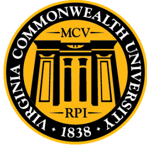


Virginia Commonwealth University
VCU Scholars Compass

Theses and Dissertations

Graduate School

2006

K x N Trust-Based Agent Reputation

Christopher Alonzo Parker
Virginia Commonwealth University

Follow this and additional works at: https://scholarscompass.vcu.edu/etd

Part of the Computer Sciences Commons

© The Author

Downloaded from
https://scholarscompass.vcu.edu/etd/702

This Thesis is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.

المنارة للاستشارات

www.manaraa.com

School of Engineering
Virginia Commonwealth University

This is to certify that the thesis prepared by Christopher Alonzo Parker entitled K x N Trust-Based Agent Reputation has been approved by his or her committee as satisfactory completion of the thesis or dissertation requirement for the degree of Masters of Science in Computer Science

---

Dr. David Primeaux, Associate Professor of Computer Science, School of Engineering

---

Dr. Chao-Kun Cheng, Associate Professor of Computer Science, School of Engineering

---

Dr. Gurpreet Dhillon, Professor of Information Systems, School of Business

---

Dr. Dan Resler, Interim Chairman, Department of Computer Science, School of Engineering

---

Dr. Russell D. Jamison, Dean, School of Engineering

Dr. F. Douglas Boudinot, Dean of the School of Graduate Studies

July 31th, 2006

المنارة للاستشارات

www.manaraa.com

© Christopher Alonzo Parker, 2006

All Rights Reserved

www.manaraa.com

K x N TRUST-BASED AGENT REPUTATION

A Thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Computer Science at Virginia Commonwealth University.

by

CHRISTOPHER ALONZO PARKER
Bachelors of Science in Computer Science, Virginia Commonwealth University, 2000

Director: DR. DAVID PRIMEAUX
ASSOCIATE PROFESSOR OF COMPUTER SCIENCE, DEPARTMENT OF COMPUTER SCIENCE

Virginia Commonwealth University
Richmond, Virginia
August, 2006

المنارة للاستشارات
www.manaraa.com

Acknowledgement

I would like to thank my parents for preparing me for life's adventure. A special acknowledgement is reserved for my former co-workers who allowed me to work on homework, papers, and test preparation while affording the occasional opportunity for "cat-naps" before early morning classes after a full-time graveyard shift. I would like to thank my wife, Bessetta Parker, for continued support, encouragement, and prayers. I would like to thank my thesis advisor, Dr. David Primeaux, for guidance, direction, and a secondary viewpoint that helped to bring clarity throughout this process.

للاستشارات المنارة

www.manaraa.com

Table of Contents

Page

Acknowledgements .... **ii**
List of Tables .... **v**
List of Figures .... **vi**
1. SOFTWARE AGENTS **1**
1.1. Introduction 1
1.1.1. Introduction to Research 1
1.1.2. Human Agency 3
1.1.3. Software Agency 4
1.2. Categories of Agents 6
1.2.1. Intelligent 6
1.2.2. Learning/Adaptive 7
1.2.3. Mobile 8
1.2.4. Believable 9
1.3. Autonomy 11
1.4. Rational Agency 12
1.5. BDI Agents 15
2. DISTRIBUTED ARTIFICIAL INTELLIGENCE **17**
2.1. Overview 17
2.2. Cooperation 20
2.3. Coordination 21
2.4. Distributed Problem Solving 23
2.5. Multi-Agent Systems 25
2.6. Emergence 27
3. TRUST **31**
3.1. Introduction 31
3.2. Types of Trust 35
3.3. Computing Trust 37
3.3.1. Trust Update Function 37
3.3.2. Trust Evolution Function 42
3.4. Applications of Trust Types 47
3.4.1. Trust in Information 47
3.4.2. Trust in Information Sources 48
3.4.3. Trust in Warrantors and Authorities 50
3.4.4. Trust in Oneself 51
3.4.5. Trust in Potential Partners 52
4. MACHINE LEARNING **62**
4.1. Overview 62
4.2. Types of Learning 64
4.3. MAS Learning 67
4.4. Machine Learning and Trust 72
5. K x N TRUST-BASED AGENT REPUTATION **73**
5.1. k-NEAREST NEIGHBOR 73
5.2. KMAS 77
5.2.1. Experiment Description 77
5.2.2. Experiment 1 Hypotheses, Results, and Conclusions 93
5.2.3. Experiment 2 Hypotheses, Results, and Conclusions 108
5.2.4. Experiment 3 Hypotheses, Results, and Conclusions 119
5.3. Future Research 131

www.manaraa.com

List of References **137**
Appendices **143**
Appendix A: fixedIn.txt 144
Appendix B: Experiment Trial Input 145
Appendix C: Failure Rate Log Example 146
Appendix D: Agent Cooperation Log Example 148
Appendix E: class CreateFixedInputs 151
Appendix F: class ThesisKmas 153
Appendix G: class Kmas 155
Appendix H: class KmasAgent 168
Appendix I: Experiment: 1 Failures over time 183
Appendix J: Experiment: 2 Failures over time 189
Appendix K: Experiment: 3 Failures over time 195

المنارة للاستشارات
www.manaraa.com

List of Tables

Table 1: Experiment 1 Inputs. 94

Table 2: Experiment 1 ETIP Contents. 94

Table 3: Experiment 1 Group A Observations. 95

Table 4: Experiment 1 Group B Observations. 97

Table 5: Experiment 1 Group C Observations. 99

Table 6: Experiment 2 Inputs. 109

Table 7: Experiment 2 ETIP Contents. 109

Table 8: Experiment 2 Group A Observations. 110

Table 9: Experiment 2 Group B Observations. 112

Table 10: Experiment 2 Group C Observations. 114

Table 11: Experiment 3 Inputs. 120

Table 12: Experiment 3 ETIP Contents. 120

Table 13: Experiment 3 Group A Observations. 121

Table 14: Experiment 3 Group B Observations. 123

Table 15: Experiment 3 Group C Observations. 125

المنارة للاستشارات

www.manaraa.com

List of Figures

Figure 1: Experiment 1 Group A Failure Rate 95

Figure 2: Experiment 1 Group A Individual Failure Rate 96

Figure 3: Experiment 1 Group B Failure Rate....97

Figure 4: Experiment 1 Group B Individual Failure Rate 98

Figure 5: Experiment 1 Group C Failure Rate....99

Figure 6: Experiment 1 Group C Individual Failure Rate 100

Figure 7: Experiment 1 Group A Failures by Time Step....107

Figure 8: Experiment 2 Group A Failure Rate. 110

Figure 9: Experiment 2 Group A Individual Failure Rate 111

Figure 10: Experiment 2 Group B Failure Rate....112

Figure 11: Experiment 2 Group B Individual Failure Rate 113

Figure 12: Experiment 2 Group C Failure Rate....114

Figure 13: Experiment 2 Group C Individual Failure Rate 115

Figure 14: Experiment 3 Group A Failure Rate 121

Figure 15: Experiment 3 Group A Individual Failure Rate....122

Figure 16: Experiment 3 Group B Failure Rate....123

Figure 17: Experiment 3 Group B Individual Failure Rate 124

Figure 18: Experiment 3 Group C Failure Rate....125

Figure 19: Experiment 3 Group C Individual Failure Rate 126

المنارة للاستشارات
www.manaraa.com

Abstract

K x N Trust-Based Agent Reputation

By Christopher Parker, M.S. Computer Science

A Thesis submitted in partial fulfillment of the requirements for the degree of Computer Science, Master of Science at Virginia Commonwealth University.

Virginia Commonwealth University, 2006

Major Director: Dr. David Primeaux
Associate Professor, Computer Science

In this research, a multi-agent system called KMAS is presented that models an environment of intelligent, autonomous, rational, and adaptive agents that reason about trust, and adapt trust based on experience. Agents reason and adapt using a modification of the k-Nearest Neighbor algorithm called (k X n) Nearest Neighbor where ***k*** neighbors recommend reputation values for trust during each of ***n*** interactions. Reputation allows a single agent to receive recommendations about the trustworthiness of others. One goal is to present a recommendation model of trust that outperforms MAS architectures relying solely on direct agent interaction. A second goal is to

www.manaraa.com

converge KMAS to an emergent system state where only successful cooperation is allowed. Three experiments are chosen to compare KMAS against a non-(k X n) MAS, and between different variations of KMAS execution. Research results show KMAS converges to the desired state, and in the context of this research, KMAS outperforms a direct interaction-based system.

المنارة للاستشارات

www.manaraa.com

CHAPTER 1 SOFTWARE AGENTS

SECTION 1.1: INTRODUCTION

1.1.1 INTRODUCTION TO RESEARCH

Trust has been proposed as a way to allow agents to cooperate while mitigating the risks associated with harmful interactions with untrustworthy partners. In this research, a multi-agent system called KMAS is presented. KMAS models an environment of intelligent, autonomous, rational, and adaptive agents. The agents are intelligent, autonomous, and rational because of their ability to reason about the trustworthiness of other agents and autonomously decide which agents to interact with based on self-interest and risk. Agents are adaptive in their ability to update trust based on experience with cooperative partners. Before presenting the KMAS model, Chapters 1 through 4 will present pertinent subject matter from the areas of Software Agency, Distributed Artificial Intelligence, Trust, and Machine Learning, as well as existing research.

Trust is further modeled in an adaptive manner by allowing an agent seeking a cooperative engagement, to receive recommendations from other agents as to the

المنارة للاستشارات
www.manaraa.com

trustworthiness of the selected interaction partner. This type of trust is called reputation, and will be discussed in Chapter 3. Reputation is modeled by employing an adaptation of the k-Nearest Neighbor algorithm where nearest neighbors are providing trust values as advisors in an indirect-supervised learning process. Indirect-supervised learning is explained in Section 4.2 The adaptation is described as (k X n) Nearest Neighbor because the k-Nearest Neighbor algorithm is performed n times where n represents the number of interactions requiring k-Nearest Neighbor to be executed with k neighbors recommending reputation values for trust. The goal of this research is to present the KMAS system as a way to model trust in the form of recommendation-based reputation that will outperform MAS architectures that rely solely on direct interaction between agents to update trust based on practical experience. Performance is measured by task completion rate. A second goal is to converge KMAS to a system state where only successful cooperation is allowed to occur as an emergent property of the system.

This research will conduct experiments to compare performance between KMAS and a system that is not using the nearest neighbor algorithm. Performance is measured by each system's ability to only allow cooperation between an agent seeking to engage in a cooperative task and a partner that is not only "trusted", but also will not cause a harmful interaction in terms of unsuccessful cooperation. Experiments will also be used to compare different variations of KMAS implementation. Experiment 1 will contrast KMAS and non-KMAS performance. Experiment 2 will investigate using 3 different values for the number of nearest neighbors in KMAS

المنارة للاستشارات
www.manaraa.com

execution. Experiment 3 will compare different values for how often (k X n) Nearest Neighbor is performed during execution. The default KMAS configuration is to only ask for recommendations if an agent is unknown. It may be beneficial to ask for recommendations for known agents. Research results will show that KMAS can indeed converge to the desired state, and outperform a system that does not use recommendation-based reputation.

1.1.2 HUMAN AGENCY

The notion of agency outside the realm of computer technology is by no means a new or novel concept. In fact, it is not a stretch of the imagination to suggest that a vast majority of individuals in human society have interacted with at least one "agent" during their lifetime. Human agents are often described as being focused on a specific task, having skills in an area in which they are deemed to be specialists, having access to information relevant to a specific task, having the necessary contacts to provide a service, being able to provide a service at a lower cost than the requester of the service, and having the ability to provide a service that the requester cannot receive in any other way [Murch and Johnson, 1999]. Upon listing these attributes of human agents, it is quite easy to describe or list some of the numerous services that human agents provide on a daily basis. In regards to information, human agents provide detailed background information, specifications, requirements, statistical, and other pertinent information concerning products, services, or subject matter. Headhunter agencies assist the job seeker by targeting national and international career opportunities in a fraction of the

www.manaraa.com

time and cost. Human agents are often used to negotiate agreements between buyer and seller of real estate, while other agents prepare the necessary contracts and agreements. Agents are also helpful in the area of managing personal finances where they lend expertise in diverse areas ranging from debt management to retirement planning. In [Murch and Johnson, 1999], the authors simply describe a human agent as "someone who performs some act on behalf of another that he or she is uniquely qualified to undertake."

1.1.3 SOFTWARE AGENCY

A wide range of definitions and characteristics has been associated with the term "agent" as it applies to computer software systems. Agents have been defined with descriptions ranging from "independent entities equipped with some amount of decision making power" [Barber et al., 2000], to "an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives" [N. R. Jennings, 1999]. The latter definition is actually a refinement of the first, and describes what is commonly referred to as an "autonomous agent" and will be described in subsequent sections. This paper is concerned with the types of agents that are distinguished from others by the environment that they are situated in. Our research is focused on software agents that occupy software environments as opposed to robot agents that inhabit physical environments.

المنارة للاستشارات
www.manaraa.com

Software agents can perform many of the tasks that our human agents are presently performing. Benefits obtained from using software agents can be found in many sectors ranging from consumer and business markets to professional services. For consumers, agents can make our lives more productive by freeing up time from certain routine tasks such as paying bills or shopping, and can find information relating to a wealth of subjects on our behalf. Automating financial management as an alternative to "hard to stick by budgets" has been proposed as a method to ensure financial security well into the retirement years [Bach, 2004]. For businesses, software agents can help companies be more efficient and lower costs. The healthcare industry has used software agents to help healthcare professionals manage patient care by assisting with diagnoses and prescription recommendations [Murch and Johnson, 1999]. For law enforcement, the National Center for Missing and Exploited Children uses intelligent agents and facial recognition algorithms to scan photographs from multiple Internet sources and anticipate the likelihood of success while assessing leads [Romaniuk, 2000]. These few examples alone highlight the impact of agent research on our everyday lives.

المنارة للاستشارات
www.manaraa.com

SECTION 1.2: CATEGORIES OF AGENTS

1.2.1 INTELLIGENT

Now that we have defined what software agents are and how they impact society, a further discussion is warranted to describe the different categories that agents can be subdivided into. These categories are not meant to be mutually exclusive and are simply mentioned to highlight agent diversity. The first category of agents can be described as intelligent agents. A working definition from Gilbert, et al. (1995) [as cited in Hermans, 1996, p. 17], is as follows:

> Intelligence is the degree of reasoning and learned behavior: the agent's ability to accept the user's statement of goals and carry out the task delegated to it. At a minimum, there can be some statement of preferences, perhaps in the form of rules, with an inference engine or some other reasoning mechanism to act on these preferences. Higher levels of intelligence include user model or some other form of understanding or reasoning about what a user wants done, and planning the means to achieve this goal. Further out on the intelligence scale are systems that learn and adapt to their environment, both in terms of the user's objectives, and in terms of the resources available to the agent. Such a system might, like a human assistant, discover new relationships, connections, or concepts independently from the human user, and exploit these in anticipating and satisfying user needs.

المنارة للاستشارات
www.manaraa.com

Therefore, intelligent agents not only work on our behalf, but can also reason on our behalf while adapting to changes in our objectives and the resources needed to carry them out.

1.2.2 LEARNING/ADAPTIVE

The second category of agents is adaptive or learning agents. Murch and Johnson [Murch and Johnson, 1999] define learning agents as "software agents that basically learn from the user or owner". They define learning as the "modification of behavior through experience or judgment". We will present an explanation of learning as it applies to computer science later in the paper. Learning or adaptation can be applied in single agents or groups of agents. Adaptive agents can be useful in designing intelligent user interfaces where the system adapts to individual differences across users. In the VIENA system [Lenzmann and Wachsmuth, 1996], a system of intelligent agents is used to create interactive manipulation of 3D graphical scenes. The system translates verbal commands from the user into technical commands that update the visual model. Agents have different tasks such as translating the command "left" into screen coordinates based on some built-in special preference that determines how left is carried out. Spatially, "left" can be carried out in a way that is closer or father away from the user. The user gives implicit feedback by way of correcting solutions offered by agents. For example, verbal feedback such as "a bit less", can correct the solution offered by the agent tasked with performing "move chair left". Agents that meet user

المنارة للاستشارات
www.manaraa.com

expectations are "credited", while agents that do not are "discredited". VIENA adapts to user preferences by learning from direct feedback until agents that generate preferred solutions are dominant in the system.

1.2.3 MOBILE

In addition to having the ability to learn or adapt, agents can also be designed with mobility. Mobile agents can travel across a network of computers, including the Internet, to execute tasks. They are often used to collect data, information, or changes. Mobile agents have been discussed as a way to enhance search capabilities over existing methods. Traditional search engines use web crawlers within a client-server architecture. The web crawlers are programs that search web pages for keywords and store the web page indices into massive databases. This creates a tremendous load on network resources, as raw data must be sent across the network to be processed on the server by manner of the web crawler. Often, only a small portion of this data is actually needed. [Mandalapu and Adya, (n.d.)] have proposed mobile agents as a way to move processing to the raw data as opposed to moving the raw data to the processing. An advantage of using such agents is that after being dispatched, the mobile agent is not constrained by whether or not the dispatcher is on- or off-line.

المنارة للاستشارات
www.manaraa.com

1.2.4 BELIEVABLE

The last category of agents can be described as "believable" agents. Believable agents are agents that show personality, emotion, and give the illusion of life to the user or the individual that interacts with such an agent. [Mateas, 1997] describes the philosophy behind believable agents as it relates to the Oz Project, a research group that studies believable agents in interactive drama. He describes research goals of developing agent personalities, giving the audience the perception that the agents are "believable" in the sense that they appear lifelike and display actions that make sense, and creating agents that are developed as specific characters (artistic abstractions of reality).

The specific type of software agents that will be used in our research are intelligent, adaptive (learning) agents. Our agents will be intelligent in their ability to reason about the trustworthiness of other agents. It is most likely that the agents in this research, at best, exhibit low level intelligence. It is also acknowledged that the agents' ability to display intelligence by reasoning about trust is debatable. In this paper, it is claimed that reasoning about the trustworthiness of another agent is a computational process consisting of two input categories: 1) the output created by performing the nearest neighbor algorithm and criteria used to select nearest neighbors such as agent age in the system, successful interaction history, and agent predisposition to risk and trust, and 2) updated trust based on past interaction experiences with the neighbor

www.manaraa.com

whose trustworthiness is being computed. The computational process allows an agent to make a decision about the trustworthiness of another agent in the context of a potential interaction, and to equate this decision with a numerical, discrete value. It is unclear whether or not this decision alone provides enough justification to classify KMAS agents as being intelligent or capable of reason. This question is left open to the reader. Agents will be able to adapt through experience as interaction with other agents forces them to update their beliefs about trust and the risks associated with cooperating.

المنارة للاستشارات
www.manaraa.com

SECTION 1.3: AUTONOMY

In the beginning of the paper, we defined an autonomous agent as one that is capable of flexible, autonomous action in order to meet its design objectives. [Murch and Johnson, 1999] define the property of autonomy as the notion that the agent "exercises control over its own action", and that "autonomous execution is clearly central to agency". [D'Inverno and Luck, 2001, 2004] add that autonomous agents are self-motivated in the sense that they "create and pursue their own agendas" as opposed to being under the control of another agent. In this sense, autonomous agents do not simply act because they are "told what to do". They act because of some internal motivation which D'Inverno and Luck define as "any desire or preference that can lead to the generation and adoption of goals and that affects the outcome of the reasoning or behavioral task intended to satisfy those goals". [Ossowski, 1999] further adds that not only do autonomous agents "make their own decisions" with respect to goal adoption, but they also choose how to pursue those goals. Autonomous agents can even choose to adopt the goals of other agents if those goals are in line with their own personal motivations.

المنارة للاستشارات
www.manaraa.com

SECTION 1.4: RATIONAL AGENCY

By discussing autonomy, we describe a property of certain agents (that is autonomous agents) which allows us to understand how certain agents adopt goals. After goal adoption, it is important to understand why an agent chooses a particular action to realize those goals, especially if there are multiple actions that can achieve the desired result. In such cases, agents must make a rational choice between competing actions. [Wooldridge and Rao, 1999] provide a simple definition of rational agents as "software entities that perceive their physical or software environment through appropriate sensors; have a model and can reason about the environment that they inhabit; and based on their own mental state take actions that change their environment". They further expand this definition by stating that the key aspects of rationality are: 1) balancing reactive and proactive behavior, 2) balancing perception, deliberation, and action, especially when there are limited resources, and 3) balancing self-interest and community interest. It is clear then, that rationality does indeed involve choices.

[Russell, 1999] states that the actions that are best suited "make sense from the point of view of the information possessed by the agent and its goals". Why would an agent decide that it "makes sense" to undertake an action? To answer this question, we will first present Russell's four definitions of agent rationality. Perfect rationality is the capacity to generate maximally successful behavior given the available information.

المنارة للاستشارات
www.manaraa.com

Calculative rationality is the in-principle capacity to compute the perfectly rational decision given the initially available information. Metalevel rationality is the capacity to select the optimal combination of computation-sequence-plus-action, under the constraint that the action must be selected by the computation. Lastly, bounded optimality (bounded rationality) is the capacity to generate maximally successful behavior given the available information and computational resources. Here, the author's use of behavior implies the actions that can be performed by the agent. Upon seeing the terms "maximally", "perfectly", and "optimal", we see that agents "pursue tasks in a rational manner by choosing the action that they believe to be best in order to achieve a task" [Ossowski, 1999]. Wooldridge and Rao's third key aspect of rationality identifies this philosophy as self-interest. [Klusch et al., 2003] build upon this concept by stating that rational agents "behave in a utilitarian way in an economic sense. They act, and may even collaborate, to increase their own benefits."

In this paper, we are particularly interested in agents that are both autonomous and rational. From [Wooldridge, 2000] we have the following four characteristics of autonomous, rational agents, and will adopt these characteristics for the purposes of this work:

1. autonomy: having independent decision making and acting capabilities
2. proactiveness: exhibiting goal directed behavior
3. reactivity: being responsive to environmental changes
4. social ability: interacting with other agents

المنارة للاستشارات
www.manaraa.com

Agents used in our research will be autonomous, rational agents. They will be autonomous in the sense that they will choose which agents will be selected as interaction partners, and based on some criteria, will choose to engage in cooperative action with the chosen partner. Proactiveness is a displayed as the cooperative action is undertaken to achieve some goal. Reactivity is shown as agents are responsive to the agent society around them as new agents enter the system environment. Social ability is required for agents that must interact and cooperate.

المنارة للاستشارات

www.manaraa.com

SECTION 1.5: BDI AGENTS

As stated previously, rational agents autonomously decide upon an objective to achieve and by doing so, exhibit goal directed behavior. One such proposed and widely accepted model of agent rationalism is the BDI agent model [Wooldridge, 2000]. Since agents in real-time commercial environments exist in a dynamic setting, they must constantly assess their surroundings. The BDI model allows agents to be implemented in such a way as to allow them to react to change in the environment and adjust their goals accordingly.

The BDI acronym stands for beliefs, desires, and intentions. As an agent observes its environment, itself, and other agents, its perceptions are the basis for beliefs about the surrounding world. Therefore, the agent encapsulates within itself a model of the environment that is static until future perceptions detect changes in the actual environment. After the agent has modified beliefs as a result of change, it may form desires in response which displays the rational agent trait of reactivity. Desires are the actual goals that the agent wishes to bring about. Developing an intention is simply agent commitment to achieving a goal.

The agents in this research will not use the BDI model. Although our agents are autonomous and rational, we are merely concerned with the aspect of agent decision making that is based on the risk associated with agent partnerships during cooperative

المنارة للاستشارات
www.manaraa.com

task execution. Agent goals are not defined, and cooperative tasks are assumed rather than explicitly modeled.

المنارة للاستشارات

www.manaraa.com

CHAPTER 2 DISTRIBUTED ARTIFICIAL INTELLIGENCE

SECTION 2.1: OVERVIEW

Earlier, we represented the VIENA system as an example of how a system of intelligent agents can be used to solve a particular problem. In VIENA's case, the problem that the system addresses is "how to adapt the user interface to meet the needs of various users with differing preferences". We saw that VIENA achieved this by decomposing the overall problem into subproblems or tasks that the system was able to solve at the agent level. The research area of DAI, Distributed Artificial Intelligence, is concerned with systems such as VIENA, where several systems or system components interact in order to solve a shared or common problem. With VIENA, computers and people are the two "systems" that must interact in order to solve the problem. [Moulin and Chaib-Draa, 1996] define DAI as a "subfield of artificial intelligence which has, for more than a decade now, been investigating knowledge models, as well as communication and reasoning techniques that computational agents might need to participate in societies composed of computers and people".

للاستشارات المنارة
www.manaraa.com

Moulin and Chaib-Draa identify many reasons for why research in this particular area of computer science is important. DAI can aid in knowledge representation and problem solving by providing richer scientific formulations and more realistic representations in practice. As with VIENA, it may be better to break down a complicated system into different cooperative entities, such as intelligent agents, to obtain efficiency. DAI systems can also provide a framework to test theories about reasoning processes based on knowledge, actions, and planning, as well as contribute to our understanding of communication processes based on natural language. [Gasser, 1992] provides examples of typical problem domains that DAI can be applied to. He describes these domains as those in which there can be found:

1) Clear (possibly hierarchical) structures of time, knowledge, communication, goals, planning, or action
2) Natural (not forced) distribution of actions, perceptions, authority, and/or control
3) Interdependencies because local decisions may have global impacts, or there may be harmful interactions among agents
4) Possible limits on communication time, bandwidth, etc., so that a global viewpoint, controller, or solution is not possible

المنارة للاستشارات
www.manaraa.com

Application domains, to list a few, can be found in the areas of speech and language processing, manufacturing, robotics, design (VIENA), monitoring and control, and specialized research problems such as the prisoner’s dilemma.

المنارة للاستشارات
www.manaraa.com

SECTION 2.2: COOPERATION

Our overview of the field of DAI research identifies interaction to solve a problem as a central theme. In order to solve problems involving the usage of more than one system or system component, cooperation between individual entities must exist. Users must cooperate with a system by providing commands, information, feedback, and the system must respond in kind. Within systems involving multiple components such as agents that have various tasks and responsibilities, cooperation must take place to achieve the overall design goal of the system. [Durfee et al., 1989] outline 4 generic goals for cooperation within DAI. The authors believe cooperation can increase the task completion rate through parallelism, increase the set or scope of achievable tasks by sharing resources such as information and expertise, increase the likelihood of completing tasks by undertaking duplicate tasks with possibly different methods of performing these tasks, and decrease interference between tasks by avoiding harmful interactions.

المنارة للاستشارات
www.manaraa.com

SECTION 2.3: COORDINATION

In the presence of cooperating entities, DAI research must manage interdependencies between systems or system components. This management of interdependencies is called the process of coordination. [Malone, 1990] states that "the two most fundamental components of coordination are the allocation of scarce resources and the communication of intermediate results". In the case of DAI systems using agents, Moulin and Chaib-Draa indicate that "without coordination, a group of agents can quickly degenerate into a chaotic collection of individuals, since an agent only has a partial and imprecise view of the overall system and its actions may interfere with rather than support other agents' actions". [Jennings, 1996] states that the three main reasons for coordinating agents are dependencies between agents' actions on one another, the need to meet global constraints, and that no individual agent has sufficient competence, resources, or information to solve the entire problem. Coordination is needed to ensure that all portions of the overall problem are being addressed by some agent, agent interactions lead to the problem solution, and system goals are realizable in the presence of limited or scarce resources. Coordination also allows agents to view others as being committed to the interactions that lead to the problem solution.

To enable the coordination process, many techniques have been proposed such as negotiation [Ashri et al., 2003], arbitration [Barber et al., 2000], voting [Barber et al., 2000], self-modification [Barber et al., 2000], organization [Schumacher, 2000], and

المنارة للاستشارات
www.manaraa.com

multi-agent planning [Ossowski, 1999]. Of these examples, the major coordination techniques have been organization, negotiation, and multi-agent planning. In organizational structures, agents have a priori defined roles that other agents have knowledge of. These roles ensure that agents commit to the behavior that the roles represent. Negotiation solves the coordination problems of task and resource allocation. Negotiation can also limit or remove potential harmful interactions between agents. Multi-agent planning provides agent plans that specify future actions and interaction to not only allow agents to be aware of other agent responsibilities, but also displays agent committal to an action or interaction that other agents rely on. All three coordination strategies ensure that the overall problem is being addressed.

Coordination techniques and agent cooperation are the framework that allows groups of agents to fulfill cooperative problem solving goals. Within DAI, there are two major areas of research: DPS (Distributed Problem Solving, and MAS (Multi-Agent Systems). Both approaches are similar in their usage of agents to solve cooperative problems. Their differences lie in the type of agents employed and the goals of the researchers.

المنارة للاستشارات
www.manaraa.com

SECTION 2.4: DISTRIBUTED PROBLEM SOLVING

Distributed problem solving studies how problems can be solved by task allocation to a group of cooperating agents that are coordinated by some process. The coordination techniques discussed in the previous paragraphs allow for agents to exist in a cooperative system that has been designed with a structure that allows agents to know their place within the structure, the scope of what parts other agents play in the problem solving process, and how interactions are defined. DPS systems assume that cooperation among agents takes place. This is aided by the type of agents that are actors in the system. Agents are assumed to be benevolent in the sense that they have common or non-conflicting goals with other agents, and in contrast with our discussion of rational agency, these agents do not seek self-interest. Agents are also homogeneous in the sense of common architectures, ontologies, knowledge representations, communication languages, degree of problem solving capability, and preference criteria. According to [Ossowski, 1999], these are traditional assumptions of DPS research. DPS research goals are aimed at creating predefined system functionality or properties for a group of cooperating agents whose characteristics are controlled.

المنارة للاستشارات
www.manaraa.com

Because cooperation is assumed, DPS is often called cooperative distributed problem solving. [Wooldridge, 2000] describes four stages of a cooperative problem solving process model.

1) Recognition: Agents recognize potential for cooperative action.
2) Team Formation: Agents solicit assistance.
3) Plan formation: Agents develop a joint plan to achieve the goal.
4) Team Action: Agents cooperatively execute the joint plan.

