distributed, autonomous agents to reach a desired consensus in a decentralized, self-organizing way. The creation of the right model can be validated if it produces the expected result with sufficient degree of accuracy. The definition of the "optimal" result, however, is variable, as is the minimum frequency of consensus formation needed to declare success.

In (Tan, 2010), the optimal result of the NBVP is the result held by the majority of the voters at the beginning of the deliberations, but the canonical NBVP only considers two possible decision values. As it has been shown, additional possible decision values create a condition where the optimal choice can be defined differently than just the majority

opinion. In these experiments, the desired consensus result is considered to be one that agrees with a majority of the three centralized social choice protocols mentioned in section 4.2.1.2, and any case that results in a consensus that is not in the majority result returned by the social welfare metrics, including cases where the metaheuristic fails to reach a consensus in the allotted time, is considered an undesirable result. Based on these definitions, comparison to other models is being used, as described in (Robert G Sargent, 2005), as the validation technique.

Similarly, it is not specified how frequently an appropriate outcome must be chosen in order for the algorithm to be considered successful because, as observed by Dréo et al. (2006), there is no straightforward way to compare iterative optimization methods; the quality of the result is often dependent upon the length of time the algorithm is allowed to run, which can be freely chosen by the user, and the runtime characteristics of metaheuristics are strongly tied to the chosen parameter values. Instead, the goal is to characterize how frequently the algorithm results in consensus, given a specific amount of allotted negotiation rounds, for different experimental conditions such as network topology, number of voters, and quorum size in order to allow those implementing the algorithm to engineer its performance in accordance with their needs. As a general benchmark, however, it is clear that a success ratio at least significantly greater than $1/x$, where x is the number of choices would be desired, since a value close to $1/x$ would be expected to be obtained by random chance.

With regard to verification, it must be ensured that the entities, processes, and associated constraints and assumptions of the programmed model are an accurate realization of the proposed design. This can be achieved with model analysis and testing. Model analysis is a static analysis of the software to predict control and data-flow properties of interest. Model testing is achieved by subjecting the implementation to test cases for which the expected result in known and comparing the actual output to the expected output (Yilmaz, 2006).

Prior to conducting experiments with HBC, validation and verification was performed by repeatedly running HBC on trial networks and preference distributions and

using the simulation framework's visualization capabilities to observe the progression of negotiation and consensus formation in the network. A histogram of the number of agents preferring each possible outcome provided a visual indication of the preference distribution upon initialization and as the simulation progressed to completion. Color-coding of the agents in the network visualization showed which agents preferred which outcomes at different times. Observation of these visualizations provided face validation that HBC was performing as intended. Comparison of the initial preference distribution to the resulting consensus, as shown in Figure 4.2, allowed verification that the expected result was routinely returned, either by way of returning the clear majority winner or, in some cases, forming consensus for an option that was a close runner-up, indicating the return of a consensus likely to be favored by Borda or range voting.

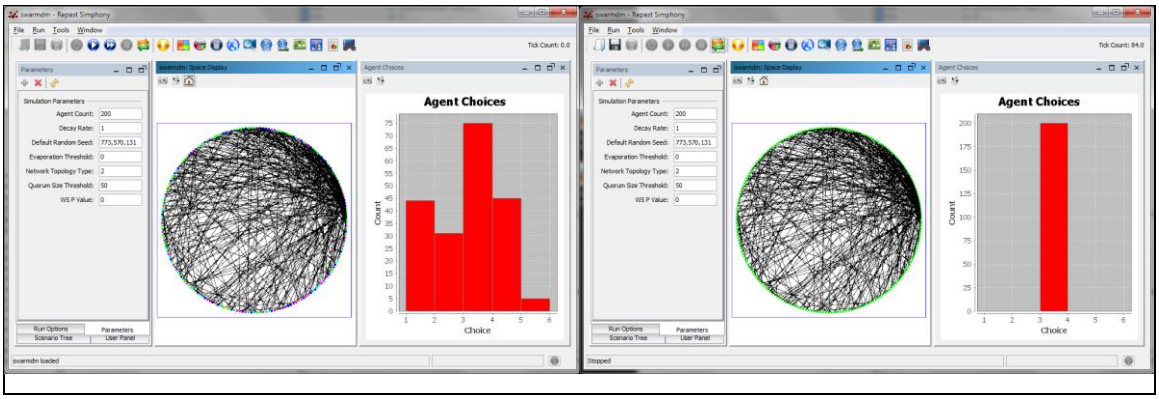


Figure 4.2. Histogram and network visualization showing initial and final agent preferences and distribution. In this run, the clear majority preference at initialization is chosen by all agents after only 84 negotiation rounds.

4.3. Experiment Design

Honey Bee Consensus was evaluated on social networks of size *n* equal to 25, 200, and 1,000 agents. These values were chosen in order to provide a sample of network sizes that spanned several orders of magnitude so that significant differences could be detected as the network size grew and to test the scalability of the technique. For each network size, a set of random social networks was produced using the three aforementioned models. Some of the model parameters are bounded by the number of nodes in the network (e.g.:

neighbors per node in Watts-Strogatz and number of initially-connected nodes in Barabási-Albert), therefore, for these parameters, values from a predefined range of low, medium, and high values from between 1 and $n/2$ were chosen. These range definitions are provided in Appendix A.

Choosing parameters for the rewiring probability parameter, p, in the Watts-Strogatz models was less straightforward. Initial sensitivity tests showed that for $n = 200$ and $k = 20$, p values in the range [0, 1] at steps of 0.1 produced almost identical metaheuristic performance for $p \geq 0.1$. It was determined that the cause of this was the fact that these p values did not yield social networks with a sufficiently diverse range of diameters for producing different results. In order to remedy this, values for p were generated in the range (0, 1] such that each network at a given k value had a different diameter. The value $p = 0$ was excluded, since, by the Watts-Strogatz model, this would result in a k-regular large-world ring network different than the structures to be studied. Watts-Strogatz models generated with $p = 1$ simply yield Erdős-Rényi random graphs in accordance with the Erdős-Rényi G(n, M) model, so this was the technique used to create the Erdős-Rényi model social networks. Appendix A contains a comprehensive table of the social network parameter configurations used in experiments.

As indicated by the research questions, the effects of the quorum size parameter value was of particular interest. By definition, a quorum size greater than $n/2$ represents a simple majority, but it is desired to achieve consensus with considerably less global knowledge; therefore, testing of quorum size values was limited to those less than or equal to half of the social network population. In the cases of population sizes 50 and 200, quorum sizes were tested in increments of 5, but as the size of the population reached 1,000 this became infeasible due to the time required to run many random trials, so quorum sizes in increments of 25 were used for these trials.

Experimenting with common random numbers, for a given network size, 30 sets of random preference weight configurations were used, and these preference weights were initialized to the same network locations, with the same neighbors, for each of the random social network models created of that network size, as listed in Appendix A. Using the

metaheuristic, trials were run on each of these 30 configurations for each of the social network configurations for every quorum level, giving a total of $(30 \times num\ networks \times num\ quorum\ size\ levels)$ trials for each of the social network model and population size combinations. The results from these trials were used to calculate the average performance and variance of the metaheuristic over a range of social network model variations at each quorum size level.

For all of the experiments, the evaporation rate and evaporation threshold were maintained as control variables. While these variables guide the propagation of consensus through the social network, initial sensitivity tests suggested that they did not have a significant impact on the results when the values were the same for all agents, and adding experimentation levels would have added independent variable complexity in excess of that required for the scope of this research. Further exploration of the impacts of these parameters is proposed as future work, however.

4.4. Chapter Summary

This chapter has described the social network topological conditions under which HBC's performance was studied and the criteria used to evaluate that performance. It has been explained how the parameter values for the metaheuristic model, described in the previous chapter, were chosen and how the experimental design was structured for the simulation trials in order to answer the research questions regarding the performance of the metaheuristic. Table 4.4 provides a summary of the design variables. The next chapter presents the results of these experiments.

Table 4.4. Summary of experiment design variables.

Dependent Variables	
Negotiation Rounds (Simulation Ticks)	The number of rounds of negotiation required to reach a consensus.
Percent Desired Outcome Returned Given Consensus Reached	The percentage of trials resulting in a consensus in accord with the majority of the reference voting protocols, given that a consensus was returned.
Percent Desired Outcome Returned Overall	The percentage of trials resulting in a consensus in accord with the majority of the reference voting protocols out of all attempts, including failures.
Independent Variables	
Quorum Size	The number of agents that must agree on an outcome preference to form a quorum that drives undecided agents to go along with their preference.
Population Size (n)	The number of agents in the social network.
Social Network Model	The model used to generate the random social networks (Watts-Strogatz, Barabási-Albert, or Erdős-Rényi).
Social Network Model Parameters	The model-specific parameters controlling random variability (e.g.: p and k in Watts-Strogatz or m in Barabási-Albert).
Pseudorandom number generation seed.	The seed for the pseudorandom number generator that determines the initial preference distributions and the stochastic agent choices in each negotiation round.
Control Variables	
Evaporation Rate	The rate at which an agent's preference strength for the currently-preferred outcome decays toward zero.
Evaporation Threshold	The preference level below which an agent becomes undecided.
Maximum Negotiation Rounds	The number of rounds of negotiation allowed before aborting the simulation.

5. Experiment Results

5.1. Overview

This chapter presents the results of the experiments conducted. As described in Chapter 4, a full factorial experiment design was used to evaluate HBC performance in terms of speed to consensus, the percentage of desired outcomes when consensus was reached, and the percentage of failed outcomes out of all the trials with respect to quorum size, population size, social network model, and 30 pseudorandom number generator (PRNG) seeds. The experiment design matrix is given in Table 5.1. Quorum sizes and social network parameters limited by the population size are divided into low, medium, and high ranges, as detailed in Appendix A, to allow comparison across trials of different population sizes.

Table 5.1. Factorial design matrix.

<table>
<tr><th>Factor</th><th>Levels</th><th colspan="3">Sub-Factor Levels</th></tr>
<tr><td>PRNG Seed</td><td>0 - 29</td><td rowspan="3">p</td><td rowspan="3">k</td><td rowspan="3">m</td></tr>
<tr><td>Population Size</td><td>50, 200, 1000</td></tr>
<tr><td>Quorum Size</td><td>Low, Med, High</td></tr>
<tr><td rowspan="3">Social Network Model</td><td>Watts-Strogatz</td><td>Low, Med, High</td><td>Low, Med, High</td><td></td></tr>
<tr><td>Barabási-Albert</td><td></td><td></td><td>Low, Med, High</td></tr>
<tr><td>Erdős-Rényi</td><td></td><td>Low, Med, High</td><td></td></tr>
</table>

Experiment results empirically show that HBC performs significantly better than chance at yielding a socially desired consensus from among two or five possible outcomes on all three of the social network models tested for population sizes ranging from 50 to 1,000. It is also found that the quorum size parameter has a significant impact on the number of negotiation rounds required to reach the desired consensus in all of the social network models. The number of negotiation rounds does not appear to be significantly impacted by the social network model for small populations, but differences in this metric

become more pronounced as the population size increases. The plots of the performance data are presented in the remainder of this chapter.

5.2. Honey Bee Consensus Performance Results

The first metric used to evaluate HBC's performance was the number of negotiation rounds required to reach a consensus, which is the equivalent of the number of simulation ticks required to reach consensus. Figures 5.1 through 5.3 contain the plots of the average number of negotiation rounds, with 95% confidence intervals, required for reaching consensus versus the quorum size parameter value for each of the social network sizes and models over five choices of outcome.

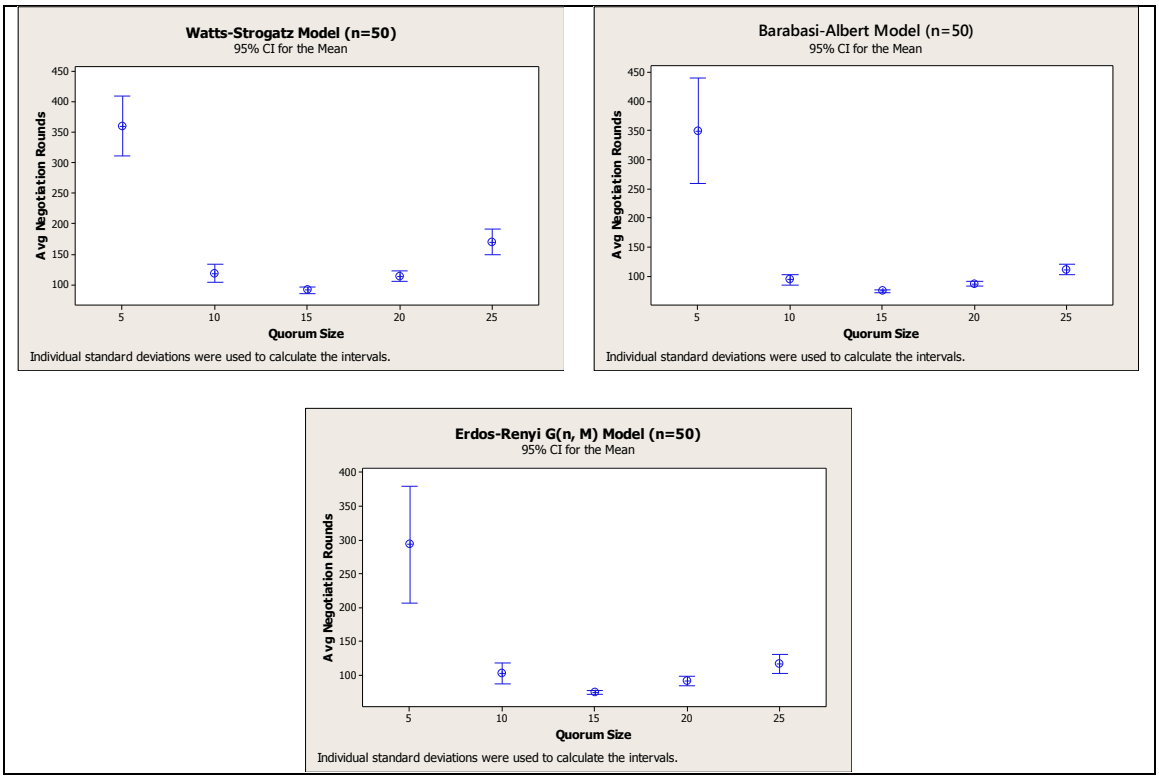


Figure 5.1. The 95% confidence interval plots for average negotiation rounds to consensus for all social network models of population size 50.

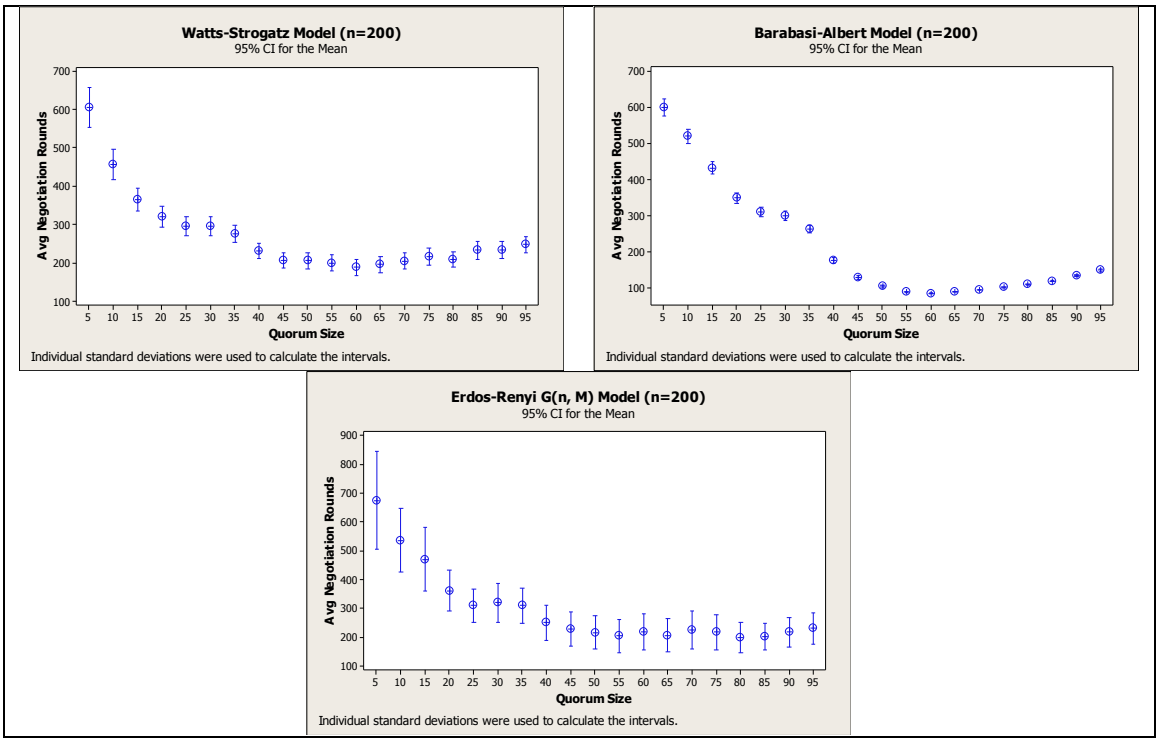


Figure 5.2. The 95% confidence interval plots for average negotiation rounds to consensus for all social network models of population size 200.

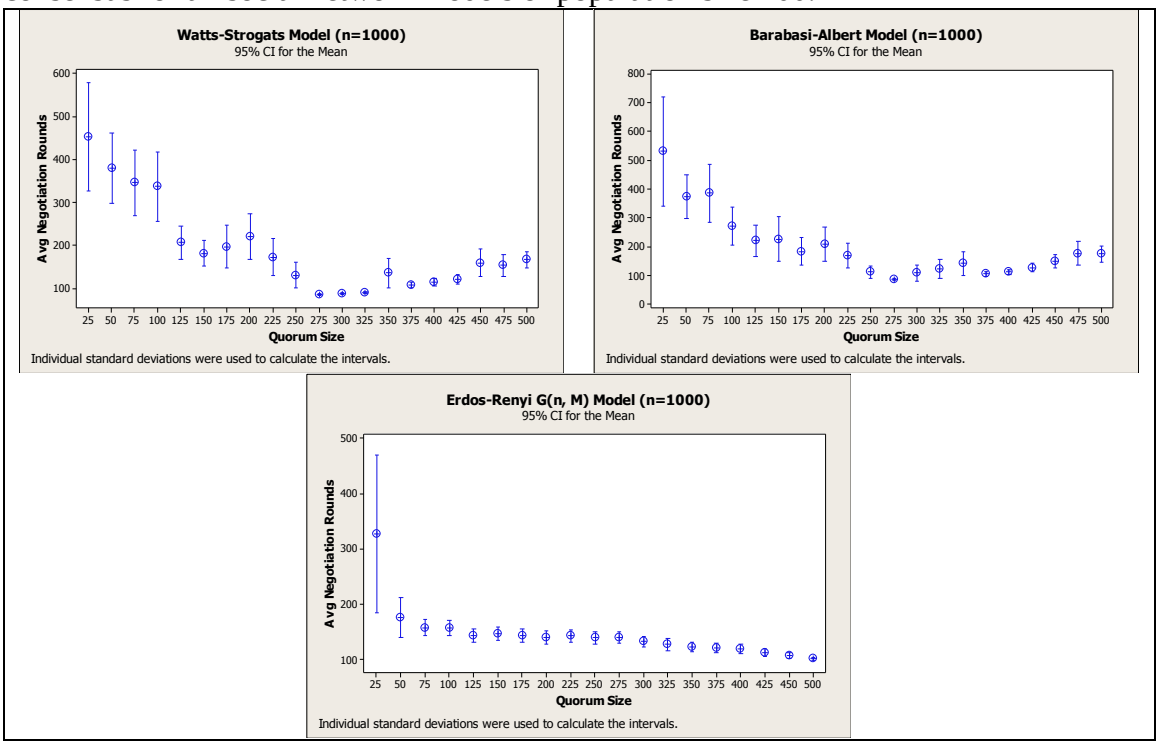


Figure 5.3. The 95% confidence interval plots for average negotiation rounds to consensus for all social network models of population size 1,000.

These plots show that, regardless of the social network model and population size, there is at least one quorum size for each that can be reliably expected to yield consensus in a reasonable number of rounds of negotiation. It is observed that extremely small quorum sizes, relative to the total population size, require significantly more negotiation rounds than larger quorum size values to reach consensus. It is also apparent that the smallest numbers of rounds required to reach consensus tend to cluster around quorum size values slightly greater than one quarter of the total population size. Finally, the general shapes of the plots often exhibit similar characteristics across different social network models. For instance, the shapes of all of the plots for $n = 50$ are strikingly similar, and, while the similarities become less pronounced as the population size increases, similarities are still noted, such as the performance plateau between quorum size values of 20 and 35 in the networks of population size 200 and the peaks at quorum size values of 200 and 350 in Watts-Strogatz and Barabási-Albert networks of population size 1,000.

While speed to consensus is an important performance metric, the plots in Figures 5.1 through 5.3 are too generous in that they show the number of negotiation rounds to reach any consensus, regardless of whether or not it is the desired consensus as defined in Chapter 4. Figures 5.4 through 5.6 contain the plots showing the percentage of trials, with 95% confidence intervals, in which the final consensus reached is a desired consensus versus the quorum size parameter value for cases in which HBC terminates in the allotted time.

The plots in Figure 5.4 through 5.6 show that the percentage of consensuses reached in accord with the desired outcome increases as the population size increases. It is also observed that, when there is a statistical significance between the results at each quorum size, the best consensuses are obtained by the smallest quorum sizes, and the range of quorum sizes yielding a high percentage of desirable outcomes increases with the population size. This performance is in contrast with the results in Figures 5.1 through 5.3 where the smallest quorum sizes produced the worst results in terms of the number of negotiation rounds.

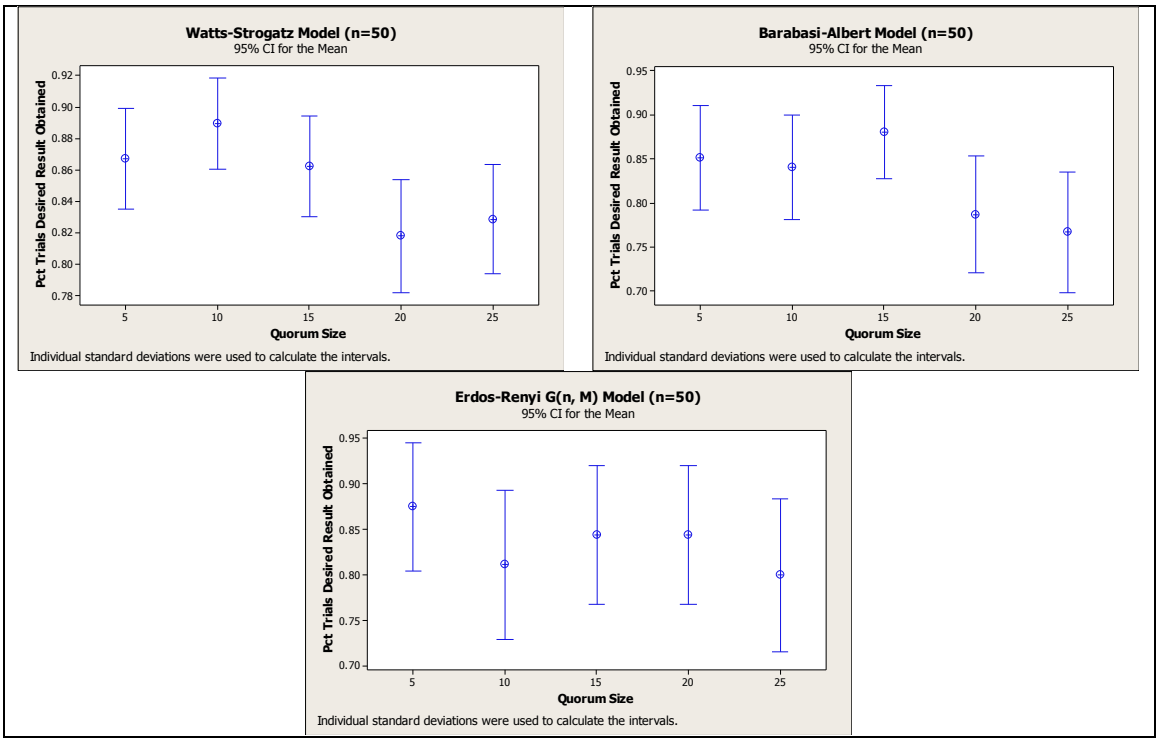


Figure 5.4. The 95% confidence interval plots for percentage of success in achieving desired consensus for all social network and models of population size 50.

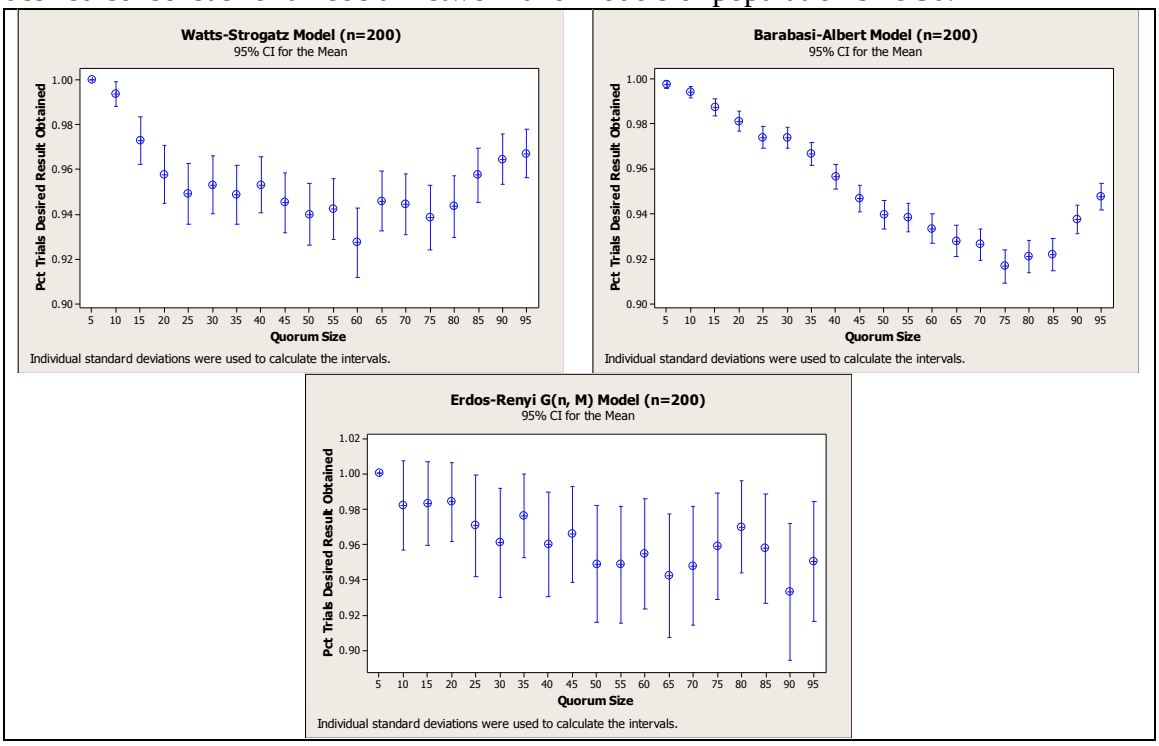


Figure 5.5. The 95% confidence interval plots for percentage of success in achieving desired consensus for all social network and models of population size 200.