المنارة للاستشارات
www.manaraa.com

SECTION 2.5: MULTI-AGENT SYSTEMS

In contrast to DPS systems, a MAS (Multi-Agent System) architecture allows for agents to be heterogeneous. Agent architectures, problem solving capabilities, expertise, communications languages, etc., can vary from agent to agent. In fact, agents are not assumed to be benevolent. Existence of multiple, conflicting goals may be present and even encouraged. For these reasons, [Durfee et al., 1989] define a MAS as a "loosely coupled network of problem solvers that work together to solve problems that are beyond their individual capabilities". Loosely-coupled not only identifies varying architectures, languages, and goals, but it also highlights the fact that these agents have the properties of autonomy and self-interested rationality. Rational agents may have local goals that could conflict with the goals of the system as a whole. Advantages of this type of system over single agent systems are parallelism which can provide faster problem solving, scalability, robustness, decreased communication by transmitting only partial solutions across agents as opposed to raw data processing at one central site, increased reliability by allowing agents to take on responsibilities of another agent that failed, intelligence, and the implementation of "real-world" simulations. Distributed processing in a concurrent manner can increase efficiency of the system when handling multiple sources of knowledge or multiple activities. This leads to the MAS having the system property of scalability. Scalability measures a system's ability to enhance performance through parallelism without loss of efficiency. For robustness, agents embodying system processes can be designed within cooperation

المنارة للاستشارات
www.manaraa.com

frameworks that can promote conflict resolution and deadlock prevention. Agent interactions can also allow individual agents, or groups, the benefit of increased levels of intelligence with respect to the system environment, problem domain knowledge, and inter-agent cooperation. System intelligence as a whole can be achieved by the intelligence of the agents that make up the system. Lastly, multi-agent based simulation can be used to model complex social and cooperative structures to aid researchers in understanding collective behavior and intelligence.

المنارة للاستشارات
www.manaraa.com

SECTION 2.6: EMERGENCE

The MAS approach also differs from the traditional DPS approach because researchers are concerned with properties or functionalities of a system that arise through interactions among a diverse group of agents. This is the idea of emergence. Clarke, Irwig, and Wobcke describe emergence as a property of a system through their work with Tileworld [Clark et al., 1997]. The emergent property is observed when viewing all of the system components as a whole. In the case of a MAS, the emergent system property is observed when viewing collective agent properties and collective agent interactions. Of particular interest is the fact that these authors choose to use agents based on a BDI architecture. As rational agents, they seek to maximize their utility without regard to other agents' welfare or the welfare of the system as a whole. This does not automatically lead to "malicious behavior", but affords the opportunity. In the case of emergent properties, an agent acting to seek its own benefit may unknowingly contribute to the overall utility of the system.

In Tileworld, agents compete for a food resource. Agents score points on a 2-dimentional grid by moving to *holes*. When a hole is reached, it is filled. An agent's performance is measured by the number of holes it fills. A "controller" agent uses the given number of holes and a vanishing rate to determine hole placement. At the beginning of each execution cycle the controller agent ensures that the given number of holes is present on the grid. A vanishing rate of 0 means that a hole disappears once

المنارة للاستشارات
www.manaraa.com

filled in. A value of 1 ensures that every unfilled hole will disappear and reappear in a random position at each execution cycle. Thus, the vanishing rate is a value within the interval [0,1], and represents the dynamic nature of the environment.

The BDI agent's goal in Tileworld is to fill a hole while forming an intention to fill the one closest to it. Three different types of agents are used, characterized by their level of communications. Only agents of the same type may form teams and communicate. Type 1 is non-communicating. Type 2 agents only inform the closest team member (within a range r) of an intention to fill a hole. The team members will then never form an intention to fill the same hole. Type 3 agents are different in the respect that such an agent <u>will</u> form an intention to fill a hole despite being told of other team members' wishes to fill the same hole if the agent is closer to the hole than those team members. If this occurs, the other team members are then forced to abandon their intentions.

After experimentation with the agents in Tileworld, the authors found that up to a certain limit, individual performance of the team members increased as the size of teams increased. As members and hole consumption increase, more holes are replenished by the "controller" agent to maintain the given amount. Another observed property of the system is that communicating agents avoid interfering with each other, which decreases time wasted on trying to fulfill unobtainable intentions. The emergent system property is the advantage of working in teams. The authors also expected that as

المنارة للاستشارات
www.manaraa.com

the range of communication grew, average team performance would increase. They found another emergent system property in that the performance varied logarithmically with the range. As a result, it can be shown that desired properties of a system can be produced by interactions between rational, autonomous agents, although the agents themselves are not concerned with overall system utility.

We identified that this paper would be concerned with autonomous, rational agents. In this paper, a system of intelligent agents will be presented in the form of a multi-agent system called KMAS. Like VIENA, KMAS is a system of intelligent agents that will be used to solve a problem. The problem can be simply represented as "finding and isolating deceptive agents" as the system converges towards a state that only allows cooperation with non-deceptive agents. Sub-problems are solved at the agent level to identify deceptive agents. As a DAI system, KMAS can test theories about trust and trust representations. As indicated in Section 2.2, one of the goals of cooperation is to increase task completion rate by avoiding harmful interactions. KMAS is used to research whether or not trust can be used as a cooperation strategy to achieve this as a system goal. KMAS is intelligent at the system level because of the intelligence of the individual agents that exist and cooperate within the system. As stated in Section 1.2, our agents are intelligent because of their ability to reason about agent trustworthiness. Because cooperation is not assumed in all cases, our rational agents may malevolently possess a goal that allows for breaking of commitments. An agent may agree to cooperate, but then choose not to honor this agreement if it is not in

المنارة للاستشارات
www.manaraa.com

the agent's best interest (self-interest). We are also interested in determining if our experiments provide the opportunity to observe emergent properties of the KMAS system.

www.manaraa.com

CHAPTER 3 TRUST

SECTION 3.1: INTRODUCTION

At the heart of MAS research is the engagement in cooperative partnerships between intelligent agents for the purpose of achieving goals through cooperative tasks or the sharing of information. In particular, autonomous, rational agents face distinct challenges when deciding the feasibility or merits of cooperation with potential partners. The coordination process allows agents to expect that other agents will be committed to an interaction. If agents are rational, there is a risk that commitments will be broken, interactions will result in harmful consequences, and misleading or inaccurate information will be shared. Inherent is the possibility that agents will act malevolently (as opposed to benevolently) while pursuing self-interested goals. Trust has been introduced as a technique that rational agents may use as part of the deliberation process to assess whether or not cooperation will occur. It is also valuable in determining acceptance of information received from agent information sources. As a learned function, trust can be derived from previous agent experiences, and can be updated according to new or future experiences. This paper will explore the usage of rational, intelligent agents in a MAS environment where trust is used as a computational

www.manaraa.com

function to determine the trustworthiness of others. Existing research will be presented to describe the various applications of trust within the field.

In Chapter 3, we will discuss the benefits of using trust to justify interactions within DAI systems to aid in the coordination or cooperation processes, and how trust can be thought of conceptually and computationally. We will also discuss trust as a way to judge information and information sources in the form of other agents.

In Chapter 4, we will discuss the usage of machine learning techniques to aid in the coordination and cooperation processes of specific DAI research in the area of multi-agent systems. MAS architectures where trust is used as a basis to select information or potential agent interaction partners will also be investigated. The different types of learning will be described. Learning will be discussed as a process used to identify optimal strategies, or the best agent to gather advice from.

There is no universal, mutually agreed upon definition of trust. What is acknowledged is that trusting relationships between parties implied some form of risk to both. The wide array of current definitions of trust has been discussed in [Falcone et al., 2001], [Marsh, 1994], and [Griffiths and Luck, 1999]. Marsh defines trust as taking an ambiguous path where an assumption is made that positive effects outweigh the negative. He also describes trust as a continuum of varying degrees of trust delimited by blind trust and complete mistrust. To Marsh, trust is dynamic in nature, and viewing

المنارة للاستشارات
www.manaraa.com

trust as a continuum, allows trust to change along the points of this continuum. Thus, agents can be more or less trusting of others based on experiences. A different approach used by [McKnight and Chervany, 2001] seeks to define trust as a set of high level concepts because "trust is by nature hard to narrow down to one specific definition because of the richness of meanings the term conveys in everyday usage". They divide trust into categories such as trusting intentions, trust related behavior, trusting beliefs, institutional-based trust, and disposition to trust. These broad, high level concepts can then be described by a series of measurable subsets such as information sharing, predictability, and trusting stance. In addition to implied risk, trust is also described as an inherent dependency between two parties where party A applies trust as the assessment by which A expects party B to perform (or not perform) a given action on which A's welfare depends [Witkowski et al., 2001].

The importance of trust has emerged in many problem domains such as E-Commerce, Agent Modeling, HCI, Computer Supported Cooperative Work, Mixed Initiative and Adjustable Autonomy, and Ubiquitous Computing [Falcone et al., 2001]. Marsh [Marsh, 1994] outlines the following advantages of trust:

1) Allows an agent to prepare itself for malevolent behavior.
2) Ensures robustness with respect to unknown agents and unforeseen interactions.
3) Helps in formation of groups.

المنارة للاستشارات
www.manaraa.com

4) Reduces complexity (agents need only to consider world states that arise from trusted actions).

5) Allows for validation of information and source in information sharing.

6) Justifies interactions because DAI lacks central authority.

Trust is complex by nature, and "should not be reduced to mere security" [Falcone et al., 2001]. Trust, in fact, has an advantage over security. Falcone et al suggest that the world is principally insecure and that relying on another in a risky situation is inevitable. Trusting implies operating in the absence of, or under varying levels of security in which security alone is not a sufficient determinant.

المنارة للاستشارات
www.manaraa.com

SECTION 3.2: TYPES OF TRUST

Because trust is such a broad concept, and can be divided into many categories as previously discussed, researchers must decide how trust should be modeled, designed, and implemented. This involves identifying different kinds of measurable trust types [Falcone et al., 2001]. Falcone, Singh, and Tan identify these types of trust as trust in environment and infrastructure, trust in personal and mediating agents, trust in potential partners, trust in information sources, and trust in warrantors and authorities. In addition, this paper will discuss trust in information itself (for information sharing), and trust in oneself (self confidence). In general, trust types can be identified in terms of the object of trust. In terms of reducing the complexity of modeling trust or identifying trust types, trust can be measured by using one or more characteristics of the trustee. As a whole, trust can be thought of as being applied to at least two distinct, but interrelated domains: local and global. Each trust domain is defined by a set of measurable attributes. The local, or individual domain, can be said to pertain to an individual or specific object of trust. In the case of an MAS, the object of trust is most often another agent. Local trust attributes can be among the following: public record or reputation, appearance or personality, experience from the trustee perspective such as age (in terms of system life cycles), competence in the form of licenses or certifications, experience from the perspective of the one who is trusting (such as past agreements and/or outcomes), trustee/truster similar characteristics, situational/task dependent, and agent types. Global trust attributes are those that pertain

المنارة للاستشارات
www.manaraa.com

to trust in general or society as a whole. For agents in an MAS environment, global attributes would allow trust to be viewed across the entire system. Global attributes would include openness to trust in general, situations across all agents, and class preference (trust groups or profiles).

Using trust attribute form to represent trust decreases complexity by allowing trust to be computed in simple parts that can be combined into an overall trust value. This modular approach, by nature, lends itself to the concept of reusability. Runtime usage of different trust types or combination of trust types can be determined based on environmental changes, tasks, goals, or a change in agent requirements. The values for certain attributes can be reused within many different attribute combinations as needed. An example would be a task that not only requires the situational trust attribute, but also depends on openness to trust in general.

المنارة للاستشارات
www.manaraa.com

SECTION 3.3: COMPUTING TRUST

3.3.1 TRUST UPDATE FUNCTION

In an environment, agents can employ a computational model of trust that allows for trust to increase, decrease, or stay the same. One way to model trust computationally is to use what can be described as a trust update function. In [Jonker and Treur, 1999], a trust update function is defined as an inductive mathematical function that relates a current trust representation and a current experience to the next trust representation. The authors define experience as a group of evaluated events where each event can influence the degree of trust that an agents has in another. An event is evaluated as trust-positive or trust-negative depending upon whether or not the degree of trust is increased or decreased. The trust update function, tu, is modeled as tu : $E \times T \rightarrow T$ where E is the set of single experiences, and T is the set of trust values such as an interval in the set of real numbers, [-1, 1].

A simple way to model a trust update function is to take the current value of trust, and then add or subtract its weighted value. Changing the weight gives an agent the ability to increase or decrease trust in a more rapid or slower manner. This is similar to the usage of learning rates which are employed by machine learning algorithms, some of which will be discussed later. More complex functions of trust update can take the form of a multiple termed equation where current trust is not the

www.manaraa.com

lone factor in determining the new trust value. The remainder of the section is devoted to presenting an example of an application of a trust update function.

[Witkowski et al., 2001] utilize an OTB-Agent (objective-trust based agent) that selects who it will trade with primarily on the basis of a trust measure built on past experience of trading partners within a telecommunications intelligent network. The authors' example is a MAS in the form of a trading environment in which many individual agents must select partners with which they will trade on an ongoing basis. The exact nature of this trading is in the form of telecommunications management of network bandwidth. The interactions allow for trust relationships to be made, sustained, or broken over an extended period. Two types of objective-trust based agents are employed: SCP agents (service control point agents that manage access portals to the network), and SSP agents (service switching point agents that manage access points for consumers desiring telecom services).

At the beginning of each trading cycle, every SSP agent receives a demand for a resource and submits bids to SCP agents in two ways. If the SSP agent is allowed to explore by ignoring the initial trust representation of a randomly selected SCP agent, a bid will be sent to that agent. If the SSP agent does not explore, the most trusted SCP agents are successively selected until demand requirements are met. The bid size is determined by the actual demand divided by the number of SCP agents that the SSP agent will allow to receive bids. The demand rate can be inflated by an overbid rate

المنارة للاستشارات
www.manaraa.com

which determines how much extra resources an SSP agent will bid above the actual demand. In response, SCP agents attempt to distribute supply of bandwidth resources to SSP agents by making offers. When bids exceed supply, SCP agents distribute resources to the most trusted SSP agents first. If all resources are used up, other bids are rejected. SSP agents update trust in SCP agents based on the honoring of bids. Trust is increased more if an offer meets or exceeds the bid request and is increased less if the offer returned is less than the bid. Trust is decreased if the SCP agent does not return an offer at all. After offers are received, SSP agents utilize the allocated resources. SCP agents update trust in SSP agents based on resources utilized. Trust is increased the most if the resources requested are all utilized. Trust is increased less in the presence of an overbid which indicates that resources have been wasted. If the resource has not been utilized at all, trust in the SSP agent is decreased.

It is easy to see that this type of environment fosters quick pairing of trading partners. As successful interactions increase, trust in agents that are involved in these early relationships quickly surpasses trust in agents where cooperation has not taken place. In short, the MAS will quickly converge in terms of long-term partnerships between SSP and SCP agents. The authors found that when supply is less than demand, SCP agents maintain a smaller number of customers and trading partner pairing is even more isolated. As supply lessens, trust becomes a greater factor in selecting partners. They also found that loyalty to trading partners will exist in these circumstances because the less trusted SSP agents will be the first relationships to be lost since

المنارة للاستشارات
www.manaraa.com

resources are distributed to the most trusted partners first. It was also found that greedy SSP behavior in the form of overbidding was rewarded. Even though trust is lessened, when supply is equal to demand, greedy agents receive more offers causing overall delivery performance to be better than “honest” agents. Greedy agents maintain relationships with preferred, “most trusted” suppliers, but lose relationships with lesser preferred suppliers when overbid resources are not utilized. In this case, the lesser preferred suppliers are able to protect themselves from this non-benevolent behavior.

In the previous example, trust was used to rank individual agents. Trust can also be used to select “types” of agents that are desirable as interaction partners. [Birk, 2001] uses trust update to help agents learn cooperation strategies that are most appropriate for their environment with respect to the behavior of other agents and outcomes of cooperative interactions. To achieve this, the author embodies within each agent a set of hypotheses which serve to represent strategies and labels to employ during iterated games. Labels represent a form of subjective criteria to aid in partner selection. A weight, w_i, is attached to each possible label. The values of this weight are in the interval [0.0, 1.0]. To model trust, the trust function is set equal to w such that trustworthiness increases as w approaches 1.0, and decreases as w approaches 0.0. According to a threshold of trust, an agent will interact with another agent who displays the trusted label. Each strategy is described as a hypothesis because it is a potential solution to the problem of selecting the appropriate strategy. Each hypothesis, strategy and label, is ranked by a preference function which is used to select the strategy or label

المنارة للاستشارات
www.manaraa.com

to be tested. At the beginning of game execution, agents use a preference function to signal the label that they wish to display and groups are formed. A group is formed by randomly selecting one agent, and then selecting additional agents who display labels that are most trusted by the group as a whole. This is achieved by summing the weights of a particular label among each agent that is already a part of the group. After all agents have been placed in a group, each agent selects a strategy using a preference function and plays a single game. At the end of execution, all agents update preferences for strategies and labels based on the payoff received from cooperating. The payoff function is based on the level of cooperation that an agent displayed (investment), and the cooperation level displayed by the other agents in the group (gain). The resulting payoff value is positive or negative, and causes preferences for labels and strategies to increase, decrease, or stay the same. Trust for each label is updated based on the current trust value, previous payoff, and the number of agents in the current group. Afterwards, groups are disbanded and execution begins at a new time step. Modeling trust in this fashion allows for trust to be updated based on how well the previous time step produced updates that led to cooperation with more trusted agents.

Since KMAS agents use a computational function to reason about trust, trust will be computed using a trust update function. An agent will store a trust value for all agents that it has cooperated with. After each experience, trust is increased or decreased based on the results of cooperating.

المنارة للاستشارات
www.manaraa.com

3.3.2 TRUST EVOLUTION FUNCTION

In contrast to trust update in which agents use a current trust value for computations, trust evolution functions allow an agent to use a set of "remembered" experiences to derive a new trust representation. [Jonker and Treur, 1999] define a trust evolution function as a "mathematical function that relates sequences of experiences to trust representations". Trust evolution requires more computational overhead, but may be more desirable in cases where a potential partner's overall performance should be judged as opposed to the outcome of the most recent interaction. The trust evolution function, te, is defined as te: $ES \times N \rightarrow T$ where ES is the set of experience sequences, N is the set of natural numbers, and T is the set of trust qualifications.

Marsh also allows for the concept of dynamic trust that changes with experience of the action of other agents [Marsh, 1994]. Trustworthy behavior causes trust in an agent to increase, while untrustworthy behavior results in trust reduction. The three types of dynamic trust are basic trust, T_x, general trust $T_x(y)$ where agent x trust agent y, and situational trust $T_x(y,\alpha_x)$ for a given situation α where x must trust y to perform correctly in α_x. We now investigate the proposed formalisms of Marsh, which the author suggests may avoid ambiguities, aid in implementation of trust within an agent, and justify proposed theories with working examples.

المنارة للاستشارات
www.manaraa.com

Marsh's Formalisms

'x' and 'y' denote agents.

All values are in the range [-1,1]

basic trust: T_x (general trusting disposition of x)

situational trust: $T_x(y,\alpha_x) = T_x(y)U_x(\alpha_x)I_x(\alpha_x)$

Trust is informally defined as the probability weighted by UI that x acts to achieve an outcome as if it trusts y. General trust, $T_x(y)$, is an estimate.

$U_x(\alpha_x)$ is a utility function of costs and benefits, $C_x(\alpha_x)$ and $B_x(\alpha_x)$.

$I_x(\alpha_x)$ is x's measure of the importance of the situation.

general trust: $T_x(y) = (1 / |A|) * \Sigma_{\alpha \in A} T_x(y,\alpha_x)$

This equation sums all of the situational trust values for all the tasks in A where x computes a value for agent y. These are tasks in which x can allow y to participate in if x chooses to and y is a willing participant.

cooperation threshold: $CT_x(\alpha_x) = [R_x(\alpha_x) / (PC_x(y,\alpha_x) + T_x(y))] * I_x(\alpha_x)$

R = risk, PC = perceived competence

المنارة للاستشارات
www.manaraa.com

perceived risk: $R_x(\alpha_x) = (1 / |A|) * \Sigma_{\alpha \in A} (C_x(\alpha_x) / B_x(\alpha_x))$

perceived competence: Can be measured in three ways.

Equals basic trust if the agent is unknown

Equals $T_x(y)$ if the agent is known, but not in the situation being considered

If the agent and the situation are known, the following are factors:

1. experience of the trusting agent (x) in similar situations
2. experience of agent y in similar situations
3. capabilities of y in similar situations

If $CT_x(\alpha_x) \leq T_x(y,\alpha_x)$, agent x will cooperate with agent y.

Our first topic of discussion regarding the proposed formalisms will be the situational trust value. It appears that the dominant determinant for situational trust is the change in the term UI. We investigate this by asking the question; how can situational trust increase? Suppose agent y is initially unknown. We expect $T_x(y)$ to be very low. If a number of subsequent situations have low importance and utility values, situational trust will continue to be low, and it will also contribute to a low value for the recursive natured $T_x(y)$ which depends on past situational trust values. Therefore, an increase of utility or importance will increase both situational and general trust.

المنارة للاستشارات
www.manaraa.com

Marsh says that the general trust value is "a view of a particular agent of another with regard to the trusted agent's general capabilities." If the initial $T_x(y)$ is based upon y's capability, this is only a factor when the agent is first known. Still, $T_x(y)$ is updated by the UI product of future situations. We propose that in Marsh's case, trust is not a view of agent capabilities, but rather the amount of trust needed to cooperate with that agent. Therefore, to cooperate with an agent that has had historically low situational trust values, one or more of the following must occur upon judging $CT_x(\alpha_x) \leq T_x(y,\alpha_x)$:

1. $I_x(\alpha_x)$ must be low
2. $R_x(\alpha_x)$ must be low
3. $PC_x(y,\alpha_x)$ must be high

We can then conclude that trust is the actual threshold that needs to be overcome in order for cooperation to exist. Thus, an agent that has not been required to have large situational trust values in the past can become a partner in a highly important, high-risk situation if the appropriate competence is shown.

Along with trust, Marsh brings up the concept of "experience". He states that trust relies on judgment based on experience, and if known, past knowledge and behavior of the agent to be trusted. He describes the basic trust value as being "derived from previous experience", and is "dynamically altered in the light of all experience." Although never defined, experience in this sense is intuitively derived from the results

المنارة للاستشارات
www.manaraa.com

of interactions with other agents. The results cannot simply be observed trust values themselves or competence measures, because past experiences based on past tasks must be the result of trusted interactions where cooperation has occurred. To illustrate this, the following question is posed. Should an agent be distrusting in general because it has not entered into a cooperative agreement? On the contrary, an agent should gradually change its trust disposition in light of positive or negative experiences as a result of cooperation. It is unclear whether or not the general trust estimate uses situational trust from specifically successful interactions, or all potential interactions. Although it may seem contradictory to the above stated view of basic trust, we assume all potential interactions are "remembered". If not, general trust in an agent will never diminish, and this is not realistic. Therefore we define basic trust as an update function, but general trust as a trust evolution function.

While Marsh promotes the value of trust within agent cooperation, he concedes that trust alone is not a sufficient decision making criterion. He suggests that by adding other methods such as utility theory or theories of rational behavior, a more powerful and useful tool will be provided to the agents when judging potential interactions. There is also the matter of deciding how to obtain or calculate the initial trust value itself, or how to derive a value that represents an agent's capabilities.

المنارة للاستشارات
www.manaraa.com

SECTION 3.4: APPLICATIONS OF TRUST TYPES

3.4.1 TRUST IN INFORMATION

Trust has been used in MAS environments where researchers are concerned with system knowledge that is partial, incomplete, uncertain, incorrect, or originating from multiple, diverse information sources [Barber and Kim, 2001]. Information can also be dispersed through malicious intent in the case of non-benevolent agents. In instances where malice is not present, incompetence of an information source can lead to the presence of information that can be described as untrustworthy or non-credible. The following paragraphs will present two approaches to computing trust as it pertains to information and information sources. The first will model trust solely as it applies to the information given, while the second approach takes into account information and the agent information sources.

We have been introduced to the notion of trust as a means of validating information and its source to determine whether or not the shared information should be accepted. This concept has been presented in a form of trust-based learning [Primeaux, 2000] using an “actual entity” (AE). According to Primeaux, the AE is a process that "is identifiable by its state; changes state with each input, and outputs its current state." Primeaux asserts that AE's will tend to invest relatively more trust in input that is closer to the values in its current state. Input with values beyond a certain threshold range is

www.manaraa.com

ignored and the AE will not change its state. This implies that the values are not trusted. When the AE adapts its state, it is said to be learning. A variable representing the general trusting disposition of the AE towards the set of all inputs, is represented by a monotonically, non-increasing function that converges to 0 as the AE's state becomes less receptive to change.

3.4.2 TRUST IN INFORMATION SOURCES

[Barber and Kim, 2001] present a model of trust that takes into account an agent's confidence in another to provide correct information, as well as the reputation of the agent that is providing the information. The authors define trust as the "confidence in the ability and intention of an information source to deliver correct information", and reputation as the "amount of trust an agent gives an information source based on previous interactions among them". The information itself is weighed according to information certainty which is defined as the confidence with respect to quality of a statement. As a computed trust value, reputation can be increased by consistently providing trustworthy information to other agents. It can be decreased by incompetence or malicious behavior.

In the model, reputation of an information source S_1 is represented as $P(S_1^{reliable})$ and has the form of a probability distribution where $P(S_1^{reliable}) + P(S_1^{unreliable}) = 1$. The authors model a belief revision process based on information source reputation and two

المنارة للاستشارات
www.manaraa.com

types of agent belief bases. "KB" is the background knowledge base that contains knowledge that an agent has accumulated, and can be inconsistent. "K" is the working knowledge bases and it is a maximally consistent set of knowledge on "KB" which serves as a foundation for reasoning and decision processes. When an agent communicates knowledge "q", it sends the knowledge to be transferred, and the certainty that the sender has on the knowledge being true or accurate. The agent receiving the knowledge calculates its own certainty on "q" in "KB" based on information previously received from other agents. The receiving agent also uses reputation values for the agents that have supplied "q". If there are no conflicts in "KB", "q" enters K. If there is conflicting knowledge, the knowledge with the higher certainty enters "K".

Reputation of an information source is revised in two ways, indirect and direct. Indirect reputation revision occurs when there are conflicts between acquired knowledge. The resulting certainty of the conflicting knowledge is used to update the reputation of the sender. If certainty of the knowledge is revised to be higher than previously stored in "KB", agent reputation will be made higher. Conversely, if the certainty of the information is lower, agent reputation will suffer. The second means of reputation revision takes place if agents have the ability to revise their beliefs on the reputation of another by eliciting reputation belief from other agents. Here, indirect and direct refer to the process of revision. Later, we will see indirect and direct described as a form of agent interaction.

المنارة للاستشارات
www.manaraa.com

3.4.3 TRUST IN WARRANTORS AND AUTHORITIES

The coordination technique of organization establishes roles within a system of agents, and allows for the visibility of these roles to be available to all agents within the system. In the case of open systems, and particularly with internet based systems, agents that wish to interact within the system can be unknown at any given time if there is no barrier to entry. In such cases where agents are self-interested and utility maximizing, care must be taken to protect an agent from potential harmful and malicious actions. [Mass and Shehory, 2001] have proposed digital certificates as a way to dynamically update trust in potential partners as well as a way to verify capabilities claimed by other agents and to establish agent roles. Each agent may have one or more certificates certifying capabilities or performance. One may be issued by the developer, while others can be issued by 3rd parties who have used the agent's services, and can provide recommendations about performance and trustworthiness.

Review of these certificates takes place before interaction is allowed to occur. As an example, requester agent X sends a request to agent Y with certificates attached. The request may be to access some service or resource held by Y. Y sends the certificates to a role assignment module and the request itself to a deliberation module. The role assignment module retrieves role assignment policy, and according to the certificates presented, assigns a role to agent X. Agent Y can also request the certificates of other agents that issued the certificates to X. By doing this, agents can

المنارة للاستشارات
www.manaraa.com

establish whether or not the issuers of certificates are trusted. The deliberation module analyzes the request to find resources and actions needed for its fulfillment. Finally, the access control module takes the roles, resources, actions, and trust policy as input to determine whether or not to accept or deny the request from X. After results of the interaction are obtained, Y can downgrade the trust level of X, those who presented certificates from X, and those who granted certificates to X if the interaction results are not successful, or are harmful to Y. The updated trust policy can then be used to protect agent Y from future negative interactions. In the case of unknown agents, trustworthiness can be derived from the trustworthiness of other agents who have given certificates to the unknown agent.

3.4.4 TRUST IN ONESELF

Many implementations of trust concepts have been applied to trust as it relates to other agents. The research in [Lenzmann and Wachsmuth, 1997] answers the question: Can an agent have a measure of trust in its own capabilities? Their work describes a MAS system where agents customize themselves based on user preferences. The goal is to effectively automate user actions. The chosen cooperation framework is the contract net process where contractor agents receive announcements of tasks from manager agents. Contractors send bids in response to the manager which then chooses the contractor with the best bid to process the task. The authors define confidence as "the trust a contractor has doing the task successfully". This can be described as the

المنارة للاستشارات
www.manaraa.com

contractor’s view of its own abilities to meet the user’s current need. This measure is a function of performance with respect to the previous task, interaction history, and how well the user preference (that the agent embodies) fits in with the current situation. This takes into account the notion that a given user's preference may depend on situational circumstances. A higher confidence level will make a contractor's bid more attractive, while lower confidence weakens bids.

3.4.5 TRUST IN POTENTIAL PARTNERS

We will now investigate examples of applications of trust related to the trust that an agent must have in order to cooperate with potential partners. In our discussion of DAI systems, we indicated that DAI can provide a framework to test theories about reasoning processes. In [Nooteboom et al., 2001], the authors have devised a methodology called Agent-based Computational Economics (ACE) and define it as a process of “boundedly rational adaptation, based on mutual evaluation of transaction partners that takes into account trust and profits”. Economic activity emerges from the process of interaction between agents as they adapt decisions to past experience. Agents adapt the weight they attach to trust and their own loyalty as a function of realized profits. Trustworthiness is realized as a commitment to an ongoing trading partner relationship (loyalty). There is a threshold of resistance to temptation, below which an agent will not defect to a more alternative in terms of realizable profits. Profit can be increased by switching suppliers when products are differentiated. Agents may

المنارة للاستشارات
www.manaraa.com

incur costs associated with switching to a new trading partner in terms of loss of investments, and new investments that must be made. They also loose advantages gained by process improvements that normally occur between partners who have interacted in long term relationships over time.