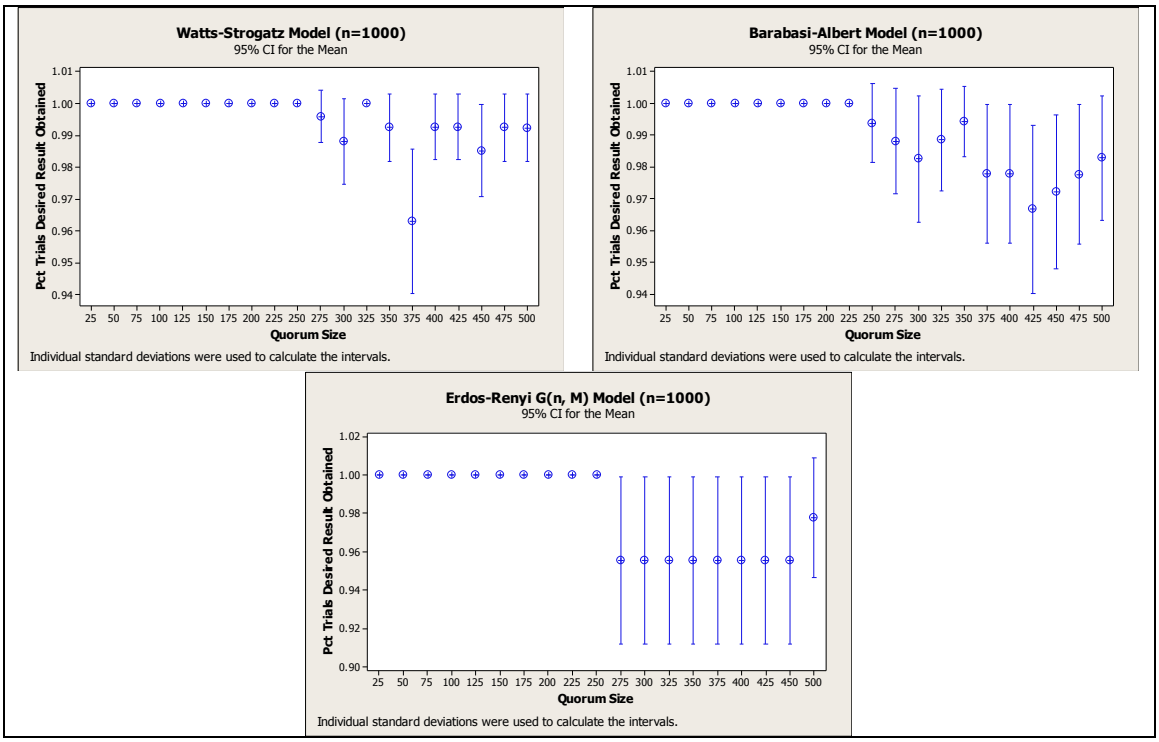


Figure 5.6. The 95% confidence interval plots for percentage of success in achieving desired consensus for all social network and models of population size 1,000.

As explained in Chapter 4, the plots in Figures 5.1 through 5.6 consider only those cases in which a consensus was ultimately reached. Trials that exceeded the maximum number of negotiation rounds are not included. It is logical to consider these excluded cases as failures, even though they might have ultimately reached a consensus if allowed to continue. Plotting the percentage of trials, with 95% confidence intervals, in which either no consensus or a non-desired consensus is reached versus the quorum size parameter value, the results shown in Figures 5.7 through 5.9 are obtained.

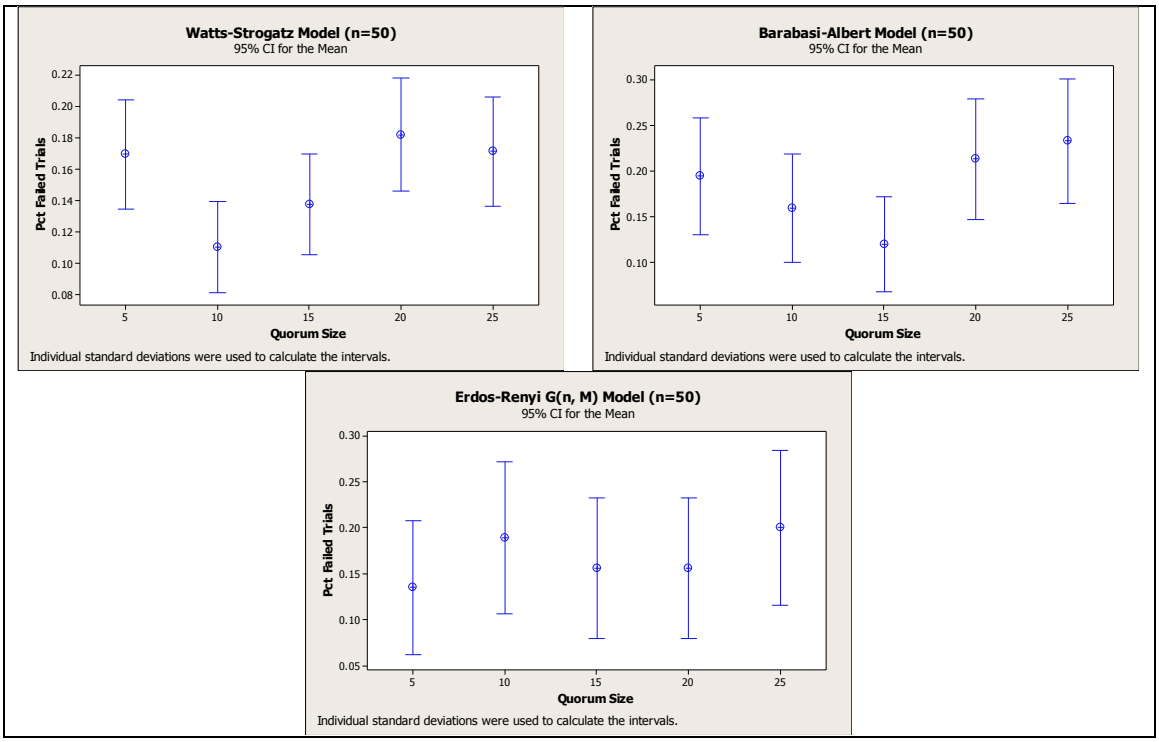


Figure 5.7. The 95% confidence interval plots for percentage of failure in achieving any consensus or the desired consensus for all social network models of population size 50.

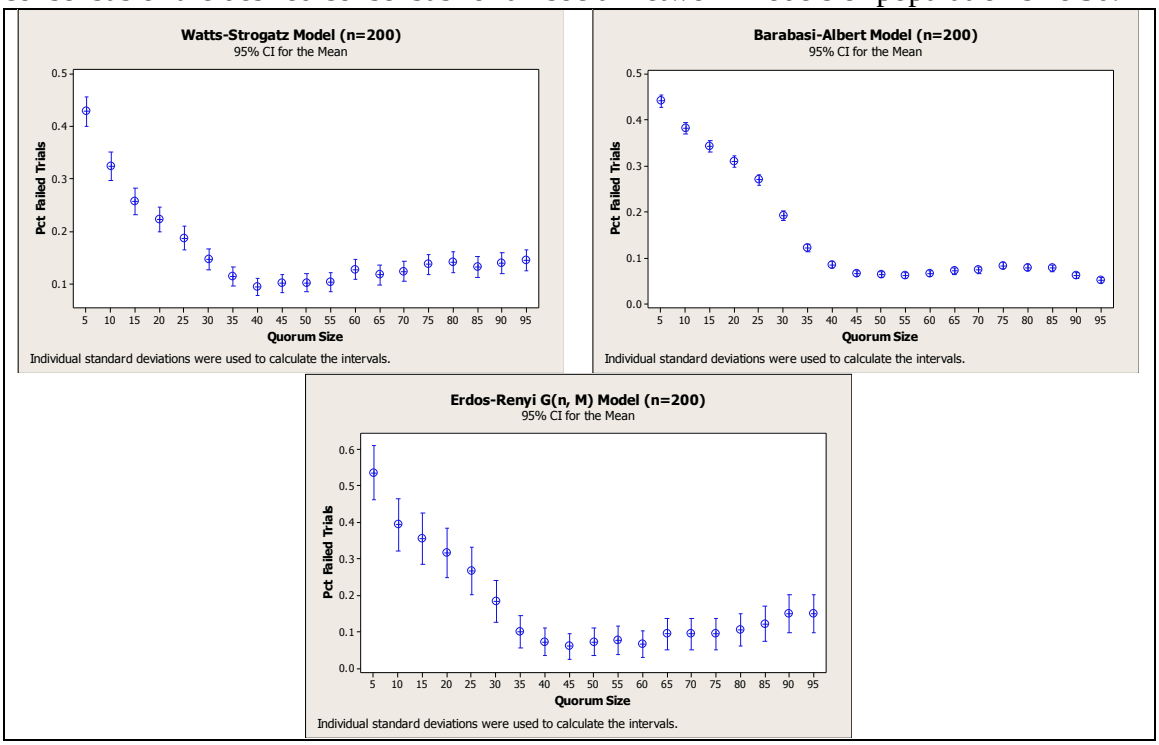


Figure 5.8. The 95% confidence interval plots for percentage of failure in achieving any consensus or the desired consensus for all social network models of population size 200.

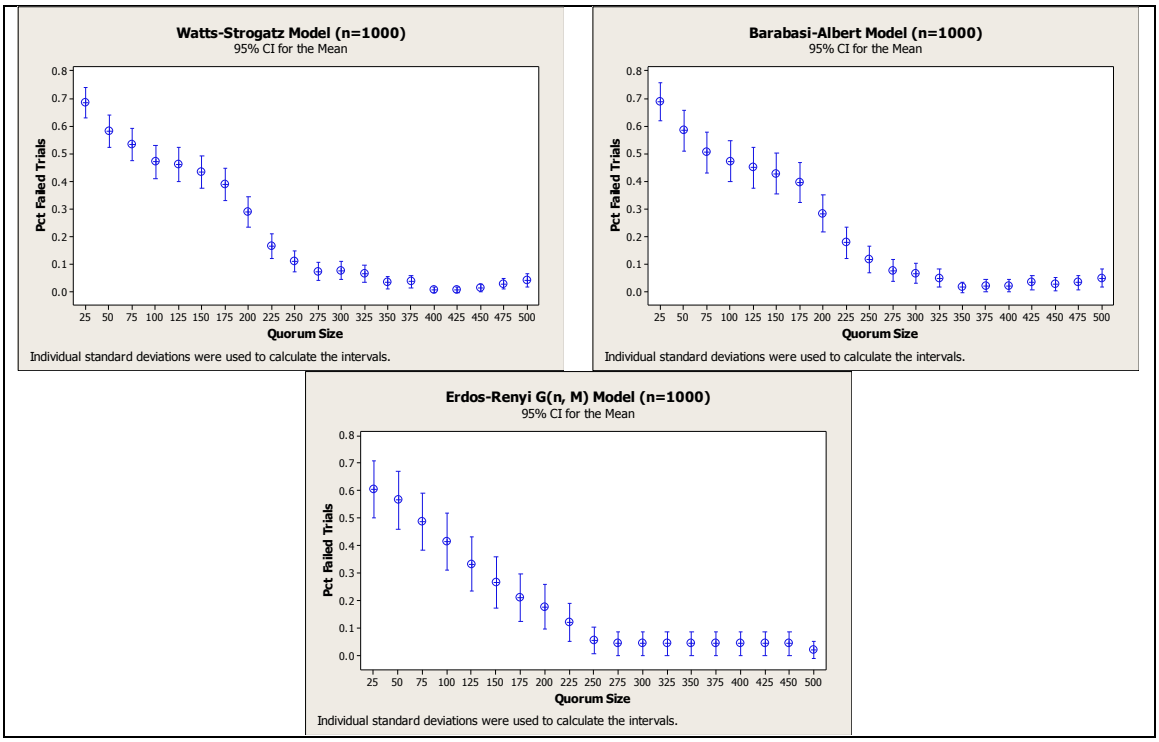


Figure 5.9. The 95% confidence interval plots for percentage of failure in achieving any consensus or the desired consensus for social network models of population size 1,000.

In the cases where all trials for a given quorum size parameter terminated with some consensus, these plots are simply the opposite of those in Figures 5.4 through 5.6; however, the results in Figures 5.7 through 5.8 contain the additional information of runs that exceeded the allotted number of negotiation rounds. As opposed to the results from Figures 5.4 through 5.6, where the best results were obtained at the smallest quorum sizes, in Figures 5.7 through 5.8 the smallest percentages of failures, when considering trials that exceeded the allotted time, are observed at generally larger quorum sizes.

From these plots, it can also be seen that each social network model and population size has a range of quorum sizes for which the success rate of HBC is significantly higher than that which would be expected by random chance. This validates HBC's effectiveness for the assigned task. It remains to determine how to select a quorum size that results in both effective and feasible performance, which is discussed in Chapter 6.

5.3. Impact of the Quorum Size

Figures 5.1 through 5.3 indicate that the quorum size parameter has a significant impact on the number of negotiation rounds required for HBC to yield a consensus. ANOVA were conducted for each of the tested social network models in order to confirm the significance of the quorum size parameter and identify any possible confounding effects on consensus speed introduced by the parameters used to generate the social networks. Appendix B contains extended ANOVA results; a summary of the ANOVA results for each class of social network are shown in Figures 5.10, 5.11, and 5.12.

In these ANOVA, the null hypothesis is that there is no difference between the mean number of negotiation rounds required to reach consensus for different combinations of quorum size values and social network model parameters; however, the results of the ANOVA show that the quorum size does have a significant impact on the number of negotiation rounds required. For each social network model, $p < 0.001$ for the quorum size factor, so the null hypothesis is rejected. The p-values for other factors and factor combinations indicate a significant impact in the ANOVA results as well, but the F-values of the quorum size level are significantly higher than the other factors' F-values making it more likely that the difference in samples is due mostly to the quorum size factor. The p-values for most of the model parameters (e.g.: *p*, *m*, and *k* in the Watts-Strogatz, Barabási-Albert, and Erdős-Rényi models, respectively) do not appear significant, indicating that HBC's performance characteristics are robust to variations within the individual social network models, regardless of the parameters used to construct them. One exception is the *k* parameter in the Watts-Strogatz model, which affects the number of neighbors each agent has and thus the size of the small world clusters.

General Linear Model: ticks versus n, k-level, p-level, qs-level

```
Method

Factor coding           (-1, 0, +1)

Box-Cox transformation
Rounded λ               -0.967645
Estimated λ             -0.967645
95% CI for λ            (-0.983145, -0.953145)


Factor Information

Factor    Type   Levels  Values
n         Fixed       3  50, 200, 1000
k-level   Fixed       3  1, 2, 3
p-level   Fixed       3  1, 2, 3
qs-level  Fixed       3  0, 1, 2


Analysis of Variance for Transformed Response

Source                           DF   Adj SS    Adj MS  F-Value  P-Value
  n                               2  0.00405  0.002023    91.03    0.000
  k-level                         2  0.00145  0.000723    32.54    0.000
  p-level                         2  0.00006  0.000030     1.36    0.258
  qs-level                        2  0.02798  0.013992   629.68    0.000
  n*k-level                       4  0.00106  0.000265    11.93    0.000
  n*p-level                       4  0.00032  0.000080     3.62    0.006
  n*qs-level                      4  0.00101  0.000253    11.40    0.000
  k-level*p-level                 4  0.00018  0.000044     1.99    0.093
  k-level*qs-level                4  0.00339  0.000847    38.11    0.000
  p-level*qs-level                4  0.00031  0.000077     3.45    0.008
  n*k-level*p-level               8  0.00117  0.000146     6.58    0.000
  n*k-level*qs-level              8  0.00197  0.000247    11.11    0.000
  n*p-level*qs-level              8  0.00024  0.000030     1.35    0.211
  k-level*p-level*qs-level        8  0.00022  0.000028     1.25    0.265
  n*k-level*p-level*qs-level     16  0.00039  0.000025     1.11    0.339
Error                         41874  0.93049  0.000022
Total                         41954  1.06840


Model Summary for Transformed Response

        S    R-sq  R-sq(adj)  R-sq(pred)
0.0047139  12.91%     12.74%      12.62%
```

Figure 5.10. General ANOVA results for significance of factor impacts on negotiation rounds required to reach consensus on Watts-Strogatz model random networks.

General Linear Model: ticks versus n, m-level, qs-level

```
Method

Factor coding           (-1, 0, +1)

Box-Cox transformation
Rounded λ               -1.28924
Estimated λ             -1.28924
95% CI for λ            (-1.30174, -1.27674)


Factor Information

Factor    Type   Levels  Values
n         Fixed       3  50, 200, 1000
m-level   Fixed       3  1, 2, 3
qs-level  Fixed       3  0, 1, 2


Analysis of Variance for Transformed Response

Source                   DF    Adj SS    Adj MS  F-Value  P-Value
  n                       2  0.000477  0.000239   184.46    0.000
  m-level                 2  0.000004  0.000002     1.41    0.244
  qs-level                2  0.001605  0.000803   620.11    0.000
  n*m-level               4  0.000012  0.000003     2.34    0.053
  n*qs-level              4  0.000472  0.000118    91.18    0.000
  m-level*qs-level        4  0.000026  0.000006     4.98    0.001
  n*m-level*qs-level      8  0.000033  0.000004     3.16    0.001
Error                 95197  0.123199  0.000001
Total                 95223  0.149644


Model Summary for Transformed Response

        S    R-sq  R-sq(adj)  R-sq(pred)
0.0011376  17.67%     17.65%      17.62%
```

Figure 5.11. General ANOVA results for significance of factor impacts on negotiation rounds required to reach consensus on Barabási-Albert model random networks.

General Linear Model: ticks versus n, k-level, qs-level

```
Method

Factor coding           (-1, 0, +1)

Box-Cox transformation
Rounded λ               -1.11988
Estimated λ             -1.11988
95% CI for λ            (-1.16838, -1.07138)


Factor Information

Factor    Type   Levels  Values
n         Fixed       3  50, 200, 1000
k-level   Fixed       3  1, 2, 3
qs-level  Fixed       3  0, 1, 2


Analysis of Variance for Transformed Response

Source                  DF    Adj SS    Adj MS  F-Value  P-Value
  n                      2  0.000974  0.000487    91.74    0.000
  k-level                2  0.000001  0.000000     0.07    0.937
  qs-level               2  0.001841  0.000920   173.32    0.000
  n*k-level              4  0.000169  0.000042     7.95    0.000
  n*qs-level             4  0.000771  0.000193    36.32    0.000
  k-level*qs-level       4  0.000245  0.000061    11.56    0.000
  n*k-level*qs-level     8  0.000156  0.000020     3.68    0.000
Error                 4778  0.025370  0.000005
Total                 4804  0.031577


Model Summary for Transformed Response

        S    R-sq  R-sq(adj)  R-sq(pred)
0.0023043  19.66%     19.22%      18.81%
```

Figure 5.12. General ANOVA results for significance of factor impacts on negotiation rounds required to reach consensus on Erdős-Rényi model random networks.

5.4. Impact of the Social Network Model

Each ANOVA in the previous section combined different sizes of social networks created by the same model. To determine if the social network model, itself, has any significant impact on the metaheuristic, results for different social network models of the same size were combined for ANOVA. These results are presented in Figures 5.13, 5.14, and 5.15, with the full ANOVA results presented in Appendix B.

General Linear Model: ticks versus model, qs-level

```
Method

Factor coding           (-1, 0, +1)

Box-Cox transformation
Rounded λ               -1.16755
Estimated λ             -1.16755
95% CI for λ            (-1.24005, -1.09705)


Factor Information

Factor    Type   Levels  Values
model     Fixed       3  1, 2, 3
qs-level  Fixed       3  0, 1, 2


Analysis of Variance for Transformed Response

Source              DF    Adj SS    Adj MS  F-Value  P-Value
  model              2  0.000030  0.000015     3.14    0.044
  qs-level           2  0.001920  0.000960   197.81    0.000
  model*qs-level     4  0.000072  0.000018     3.69    0.005
Error             2520  0.012228  0.000005
Total             2528  0.014544


Model Summary for Transformed Response

        S    R-sq  R-sq(adj)  R-sq(pred)
0.0022028  15.92%     15.65%      15.29%
```

Figure 5.13. General ANOVA results for significance of social network model impact on negotiation rounds required to reach consensus for social networks of population size 50.

General Linear Model: ticks versus model, qs-level

```
Method

Factor coding           (-1, 0, +1)

Box-Cox transformation
Rounded λ               -1.21023
Estimated λ             -1.21023
95% CI for λ            (-1.22073, -1.19873)


Factor Information

Factor    Type   Levels  Values
model     Fixed       3  1, 2, 3
qs-level  Fixed       3  0, 1, 2


Analysis of Variance for Transformed Response

Source                DF    Adj SS    Adj MS  F-Value  P-Value
  model                2  0.000592  0.000296   111.18    0.000
  qs-level             2  0.011365  0.005683  2133.79    0.000
  model*qs-level       4  0.007378  0.001844   692.56    0.000
Error             115065  0.306436  0.000003
Total             115073  0.362373


Model Summary for Transformed Response

        S    R-sq  R-sq(adj)  R-sq(pred)
0.0016319  15.44%     15.43%      15.42%
```

Figure 5.14. General ANOVA results for significance of social network model impact on negotiation rounds required to reach consensus for social networks of population size 200.

General Linear Model: ticks versus model, qs-level

```
Method

Factor coding           (-1, 0, +1)

Box-Cox transformation
Rounded λ               -1.65271
Estimated λ             -1.65271
95% CI for λ            (*, *)


Factor Information

Factor    Type   Levels  Values
model     Fixed       3  1, 2, 3
qs-level  Fixed       3  0, 1, 2


Analysis of Variance for Transformed Response

Source              DF    Adj SS    Adj MS  F-Value  P-Value
  model              2  0.000005  0.000002    52.93    0.000
  qs-level           2  0.000055  0.000028   634.03    0.000
  model*qs-level     4  0.000012  0.000003    68.40    0.000
Error             8414  0.000365  0.000000
Total             8422  0.000463


Model Summary for Transformed Response

        S    R-sq  R-sq(adj)  R-sq(pred)
0.0002084  21.09%     21.02%      20.94%
```

Figure 5.15. General ANOVA results for significance of social network model impact on negotiation rounds required to reach consensus for social networks of population size 1,000.

These results show that the social network model does, in addition to quorum size, have a significant impact on the number of negotiation rounds required to reach consensus, and this significance increases as the population size increases; however, as shown in Figure 5.16, the model that produces the effect and the quorum size levels at which the effects are realized are not consistent for differing population sizes. For population sizes of 1,000 agents, the Erdős-Rényi model at low and high quorum sizes has the largest impact, whereas for population size 200, Watts-Strogatz and Erdős-Rényi have a large effect at low quorum sizes and Barabási-Albert has a large effect at high quorum sizes. At population size 50, none of the models appear significantly better than their peers with the exception

of slightly worse performance by the Watts-Strogatz model at high quorum sizes. Precise reasons for these differences require additional experimentation and are proposed as future work; however, it is hypothesized that small population sizes do not allow significant differentiation between the models in cluster and influential hub formation. As the population size increases the effects of clusters and hubs become more influential, resulting in random networks outperforming those with highly clustered and scale-free topologies. These cluster and hub effects are moderated by increased quorum sizes, effectively reducing their influence and causing the networks to behave more like those based on the Erdős-Rényi model. One trend that is clear is that medium and high quorum sizes both perform significantly better than low quorum sizes across all models and population sizes.

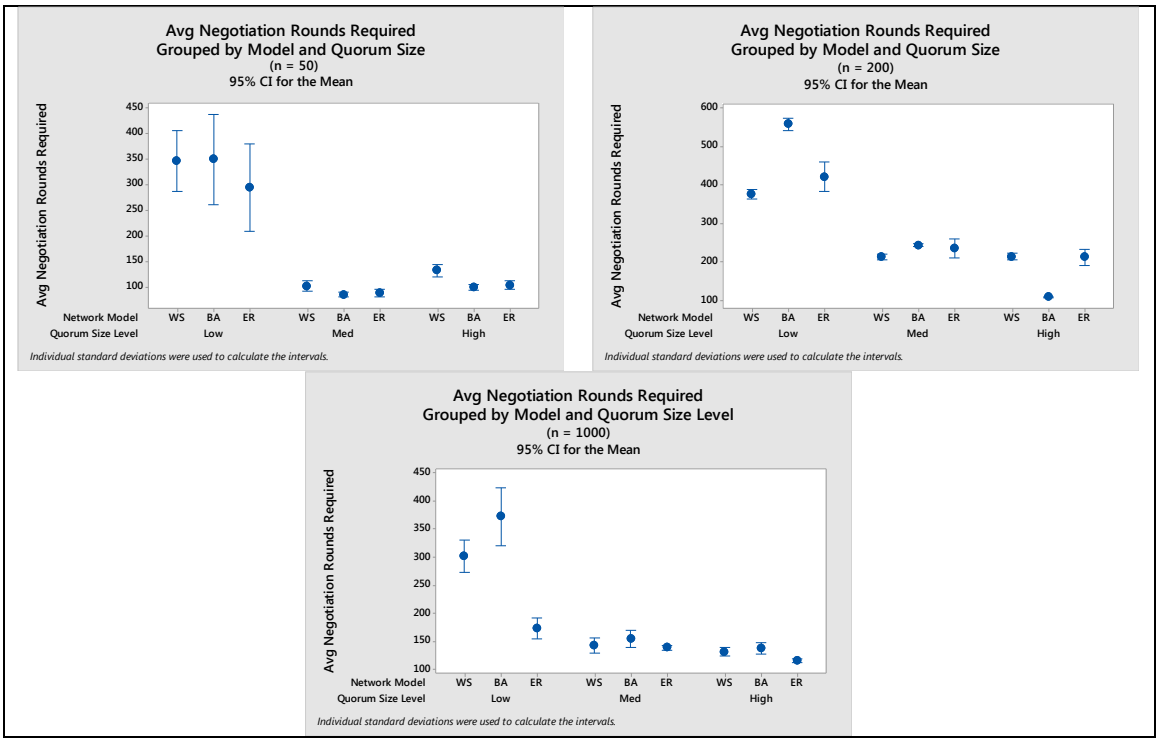


Figure 5.16. Model and quorum size level effect comparisons for different sizes of populations.

5.5. Comparisons of Performance on Two Decision Values

A key contribution of HBC over existing techniques is its ability to facilitate distributed consensus negotiation over more than two possible decision values. Nevertheless, examination of HBC's ability to correctly handle two-choice consensus provides further validation of the metaheuristic. Here, three existing non-metaheuristic techniques that can only accommodate consensus from among two possible outcomes are compared to HBC on the basis of expected running time. The first existing technique is proposed by Tan (2010) as a solution to the Networked Biased Voter Problem. The remaining two are proposed by Mossel and Schoenebeck (2010) as solutions to the Majority Coordination Problem.

Tan's solution consists of a nested iterative solution in which the iterative classic voter model (Clifford & Sudbury, 1973; Holley & Liggett, 1975), which is expected to reach *some* consensus in a finite amount of time, is repeated a number of times sufficient to expect that the result of the majority of the iterations is the desired majority outcome with high, user-selectable, probability. Tan shows that this nested iterative solution runs in expected time $O(n^7)$ (Mossel & Schoenebeck, 2010; Tan, 2010). While this solution is in polynomial time, the high degree of the polynomial quickly makes this technique infeasible as population size increases.

Mossel and Schoenebeck propose two techniques with running times faster than Tan's solution, although they rely on the availability of global knowledge and the ability to propagate selected information through the social network. Mossel and Schoenebeck's Strong Weak model reaches majority consensus in expected time $O(n^3)$. Their modified Wait-And-See model runs in expected time, related to the diameter *d* of the social network, of $O(d + \log(n))$; however, the largest possible diameter for a connected network is $n - 1$ when the nodes are connected in a straight line, giving a worst-case time complexity of $O(n)$. For the faster Wait-And-See model it is noted that it may not always converge to a consensus, in which cases the model should be restarted (Mossel & Schoenebeck, 2010). The number of restarts that may be required to achieve successful consensus is not specified.

In comparison to the previous techniques, HBC runs in expected time $O(n^2)$, which is faster than Tan's solution and Mossel and Schoenebeck's Strong Weak model, but slower than Mossel and Schoenebeck's Wait-And-See model.

Proof. The maximum number of edges in a simple, connected graph is $\frac{n(n-1)}{2}$, where n is the number of vertices in the graph, or, in this case, agents in the social network. For every negotiation round, each agent polls each of its neighbors for information, therefore each edge in the social network is polled twice per negotiation round. Thus, the maximum possible number of edge communications per round of negotiations for any social network is $n(n-1)$.

The number of negotiation rounds is bounded by a predetermined constant, c, giving a total run time of $cn(n-1)$ for $c \ll n$, which is $O(n^2)$. ■

In practice, most social networks will not be fully-connected, so the expected runtime is more precisely stated in terms of the number of edges in the network as $\theta(|\boldsymbol{E}|)$.