Within ACE, buyer and supplier agents use matching algorithms to create potential relationships on the basis of individual agent preference rankings over other agents. Each agent assigns a score to all matches. Score = $\text{profitability}^{\alpha} \cdot \text{trust}^{1-\alpha}$ where $(1-\alpha)$ is defined as the weight attached to trust, and $\alpha \in [0,1]$. In the case of a buyer, an agent assigns a score to itself if it is able to produce the product that it wishes to sell. This score is based on potential profit and trust. Agents also use an adaptable importance measure that determines how important profitability is relative to trust. Any suppliers not ranked higher than the buyer himself are not acceptable. Buyers send requests to the most preferred suppliers, and suppliers accept requests from the most preferred buyers according to the allowable number of matches. Buyers continue to initiate requests until a supplier accepts.

As discussed, agents rationally choose partners based on potential profits. A buyer's potential to generate profit is based on its position on the final market when it is a seller. A supplier's potential to generate profit is determined by the supplier's efficiency in producing for the buyer. This efficiency can increase as buyers and suppliers gain knowledge of each other's processes and make improvements during

المنارة للاستشارات
www.manaraa.com

long-term relationships. Also, differentiated product allows buyers to increase profit margin by acquiring lower acquisition costs or selling products that can be priced higher to consumers. Trust is updated according to the law of diminishing returns during the uninterrupted duration of the relationship, and is not decreased until a trading partner defects from the supplier/buyer relationship.

In their research, the authors allow for the buyers to adapt the values used for the importance measure and the threshold of defection τ. They expected that adaptive agents evolve to relatively high levels of trustworthiness, less frequent switching, higher perceived commitment/trust, and a high weight attached to trust when evaluating partners. As observed results, during the 1st 25 runs with fixed product differentiation, the agents were found to migrate to three main locations in the problem space. The authors plot the problem space on a two-dimensional grid where the x axis plots α, and the y axis plots τ. Loyalty was decreased in the presence of a decreased weight attached to trust (increased α, decreased τ), loyalty was unchanged in the presence of an increased weight attached to trust (decreased α, stable τ), or loyalty was increased in the presence of a decreased weight attached to trust(increased α, increased τ). This showed that agents may place value on strategies of trust, loyalty, and opportunism. An opportunistic agent may not gain a large amount of trust in the eyes of others because the opportunistic agent continuously breaks relationships and switches partners. However, such agents still receive profit based on the short term advantages of selecting suppliers that help them increase profit margins through lower costs. Agents that place

المنارة للاستشارات
www.manaraa.com

a high value on loyalty and profit achieve higher profit margins as a result of improvements gained during long-term relationships. As a result, these agents will appear to be more trusted in the eyes of their partners. An average of all 25 runs tended to show regions of both higher and lower loyalty as well.

Another example of modeling DAI to understand reasoning processes is the BDI agent model discussed in Chapter 1. When a BDI agent forms an intention to achieve a given goal, it does so by committing to a plan to achieve the goal. In general, the plan is chosen from a plan library, which is composed of partial plans that are incomplete and contain both actions and subgoals. [Griffiths and Luck, 1999] propose a way that trust can be used as one of the deciding factors when choosing between competing plans. In particular, this is very important when an agent must decide between a plan requiring cooperation and a plan where cooperation is not necessary. The perceived risk of cooperation with an agent is measured by trust.

$R = 1/T$, Risk is inversely proportional to trust. $T \in [0,1]$

Each agent has a representation of other agents which forms part of the agent's beliefs. This information is comprised of an agent id, agent capabilities (such as: able to perform tasks, x, y, and z), and the trust value that the agent has in the other. Before a plan can be chosen, both a standard and a cooperative rating must be calculated. The standard rating can be assessed using heuristics such as "length of plans as the number

المنارة للاستشارات
www.manaraa.com

of actions", "cost based on cost of actions", and "duration of plan execution based on duration of individual actions". For the cooperative rating (if cooperation is not necessary to perform the task, the rating will equal 0), agents in the set $\{\alpha_1, \alpha_2, ...\alpha_n,\}$ are ordered by trust such that $T(\alpha_{X-1}) \geq T(\alpha_X)$. $T(\alpha_X)$ denotes the trust that the trusting agent has in agent α_X. These agents are those that are found to have the capabilities needed to perform the action. Thus, the risk associated with the action is 1/ (Σ from i = 1 to n of [$T(\alpha_i)/i$]). This avoids considering the most trusted agent only, which may not be the actual partner at the time of execution. It also provides for the most trusted agents to have a greater bearing on the risk of cooperating. For a plan with m actions, $a_1, a_2, ..., a_m$, the cooperative rating $C = \Sigma_{(\text{from } i = 1 \text{ to } m)} R(a_i)$.

Once the ratings have been established, an overall plan quality measure Q is calculated using $Q = (w_s * S) + (w_c * C)$. The weights w_s and w_c vary for each agent and have the effect of allowing them to have a deeper level of rationality. For example, an agent that places a high importance on minimizing costs will place a greater importance on the standard rating. The authors use the plan quality measure Q in two distinct techniques to elaborate plans in the plan library with a "pre-execution assessment" of the entire library. This can be accomplished while an agent is not occupied, or when the change in trust of the other agents exceeds a threshold.

As with [Marsh, 1994], the approach of the authors does not take into account how trust in another agent is actually computed, but only that it is based upon factors

المنارة للاستشارات
www.manaraa.com

such as agent capability or competence. One difference is that utility and importance can be extracted out into the standard plan rating and kept separate from trust derivation. While Marsh focuses on selecting a single agent to cooperate with, [Griffiths and Luck, 1999] take into account all agents that can be cooperated with. The advantage is that at execution time, we are not relying only on a single agent for execution. Marsh also has no way of determining which agent to cooperate with if more than one agent has a sufficient trust value. Griffiths and Luck do not either, but are able to assess the risk of multiple potential partners as a whole when determining plan selection. They ignore the issue of updating trust and deem situational trust as being too computationally expensive. Whether or not this is the case, the authors will allow for cooperation with the least trusted agent involved in the cooperative plan rating.

This type of cooperation may not be desirable. However, situational trust could provide a barrier against this. Since computational overhead might limit taking situational trust into account for every action in the plan library, perhaps it might be done only for tasks that have importance greater than a pre-defined threshold. In any case, there are many similarities and differences between this usage of trust within BDI agent architecture and the research investigated thus far. The main relevant concept for this BDI architecture is that multiple plans are distinguishable in part through trust. Multiple agents can be considered when cooperation is necessary, and as a result, Marsh's work can be extended.

للاستشارات المنارة
www.manaraa.com

In the example in Section 3.4.2, [Barber and Kim, 2001] showed that an agent can use reputation as a measure of trust with respect to an information source. Even the example of certificate-based trust benefits from reputation as unknown agents present certificates issued by other agents who can vouch for its capabilities, identity, and trustworthiness. [Barber et al., 2003] identifies key challenges for systems that employ reputation-based trust models. Cooperation in uncertain environments exposes risk in the form of inaccurate information or failed goal realization. Reputation-based interactions that exist only through direct interaction between truster and trustee pose risks until the trust model allows for recognition of an agent that should not be trusted. This type of model forces agents to undergo repeated exposure to negative interactions until trust values can converge to appropriate levels.

The second form of reputation is recommendation-based reputation. This form of interaction is not dependent on direct interaction in the long-run. Risk is still present because some default reputation value must be determined for agents that are totally new and unknown to the system. This default value can only be computed through direct interaction. An agent must also trust the recommendations received from other agents, and at the same time, assess the trustworthiness of the source of the recommendation. Agents must also have criteria that allow them to seek out other agents who will provide recommendations. If an agent is not new to the system, direct interaction must still occur to build the appropriate base of recommendations, and to

المنارة للاستشارات
www.manaraa.com

allow those recommendations to deliver the accurate trustworthiness of an agent to others. The authors identify recommendation-based reputation as advantageous over direct interaction models. Overall, recommendations allow the truster to form reputation without being exposed to the risks of direct cooperative interaction, and the system as a whole has as cheap, low-risk way of communicating knowledge.

As a concrete example, [Mui et al., 2003] provide experimentation to compare the performance of agents that used varying reputation models. They describe recommendation-based reputation as being derived indirectly or by word-of-mouth, and having its value propagated through the system based on information from others. The authors seek to discover which notion of reputation provides the highest utility using the Prisoner's Dilemma game. They identify four types of reputation modes: encounter-derived (direct interaction), observed individual, group-derived, and propagated. In observed individual reputation, agents designate a random number of other agents as being observed. All encounters by these agents are observed and recorded. Reputation is derived by dividing the number of times cooperation has occurred by the number of defections. The only interactions used in the calculation are those between the observed agents and the agent whose reputation is being calculated. For group-derived reputation, all agents with the same characteristics such a cooperation strategy, are grouped together in the eyes of the agent that is determining the agent reputation value. As with the observed reputation measure, only the interactions between agents in the group and the potential partner are counted. Reputation is determined by dividing the

المنارة للاستشارات
www.manaraa.com

number of times cooperation has occurred with group members by the total number of encounters with the given agent. Using propagated reputation, agents will recursively ask past interaction partners for reputation estimates of the unknown partner whose reputation must be calculated. For both group and propagation, after the first encounter, all subsequent decisions are made using encounter-derived reputation. All agents have a threshold of reputation below which they will defect instead of cooperating with an undesirable partner. The authors found that propagated reputation outperformed the other reputation-based strategies. One reason is that direct-interaction, as discussed previously, does not converge fast enough to weed out undesirable partners. It was also found that by expanding the number of recommendations gathered, performance was further increased.

KMAS will use reputation as a measurement of trust and a determinant for cooperation with potential partners. Trust will be updated through indirect and direct revision. Indirect revision will occur when trust is updated based on the completion of a cooperative task or action through direct interaction with an interaction partner. Direct revision will be achieved through recommendation-based reputation which will be performed for all unknown agents as a KMAS system default. KMAS execution can be parameterized to perform direct revision for known agents as well. Recommendation-based reputation will be propagated throughout the system as KMAS agents interact and engage in cooperative tasks, and solicit the reputation of potential interaction partners in the form of recommendations from other agents. The revision of trust provides the

المنارة للاستشارات
www.manaraa.com

adaptive mechanism for intelligent, adaptive KMAS agents. The KMAS experiment will seek to determine whether or not the KMAS model of recommendation-based reputation will also be advantageous over a reputation model based solely on direct interaction.

www.manaraa.com

CHAPTER 4 MACHINE LEARNING

SECTION 4.1: OVERVIEW

The field of machine learning studies computational processes that result in learning in both humans and machines. Machine learning has been used in problem domains such as speech recognition, problem solving, data mining, motor control, and game playing. [Langley, 1996] identifies four basic goals of researchers. The first goal is psychological in nature. Researchers develop learning algorithms that model human cognitive architecture, and by doing so, can use this knowledge to explain specific, observed learning behaviors. An example is an artificial neural network which is computationally analogous to the complex web of neurons in the human brain. The second goal is empirical, and aims to discover general principles that relate the characteristics of learning algorithms, and the domain in which they operate, to learning behavior. This area of research basically compares and contrasts different learning methods to provide generalizations about alternatives, methods, areas of weakness, sources of task difficulty, and ideas for improved algorithms. The mathematical goal involves formulating and proving theorems about the characteristics of entire classes of learning problems and the algorithms applied to solve them. This goal is the groundwork for developing a computational theory of learning. The fourth and final

المنارة للاستشارات
www.manaraa.com

goal is the application of machine learning techniques to real-world problems. An example would be automating the process of knowledge acquisition. This paper uses the following definition of machine learning found in [Mitchell, 1997]. “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, measured by P, improves with experience E.” In the KMAS experiment, experience will be represented by direct interactions between agents. Tasks are simple cooperative tasks involving an agent and its interaction partner. As previously indicated, KMAS seeks to use trust as a cooperation strategy to achieve the DAI goal of increasing task completion rate. Performance will be measured by the task completion rate determined by successful or unsuccessful outcomes of agent cooperation.

للاستشارات المنارة
www.manaraa.com

SECTION 4.2: TYPES OF LEARNING

Various forms of learning and feedback allow an agent to adapt its behavior or learn new concepts. The following descriptions are widely accepted concepts, and one or more terms can be found in most works contributing to the field of machine learning. In terms of the generic machine learning definition proposed by [Mitchell, 1997], a machine learning algorithm learns from experience in the form of a set of training examples. In an example learning problem, an algorithm can be designed that will allow a system to learn verbal commands after repeatedly receiving input from various users, and processing electronic speech patterns. After training concludes, each new instance of the problem domain must be classified by generalizing beyond the training data. Each classification is directly related to the tasks T and the performance measure P. Generalizing in computational terms may be described as using the training data as input to approximate the learned target function. This generalization can occur in two ways. In lazy learning, generalizing is done at runtime for each new instance of the problem. Eager learning techniques generalize beyond the training data before any instances are classified. In this latter case, after training ends, there is a fixed global approximation of the target function. This can be found, for example, in the fixed network weights established by Artificial Neural Networks that are used to classify all new problem instances. k-Nearest Neighbor is an example of a lazy learning method. Such methods are advantageous from the standpoint of not being constrained to a global approximation of the target function. Lazy methods can use many local approximations

www.manaraa.com

to the target function. This is helpful in instances where global approximations may be over fitted to a certain area of the search space, and do not perform well outside this area.

Learning strategies can be described by five areas of distinction in terms of how learning is acquired. In *rote learning*, direct implantation of knowledge and skills is given without requiring further inference or transformation from the learner. Through *learning from instruction* and by taking advice, a learner can transform knowledge into an internal representation, and combine it with existing knowledge and skills. *Learning from examples and practice* allows existing knowledge and skills to be refined by positive and negative examples or practical experience. The KMAS system will use learning from examples and practice to refine knowledge about other agents in the form of trust. *Learning by analogy* allows solutions for unsolved problems to be derived from similar, solved problems. The last strategy, *learning by discovery*, allows for the gathering of new knowledge and skills by observations, conduction experiments, and the generating and testing of hypotheses or theories based on observed and experimental results.

Feedback allows the learner to measure performance levels achieved so far with respect to the class of tasks in T. In *supervised learning*, the feedback specifies the desired activity of the learner. The objective of learning is to match the desired action as closely as possible. An example of this was found in the VIENA system to correct

المنارة للاستشارات
www.manaraa.com

agent actions. The system as a whole received supervised learning in the form of user supplied language such as "a bit less". In reinforcement learning, feedback only specifies the utility of the actual activity of the learner and the objective is to maximize this utility. Feedback is given by a critic that has the ability to determine appropriate utility measures. The learning agents in VIENA are given this type of feedback in order to place credit or blame for agent actions that result in correct or incorrect system responses. *Unsupervised learning* does not have explicit feedback. Agents must learn to improve performance by trial and error or self-organization. In the KMAS environment, agents learn the trustworthiness of other agents through direct interaction by practical experience (unsupervised learning), and by using indirect-supervised learning where other agents are advisors. Classical supervised learning allows teachers to provide target function classifications based on examples. In the traffic signal research which will be presented in the next section (Section 4.3), it is suggested that peer advice can provide output that can be back-propagated to update neural network weights in an advisee agent that is requesting advice from advisors. For KMAS, neighbors are actually providing advice that is input into the target function (which is the act of computing k-nearest neighbor) instead of providing the result of the application of the target function. In this paper, this type of process is described as indirect-supervised learning because inputs in this sense are one step removed from the target function classification.

المنارة للاستشارات
www.manaraa.com

SECTION 4.3: MAS LEARNING

As demonstrated by VIENA, there are advantages to developing multi-agents systems that learn and adapt. In VIENA's case, a single system can learn to adapt to multiple users. A further development of this area of adaptive systems exists in the combining of the fields of machine learning and multi-agent systems. Merging the two areas of research presents distinct challenges as well as advantages. Specifically, in [Vidal, 2003], it is stated that the definition of machine learning is essentially violated within multi-agent systems because an agent is no longer learning from a fixed set of experiences (training examples). Since E changes, the learned target function changes. [Singh and Huhns, 1997] identify differences between challenges faced by traditional machine learning research and research involving machine learning within cooperative agent systems. In a traditional agent-based, machine learning system, an agent must learn and adapt to an environment that is passive and has no intentions. The agent may also have imprecise sensors that cause it to learn inaccurate information about the environment. In machine learning with systems of multiple agents, an agent learns about its environment which is active, because it includes other agents who have intentions, commitments, beliefs, abilities, and can also learn. An agent might also be deliberately misled about the environment by other agents. The different challenges highlight the fact that learning has moved from being single-agent oriented to multi-agent oriented. [Weiss, 1995] describes the two types as isolated learning and interactive learning, respectively. Agents in a MAS can learn communally because

المنارة للاستشارات
www.manaraa.com

learning can be influenced by exchanged information, shared assumptions, commonly developed viewpoints of their environment, and commonly accepted social and cultural conventions and norms. Weiss also identifies two problems that researchers must address when determining the source of impact on performance. Credit (or blame) for an overall performance change must be assigned to an external agent to agent interaction, or credit (or blame) for an action must be assigned to an internal agent decision.

There are two major areas of application of machine learning techniques to multi-agent systems: learning to coordinate or cooperate, and learning from other agents through the exchange of information (cooperative learning) to improve learning performance of each agent, or the system as a whole. [Nunes and Oliveira, 2003] seek to perform the latter by modeling human cooperative learning in a team based on the exchanging of advice. The authors employ agents that are heterogeneous with respect to learning algorithms in the hope that different algorithms solving similar problems may lead to different forms of exploration of the same search space, increasing the probability of finding a good solution. The problem domain is a simplified traffic-control problem where each agent must control four traffic lights at an intersection. Learning parameters are adapted using two methods: 1) reinforcement-based, unsupervised learning using a quality measure that is directly supplied by the environment, and 2) supervised learning using peer advice as the desired response. Agents request advice when their current average quality since the beginning of the

المنارة للاستشارات
www.manaraa.com

present time epoch drops below a certain percentage of the best average quality reported by its peers at the beginning of the present epoch. Average quality is assessed at the beginning of each green-yellow-red traffic cycle. Quality is determined by how well the agent has managed the traffic flow. When advice is requested, the advisee sends the current state of traffic to the advisor who has the best overall score reported at the start of epoch. The advisor then switches its internal learning representation back to what was reflected at the beginning of epoch, and runs the state communicated by the advisee to give advice in the form of a suggested response to the current state. For a neural network implementation, this would simply involve setting the network weights back to the values present at the beginning of the epoch for the advisor. The advisee would then use the response to update its own internal learning representation. In the case of a neural network implementation, the advisor's response would be backpropagated to adjust network weights accordingly. The researches found that advice exchange causes a fast increase of quality at early stages as good responses are shared. After comparing against agents that employed stand-alone, isolated learning, it was found that advice seeking agents fall less commonly into local optima because they are better at overcoming bad initial parameters. This is due to the fact that supervised learning allows exploration of more promising regions of the search space. This is an important benefit of supervised learning that will be discussed in Section 5.2.3 with experiment examples where KMAS performs direct revision for known agents.

المنارة للاستشارات
www.manaraa.com

Along with cooperative learning, researchers have found machine learning techniques as valuable tools to aid in the coordination process of multi-agent systems. Traditional coordination mechanisms such as negotiation must rely heavily on communication between agents. [Bazzan, 1997] identifies this communication bottleneck as a major shortcoming in existing coordination frameworks. Bazzan hopes to demonstrate research that minimizes or even eliminates the need for communication when coordinating agent activities. Like our first example, the problem domain is traffic-control, but only one learning technique is used, and agents do not communicate. Agents only know their own utility payoffs, and not those of others. Reinforcement learning is applied by way of a critic, “nature”, that provides local and global payoff utility. The global payoff utility acts as an incentive to coordinate toward the global goal of stabilizing coordination such that traffic flows as long as possible without stopping at red lights.

The learning algorithm is a genetic algorithm that models strings of chosen strategies employed in the past. During the learning process, a fitness for each string is computed, and this influences the next generation of strategies used. Fitness is determined by calculating the cumulative payoff of a specific strategy available, with increasingly discounted payoffs for strategies chosen farther in the past. This specific strategy is then compared against the cumulative payoffs of all strategies. Payoff is only calculated for the time interval between the current learning period and the last time period where a change in normal traffic pattern was determined. At the beginning

المنارة للاستشارات
www.manaraa.com

of each time step, if a change in local or global traffic pattern has not occurred, and a learning period has not started, each agent will act according to a strategy chosen by fitness. This strategy will yield a payoff determined by nature, and will be used in subsequent learning periods. If a change in normal traffic pattern occurs, strategies are chosen according to the direction of the highest flow of traffic. A strategy simply corresponds to giving more green time to a certain direction of the traffic flow.

The researchers found, not surprisingly, that coordination is reached faster when global traffic pattern seldom changes. It was also found that higher learning frequency (more learning periods) provided a good counter measure to environments with higher rates of individual traffic pattern changes at each intersection.

المنارة للاستشارات

www.manaraa.com

SECTION 4.4: MACHINE LEARNING and TRUST

The similarities between trust and machine learning research provide interest in research aimed at combining the areas of trust, machine learning, and multi-agent systems. In particular, this paper will now present research that uses an adaptation of the k-Nearest Neighbor machine learning algorithm described later in Section 4.1 to provide recommendation-based reputation of unknown agents. The nearest neighbor algorithm will allow the intelligent agents to reason about the trustworthiness of other agents along with direct interaction-based reputation and a trust update function. The k-Nearest Neighbor algorithm is also part of the adaptive mechanism of KMAS agents. It is theorized that recommendation-based reputation of unknown agents can provide some protection from non-benevolent and potentially malicious interaction partners within multi-agent systems.

المنارة للاستشارات
www.manaraa.com

CHAPTER 5 K x N TRUST-BASED AGENT REPUTATION

SECTION 5.1: k-NEAREST NEIGHBOR and EXPERIMENT HYPOTHESES

In [Mitchell, 1997], the k-Nearest Neighbor learning algorithm is described as a lazy learning method that uses stored training examples that are similar to the new instance that needs to be classified. The algorithm assumes that all instances correspond to points in the entire instance or problem space. The nearest neighbors are the ***k*** closest training example instances with respect to the Euclidean distances between ***k*** neighbors and the new instance. The nearest neighbor values are used to make a local approximation of the target function.

k-Nearest Neighbor is performed by one agent learning in isolation. In this paper, we first adapt k-Nearest Neighbor by changing it to (k X n) Nearest Neighbor. Because the agent is now learning in an interactive environment, other agents are learning as well. If there are ***n*** agents in the system that are learning one at a time, then (k X n) neighbors are used to approximate ***n*** target functions. In our research, each agent must classify an unknown agent as being either trusted, or distrusted. A simple application of (k X n) would be to store instances composed of a tuple containing agent

www.manaraa.com

characteristics, an action, and a classification of that instance with respect to the target function. A new instance would be approximated as being trusted or distrusted according to the classifications provided by the k-nearest neighbors. In Section 5.3, future research, an implementation of KMAS is proposed that would use tuples of agent characteristics.

The second adaptation in this paper is more radical. This paper is focused on the Euclidean position of the agents themselves within the search space which might also be called the “instance space”, or the “agent space”. We can still choose neighbors based on their agent characteristics, but these characteristics must be close in Euclidean distance to the agent that needs to perform the classification as opposed to being closest to the agent that needs to be classified. As in [Primeaux, 2000], this models increased trust in neighbors who are “alike”. In human society, this is similar to the increased trust that one would have in human neighbors situated in the same living environment and persons that are similar in characteristics such as age, occupation, social status, income, etc.

In this research, the local approximation of a target function changes from being derived from unsupervised learning examples, to being derived from advisors engaged in indirect-supervised learning. Learning is an activity that each agent and the system as a whole participate in. Each agent classifies others using a trust-updated function refined by the feedback of other agents after execution of k-Nearest Neighbor. If the

المنارة للاستشارات
www.manaraa.com

system were viewed as a single agent with the task of coordinating system components such that harmful interactions were not allowed to occur, it would be expected that the emergent property of the system as a whole would be the learning of all untrustworthy agents. Learning would be achieved by an application of k-Nearest Neighbor, where the coordinated system components are learning, adaptive agents in the system environment, neighbors are agent advisors that provide trust recommendations, the target function represents the trustworthiness of an individual agent, and training consists of interactions that occur during system life cycles.

Using the two adaptations of k-Nearest Neighbor, this research attempts to investigate the benefits of using (k X n) Nearest Neighbor as a model of recommendation-based agent reputation. As stated in Section 4.4, it is theorized that recommendation-based reputation of unknown agents can provide some protection from non-benevolent and potentially malicious interaction partners within multi-agent systems. The following hypotheses represent the foundation for experimentation that this paper will discuss. The experiment results will be used as an attempt to justify or explain the benefits that may arise during the usage of the KMAS model and (k X n) Nearest Neighbor.

المنارة للاستشارات
www.manaraa.com

Hypotheses:

- **1.1** - A system performing k-Nearest Neighbor will outperform a system that does not perform k-Nearest Neighbor, where performance is measured by the system's ability to only allow cooperation between a requester agent and a partner that is non-deceptive.

- **1.2** - Over time, a system using trust-based agent recommendation will converge towards a state where cooperation with deceptive agents will not occur.

- **2.1** - The number of executed life cycles needed to reach maximum failure rate will decrease as the number of nearest neighbors increases, despite randomness in both interaction relationship pairings and when new agents are made active in the system.

- **2.2** - Curve slope, as a measure of average velocity and calculated by ($y_2 – y_1$ / $x_2 – x_1$) where y's represent the range of failure rates and x's represent the range of time steps, will increase as the number of neighbors increases.

- **2.3** - As the number of neighbors increases, elapsed time in life cycles between the maximum failure rate and the benchmark failure rate (Elapsed Time_B) will decrease as the number of neighbors increases.

- **3.1** - The number of executed life cycles needed to reach maximum failure rate will decrease as the learning rate decreases, allowing more exploration.

- **3.2** - - Curve slope, as a measure of average velocity and calculated by ($y_2 – y_1$ / $x_2 – x_1$) where y's represent the range of failure rates and x's represent the range of time steps, will increase as learning rate decreases (exploration increases), indicating a greater return. This result is expected for both time periods between max failure rate and relative convergence (Elapsed Time_R), as well as the period between max failure rate and the benchmark failure rate (Elapsed Time_B).

- **3.3** - Elapsed time between the maximum failure rate and the benchmark failure rate (Elapsed Time_B) will decrease as the learning rate decreases.

المنارة للاستشارات
www.manaraa.com

SECTION 5.2: KMAS

5.2.1 EXPERIMENT DESCRIPTION

Terms and Definitions

KMAS – An MAS that uses the k-Nearest Neighbor algorithm where nearest neighbors are agents $a_i, \ldots\ldots, a_k$ where a is an agent in A, the set of all agents active in the MAS.
ETIP – Experimental trial input parameter file. The file that determines execution parameters to execute life cycles in the KMAS experiment.
LIFE CYCLE – A single execution of the KMAS environment where inter-agent interaction will take place.
LEARNING RATE – η_l, the number of life cycles between trust explorations is equivalent to learning rate – 1, or exploration occurs during every i^{th} life cycle where the integer ***i*** is the learning rate.
EXPLORATION – The process of performing the k-Nearest Neighbor algorithm for known agents. Determined by learning rate. During exploration, recommended trust values (agent reputation) replace general trust if reputation is the lower of the two values. Exploration increases as the learning rate decreases.
INTERACTION – The process of selecting a partner and engaging in cooperation with that partner if cooperation is desired. If cooperation is refused by the requester, this is still part of the interaction process.
COOPERATION – The process of participating with another agent (partner) to accomplish some goal or task through interaction.
BASIC TRUST – T_x where basic trust is the trusting disposition of agent x towards society. A global attribute as defined in Section 3.2.
BASIC RISK – R_x where basic risk is the disposition of agent x towards involvement in potentially harmful interactions with deceptive interaction partners. A global attribute as defined in Section 3.2.
SITUATIONAL TRUST – $T_x(y,\alpha_x)$ where situational trust is the calculated trustworthiness of agent y during interaction α from the perspective of agent x. $T_x(y,\alpha_x) \in Q : 0 \leq T_x(y,\alpha_x) \geq 1.0$,
$T_x(y,\alpha_x) = (T_x)(T_x(y))$
GENERAL TRUST – $T_x(y)$ where general trust is the general trustworthiness of agent y in the eyes of agent x.
$T_x(y) \in Q : 0 \leq T_x(y) \geq 1.0$
TRUST UPDATE RATE – η_{tu} where trust update is a term used to scale the impact of successful or unsuccessful interactions on general trust.

www.manaraa.com

DECEPTION – Degree or level of agent deceptiveness or propensity to act deceptively or defect during a cooperative task or goal.
DECEPTIVE THRESHOLD – A measure of deceptiveness where a deceptive agent will practice deception if its personal level of deception is above this value.
TIME STEP – Unit of time equivalent to the time needed to execute one system life cycle.
REQUESTER or REQUESTER AGENT– An agent that initiates a request for interaction by selecting an exclusive, potential interaction partner among the active agents in KMAS. An agent designated as a requester cannot be selected by another requester agent during the same life cycle.
RELATIVE CONVERGENCE – The point at which subsequent KMAS executions (life cycles) are completed without the presence of interactions with harmful/deceitful agents, or such interactions occur in "extreme rarity", where "extreme rarity" is subjective to the conductor of the experiment.
UNKNOWN AGENT – From the perspective of a requester agent, an unknown agent is an agent that has not been interacted with.
FAILURE RATE – Cumulative measurement of system performance at a given life cycle. Determined by taking the total number of failures experienced during KMAS current and prior executions, and dividing it by the current time step which also serves as the elapsed time in execution life cycles.
MAXIMUM FAILURE RATE - The maximum observed failure rate among trial time steps after initial, local max failure rates have been produced. Local max failure rates may occur when an agent first begins activity in the system. Early interactions may involve many encounters with deceptive agents, thus producing an artificial maximum failure rate. In these cases, failure rate will decrease, then peak again at a later time step. The latter time step is chose as the maximum failure rate.
K[x] – indicates the usage of the k-Nearest Neighbor algorithm in an experiment trial where ***x*** is the number of nearest neighbors allowed. If n is equal to zero, the effect is that k-Nearest Neighbor is never performed.
KE[x,y] – indicates the usage of the k-Nearest Neighbor algorithm with exploration in an experiment trial where ***x*** is the number of nearest neighbors allowed, and ***y*** is the value for exploration equivalent to the learning rate η_l.