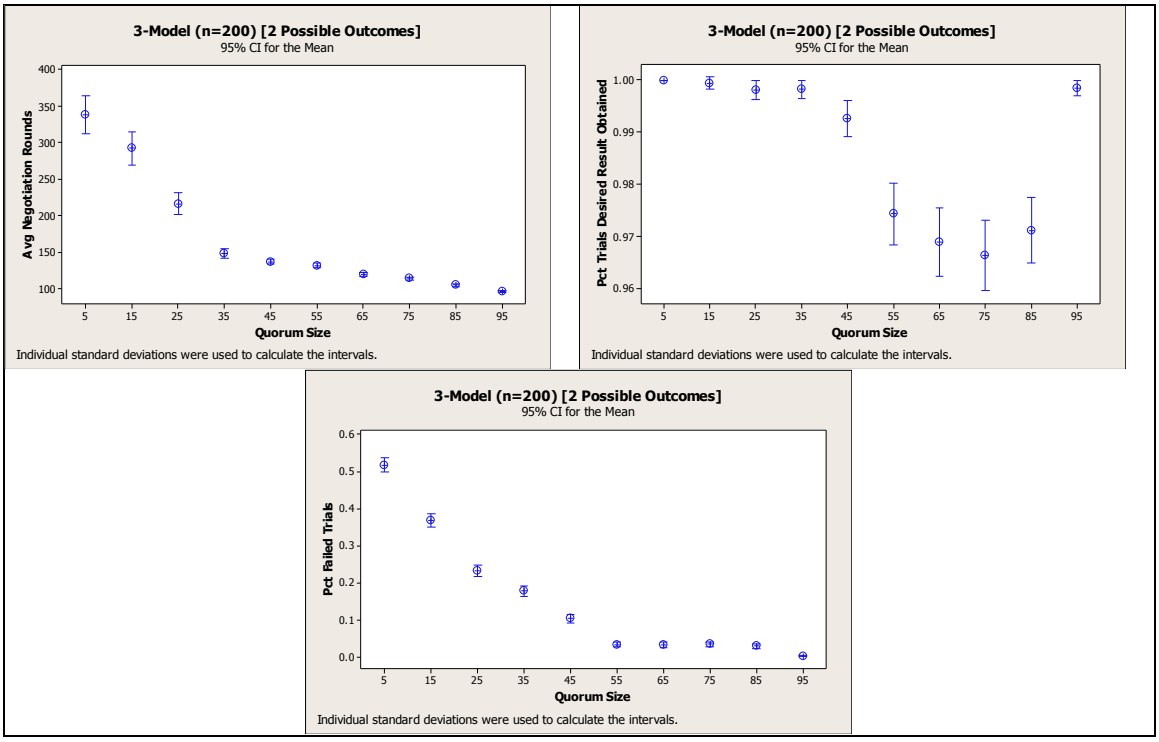


Figure 5.17. Consolidated performance metric plots for all three social network models of 200 agents on two possible outcomes.

Figure 5.17 shows results from empirical tests of HBC performance over two potential outcomes for 200 agents. It should be noted, as described in Appendix A, for these trials the social network model used for the Erdős-Rényi random networks was the $G(n, p)$ variant. This variation was chosen in order to obtain an even wider range of random topologies for study and to ensure that the empirical results presented here were comprehensive in their depiction of HBC performance on a variety of network models.

Despite the smaller number of possible outcomes and the Erdős-Rényi model variation, the results are substantially similar to those previously presented for population size 200, although the percentage of trials in which the desired result is obtained, given that the metaheuristic successfully terminates in the allotted time, demonstrates a more distinct phase transition point than what was observed with five possible outcomes. This is a result more similar to that observed in population sizes of 1,000.

5.6. Scalability to Larger Numbers of Decision Values

The results presented thus far have demonstrated HBC performance for various population sizes connected by random networks generated using three different social network models and seeking consensus on one of five possible outcomes. For population sizes of 200, performance has been further explored on the three social network models when seeking consensus on one of two possible outcomes. In all of these trials, it has been shown that in the tradeoffs between speed to consensus, desirability of achieved consensus, and the rate of failure to reach the desired, or even any, consensus is contingent in large part upon the chosen quorum size, with high quorum sizes yielding faster and less failure prone results, and lower quorum sizes yielding more desirable results, so long as a consensus is actually achieved in the allotted time. It has also been shown that there is a quorum size at which a distinct phase transition point can be observed for each network and population combination.

To further determine the strengths of these observed relationships as the number of possible outcomes varies, further experimentation was performed on networks of population size 200 with 10 possible outcomes. The underlying random networks were

identical to those used in the trials on two possible outcomes. Figure 5.18 contains the results of these trials.

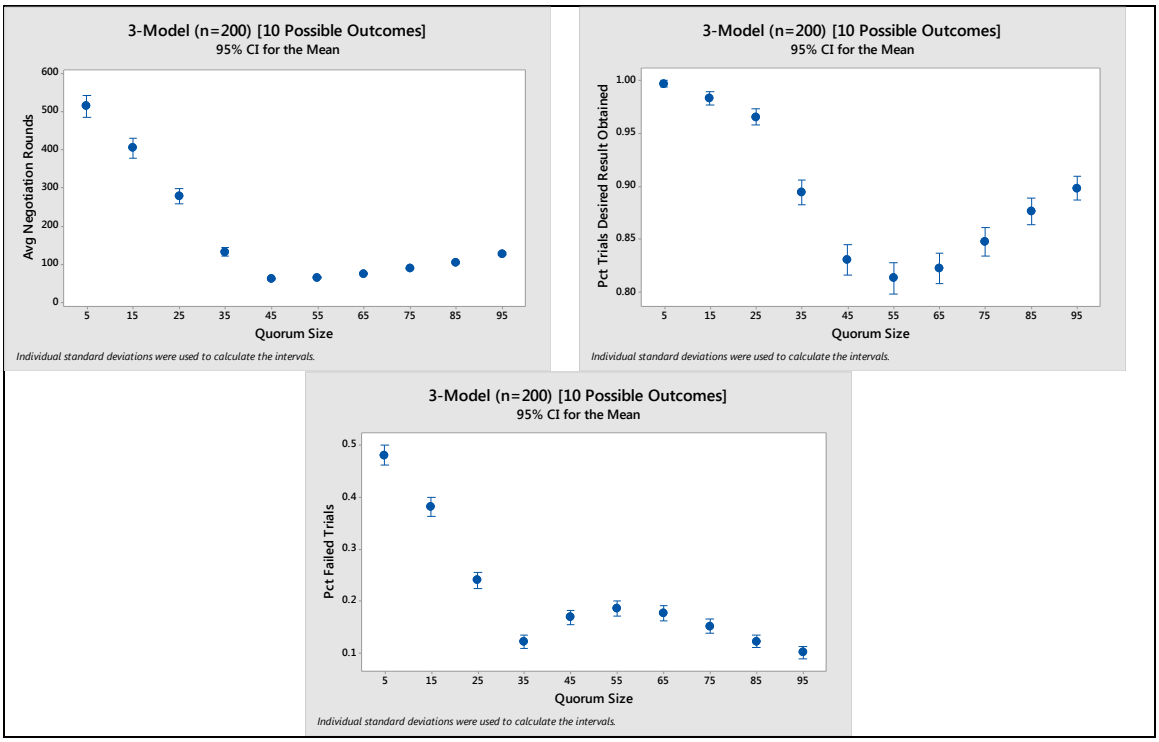


Figure 5.18. Consolidated performance metric plots for all three social network models of 200 agents on ten possible outcomes.

Figure 5.18 demonstrates that HBC performance continues to maintain its observed properties as the number of possible outcomes increases. This third data point also indicates a trend that as the number of possible choices increases, the worst-case percentage of trials reaching a desirable consensus, given that a consensus is reached, does move progressively lower, although low quorum sizes continue to yield highly favorable results by that metric. Similarly, higher failure rates are observed at some quorum sizes, but the general trend of high failure rates for low quorum sizes with a sharp phase transition to more moderate failure rates continues to hold. These results are intuitive as a larger number of possible outcomes to choose from increases the likelihood that the desired outcome will not be the one chosen, however, the performance of HBC continues to be better than that expected by random chance.

5.7. Chapter Summary

This chapter presented both empirical and analytical time complexity results for HBC's performance on three different social network models of varying sizes and possible numbers of outcomes. An analysis of the significance of different independent variable model parameters such as quorum size and social network model was also presented.

The results show that, of the independent variables tested, the choice of quorum size has the most significant impact on the number of negotiation rounds required to reach consensus, the accuracy of the consensus obtained, and the rate of failure to reach any or a desirable consensus. Generally, lower quorum size yields higher accuracy when consensus is reached, but higher quorum sizes are faster and less likely to completely fail to reach any consensus. When an appropriate quorum size is selected, HBC returns a desired consensus more frequently than would be expected by chance.

The social network model is also shown to have a significant impact, especially as population sizes increase. At larger population sizes, random social networks tend to perform better than those with Watts-Strogatz small-world or Barabási-Albert scale-free structures. It is believed that this is due to the absence of overly-influential clusters or hub individuals in Erdős-Rényi model networks as compared to the other two models.

Honey Bee Consensus performs well in consensus tasks seeking the majority consensus of two possible outcomes and, for more than two possible outcomes, the consensus aligned with the majority of three common voting protocols that consider global preference information. When compared to the runtimes of existing solutions for the Networked Biased Voting and Majority Consensus Problems, HBC's worst-case runtime is more favorable than the majority of the compared expected runtimes; in addition, HBC provides the capability of handling more than two possible outcomes.

In the next chapter, these results will be examined and explained. The broader implications and applications of these results will also be presented.

6. Discussion

6.1. Overview

Empirical results have shown that HBC produces distributed consensus results that are significantly better than random, as intended. Nevertheless, HBC is a metaheuristic and cannot claim to be able to guarantee optimal results. Instead, HBC produces results that are expected to be near-optimal with respect to balancing the time required to reach consensus, the consensus success rate, and the consensus accuracy as measured by social utility metrics. As with all metaheuristics, appropriate choice of configuration parameters can have an impact on the effectiveness of the technique.

In the previous chapter, it was shown that the quorum size and social network model parameters have significant impact on the success of the Honey Bee Consensus metaheuristic. Here, these results are further analyzed and explained in order to provide guidance in the application of HBC and the parameter selection of the quorum sensing agent-oriented design pattern, described in Chapter 2. This chapter also discusses the implications of using HBC and quorum sensing on networks conforming to the different models studied.

6.2. Suitability and Limitations of Honey Bee Consensus

While HBC has been shown in these experiments to perform well for large, distributed populations with more than two possible consensus values and where it is impossible or infeasible to centrally collect global vote tallies, it is important to recognize certain limitations of the technique as well. Study and remediation of many of the shortcomings described in this section is proposed as future work.

Given the metaheuristic and probabilistic nature of HBC, it is not appropriate for use when direct calculation methods or those with guaranteed performance bounds can be used if guaranteed results or performance are required. Like other metaheuristic optimization techniques, it is also not particularly well suited to online or hard real-time systems due to its iterative nature and the fact that performance depends on the number of iterations the metaheuristic is allotted.

One of the key contributions of HBC is its ability to accommodate more than two possible consensus outcomes; however HBC effectiveness decreased slightly as possible consensus outcomes increased. This was expected, since a larger set of possible outcomes should result in more diversity of opinion among the population, thus making a desirable consensus harder to negotiate in a distributed setting. Nevertheless, HBC consistently resulted in consensus negotiation of the desired outcome more frequently than would be expected by random chance, and even for the largest number of possible consensus outcomes tested, failure rates remained below 20% for quorum sizes in the intermediate and high ranges. These failure rates decreased as the number of possible consensus outcomes was reduced.

Due to its basis in natural honey bee consensus negotiation, it is not known how extensible HBC is to competitive or game theoretic situations. Honey bees are eusocial insects and consensus on the best nest site location is a goal that is shared by all members of the swarm, thus is it counterproductive for any of the members to selfishly advocate a poor choice or to lie about the quality of a potential nest site. Furthermore, it is sensible to expect that all honey bees have evolved to have essentially the same preferences with regard to potential nest sites. That is to say, two correctly behaving scouts should be expected to have the same preference for a given potential nest site, within some small amount of error or subject to a small degree of data or measurement noise. The artificial, stochastic preferences generated for the trials performed in this research allows for wider variation among individual preferences than might be expected in actual honey bees. Attempts to deliberately subvert consensus were not evaluated at all.

Finally, for HBC to work, stubbornness cannot be allowed. Each member of the population must have some non-zero preference for each of the possible outcomes and all possible outcomes must be known prior to the beginning of negotiations. This also implies a discrete set of possible outcomes from which the population can choose. Real-valued consensus was not explored.

6.3. Optimal Quorum Size

Fernandez-Marquez (2012) identifies the quorum sensing agent-oriented design pattern, but unlike design patterns described in (Gamma et al., 1994), the Fernandez-Marquez paper provides no consequences for the use of the pattern. The absence of this content leaves pattern users uninformed as to how the pattern parameters can be expected to influence the performance results and trade-offs. Also, none of the documentation of existing self-organization patterns thoroughly analyze the systemic behavior via simulation experiments to suggest control strategies to influence behavior. One of the primary goals of this research is to provide guidance in these areas.

Chapter 5 illustrated results showing that the quorum size parameter has a significant impact on consensus speed and success. Plots from the figures in Chapter 5 have been stacked in Figures 6.1a-f to enable comparison of consensus speed and failure rates at each quorum size for all of the population sizes tested. In each of these plots, a phase transition point can be seen at quorum sizes near 25% of the population size, indicated by the vertical lines, where both the average number of negotiation rounds required to reach consensus and the percentage of failure cases approach their lowest values. For population size 50, the phase transitions are clearer and more statistically significant with respect to the number of negotiation rounds required as compared to the failure rate; however, the phase transition point for the failure rate becomes more apparent and significant as population size increases. It is hypothesized that this is because achieving consensus in small populations is intuitively easier than in large populations, so there is less variation in the rate of success in small populations, regardless of the quorum size used.

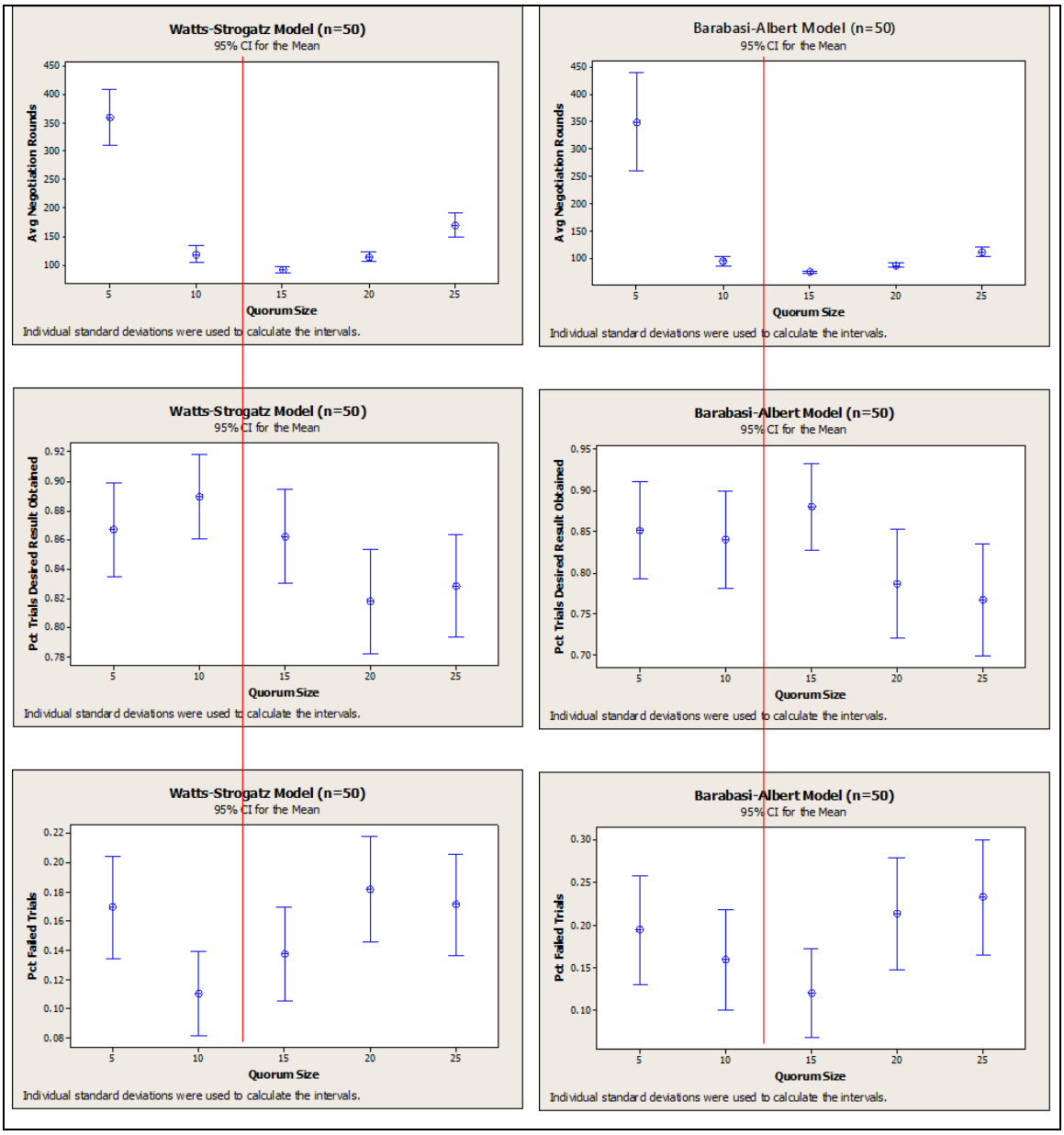


Figure 6.1a. Vertical comparison of quorum size vs average negotiation rounds required and percentage of failed trials. Vertical lines across plots show the point at which the quorum size equals 1/4 of the population size.

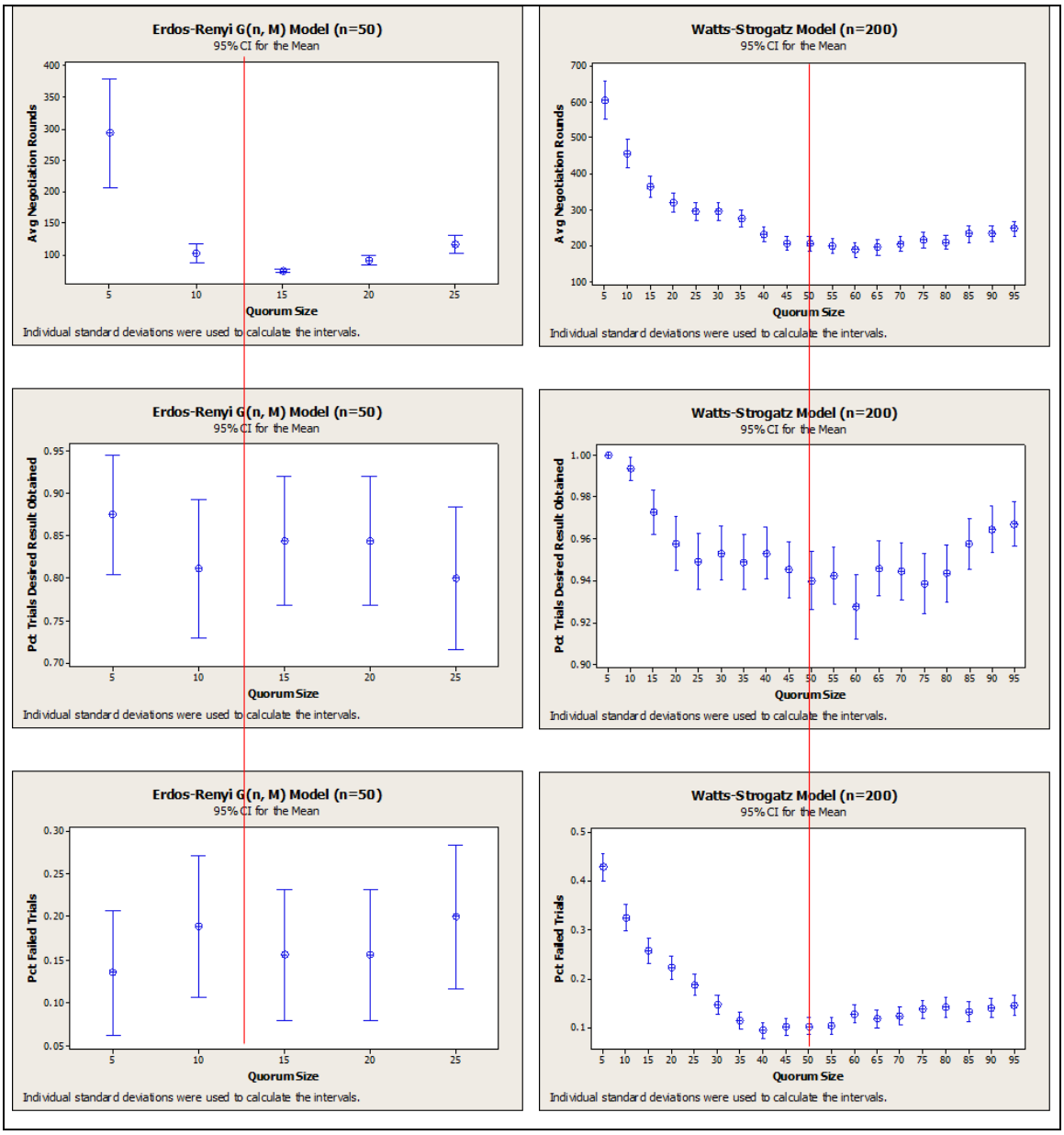


Figure 6.1b. Vertical comparison of quorum size vs average negotiation rounds required and percentage of failed trials. Vertical lines across plots show the point at which the quorum size equals 1/4 of the population size.

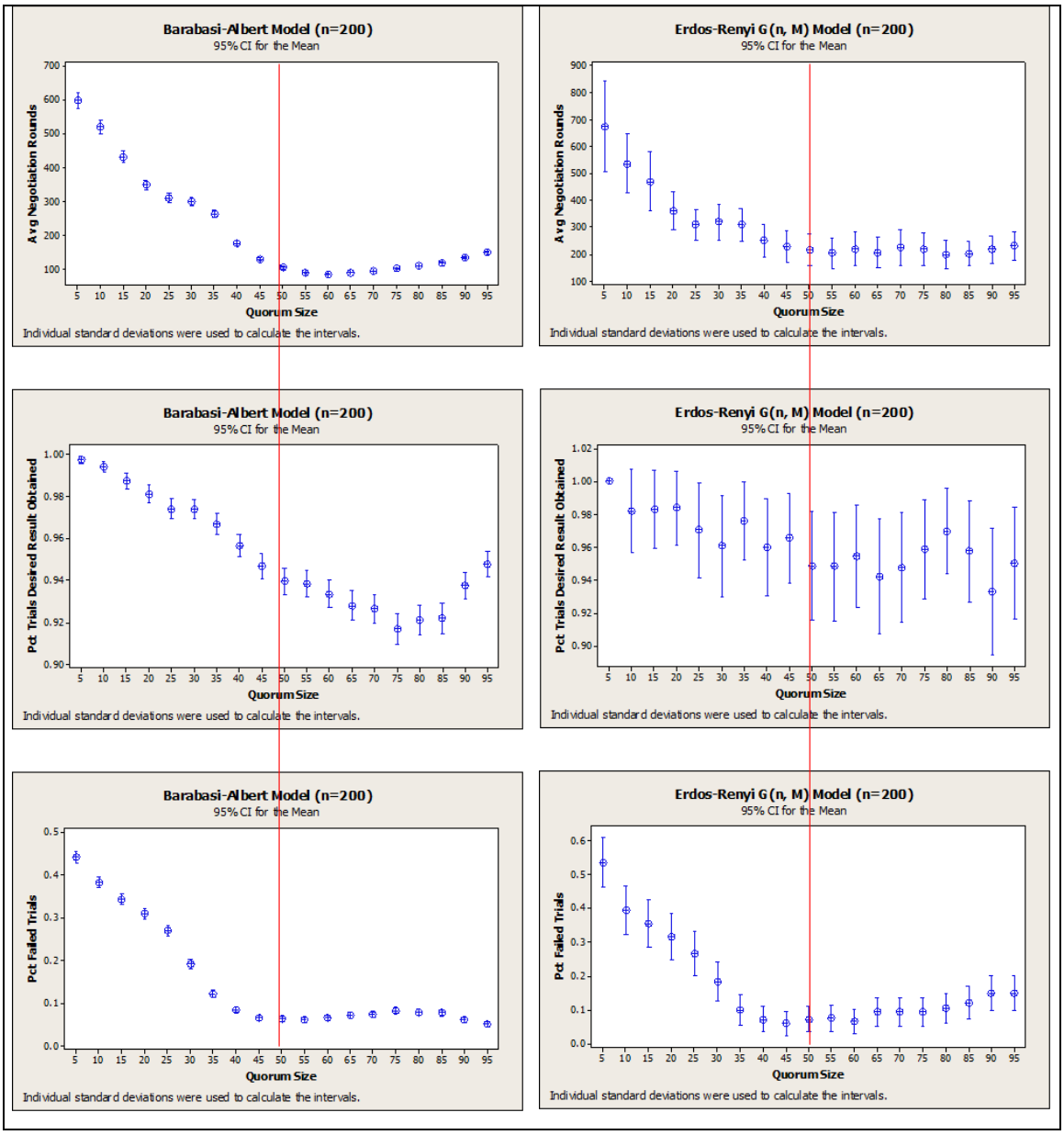


Figure 6.1c. Vertical comparison of quorum size vs average negotiation rounds required and percentage of failed trials. Vertical lines across plots show the point at which the quorum size equals 1/4 of the population size.

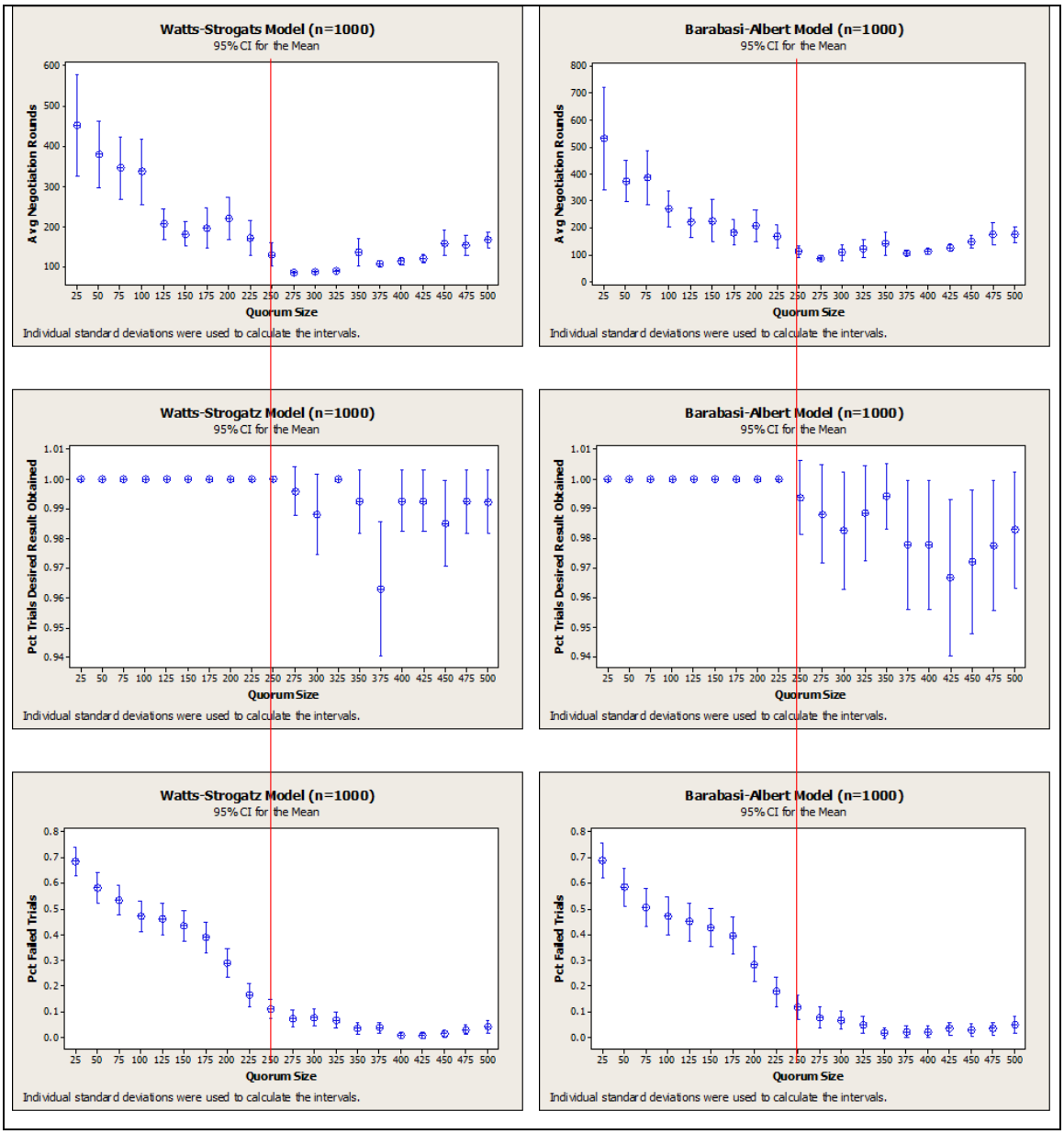


Figure 6.1d. Vertical comparison of quorum size vs average negotiation rounds required and percentage of failed trials. Vertical lines across plots show the point at which the quorum size equals 1/4 of the population size.

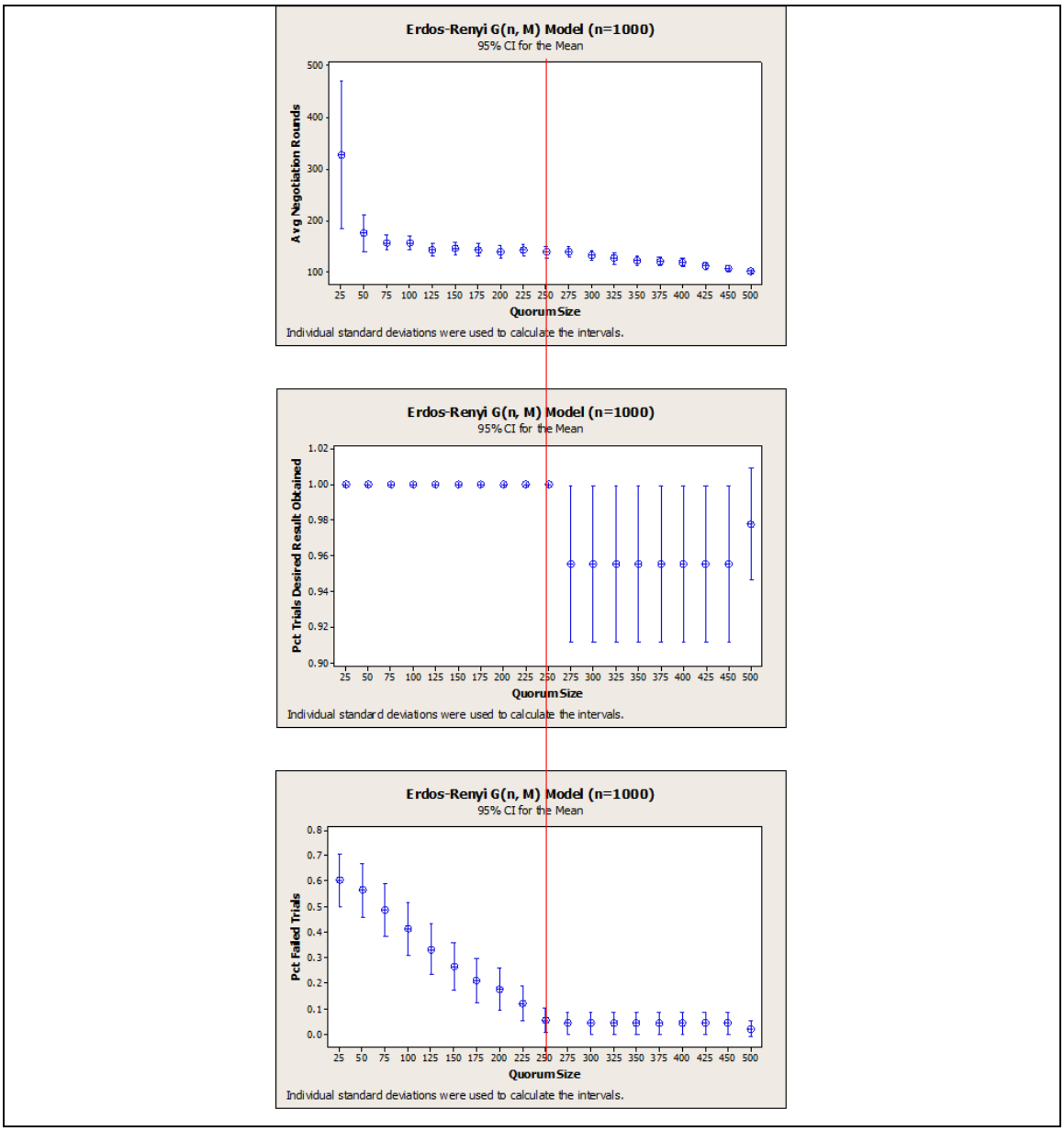


Figure 6.1e. Vertical comparison of quorum size vs average negotiation rounds required and percentage of failed trials. Vertical lines across plots show the point at which the quorum size equals 1/4 of the population size.

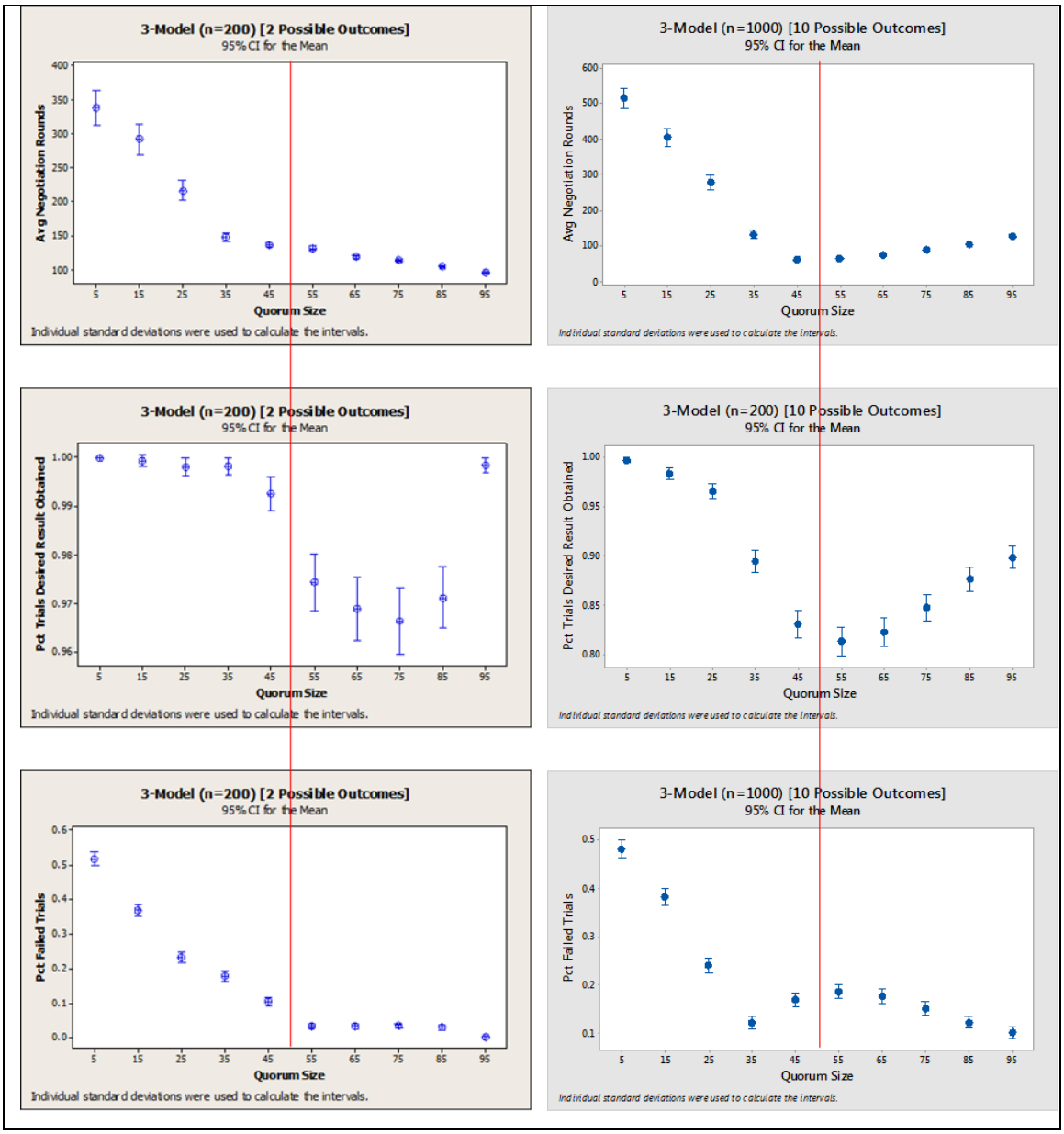


Figure 6.1f. Vertical comparison of quorum size vs average negotiation rounds required and percentage of failed trials. Vertical lines across plots show the point at which the quorum size equals 1/4 of the population size.

It is frequently the case when evaluating a solution with respect to multiple metrics that it is not possible to improve performance relative to one metric without causing the degradation in performance relative to one or more of the other metrics. Such a solution is said to be Pareto optimal, and the set of all possible Pareto optimal solutions constitute the

Pareto front. By choosing a solution from the Pareto front, one can control the balance of performance provided by the solution.

The Pareto optimality of each quorum size, with respect to speed to consensus and failure rate was examined to see if the observed phase transition point was also an optimal value. Figure 6.1a-e depict performance over five possible outcomes. In eight of these plots, the quorum size of 25% of the total population is equal to, or directly adjacent to, a quorum size on the Pareto front for negotiation rounds to consensus versus failure rate. The exception to this trend was the Erdős-Rényi network of size 1,000 where the only quorum size on the Pareto front occurred at 50% of the population size.

Like the 1,000-member Erdős-Rényi network, for the pair of plots combining all three network models of size 200 on two possible outcomes, the only point on the Pareto front occurs near 50% of the population rather than near the phase transition point at 25%. Despite this, the phase transition point at 25% of population size is still nearly optimal for these trials and the plots show favorable performance near that quorum size. Furthermore, increasing the number of possible outcomes to 10 for population size 200 results in quorum sizes on the Pareto front at the quorum size value of 45, 75, 85, and 95 for all social network models. The inclusion of quorum size 45 here continues to reinforce the favorability of quorum sizes near 25% of the population size.

These results, showing a phase transition in speed and accuracy frequently co-incident with Pareto optimality, indicate that 25% of the population size is a good starting point to use as a heuristic for choosing a quorum size with HBC. The frequency of Pareto optimality of this quorum size tended to increase with the number of possible outcomes, while the average number of rounds required to reach consensus continued to remain low with respect to other quorum size options. Furthermore, as the number of possible outcomes increased, more of the quorum size values between 25% and 50% of the population size fell on the Pareto front. This suggests that as the number of possible outcomes increases, the suitability of HBC also increases as it becomes more likely to produce Pareto optimal results.

The explanation for why 25% of the population size is a good guideline for selecting a quorum size follows directly from the premise that the quorum sensing technique is a method for balancing the tradeoff between speed and accuracy of consensus. Definition of a quorum size is a way to specify some amount of required agreement that falls between complete discord and unanimity. Consider the two extremes of a quorum size of one (complete self-determination) and a quorum size of at least 50% of the population (simple plurality).

The smallest quorum size is trivial to obtain since all agents are expected to at least agree with themselves, but it is intuitively obvious that small quorums should be easier to achieve than larger ones, especially in the presence of fairly uniform preference distribution. If an agent finds it easy to form a coalition of others in agreement with itself and sufficient to form a quorum, this results in stubborn behavior between multiple competing quorums that will delay consensus. If the delay is eventually overcome by one of the quorums, then the resulting consensus is likely to be a good one with respect to social utility; however, in a time-constrained environment, the competition between many small quorums is more likely to result in a split decision at the end of the allotted time.

On the other hand, by definition, achieving a quorum size of more than 50% of the population precludes the remainder of the population from achieving a quorum. Forming a quorum of this size is relatively hard, especially for more than two possible consensus outcomes, unless one of the outcomes has an especially large following at the outset of negotiations; therefore, at the beginning of negotiations, in the absence of a clear quorum establishment, agents choose new preferences with probability proportional to the preferences of their immediate neighbors. This eventually leads to the establishment of a critical mass of agreement (exposure threshold) that easily cascades to create a quorum of the requisite size. Unfortunately, this cascade effect can lead to undue marginalization of minority opinions that results in lower social utility.

Quorum sizes of 25% of the population strike the balance between these two extremes. Models of natural honey bee swarms show the same propensity to favor intermediate quorum sizes for similar reasons. In the honey bee models, low quorum sizes

result in split or no decisions and failure; high quorum sizes require more time and communication energy from the bees. Researchers have found that honey bee models perform best with quorum sizes of between 15% and 25%, which is similar to the findings presented in this research (Passino & Seeley, 2006). The natural honey bee's ability to form consensus with quorum sizes even smaller than 25% of the population is likely due to two key differences between actual honey bee swarms and the presented HBC metaheuristic: 1) honey bee swarm social networks are not static, and 2) honey bees have a common, global communications channel. When honey bees detect a quorum at a potential nest site, they generate a "piping" signal at the swarm by rubbing their wings together. This piping can be heard by every other bee, which means that, in this respect, honey bees have a fully-connected social network for communication and access to some form of global information. In contrast, the HBC metaheuristic does not assume a fully-connected social network and expressly forbids any analogous functionality, restricting all communication to immediate neighbors. This enhances the scalability of HBC.

6.4. Social Network Models

As shown in the previous chapter in Figure 5.10, within a given quorum size level of low, medium, or high, HBC frequently performs similarly in speed to consensus, regardless of the social network topology. The performance of Barabási-Albert networks at high quorum size levels in population sizes of 200 and Erdős-Rényi networks at low quorum size levels in populations sizes of 1,000 are the only notable outliers. If the guideline from the previous section is followed, however, and a moderate quorum size of 25% of the total population is used, speed to consensus can be expected to be similar for the population, regardless of the underlying social network model.

This similar performance across network models with different clustering and degree distribution properties shows that quorum sensing serves to mitigate the influence of clusters and hubs found in Watts-Strogatz and Barabási-Albert model networks, as long as communication is not bottlenecked through a cluster or hub. It is trivially possible to construct networks that result in poor consensus performance by forcing all communication

to travel through a single node, for example. A diversity of opinions must be available to each agent in order for them to be able to properly select some alternative outcome when the preference for their first outcome expires.

6.5. Chapter Summary

In this chapter, the experiment results from the previous chapter have been consolidated and analyzed. While, like all metaheuristics, HBC has its limitations, its suitability for purpose has been shown. Performance trends discovered in the experiments of the previous chapter have been explained with respect to quorum size and social network model. A quorum size of 25% of the total population has been shown to produce consensus performance balanced between speed and accuracy, and the application of this knowledge has been proposed for use in engineering the performance of systems that use the quorum sensing pattern and understanding the performance of quorum sensing systems in nature.

7. Conclusion

7.1. Summary of Contributions

The thesis of this dissertation is that a metaheuristic based on the technique used by honey bees to select a new nest site is feasible for engineering distributed consensus negotiation in social networks and yields insight into the influence of the quorum sensing pattern on consensus negotiation in social groups. In support of the thesis, this dissertation presents the Honey Bee Consensus metaheuristic and empirical results on randomized trials that answer the research questions posed in Chapter 1.

In Chapter 3, it is shown how honey bee nest site selection can be implemented as a metaheuristic model for reaching consensus in a distributed, self-organized way. The Honey Bee Consensus agent-based model is formally defined, observed honey bee behaviors are linked directly to agent behaviors in HBC, and metaheuristic aspects of HBC are described. The incorporation of the quorum sensing agent-oriented design pattern and the associated design decisions are explained. Chapter 4 presents the metrics used to judge the success of the metaheuristic in achieving optimal consensus and describes the model validation.

Experiment and time complexity results show that HBC is a suitable and feasible technique for negotiating distributed consensus on more than two decision values in a population with weighted biases. Experiments on randomly generated social networks created using the Watts-Strogatz, Barabási-Albert, and Erdős-Rényi models of population sizes ranging from 50 to 1,000 agents are presented in Chapters 5 and 6. The results of these experiments show that HBC results in predictable performance for consensus negotiation across all social network models and population scales tested. It is shown that the selection of the quorum size parameter has a significant impact on the speed-accuracy tradeoff in consensus negotiation and that intermediate quorum sizes of approximately 25% of the total population size yield results approaching the Pareto front. This same quorum size also moderates the differences in consensus performance in the different social network models; therefore, it is proposed as a heuristic starting point for the general application of the quorum sensing pattern.

Other general lessons in the application of the agent-based quorum sensing pattern have also been revealed in this implementation. In the application of quorum sensing to consensus formation, HBC contributes to the understanding of bio-inspired design patterns as formulated in (Fernandez-Marquez et al., 2012). The pattern relationships presented by Fernandez-Marquez are both validated and challenged by HBC. This research validates Fernandez-Marquez's presentation of quorum sensing as a high level design pattern, fundamentally composed of the basic patterns of evaporation, aggregation, and spreading (i.e., propagation); however, Fernandez-Marquez's hierarchy describes quorum sensing as a higher-level version of the gradient pattern with evaporation as an optional component. This does not reflect the way the quorum sensing pattern has been typically implemented in practice or observed in nature. Honey Bee Consensus applies the composition of aggregation and spreading into the intermediate-level gossip pattern and then derives quorum sensing from the combination of the gossip and evaporation patterns. Since quorum sensing is a mechanism for obtaining consensus, the evaporation pattern is essential to allowing agents to compromise; therefore, the evaporation pattern should be considered a mandatory component of the high level quorum sensing pattern in Fernandez-Marquez's taxonomy of bio-inspired design patterns. This more accurately reflects the ways in which the pattern occurs in natural and artificial systems.

7.2. Extension of Results to Other Applications

In addition to the direct application of HBC to the task of self-organized, distributed consensus negotiation and general guidance for quorum size selection for tuning performance when using the quorum sensing pattern, the results of this research have applicability to the study of mechanisms of consensus formation in social networks and the function of quorums within these mechanisms. This study of strategic interactions in networks encompasses elements of biology, economics, game theory, network science, and computer science.

The effects of quorums in consensus negotiation and strategic interaction can be studied from the perspective of cooperation or competition. In the former case, quorum

sensing has been shown to be a mechanism for biological systems to tune the balance between speed and accuracy of decision making (Nigel R Franks, Dornhaus, Fitzsimmons, & Stevens, 2003; Passino & Seeley, 2006). Some of this research has also demonstrated biological collectives that dynamically modify quorum sizes based on environmental factors in order to achieve a desired tradeoff. Recent research in this area has raised the question of how this balance can be further calibrated to task requirements (Chittka et al., 2009). The research presented in this dissertation, showing a correlation between population sizes and quorum sizes for phase transition points affecting the speed-accuracy tradeoff, can be used to further define the Pareto front along which these optimal values should be found.

With regard to competitive environments, this research has shown that quorum sensing can result in similar performance across social models regardless of clustering, high-degree hubs, or random connectivity. This is in contrast to research on human cooperation presented by Kearns which found consensus formation in systems with biased voting to be easier in preferential attachment networks (Kearns et al., 2009) and concluded that this class of problem was harder to solve on networks with Erdős-Rényi connectivity. For other classes of problems in which clusters and hubs make problem solving more difficult, the results of this research show that quorum sensing allows HBC to overcome the effects of clusters and hubs in Watts-Strogatz and Barabási-Albert networks, respectively. This research did not study intentional manipulation of consensus results. A potential limitation of this technique is its reliance on the eusocial motivation and homogeneous social goals of the participants. Further work is proposed to determine if quorum sensing can be used to counteract purposeful manipulation and, if so, what the optimal quorum size would be required to do so. This will also serve to identify the limits of systems' abilities to self-organize in the presence of conflict.

Finally, observations about the behavioral space of quorum sensing can be formalized to provide run-time guidance for adjusting quorum sizes to steer behavior as context changes. For example, the quorum size can be dynamically adjusted by a self-

adapting system to further optimize the speed-accuracy tradeoff or learning behavior can be incorporated to facilitate repeated consensus formation.

7.3. Future Work

This research presents many avenues for future work to build on its contributions. An area of active research involves distributed consensus formation on non-static social networks. Honey Bee Consensus is directly modeled on a system in which the communication network topology is constantly switching, therefore it is predicted that this problem is an appropriate application for HBC.

The ability for quorum sensing to mitigate the impacts of clusters and hubs in consensus on social networks implies that the incorporation of quorum definition and detection could be of use in facilitating human problem solving on distributed networks in tasks that are inhibited by social networks exhibiting these structures. To determine this, it is proposed as future work to apply HBC on social networks and data sets using real-world connectivity and preferences. Furthermore, distributed social cooperation and differentiation experiments similar to those performed in (Kearns, 2012; Kearns et al., 2009; Kearns, Suri, & Montfort, 2006) could be conducted in which the user interface also provides quorum information or allows a quorum size to be set and or adjusted by the participant. This could provide insight into how humans value quorums in their decision making process.

Considering HBC's basis in a system of cooperating agents, the technique's performance under competitive conditions is an area ripe for exploration. How robust quorum sensing is to malicious manipulation and other game theoretic aspects is an open question and remains to be understood. This exploration also naturally leads to possible extensions of the system to include behavioral learning and self-adaptation to facilitate consensus.

References

1. Amaral, L. A. N., Scala, A., Barthélémy, M., & Stanley, H. E. (2000). Classes of small-world networks. *Proceedings of the National Academy of Sciences, 97*(21), 11149-11152. doi: 10.1073/pnas.200327197

2. Aridor, Y., & Lange, D. B. (1998). *Agent design patterns: elements of agent application design.* Paper presented at the Second international conference on Autonomous agents, Minneapolis, Minnesota.

3. Arrow, K. J. (1950). A difficulty in the concept of social welfare. *The Journal of Political Economy, 58*(4), 328-346.

4. Baarslag, T. H., Koen, & Jonker, C. (2011, 3 May). *Acceptance Conditions in Automated Negotiation.* Paper presented at the 4th International Workshop on Agent-based Complex Automated Negotiations, Taipei, Taiwan.

5. Babaoglu, O., Canright, G., Deutsch, A., Di Caro, G. A., Ducatelle, F., Gambardella, L. M., . . . Urnes, T. (2006). Design patterns from biology for distributed computing. *ACM Transactions on Autonomous and Adaptive Systems, 1*(1), 26-66.

6. Balci, O. (1986). *Credibility assessment of simulation results.* Paper presented at the Proceedings of the 18th conference on Winter simulation.

7. Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *Science, 286*(5439), 509-512.

8. Barborak, M., Dahbura, A., & Malek, M. (1993). The Consensus Problem in Fault-Tolerant Computing. *ACM Comput. Surv., 25*(2), 171-220. doi: 10.1145/152610.152612

9. Bassler, B. L. (1999). How bacteria talk to each other: regulation of gene expression by quorum sensing. *Current Opinion in Microbiology, 2*, 582-587.

10. Bracha, G., & Toueg, S. (1983). *Resilient consensus protocols*. Paper presented at the Proceedings of the second annual ACM symposium on Principles of distributed computing, Montreal, Quebec, Canada.

11. Brams, S. J., & Fishburn, P. C. (2002). Voting procedures. *Handbook of social choice and welfare, 1*, 173-236.

12. Britton, M., & Sacks, L. (2004). *The SECOAS project-Development of a self-organising, wireless sensor network for environmental monitoring*. Paper presented

at the Second International Workshop on Sensor and Actor Network Protocols and Applications, Boston, MA.

13. Chevaleyre, Y., Dunne, P., Endriss, U., Lang, J., Lemaître, M., Maudet, N., . . . Sousa, P. (2006). Issues in multiagent resource allocation. *Informatica, 30*. doi: citeulike-article-id:7202976

14. Chittka, L., Skorupski, P., & Raine, N. E. (2009). Speed–accuracy tradeoffs in animal decision making. *Trends in Ecology & Evolution, 24*(7), 400-407.

15. Clifford, P., & Sudbury, A. (1973). A model for spatial conflict. *Biometrika, 60*(3), 581-588.

16. Conradt, L., & Roper, T. J. (2003). Group decision-making in animals. *Nature, 421*(6919), 155-158. doi: http://www.nature.com/nature/journal/v421/n6919/suppinfo/nature01294_S1.html

17. Conradt, L., & Roper, T. J. (2005). Consensus Decision Making in Animals. *Trends in Ecology & Evolution, 20*(8), 449-456.

18. Couzin, I. D., Ioannou, C. C., Demirel, G., Gross, T., Torney, C. J., Hartnett, A., . . . Leonard, N. E. (2011). Uninformed Individuals Promote Democratic Consensus in Animal Groups. *Science, 334*(6062), 1578-1580. doi: 10.1126/science.1210280

19. Couzin, I. D., & Krause, J. (2003). Self-Organization and Collective Behavior in Vertebrates. *Advances in the Study of Behavior, 32*, 1-75.

20. Couzin, I. D., Krause, J., Franks, N. R., & Levin, S. A. (2005). Effective leadership and decision-making in animal groups on the move. *Nature, 433*(7025), 513-516. doi: http://www.nature.com/nature/journal/v433/n7025/suppinfo/nature03236_S1.html

21. De Wolf, T., & Holvoet, T. (2006). A Catalogue of Decentralised Coordination Mechanisms for Designing Self-Organising Emergent Applications (D. o. C. Science, Trans.): Katholieke Universiteit Leuven.

22. De Wolf, T., & Holvoet, T. (2007). *Design patterns for decentralised coordination in self-organising emergent systems.* Paper presented at the 4th International Conference on Engineering Self-Organising Systems, Hakodate, Japan.

23. Deugo, D., Weiss, M., & Kendall, E. A. (2001). Reusable patterns for agent coordination. In A. Omicini, F. Zambonelli, M. Klusch & R. Tolksdorf (Eds.), *Coordination of Internet Agents: Models, Technologies, and Applications* (pp. 347-368). London, UK: Springer.

24. Doyle, J. (1992). Rationality and its roles in reasoning. *Computational Intelligence, 8*(2), 376-409.

25. Dréo, J., Pétrowski, A., Siarry, P., & Taillard, E. (2006). *Metaheuristics for Hard Optimization: Methods and Case Studies*. Berlin: Springer-Verlag.

26. Durfee, E. H., & Rosenschein, J. S. (1994, 1994). *Distributed Problem Solving and Multi-Agent Systems: Comparisons and Examples.* Paper presented at the Thirteenth International Workshop on Distributed Artificial Intelligence, Olympic Peninsula, WA.

27. Ehtamo, H., Verkama, M., & Hamalainen, R. P. (1999). How to select fair improving directions in a negotiation model over continuous issues. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 29*(1), 26-33. doi: 10.1109/5326.740667

28. Endriss, U., Maudet, N., Sadri, F., & Toni, F. (2006). Negotiating socially optimal allocations of resources. *J. Artif. Int. Res., 25*(1), 315-348.

29. Ephrati, E., & Rosenschein, J. S. (1996). Deriving consensus in multiagent systems. *Artificial Intelligence, 87*(1–2), 21-74. doi: http://dx.doi.org/10.1016/0004-3702(95)00105-0

30. Erdos, P., & Rényi, A. (1959). On Random Graphs I. *Publicationes Mathematicae, 6*, 290-297.

31. Fax, J. A., & Murray, R. M. (2004). Information flow and cooperative control of vehicle formations. *Automatic Control, IEEE Transactions on, 49*(9), 1465-1476. doi: 10.1109/TAC.2004.834433

32. Feldman, J. A., & Sproull, R. F. (1977). Decision theory and artificial intelligence II: The hungry monkey. *Cognitive Science, 1*(2), 158-192.

33. Fernandez-Marquez, J. L., Marzo Serugendo, G., Montagna, S., Viroli, M., & Arcos, J. (2012). Description and composition of bio-inspired design patterns: a complete overview. *Natural Computing*, 1-25. doi: 10.1007/s11047-012-9324-y

34. Ferreira de Araújo Lima, E., Duarte de Lima Machado, P., Abrantes de Figueiredo, J. C., & Sampaio, F. R. (2003). Implementing mobile agent design patterns in the JADE framework. *Search of Innovation, 3*(3).

35. Fischer, M. J., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM, 32*(2), 374-382.

36. Franks, N. R., Dornhaus, A., Fitzsimmons, J. P., & Stevens, M. (2003). Speed versus accuracy in collective decision making. *Proceedings of the Royal Society of London. Series B: Biological Sciences, 270*(1532), 2457-2463.

37. Franks, N. R., Mallon, E. B., Bray, H. E., Hamilton, M. J., & Mischler, T. C. (2003). Strategies for Choosing Between Alternatives With Different Attributes: Exemplified by House-Hunting Ants. *Animal Behaviour, 65*, 215-223.

38. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley.

39. Gardelli, L., Viroli, M., & Omicini, A. (2007). *Design patterns for self-organizing multiagent systems.* Paper presented at the Engineering Emergence in Decentralised Autonomic Systems.

40. Gatti, M. A. d. C., de Lucena, C. J. P., & Garcia, A. F. (2009). A pattern language for self-organizing systems. *Monografias em Ciência da Computação, 16*(9).

41. Gifford, D. K. (1979). *Weighted voting for replicated data*. Paper presented at the Proceedings of the Seventh ACM Symposium on Operating Systems Principles, Pacific Grove, California, United States.

42. Gilbert, E. N. (1959). Random Graphs. 1141-1144. doi: 10.1214/aoms/1177706098

43. Gmytrasiewicz, P. J., Durfee, E. H., & Wehe, D. K. (1991). *A Decision-Theoretic Approach to Coordinating Multi-agent Interactions.* Paper presented at the IJCAI.

44. Grimm, V., Berger, U., Bastiansen, F., Eliassen, S., Ginot, V., Giske, J., . . . Huse, G. (2006). A standard protocol for describing individual-based and agent-based models. *Ecological modelling, 198*(1), 115-126.

45. Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008, August 2008). *Exploring Network Structure, Dynamics, and Function Using NetworkX.* Paper presented at the 7th Python in Science Conference, Pasadena, CA.

46. Heiskanen, P. (1999). Decentralized Method for computing Pareto solutions in multiparty negotiations. *European Journal of Operational Research, 117*, 578-590.

47. Heiskanen, P., Ehtamo, H., & Hämäläinen, R. P. (2001). Constraint proposal method for computing Pareto solutions in multi-party negotiations. *European Journal of Operational Research, 133*(1), 44-61. doi: 10.1016/S0377-2217(00)00179-X

48. Holley, R. A., & Liggett, T. M. (1975). Ergodic theorems for weakly interacting infinite systems and the voter model. *The annals of probability*, 643-663.

49. Holvoet, T., Weyns, D., & Valckenaers, P. (2009, 14-18 Sept. 2009). *Patterns of Delegate MAS.* Paper presented at the Self-Adaptive and Self-Organizing Systems, 2009. SASO '09. Third IEEE International Conference on.

50. Holvoet, T., Weyns, D., & Valckenaers, P. (2010). *Delegate MAS patterns for large-scale distributed coordination and control applications*. Paper presented at the Proceedings of the 15th European Conference on Pattern Languages of Programs, Irsee, Germany.

51. Horvitz, E. J., Breese, J. S., & Henrion, M. (1988). Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning, 2*(3), 247-302. doi: http://dx.doi.org/10.1016/0888-613X(88)90120-X

52. Ito, T., Klein, M., & Hattori, H. (2008). A multi-issue negotiation protocol among agents with nonlinear utility functions. *Multiagent Grid Syst., 4*(1), 67-83.

53. Jacobs, W., & Kiefer, M. (1973). *Robot Decisions Based on Maximizing Utility.* Paper presented at the IJCAI.

54. Jennings, N. R., Faratin, P., Lomuscio, A. R., Parsons, S., Wooldridge, M. J., & Sierra, C. (2001). Automated Negotiation: Prospects, Methods and Challenges. *Group Decision and Negotiation, 10*(2), 199-215. doi: 10.1023/A:1008746126376

55. Kasinger, H., Bauer, B., & Denzinger, J. (2009, April 14-16). *Design pattern for self-organizing emergent systems based on digital infochemicals.* Paper presented at the EASe 2009.

56. Kearns, M. (2012). Experiments in social computation. *Commun. ACM, 55*(10), 56-67. doi: 10.1145/2347736.2347753

57. Kearns, M., Judd, S., Tan, J., & Wortman, J. (2009). Behavioral experiments on biased voting in networks. *Proceedings of the National Academy of Sciences*. doi: 10.1073/pnas.0808147106

58. Kearns, M., Suri, S., & Montfort, N. (2006). An experimental study of the coloring problem on human subject networks. *Science, 313*(5788), 824-827.

59. Kearns, M., & Tan, J. (2008). *Biased Voting and the Democratic Primary Problem.* Paper presented at the 4th International Workshop on Internet and Network Economics.

60. Kendall, E. A., Krishna, P. V. M., Pathak, C. V., & Suresh, C. B. (1998). *Patterns of intelligent and mobile agents*, Minneapolis, Minnesota.

61. Kennedy, J., & Eberhart, R. (1995, Nov/Dec). *Particle swarm optimization.* Paper presented at the IEEE International Conference on Neural Networks 1995.

62. Kirkpatrick, S., Jr., D. G., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671-680.

63. Klein, M., Faratin, P., Sayama, H., & Bar-Yam, Y. (2003). Protocols for negotiating complex contracts. *Intelligent Systems, IEEE, 18*(6), 32-38. doi: 10.1109/MIS.2003.1249167

64. Knoester, D. B., Goldsby, H. J., & McKinley, P. K. (2013). Genetic Variation and the Evolution of Consensus in Digital Organisms. *Evolutionary Computation, IEEE Transactions on, 17*(3), 403-417. doi: 10.1109/TEVC.2012.2201725

65. Knoester, D. B., & McKinley, P. K. (2009, 14-18 Sept. 2009). *Evolution of Probabilistic Consensus in Digital Organisms.* Paper presented at the Self-Adaptive and Self-Organizing Systems, 2009. SASO '09. Third IEEE International Conference on.

66. Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst., 4*(3), 382-401. doi: 10.1145/357172.357176

67. Li, M., Vo, Q. B., & Kowalczyk, R. (2009). *Searching for fair joint gains in agent-based negotiation*. Paper presented at the Autonomous Agents & Multiagent Systems/Agent Theories, Architectures, and Languages.

68. List, C., Elsholtz, C., & Seeley, T. D. (2009). Independence and interdependence in collective decision making: an agent-based model of nest-site choice by honeybee swarms. *Philosophical Transactions of the Royal Society Biological Sciences*(364), 755-762. doi: 10.1098/rstb.2008.0277

69. Lopez-Carmona, M. A., Marsa-Maestre, I., & Klein, M. (2011, 3 May). *Consensus Policy Based Multi-Agent Negotiation.* Paper presented at the 4th International Workshop on Agent-based Complex Automated Negotiations, Taipei, Taiwan.

70. Mentis, A. S., & Yilmaz, L. (2013, 7-10 July). *Adapting a Natural System Simulation Model to a General-Purpose Metaheuristic: Toward Engineering Emergent Distributed Decision-Making.* Paper presented at the Summer Simulation Multi-Conference 2013, Toronto, ON.

71. . Minitab 17 Statistical Software. (2010). State College, PA: Minitab, Inc. Retrieved from www.minitab.com

72. Mossel, E., & Schoenebeck, G. (2010). *Reaching Consensus on Social Networks.* Paper presented at the ICS.

73. Nalebuff, B. J., & Brandenburger, A. (1996). *Co-opetition*: HarperCollinsBusiness.

74. *Network Science*. (2005). The National Academies Press.

75. Niven, J. E. (2012). How Honeybees Break a Decision-Making Deadlock. *Science, 335*(6064), 43-44. doi: 10.1126/science.1216563

76. North, M., Collier, N., Ozik, J., Tatara, E., Macal, C., Bragen, M., & Sydelko, P. (2013). Complex adaptive systems modeling with Repast Simphony. *Complex Adaptive Systems Modeling, 1*(1), 3.

77. North, M. J., & Macal, C. M. (2011, 11-14 Dec. 2011). *Product design patterns for agent-based modeling.* Paper presented at the 2011 Winter Simulation Conference.

78. Pandey, P., & Tripathi, M. (2012). Exploiting Logical Structures to Reduce Quorum Sizes of Replicated Databases. *Advanced Computing: An International Journal, 3*(1), 99-104.

79. Parunak, H. V. D., & Brueckner, S. A. (2011). *Software Engineering for Self-Organizing Systems.* Paper presented at the Agent Oriented Software Engineering.

80. Passino, K. M., & Seeley, T. D. (2006). Modeling and analysis of nest-site selection by honeybee swarms: the speed and accuracy trade-off. *Behavioral Ecology and Sociobiology, 59*(3), 427-442.

81. Pease, M., Shostak, R., & Lamport, L. (1980). Reaching Agreement in the Presence of Faults. *J. ACM, 27*(2), 228-234. doi: 10.1145/322186.322188

82. Peysakhov, M. D., & Regli, W. C. (2005). *Ant inspired server population management in a service based computing environment.* Paper presented at the 2005 IEEE Swarm Intelligence Symposium.

83. Porter, M. A. (2012). Small-world network. *Scholarpedia, 7*, 1739.

84. Pratt, S. C., Mallon, E. B., Sumpter, D. J. T., & Franks, N. R. (2002). Quorum sensing, recruitment, and collective decision-making during colony emigration by the ant *Leptothorax albipennis*. *Behavioral Ecology and Sociobiology, 52*, 117-127.

85. Pratt, S. C., Sumpter, D. J. T., Mallon, E. B., & Franks, N. R. (2005). An agent-based model of collective nest choice by the ant Temnothorax albipennis. *Animal Behaviour, 70*, 1023-1036.

86. Ren, W., Beard, R. W., & Atkins, E. M. (2005, June 8-10). *A survey of consensus problems in multi-agent coordination.* Paper presented at the American Control Conference, Portland, OR.

87. Sargent, R. G. (1996). *Verifying and validating simulation models.* Paper presented at the Simulation Conference, 1996. Proceedings. Winter.

88. Sargent, R. G. (2005). *Verification and validation of simulation models.* Paper presented at the Proceedings of the 37th conference on Winter simulation.

89. Seeley, T. D. (2003). Consensus building during nest-site selection in honey bee swarms: the expiration of dissent. *Behavioral Ecology and Sociobiology, 53*(6), 417-424. doi: 10.1007/s00265-003-0598-z

90. Seeley, T. D., & Buhrman, S. C. (1999). Group decision making in swarms of honey bees. *Behavioral Ecology and Sociobiology, 45*, 19-31.

91. Seeley, T. D., & Visscher, P. K. (2003). Choosing a home: how the scouts in a honey bee swarm perceive the completion of their group decision making. *Behavioral Ecology and Scoiobiology, 54*, 511-520.

92. Seeley, T. D., Visscher, P. K., Schlegel, T., Hogan, P. M., Franks, N. R., & Marshall, J. A. R. (2012). Stop Signals Provide Cross Inhibition in Collective Decision-Making by Honeybee Swarms. *Science, 335*(6064), 108-111. doi: 10.1126/science.1210361

93. Shoham, Y., & Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge: Cambridge University Press.

94. Smith, N. A., & Tromble, R. W. (2004). *Sampling Uniformly from the Unit Simplex*. Department of Computer Science / Center for Language and Speech Processing. Johns Hopkins University. Retrieved from http://www.cs.cmu.edu/~nasmith/papers/smith+tromble.tr04.pdf

95. Smith, R. G. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *Computers, IEEE Transactions on, C-29*(12), 1104-1113. doi: 10.1109/TC.1980.1675516

96. Snyder, P., Valetta, G., Fernandez-Marquez, J. L., & Serugendo, G. D. M. (2012). Designing self-organizing software with a design pattern catalog: a case study (D. o. C. Science, Trans.): Drexel University.

97. Strogatz, S. H. (2001). Exploring complex networks. *Nature, 410*(6825), 268-276.

98. Sumpter, D. J. T., & Pratt, S. C. (2009). Quorum responses and consensus decision making. *Philosophical Transactions of the Royal Society Biological Sciences, 364*, 743-753.

99. Tan, J. (2010). *Strategic and Secure Interactions in Networks.* (Doctor of Philosophy Dissertation), University of Pennsylvania. Retrieved from http://repository.upenn.edu/edissertations/287 (287)

100. Taylor, A. D. (2008). *Mathematics and politics: strategy, voting, power, and proof* (2nd ed.): Springer.

101. Thomas, R. H. (1979). A Majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst., 4*(2), 180-209. doi: 10.1145/320071.320076

102. Visscher, P. K., & Camazine, S. (1999). Collective decisions and cognition in bees. *Nature, 397*(6718), 400-400.

103. Vo, Q. B., Padgham, L., & Cavedon, L. (2007). Negotiating flexible agreements by combining distributive and integrative negotiation. *Intelligent Decision Technologies, 1*(1-2), 33-47.

104. Wang, X., Xiao, N., Wongpiromsarn, T., Xie, L., Frazzoli, E., & Rus, D. (2013, 12-14 June 2013). *Distributed consensus in noncooperative congestion games: An application to road pricing.* Paper presented at the Control and Automation (ICCA), 2013 10th IEEE International Conference on.

105. Ward, A. J. W., Sumpter, D. J. T., Couzin, I. D., Hart, P. J. B., & Krause, J. (2008). Quorum decision-making facilitates information transfer in fish shoals. *Proceedings of the National Academy of Sciences, 105*(19), 6948-6953. doi: 10.1073/pnas.0710344105

106. Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of `small-world' networks. *Nature, 393*(6684), 440-442.

107. West, D. B. (2001). *Introduction to graph theory* (Vol. 2): Prentice hall Upper Saddle River.

108. Wokoma, I., Sacks, L., & Marshall, I. (2003, 8-9 September). *Clustering in sensor networks using quorum sensing.* Paper presented at the London Communications Symposium, University College London.

109. Wooldridge, M. (2009). *An Introduction to MultiAgent Systems* (2nd ed.). United Kingdom: Wiley.

110. Yanbin, L., & Yang, Y. R. (2003, 20-20 March 2003). *Reputation propagation and agreement in mobile ad-hoc networks.* Paper presented at the Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE.

111. Yilmaz, L. (2006). Validation and verification of social processes within agent-based computational organization models. *Computational & Mathematical Organization Theory, 12*(4), 283-312.

112. Ziang, Z., & Mo-Yuen, C. (2011, 24-29 July 2011). *Incremental cost consensus algorithm in a smart grid environment.* Paper presented at the Power and Energy Society General Meeting, 2011 IEEE.

113. Ziang, Z., Xichun, Y., & Mo-Yuen, C. (2011, 4-6 Aug. 2011). *Decentralizing the economic dispatch problem using a two-level incremental cost consensus algorithm in a smart grid environment.* Paper presented at the North American Power Symposium (NAPS), 2011.

Appendix A. Social Network Experiment Parameter Configurations and Range Definitions

A.1. Overview

This appendix contains tables showing the random generation parameters used for each social network model in the experiments and the number of random graphs created and tested for each parameter configuration. It also contains tables defining the limits of the low, medium, and high parameter value ranges for those configuration parameters used for the ANOVA analyses.

A.2. Social Network Model Generation

For most configurations, 30 random trials were executed on one instance of each configuration, but there are some exceptions. For instance, in early experiments with the Barabási-Albert model with 200 agents, 30 random trials were conducted on 30 random social networks created with the same parameters, but the technique of experimenting with common random numbers was used for the remaining experiments due to the infeasibility of conducting 900 trials for every tested network configuration, especially for large n. Similarly, when it was determined that p values greater than or equal to 0.1 for our Watts-Strogatz graphs produced nearly identical performance, much smaller p values were selected, two of which (0.1 and 0.5) overlapped with previously collected data. Nevertheless, the data from these additional runs was kept and analyzed in the final analysis for the sake of completeness.

Table A.1. Parameter configurations used for scale-free networks.

Barabási-Albert Model NetworkX function: barabasi_albert_graph(n, m)		
n	*m*	number of random graphs tested
50	2	1
50	5	1
50	12	1
50	18	1
50	25	1
200	2	30
200	6	30
200	10	30
200	20	30
200	60	30
200	100	30
1000	15	1
1000	30	1
1000	60	1
1000	250	1
1000	500	1

Table A.2. Parameter configurations used for Erdős-Rényi G(*n*, *M*=*k*) random networks.

Erdős-Rényi Model NetworkX function: watts_strogatz_graph(n, k, p=1)			
n	*k*	*p*	number of random graphs tested
50	5	1	1
50	10	1	1
50	25	1	1
200	2	1	1
200	6	1	1
200	10	1	1
200	20	1	1
200	60	1	1
200	100	1	1
1000	100	1	1
1000	200	1	1
1000	500	1	1

Table A.3. Parameter configurations used for small-world networks.

Watts-Strogatz Model NetworkX function: watts_strogatz_graph(n, k, p)			
n	*k*	*p*	number of random graphs tested
50	5	0.0078125	1
50	5	0.0625	1
50	5	0.5	1
50	10	0.0078125	1
50	10	0.0625	1
50	10	0.5	1
50	25	0.0078125	1
50	25	0.0625	1
50	25	0.5	1
200	2	0.03125	1
200	2	0.0625	1
200	2	0.125	1
200	2	0.25	1
200	2	0.5	1
200	2	0.75	1
200	6	0.0078125	1
200	6	0.00390625	1
200	6	0.015625	1
200	6	0.03125	1
200	6	0.0625	1
200	6	0.125	1
200	6	0.25	1
200	10	0.00390625	1
200	10	0.0078125	1
200	10	0.015625	1
200	10	0.03125	1
200	10	0.0625	1
200	10	0.125	1
200	20	0.005	1
200	20	0.006	1
200	20	0.015	1
200	20	0.02	1
200	20	0.1	2
200	20	0.2	1
200	20	0.3	1
200	20	0.4	1
200	20	0.5	2

200	20	0.6	1
200	20	0.7	1
200	20	0.8	1
200	20	0.9	1
200	60	0.03137178632607688	1
200	60	0.0625	1
200	60	0.125	1
200	60	0.875	1
200	100	0.027448734302918387	1
200	100	0.10073719442448524	1
200	100	0.5	1
1000	100	0.0078125	1
1000	100	0.0625	1
1000	100	0.5	1
1000	200	0.0078125	1
1000	200	0.0625	1
1000	200	0.5	1
1000	500	0.0078125	1
1000	500	0.0625	1
1000	500	0.5	1

A.3. Social Network Model Parameters for 2- and 10-Choice Trials

The trials testing the scalability of HBC to 2 or 10 potential outcomes were performed exclusively on social networks of size 200. The same set of 90 graphs, encompassing 30 random graphs for each of the three social network models, were used for both the 2- and 10-choice trials. Tables 4 through 6 show the parameters used for each of the random networks generated. Values for the parameters were chosen uniformly at random from the parameter ranges defined in the following section. In contrast to the Erdős-Rényi variant used for the other trials, the set of Erdős-Rényi networks used in these trials were constructed with the $G(n, p)$ variant of the model.

Table A.4. Watts-Strogatz model parameters for 200-agent graphs used in 2- and 10-choice trials.

Graph #	*k*	*p*
0	44	0.38497829157803887
1	44	0.44763462653321456
2	76	0.3859939619018211
3	90	0.17150176131261377
4	20	0.4183793975742383
5	35	0.07575130829172926
6	89	0.45455589109109146
7	63	0.15235876349358535
8	86	0.30366233019085853
9	76	0.19658322899815206
10	47	0.37261657195993153
11	82	0.030825740438274345
12	53	0.26962163785748794
13	35	0.06042612329286466
14	69	0.29863359482217133
15	94	0.01806977623964902
16	83	0.08164703692782228
17	39	0.16066764059030406
18	91	0.1987853432074423
19	20	0.047765030122363744
20	58	0.4575071943558304
21	35	0.37645377856781204
22	22	0.18176686714916224
23	93	0.43074981763931675
24	25	0.2619261805016232
25	100	0.4758434144523738
26	64	0.14606652703264314
27	35	0.24925853343142873
28	99	0.4666190257567396
29	16	0.06549377631849304

Table A.5. Barabási-Albert model parameters for 200-agent graphs used in 2- and 10-choice trials.

Graph #	*m*
0	38
1	42
2	74
3	39
4	96
5	51
6	8
7	68
8	69
9	25
10	73
11	100
12	34
13	50
14	59
15	94
16	85
17	57
18	11
19	88
20	3
21	61
22	67
23	92
24	90
25	53
26	21
27	95
28	53
29	1

Table A.6. Erdős-Rényi $G(n, p)$ model parameters for 200-agent graphs used in 2- and 10-choice trials.

Graph #	p
0	0.028477379829144896
1	0.06405580534577897
2	0.2481078277627287
3	0.34033084213930204
4	0.39508533480356195
5	0.07196160398648607
6	0.061661889956799554
7	0.24727375206658633
8	0.09507741400713546
9	0.09397679320043578
10	0.2444471957698016
11	0.4291653947704135
12	0.19900851681787854
13	0.1972812395964747
14	0.0676346429223773
15	0.2646675293142114
16	0.06846907989536011
17	0.1748315506854984
18	0.2424329527813107
19	0.37547030510859386
20	0.2303369560790458
21	0.3448361463570411
22	0.4215499246298953
23	0.3200531705339942
24	0.24577267580673257
25	0.2431430413528492
26	0.43023762535722543
27	0.32977240839737193
28	0.34396442460139137
29	0.4668148031836409

A.4. ANOVA Analysis Parameter Ranges

ANOVA analyses were conducted to test metaheuristic sensitivity to social network model and metaheuristic parameters. For each test configuration in Tables A.1, A.2, and A.3, social network model and quorum size parameters were assigned to a range of low,

medium, or high in accordance with Tables A.4, A.5, and A.6. In each table, *n* is the population size of the social network.

Table A.7. Definition of low, medium, and high ranges for *m* and quorum size parameters in Barabási-Albert scale-free network ANOVA analysis.

	Low	Medium	High
m	$\leq 0.0625 \cdot n$	$\leq 0.25 \cdot n$	$\leq 0.5 \cdot n$
quorum size	$0.1 \cdot n$	$0.3 \cdot n$	$0.5 \cdot n$

Table A.8. Definition of low, medium, and high ranges for *p*, *k*, and quorum size parameters in Watts-Strogatz small-world network ANOVA analysis.

	Low	Medium	High
p	≤ 0.03515625	≤ 0.28125	> 0.28125
k	$\leq 0.1 \cdot n$	$\leq 0.3 \cdot n$	$\leq 0.5 \cdot n$
quorum size	$\leq 0.15 \cdot n$	$\leq 0.3 \cdot n$	$\leq 0.5 \cdot n$

Table A.9. Definition of low, medium, and high ranges for *k*, and quorum size parameters in $G(n, M)$ model Erdős-Rényi random network ANOVA analysis constructed with Watts-Strogatz model and $p = 1$.

	Low	Medium	High
k	$\leq 0.1 \cdot n$	$\leq 0.3 \cdot n$	$\leq 0.5 \cdot n$
quorum size	$\leq 0.15 \cdot n$	$\leq 0.3 \cdot n$	$\leq 0.5 \cdot n$

Appendix B. Full ANOVA Results

B.1. Overview

This appendix contains extended results of ANOVA output. General linear model ANOVA were used to accommodate the unbalanced data resulting from differing numbers of social networks for different models and the elimination of outlier cases when Honey Bee Consensus exceeded the allotted number of negotiation rounds.

The Box-Cox transformation method was used to normalize the residuals and four-in-one residual plots are provided for all ANOVA results. Minitab did not report any goodness of fit errors for any of the analyses.

B.2. Impact of Quorum Size and Model Parameters

The following ANOVA results are for analyses of the same social network models of varying population sizes. Effects of quorum size and model parameters were examined.

B.2.1. Watts-Strogatz Model

General Linear Model: ticks versus k-level, p-level, qs-level

```
Method

Factor coding           (-1, 0, +1)

Box-Cox transformation
Rounded λ               -0.962636
Estimated λ             -0.962636
95% CI for λ            (-0.978136, -0.948136)


Factor Information

Factor    Type   Levels  Values
k-level   Fixed       3  1, 2, 3
p-level   Fixed       3  1, 2, 3
qs-level  Fixed       3  0, 1, 2


Analysis of Variance for Transformed Response

Source                        DF   Adj SS    Adj MS  F-Value  P-Value
  k-level                      2  0.01108  0.005541   236.89    0.000
  p-level                      2  0.00053  0.000263    11.23    0.000
  qs-level                     2  0.07911  0.039553  1690.94    0.000
  k-level*p-level              4  0.00168  0.000419    17.90    0.000
  k-level*qs-level             4  0.01370  0.003425   146.42    0.000
```

```
  p-level*qs-level              4  0.00086  0.000215     9.18    0.000
  k-level*p-level*qs-level      8  0.00286  0.000357    15.27    0.000
Error                       41928  0.98074  0.000023
Total                       41954  1.11177
```

Model Summary for Transformed Response

```
        S    R-sq  R-sq(adj)  R-sq(pred)
0.0048364  11.79%     11.73%      11.69%
```

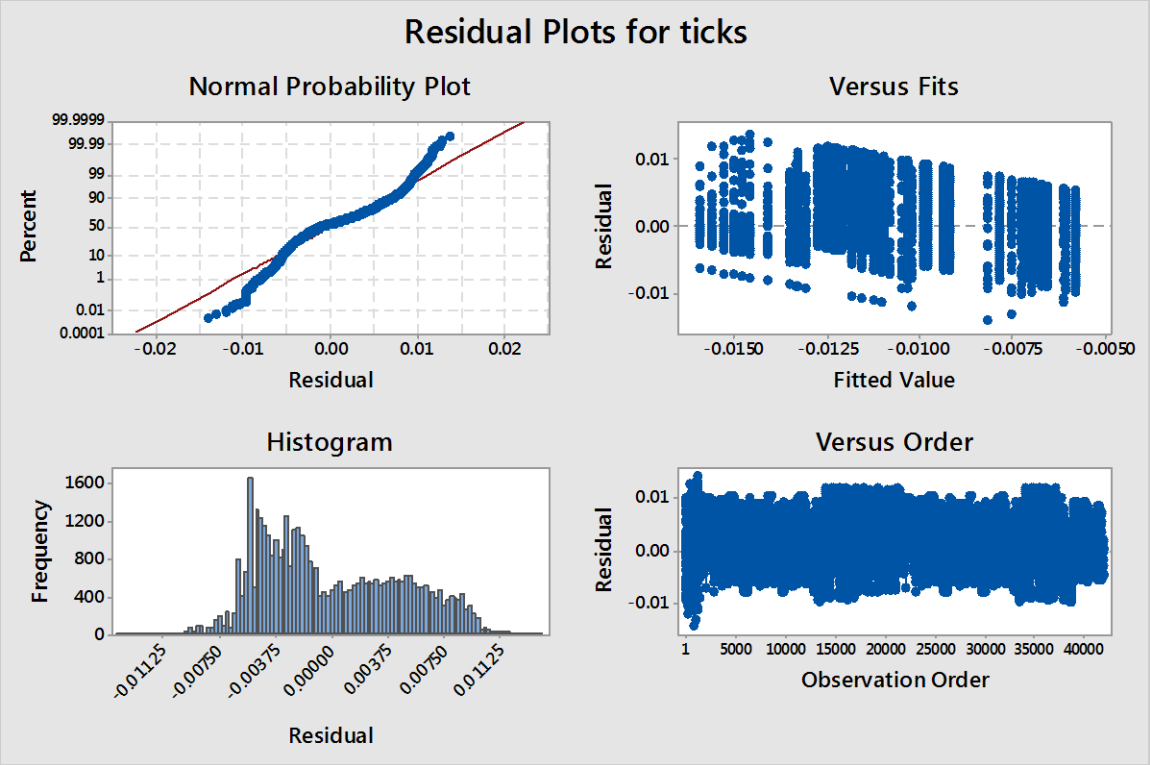
Coefficients for Transformed Response

```
Term                            Coef   SE Coef  T-Value  P-Value   VIF
Constant                   -0.010351  0.000035  -298.18    0.000
k-level
  1                         0.000835  0.000038    21.73    0.000  1.22
  2                        -0.000414  0.000052    -7.89    0.000  1.25
p-level
  1                        -0.000203  0.000050    -4.11    0.000  2.84
  2                        -0.000009  0.000047    -0.18    0.854  2.34
qs-level
  0                         0.003134  0.000054    58.02    0.000  3.37
  1                        -0.001839  0.000048   -38.61    0.000  2.98
k-level*p-level
  1 1                      -0.000347  0.000054    -6.43    0.000  2.97
  1 2                      -0.000107  0.000053    -2.02    0.043  2.42
  2 1                       0.000156  0.000076     2.05    0.040  1.58
  2 2                       0.000055  0.000070     0.79    0.430  1.54
k-level*qs-level
  1 0                      -0.001426  0.000059   -24.05    0.000  3.58
  1 1                       0.000650  0.000053    12.29    0.000  3.14
  2 0                       0.000591  0.000082     7.23    0.000  1.99
  2 1                      -0.000255  0.000072    -3.54    0.000  1.85
p-level*qs-level
  1 0                      -0.000361  0.000077    -4.68    0.000  4.73
  1 1                       0.000029  0.000068     0.42    0.673  4.10
  2 0                       0.000015  0.000074     0.21    0.837  3.73
  2 1                       0.000041  0.000065     0.62    0.533  3.31
k-level*p-level*qs-level
  1 1 0                    -0.000628  0.000083    -7.54    0.000  4.89
  1 1 1                    -0.000008  0.000074    -0.11    0.909  4.30
  1 2 0                    -0.000103  0.000081    -1.27    0.204  3.75
  1 2 1                     0.000102  0.000073     1.40    0.160  3.39
  2 1 0                     0.000345  0.000118     2.91    0.004  2.51
  2 1 1                    -0.000014  0.000104    -0.13    0.895  2.35
  2 2 0                    -0.000021  0.000109    -0.19    0.846  2.45
  2 2 1                     0.000024  0.000096     0.25    0.805  2.30
```