المنارة للاستشارات

www.manaraa.com

Experiment Overview

KMAS represents an attempt to model an environment of intelligent, autonomous, rational, adaptive, and cooperating agents within a MAS distributed agent architecture. The KMAS agents use machine learning and a trust update function to reason about whether or not to cooperate with other agents on the basis of trustworthiness, and to adapt to the dynamic nature of trust. The agents use k-Nearest Neighbor as a machine learning algorithm to model recommendation-based agent reputation where reputation of individual agents is propagated throughout the system. Reputation is used as a measure of trust that is updated as agents revise their beliefs about other agents. Agents learn the trustworthiness of other agents through direct interaction by practical experience, and by using the nearest neighbor algorithm as a form of indirect-supervised learning where other agents are advisors. The KMAS system as a whole learns by performing the nearest neighbor algorithm ***n*** times where the integer n is the number of active agents performing local k-Nearest Neighbor at a given time step of execution. KMAS performs (k X n) Nearest Neighbor to improve performance in terms of only allowing cooperation with trusted agents. Thus, KMAS uses trust as a cooperation strategy. The system goal is to converge towards a system state that only allows cooperation between a requester agent and a non-deceptive agent. In order for this to occur, any requester agent must recognize a deceptive agent as untrustworthy. A “deceptive agent” is defined as an agent that will defect from a cooperative task after it is assumed to be committed to the interaction. As a consequence, cooperation with a deceptive agent will result in failure.

المنارة للاستشارات
www.manaraa.com

Cooperative tasks are assumed, but not defined. Upon completion of agent cooperation, the result of the assumed task is reported as a success or failure to the requester. It is theorized that agents performing the k-Nearest Neighbor algorithm will increase task completion rates by avoiding harmful interactions with "deceitful" agents and will outperform systems that do not use recommendation-based reputation.

All experiments in this research used the same fixed inputs and the contents of file fixedIn.txt which is explained later in this section. Additionally, some ETIP file inputs (also explained later in this section) were fixed. These fixed inputs describe the number of agents in the MAS, the maximum number of executable time steps, the maximum number of agents initially active in the MAS, the number of deceptive agents, weights used in the k-Nearest Neighbor algorithm (age, successful tasks, basic trust, and risk), and the rate of trust update. The number of agents in the MAS differs from the number of agents initially active. Upon creation of the KMAS environment, a "pool" of available agents is created. The maximum number of agents in the "pool" is equivalent to the fixed input value that describes the number of agents in the MAS. KMAS randomly selects agents from the agent pool until the maximum number of allowable, "initially active" agents is met. An "initially active" agent is an agent that is allowed to execute within the KMAS environment starting at the beginning of the first time step. Initially "inactive" agents are not allowed to execute until they are randomly chosen after completion of the first time step. Input values that are allowed to change are the number of nearest neighbors as described by the value $\boldsymbol{x}$ in the definition of the

المنارة للاستشارات
www.manaraa.com

symbol **K[x]**, and the exploration value identified by the learning rate η_l or the symbol **KE[x,y]**, where ***y*** is equivalent to the learning rate.

Although fixed inputs were used, a discussion is warranted to describe the effects of changing the values of these inputs. What is the effect of adding more or less agents and maximum time steps to experiment input? In general, one could expect that adding more agents to the KMAS environment may aid an agent in its goal of avoiding harmful interactions. Because agents randomly select other agents as potential cooperative partners and share interaction experiences in the form of reputation, an assumption can be made that by the time an agent randomly selects a deceitful agent, enough interactions have occurred with neighbors to properly model a reputation of "distrust". This is based on a statistical assumption that as the number of active agents in the system increased, the chances of choosing a deceitful agent as a potential partner decreases if the number of deceitful agents remains fixed. However, the first assumption is possibly flawed if the number of nearest neighbors is not increased along with the number of agents in the MAS. Based on recorded and unrecorded experiment results, adding more agents and maximum executable time steps creates more opportunities for meaningful and measurable experiment results. In particular, a minimum number of time steps is needed to allow KMAS system convergence as described in the beginning of the experiment overview.

المنارة للاستشارات
www.manaraa.com

As described in Section 5.1, neighbors are chosen based on the Euclidean distance between the characteristics of the agent soliciting reputation, and the agent characteristics of prospective neighbors. In this research, the agent characteristics or attributes are defined as age (number of time steps the agent has been active), successful tasks (the number of cooperative tasks resulting in success), basic trust, and basic risk. The nearest neighbor algorithm calculates the Euclidean distance based on these attributes and weights. The weight of each attribute decreases or increases its contribution to the Euclidean distance. Although arbitrarily fixed and unstudied for purposes of this research, the usage of weighted agent attributes may be an important way to measure the behavior of certain classes of agents. This is discussed in Section 5.3, Future Research.

The final fixed input that produces an impact, if changed, is the trust update rate. Trust (here general trust) is updated based on direct interaction if a partner is unknown. Before the addition of the trust update rate, trust was discounted too quickly towards total "distrust", represented by a low general trust value. A low general trust value produces a low situational trust value. In early experiments, this had the effect of stopping interactions with deceptive agents after the first interaction experience. Although desirable in concept, this was not conducive to recording and measuring experiment results. During experiments where exploration was used, the absence of a trust update rate did not allow the acceptance of recommended trust values from nearest neighbors. The reason is that general trust was already lower than the calculated

المنارة للاستشارات
www.manaraa.com

reputation value which was based on an average of neighbor responses. During periods of exploration, the lower trust value will be accepted as the new general trust value, and will be used in subsequent calculations. A lower trust update rate produces a smaller increase or decrease of general trust after cooperation.

After all fixed inputs were chosen, three experiments were performed. Within each experiment, each execution of KMAS with a unique set of inputs represents one trial. In all, three trials are defined for each experiment. Upon execution of each trial once, the results are grouped together and repeated to create experiment groups A, B, and C for each experiment. Each group represents an execution of three trials. Experiment 1 compares KMAS execution with that of a MAS that does not use the nearest neighbor algorithm. Experiment 2 compares KMAS with different numbers of allowable nearest neighbors per trial. Experiment 3 compares KMAS with different values for exploration in each trial. To measure performance, relative convergence will be used as a point in time. As defined previously, relative convergence is the point at which subsequent KMAS executions (life cycles) are completed without the presence of interactions with harmful/deceitful agents, or such interactions occur in "extreme rarity" where "extreme rarity" is subjective to the conductor of the experiment. Relative convergence is measured in elapsed life cycles. For example, a relative convergence point of 1000 indicates that relative convergence has been reached at the 1000th life cycle or time step. Additionally, a benchmark failure rate is used for performance comparisons as well. The benchmark failure rate is an actual, measured failure rate

المنارة للاستشارات

www.manaraa.com

recorded in the failure rate log as described in the list of experiment reports found later in this section. Time is defined as elapsed or completed system life cycles. Each trial is depicted by a graph where the y axis represents failure rate, and the x axis represents time in life cycles with an offset of + 1 where life cycle 0 (first executed life cycle according to the failure log) is recorded as time step 1.

Experiment Inputs

fixedIn.txt (See Appendix A)

1) **Basic Trust**, where $T_x \in Q : 0 \leq T_x \geq 1.0$
2) **Risk** (same as Basic Risk), where $R_x \in Q : 0 \leq R_x \geq 1.0$
3) **Deception**, where $d_x \in Q : 0 \leq d_x \geq 1.0$
4) **Deceptive Threshold**, where $dT_x \in Q : 0 \leq dT_x \geq 1.0$

In all experiments, the following values were used:

1) **Basic Trust** = 0.9
2) **Risk** = 0.25
3) **Deception** = 0.1
4) **Deceptive Threshold** = 0.0

Command line file input for trial parameters (ETIP)
(See Appendix B)

1) **MAS_Size** – Maximum number of active agents allowed in the system.
2) **TimeSteps** – Maximum number of system execution (life) cycles.
3) **NumAlive** – Number of agents active in the system at the beginning of execution.
4) **NumK** – Number of nearest neighbors used by the k-Nearest Neighbor algorithm.
5) **NumDeceptive** – Maximum number of deceptive agents allowed in the system.
6) **WeightAge** – Euclidean distance weight for age attribute.
7) **WeightSuccessfulTasks** – Euclidean distance weight for successful task attribute.

المنارة للاستشارات
www.manaraa.com

8) **WeightBasicTrust** – Euclidean distance weight for basic trust attribute.
9) **WeightRisk** – Euclidean distance weight for risk attribute.
10) **TrustUpdateRate** – where $\eta_{tu} \in Q : 0 \leq \eta_{tu} \geq 1.0$
11) **LearningRate –** where $\eta_{l} \in Z : 0 \leq \eta_{l}$ where 0 is the absence of learning or exploration.
12) **Debug** – Y or N, debug mode to print agent cooperation logs.
13) **Outfile** – path/filename, execution report of failure rate by time step.
14) **FixedInput** – Y or N, to determine usage of fixed inputs.
15) **FixedInputFile** – path/name/extension of fixed inputs.
16) **FixedDeception** – Y or N, to determine if fixed input for deceptive agents will be captured from the fixed input file or bypassed. If set to N, deceptive agents will change the value for deceptive threshold.

Experiment Reports

<u>Failure Rate Log</u> (See Appendix C)

1) This log captures the number of active ("alive") agents in the system, the number of active, deceptive agents, number of failures, and the cumulative failure rate by completed time step.
2) The file is named using the Outfile parameter of the ETIP file, concatenated with "failures.", mmddyyhhmmss date format, and a ".txt" extension.

<u>Initial Values</u> (See Appendix D)

1) This log echoes contents of the ETIP file.
2) Named using the Outfile parameter of the ETIP file, concatenated with "init" and a ".txt" extension.
3) This log lists the beginning values of each agent and records:
   a. **Agent ID**
   b. **Alive** – a Boolean to indicate if the agent is active
   c. **Partner** – internal agent variable indicating if the agent currently has a partner. Initially -1 before interactions.
   d. **Deceptive** – valued as either 0 or 1. 1 indicates a deceptive agent.
   e. If deceptive, **Deception Level** and **Deception Threshold** are shown.
   f. Basic Trust

www.manaraa.com

g. **Risk**

Agent Cooperation Log (See Appendix D)

1) This log traces cooperation history of each agent during KMAS execution if debug mode is set to “Y” in the ETIP file.
2) This file is named using the Outfile parameter of the ETIP file, concatenated with “Agent”, agent id, and a “.txt” extension.
3) The information recorded in this log includes:
   a. **Time Step**
   b. **Agent Age**
   c. **Alive** – a Boolean to indicate if the agent is active
   d. **Agent ID**
   e. **Requester Agent** – a Boolean to indicate if the agent is a requester of interaction or a requested partner
   f. **Total Successes** – cumulative number of interactions with successful cooperation results
   g. **Total Failures** – cumulative number of interactions with cooperation results recorded as failures
   h. **Has Partner** – a Boolean to indicate if the agent has an interaction partner at this time step
   i. **Partner ID**
   j. **Nearest Neighbors** – listing of the agent ID’s of the nearest neighbors if performing k-Nearest Neighbor at this time step
   k. **Basic Trust**
   l. **Old General Trust in Partner** – general trust before cooperation, or general trust after performing k-Nearest Neighbor
   m. **New General Trust in Partner** – general trust in partner after cooperation and trust update
   n. **Situation Trust**
   o. **Risk**
   p. **Will Cooperate** – a Boolean to indicate if the agent requesting interaction will cooperate
   q. **Success** – a Boolean indicating if cooperation resulted in success or a failure
   r. **Num Success with Partner** – cumulative number of successes with the selected interaction partner in previous time steps, and including the current time step
   s. **Num Failure** – cumulative number of failures with the selected interaction partner in previous time steps, and including the current time step

المنارة للاستشارات

www.manaraa.com

Order of Execution

Create the fixed input file by executing class CreateFixedInputs with command line parameters of two integers (example: **java CreateFixedInputs 5 2**). The first should be the number of agents that matches the ETIP file parameters. The second integer should be the number of deceptive agents found in the ETIP file. The fixed input file can be modified if desired. The filename and path of the fixed input file is fixed within the compiled code of class CreateFixedInputs.

Execute class ThesisKmas for each trial desired by specifying the path and filename of the ETIP file on the command line (example: **java ThesisKmas c:\javatst\exampleTrial.txt**). Results can be viewed by looking at the time-stamped outfile specified in the ETIP file. If debugging is turned on, each agent will have an associated cooperation log as specified in the reports section.

Execution Flow by Java Program Object (Class)

class CreateFixedInputs (See Appendix E)

1) Receives two integer command line inputs, one equal to the maximum number of agents in the MAS, and the other equal to the maximum number of deceptive agents in the MAS.
2) Outputs two column rows up to the maximum number of agents with the first column consisting of basic trust values, and the second consisting of risk values. All values created using a random number generator.

المنارة للاستشارات
www.manaraa.com

3) Outputs two column rows up to the maximum number of deceptive agents with the first column consisting of the agent's level of deception, and the second consisting of the agent's deceptive threshold. All values are created using a random number generator.

<u>class ThesisKmas</u> (See Appendix F)

Drives KMAS experiment trial execution

1) Receives command line input that specifies path and name of the ETIP file.
2) Creates a KMAS object which is the executable experiment trial.
3) Feeds ETIP file contents into the KMAS experiment environment.
4) Populates KMAS with randomly selected agents from the pool of available agents, and activates them according to the number of initially "alive" agents specified in the ETIP file.
5) Randomly selects active agents and makes them deceptive according to the number specified in the ETIP file.
6) Executes KMAS according to the maximum number of time steps specified in the ETIP file. Outputs cooperation log if in debug mode as well as a listing of initial agent values.
7) Outputs failure rate log.

<u>class Kmas</u> (See Appendix G)

Executable experiment that defines the MAS and the necessary methods to execute one experiment life cycle.

1) After instantiation by class ThesisKmas, receives and stores ETIP file contents.
2) Creates all agents and randomly sets a maximum number of agents to be initially active in the system. Once an agent is made active, it stays active.
3) Randomly selects a set number of agents to be deceptive. Initially, deceptive agents can be active or inactive.

المنارة للاستشارات

www.manaraa.com

4) Uses contents of fixed input file to give basic trust, risk, deception, and deception threshold values to all agents if ETIP file parameter **FixedInput** is set to **Y**. If not, random values are created using a random number generator. If fixed deception is turned on, deceptive agents receive values for deception and deceptive threshold. If ETIP file parameter **FixedDeception** is **N**, random values are given and each agent agent will produce a new deceptive threshold value each time cooperation is required.
5) At the beginning of time step (life cycle), randomly adds or does not add a new agent to the system by making a non-active agent active. Resets agent cooperation variables to default (agent ID of cooperative partner, decision to cooperate, "has partner" flag, cooperation success flag, requester agent designator flag).
6) Initiates interaction between requester agents and selected partners and outputs to cooperation log if in debug mode.
7) Records and stores data needed to create the failure rate log.

<u>class KmasAgent (See Appendix H)</u>

Encapsulation of a single intelligent agent with the functionality needed to perform k-Nearest Neighbor, store and update trust values for known interaction partners, find potential partners as an agent requesting interaction, decide if cooperation with a selected partner is desired based on situation trust and risk, determine the results of cooperation as being success or failure, and practice deception if the agent is a deceptive agent.

1) If deceptive, receive values for agent level of deception and deceptive threshold through fixed input or random values. The choice is based on the FixedDeception flag in the ETIP file.
2) If a requester agent, class KMAS will direct the agent to find a potential partner through random selection. Once the partner is selected, the partner will be locked into an exclusive partnership and agent ID's will be exchanged.
3) Starts cooperation decision logic by using trustworthiness of selected interaction partners. If partner is unknown, or exploration is desired, perform k-Nearest Neighbor algorithm using Euclidean distance with weighted variables age, successful tasks, basic trust, and risk to select ***k*** neighbors.
4) Calculates situational trust to determine if cooperation is warranted.

المنارة للاستشارات
www.manaraa.com

5) If cooperation is warranted, cooperate, and store the result of cooperating. If the agent is an interaction partner, defect if $d_x > dT_x$.
6) Uses the result to update general trust and cumulative totals of successful or non-successful (failures) cooperation results as a whole, and also by the interaction partner involved in the cooperative activity.

Cooperation Decision Logic

Cooperation is coordinated through class KMAS by determining the maximum number of possible interaction partnerships consisting of active agents, and then randomly selecting which agents will initiate interaction requests. The requester agents are then directed to find and interact with potential cooperative partners. Once a requester agent has found a partner, it must then use the cooperation decision logic to first decide if the agent (partner) is trustworthy. All agents store trust values for all known agents. If an agent has a potential cooperative partner where trust is unknown, trust in that agent is initialized to 0.0. If a partner is unknown, or exploration is desired, k-Nearest Neighbor is performed using Euclidean distance with weighted variables age, successful tasks, basic trust, and risk to select the closest ***k*** neighbors. In the absence of exploration, the k-Nearest Neighbor algorithm is performed only once by a requester agent. Afterwards, the partner is known and subsequent trust updates are performed through the result of direct interaction. If the interaction partner is unknown by all neighbors, trust in the current partner is set to .50, representing a 50% chance that the unknown agent is trustworthy. If the partner is known by at least one neighbor, general trust becomes the average value among all contributing neighbors. If the current time

المنارة للاستشارات
www.manaraa.com

step is a period of exploration, the general trust recommendation from the k-Nearest Neighbor algorithm is used regardless of whether or not the agent is known. The recommended general trust value is used if it is less than the general trust value already stored for a known partner. If the partner is unknown, it is equivalent to using k-Nearest Neighbor without exploration for unknown partners.

After k-Nearest Neighbor is performed or not performed, situational trust is calculated. Situational trust represents the trustworthiness of the interaction partner in the eyes of the requester. Cooperation has not yet taken place, and the requester agent must decide whether or not the risk warrants participating in the cooperative task. Situation trust is calculated by the equation $T_x(y,\alpha_x) = T_x(T_x(y))$ where α_x is the current interaction between requester agent x and interaction partner y. If situational trust is $\geq R_x$, x's general disposition to risk, agent x will decide to cooperate, and will record the results of cooperation. If agent y is non-deceptive, successful cooperation will be recorded as true. If agent y is deceptive, and its level of deception is greater than its deceptive threshold, successful cooperation is false and a failure will be recorded. If the experiment is not in fixed deception mode, each time a deceptive agent is involved in a cooperative task, the deceptive threshold is given a random value such that a deceptive agent will not practice deception in every partnership. In this research, all experiment trials use fixed deception mode. After cooperation has occurred, trust is updated by the equation $T_x(y)' = T_x(y)'' + \Delta_t(1 - \Delta_t)\eta_{tu}$ where $T_x(y)''$ is general trust prior to interaction.

المنارة للاستشارات

www.manaraa.com

Δ_t = (Successes$_x$(y) / Successes$_x$(y) + Failures$_x$(y)) - $T_x(y)''$. If there have been no recorded successes or failures, $\Delta_t = T_x(y)''$. As the number of failures increases, Δ_t will become negative, ultimately causing general trust to be reduced. $T_x(y)'$ is then fixed at 1.0 if $T_x(y)' > 1.0$ and .001 if the $T_x(y)'$ is < 0.0.

المنارة للاستشارات

www.manaraa.com

5.2.2 EXPERIMENT 1 HYPOTHESES, RESULTS, and CONCLUSIONS

Experiment 1 Hypotheses:

- **1.1** - A system performing k-Nearest Neighbor (KMAS) will outperform a system that does not perform k-Nearest Neighbor (NumK = 0), where performance is measured by the system's ability to only allow cooperation between a requester agent and a partner that is non-deceptive.

- **1.2** - Over time, a system using trust-based agent recommendation will converge towards a state where cooperation with deceptive agents will not occur.

Experiment 1 Description:

Experiment 1 compares K[0], K[6], and KE[6,10] to compare trials using k-Nearest Neighbor, and one trial that does not. Relative convergence is set at 1000 life cycles. The benchmark failure rate is set to 2.166. Experiment 1 is executed three times to produce experiment groups A, B, and C.

Trial1 = K[0], Trial2 = K[6], Trial3 = KE[6,10]

To assist the reader, if needed, the recorded failures for each time step used to calculate the failure rate are represented in graphical format in Appendix I.

المنارة للاستشارات
www.manaraa.com

Table 1: Experiment 1 Inputs

Exp:1	**MAS Size**	**Time Steps**	**Num Alive**	**Num K**	**Num Deceptive**	W_A	W_S	W_{BT}	W_R	η_{tu}	η_l
Trial: 1	50	3000	25	0	25	1.0	.05	1.0	1.0	.10	0
Trial: 2	50	3000	25	6	25	1.0	.05	1.0	1.0	.10	0
Trial: 3	50	3000	25	6	25	1.0	.05	1.0	1.0	.10	10

Table 2: Experiment 1 ETIP Contents

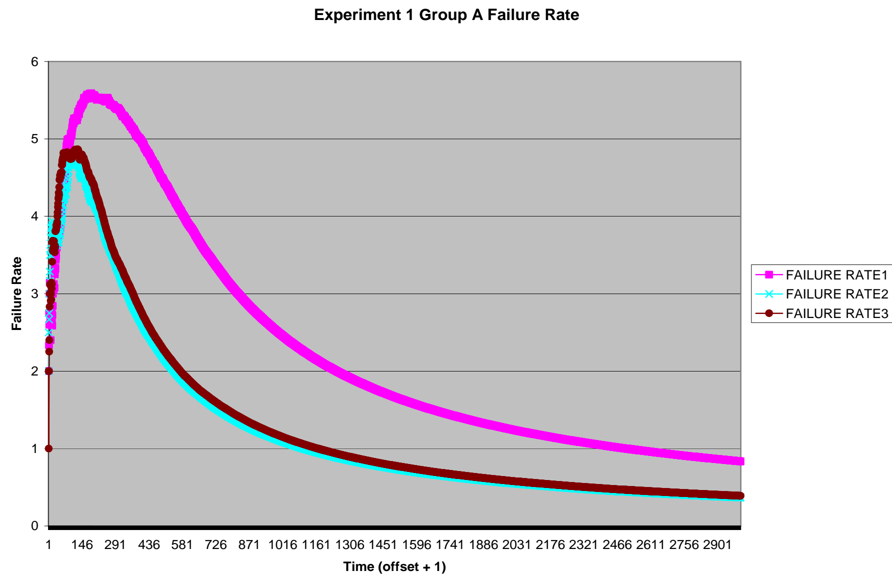
Trial: 1	Trial: 2	Trial: 3
MAS_Size: 50	MAS_Size: 50	MAS_Size: 50
TimeSteps: 3000	TimeSteps: 3000	TimeSteps: 3000
NumAlive: 25	NumAlive: 25	NumAlive: 25
NumK: 0	NumK: 6	NumK: 6
NumDeceptive: 25	NumDeceptive: 25	NumDeceptive: 25
WeightAge: 1.0	WeightAge: 1.0	WeightAge: 1.0
WeightSuccessfulTasks: 0.5	WeightSuccessfulTasks: 0.5	WeightSuccessfulTasks: 0.5
WeightBasicTrust: 1.0	WeightBasicTrust: 1.0	WeightBasicTrust: 1.0
WeightRisk: 1.0	WeightRisk: 1.0	WeightRisk: 1.0
TrustUpdateRate: .10	TrustUpdateRate: .10	TrustUpdateRate: .10
LearningRate 0	LearningRate 0	LearningRate 10

www.manaraa.com

Table 3: Experiment 1 Group A Observations

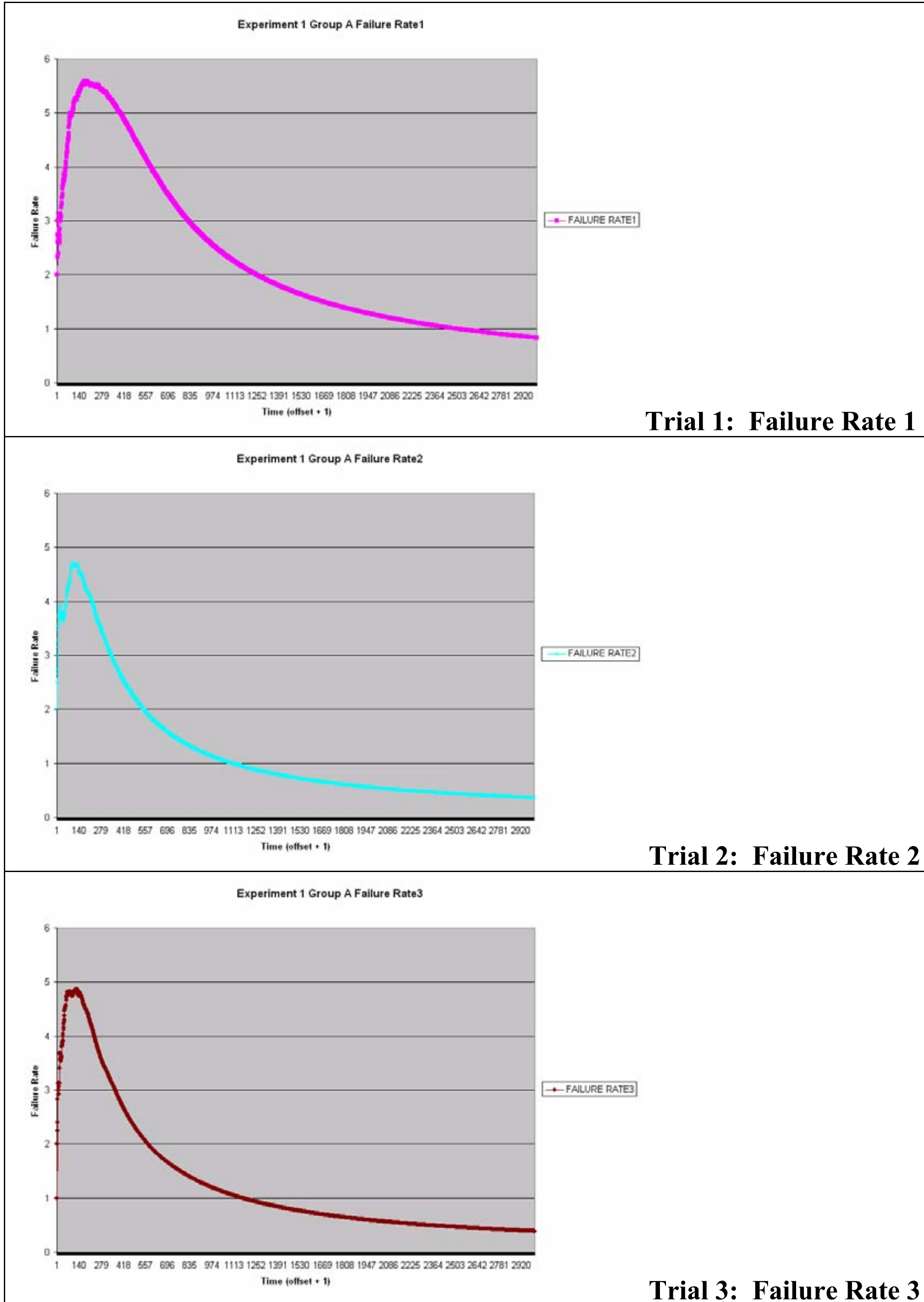
	Trial1	Trial2	Trial3
Max Failure Rate	5.591398	4.7083335	4.866142
Time Step at Max	185	95	126
$Slope_{LR}$	-0.001537675	-0.00100691	-0.0010049
Elapsed Time (End – Max)	2814	2904	2873
$Input\ Time_B$	1152	501	526
$Failure_B$	2.165655	2.1653388	2.166983
$Avg_Velocity_B$	-0.00354265	-0.00626353	-0.0067479
$Elapsed\ Time_B$ (Input – Max)	967	406	400
$Input\ Time_R$	1000	1000	1000
$Failure_R$	2.4885116	1.1068931	1.1658342
$Avg_Velocity_R$	-0.003807223	-0.00397949	-0.0042338
$Elapsed\ Time_R$ (Input – Max)	815	905	874