Regression Equation

```
-ticks^-0.962636 = -0.010351 + 0.000835 k-level_1 - 0.000414 k-level_2
- 0.000421 k-level_3
                   - 0.000203 p-level_1 - 0.000009 p-level_2 + 0.000212 p-
level_3
                   + 0.003134 qs-level_0 - 0.001839 qs-level_1 - 0.001295 qs-
level_2
```

```
                  - 0.000347 k-level*p-level_1 1 - 0.000107 k-level*p-level_1
2
                  + 0.000454 k-level*p-level_1 3 + 0.000156 k-level*p-level_2
1
                  + 0.000055 k-level*p-level_2 2 - 0.000211 k-level*p-level_2
3
                  + 0.000191 k-level*p-level_3 1 + 0.000052 k-level*p-level_3
2
                  - 0.000243 k-level*p-level_3 3 - 0.001426 k-level*qs-level_1
0
                  + 0.000650 k-level*qs-level_1 1 + 0.000776 k-level*qs-
level_1 2
                  + 0.000591 k-level*qs-level_2 0 - 0.000255 k-level*qs-
level_2 1
                  - 0.000336 k-level*qs-level_2 2 + 0.000835 k-level*qs-
level_3 0
                  - 0.000395 k-level*qs-level_3 1 - 0.000440 k-level*qs-
level_3 2
                  - 0.000361 p-level*qs-level_1 0 + 0.000029 p-level*qs-
level_1 1
                  + 0.000333 p-level*qs-level_1 2 + 0.000015 p-level*qs-
level_2 0
                  + 0.000041 p-level*qs-level_2 1 - 0.000056 p-level*qs-
level_2 2
                  + 0.000346 p-level*qs-level_3 0 - 0.000069 p-level*qs-
level_3 1
                  - 0.000277 p-level*qs-level_3 2 - 0.000628 k-level*p-
level*qs-level_1 1 0
                  - 0.000008 k-level*p-level*qs-level_1 1 1
                  + 0.000636 k-level*p-level*qs-level_1 1 2
                  - 0.000103 k-level*p-level*qs-level_1 2 0
                  + 0.000102 k-level*p-level*qs-level_1 2 1
                  + 0.000001 k-level*p-level*qs-level_1 2 2
                  + 0.000731 k-level*p-level*qs-level_1 3 0
                  - 0.000094 k-level*p-level*qs-level_1 3 1
                  - 0.000637 k-level*p-level*qs-level_1 3 2
                  + 0.000345 k-level*p-level*qs-level_2 1 0
                  - 0.000014 k-level*p-level*qs-level_2 1 1
                  - 0.000331 k-level*p-level*qs-level_2 1 2
                  - 0.000021 k-level*p-level*qs-level_2 2 0
                  + 0.000024 k-level*p-level*qs-level_2 2 1
                  - 0.000003 k-level*p-level*qs-level_2 2 2
                  - 0.000323 k-level*p-level*qs-level_2 3 0
                  - 0.000010 k-level*p-level*qs-level_2 3 1
                  + 0.000333 k-level*p-level*qs-level_2 3 2
                  + 0.000283 k-level*p-level*qs-level_3 1 0
                  + 0.000022 k-level*p-level*qs-level_3 1 1
                  - 0.000306 k-level*p-level*qs-level_3 1 2
                  + 0.000125 k-level*p-level*qs-level_3 2 0
                  - 0.000126 k-level*p-level*qs-level_3 2 1
                  + 0.000002 k-level*p-level*qs-level_3 2 2
                  - 0.000408 k-level*p-level*qs-level_3 3 0
                  + 0.000104 k-level*p-level*qs-level_3 3 1
                  + 0.000304 k-level*p-level*qs-level_3 3 2
```



B.2.2. Barabási-Albert Model

General Linear Model: ticks versus n, m-level, qs-level

```
Method

Factor coding            (-1, 0, +1)

Box-Cox transformation
Rounded λ                -1.28924
Estimated λ              -1.28924
95% CI for λ             (-1.30174, -1.27674)


Factor Information

Factor     Type   Levels  Values
n          Fixed       3  50, 200, 1000
m-level    Fixed       3  1, 2, 3
qs-level   Fixed       3  0, 1, 2


Analysis of Variance for Transformed Response

Source                    DF     Adj SS     Adj MS  F-Value  P-Value
  n                        2  0.000477  0.000239   184.46    0.000
  m-level                  2  0.000004  0.000002     1.41    0.244
```

```
  qs-level                2  0.001605  0.000803   620.11    0.000
  n*m-level               4  0.000012  0.000003     2.34    0.053
  n*qs-level              4  0.000472  0.000118    91.18    0.000
  m-level*qs-level        4  0.000026  0.000006     4.98    0.001
  n*m-level*qs-level      8  0.000033  0.000004     3.16    0.001
Error                 95197  0.123199  0.000001
Total                 95223  0.149644
```

Model Summary for Transformed Response

```
        S    R-sq  R-sq(adj)  R-sq(pred)
0.0011376  17.67%     17.65%      17.62%
```

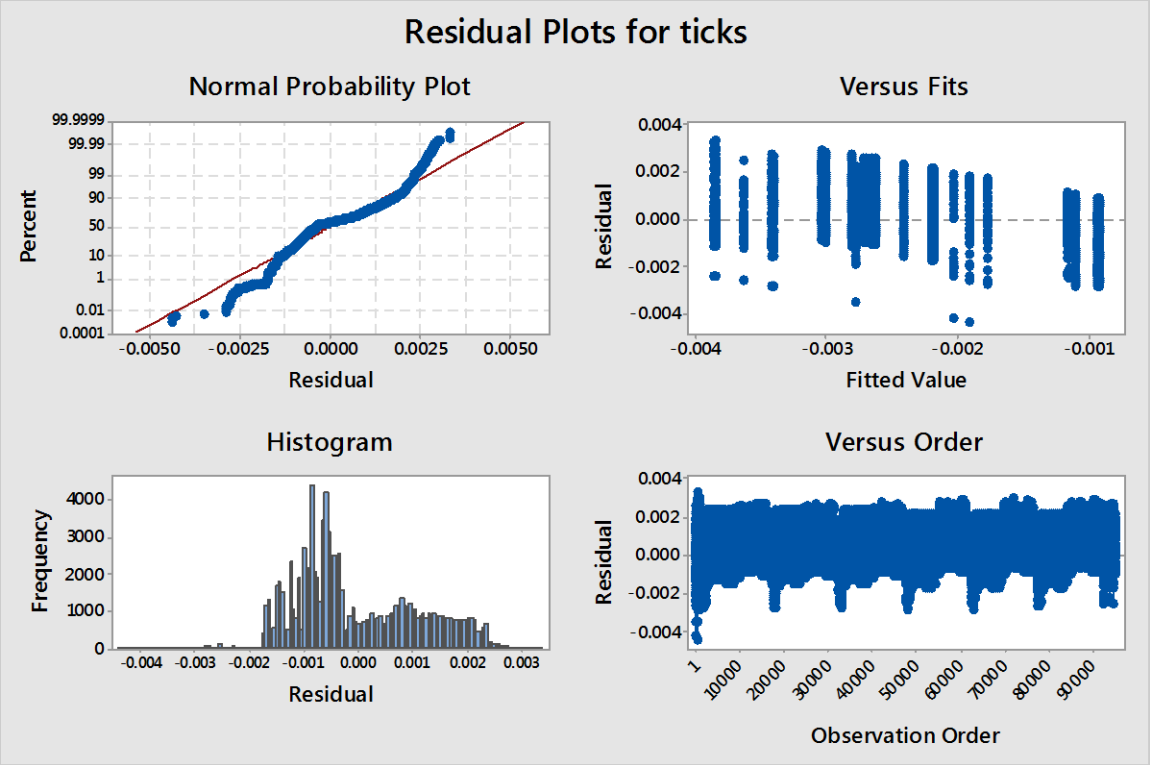
Coefficients for Transformed Response

```
Term                      Coef   SE Coef  T-Value  P-Value    VIF
Constant            -0.002393  0.000018  -129.97    0.000
n
  50                -0.000563  0.000033   -17.29    0.000   2.89
  200                0.000330  0.000019    17.59    0.000   3.15
m-level
  1                  0.000044  0.000027     1.62    0.105  43.70
  2                 -0.000029  0.000025    -1.18    0.238  20.86
qs-level
  0                  0.001057  0.000030    35.20    0.000  26.00
  1                 -0.000517  0.000024   -21.60    0.000  38.37
n*m-level
  50 1               0.000103  0.000050     2.04    0.041   4.55
  50 2              -0.000068  0.000043    -1.58    0.115   2.88
  200 1             -0.000082  0.000028    -2.98    0.003  44.89
  200 2              0.000049  0.000025     1.95    0.051  21.56
n*qs-level
  50 0               0.000002  0.000052     0.03    0.977   4.58
  50 1              -0.000306  0.000043    -7.15    0.000   4.39
  200 0              0.000029  0.000031     0.94    0.345  28.40
  200 1              0.000319  0.000024    13.15    0.000  39.34
m-level*qs-level
  1 0               -0.000126  0.000044    -2.85    0.004  70.41
  1 1               -0.000030  0.000036    -0.85    0.396  72.32
  2 0                0.000031  0.000040     0.76    0.447  34.86
  2 1                0.000029  0.000032     0.92    0.359  34.31
n*m-level*qs-level
  50 1 0            -0.000148  0.000080    -1.85    0.065   7.08
  50 1 1             0.000028  0.000066     0.43    0.670   6.93
  50 2 0             0.000057  0.000069     0.83    0.407   4.52
  50 2 1             0.000004  0.000057     0.06    0.950   4.29
  200 1 0            0.000050  0.000045     1.11    0.266  72.37
  200 1 1           -0.000073  0.000036    -2.03    0.043  73.43
  200 2 0           -0.000008  0.000042    -0.19    0.852  36.67
  200 2 1            0.000024  0.000033     0.73    0.465  35.47
```

Regression Equation

```
-ticks^-1.28924 = -0.002393 - 0.000563 n_50 + 0.000330 n_200 + 0.000234 n_1000
                  + 0.000044 m-level_1 - 0.000029 m-level_2 - 0.000015 m-
level_3
```

```
                  + 0.001057 qs-level_0 - 0.000517 qs-level_1 - 0.000541 qs-
level_2
                  + 0.000103 n*m-level_50 1 - 0.000068 n*m-level_50 2
- 0.000034 n*m-level_50
                  3 - 0.000082 n*m-level_200 1 + 0.000049 n*m-level_200 2
                  + 0.000033 n*m-level_200 3 - 0.000020 n*m-level_1000 1
                  + 0.000019 n*m-level_1000 2 + 0.000001 n*m-level_1000 3
                  + 0.000002 n*qs-level_50 0 - 0.000306 n*qs-level_50 1
                  + 0.000304 n*qs-level_50 2 + 0.000029 n*qs-level_200 0
                  + 0.000319 n*qs-level_200 1 - 0.000348 n*qs-level_200 2
                  - 0.000031 n*qs-level_1000 0 - 0.000014 n*qs-level_1000 1
                  + 0.000044 n*qs-level_1000 2 - 0.000126 m-level*qs-level_1 0
                  - 0.000030 m-level*qs-level_1 1 + 0.000156 m-level*qs-level_1
2
                  + 0.000031 m-level*qs-level_2 0 + 0.000029 m-level*qs-level_2
1
                  - 0.000060 m-level*qs-level_2 2 + 0.000095 m-level*qs-level_3
0
                  + 0.000001 m-level*qs-level_3 1 - 0.000096 m-level*qs-level_3
2
                  - 0.000148 n*m-level*qs-level_50 1 0 + 0.000028 n*m-level*qs-
level_50 1 1
                  + 0.000120 n*m-level*qs-level_50 1 2 + 0.000057 n*m-level*qs-
level_50 2 0
                  + 0.000004 n*m-level*qs-level_50 2 1 - 0.000061 n*m-level*qs-
level_50 2 2
                  + 0.000091 n*m-level*qs-level_50 3 0 - 0.000032 n*m-level*qs-
level_50 3 1
                  - 0.000059 n*m-level*qs-level_50 3 2 + 0.000050 n*m-level*qs-
level_200 1 0
                  - 0.000073 n*m-level*qs-level_200 1 1 + 0.000023 n*m-
level*qs-level_200 1 2
                  - 0.000008 n*m-level*qs-level_200 2 0 + 0.000024 n*m-
level*qs-level_200 2 1
                  - 0.000016 n*m-level*qs-level_200 2 2 - 0.000042 n*m-
level*qs-level_200 3 0
                  + 0.000049 n*m-level*qs-level_200 3 1 - 0.000007 n*m-
level*qs-level_200 3 2
                  + 0.000098 n*m-level*qs-level_1000 1 0 + 0.000045 n*m-
level*qs-level_1000 1
                  1 - 0.000143 n*m-level*qs-level_1000 1 2 - 0.000050 n*m-
level*qs-level_1000
                  2 0 - 0.000027 n*m-level*qs-level_1000 2 1
                  + 0.000077 n*m-level*qs-level_1000 2 2 - 0.000049 n*m-
level*qs-level_1000 3
                  0 - 0.000017 n*m-level*qs-level_1000 3 1 + 0.000066 n*m-
level*qs-level_1000
                  3 2
```



B.2.3. Erdős-Rényi Model

General Linear Model: ticks versus n, k-level, qs-level

```
Method

Factor coding            (-1, 0, +1)

Box-Cox transformation
Rounded λ                -1.11988
Estimated λ              -1.11988
95% CI for λ             (-1.16838, -1.07138)


Factor Information

Factor    Type   Levels  Values
n         Fixed       3  50, 200, 1000
k-level   Fixed       3  1, 2, 3
qs-level  Fixed       3  0, 1, 2


Analysis of Variance for Transformed Response

Source                  DF    Adj SS    Adj MS  F-Value  P-Value
  n                      2  0.000974  0.000487    91.74    0.000
  k-level                2  0.000001  0.000000     0.07    0.937
```

```
  qs-level               2  0.001841  0.000920   173.32    0.000
  n*k-level              4  0.000169  0.000042     7.95    0.000
  n*qs-level             4  0.000771  0.000193    36.32    0.000
  k-level*qs-level       4  0.000245  0.000061    11.56    0.000
  n*k-level*qs-level     8  0.000156  0.000020     3.68    0.000
Error                 4778  0.025370  0.000005
Total                 4804  0.031577
```

Model Summary for Transformed Response

```
        S    R-sq  R-sq(adj)  R-sq(pred)
0.0023043  19.66%     19.22%      18.81%
```

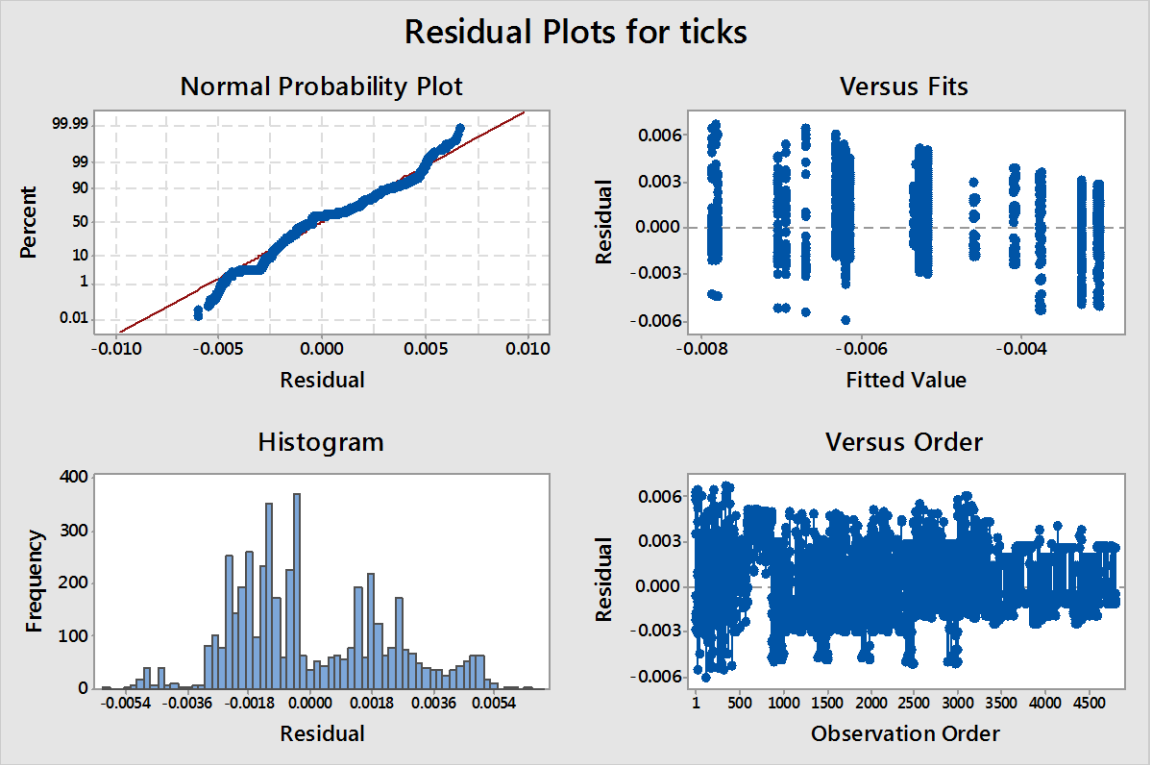
Coefficients for Transformed Response

```
Term                      Coef   SE Coef  T-Value  P-Value   VIF
Constant            -0.005350  0.000047  -113.02    0.000
n
  50                -0.001082  0.000081   -13.29    0.000  2.09
  200                0.000397  0.000056     7.02    0.000  2.33
k-level
  1                 -0.000023  0.000065    -0.35    0.728  2.61
  2                  0.000017  0.000068     0.25    0.802  1.94
qs-level
  0                  0.001383  0.000074    18.58    0.000  3.07
  1                 -0.000769  0.000063   -12.13    0.000  2.78
n*k-level
  50 1              -0.000438  0.000114    -3.83    0.000  3.05
  50 2               0.000216  0.000116     1.86    0.063  3.15
  200 1              0.000419  0.000074     5.64    0.000  3.07
  200 2             -0.000202  0.000083    -2.44    0.015  2.50
n*qs-level
  50 0               0.000319  0.000129     2.48    0.013  3.70
  50 1              -0.000636  0.000108    -5.88    0.000  3.25
  200 0              0.000480  0.000088     5.44    0.000  4.14
  200 1             -0.000215  0.000076    -2.82    0.005  3.65
k-level*qs-level
  1 0               -0.000686  0.000103    -6.68    0.000  4.71
  1 1                0.000262  0.000087     3.01    0.003  4.04
  2 0                0.000337  0.000107     3.16    0.002  3.10
  2 1               -0.000133  0.000091    -1.46    0.144  2.73
n*k-level*qs-level
  50 1 0            -0.000815  0.000180    -4.52    0.000  4.88
  50 1 1             0.000244  0.000151     1.61    0.107  4.27
  50 2 0             0.000397  0.000184     2.16    0.031  5.06
  50 2 1            -0.000149  0.000154    -0.97    0.334  4.39
  200 1 0            0.000139  0.000116     1.20    0.230  5.57
  200 1 1            0.000007  0.000100     0.07    0.941  4.86
  200 2 0           -0.000061  0.000129    -0.47    0.636  3.98
  200 2 1            0.000030  0.000112     0.27    0.790  3.55
```

Regression Equation

```
-ticks^-1.11988 = -0.005350 - 0.001082 n_50 + 0.000397 n_200 + 0.000685 n_1000
                  - 0.000023 k-level_1 + 0.000017 k-level_2 + 0.000006 k-
level_3
```

```
                  + 0.001383 qs-level_0 - 0.000769 qs-level_1 - 0.000615 qs-
level_2
                  - 0.000438 n*k-level_50 1 + 0.000216 n*k-level_50 2
+ 0.000222 n*k-level_50
                  3 + 0.000419 n*k-level_200 1 - 0.000202 n*k-level_200 2
                  - 0.000217 n*k-level_200 3 + 0.000018 n*k-level_1000 1
                  - 0.000014 n*k-level_1000 2 - 0.000004 n*k-level_1000 3
                  + 0.000319 n*qs-level_50 0 - 0.000636 n*qs-level_50 1
                  + 0.000317 n*qs-level_50 2 + 0.000480 n*qs-level_200 0
                  - 0.000215 n*qs-level_200 1 - 0.000265 n*qs-level_200 2
                  - 0.000799 n*qs-level_1000 0 + 0.000851 n*qs-level_1000 1
                  - 0.000052 n*qs-level_1000 2 - 0.000686 k-level*qs-level_1 0
                  + 0.000262 k-level*qs-level_1 1 + 0.000423 k-level*qs-level_1
2
                  + 0.000337 k-level*qs-level_2 0 - 0.000133 k-level*qs-level_2
1
                  - 0.000205 k-level*qs-level_2 2 + 0.000349 k-level*qs-level_3
0
                  - 0.000130 k-level*qs-level_3 1 - 0.000219 k-level*qs-level_3
2
                  - 0.000815 n*k-level*qs-level_50 1 0 + 0.000244 n*k-level*qs-
level_50 1 1
                  + 0.000571 n*k-level*qs-level_50 1 2 + 0.000397 n*k-level*qs-
level_50 2 0
                  - 0.000149 n*k-level*qs-level_50 2 1 - 0.000248 n*k-level*qs-
level_50 2 2
                  + 0.000418 n*k-level*qs-level_50 3 0 - 0.000095 n*k-level*qs-
level_50 3 1
                  - 0.000322 n*k-level*qs-level_50 3 2 + 0.000139 n*k-level*qs-
level_200 1 0
                  + 0.000007 n*k-level*qs-level_200 1 1 - 0.000147 n*k-
level*qs-level_200 1 2
                  - 0.000061 n*k-level*qs-level_200 2 0 + 0.000030 n*k-
level*qs-level_200 2 1
                  + 0.000032 n*k-level*qs-level_200 2 2 - 0.000078 n*k-
level*qs-level_200 3 0
                  - 0.000037 n*k-level*qs-level_200 3 1 + 0.000115 n*k-
level*qs-level_200 3 2
                  + 0.000675 n*k-level*qs-level_1000 1 0 - 0.000251 n*k-
level*qs-level_1000 1
                  1 - 0.000424 n*k-level*qs-level_1000 1 2 - 0.000336 n*k-
level*qs-level_1000
                  2 0 + 0.000119 n*k-level*qs-level_1000 2 1
                  + 0.000217 n*k-level*qs-level_1000 2 2 - 0.000339 n*k-
level*qs-level_1000 3
                  0 + 0.000132 n*k-level*qs-level_1000 3 1 + 0.000207 n*k-
level*qs-level_1000
                  3 2
```



B.3. Impact of Quorum Size and Social Network Model

The following ANOVA results are for analyses of the different social network models of the same population sizes. Effects of quorum size and social network model were examined.

B.3.1. Population Size 50

General Linear Model: ticks versus model, qs-level

```
Method

Factor coding            (-1, 0, +1)

Box-Cox transformation
Rounded λ                -1.16755
Estimated λ              -1.16755
95% CI for λ             (-1.24005, -1.09705)


Factor Information

Factor    Type   Levels  Values
model     Fixed       3  1, 2, 3
qs-level  Fixed       3  0, 1, 2
```

Analysis of Variance for Transformed Response

Source	DF	Adj SS	Adj MS	F-Value	P-Value
model	2	0.000030	0.000015	3.14	0.044
qs-level	2	0.001920	0.000960	197.81	0.000
model*qs-level	4	0.000072	0.000018	3.69	0.005
Error	2520	0.012228	0.000005		
Total	2528	0.014544			

Model Summary for Transformed Response

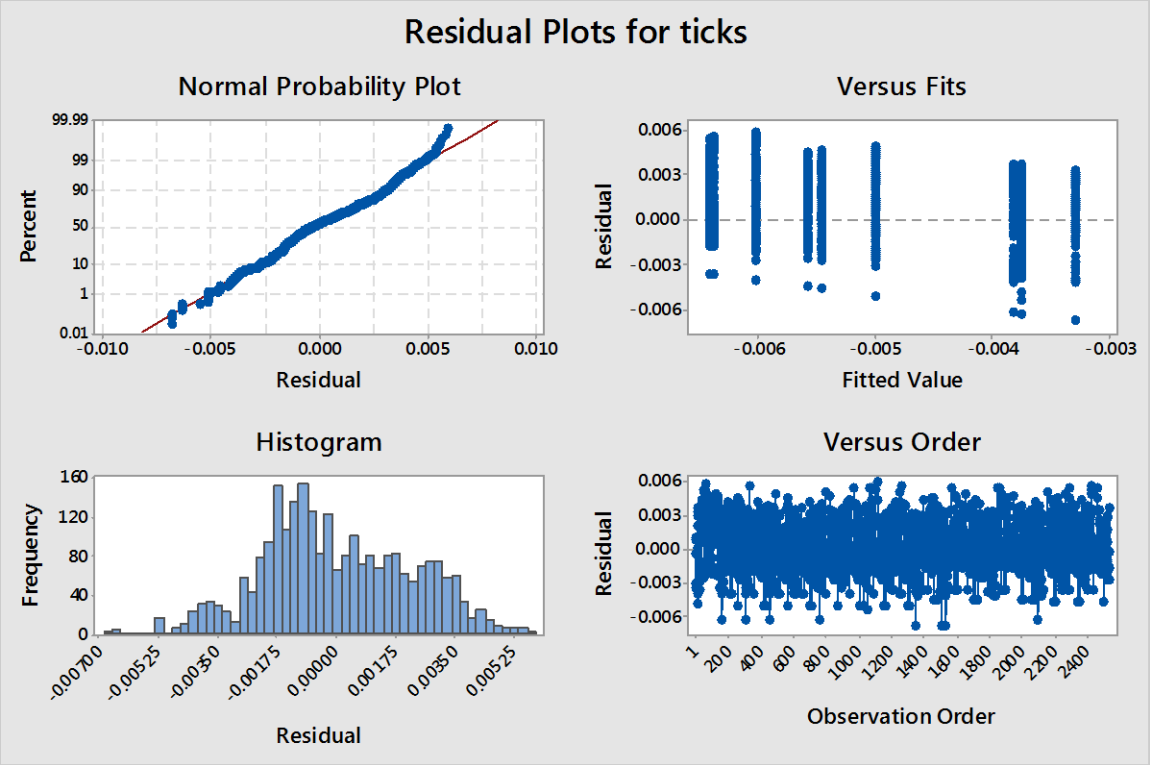
S	R-sq	R-sq(adj)	R-sq(pred)
0.0022028	15.92%	15.65%	15.29%

Coefficients for Transformed Response

Term	Coef	SE Coef	T-Value	P-Value	VIF
Constant	-0.005081	0.000051	-99.38	0.000	
model					
1	0.000155	0.000063	2.46	0.014	1.21
2	-0.000015	0.000071	-0.21	0.837	1.21
qs-level					
0	0.001456	0.000081	17.93	0.000	1.90
1	-0.001192	0.000067	-17.68	0.000	1.91
model*qs-level					
1 0	-0.000293	0.000100	-2.92	0.004	2.18
1 1	0.000100	0.000083	1.21	0.227	2.05
2 0	0.000347	0.000113	3.06	0.002	1.88
2 1	-0.000126	0.000094	-1.34	0.180	1.74

Regression Equation

```
-ticks^-1.16755 = -0.005081 + 0.000155 model_1 - 0.000015 model_2
- 0.000140 model_3
                  + 0.001456 qs-level_0 - 0.001192 qs-level_1 - 0.000264 qs-
level_2
                  - 0.000293 model*qs-level_1 0 + 0.000100 model*qs-level_1 1
                  + 0.000192 model*qs-level_1 2 + 0.000347 model*qs-level_2 0
                  - 0.000126 model*qs-level_2 1 - 0.000221 model*qs-level_2 2
                  - 0.000055 model*qs-level_3 0 + 0.000026 model*qs-level_3 1
                  + 0.000029 model*qs-level_3 2
```

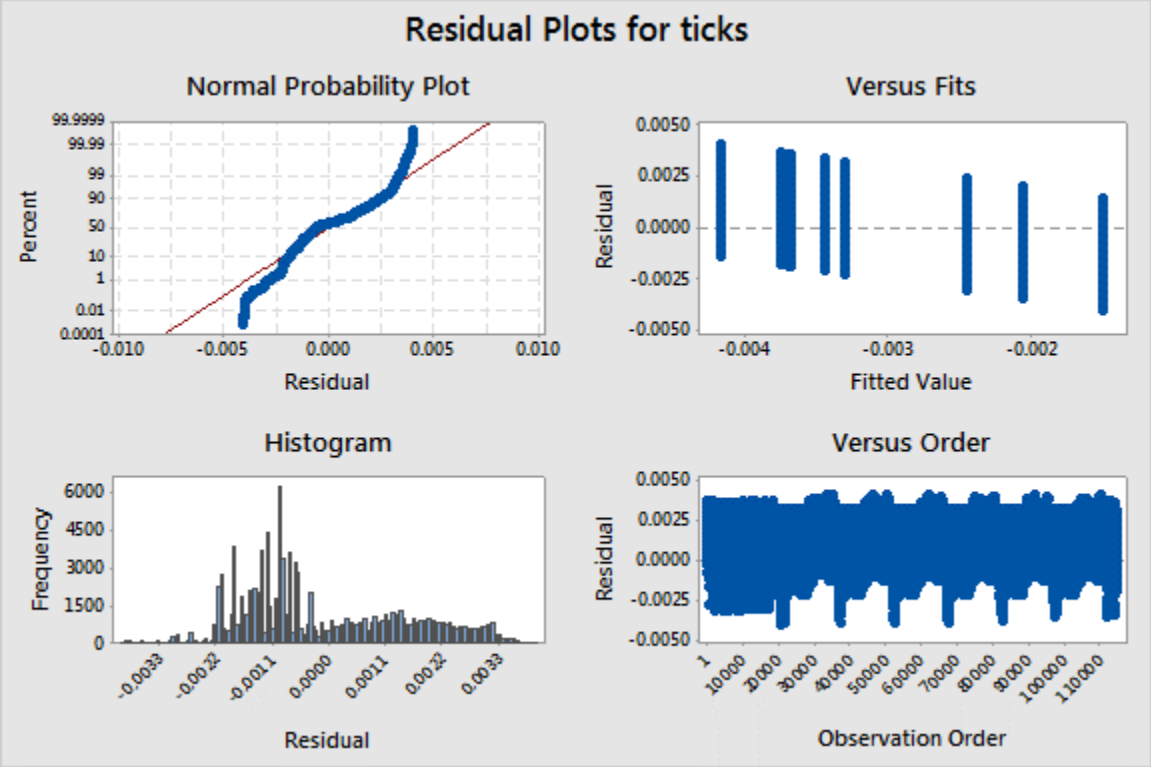


B.3.2. Population Size 200

General Linear Model: ticks versus model, qs-level

```
Method

Factor coding            (-1, 0, +1)

Box-Cox transformation
Rounded λ                -1.21023
Estimated λ              -1.21023
95% CI for λ             (-1.22073, -1.19873)


Factor Information

Factor     Type   Levels  Values
model      Fixed       3  1, 2, 3
qs-level   Fixed       3  0, 1, 2


Analysis of Variance for Transformed Response

Source                DF     Adj SS     Adj MS  F-Value  P-Value
  model                2   0.000592   0.000296   111.18    0.000
  qs-level             2   0.011365   0.005683  2133.79    0.000
  model*qs-level       4   0.007378   0.001844   692.56    0.000
```

```
Error             115065  0.306436  0.000003
Total             115073  0.362373


Model Summary for Transformed Response

        S    R-sq  R-sq(adj)  R-sq(pred)
0.0016319  15.44%     15.43%      15.42%


Coefficients for Transformed Response

Term                  Coef   SE Coef  T-Value  P-Value   VIF
Constant         -0.003108  0.000011  -276.49    0.000
model
  1              -0.000078  0.000013    -5.97    0.000  1.33
  2               0.000124  0.000012    10.23    0.000  1.44
qs-level
  0               0.001112  0.000017    64.68    0.000  5.86
  1              -0.000465  0.000015   -30.14    0.000  9.01
model*qs-level
  1 0            -0.000366  0.000020   -18.46    0.000  2.26
  1 1            -0.000032  0.000018    -1.75    0.080  2.08
  2 0             0.000372  0.000019    19.56    0.000  5.67
  2 1             0.000154  0.000016     9.46    0.000  8.62


Regression Equation

-ticks^-1.21023 = -0.003108 - 0.000078 model_1 + 0.000124 model_2
- 0.000046 model_3
                  + 0.001112 qs-level_0 - 0.000465 qs-level_1 - 0.000647 qs-
level_2
                  - 0.000366 model*qs-level_1 0 - 0.000032 model*qs-level_1 1
                  + 0.000397 model*qs-level_1 2 + 0.000372 model*qs-level_2 0
                  + 0.000154 model*qs-level_2 1 - 0.000527 model*qs-level_2 2
                  - 0.000006 model*qs-level_3 0 - 0.000123 model*qs-level_3 1
                  + 0.000129 model*qs-level_3 2
```

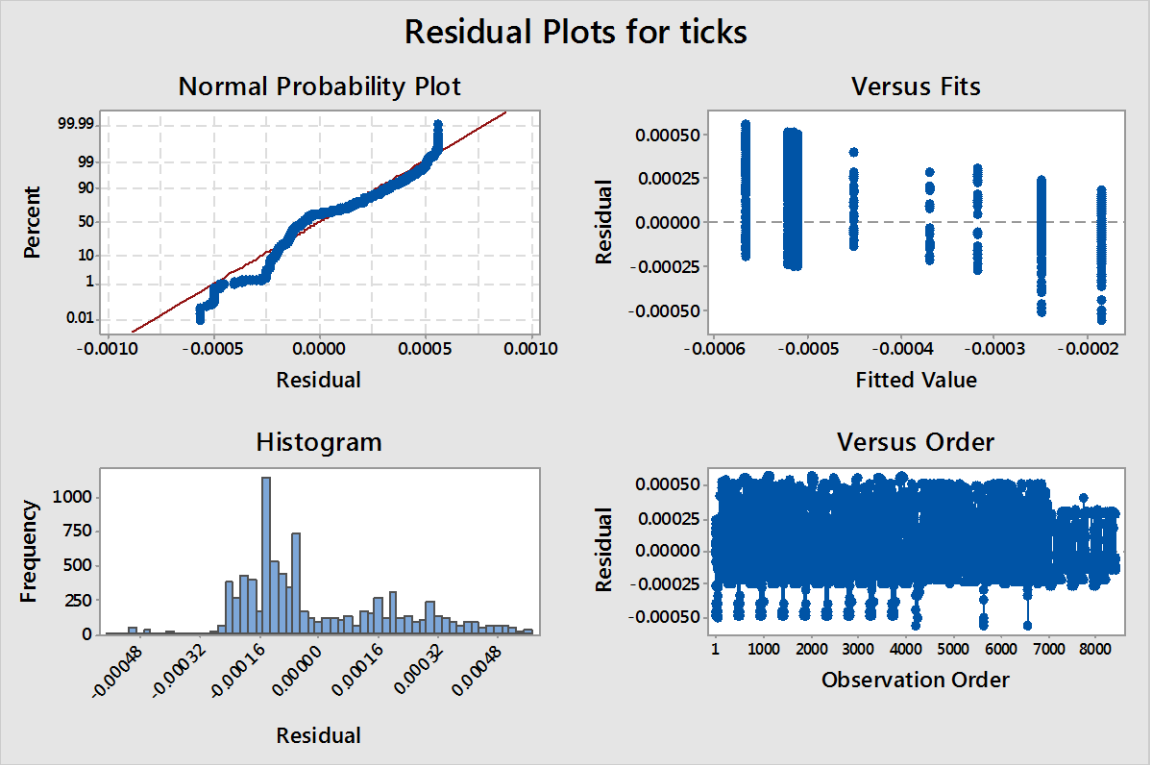


B.3.3. Population Size 1,000

General Linear Model: ticks versus model, qs-level

```
Method

Factor coding            (-1, 0, +1)

Box-Cox transformation
Rounded λ                -1.65271
Estimated λ              -1.65271
95% CI for λ             (*, *)


Factor Information

Factor     Type   Levels  Values
model      Fixed       3  1, 2, 3
qs-level   Fixed       3  0, 1, 2


Analysis of Variance for Transformed Response

Source              DF    Adj SS    Adj MS  F-Value  P-Value
  model              2  0.000005  0.000002    52.93    0.000
  qs-level           2  0.000055  0.000028   634.03    0.000
  model*qs-level     4  0.000012  0.000003    68.40    0.000
```

```
Error             8414  0.000365  0.000000
Total             8422  0.000463


Model Summary for Transformed Response

        S    R-sq  R-sq(adj)  R-sq(pred)
0.0002084  21.09%     21.02%      20.94%


Coefficients for Transformed Response

Term                  Coef   SE Coef  T-Value  P-Value   VIF
Constant         -0.000409  0.000003  -147.47    0.000
model
  1              -0.000035  0.000003   -10.07    0.000  1.27
  2               0.000004  0.000004     1.07    0.285  1.42
qs-level
  0               0.000160  0.000005    35.45    0.000  2.16
  1              -0.000077  0.000004   -20.73    0.000  2.16
model*qs-level
  1 0             0.000036  0.000006     6.50    0.000  2.60
  1 1            -0.000047  0.000005   -10.24    0.000  2.22
  2 0             0.000062  0.000007     9.42    0.000  2.59
  2 1            -0.000039  0.000005    -7.74    0.000  2.16


Regression Equation

-ticks^-1.65271 = -0.000409 - 0.000035 model_1 + 0.000004 model_2
+ 0.000030 model_3
                  + 0.000160 qs-level_0 - 0.000077 qs-level_1 - 0.000083 qs-
level_2
                  + 0.000036 model*qs-level_1 0 - 0.000047 model*qs-level_1 1
                  + 0.000011 model*qs-level_1 2 + 0.000062 model*qs-level_2 0
                  - 0.000039 model*qs-level_2 1 - 0.000023 model*qs-level_2 2
                  - 0.000098 model*qs-level_3 0 + 0.000087 model*qs-level_3 1
                  + 0.000011 model*qs-level_3 2
```


Residual Plots for ticks
Normal Probability Plot
Percent
99.99
99
90
50
10
1
0.01
-0.0010
-0.0005
0.0000
0.0005
0.0010
Residual
Versus Fits
Residual
0.00050
0.00025
0.00000
-0.00025
-0.00050
-0.0006
-0.0005
-0.0004
-0.0003
-0.0002
Fitted Value
Histogram
Frequency
1000
750
500
250
0
-0.00048
-0.00032
-0.00016
0.00000
0.00016
0.00032
0.00048
Residual
Versus Order
Residual
0.00050
0.00025
0.00000
-0.00025
-0.00050
1
1000
2000
3000
4000
5000
6000
7000
8000
Observation Order

Appendix C. Source Code

C.1. Core Model Classes

C.1.1. Agent Class (Agent.java)

```
package swarmdm.agents;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.SortedMap;
import java.util.TreeMap;
import java.util.TreeSet;

import repast.simphony.random.RandomHelper;
import repast.simphony.util.SimUtilities;
import swarmdm.strategies.DecayFunction;
import swarmdm.strategies.InitFunction;
import swarmdm.strategies.LinearDecay;
import swarmdm.strategies.ProbabilityToDouble;

/**
 * A member of the collective needing to reach a decision.
 * An Agent has a decision preference that changes based on influence from
 * neighboring Agent preferences and a progressive decay toward neutrality.
 */

/**
 * @author alexander.mentis
 *
 */
public class Agent implements Comparable<Agent>
{
	// A unique identifier for the agent.
	private int agentID;

	// The neighbors that influence this agent
	private Set<Agent> neighbors = null;

	// Map of decision id's to preference values.
	private Map<Integer, Double> decisionPrefs = null;

	// ID of preferred choice; 0 means not committed
	private int choice = 0;

	// Duration this agent will remain committed to its choice.
	private double duration = 0.0;

	// Concrete interface implementation that returns a duration value, given
	// a preference value.
	private InitFunction init = new ProbabilityToDouble();

	// The rate to be used in the decay function.
	private double decayRate = 1.0;
```

```
// The point at or below which the duration of commitment expires.
private double evapThreshold = 0.0;

// The decay function used to reduce the remaining commitment duration at
// each round.
private DecayFunction decayFunc = new LinearDecay(decayRate);

// Preferred quorum data.
private QuorumData prefQuorum = null;

// Threshold number of agents that must agree in order to constitute a
// quorum.
private int quorumThreshold = 0;

/**
 * Constructs a new agent with its initial preference weights and
 * initializes its preference and starting Quorum Data object.
 *
 * @param prefs
 *            the preference value for each possible decision
 * @param quorumThreshold
 *            the number of agents that a quorum data object must contain
 *            to achieve quorum
 * @param evapThreshold
 *            the duration value below which commitment expires
 */
public Agent(Map<Integer, Double> prefs, int quorumThreshold,
             double decayRate, double evapThreshold, int id)
{
       this.agentID = id;
       this.decisionPrefs = prefs;

       int maxPref = getMaxPref(prefs);
       this.choice = maxPref;
       this.duration = init.execute(prefs.get(maxPref));

       // add self to preferred quorum
       prefQuorum = new QuorumData(choice);
       prefQuorum.addAgent(this);

       this.quorumThreshold = quorumThreshold;
       this.decayRate = decayRate;
       this.evapThreshold = evapThreshold;

       this.neighbors = new TreeSet<Agent>();
}

/**
 * Accessor method for agentID.
 *
 * @return this agent's unique identifier
 */
public int getAgentID()
{
       return agentID;
}

/**
 * Mutator method for neighbors. Add an Agent to this Agent's visible
 * neighborhood.
```

```
 *
 * @param agent
 *            the Agent to be added to the neighborhood
 * @return <code>true</code> if this neighborhood did not already contain
 *         the specified Agent
 */
public boolean addNeighbor(Agent agent)
{
	return this.neighbors.add(agent);
}

/**
 * Accessor method for this agent's neighbor set.
 *
 * @return a copy of the neighbor set
 */
public TreeSet<Agent> getNeighbors()
{
	return new TreeSet<Agent>(neighbors);
}

/**
 * Return the number of neighbors this agent has.
 *
 * @return the number of neighbors
 */
public int neighborCount()
{
	return neighbors.size();
}

/**
 * Accessor method for this agents preference mapping.
 *
 * @return a map of choices to preferences
 */
public Map<Integer, Double> getPreferences()
{
	return this.decisionPrefs;
}

/**
 * Accessor method for agent choice.
 */
public int getChoice()
{
	return choice;
}

/**
 * Accessor method for duration.
 *
 * @return this agent's remaining duration for its choice
 */
public double getDuration()
{
	return duration;
}

/**
```

```
 * Finds the choiceID with the highest preference.
 *
 * @param prefs
 *            the mapping of choice IDs to preferences
 * @return the choice ID with the highest preference
 */
private int getMaxPref(Map<Integer, Double> prefs)
{
	// Convert preferences to a shuffleable list. We shuffle it to
	// randomize tie-breaking between equal preferences.
	ArrayList<Map.Entry<Integer, Double>> prefList =
			new ArrayList<Map.Entry<Integer, Double>>();

	prefList.addAll(prefs.entrySet());
	SimUtilities.shuffle(prefList, RandomHelper.getUniform());

	int maxID = prefList.get(0).getKey();
	double maxPref = prefs.get(maxID);
	for (Map.Entry<Integer, Double> preference : prefList)
	{
		Double thisPref = preference.getValue();
		// select the most preferred choice
		if (thisPref > maxPref)
		{
			maxID = preference.getKey();
			maxPref = thisPref;
		}
	}

	return maxID;
}

/**
 * Returns the quorum data object containing information about this
 * agent's preferred quorum membership.
 *
 * @return this agent's preferred quorum
 */
public QuorumData getQuorumData()
{
	return prefQuorum;
}

/**
 * Merges src quorum into dest quorum.
 *
 * @param src
 *            the source QuorumData object
 * @param dest
 *            the destination QuorumData object or null
 */
private void mergeQuorums(QuorumData src, QuorumData dest)
{
	// merge src into destination
	for (Map.Entry<Integer, Double> e : src.entrySet())
	{
		Integer agentID = e.getKey();
		Double agentDuration = e.getValue();
		LinkedHashMap<Integer, Double> destSet =
			dest.getAgentSet();
```

```
                if (!destSet.containsKey(agentID)
                                || destSet.get(agentID) < agentDuration)
                {
                        destSet.put(agentID, agentDuration);
                }
        }
}

/**
 * If one and only one quorum data object has reached a quorum threshold,
 * it's ID choice is returned, otherwise, none or more than one quorum
 * data object has a quorum, so return null.
 *
 * @param quorums
 *            the Map of choiceIDs to quorum data objects
 * @return the choiceID of the one quorum data with a quorum or null if
 *         there are 0 or more than one quorum data objects achieving a
 *         quorum
 */
private Integer uniqueQuorum(SortedMap<Integer, QuorumData> quorums)
{
        boolean quorumDetected = false;
        Integer uniqueQuorum = null;

        for (Map.Entry<Integer, QuorumData> e : quorums.entrySet())
        {
                if (e.getValue().getAgentSetSize() > quorumThreshold)
                {
                        if (!quorumDetected) // first quorum found
                        {
                                uniqueQuorum = e.getKey();
                                quorumDetected = true;
                        }
                        else
                        // more than one quorum detected => not unique
                        {
                                return null;
                        }
                }
        }
        return uniqueQuorum;
}

/**
 * Step method called on every iteration of the simulation.
 */
public void step()
{
        // collect QuorumData objects from all neighbors, but only
        // merge those that agree with this agent's current choice,
        // keeping duplicates with the greatest remaining duration

        // shuffle the neighbor polling
        // TODO: can this be made more efficient rather than doing it on
        // every step?
        List<Agent> neighborList =
                        Arrays.asList(neighbors.toArray(new Agent[0]));
        SimUtilities.shuffle(neighborList, RandomHelper.getUniform());
        for (Agent neighbor : neighborList)
        {
```

```
                QuorumData receivedQuorum = neighbor.getQuorumData();
                if (receivedQuorum.getChoiceID() == choice)
                {
                        mergeQuorums(receivedQuorum, prefQuorum);
                }
        }
}

/**
 * Decays the values in the quorum data object. When quorum data object
 * values expire, remove them from the set.
 */
public void decayQuorum()
{
        // (This was added *after* HICSS code)
        prefQuorum.decayAgentDurations(decayFunc, evapThreshold);
}

/**
 * Decays the agent duration. When it expires, set prefQuorum to null and
 * choiceID to 0.
 */
public void decay()
{
        duration = decayFunc.execute(duration);
        if (duration <= evapThreshold)
        {
                choice = 0;

                SortedMap<Integer, QuorumData> neighborQuorums =
                                new TreeMap<Integer, QuorumData>();

                SortedMap<Integer, Integer> prefCount =
                                new TreeMap<Integer, Integer>();

                // collect QuorumData objects from all neighbors
                for (Agent neighbor : neighbors)
                {
                        QuorumData receivedQuorum = neighbor.getQuorumData();
                        Integer neighborChoice =
                                receivedQuorum.getChoiceID();

                        // keep count of how many prefer each choice
                        if (prefCount.containsKey(neighborChoice))
                        {
                                // this choice has a quorum data object in the
                                // collection,
                                // so we just need to increase its counter
                                prefCount.put(neighborChoice,
                                                prefCount.get(neighborChoice) +
                                                                1);
                        }
                        else
                        {
                                // this choice doesn't have a quorum data
                                // object in the
                                // collection, so initialize the counter
                                prefCount.put(neighborChoice, 1);

                                // and add an empty quorum object to the
```

```
                        // collection for this choice

                        QuorumData destQuorum = new
                                QuorumData(neighborChoice);
                        neighborQuorums.put(neighborChoice,
                                        destQuorum);
                }

                mergeQuorums(receivedQuorum,
                                neighborQuorums.get(neighborChoice));
        }

        // check for unique quorum and, if found, select it
        Integer unique = null;
        if ((unique = uniqueQuorum(neighborQuorums)) != null)
        {
                // commit to unique quorum found
                choice = unique;
        }
        else
        {
                // peform fitness proportionate selection (roulette
                // wheel)
                double r = RandomHelper.nextDouble();
                double wheelVal = 0.0;
                choice = 0;
                for (Map.Entry<Integer, Integer> e :
                        prefCount.entrySet())
                {
                        wheelVal +=
                                        (double) e.getValue() /
                                        (double) neighbors.size();
                        if (r < wheelVal)
                        {
                                choice = e.getKey();
                                break;
                        }
                }

//              try
//              {
//                      if (choice == 0) throw new
//                              NullPointerException("Choice =
//                                                      0 ERROR");
//              }
//              catch (NullPointerException e)
//              {
//                      System.out.println(e.getMessage());
//                      System.out.println(prefCount.toString());
//              }
        }

        // update quorum and duration based on committment
        prefQuorum = neighborQuorums.get(choice);

        // get this agent's duration for the new choice and add it
        // to the quorum
//      try
//      {
                duration = init.execute(decisionPrefs.get(choice));
```

```
//                  }
//                  catch (NullPointerException e)
//                  {
//                        System.out.println("Agent: " + this.agentID);
//                        System.out.print("Choice: " + choice + " = ");
//                        if (decisionPrefs.containsKey(choice))
//                        {
//                              System.out.println(decisionPrefs.get(choice));
//                        }
//                        else
//                        {
//                              System.out.println("null");
//                        }
//                  }
                    prefQuorum.addAgent(this);
              }
        }

        @Override
        public int compareTo(Agent o)
        {
              return this.agentID - o.agentID;
        }

}
```