Figure 1: Experiment 1 Group A Failure Rate



المنارة للاستشارات
www.manaraa.com

Figure 2: Experiment 1 Group A Individual Failure Rate

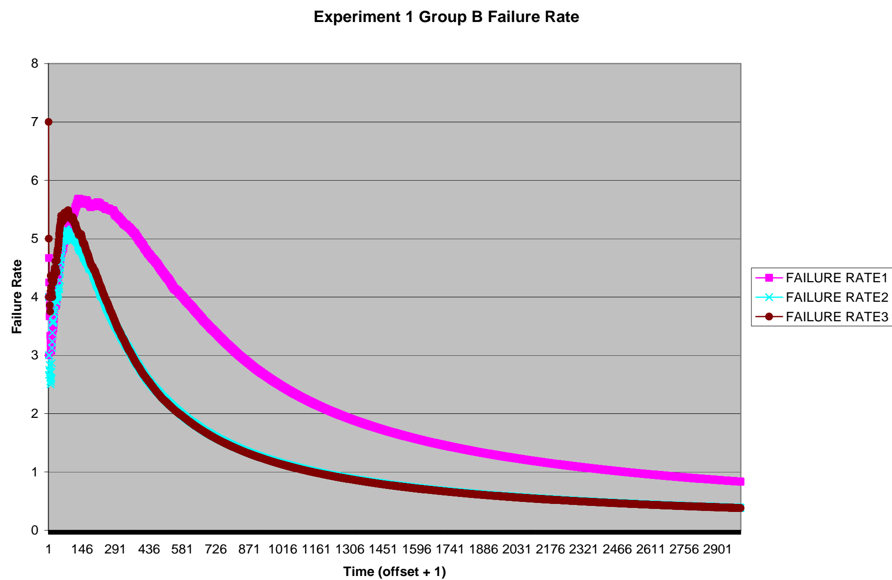


www.manaraa.com

Table 4: Experiment 1 Group B Observations

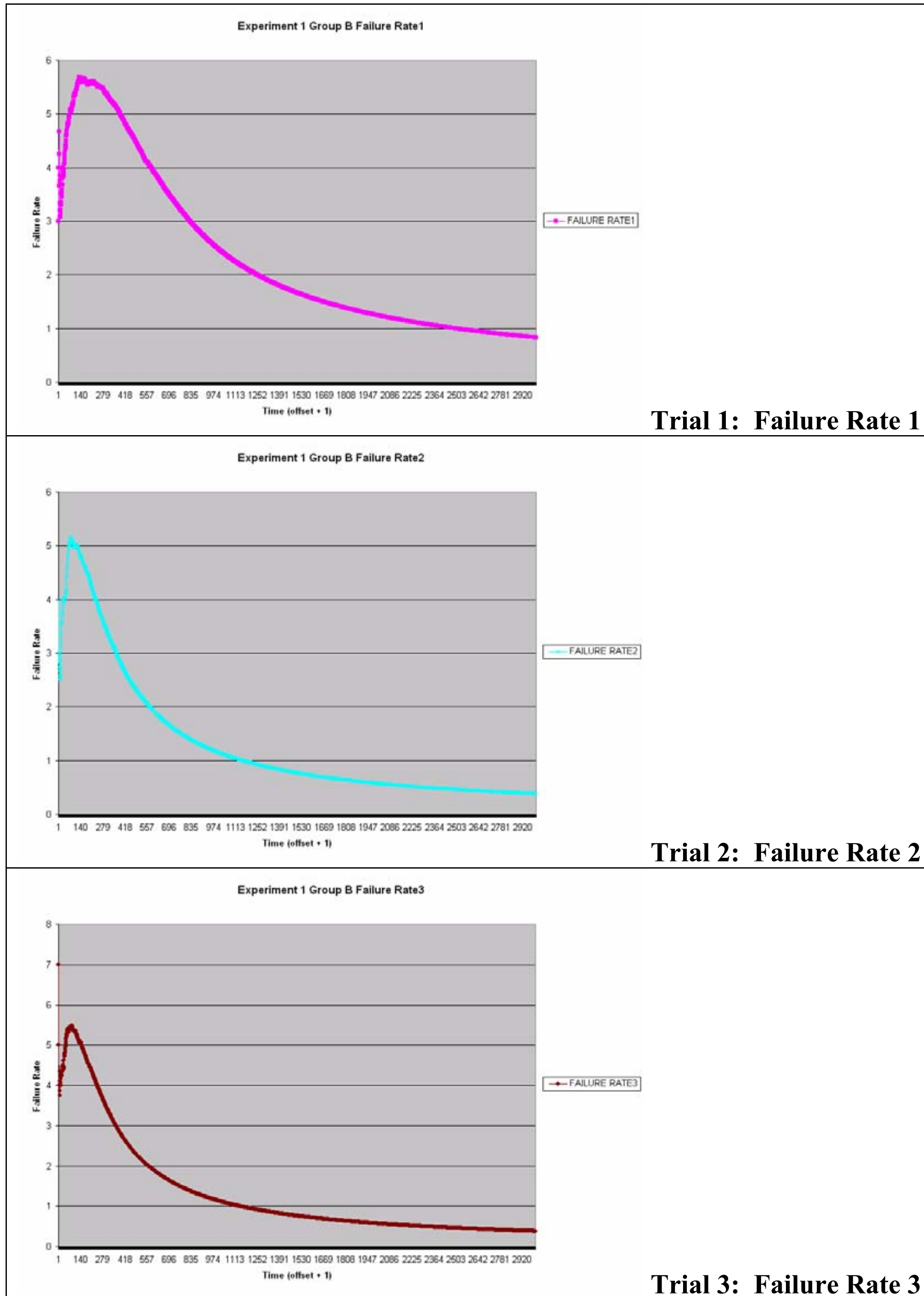
	Trial1	Trial2	Trial3
Max Failure Rate	5.6842103	5.1621623	5.4883723
Time Step at Max	132	73	85
$Slope_{LR}$	-0.00158314	-0.00108929	-0.0012008
Elapsed Time (End – Max)	2867	2926	2914
Input $Time_B$	1153	526	518
$Failure_B$	2.1663778	2.1650853	2.1657033
$Avg_Velocity_B$	-0.003445477	-0.00661606	-0.0076736
Elapsed $Time_B$ (Input – Max)	1021	453	433
Input $Time_R$	1000	1000	1000
$Failure_R$	2.4955044	1.1578422	1.1418581
$Avg_Velocity_R$	-0.003673624	-0.00431965	-0.0047503
Elapsed $Time_R$ (Input – Max)	868	927	915

Figure 3: Experiment 1 Group B Failure Rate



www.manaraa.com

Figure 4: Experiment 1 Group B Individual Failure Rate



Trial 1: Failure Rate 1

Trial 2: Failure Rate 2

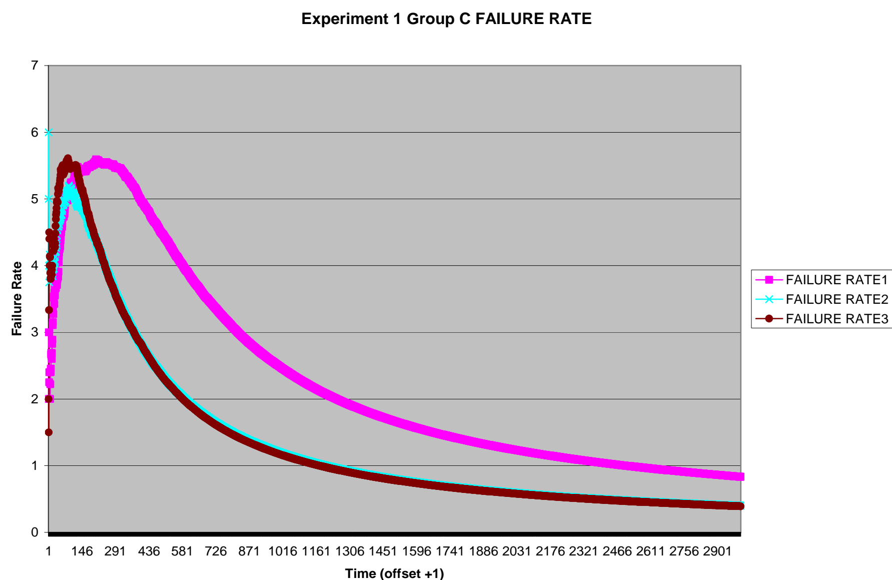
Trial 3: Failure Rate 3

www.manaraa.com

Table 5: Experiment 1 Group C Observations

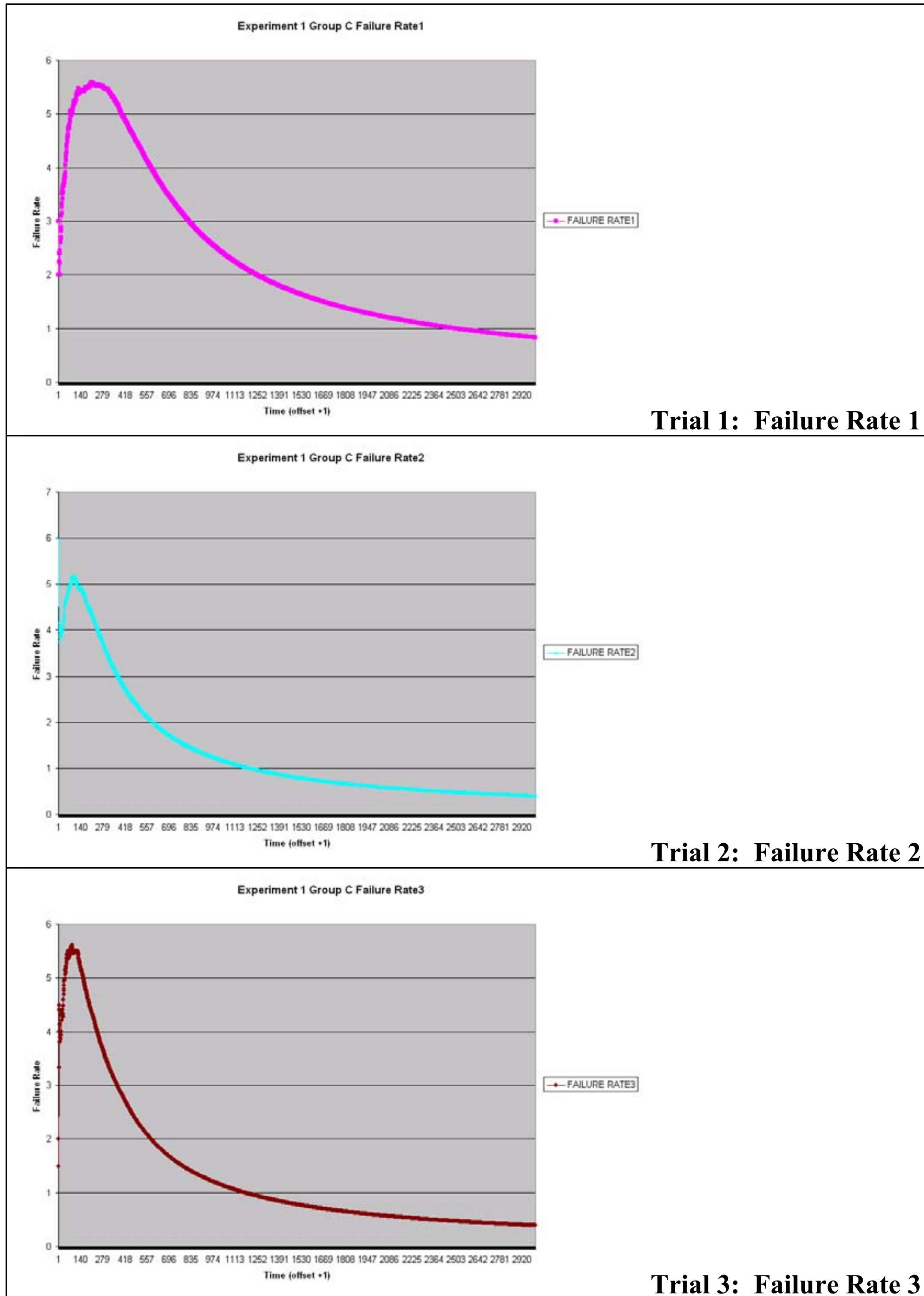
	Trial1	Trial2	Trial3
Max Failure Rate	5.594203	5.202247	5.611765
Time Step at Max	206	88	84
$Slope_{LR}$	-0.001522762	-0.00118322	-0.0011142
Elapsed Time (End – Max)	2793	2911	2915
$Input\ Time_B$	1151	541	536
$Failure_B$	2.1666667	2.1660516	2.1657355
$Avg_Velocity_B$	-0.003627023	-0.00670242	-0.007624
$Elapsed\ Time_B$ (Input – Max)	945	453	452
$Input\ Time_R$	1000	1000	1000
$Failure_R$	2.4905095	1.2007992	1.1728271
$Avg_Velocity_R$	-0.003908934	-0.00438755	-0.004846
$Elapsed\ Time_R$ (Input – Max)	794	912	916

Figure 5: Experiment 1 Group C Failure Rate



المنارة للاستشارات
www.manaraa.com

Figure 6: Experiment 1 Group C Individual Failure Rate



www.manaraa.com

Experiment 1 Conclusions:

Across all three trials, there is a stark contrast between K[0] (Trial 1) and the two variances of k-Nearest Neighbor (Trials 2 and 3). The first differences are in the charted curves representing failure rate over time as measured by life cycles (Figures 1 through 6). With respect to time, the max failure rate for all three trials peaks early. Since all agents are initially unknown, and half of the agents have yet to become active within the system, time is needed to not only introduce new agents (of which some could be deceitful), but also to allow agents to develop an appropriate amount of distrust through updated trust values after direct interaction in K[0], and propagated recommendation-based reputation in K[6] (Trial 2) and KE[6,10] (Trial 3). Max failure rates occur in earlier observed life cycles in the trials that use the k-Nearest Neighbor approach. This was clearly observed across all three trials and groups with an average distance of 89 life cycles between the max failure rates of K[0] and K[6], and 76 life cycles between K[0] and KE[6,10]. The graphs representing individual failure rates (Figures 2, 4, and 6) clearly show the earlier observed occurrences of max failure rate by looking at the X-axis which represents the time step. If hypothesis 1.1 holds true, we expect that the maximum failure rate will be achieved sooner for systems that use the k-Nearest Neighbor algorithm. As the failure rate decreases, this indicates the presence of fewer interactions that lead to cooperation between a requester agent and a deceptive partner. These experiments support hypothesis 1.1.

المنارة للاستشارات
www.manaraa.com

In addition to curve peaks in the charted graphs, the curves are also differentiated in terms of the slope that can be measured. The first slope measured is the slope of the linear regression line through the curve representing each trial execution (labeled as $Slope_{LR}$). The time period is between the first local or max failure rate (which ever occurs first) and the last executed life cycle. For example, in Experiment 1 Group B Trail3/Failure Rate3, the graph in Figure 4 depicts a local max of 7. In the column labeled Trail 3 in Table 4, the maximum failure rate has a value of 5.488. The reason for this difference is the desire to measure $Slope_{LR}$ starting from the highest recorded failure rate to properly measure change. The difference between local and max failure rate is described in the terms and definitions of Section 5.2. The slope of the line represents the average velocity measured over time.

It is observed that velocity is higher for K[0] compared to the other trials. This holds true in all three groups of Experiment 1. Part of the explanation is due to the max failure rate which is generally highest for K[0] across all group trials. This is an expected result as direct interaction is random with deceitful agents. Cooperation will continue to occur until general trust is sufficiently lowered through trust update. There is no ability to receive recommendations from other agents who have already identified a deceitful agent as untrustworthy. At the end of the last executed life cycle, failure rate receives the greatest displacement for K[0]. It is expected that for longer periods of execution, failure rate will eventually converge to zero, and a trial with the highest peak failure rate will always have the largest value for displacement or slope. Therefore, it is

www.manaraa.com

more important to view the average velocity in increasingly smaller life cycle periods because this approaches instantaneous velocity and a more accurate measure of how well the trials are performing. It is also important to note that linear regression attempts to plot a straight line. While the graphs represented in this research are not linear, we simply use linear regression to measure change.

There are two other ways to observe the slope of the charted curves for the experiment trials using smaller changes in time. First, slope can be measured from max failure rate to a given failure rate across all three trials ($Avg_Velocity_B$), or slope can be measured from max failure rate to a given life cycle which is representative of relative convergence ($Avg_Velocity_R$). Smaller intervals of time will allow the slope to become more valuable in terms of measuring performance. Instead of linear regression, the slope will be calculated using a simple rise over run method ($y_2 - y_1 / x_2 - x_1$). Across all groups, the slope measured from max failure rate to a relative convergence point of 1000 life cycles executed ($Avg_Velocity_R$), provides values that collaborate expected results if hypothesis 1.1 holds true. k-Nearest Neighbor approaches have the largest slope values with KE[6,10] dominating all three trials for both types of slope measurements. In the case of group A (Table 3 and Figures 1 & 2), even though KE[6,10] does not have a lower failure rate at relative convergence ($Failure_R$) than K[6], it does have a higher slope indicating a more rapid drop towards convergence. The max failure rate and associated time cycle are obviously factors in the slope

المنارة للاستشارات
www.manaraa.com

calculation, but they are not the major determinants in the observed results. The difference in elapsed time is only 31 life cycles.Using the elapsed time value of 905 life cycles for the Trial 3 relative convergence slope calculation still yields a larger slope than Trial 2 with 905 life cycles.

The second slope observed, $Avg_Velocity_B$, is the slope between max failure rate and a chosen benchmark failure rate close to 2.166 for all experiment groups ($Failure_B$). K[6] and in particular KE[6,10] still show dominance. Using group A as a reference again, the difference in elapsed life cycles is now only 6 life cycles between K[6] and KE[6,10] in terms of the number of life cycles needed to reach a failure rate close to 2.166 failures per life cycle. One observation is that the slope is higher for k-Nearest Neighbor trials when using the benchmark as opposed to relative convergence. K[0] records a higher slope when using relative convergence. The reason is that it takes longer for K[0] to have an impact on failure rate because of the number of necessary direct interactions. By the time the shared relative convergence point has been reached, the k-Nearest Neighbor trials have curves that are already starting to "smooth out". The instantaneous velocity of the curves is decreasing as the failure rates converge towards zero.

We have taken a brief glance at elapsed time as it pertains to slope calculations and the amount of life cycles needed to reach the max failure rate. Elapsed time can also be looked at when viewing how much time is required to reach an arbitrary failure

المنارة للاستشارات
www.manaraa.com

rate during each trial starting from the first time step. Using the 2.166 benchmark failure rate again, we observe that on average, K[0] needs 1152 life cycles, K[6] requires 522 life cycles, and KE[6,10] utilizes 526 life cycles to reach this point. Again, k-Nearest Neighbor trials clearly dominate and support hypothesis 1.1 as true. Despite a larger average for KE[6,10] when compared to K[6], we showed that the slope proves a faster convergence. This difference can be explained by the random nature of agent interaction in early life cycles. In groups A and B, K[6,10] reached a max failure rate at later life cycles than K[6], thus requiring more time to converge towards 2.166. This highlights the fact that recommendation-based reputation is still dependent upon direct interaction until reputation of deceitful agents has been determined by a sufficient number of nearest neighbors, and the reputation propagated by those neighbors is sufficient enough to allow the requester agent to identify the deceitful agents as untrustworthy.

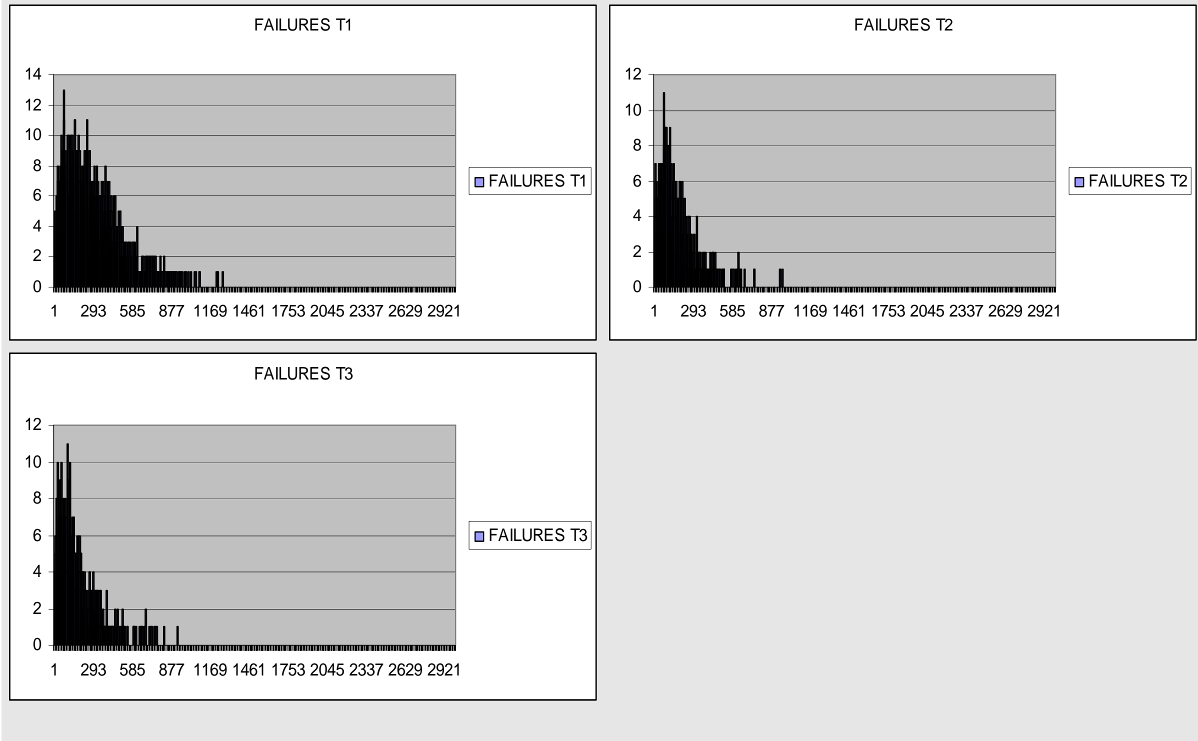
In summary, hypothesis 1.1 is supported by observed experiment results. Trials using k-Nearest Neighbor outperform trials that do not use k-Nearest Neighbor in terms of how fast the system reaches desirable states of execution where cooperation resulting in failures is decreasing as the system converges towards a state of zero occurrences of failures. This is justified by observing that for trials utilizing some form of the k-Nearest Neighbor algorithm, max failure rates occur in earlier life cycles, slope measurements between max failure rates and chosen time periods show larger rates of convergence as slope increases, and the benchmark failure rate of 2.166 is reached

المنارة للاستشارات
www.manaraa.com

sooner. Hypothesis 1.2 is supported by observing Appendix I and the graphs representing number of failures over time. These graphs represent the outputted failure logs for Experiment 1 groups and associated trials. Three of the graphs are shown on the next page, and represent the number of failures recorded at each time step for Experiment 1, Group A. Looking at the recorded failures past the relative convergence point of 1000 life cycles, the number of recorded failures eventually reaches the desired value of zero occurrences during any executed life cycle. This holds true for both trust-based recommendation, and pure direct interaction. In fact, for both trials using recommendation-based reputation (represented by graphs T2/trial 2 and T3/trial 3), no failures are recorded past the convergence point of 1000 life cycles. Direct interaction, shown in graph Failures T1, records only a few occurrences.

المنارة للاستشارات
www.manaraa.com

Figure 7: Experiment 1 Group A Failures by Time Step



www.manaraa.com

5.2.3 EXPERIMENT 2 HYPOTHESES, RESULTS, and CONCLUSIONS

Experiment 2 Hypotheses:

- **2.1** - The number of executed life cycles needed to reach maximum failure rate will decrease as the number of nearest neighbors increases, despite randomness in both interaction relationship pairings and when new agents are made active in the system

- **2.2** - Curve slope, as a measure of average velocity and calculated by ($y_2 – y_1$ / $x_2 – x_1$) where y's represent the range of failure rates and x's represent the range of time steps, will increase as the number of neighbors increases.

- **2.3** - As the number of neighbors increases, elapsed time in life cycles between the maximum failure rate and the benchmark failure rate (Elapsed $Time_B$) will decrease.

Experiment 2 Description:

Experiment 2 compares K[4], K[8], and K[12] to compare trials using three different values for the number of nearest neighbors. Relative convergence is set at 800 life cycles. The benchmark failure rate is set to 2.166. Hypotheses 2.1 and 2.3 use the term maximum failure rate as defined in Section 5.2.1 as the maximum observed failure rate among trial time steps after initial, local max failure rates have been achieved. Experiment 2 is executed three times to produce experiment groups A, B, and C.

Trial1 = K[4], Trial2 = K[8], Trial3 = K[12]

To assist the reader, if needed, the recorded failures for each time step used to calculate the failure rate are represented in graphical format in Appendix J.

المنارة للاستشارات
www.manaraa.com

Table 6: Experiment 2 Inputs

Exp:2	MAS Size	Time Steps	Num Alive	Num K	Num Deceptive	W_A	W_S	W_{BT}	W_R	η_{tu}	η_l
Trial: 1	50	3000	25	4	25	1.0	.05	1.0	1.0	.10	0
Trial: 2	50	3000	25	8	25	1.0	.05	1.0	1.0	.10	0
Trial: 3	50	3000	25	12	25	1.0	.05	1.0	1.0	.10	0

Table 7: Experiment 2 ETIP Contents

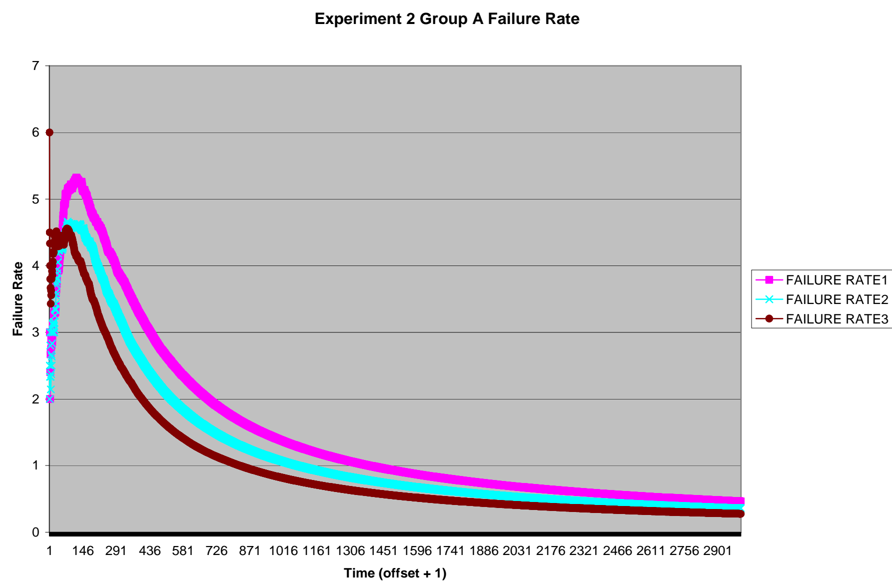
Trial: 1	Trial: 2	Trial: 3
MAS_Size: 50	MAS_Size: 50	MAS_Size: 50
TimeSteps: 3000	TimeSteps: 3000	TimeSteps: 3000
NumAlive: 25	NumAlive: 25	NumAlive: 25
NumK: 4	NumK: 8	NumK: 12
NumDeceptive: 25	NumDeceptive: 25	NumDeceptive: 25
WeightAge: 1.0	WeightAge: 1.0	WeightAge: 1.0
WeightSuccessfulTasks: 0.5	WeightSuccessfulTasks: 0.5	WeightSuccessfulTasks: 0.5
WeightBasicTrust: 1.0	WeightBasicTrust: 1.0	WeightBasicTrust: 1.0
WeightRisk: 1.0	WeightRisk: 1.0	WeightRisk: 1.0
TrustUpdateRate: .10	TrustUpdateRate: .10	TrustUpdateRate: .10
LearningRate 0	LearningRate 0	LearningRate 0

المنارة للاستشارات
www.manaraa.com

Table 8: Experiment 2 Group A Observations

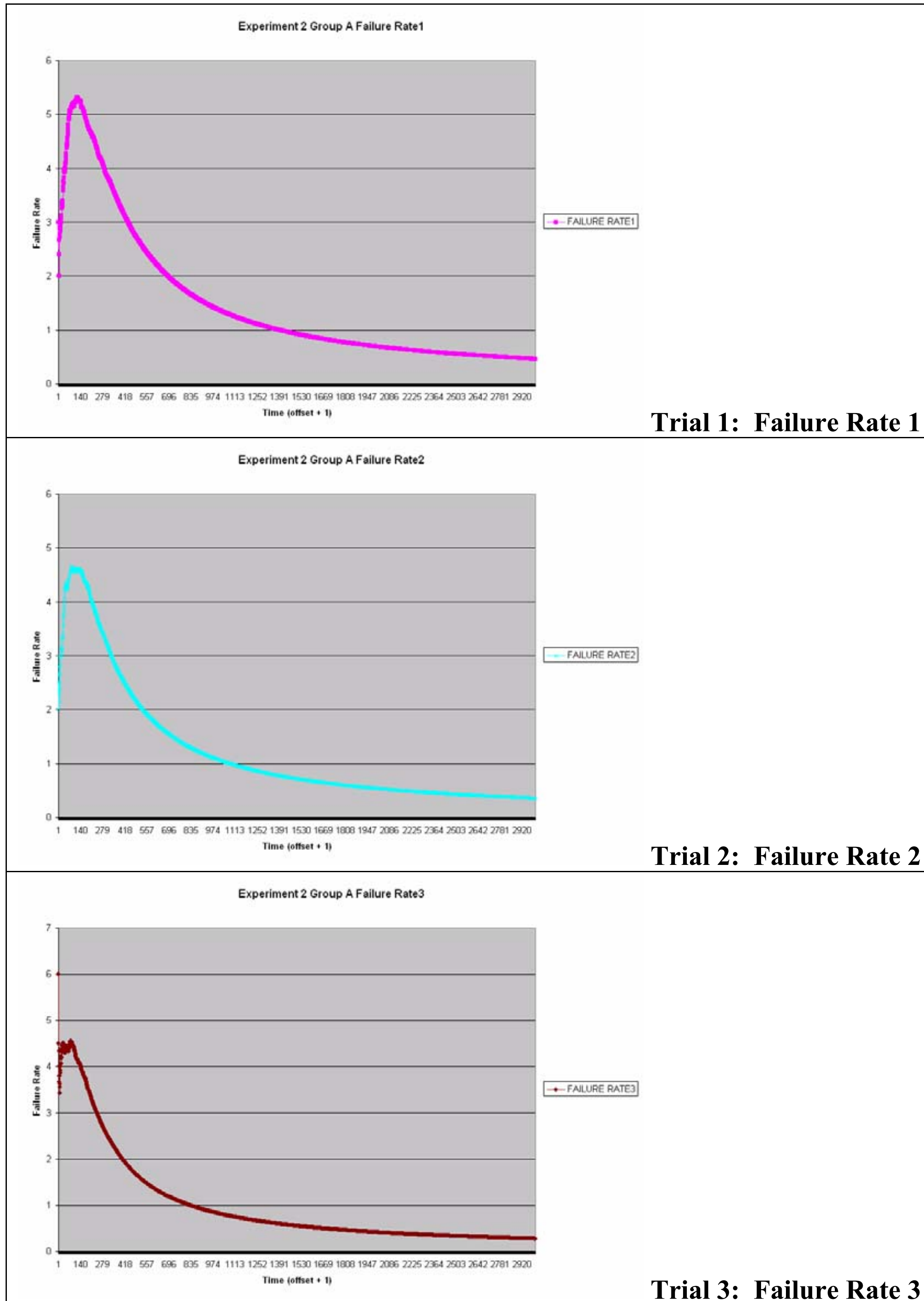
	Trial1	Trial2	Trial3
Max Failure Rate	5.322034	4.6625	4.5584416
Time Step at Max	117	79	76
$Slope_{LR}$	-0.001153685	-0.00101265	-0.0009458
Elapsed Time (End – Max)	2882	2920	2923
Input $Time_B$	633	486	368
$Failure_B$	2.1671925	2.1663244	2.1653116
$Avg_Velocity_B$	-0.006114034	-0.00613311	-0.0081957
Elapsed $Time_B$ (Input – Max)	516	407	292
Input $Time_R$	800	800	800
$Failure_R$	1.7265917	1.3420724	1.0299625
$Avg_Velocity_R$	-0.005264191	-0.00460531	-0.0048736
Elapsed $Time_R$ (Input – Max)	683	721	724

Figure 8: Experiment 2 Group A Failure Rate



المنارة للاستشارات
www.manaraa.com

Figure 9: Experiment 2 Group A Individual Failure Rate

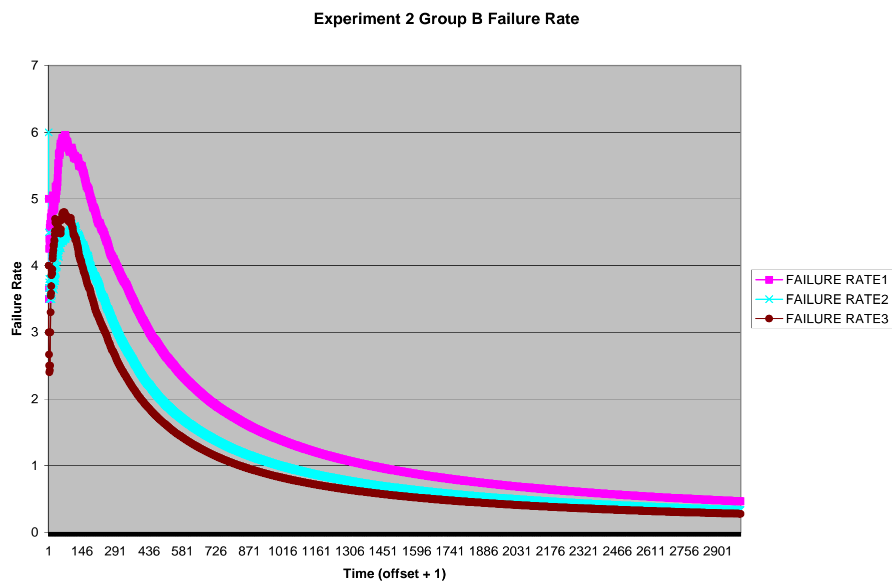


المنارة للاستشارات
www.manaraa.com

Table 9: Experiment 2 Group B Observations

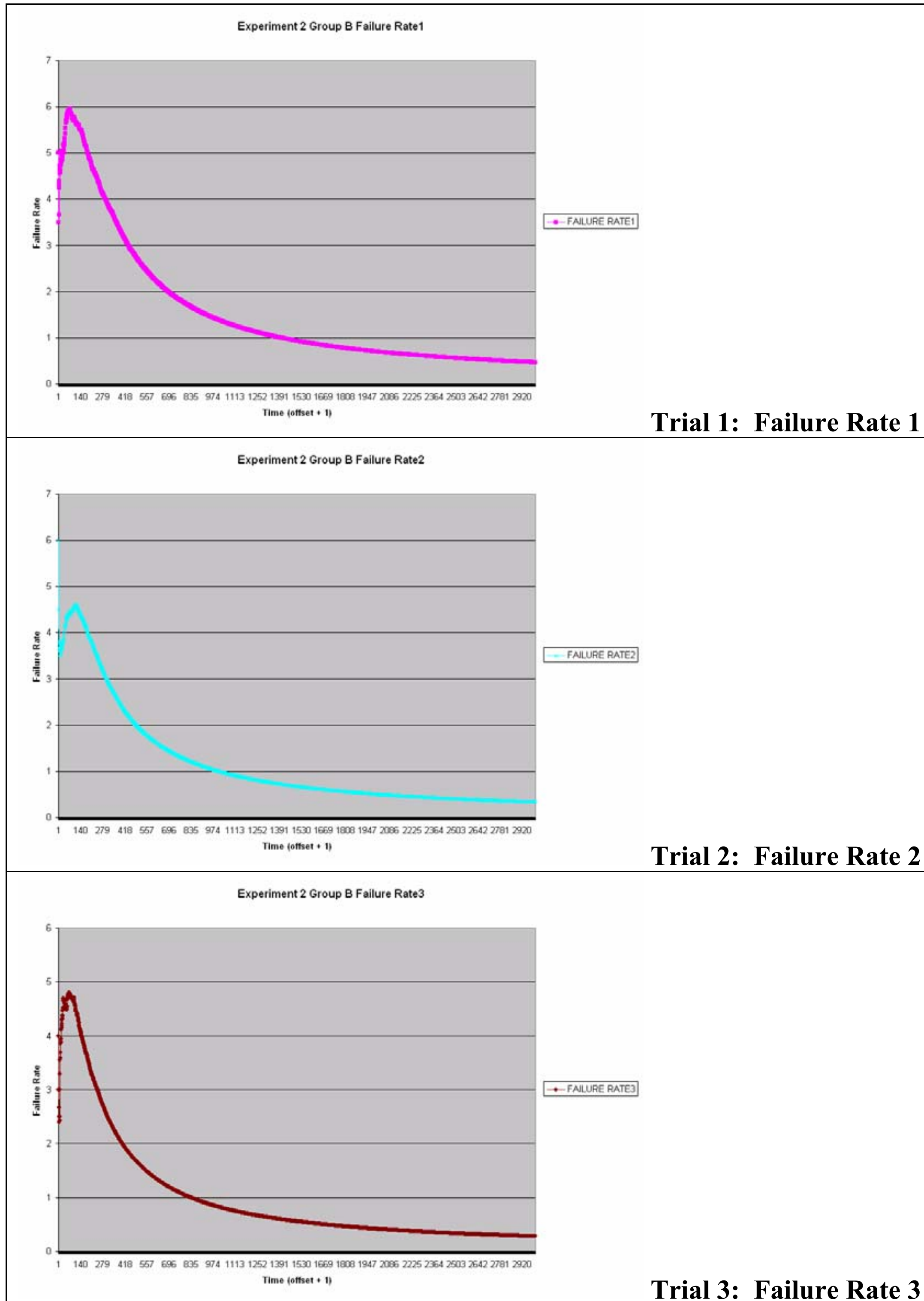
	Trial1	Trial2	Trial3
Max Failure Rate	5.9594593	4.6213593	4.8030305
Time Step at Max	73	102	65
$Slope_{LR}$	-0.001256279	-0.00103427	-0.0008718
Elapsed Time (End – Max)	2926	2897	2934
Input $Time_B$	635	446	366
$Failure_B$	2.1650944	2.165548	2.1662126
$Avg_Velocity_B$	-0.006751539	-0.00713899	-0.0087602
Elapsed $Time_B$ (Input – Max)	562	344	301
Input $Time_R$	800	800	800
$Failure_R$	1.7365793	1.2509364	1.0362047
$Avg_Velocity_R$	-0.005808638	-0.00482869	-0.0051249
Elapsed $Time_R$ (Input – Max)	727	698	735

Figure 10: Experiment 2 Group B Failure Rate



www.manaraa.com

Figure 11: Experiment 2 Group B Individual Failure Rate

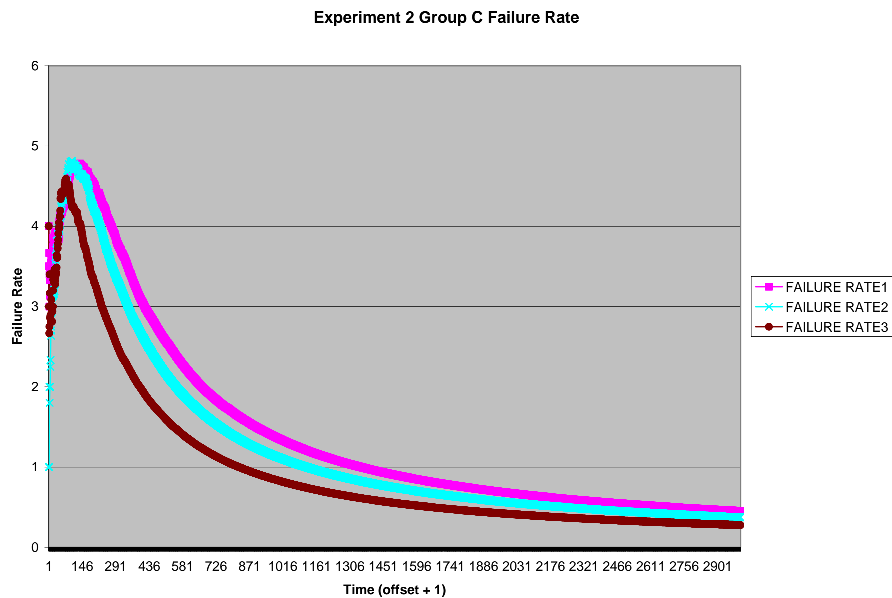


المنارة للاستشارات
www.manaraa.com

Table 10: Experiment 2 Group C Observations

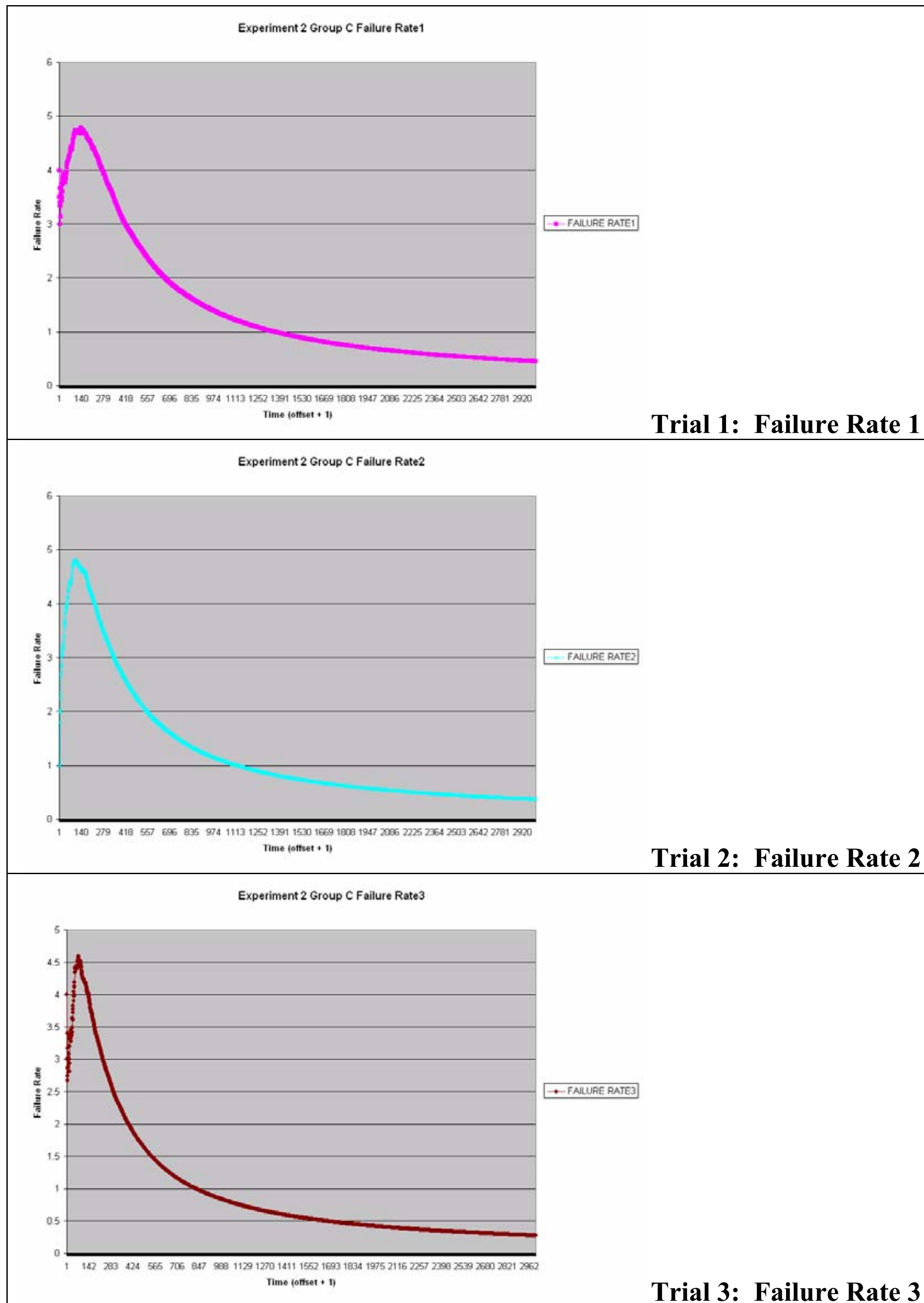
	Trial1	Trial2	Trial3
Max Failure Rate	4.781022	4.815534	4.5921054
Time Step at Max	136	102	75
$Slope_{LR}$	-0.001077035	-0.00100676	-0.0008279
Elapsed Time (End – Max)	2863	2897	2924
$Input Time_B$	611	507	359
$Failure_B$	2.1650326	2.1653543	2.1666667
$Avg_Velocity_B$	-0.005507346	-0.00654365	-0.0085403
$Elapsed Time_B$ (Input – Max)	475	405	284
$Input Time_R$	800	800	800
$Failure_R$	1.6803995	1.3932585	1.0274657
$Avg_Velocity_R$	-0.004669612	-0.00490297	-0.0049167
$Elapsed Time_R$ (Input – Max)	664	698	725

Figure 12: Experiment 2 Group C Failure Rate



المنارة للاستشارات
www.manaraa.com

Figure 13: Experiment 2 Group C Individual Failure Rate



المنارة للاستشارات
www.manaraa.com

Experiment 2 Description:

In all three groups with the exception of group B Trial 2, hypothesis 2.1 (asserting that the number of life cycles needed to reach maximum failure rate decreases as the number of nearest neighbors increases)is supported. In group B (Table 9 and Figures 10-11) , K[4] reaches max failure rate in 73 life cycles, K[8] reaches it in 102 life cycles, and K[12] reaches max failure rate at 65 life cycles. The reason for this anomaly is that K[8] (Trial 2) reaches a local max at time t_0 as indicated by Figure 11. Also, at this time, the system is engaged in total direct interaction because all agents are initially unknown to each other.

Across all three groups, hypothesis 2.2 (asserting that the slope will increase as the number of neighbors increases) is supported if slope is taken as the average velocity of the curve between max failure rate and the benchmark failure rate of 2.166 (labeled $Avg_Velocity_B$ in Tables 12, 13, and 14). In all three groups, there is a drastic difference in elapsed time between K[4] and K[12] as it relates to the benchmark. Here, elapsed time is measured from the time max failure rate is reached, to the time step that the benchmark failure rate is reached ($Elapsed\ Time_B$ (Input – Max)). The average difference is in the order of 255 life cycles for Experiment 2 as a whole indicating a very large comparative average velocity for K[12] (Trial 3 for all groups). When measuring slope $Avg_Velocity_R$ for relative convergence, the hypothesis 2.2 only holds true for group C. What this means is that K[4] has the highest rate of change between max failure rate and relative convergence for two out of three trials.

المنارة للاستشارات
www.manaraa.com

It is also important to note that K[8] and K[12] have very similar slope values that differ only by -1.93 x 10^4 as an Experiment 2 average. This indicates the presence of diminishing returns as the number of nearest neighbors increases. The greatest amount of change takes place early on as indicated by our benchmark failure rate of 2.166 and the slope between it and the max failure rate. Past some arbitrary point, instantaneous velocity, or slope, starts decreasing as the curve smoothes towards convergence.

Among all three groups, hypothesis 2.3 (asserting that as the number of neighbors increases, elapsed time in life cycles between the maximum failure rate and the benchmark failure rate will decrease) is supported with K[12] reaching the benchmark failure rate in the smallest elapsed time when measured from maximum failure rate. On average, K[4] achieves the benchmark in 517 life cycles, K[8] in 385 life cycles, and K[12] in 292 life cycles. It was also discovered that as the number of nearest neighbors increases, the elapsed time between max failure rate and relative convergence increases. Since relative convergence is fixed, the earliest max failure rate will yield the greatest elapsed time. This system characteristic is straight forwardly derived from the argument supporting hypothesis 2.1 so that as the number of nearest neighbors increases, max failure rate is achieved sooner causing the greater observed elapsed time.

المنارة للاستشارات

www.manaraa.com

In summary, hypotheses 2.1 and 2.3 are supported by experiment results suggesting that as the number of nearest neighbors increases, maximum failure rate is achieved sooner, and elapsed time between max failure rate and the benchmark decreases. Hypothesis 2.3 is further supported by the argument supporting hypothesis 2.2, and showing that slope increases between max failure rate and the benchmark failure rate as the number of neighbors increases. It was also found that this time period exhibits a rate of higher returns when the number of nearest neighbors is increased, and that past some arbitrary point in time, gains in terms of system performance will diminish. It might be possible that a KMAS system of certain size (maximum number of agents), can be defined with a minimal number of allowable nearest neighbors to achieve maximal performance.

المنارة للاستشارات
www.manaraa.com

5.2.4 EXPERIMENT 3 HYPOTHESES, RESULTS, and CONCLUSIONS

Experiment 3 Hypotheses:

- **3.1** - The number of executed life cycles needed to reach maximum failure rate will decrease as the learning rate decreases, allowing more exploration prior to relative convergence.

- **3.2** - Curve slope, as a measure of average velocity and calculated by ($y_2 - y_1$ / $x_2 - x_1$) where y's represent the range of failure rates and x's represent the range of time steps, will increase as learning rate decreases (exploration increases), indicating a greater return. This result is expected for both time periods between max failure rate and relative convergence (Elapsed $Time_R$), as well as the period between max failure rate and the benchmark failure rate (Elapsed $Time_B$).

- **3.3** - Elapsed time between the maximum failure rate and the benchmark failure rate (Elapsed $Time_B$) will decrease as the learning rate decreases.

Experiment 3 Description:

Experiment 3 compares KE[4,10], KE[4,5], and KE[4,1] to compare trials using different strategies of exploration. The number of nearest neighbors is fixed at four for each trial. Relative convergence is varied by group and trial as indicated by Input $Time_R$ in the table of experiment results shown in section 5.2.3. The benchmark failure rate is set to 2.166. Experiment 3 is executed three times to produce experiment groups A, B, and C.

Trial1 = KE[4,10], Trial2 = KE[4,5], Trial3 = KE[4,1]

To assist the reader, if needed, the recorded failures for each time step used to calculate the failure rate are represented in graphical format in Appendix K.

المنارة للاستشارات

www.manaraa.com

Table 11: Experiment 3 Inputs

Exp:3	**MAS Size**	**Time Steps**	**Num Alive**	**Num K**	**Num Deceptive**	W_A	W_S	W_{BT}	W_R	η_{tu}	η_l
Trial: 1	50	3000	25	4	25	1.0	.05	1.0	1.0	.10	10
Trial: 2	50	3000	25	4	25	1.0	.05	1.0	1.0	.10	5
Trial: 3	50	3000	25	4	25	1.0	.05	1.0	1.0	.10	1

Table 12: Experiment 3 ETIP Contents

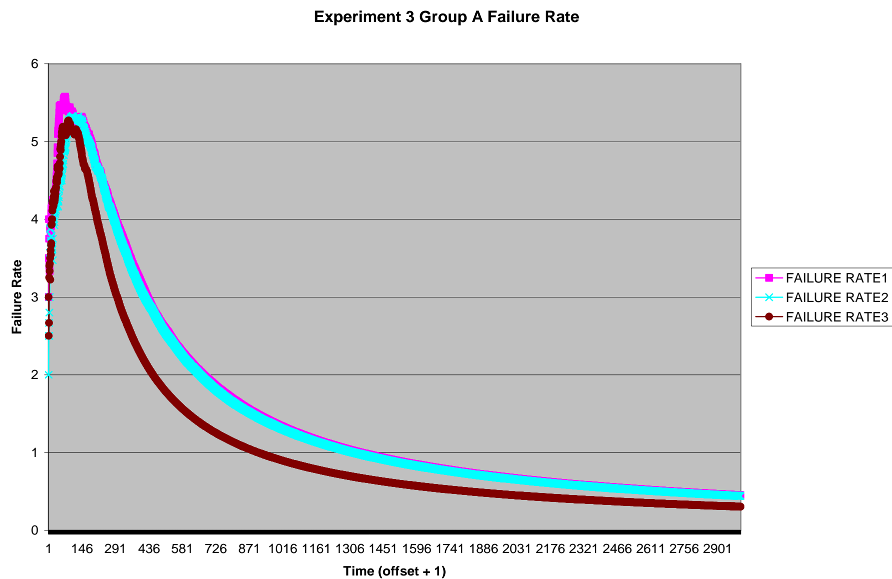
Trial: 1 MAS_Size: 50 TimeSteps: 3000 NumAlive: 25 NumK: 4 NumDeceptive: 25 WeightAge: 1.0 WeightSuccessfulTasks: 0.5 WeightBasicTrust: 1.0 WeightRisk: 1.0 TrustUpdateRate: .10 LearningRate 10	Trial: 2 MAS_Size: 50 TimeSteps: 3000 NumAlive: 25 NumK: 4 NumDeceptive: 25 WeightAge: 1.0 WeightSuccessfulTasks: 0.5 WeightBasicTrust: 1.0 WeightRisk: 1.0 TrustUpdateRate: .10 LearningRate 5	Trial: 3 MAS_Size: 50 TimeSteps: 3000 NumAlive: 25 NumK: 4 NumDeceptive: 25 WeightAge: 1.0 WeightSuccessfulTasks: 0.5 WeightBasicTrust: 1.0 WeightRisk: 1.0 TrustUpdateRate: .10 LearningRate 1

المنارة للاستشارات
www.manaraa.com

Table 13: Experiment 3 Group A Observations

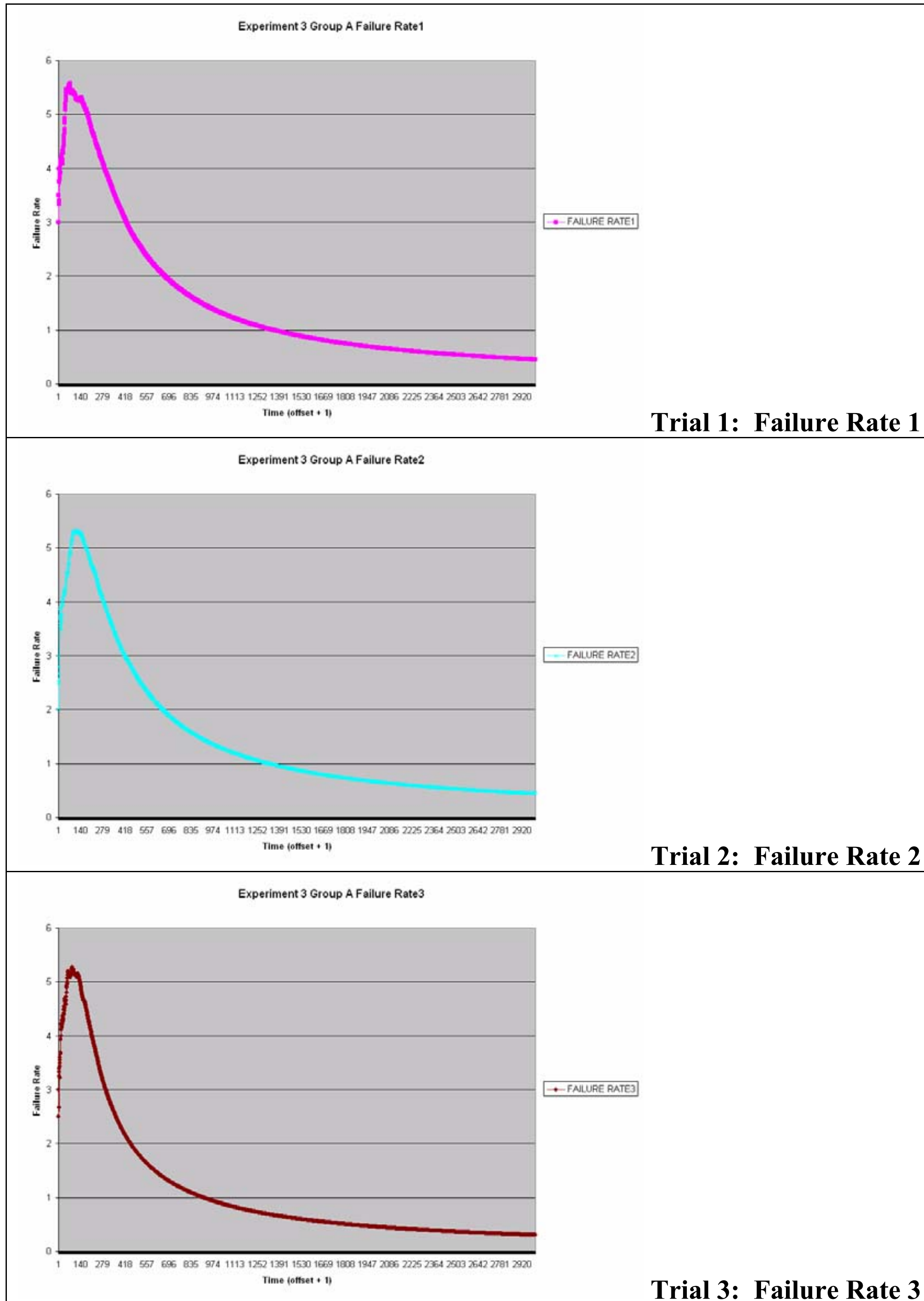
	Trial1	Trial2	Trial3
Max Failure Rate	5.5753427	5.3301888	5.2727275
Time Step at Max	72	105	87
$Slope_{LR}$	-0.00122704	-0.00115629	-0.0009671
Elapsed Time (End – Max)	2927	2894	2912
$Input\ Time_B$	614	601	418
$Failure_B$	2.1658537	2.1677742	2.1646779
$Avg_Velocity_B$	-0.00629057	-0.00637584	-0.0093899
$Elapsed\ Time_B$ (Input – Max)	542	496	331
$Input\ Time_R$	800	700	400
$Failure_R$	1.6828964	1.8744651	2.2618454
$Avg_Velocity_R$	-0.005346767	-0.00580794	-0.0096194
$Elapsed\ Time_R$ (Input – Max)	728	595	313

Figure 14: Experiment 3 Group A Failure Rate



المنارة للاستشارات
www.manaraa.com

Figure 15: Experiment 3 Group A Individual Failure Rate

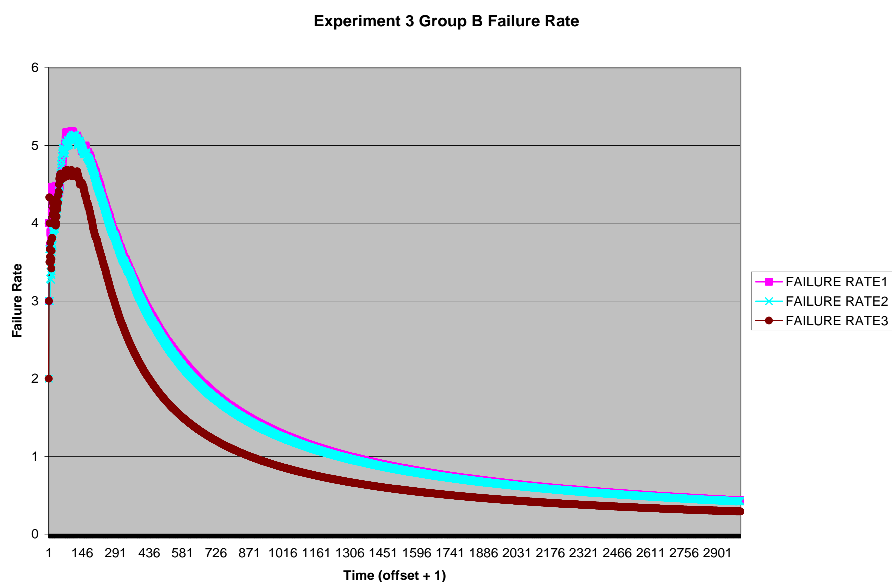


www.manaraa.com

Table 14: Experiment 3 Group B Observations

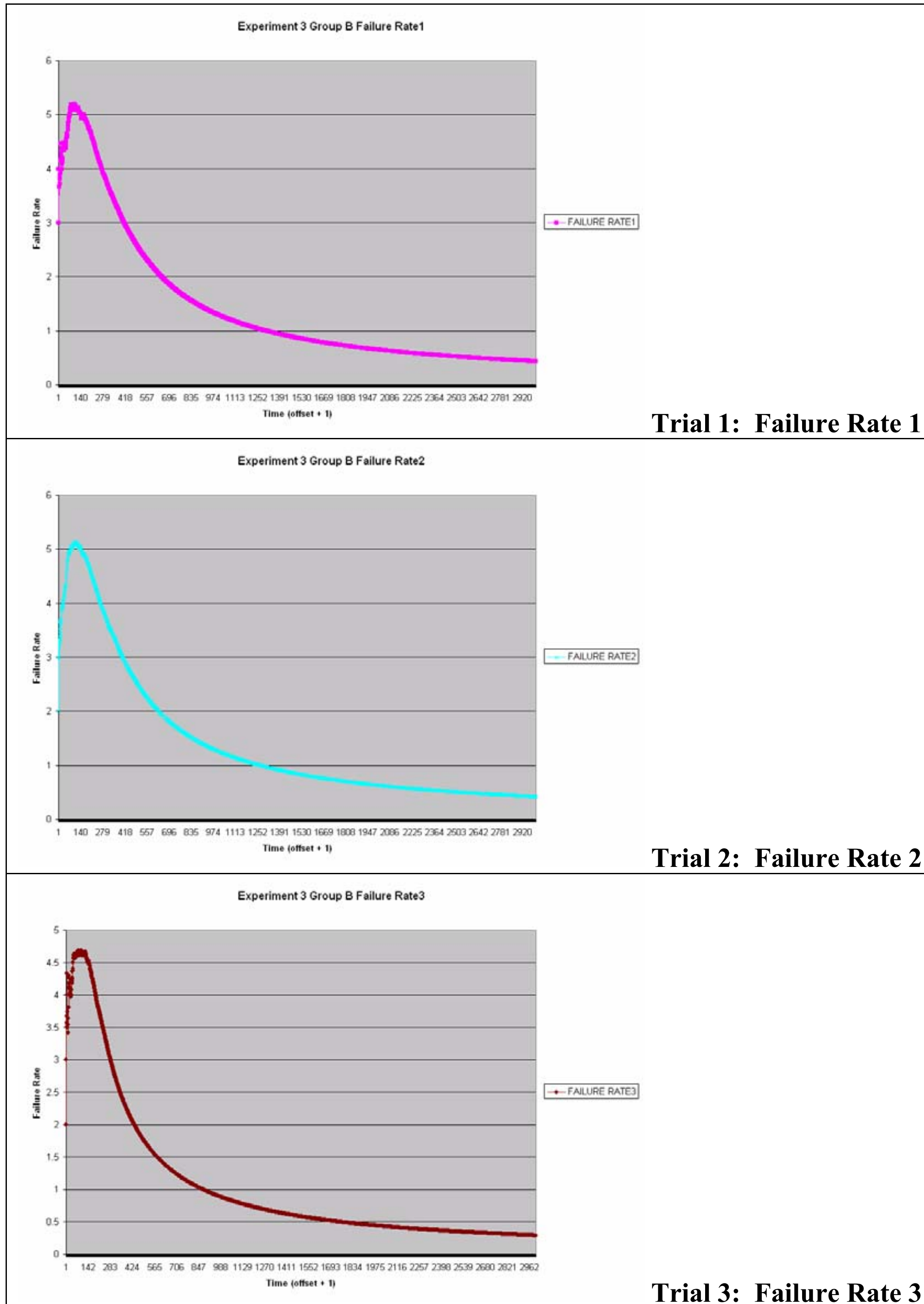
	Trial1	Trial2	Trial3
Max Failure Rate	5.19	5.141593	4.6875
Time Step at Max	99	112	79
$Slope_{LR}$	-0.001143028	-0.00110183	-0.0009231
Elapsed Time (End – Max)	2900	2887	2920
$Input\ Time_B$	597	578	401
$Failure_B$	2.165552	2.1658032	2.164179
$Avg_Velocity_B$	-0.006073189	-0.00638582	-0.0078364
$Elapsed\ Time_B$ (Input – Max)	498	466	322
$Input\ Time_R$	800	700	500
$Failure_R$	1.6229713	1.7960057	1.742515
$Avg_Velocity_R$	-0.005088486	-0.00568977	-0.0069952
$Elapsed\ Time_R$ (Input – Max)	701	588	421