C.1.2. Population Class (Population.java)

```
/**
 * A collection of Agents, the collection's global or aggregate properties, and
 * methods for their manipulation. In order to ensure that individual Agent
 * preferences are consistent with the distribution we want in the population,
 * preferences for agents are created and stored here, then supplied to the
 * agent at instantiation.
 */
package swarmdm.agents;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Date;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.NavigableMap;
//////////////////////////////////////////////////////////////////////////////
//import java.util.Random;
//////////////////////////////////////////////////////////////////////////////
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;

import repast.simphony.engine.environment.RunEnvironment;
import repast.simphony.parameter.Parameters;
import repast.simphony.random.RandomHelper;
import repast.simphony.util.SimUtilities;
import swarmdm.common.Constants;
import swarmdm.common.Utils;

/**
 * @author alexander.mentis
 *
 */
public class Population
{
	// preference data for global analysis
	private double[][] rawPrefs = null;
	private int[][] prefRanks = null;

	// ////////////////////////////////////////////////////////////////////////
	// // Random generator seed for generator from java.util, separate from
	// the one
	// // used by Repast for simulation operations, in order to separate
	// // the random preference value generation from randomized agent
	// // behavior. This allows us to generate multiple runs that use the
	// // same preference data but allow the agent behavior between those
	// // runs to vary stochastically.
	// private long dataSeedParam = 0L;
	// private Random rand = null;
	// ////////////////////////////////////////////////////////////////////////

	// members of the population
```

```
private ArrayList<Agent> members = new ArrayList<Agent>();

// output file data
private String fnPattern = null;
private String filename = null;
private PrintWriter write = null;
// ########################################################################
// private PrintWriter graphFile = null;
// ########################################################################

// parameters used to create this population
private Parameters parameters = null;

public Population(Parameters parameters)
{
	this.parameters = parameters;

	this.rawPrefs =
			new double[Constants.CHOICE_COUNT][((Integer)
				parameters
				.getValue(Constants.PARAM_AGENT_COUNT))
				.intValue()];
	this.prefRanks =
			new int[Constants.CHOICE_COUNT][((Integer) parameters
			.getValue(Constants.PARAM_AGENT_COUNT)).intValue()];
	// ########################################################################
	// long dataSeedParamInput =
	// ((Long) parameters.getValue(Constants.PARAM_DATA_RAND_SEED))
	// .longValue();
	//
	//
	// this.dataSeedParam = dataSeedParamInput == 0L
	// ? System.currentTimeMillis()
	// : dataSeedParamInput;
	// this.rand = new Random(dataSeedParam);
	// ########################################################################

	int numAgents =
			((Integer)
			parameters.getValue(Constants.PARAM_AGENT_COUNT))
			.intValue();

	// create preferences for each agent in accordance with desired
	// distribution
	ArrayList<Double> pmf = genUniformPMF(Constants.CHOICE_COUNT);

	// common agent-creation parameters
	int quorumThreshold =
			((Integer) parameters
			.getValue(Constants.PARAM_QUORUM_THRESHOLD))
			.intValue();
	double decayRate =
			((Double)
			parameters.getValue(Constants.PARAM_DECAY_RATE))
			.doubleValue();
	double evaporationThreshold =
			((Double)
			parameters.getValue(Constants.PARAM_EVAP_THRESHOLD))
			.doubleValue();
```

```
for (int agentID = 0; agentID < numAgents; ++agentID)
{
       Map<Integer, Double> prefs = genPrefs(pmf,
                                          Constants.CHOICE_COUNT);

       members.add(new Agent(prefs, quorumThreshold, decayRate,
                     evaporationThreshold, agentID));
}

// ###############################################################
// # Don't believe this is necessary, since the topology for a
// # given preference distribution changes with each topology
// # class.
//
// // shuffle Agent locations in the network
// SimUtilities.shuffle(members, RandomHelper.getUniform());
// ###############################################################

// Record preferences. Note: the preferences will be recorded
// as they relate to the location of the agent in the network
// (agentIndex) rather than the agent's unique identifier
// (agentID).
for (int agentIndex = 0; agentIndex < members.size();
       ++agentIndex)
{
       storePrefs(members.get(agentIndex).getPreferences(),
                     agentIndex);
}

// write settings out to file
String Q =
              ((Integer) parameters
              .getValue(Constants.PARAM_QUORUM_THRESHOLD))
              .toString();
String A =    String.format("%02d", (Integer) parameters
              .getValue(Constants.PARAM_SIM_RAND_SEED));
String P =    String.format("%02d", (Integer) parameters
              .getValue(Constants.PARAM_WS_P_VALUE));
// ###############################################################
// String D =
// String.format("%02d", (Long) parameters
// .getValue(Constants.PARAM_DATA_RAND_SEED));
// ###############################################################
this.fnPattern = "yyyy-MM-dd_HH.mm.ss'-Q" + Q + "P" + P + "A" + A
                     + "'";
this.filename = new SimpleDateFormat(fnPattern).format(new
                     Date());

try
{
       this.write = new PrintWriter(new File(filename + ".dat"));
       // ###########################################################
       // this.graphFile = new PrintWriter(new File(filename +
       // "graph.dat"));
       // ###########################################################
}
catch (FileNotFoundException e)
{
       // TODO Auto-generated catch block
       e.printStackTrace();
```

```
        }

        writeInitialData(pmf);
    }

    // accessors and mutators
    /**
     * Get a copy of the population member list.
     *
     * @return a copy of the population member list
     */
    public ArrayList<Agent> getMembers()
    {
        return new ArrayList<Agent>(members);
    }

    // population creation methods

    // Generate population preferences according to a probability mass
    // function.
    private Map<Integer, Double>
                genPrefs(ArrayList<Double> pmf, int numChoices)
    {
        // generate a uniform pmf of preferences, sorted ascending
        ArrayList<Double> initialPrefs = genUniformPMF(numChoices);
        Collections.sort(initialPrefs);

        Map<Integer, Double> finalPrefs = new LinkedHashMap<Integer,
                                                            Double>();
        NavigableMap<Integer, Double> pmfMap = new TreeMap<Integer,
                                                            Double>();
        for (int i = 0; i < pmf.size(); ++i)
        {
            pmfMap.put(i + 1, pmf.get(i));
        }

        // assign the initial preferences to a choice value in accordance
        // with
        // the provided PMF such that the highest-valued initial
        // preference
        // is most likely to be in the slot that should be chosen first
        // most
        // frequently according to the PMF
        while (!pmfMap.isEmpty())
        {
            // ########################################################
            // int assignedChoice =
            // Utils.rouletteSelect(pmfMap, rand.nextDouble());
            int assignedChoice =
                        Utils.rouletteSelect(pmfMap,
                                RandomHelper.nextDouble());
            // ########################################################
            finalPrefs.put(assignedChoice,
                        initialPrefs.get(initialPrefs.size() - 1));
            initialPrefs.remove(initialPrefs.size() - 1);
            pmfMap.remove(assignedChoice);
        }

        return finalPrefs;
    }
```

```
// Generate a probability mass function from a uniform distribution for a
// specified number of discrete choices using the Technique from Smith04:
// Sampling Uniformly from the Unit Simplex.
private ArrayList<Double> genUniformPMF(int numChoices)
{
	TreeSet<Integer> distVector = new TreeSet<Integer>();

	int lo = 0;
	int hi = 100;

	// generate numChoices - 1 random values, without replacement, in
	// the desired range
	while (distVector.size() < numChoices - 1)
	{
		// constrain values to lo+1..hi-1
		// ##########################################################
		// distVector.add(rand.nextInt((hi - lo) - 1) + (lo + 1));
		distVector.add(RandomHelper.nextIntFromTo(lo + 1, hi - 1));
		// ##########################################################
	}

	ArrayList<Double> pmf = new ArrayList<Double>();

	int j = lo;
	for (Integer i : distVector)
	{
		// each choice's probabilistic weight is the proportion of
		// the range between lo and hi that the segment occupies
		pmf.add((double) (i - j) / (double) (hi - lo));
		j = i;
	}
	pmf.add((double) (hi - j) / (double) (hi - lo));

	return pmf;
}

// Place initial preference data into arrays for global analysis.
private void storePrefs(Map<Integer, Double> prefs, int agentIndex)
{
	// Maintain a list of preference values sorted in reverse order.
	// By finding the index of the first appearance of a value, we
	// can determine its order in the list of preferences. Tied values
	// receive the same rank, which is what we want for the Borda
	// count and other rank-based voting systems.
	ArrayList<Double> sortedPrefs = new
				ArrayList<Double>(prefs.values());
	Collections.sort(sortedPrefs, Collections.reverseOrder());

	for (int i : prefs.keySet())
	{
		rawPrefs[i - 1][agentIndex] = prefs.get(i);
		prefRanks[i - 1][agentIndex] =
				sortedPrefs.indexOf(prefs.get(i)) + 1;
	}
}

// Write initial preferences and resulting outcomes for various voting
// methods to an output file.
private void writeInitialData(ArrayList<Double> pmf)
```

```
{
        // Write global preference data to file and generate results from
        // other voting methods (Condorcet, simple plurality, range
        // voting, Borda count).

        // parameters
        write.println("Random Seed (agent actions): "
                    + ((Integer)
                    parameters.getValue(Constants.PARAM_SIM_RAND_SEED))
                                .toString());
        // ###################################################################
        // write.println("Random Seed (data)         : " + dataSeedParam);
        // ###################################################################
        write.println("Agent Count                : "
                    + ((Integer)
                    parameters.getValue(Constants.PARAM_AGENT_COUNT))
                                .toString());
        write.println("Quorum Size Threshold      : "
                    + ((Integer) parameters
                    .getValue(Constants.PARAM_QUORUM_THRESHOLD))
                    .toString());
        write.println("Evaporation Threshold      : "
                    + ((Double)
                    parameters.getValue(Constants.PARAM_EVAP_THRESHOLD))
                                .toString());

//#########################################################################
        // write.println("Minimum Forward Degree     : "
        // + ((Integer) parameters
        // .getValue(Constants.PARAM_MIN_FWD_DEGREE)).toString());
        // write.println("Maximum Forward Degree     : "
        // + ((Integer) parameters
        // .getValue(Constants.PARAM_MAX_FWD_DEGREE)).toString());

//#########################################################################

        // agent pref data
        for (int agentIndex = 0; agentIndex < members.size();
                ++agentIndex)
        {
                write.print(agentIndex + "; ");
                for (int prefNum = 0; prefNum < Constants.CHOICE_COUNT;
                        ++prefNum)
                {
                        write.print(rawPrefs[prefNum][agentIndex] + "; ");
                }

                for (int prefNum = 0; prefNum < Constants.CHOICE_COUNT - 1;
                        ++prefNum)
                {
                        write.print(prefRanks[prefNum][agentIndex] + "; ");
                }
                write.println(prefRanks[Constants.CHOICE_COUNT -
                                        1][agentIndex]);
        }

        // leave a space for summary data in Excel
        write.println();

        // Print the target choice PMF used
```

```
        write.print("PMF: ");
        for (Double d : pmf)
        {
               write.print(d + ", ");
        }
        write.println();
        // summary data
        write.println("Simple plurality winners: " +
                      calcPluralityWinners());
        write.println("Range voting winners:     " +
                      calcRangeVoteWinners());
        write.println("Borda count winners:      " +
                      calcBordaCountWinners());
}

// ########################################################################
// /**
// * Write the adjacency list representation of the population graph to
// * the data file and send it to networkx for graph property analysis.
// */
// public void writeGraph()
// {
// // Send the adj list data to networkx in an external file
//
// for (Agent agent : members)
// {
// graphFile.print(agent.getAgentID() + " ");
// for (Agent neighbor : agent.getNeighbors())
// {
// graphFile.print(neighbor.getAgentID() + " ");
// }
// graphFile.println();
// }
// graphFile.close();
// }
// ########################################################################

// ########################################################################
// /**
// * Call external script to calculate metrics, then write them to the
// * graphFile.
// */
// public void calcGraphMetrics()
// {
// // external call to Python script with filename + graph.dat
// try
// {
// Runtime r = Runtime.getRuntime();
// Process p =
// r.exec("python graph_stats.py " + filename + "graph.dat");
// p.waitFor();
// }
// catch (Exception e)
// {
// e.printStackTrace();
// }
// }
// ########################################################################

/**
```

```
 * Write out the final selected choice and time required.
 */
public void writeFinalData()
{
	write.println("Final choice: " + members.get(0).getChoice());
	write.println("Ticks required: "
			+ RunEnvironment.getInstance().getCurrentSchedule()
					.getTickCount());
	write.close();
}

/**
 * Write data indicating that no consensus was reached within the
 * allotted time.
 */
public void writeAborted()
{
	write.println("Final choice: 0");
	write.println("Ticks required: "
			+ RunEnvironment.getInstance().getCurrentSchedule()
					.getTickCount());
	write.println("Run timed out");
	write.close();
}

// Calculates the winner based on initial preferences using a Condorcet
// method.
// private String calcCondorcetWinners(int[][] prefRanks)
// {
// // TODO Auto-generated method stub
// return null;
// }

// Calculates the winner based on initial preferences using the Borda
// count.
private String calcBordaCountWinners()
{
	int max = 0;
	int[] bordaCounts = new int[prefRanks.length];
	Arrays.fill(bordaCounts, 0);

	for (int pref = 0; pref < prefRanks.length; ++pref)
	{
		for (int agent = 0; agent < prefRanks[pref].length;
				++agent)
		{
			bordaCounts[pref] += prefRanks.length -
					prefRanks[pref][agent];
		}
		max = Math.max(max, bordaCounts[pref]);
	}

	String results = new String("");
	for (int pref = 0; pref < bordaCounts.length; ++pref)
	{
		if (bordaCounts[pref] == max)
		{
			if (!results.isEmpty())
			{
				results = results.concat(", ");
```

```
                    }
                    results = results.concat(Integer.toString(pref + 1));
                }
        }
        return results;
}

// Calculates the winner based on initial preferences using range voting.
private String calcRangeVoteWinners()
{
        double max = 0.0;
        double[] avg = new double[rawPrefs.length];

        for (int pref = 0; pref < rawPrefs.length; ++pref)
        {
                double sum = 0.0;
                for (int agent = 0; agent < rawPrefs[pref].length; ++agent)
                {
                        sum += rawPrefs[pref][agent];
                }
                avg[pref] = sum / (double) rawPrefs[pref].length;
                max = Math.max(avg[pref], max);
        }

        String results = new String("");
        for (int pref = 0; pref < avg.length; ++pref)
        {
                if (avg[pref] == max)
                {
                        if (!results.isEmpty())
                        {
                                results = results.concat(", ");
                        }
                        results = results.concat(Integer.toString(pref + 1));
                }
        }
        return results;
}

// Calculates the winner based on initial preferences using simple
// plurality.
private String calcPluralityWinners()
{
        int max = 0;
        int[] votes = new int[prefRanks.length];

        for (int pref = 0; pref < prefRanks.length; ++pref)
        {
                int winCount = 0;
                for (int agent = 0; agent < prefRanks[pref].length;
                        ++agent)
                {
                        if (prefRanks[pref][agent] == 1)
                        {
                                ++winCount;
                        }
                }
                votes[pref] = winCount;
                max = Math.max(votes[pref], max);
        }
```

```
            String results = new String("");
            for (int pref = 0; pref < votes.length; ++pref)
            {
                  if (votes[pref] == max)
                  {
                        if (!results.isEmpty())
                        {
                              results = results.concat(", ");
                        }
                        results = results.concat(Integer.toString(pref + 1));
                  }
            }
            return results;
      }

      /**
       * Determine unanimity of choice.
       *
       * @param agents
       *            the individuals to check for agreement
       * @return <code>true</code> if all individuals have the same choice
       *     value;
       *          <code>false</code> otherwise
       */
      public boolean unanimousChoice()
      {
            Set<Integer> uniqueChoices = new TreeSet<Integer>();
            for (Agent a : members)
            {
                  uniqueChoices.add(a.getChoice());
            }

            return uniqueChoices.size() == 1;
      }
}
```

C.1.3. Quorum Data Class (QuorumData.java)

```
package swarmdm.agents;

import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Set;

import swarmdm.strategies.DecayFunction;

/**
 * A structure associated with a candidate choice and containing a set of
 * agents committed to that choice. Agents use the number of agents in the
 * set of committed agents to detect when a quorum has been reached.
 */
public class QuorumData
{
	/**
	 * The ID of the candidate choice all agents in this object's agent set
	 * were committed to when they joined this group.
	 */
	private int choiceID;

	/**
	 * A map of a set of agents in the group to each agent's duration until
	 * membership in the set expires.
	 */
	private LinkedHashMap<Integer, Double> agentSet;

	/**
	 * Constructs a new QuorumData object with the id of the choice it
	 * represents and the agents that prefer that choice at the time of
	 * creation.
	 *
	 * @param choiceID
	 *            the identifier of the candidate choice this QuorumData
	 *            object represents
	 * @param agentSet
	 *            the map of the set of agents included in this group to
	 *            their membership expiration tick
	 */
	public QuorumData(int quorumID)
	{
		this.choiceID = quorumID;
		this.agentSet = new LinkedHashMap<Integer, Double>();
	}

	/**
	 * Returns the id of the choice preferred by all members of this group.
	 *
	 * @return the choiceID preferred by all members of this group
	 */
	public int getChoiceID()
	{
		return choiceID;
	}

	/**
```

```
 * @return the agentSet
 */
public LinkedHashMap<Integer, Double> getAgentSet()
{
	return agentSet;
}

/**
 * Returns the size of the agent set.
 *
 * @return the agent set size
 */
public int getAgentSetSize()
{
	return agentSet.size();
}

/**
 * Gets the entry set (set of agent ID to duration mappings) of the
 * agentSet contained in this QuorumData object.
 *
 * @return an entry set backed by the agentSet's LinkedHashMap
 */
public Set<Map.Entry<Integer, Double>> entrySet()
{
	return agentSet.entrySet();
}

/**
 * Adds an agent to this QuorumData's agentSet.
 *
 * @param a
 *            the agent
 */
public void addAgent(Agent a)
{
	agentSet.put(a.getAgentID(), a.getDuration());
}

/**
 * Applies the provided decay function to the duration of all agents in
 * this quorum and removes an agent when its duration reaches the
 * specified evaporation threshold.
 *
 * @param decay
 *            a function implementing the {@link DecayFunction
 *            DecayFunction} interface
 * @param evapThreshold
 *            the duration value at or below which an agent is removed
 *            from the agentSet
 */
public void decayAgentDurations(DecayFunction decay,
		double evapThreshold)
{
	Iterator<Map.Entry<Integer, Double>> iter =
			agentSet.entrySet().iterator();
	while (iter.hasNext())
	{
		Map.Entry<Integer, Double> e = iter.next();
		double newDuration = decay.execute(e.getValue());
```

```
                        if (newDuration <= evapThreshold)
                        {
                              iter.remove();
                        }
                        else
                        {
                              e.setValue(newDuration);
                        }
                  }
            }
      }
```

C.2. Agent Strategies

C.2.1. Decay Function (DecayFunction.java)

```
package swarmdm.strategies;

/**
 * Specifies the interface of a decay function so that a function for
 * degrading agent commitment durations can be parameterized.
 */

/**
 * @author alexander.mentis
 *
 */
public interface DecayFunction
{
	/**
	 * Executes the decay function.
	 */
	public abstract double execute(double duration);
}
```

C.2.2. Linear Decay Strategy (LinearDecay.java)

```
package swarmdm.strategies;


/**
 * Decays the provided duration at a linear rate
 */

/**
 * @author alexander.mentis
 */
public class LinearDecay implements DecayFunction
{
	private double rate;

	public LinearDecay(double rate)
	{
		this.rate = rate;
	}

	/*
	 * (non-Javadoc)
	 *
	 * @see DecayFunction#execute(double)
	 */
	@Override
	public double execute(double duration)
	{
		return duration - rate;
	}
}
```

C.2.3. Initialization Function (InitFunction.java)

```
package swarmdm.strategies;

/**
 * Specifies the interface of a conversion function so that a function for
 * converting preference values to commitment durations can be parameterized.
 */

/**
 * @author alexander.mentis
 *
 */
public interface InitFunction
{
	/**
	 * Converts a preference value to a new double that represents the
	 * commitment duration.
	 *
	 * @param prefVal
	 *            the preference value to be converted
	 * @return the commitment duration value
	 */
	public abstract double execute(double prefVal);
}
```

C.2.4. Preference Value Conversion (ProbabilityToDouble.java)

```
package swarmdm.strategies;


/**
 * Takes a preference value expressed as a probability and converts it to a
 * double value that can be used as a commitment duration by multiplying it by
 * 100.
 */

/**
 * @author alexander.mentis
 *
 */
public class ProbabilityToDouble implements InitFunction
{
	/*
	 * (non-Javadoc)
	 *
	 * @see ConvertFunction#execute(double)
	 */
	@Override
	public double execute(double prefVal)
	{
		return prefVal * 100.0;
	}
}
```

C.3. Simulation Framework Integration and Utilities

C.3.1. Simulation Framework Context Builder (SwarmDMBuilder.java)

```
/**
 * Context builder for initializing the simulation.
 */
package swarmdm;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Scanner;
import java.util.Set;

import repast.simphony.context.Context;
import repast.simphony.context.DefaultContext;
import repast.simphony.context.space.continuous.ContinuousSpaceFactory;
import repast.simphony.context.space.continuous.ContinuousSpaceFactoryFinder;
import repast.simphony.context.space.graph.NetworkBuilder;
import repast.simphony.dataLoader.ContextBuilder;
import repast.simphony.engine.environment.RunEnvironment;
import repast.simphony.engine.schedule.ISchedule;
import repast.simphony.engine.schedule.ScheduleParameters;
import repast.simphony.parameter.Parameters;
import repast.simphony.random.RandomHelper;
import repast.simphony.space.continuous.ContinuousSpace;
import repast.simphony.space.continuous.SimpleCartesianAdder;
import repast.simphony.space.graph.Network;
import repast.simphony.util.SimUtilities;
import swarmdm.agents.Agent;
import swarmdm.agents.Population;
import swarmdm.common.Constants;
import swarmdm.common.SpaceCoord;

/**
 * @author alexander.mentis
 *
 */
public class SwarmDMBuilder extends DefaultContext<Object> implements
              ContextBuilder<Object>
{
       private Population population = null;

       /*
        * (non-Javadoc)
        *
        * @see
        *
repast.simphony.dataLoader.ContextBuilder#build(repast.simphony.context
        * .Context)
        */
       @Override
       public Context<Object> build(Context<Object> context)
       {
              context.setId(Constants.CONTEXT_ID);
```

```
final Parameters parameters =
            RunEnvironment.getInstance().getParameters();

ContinuousSpaceFactory spaceFactory =
            ContinuousSpaceFactoryFinder
            .createContinuousSpaceFactory(null);

ContinuousSpace<Object> space =
            spaceFactory.createContinuousSpace(
                                        Constants.SPACE_ID,
                                        context,
                                        new
                                SimpleCartesianAdder<Object>(),
                                        new
            repast.simphony.space.continuous.WrapAroundBorders(),
                                        Constants.SPACE_WIDTH,
                                        Constants.SPACE_HEIGHT);

NetworkBuilder<Object> netBuilder =
            new NetworkBuilder<Object>(Constants.NETWORK_ID,
                  context, false);

netBuilder.buildNetwork();

@SuppressWarnings("unchecked")
Network<Object> net =
            (Network<Object>)
            context.getProjection(Constants.NETWORK_ID);

// generate all agents according to parameters; includes
// generating preferences and recording them

population = new Population(parameters);

// place agents from population in context, visualization, and
// network
ArrayList<Agent> agents = population.getMembers();

try
{
      createNetwork(agents, parameters);
}
catch (FileNotFoundException e)
{
      // TODO Auto-generated catch block
      e.printStackTrace();
}
// ###################################################################
// population.writeGraph();
// population.calcGraphMetrics();
// ###################################################################

int agentNum = 0;
for (Agent a : agents)
{
      context.add(a);

      SpaceCoord coord =
                  getCoord(agentNum,
                        ((Integer) parameters
```

```
                                    .getValue(Constants.PARAM_AGENT_COUNT))
                                    .intValue(), Constants.SPACE_WIDTH,
                                    Constants.SPACE_HEIGHT);
                space.moveTo(a, coord.x, coord.y);

                for (Agent neighbor : a.getNeighbors())
                {
                        net.addEdge(a, neighbor);
                }

                ++agentNum;
        }

        // XXX: Schedule the step() method manually due to some bug
        // preventing
        // @ScheduledMethods
        // According to RepastInterest, the following fix also works:
        // add the line
        // class="basicModel.BasicModelContextBuilder"
        // to context.xml. There may be examples in the demo files.
        ISchedule schedule =
                RunEnvironment.getInstance().getCurrentSchedule();
                schedule.schedule(ScheduleParameters.createRepeating(1, 1,
                0), this, "step");

        return context;
}

// Place each agent on the perimeter of a circle.
static private SpaceCoord getCoord(int agentID, int numAgents,
                double width, double height)
{
        double angleIncrement = 360.0 / (double) numAgents;
        double radius = (Math.min(width, height) * 0.95) * 0.5;

        return new SpaceCoord((0.5 * width)
                        + (radius * Math.cos((angleIncrement * agentID)
                                        * (Math.PI / 180.0))), (0.5 * height)
                        + (radius * Math.sin((angleIncrement * agentID)
                                        * (Math.PI / 180.0))));
}

// Read the adjacency list created by NetworkX from a file and link
// agents in accordance with it.
private void createNetwork(ArrayList<Agent> members, Parameters
                                parameters)
                throws FileNotFoundException
{

        String path = null;

        switch ((Integer) parameters.getValue(Constants.PARAM_NETWK_TYPE))
        {
                case 0:

                        // reserved for one-off random runs

                        // ################################################
                        // // connect to next n agents, where n is between
                        // // PARAM_MIN_DEGREE and PARAM_MAX_DEGREE,
```

```
                // // inclusive
                // for (int agentIndex = 0; agentIndex <
                //           members.size();
                // ++agentIndex)
                // {
                // Agent thisAgent = members.get(agentIndex);
                //
                // int fwdNeighbors =
                // RandomHelper.nextIntFromTo(((Integer) parameters
                // .getValue(Constants.PARAM_MIN_FWD_DEGREE))
                // .intValue(), ((Integer) parameters
                // .getValue(Constants.PARAM_MAX_FWD_DEGREE))
                // .intValue());
                //
                // for (int neighborNum = 1; neighborNum <=
                //    fwdNeighbors;
                // ++neighborNum)
                // {
                // Agent neighbor =
                // members.get((agentIndex + neighborNum)
                // % members.size());
                // thisAgent.addNeighbor(neighbor);
                // }
                // }
                // ###################################################
                break;
            case 1: // Erdos-Renyi random network

                path =
                            "./data/graphs/er"
                                        + String.format(
                                                    "%02d",
                                                    (Integer)
                                                    parameters
                            .getValue(Constants.PARAM_WS_P_VALUE))
                                        + ".graph";

                // ###################################################
                // // create random, connected, graph, where each
                // // node has at
                // least
                // // PARAM_MIN_DEGREE neighbors
                // int minDeg =
                // (Integer) parameters
                // .getValue(Constants.PARAM_MIN_FWD_DEGREE);
                //
                // // use random walk to ensure all nodes are
                // // connected
                // Set<Integer> connectedSet = new
                //                                HashSet<Integer>();
                // connectedSet.add(0);
                // Integer currentMember = 0;
                // while (connectedSet.size() != members.size())
                // {
                // int j = RandomHelper.nextIntFromTo(0,
                //           members.size() - 1);
                // if (!connectedSet.contains(j))
                // {
                // Agent iAgent = members.get(currentMember);
                // Agent jAgent = members.get(j);
```

```
                // iAgent.addNeighbor(jAgent);
                // jAgent.addNeighbor(iAgent);
                // connectedSet.add(j);
                // }
                // currentMember = j;
                // }
                //
                // // ensure each node has the minimum number of
                // // links
                // for (int i = 0; i < members.size(); ++i)
                // {
                // Agent iAgent = members.get(i);
                // while (iAgent.neighborCount() < minDeg)
                // {
                // int j =
                // RandomHelper.nextIntFromTo(0,
                // members.size() - 1);
                //
                // Agent jAgent = members.get(j);
                // if (iAgent != jAgent
                // && !iAgent.getNeighbors().contains(jAgent))
                // {
                // iAgent.addNeighbor(jAgent);
                // jAgent.addNeighbor(iAgent);
                // }
                // }
                // }
                // ##################################################
                break;
        case 2: // Barabasi-Albert scale-free network

                path =
                                "./data/graphs/sf"
                                                + String.format(
                                                                "%02d",
                                                                (Integer)
                                                                parameters
                                .getValue(Constants.PARAM_WS_P_VALUE))
                                                + ".graph";
                break;
        case 3: // Watts-Strogatz small-world network

                path =
                                "./data/graphs/sw"
                                                + String.format("%02d",
                                                (Integer) parameters
                                .getValue(Constants.PARAM_WS_P_VALUE))
                                                + ".graph";
                break;
}

Scanner in = new Scanner(new File(path));

String line;
while ((line = in.nextLine()) != null)
{
        if (line.startsWith("# Edges"))
        {
                break;
        }
```

```
                else if (line.startsWith("#"))
                {
                        continue;
                }

                String[] adjList = line.split(" ");
                Agent i = members.get(Integer.valueOf(adjList[0]));
                for (int k = 1; k < adjList.length; ++k)
                {
                        Agent j = members.get(Integer.valueOf(adjList[k]));

                        i.addNeighbor(j);
                        j.addNeighbor(i);
                }
        }

        in.close();
}

// Simulation step function used to synchronize events. Easier to manage
// than individually-scheduled methods.
// @ScheduledMethod(start = 1, interval = 1, priorty = 0)
// (Scheduled manually in the context builder.)
public void step()
{
        List<Agent> agents = population.getMembers();

        // agent step
        SimUtilities.shuffle(agents, RandomHelper.getUniform());
        for (Agent a : agents)
        {
                a.step();
        }

        // decay all quorums at the same time to keep them in sync
        for (Agent a : agents)
        {
                a.decayQuorum();
        }

        // decay agent
        SimUtilities.shuffle(agents, RandomHelper.getUniform());
        for (Agent a : agents)
        {
                a.decay();
        }

        // stop simulation when agreement is reached
        if (population.unanimousChoice())
        {
                population.writeFinalData();
                RunEnvironment.getInstance().endRun();
        }
        else if (RunEnvironment.getInstance().getCurrentSchedule()
                        .getTickCount() == Constants.TICK_COUNT_LIMIT)
        {
                population.writeAborted();
                RunEnvironment.getInstance().endRun();
        }
}
```

```
}
```

C.3.2. Constants (Constants.java)

```
/**
 * Central repository for simulation constants.
 */
package swarmdm.common;

/**
 * @author alexander.mentis
 *
 */
public class Constants
{
	public static final String CONTEXT_ID = "swarmdm";
	public static final String SPACE_ID = "space";
	public static final String NETWORK_ID = "neighborhood";

	public static final String PARAM_SIM_RAND_SEED = "randomSeed";
	public static final String PARAM_DATA_RAND_SEED = "dataRandSeed";
	public static final String PARAM_AGENT_COUNT = "agentCount";
	public static final String PARAM_QUORUM_THRESHOLD = "quorumThreshold";
	public static final String PARAM_DECAY_RATE    = "decayRate";
	public static final String PARAM_EVAP_THRESHOLD = "evapThreshold";
	public static final String PARAM_MIN_FWD_DEGREE = "minFwdDeg";
	public static final String PARAM_MAX_FWD_DEGREE = "maxFwdDeg";
	public static final String PARAM_NETWK_TYPE = "netwkType";
	public static final String PARAM_WS_P_VALUE = "pVal";

	public static final double SPACE_WIDTH = 80.0;
	public static final double SPACE_HEIGHT = 80.0;

	public static final int CHOICE_COUNT = 10;

	public static final int TICK_COUNT_LIMIT = 3000;
}
```

C.3.3. Spatial Coordinate Container (SpaceCoord.java)

```
/**
 * Contains the coordinates in a continuous space.
 */
package swarmdm.common;

/**
 * @author alexander.mentis
 *
 */
public class SpaceCoord
{
	public double x;
	public double y;

	public SpaceCoord(double x, double y)
	{
		this.x = x;
		this.y = y;
	}
}
```

C.3.4. Utility Calculation Method Class (Utils.java)

```
/**
 * A collection of common utilities for Swarm Decision Making
 */
package swarmdm.common;

import java.util.Map;

/**
 * @author alexander.mentis
 *
 */
public class Utils
{
	/**
	 * Applies fitness proportionate selection to a map of integers to
	 * fitness values using roulette wheel selection.
	 *
	 * @param fitnessMap
	 *            a map of integers to fitness values
	 * @param randomValue
	 *            a random value used to select one of the integers in the
	 *            map
	 * @return the integer key of the winning fitness value or null if none
	 *         were selected
	 */
	static public Integer rouletteSelect(Map<Integer, Double> fitnessMap,
			double randomValue)
	{
		// get the sum of fitnesses for normalization
		Double fitnessSum = 0.0;
		for (Double value : fitnessMap.values())
		{
			fitnessSum += value;
		}

		Double fitness = 0.0;
		for (Map.Entry<Integer, Double> e : fitnessMap.entrySet())
		{
			fitness += e.getValue();
			if (randomValue < fitness / fitnessSum)
			{
				return e.getKey();
			}
		}
		return null;
	}
}
```

C.3.5. Agent Visualization Color Control (AgentStyle.java)

```
package swarmdm.observer;

import java.awt.Color;

import repast.simphony.visualizationOGL2D.DefaultStyleOGL2D;
import saf.v3d.ShapeFactory2D;
import saf.v3d.scene.VSpatial;
import swarmdm.agents.Agent;

public class AgentStyle extends DefaultStyleOGL2D
{
	private ShapeFactory2D shapeFactory;

	@Override
	public void init(ShapeFactory2D factory)
	{
		this.shapeFactory = factory;
	}

	@Override
	public Color getColor(Object o)
	{
		Agent agent = (Agent) o;
		switch (agent.getChoice())
		{
			case 1:
				return Color.BLUE;
			case 2:
				return Color.CYAN;
			case 3:
				return Color.GREEN;
			case 4:
				return Color.MAGENTA;
			case 5:
				return Color.ORANGE;
			default:
				return Color.WHITE;
		}
	}

	@Override
	public VSpatial getVSpatial(Object agent, VSpatial spatial)
	{
		if (spatial == null)
		{
			spatial = shapeFactory.createCircle(7.5f, 20);
		}
		return spatial;
	}
}
```