Figure 16: Experiment 3 Group B Failure Rate



www.manaraa.com

Figure 17: Experiment 3 Group B Individual Failure Rate

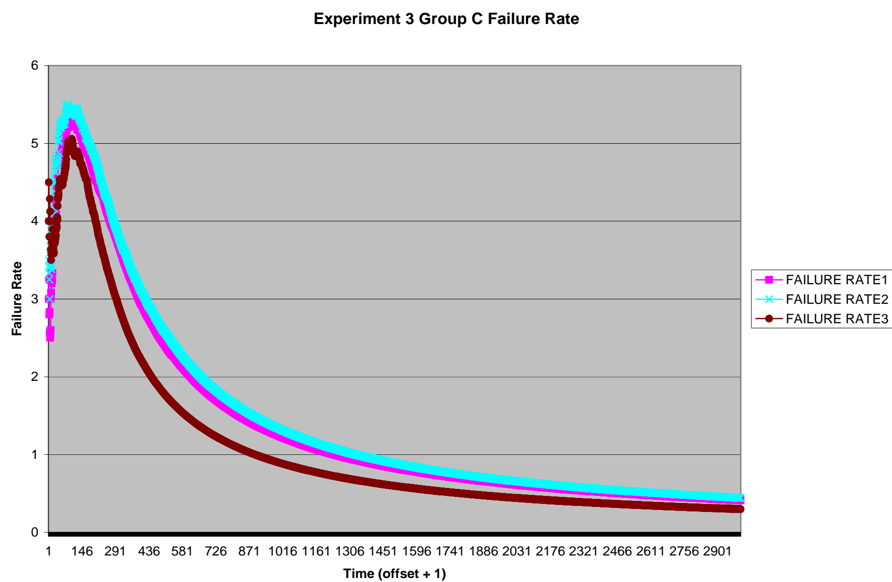


المنارة للاستشارات
www.manaraa.com

Table 15: Experiment 3 Group C Observations

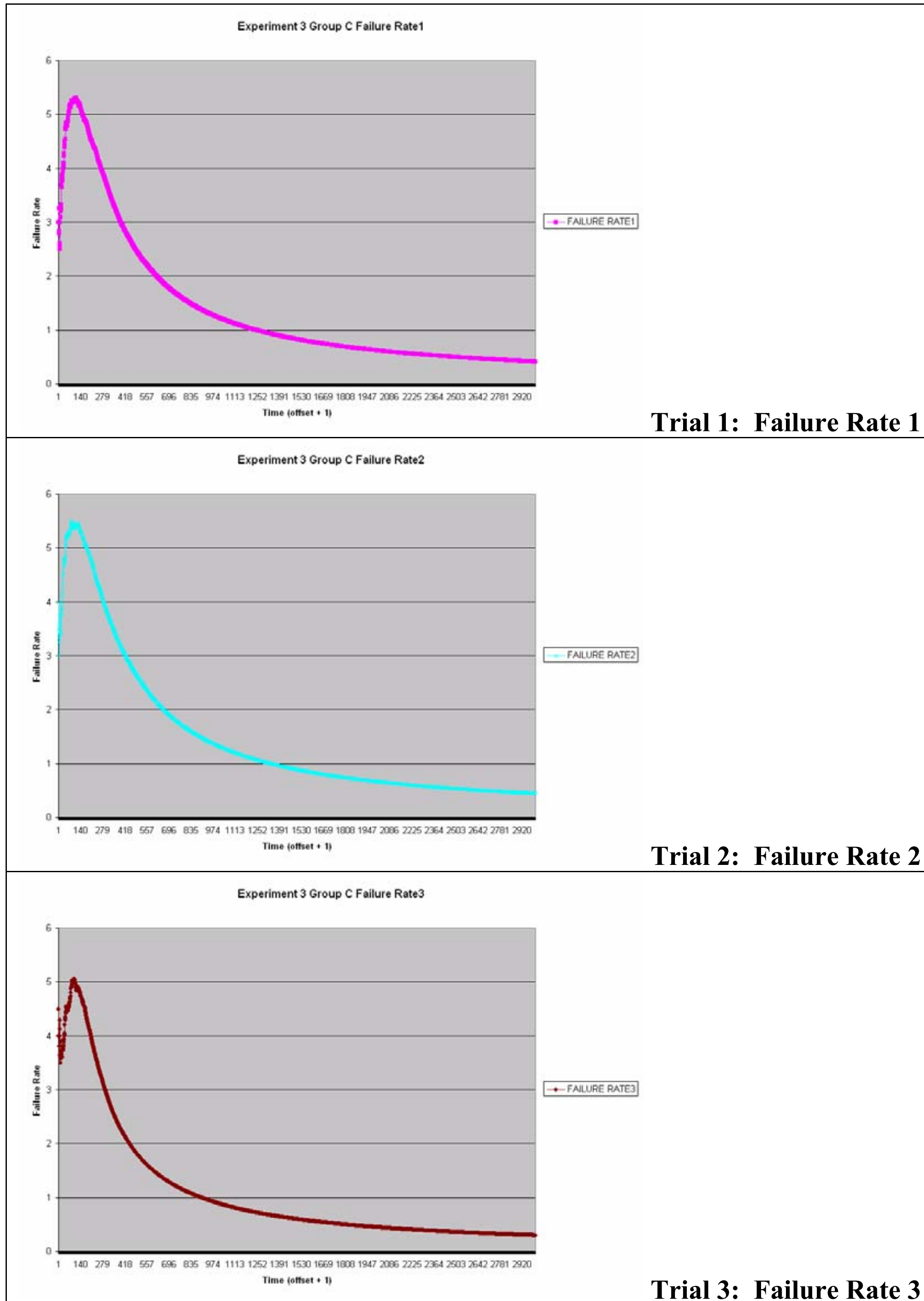
	Trial1	Trial2	Trial3
Max Failure Rate	5.3153152	5.506024	5.0588236
Time Step at Max	110	82	101
$Slope_{LR}$	-0.001104168	-0.00119997	-0.0009184
Elapsed Time (End – Max)	2889	2917	2898
$Input\ Time_B$	568	607	411
$Failure_B$	2.1652021	2.1677632	2.1674757
$Avg_Velocity_B$	-0.006877976	-0.00635859	-0.0093269
$Elapsed\ Time_B$ (Input – Max)	458	525	310
$Input\ Time_R$	800	800	500
$Failure_R$	1.548065	1.6579276	1.7864271
$Avg_Velocity_R$	-0.005459783	-0.00535947	-0.0082015
$Elapsed\ Time_R$ (Input – Max)	690	718	399

Figure 18: Experiment 3 Group C Failure Rate



www.manaraa.com

Figure 19: Experiment 3 Group C Failure Rate



www.manaraa.com

Experiment 3 Conclusions:

Hypothesis 3.1 is not supported by experiment results. Across the three groups, different trials have the lowest elapsed time needed to reach max failure rate as indicated by the rows labeled "Time Step at Max" in Tables 16, 17, and 18. This indicates that varying the learning rate does not add to performance as was expected in this aspect of the system. This can also be seen by looking at results from Experiment 1 in Tables 8, 9, and 10. K[6] and KE[6,10] of Experiment 1 (Trials 2 and 3) only differ in that KE[6,10] employs a learning rate. Only in group C of Experiment 1, does K[6,10] reach the max failure rate in less time than K[6]. Across all Experiment 1 groups, KE[6,10] does outperform K[0] (Trial 1). However, this is expected, and hypothesis 3.1 seeks to justify the advantage of using more exploration. Experiment 2 provides better results because using more neighbors can provide reputation recommendations early on. Decreasing the learning rate and allowing more exploration does not benefit the system if enough interactions have not occurred to build reputations that label agents as untrustworthy.

In contrast to the other experiments, the relative convergence point has been varied to focus only on periods of increasing returns. In experiments 1 and 2, relative convergence was fixed at 1000 life cycles for all groups and trials. In experiment 3, the chosen point of relative convergence is changed for each specific trial within an experiment group and is represented by the row labeled "Input $Time_R$" in Tables 16, 17, and 18. Between the period of max failure rate and relative convergence,

المنارة للاستشارات
www.manaraa.com

K[4,1] (Trial 3) performing exploration at every time step of life cycle execution outperforms the other trials across all groups to support hypothesis 3.2. This performance is measured by $Avg_Velocity_R$. K[4,5] (Trial 2) outperforms K[4,10] (Tial 1) in two out of three groups. In group C, the slopes for K[4,10] and K[4,5] are relatively equivalent. K[4,1] further supports hypothesis 3.2 during the period between max failure rate and the benchmark failure rate by outperforming all other trails across all three groups as measured by $Avg_Velocity_B$. Again, group C (Table 15 and Figures 18 -19) provides the only exception with K[4,10] outperforming K[4,5]. Since slope is partially determined by the change in failure rate over elapsed time, the higher slope for K[4,10] can be explained by the larger elapsed time for K[4,5] in both slope measurements. In both cases, K[4,5] does have the highest change in failure rate(change in y) between max failure and benchmark ($Avg_Velocity_B$), as well as between max failure and relative convergence ($Avg_Velocity_R$). The results show the continued dependency on direct interaction to build the pool of neighbor recommendations that will provide a low enough trust value to negate cooperation with untrustworthy partners. This could indicate that during earlier life cycles where K[4,5] reached its peak failure rate sooner, exploration allowed it to identify more deceptive agents than K[4,10], but it took longer to identify all untrustworthy agents thereby allowing failures to continue.

المنارة للاستشارات
www.manaraa.com

Finally, hypothesis 3.3 (asserting that that elapsed time between the max failure rate and the benchmark failure rate will decrease as the learning rate decreases) is supported. As indicated in the observations supporting hypothesis 3.2, group C is the only exception. On average, KE[4,10] achieves the benchmark in 499 life cycles, KE[4,5] in 495 life cycles, and KE[4,1] in 321 life cycles.

In conclusion, hypothesis 3.1 is not supported. The suggested explanation is that exploration only aids performance when neighbors are known and a sufficient number of interactions have taken place. Once this occurs, greater performance returns are found as indicated by measuring the slope of the curve to estimate average velocity towards the benchmark failure rate and convergence to support hypothesis 3.2. Hypothesis 3.3 is supported.

To summarize the results of the KMAS experiments, the goals of this research are stated here. A goal of DAI research is to develop cooperation models to increase task completion rates by avoiding harmful interactions between distributed components in a DAI system. Here, in this research, the components are autonomous, intelligent, adaptive, and rational agents that may seek self interested behavior, and in doing so, may cause harmful interactions intentionally or otherwise. DAI systems are also used to test theories about reasoning processes. In the KMAS experiment, the reasoning process has been described as a process of determining the trustworthiness of potential interaction partners to cooperate with. The goal of the KMAS research is to model a

المنارة للاستشارات
www.manaraa.com

multi-agent system composed of agents with characteristics described above that will reason about the trustworthiness of potential agent partners. In doing so, harmful interactions will be minimized, and then eliminated as the system converges to a state where only cooperation with non-deceitful partners exists as an emergent property of the system. Another research goal is to determine the benefits of a recommendation-based reputation model of trust using the k-Nearest Neighbor learning algorithm, and to measure its performance.

The documented experiment results show that a MAS system using trust can converge to a state where potentially harmful interactions are reduced, then eliminated. This is an emergent property of the KMAS system because although each agent has a model of trust that does not allow cooperation with any deceitful partner after all agents have entered the system, each agent alone cannot define a system state, only its local state within the environment. This is critical, and more discussion of this will be presented in Section 5.3. Furthermore, it is not guaranteed that all deceitful agents will be known to every other agent. However, the reputations of these agents may be available if requested. It was also demonstrated that the recommendation-based model of trust can outperform a system using only direct interaction and the absence of the nearest neighbor algorithm. This support was further bolstered by results that showed a bounded increase in performance when the number of agents providing the recommendations was increased. Performance was also enhanced by increasing the number of times recommendations were requested or provided.

المنارة للاستشارات
www.manaraa.com

SECTION 5.3: FUTURE RESEARCH

In all but one of the stated hypotheses (hypothesis 3.1 asserting that the number of life cycles needed to reach max failure rate will decrease with more exploration), experiment results fully or strongly supported hypothesis claims. Hypotheses 2.1 and 2.2 were strongly supported despite the presense of explainable experiment results in some of the trial groups which were contrary to the hypotheses. What is being attempted is a "support by example" of each hypothesis to show that the hypotheses can be supported in the context of the KMAS experiments that have been recorded and presented. However, the trial groups show that in some cases, the KMAS experiment can provide examples of an exception. Stronger support or proof techniques may be needed in future research combined with modified hypotheses.

The agents in KMAS use a tuple of weighted agent characteristics as input to the k-Nearest Neighbor algorithm. The Euclidean distance finds other agents with similar characteristics. The reasoning behind this is that similar agents may recommend the most appropriate values of reputation for the unknown trading partner. The chosen characteristics were age, number of successful tasks, basic trust, and basic risk. In theory, this would be useful in situations in which agent characteristics have a bearing on the cooperative task and its results. For instance, if chosen interaction partners defect during cooperation because of the age of the requester (seen as inexperienced),

المنارة للاستشارات
www.manaraa.com

agents of the same age may accurately model the untrustworthiness of other agents who are biased towards age.

Future research could identify the propensity of a system to display neighborhood convergence as an emergent property where agents with similar characteristics will only interact with agents that have certain characteristics themselves or desirable traits. The system could function in a similar fashion to a genetic algorithm where only the most "fit" agents are involved in cooperation.

A second area of research could involve changing the tuple representing an agent so that it reflects the characteristics of the unknown agent and the cooperative task that will be undertaken. A requester agent would then solicit recommendations from others that match in Euclidean distance to the tuple. This is more in line with the standard implementation of the k-Nearest Neighbor algorithm. In this way, agent classifications will be trusted or distrusted. If an agent is not trustworthy according to a certain task and the recommendations of others, it will not be cooperated with. The emergent property of such a system could be that it only allows cooperation with agents that are "right for the job". In this way, trust can be dependent on the situation (task) and the competence of the agent that is being requested as an interaction partner.

A third area for future research might investigate trust "direction". In KMAS, trust is only applied in one direction, from the viewpoint of the requester. If an

www.manaraa.com

interaction partner defects from the cooperative agreement, trust is decreased as a negative reward. The requester may decide not to cooperate, but this is not viewed as a defection because the cooperative agreement is not "sealed" until the requester agrees to cooperate. The system could be modified such that the decision of the requester is viewed as a defection in the eyes of the potential partner, prompting the partner to perform trust update. Time is a valuable resource and the interaction partner suffers a loss in resources for both time and the benefits of cooperating with a more agreeable requester. This is similar to the research presented by [Nooteboom et al., 2001] in their ACE model that allows for the coordination of scarce resources based on trust between suppliers and consumers. The modified system could then converge to a state where cooperative partnerships are only allowed between two willing parties. This would be highly desirable in systems where speed is a measurable benchmark for performance, and agents should decide whether or not to cooperate in the fastest amount of time. Nearest neighbors must then be chosen where neighbors recommend the trustworthiness of another agent based on interaction role. An interaction partner may defect as the requester more than they are willing to defect as a partner of a requester agent. This would exhibit rational behavior where requesters are more discriminating and have more to gain or lose based on the type of cooperative tasks. If a cooperative task only benefits a requester, such as in the case of information solicitation, they may have a high rate of defection if the information is mission critical. The risk is far less if they simply fulfill the requirement of that task by providing the information. An obvious

المنارة للاستشارات
www.manaraa.com

exception would be systems where information is generally secured and protected. The trustworthiness of the requester would then be extremely important.

In addition to changing how recommendation-based reputation is modeled, the time and place of reputation building within the system can be modeled in a different manner. As indicated in Section 3.4.5, and as shown through the KMAS experiment results, reputation computed through direct interaction is necessary for agents that are new to the system. It is also necessary as agents build reputation through practical experience that will provide an accurate recommendation-based value of trustworthiness. In a different model of KMAS, agents could be isolated from the main execution space, and placed in a test execution space where interactions and cooperative tasks could be used to build reputation values prior to allowing the agent to enter into the executing environment. This is similar to traditional machine learning approaches that allow the learner to "train" on a set of "training data" or examples that will allow the machine learning algorithm to approximate a target function [Mitchell, 1997]. In KMAS, the target function would represent reputation, and the set of training data would be cooperative tasks.

In order to support research goals, the KMAS experiments were constructed to allow agents to discount trust in untrustworthy agents, and to limit harmful interactions. Future research could allow an agent with a reputation for untrustworthiness to redeem

المنارة للاستشارات
www.manaraa.com

itself in the eyes of other agents. The computational model for trust update inherently allows for trust to be increased if trust has not fallen below an agent's risk threshold (defined as basic risk in Section 5.2.1). If trust is below basic risk after the present cooperative interaction has taken place, or if nearest neighbors provide an unacceptable reputation recommendation, future interactions with the untrustworthy agent will not occur. If agents are allowed to learn trustworthy behavior, and in doing so are "reformed", KMAS could be modified to allow these "reformed" agents to once again interact within the system.

A final area of proposed future research involves identifying areas where it is advantageous to view a MAS adaptive system as a single entity, or agent, using machine learning techniques. KMAS performed the (k X n) Nearest Neighbor algorithm as an emergent property of the system because each agent locally performed the nearest neighbor algorithm to classify other agents. Each agent is only aware of its local state, or its representation of other agents as trustworthy or untrustworthy. Local trust representations may not accurately depict the trustworthiness of an agent. As a sum of all component parts (agents), KMAS as a system environment is aware of the trustworthiness of all agents and determines this by performing the nearest neighbor algorithm globally. It can be viewed as having an i-dimensional search space where each agent resides as a point based on ***i*** characteristics. At the end of performing (k X n) Nearest Neighbor, ***n*** agents have trust values that are updated and are available for propagation throughout the system. If KMAS were modified to allow trust

المنارة للاستشارات
www.manaraa.com

recommendations based on agent characteristics as opposed to actual agents, this implementation of KMAS would provide a stronger example of emergence. In the existing research, unknown agents can only be classified as untrustworthy when enough direct interaction experiences have occurred to build a valuable and accurate pool of recommendations. Using a black box view of the system, a new agent can be introduced into the system, and KMAS would correctly classify it as trustworthy or untrustworthy based on that agent being involved in the interaction process. The agents involved in the classification have no idea that the agent is unknown to the system unless system age is a characteristic in the classification tuple. It is a realistic assumption that an implementation of KMAS in a commercial setting may not have system age as an identifiable characteristic, especially in open system models or environments with highly heterogeneous agent architectures. What important contributions might be made to the areas of agency, DAI, adaptive systems, and machine learning by taking the black box approach to agent system design, testing, and implementation?

المنارة للاستشارات
www.manaraa.com

List of References

المنارة للاستشارات

www.manaraa.com

List of References

[Ashri et al., 2000] R. Ashri, I. Rahwan, and M. Luck. Architectures for Negotiating Agents. In V. Marik, J. Muller, and M. Pechoucek, editors, Multi-Agents Systems and Applications III (LNAI Volume 2691), pages 136-146. Springer-Verlag: Berlin, Heidelberg, New York, 2000.

[Bach, 2004] D.Bach. The Automatic Millionaire. Broadway Books: New York, 2004.

[Barber and Kim, 2001] K. S. Barber and J. Kim. Belief Revision Process Based on Trust: Agents Evaluating Reputation of Information Sources. In R. Falcone, M. Singh, and Y. H. Tan, editors, Trust in Cyber-societies (LNAI Volume 2246), pages 73-82. Springer-Verlag: Berlin, Heidelberg, New York, 2001.

[Barber et al., 2000] K. S. Barber, D. C. Han, and T. H. Liu. Coordinating Decision Making Using Reusable Interaction Specifications. In C. Zhang and V. Soo, editors, Design and Applications of Intelligent Agents (LNAI Volume 1881), pages 1-15. Springer-Verlag: Berlin, Heidelberg, New York, 2000.

[Barber et al., 2003] K. S. Barber, K. Fullam, and J. Kim. Challenges for Trust, Fraud, and Deception Research in Multi-Agent Systems. In R. Falcone, S. Barber, L. Korba, and M. Singh, editors, Trust, Reputation, and Security: Theories and Practice (LNAI Volume 2631), pages 8-14. Springer-Verlag: Berlin, Heidelberg, New York, 2003.

[Bazzan, 1997] A. Bazzan. Evolution of Coordination as a Metaphor for Learning in Multi-Agent Systems. In G. Weiss, editor, Distributed Artificial Intelligence Meets Machine Learning (LNAI Volume 1221), pages 117-136. Springer-Verlag: Berlin, Heidelberg, New York, 1997.

[Birk, 2001] A. Birk. Learning to Trust. In R. Falcone, M. Singh, and Y. H. Tan, editors, Trust in Cyber-societies (LNAI Volume 2246), pages 133-144. Springer-Verlag: Berlin, Heidelberg, New York, 2001.

المنارة للاستشارات
www.manaraa.com

[Clark et al., 1997] M. Clark, K. Irwag, and W. Wobcke. Emergent Properties of Teams of Agents in the Tileworld. In L. Cavedon, A. Rao, and W. Wobcke, editors, Intelligent Agent Systems (LNAI Volume 1209), pages 164-176. Springer-Verlag: Berlin, Germany, 1997.

[D'Inverno and Luck, 2001, 2004] M. D'Inverno and M. Luck. Understanding Agent Systems. Springer-Verlag: Berlin, Heidelberg, New York, 2004.

[Durfee et al., 1989] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Trends in Cooperative Distributed Problem Solving. In IEEE Transactions on Knowledge Data Engineering, 1(1):63-83, 1989.

[Falcone et al., 2001] R. Falcone, M. Singh, and Y. H. Tan. Introduction: Bringing Together Humans and Artificial Agents in Cyber-societies: A New Field of Trust Research. In R. Falcone, M. Singh, and Y. H. Tan, editors, Trust in Cyber-societies (LNAI Volume 2246), pages 1-7. Springer-Verlag: Berlin, Heidelberg, New York, 2001.

[Gasser, 1992] L. Gasser. An Overview of DAI. In N. M. Avouris and L. Gasser, editors, Distributed Artificial Intelligence: Theory and Praxis, pages 9 - 30. Kluwer Academic Publishers: Dordrecht, Boston, London, 1992.

[Griffiths and Luck, 1999] N. Griffiths and M. Luck. Cooperative Plan Selection Through Trust. In F. J. Garijo and M. Boman, editors, Multi-Agent Systems Engineering (LNAI Volume 1647), pages 162-174. Springer-Verlag: Berlin, Heidelberg, New York, 1999.

[Hermans, 1996] B. Hermans. Intelligent Software Agents on the Internet. 1996. Available from http://www.hermans.org/agents.

[Jennings, 1996] N. R. Jennings. Coordination Techniques for Distributed Artificial Intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, Foundations of Distributed Artificial Intelligence, pages 187-210. John Wiley & Sons, Inc: New York, Chichester, Brisbane, Toronto, Singapore, 1996.

[Jonker and Treur, 1999] C. M. Jonker and J. Treur. Analysis of Models for the Dynamics of Trust Based on Experiences. In F. J. Garijo and M. Boman, editors, Multi-Agent Systems Engineering (LNAI Volume 1647), pages 221-231. Springer-Verlag: Berlin, Heidelberg, New York, 1999.

المنارة للاستشارات

www.manaraa.com

[Klusch et al., 2003] M. Klusch, S. Bergamaschi, and P. Petta. European Research and Development of Intelligent Information Agents: The AgentLink Perspective. In M. Klusch, S. Bergamaschi, P.Edwards, and P. Petta, editors, Intelligent Information Agents: The AgentLink Perspective (LNAI Volume 2586), pages 1-21. Springer-Verlag: Berlin, Heidelberg, New York, 2003.

[Langley, 1996] P. Langley. Elements of Machine Learning, Morgan Kaufmann Publishers, Inc: San Francisco, California, 1996.

[Lenzmann and Wachsmuth, 1996] B. Lenzmann and I. Wachsmuth. A User-Adaptive Interface Agency for Interaction with a Virtual Environment. In G. Weis and S. Sen, editors, Adaptation and Learning in Multi-Agent Systems (LNAI Volume 1042), pages 140-151. Springer-Verlag: Berlin, Heidelberg, New York, 1996.

[Lenzmann and Wachsmuth, 1997] B. Lenzmann and I. Wachsmuth. Contract-Net-Based Learning in a User-Adaptive Interface Agency. In G. Weiss, editor, Distributed Artificial Intelligence Meets Machine Learning (LNAI Volume 1221), pages 202-222. Springer-Verlag: Berlin, Heidelberg, New York, 1997.

[Malone, 1990] T. W. Malone. Organizing Information Processing Systems: Parallels Between Human Organizations and Computer Systems. In W.W. Zachary, S. P. Robertson, and J. B. Black, editors, Cognition, Computation, and Cooperation, pages 56-83. Ablex: Norward, New Jersey, 1990.

[Mandalapu and Adya, (n.d.)] S. Mandalapu and K. Adya. Search Engine Using Mobile Agents and Peer to Peer Network. Southern Illinois University. (n.d.). Retrieved May 6, 2005, from http://www.cs.siu.edu/~smanda/paper.pdf.

[Marsh, 1994] S. Marsh. Trust in Distributed Artificial Intelligence. In C. Castelfranchi and E. Werner, editors, Artificial Social Systems (LNAI Volume 830), pages 94-112. Springer-Verlag: Berlin, Heidelberg, New York, 1994.

[Mass and Shehory, 2001] Y. Mass and O. Shehory. Distributed Trust in Open Multi-Agent Systems. In R. Falcone, M. Singh, and Y. H. Tan, editors, Trust in Cyber-societies (LNAI Volume 2246), pages 159-173. Springer-Verlag: Berlin, Heidelberg, New York, 2001.

المنارة للاستشارات
www.manaraa.com

[Mateas, 1997] M. Mateas. An Oz-Centric Review of Interactive Drama and Believable Agents. Carnegie Mellon University. 1997. Available from
[McKnight and Chervany, 2001] D. McKnight and N. L. Chervany. Trust and Distrust Definitions: One Bite at a Time. In R. Falcone, M. Singh, and Y. H. Tan, editors, Trust in Cyber-societies (LNAI Volume 2246), pages 27-54. Springer-Verlag: Berlin, Heidelberg, New York, 2001.

[Mitchell, 1997] T. M. Mitchell. Machine Learning. The MIT Press and the McGraw-Hill Companies, Inc., 1997.

[Moulin and Chaib-Draa, 1996] B. Mouline and B. Chaib-Draa. An Overview of Distributed Artificial Intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, Foundations of Distributed Artificial Intelligence, pages 1-55. John Wiley & Sons, Inc: New York, Chichester, Brisbane, Toronto, Singapore, 1996.

[Mui et al., 2003] L. Mui, A. Halberstadt, and M. Mohtashemi. Evaluating Reputation in Multi-Agent Systems. In R. Falcone, S. Barber, L. Korba, and M. Singh, editors, Trust, Reputation, and Security: Theories and Practice (LNAI Volume 2631), pages 123-137. Springer-Verlag: Berlin, Heidelberg, New York, 2003.

[Murch and Johnson, 1999] R. Murch and J. Johnson. Intelligent Software Agents. Prentice Hall PTR: Upper Saddle River, New Jersey, 1999.

[N. R. Jennings, 1999] N. R. Jennings. Agent-Oriented Software Engineering. In F. J. Garijo and M. Boman, editors, Multi-Agent System Engineering (LNAI Volume 1647), pages 1-7. Springer-Verlag: Berlin, Heidelberg, New York, 1999.

[Nooteboom et al., 2001] B. Nooteboom, T. Klos, and R. Jorna. Adaptive Trust and Co-operation: An Agent-Based Simulation Approach. In R. Falcone, M. Singh, and Y. H. Tan, editors, Trust in Cyber-societies (LNAI Volume 2246), pages 83-109. Springer-Verlag: Berlin, Heidelberg, New York, 2001.

[Nunes and Oliveira, 2003] L. Nunes and E. Oliveira. Cooperative Learning Using Advice Exchange. In E. Alonso, D. Kudenko, and D. Kazakov, editors, Adaptive Agents and Multi-Agent Systems (LNAI Volume 2636), pages 33-48. Springer-Verlag: Berlin, Heidelberg, New York, 2003.

[Ossowski, 1999] S. Ossowski. Coordination in Artificial Agent Societies: Social Structure and Its Implications for Autonomous Problem-Solving Agents (LNAI Volume 1535). Springer-Verlag: Berlin, Heidelberg, New York, 1999.

المنارة للاستشارات
www.manaraa.com

[Primeaux, 2000] D. Primeaux. "Trust Based Learning of Data Characteristics by an Actual Entity," Proceedings of the ACIS 1st International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing (SNPD '00), Reims, France, May, 2000, pp. 239-244.

[Ramaniuk, 2000] S. G. Romaniuk. Using Intelligent Agents to Identify Missing and Exploited Children. In IEEE Intelligent Systems, 15(2):27-30, 2000.

[Russell, 1999] S. Russell. Rationality and Intelligence. In M. Wooldridge and A. Rao, editors, Foundations of Rational Agency, pages 11-33. Kluwer Academic Publishers: Dordrecht, Boston, London, 1999.

[Schumacher, 2000] M. Schumacher. Objective Coordination in Multi-Agent System Engineering (LNAI Volume 2039), Springer-Verlag: Berlin, Heidelberg, New York, 2001.

[Singh and Huhns, 1997] M. P. Singh and M. N. Huhns. Challenges for Machine Learning in Cooperative Information Systems. In G. Weiss, editor, Distributed Artificial Intelligence Meets Machine Learning (LNAI Volume 1221), pages 11-24. Springer-Verlag: Berlin, Heidelberg, New York, 1997.

[Vidal, 2003] J. M. Vidal. Learning in Multiagent Systems: An Introduction from a Game-Theoretic Perspective. In E. Alonso, D. Kudenko, and D. Kazakov, editors, Adaptive Agents and Multi-Agent Systems (LNAI Volume 2636), pages 202-215. Springer-Verlag: Berlin, Heidelberg, New York, 2003.

[Weiss, 1995] G. Weiss. Adaptation and Learning in Multi-Agent Systems: Some Remarks and a Bibliography. In G. Weiss and S. Sen, editors, Adaptation and Learning in Multi-Agent Systems (LNAI Volume 1042), pages 1-21. Springer-Verlag: Berlin, Heidelberg, New York, 1996.

[Witkowski et al., 2001] M. Witkowski, A. Artikis, and J. Pitt. Experiments in Building Experiential Trust in a Society of Objective-Trust Based Agents. In R. Falcone, M. Singh, and Y. H. Tan, editors, Trust in Cyber-societies (LNAI Volume 2246), pages 111-132. Springer-Verlag: Berlin, Heidelberg, New York, 2001.

[Wooldridge and Rao, 1999] M. Wooldridge and A. Rao. Foundations of Rational Agency. In M. Wooldridge and A. Rao, editors, Foundations of Rational Agency, pages 1-10. Kluwer Academic Publishers: Dordrecht, Boston, London, 1999.

[Wooldridge, 2000] M. Wooldrige. Reasoning about Rational Agents. The MIT Press: Cambridge, MA, 2000.
http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/oz/web/papers/CMU-CS-97-156.html.

للاستشارات المنارة
www.manaraa.com

Appendices

المنارة للاستشارات
www.manaraa.com

APPENDIX A

fixedIn.txt

Example running program CreateFixedInputs and specifying 5 agents in the MAS with 2 agents being deceptive.

```
0.18818990734061736        0.8335305736671379
0.9590903038864445         0.08191107009997667
0.10470017444724145        0.8912400614943223
0.011554138422806393       0.5192266119404704
0.5631868953287027         0.7218656928570173
---------------begin deceptive input-----------------
0.03205061564866751        0.38324359015333176
0.8528369892992353         0.3073354210419863
```

www.manaraa.com

APPENDIX B

Experiment Trial Input

```
MAS_Size:       50
TimeSteps:      3000
NumAlive:       25
NumK:           0
NumDeceptive:   25
WeightAge:      1.0
WeightSuccessfulTasks:      0.5
WeightBasicTrust:   1.0
WeightRisk:         1.0
TrustUpdateRate:    .10
LearningRate        0
Debug:              N
Outfile:            c:\javatst\kmas\reports\e1trial1
FixedInput:         Y
FixedInputFile:             c:\javatst\kmas\docs\fixedIn.txt
FixedDeception:             Y
```

المنارة للاستشارات
www.manaraa.com

APPENDIX C

Failure Rate Log Example

..\docs\exp1\e1trial1.txt

OUTPUT: c:\javatst\kmas\reports\e1trial1failures.091305210940.txt

```
TIMESTEP,NUM ALIVE,NUM ALIVE DECEPTIVE,FAILURES,FAILURE RATE
0,25,14,2,2.0
1,25,14,2,2.0
2,25,14,3,2.3333333
3,26,14,4,2.75
4,27,15,2,2.6
5,28,15,5,3.0
6,28,15,4,3.142857
7,29,15,0,2.75
8,29,15,2,2.6666667
9,30,15,0,2.4
10,31,15,5,2.6363637
11,31,15,3,2.6666667
12,31,15,5,2.8461537
13,31,15,2,2.7857144
14,31,15,3,2.8
15,31,15,0,2.625
16,32,15,2,2.5882354
17,32,15,5,2.7222223
18,33,16,5,2.8421052
19,34,17,6,3.0
20,34,17,5,3.0952382
21,35,17,2,3.0454545
22,35,17,3,3.0434783
23,36,17,5,3.125
24,37,18,2,3.08
25,37,18,3,3.0769231
26,37,18,3,3.074074
27,38,19,8,3.25
28,38,19,5,3.310345
29,38,19,4,3.3333333
30,38,19,3,3.3225806
```

www.manaraa.com

31,39,20,7,3.4375
32,39,20,4,3.4545455
33,39,20,5,3.5
34,40,20,7,3.6
35,40,20,5,3.6388888
36,40,20,2,3.5945945
37,40,20,6,3.6578948

المنارة للاستشارات
www.manaraa.com

APPENDIX D

Agent Cooperation Log Example

Example created using 5 agents, two of which are deceptive, and no fixed inputs. Initial agent values are echoed.

1) Echoed initial parameters

KMAS INPUT PARAMETERS:

MAS_Size: 5
TimeSteps: 10
Initially Alive: 2
NumK: 3
NumDeceptive: 2
WeightAge: 1.0
WeightSuccessfulTasks: 0.5
WeightBasicTrust: 1.0
WeightRisk: 1.0
TrustUpdateRate: 0.1
LearningRate: 0
Debug: true
Outfile Prefix: c:\javatst\kmas\reports\example
Fixed Input Flag: false
Fixed Input File: c:\javatst\kmas\docs\fixedIn_example.txt
Fixed Deception Flag: false

BEGINNING AGENT VALUES:

Agent ID: 0
Alive: true
Partner: -1
Deceptive: 0
Basic Trust: 0.4507852644096837
Risk: 0.8759343386302918

Agent ID: 1
Alive: true
Partner: -1
Deceptive: 1
Deception Level: 0.11101908277973327 Deception Threshold: 0.0
Basic Trust: 0.34609323969918715
Risk: 0.8945244545584091

www.manaraa.com

Agent ID: 2
Alive: false
Partner: -1
Deceptive: 0
Basic Trust: 0.7975653290553835
Risk: 0.3347057059623324

Agent ID: 3
Alive: false
Partner: -1
Deceptive: 0
Basic Trust: 0.3926212493801474
Risk: 0.16903337755600345

Agent ID: 4
Alive: false
Partner: -1
Deceptive: 1
Deception Level: 0.9144526621377972 Deception Threshold: 0.0
Basic Trust: 0.5267962513070971
Risk: 0.40443529033625214

----

2) Agent #2 cooperation log

Agent #2 performs k-Nearest Neighbor at Time Step 4 for unknown Agent #4. Agent #4 defects and trust is discounted. Agent #2 cooperates with Agent #4 again in Time Step 7, and Agent #4 again defects because it is flagged as deceptive. Trust is again updated and decreased. It will only take one more defection to cause situational trust to be below risk. When this occurs, Agent #2 will no longer cooperate with Agent #4 because it is now seen as untrustworthy.

Time Step: 3 Agent Age: 2
Alive: true
Agent ID: 2 Requester Agent: true
Total Successes: 1 Total Failures: 0
Has Partner: true
Partner ID: 1
Nearest Neighbors: 4 0 1
Basic Trust: 0.7975653290553835
Old General Trust In Partner: 0.5
New General Trust In Partner: 0.525
Situational Trust: 0.39878266452769173 Risk: 0.3347057059623324
Will Cooperate: true
Success: true
Num Success with Partner: 1 Num Failure: 0
****

المنارة للاستشارات
www.manaraa.com

Time Step: 4 Agent Age: 3
Alive: true
Agent ID: 2 Requester Agent: true
Total Successes: 1 Total Failures: 1
Has Partner: true
Partner ID: 4
Nearest Neighbors: 4 0 1
Basic Trust: 0.7975653290553835
Old General Trust In Partner: 0.5
New General Trust In Partner: 0.425
Situational Trust: 0.39878266452769173 Risk: 0.3347057059623324
Will Cooperate: true
Success: false
Num Success with Partner: 0 Num Failure: 1
****
Time Step: 6 Agent Age: 5
Alive: true
Agent ID: 2 Requester Agent: true
Total Successes: 2 Total Failures: 1
Has Partner: true
Partner ID: 3
Nearest Neighbors: 4 0 1
Basic Trust: 0.7975653290553835
Old General Trust In Partner: 0.5
New General Trust In Partner: 0.525
Situational Trust: 0.39878266452769173 Risk: 0.3347057059623324
Will Cooperate: true
Success: true
Num Success with Partner: 1 Num Failure: 0
****
Time Step: 7 Agent Age: 6
Alive: true
Agent ID: 2 Requester Agent: true
Total Successes: 2 Total Failures: 2
Has Partner: true
Partner ID: 4
Basic Trust: 0.7975653290553835
Old General Trust In Partner: 0.425
New General Trust In Partner: 0.36443749999999997
Situational Trust: 0.33896526484853795 Risk: 0.3347057059623324
Will Cooperate: true
Success: false
Num Success with Partner: 0 Num Failure: 2
****

المنارة للاستشارات
www.manaraa.com

APPENDIX E

class CreateFixedInputs

```
import java.io.*;
import java.util.*;

public class CreateFixedInputs
{


public static void main (String[] args) throws Exception {

/****************************************************************************/
/*
Method main:

1) Receives two integer command line inputs, one equal to the maximum number of agents in the MAS,
and the other equal to the maximum number of deceptive agents in the MAS.

2) Outputs two column rows up to the maximum number of agents with the first column consisting of
basic trust values, and the second consisting of risk values. All values created using a random number
generator.

3) Outputs two column rows up to the maximum number of deceptive agents with the first column
consisting of the agent's level of deception, and the second consisting of the agent's deceptive threshold.
All values are created using a random number generator.

4) Results of outputs are written to file c:\\javatst\\kmas\\docs\\fixedIn.txt
*/
/****************************************************************************/


//Variable declarations

int numAgents = Integer.parseInt(args[0]);
int numDecAgents = Integer.parseInt(args[1]);
FileWriter outfile = new FileWriter("c:\\javatst\\kmas\\docs\\fixedIn.txt");
Random rand = new Random();
```

المنارة للاستشارات
www.manaraa.com

```
//Method Execution

try
{
  for (int i = 0; i < numAgents; i++)
    outfile.write(rand.nextDouble() + "\t" + rand.nextDouble() + "\r\n");
  outfile.write("---------------begin deceptive input-----------------\r\n");
  for (int i = 0; i < numDecAgents; i++)
    outfile.write(rand.nextDouble() + "\t" + rand.nextDouble() + "\r\n");
  outfile.flush();
  outfile.close();
}
catch (Exception e) {System.err.println(e);}

}  //        End method main()

}  //        End class CreateFixedInputs
```

www.manaraa.com

APPENDIX F

class ThesisKmas

```
public class ThesisKmas
{


public static void main (String[] args) {

/*****************************************************************************/
/*
Method main:

  This method is the main executable for the KMAS experiment: a MAS society
implementing the K x N -nearest neighbor learning algorithm as a cooperation strategy
for unknown potential partners.

1)  Receives command line input that specifies path and name of a file containing experiment
input which is read by class Kmas.

2)  Creates a Kmas object which is the executable experiment trial.

3)  Feeds experiment input file contents into the KMAS experiment environment.

4)  Populates KMAS with randomly selected agents from the pool of available agents, and activates them
according to the number of initially "alive" agents specified in the experiment input file.

5)  Randomly selects active agents and makes them deceptive according to the number specified in the
experiment file.

6)  Executes KMAS according to the maximum number of time steps specified in the experiment input
file.  Outputs cooperation log if in debug mode as well as a listing of initial agent values.

7)  Outputs failure rate log.
*/
/*****************************************************************************/


//Variable declarations

  Kmas systemK;
```

المنارة للاستشارات
www.manaraa.com

```
//Method Execution

 systemK = new Kmas(args[0]);
 try
 {
  systemK.getInput();
  systemK.printInput();
  systemK.createMAS();
  systemK.setDeceptiveAgents();
  while (systemK.getTimeStep() < systemK.getMaxTime())
   systemK.executeMAS();
  systemK.printFailures();
 }
 catch(Exception e)
  {System.err.println(e);}

}  //          End method main()

}  //          End class ThesisKmas
```

المنارة للاستشارات
www.manaraa.com

APPENDIX G

class Kmas

```
import java.io.*;
import java.util.*;
import java.text.*;

public class Kmas
{


/****************************************************************************/
/*
class Kmas:

Executable experiment that defines the MAS and the necessary methods to execute one experiment life
cycle.

1)  After instantiation by class ThesisKmas, receives and stores experiment input file contents specified
as command line file input while executing ThesisKmas.

2)  Creates all agents and randomly sets a maximum number of agents to be initially active in the system.
Once an agent is made active, it stays active.

3)  Randomly selects a set number of agents to be deceptive.  Initially, deceptive agents can be active or
inactive.

4)  Uses contents of fixed input file to give basic trust, risk, deception, and deception threshold values to
all agents if input file parameter FixedInput is set to Y.  If not, random values are created using a random
number generator.  If fixed deception is turned on, deceptive agents receive values for deception and
deceptive threshold.  If experiment input file parameter FixedDeception is N, random values are given
and each agent will produce a new deceptive threshold value each time cooperation is required.

5)  At the beginning of time step (life cycle), randomly adds or does not add a new agent to the system by
making a non-active agent active.  Resets agent cooperation variables to default (agent ID of cooperative
partner, decision to cooperate, "has partner" flag, cooperation success flag, requester agent designator
flag).

6)  Initiates interaction between requester agents and selected partners and outputs to cooperation log if in
debug mode.

7)  Records and stores data needed to create the failure rate log.
*/
/****************************************************************************/
```

المنارة للاستشارات
www.manaraa.com

```
//Class variable declaration:

String [] inputVals;      //holds contents of input file specified in command line input
int mas_Size;      //input, maximum number of agents in the MAS
int maxTime;      //input, maximum number of time steps or executable life cycles
int timeStep;      //current time step
int numAlive;      //input, maximum number of agents initially active in the environment
int numDecep;      //not currently used
int totFailures;      //counter, number of cooperation failures
int [] numAgentsAlive;      //array of number of agents alive by executed time step
double [] failureRate;      //array of failure rates at the end of each time step
int [] tallyF;      //array of number of total failures by time step
int [] numDecepAlive;      //array of number of deceptive agents by time step
KmasAgent [] mas;      //array of agent objects defining the MAS environment
FileWriter [] out;      //array of agent cooperation logs
int k_Nearest;      //input, number of nearest neighbors
int numDeceptive;      //input, number of deceptive agents
double weightAge;      //input, Euclidean weight for agent age
double weightSuccessful;      //input, Euclidean weight for number of successful cooperation results
double weightBasicT;      //input, Euclidean weight for basic trust
double weightRisk;      //input, Euclidean weight for risk
double tuRate;      //input, trust update rate
int lrnRate;      //input, learning rate or value for exploration
boolean debug;      //input, debug mode
String prefix;      //input, prefix for path and beginning filename for failure rate log
boolean fixedInFlag;      //input, determines if fixed input is chosen
boolean fixedDeceptionFlag;      //input, determines if values for deception are fixed
FileReader fixedInFile;      //fixed input file
FileWriter outFile;      //output files
FileReader inFile;      //input file object
String inFileName;      //input filename


public Kmas(String theFile)      //constructor
{
 try
 {
  inFile = new FileReader(theFile);
  inFileName = theFile;
 }
 catch (Exception e) {System.err.println(e);}

}  //  end constructor


public int getTimeStep() { return timeStep; }


public int getMaxTime() { return maxTime; }
```

المنارة للاستشارات
www.manaraa.com

```
 public void getInput()
 {

/****************************************************************************/
/*
Method getInput:

  This method receives command line file input to retrieve the size of the
MAS in number of agents, max time steps, initial number of agents that will
be alive after the MAS is instantiated (must be at least 1 agent greater
than the number of nearest neighbors), number of nearest neighbors, number
deceptive agents, weight of age attribute, weight of successful tasks
attribute, weight of basic trust attribute, weight of risk attribute,
trust update rate, learning rate, debug flag, prefix for naming the output
file to write to when debug flag is set to 'Y', flag to determine if fixed
inputs will be used, the input filename for fixed inputs for basic trust
risk, deceptiveness, and deception threshold for each agent.

Class variables used:

mas_Size
maxTime
numAlive
k_Nearest
numDeceptive
weightAge
weightSuccessful
weightBasicT
weightRisk
tuRate
lrnRate
debug
prefix
fixedInFlag
fixedInFile
fixedDeceptionFlag

*/
/****************************************************************************/


   //Variable Declarations

   String oneLine;                    //store input for processing
   StringTokenizer str;               //parse input
   BufferedReader in;                 //input buffer
   int index;                         //array index
```

المنارة للاستشارات
www.manaraa.com

```
//Method Execution

index = 0;
timeStep = 0;
inputVals = new String [16];
in = new BufferedReader(inFile);
try
{
 while( (oneLine = in.readLine()) != null)
 {
  str = new StringTokenizer(oneLine);
  str.nextToken();
  inputVals[index] = str.nextToken();
  index++;
 }
 inFile.close();
 mas_Size = Integer.parseInt(inputVals[0]);
 maxTime = Integer.parseInt(inputVals[1]);
 numAlive = Integer.parseInt(inputVals[2]);
 k_Nearest = Integer.parseInt(inputVals[3]);
 numDeceptive = Integer.parseInt(inputVals[4]);
 weightAge = Double.parseDouble(inputVals[5]);
 weightSuccessful = Double.parseDouble(inputVals[6]);
 weightBasicT = Double.parseDouble(inputVals[7]);
 weightRisk = Double.parseDouble(inputVals[8]);
 tuRate = Double.parseDouble(inputVals[9]);
 lrnRate = Integer.parseInt(inputVals[10]);
 debug = (inputVals[11].equals(String.valueOf('Y'))) ? true : false;
 prefix = inputVals[12];
 fixedInFlag = (inputVals[13].equals(String.valueOf('Y'))) ? true : false;;
 fixedInFile = new FileReader(inputVals[14]);
 fixedDeceptionFlag = (inputVals[15].equals(String.valueOf('Y'))) ? true : false;;
 outFile = new FileWriter(prefix + "init.txt");
 out = new FileWriter [mas_Size];
 if (debug)
  for (int i = 0; i < mas_Size; i++)
   out[i] = new FileWriter(prefix + "Agent" + i + ".txt");
 tallyF = new int [maxTime];
 numDecepAlive = new int [maxTime];
 failureRate = new double [maxTime];
 numAgentsAlive = new int [maxTime];
 for (int i = 0; i < maxTime; i++)
 {
  tallyF[i] = 0;
  numDecepAlive[i] = 0;
  failureRate[i] = 0.0;
  numAgentsAlive[i] = 0;
 }
}
catch (Exception e) {System.err.println(e);}
mas = new KmasAgent[mas_Size];
```

المنارة للاستشارات
www.manaraa.com

```
} //        End method getInput()



public void createMAS()
{

 /*****************************************************************************/
 /*
 Method createMAS:
   This method creates the array of KmasAgent objects according to the number of maximum agents
 specified in the input file.  Agents are then randomly selected to be alive (active in the system) using
 a random number generator.  Each agent is then given a copy of the array to allow it to reference the
 array objects that identify agents in the KMAS environment.
 */
 /*****************************************************************************/

 Random r = new Random();
 int aCount = 0;
 int a;

 for (int i = 0; i < mas_Size; i++)
   mas[i] = new KmasAgent(i, 0, k_Nearest, mas_Size, weightAge,
                   weightSuccessful, weightBasicT, weightRisk,
                   tuRate, lrnRate, fixedDeceptionFlag);

 while (aCount < numAlive)   //randomly make agents alive
 {
   a = Math.abs(r.nextInt()) %mas_Size;
   if (! mas[a].getAlive())
   {
     mas[a].setAlive(true);
     aCount++;
   }
 }

 for (int i = 0; i < mas_Size; i++)
   mas[i].setKmasAgentArray(mas);
} //       end method createMAS();



public void setDeceptiveAgents()
{

 /*****************************************************************************/
 /*
 Method setDeceptiveAgents:
   This method randomly selects agents instantiated in method createMAS, and makes them deceptive.
 If the Boolean variable fixedInFlag is set to true, method getFixedInput is called to store fixed input.
```

المنارة للاستشارات
www.manaraa.com

```
  Method printBeginningVals() is called to echo input.
  */
  /*****************************************************************************/

  Random r = new Random();
  int dCount = 0;
  int a;

  while (dCount < numDeceptive)       //randomly create deceptive agents
  {
   a = Math.abs(r.nextInt()) %mas_Size;
   if (mas[a].getDeceptive() != 1)
   {
    mas[a].setDeceptive(1);
    dCount++;
   }
  }
  if (fixedInFlag)
   this.getFixedInput();
  this.printBeginningVals();
 }  //        end method setDeceptiveAgents()



 public void getFixedInput()
 {

  /*****************************************************************************/
  /*
  Method getFixedInput:
   This method reads the contents of the fixed input file to first store basic trust and risk.  If fixed
  deception is chosen (fixedDeceptionFlag is true), the level of deception and deceptive threshold are
  stored for each deceptive agent.
  */
  /*****************************************************************************/

  String oneLine;                      //store input for processing
  StringTokenizer str;                  //parse input
  BufferedReader in;                    //input buffer
  int a = 0;                          //index for agent array

  in = new BufferedReader(fixedInFile);
  try
  {
   in = new BufferedReader(fixedInFile);
   while( (oneLine = in.readLine()) != null && a != 50)
   {
    str = new StringTokenizer(oneLine);
    mas[a].setBasicTrust(Double.parseDouble(str.nextToken()));
    mas[a].setRisk(Double.parseDouble(str.nextToken()));
    a++;
```

المنارة للاستشارات
www.manaraa.com

```
    }
    if (fixedDeceptionFlag)
      for (int i = 0; i < mas_Size; i++)
        if (mas[i].getDeceptive() == 1)
        {
          oneLine = in.readLine();
          str = new StringTokenizer(oneLine);
          mas[i].setDeception(Double.parseDouble(str.nextToken()));
          mas[i].setDeceptiveThreshold(Double.parseDouble(str.nextToken()));
        }
    fixedInFile.close();
  }
  catch (Exception e) {System.err.println(e);}
} //      end method getFixedInput()



public void executeMAS()
{

  /*************************************************************************/
  /*
  Method executeMAS:
    This method causes execution of one life cycle of the KMAS environment.  The variable timeStep
  is incremented at the beginning.  If the maximum number of allowable agents in the MAS has not
  been reached yet, an agent is randomly chosen to be made active (setAlive(true)) if this timeStep
  allows the addition of another agent.  Then, agents randomly select interaction partners up to the
  maximum number of achievable agent pairs based on the number of active agents in the environment.
  Cooperation is then initiated between the paired agents.  Cooperation results are recorded, and
  cooperation logs are updated if the experiment is in debug mode.
  */
  /*************************************************************************/

  int maxRequesters;
  Random a;
  int numR = 0;
  int i = 0;
  boolean keepLooking = true;

  a = new Random();
  maxRequesters = (int) Math.floor(numAlive/2);      //max # paired agents
  timeStep++;

  for (int j = 0; j < mas_Size; j++)
    mas[j].defaultCoopVars();
  if (numAlive < mas_Size)        //add new agents to execution if available
  {
    i = Math.abs(a.nextInt()) %4;   //random number between 0 and 3
    if (i == 0 || i == 2)         //add an agent
      while (keepLooking)
      {
```

المنارة للاستشارات
www.manaraa.com

```
        i = Math.abs(a.nextInt()) %mas_Size;
        if (! mas[i].getAlive())
        {
          mas[i].setAlive(true);
          numAlive++;
          keepLooking = false;
        }
      }
    }
    numAgentsAlive[timeStep - 1] = numAlive;
    for (int j = 0; j < mas_Size; j++)
      if (mas[j].getAlive())
        mas[j].setAge(mas[j].getAge() + 1);
    a = new Random();                //new Random number generator
    while(numR < maxRequesters)
    {
      i = Math.abs(a.nextInt()) %mas_Size;          //random partner
      if (! mas[i].getHasPartner() && mas[i].getAlive())
      {
        mas[i].findPartner();
        numR++;
      }
    }
    for (int j = 0; j < mas_Size; j++)
      if (mas[j].getAlive())
        mas[j].startCooperation();
    if (debug)
      this.printCooperationLog();
    this.tallyFailures();
    this.tallyDeceptiveAgents();
  }  //        end method executeMAS()



  public void tallyFailures()
  {

    /***************************************************************************/
    /*
    Method tallyFailures:
      This method requests the cooperation results of each agent pair, and accumulates the total number of
    failures (so far) for the experiment as a whole (for failure rate log), and the number of failures for the
    current time step.
    */
    /***************************************************************************/

    //tally if alive, non-deceptive, cooperating, not successful

    for (int j = 0; j < mas_Size; j++)
      if (mas[j].getAlive())
        if (mas[j].getDeceptive() == 0)
```

www.manaraa.com

```
        if (mas[j].getCooperate())
          if (! mas[j].getSuccess())
          {
            tallyF[timeStep - 1] += 1;
            totFailures++;
          }
   failureRate[timeStep - 1] = (double) totFailures/timeStep;
}  //        end method tallyFailures()



public void tallyDeceptiveAgents()
{

  /******************************************************************************/
  /*
  Method tallyDeceptiveAgents:
    This method records the number of deceptive agents active in the system at each time step.
  */
  /******************************************************************************/

  //tally alive, deceptive agents at timestep

  for (int j = 0; j < mas_Size; j++)
    if (mas[j].getAlive())
      if (mas[j].getDeceptive() == 1)
        numDecepAlive[timeStep - 1] +=1 ;
 }  //        end method tallyDeceptiveAgents()



public void printInput()
{

  /******************************************************************************/
  /*
  Method printInput:
    This method echoes initial input values from the file specified in the constructor.
  Contents are written to the filename specified by the class variable outFile.
  */
  /******************************************************************************/

  try
  {
    outFile.write("------------------------------------------------------------\r\n");
    outFile.write("KMAS INPUT PARAMETERS:\r\n\r\n");
    outFile.write("MAS_Size: " + mas_Size + "\r\n");
    outFile.write("TimeSteps: " + maxTime + "\r\n");
    outFile.write("Initially Alive: " + numAlive + "\r\n");
    outFile.write("NumK: " + k_Nearest + "\r\n");
```

www.manaraa.com

```
    outFile.write("NumDeceptive: " + numDeceptive + "\r\n");
    outFile.write("WeightAge: " + weightAge + "\r\n");
    outFile.write("WeightSuccessfulTasks: " + weightSuccessful + "\r\n");
    outFile.write("WeightBasicTrust: " + weightBasicT + "\r\n");
    outFile.write("WeightRisk: " + weightRisk + "\r\n");
    outFile.write("TrustUpdateRate: " + tuRate + "\r\n");
    outFile.write("LearningRate: " + lrnRate + "\r\n");
    outFile.write("Debug: " + debug + "\r\n");
    outFile.write("Outfile Prefix:   " + prefix + "\r\n");
    outFile.write("Fixed Input Flag:  " + fixedInFlag + "\r\n");
    outFile.write("Fixed Input File:  " + inputVals[14] + "\r\n");
    outFile.write("Fixed Deception Flag:  " + fixedDeceptionFlag + "\r\n");
    outFile.write("-------------------------------------------------------------\r\n");
    outFile.flush();
  }
  catch (Exception e) {System.err.println(e);}
} //           End method printInput()



public void printBeginningVals()
{

  /******************************************************************************/
  /*
  Method printBeginningVals:
    This method outputs the beginning agent values before start of execution.  All agent, active and
  Inactive, are shown.  Contents are written to the filename specified by the class variable outFile.
  */
  /******************************************************************************/

  try
  {
    outFile.write("BEGINNING AGENT VALUES:\r\n\r\n");
    for (int i = 0; i < mas_Size; i++)
    {
      outFile.write("Agent ID:  " + mas[i].getID() + "\r\n");
      outFile.write("Alive:     " + mas[i].getAlive() + "\r\n");
      outFile.write("Partner:   " + mas[i].getPartner() + "\r\n");
      outFile.write("Deceptive: " + mas[i].getDeceptive() + "\r\n");
      if (mas[i].getDeceptive() == 1)
      {
        outFile.write("Deception Level: "  + mas[i].getDeception() + "    ");
        outFile.write("Deception Threshold: " + mas[i].getDThreshold() + "\r\n");
      }
      outFile.write("Basic Trust: " + mas[i].getTrustB() + "\r\n");
      outFile.write("Risk: " + mas[i].getRisk() + "\r\n\r\n");
    }
    outFile.write("-------------------------------------------------------------\r\n");
    outFile.flush();
    outFile.close();
```

المنارة للاستشارات
www.manaraa.com

```
 }
 catch (Exception e) {System.err.println(e);}
} //        end method printBeginningVals()



public void printCooperationLog()
{

 /***************************************************************************/
 /*
 Method printCooperationLog:
   This method will output an execution trace of each agent if the experiment is in debug mode.
 The cooperation history of  each agent is written to a file that is unique to each agent.  The filename
 and path of this file is specified in the FileWriter object array out[].
 */
 /***************************************************************************/

 KmasAgent [] nArray;
 int p = 0;
 try
 {
  for (int i = 0; i < mas_Size; i++)
  {
   outFile = out[i];
   if (mas[i].getRequesterVal())
   {
    outFile.write("Time Step: " + timeStep + "     ");
    outFile.write("Agent Age: " + mas[i].getAge() + "\r\n");
    outFile.write("Alive:     " + mas[i].getAlive() + "\r\n");
    outFile.write("Agent ID:  " + i + "     ");
    outFile.write("Requester Agent: " + mas[i].getRequesterVal() + "\r\n");
    outFile.write("Total Successes: " + mas[i].getTaskS() + "    ");
    outFile.write("Total Failures:  " + mas[i].getTaskU() + "\r\n");
    outFile.write("Has Partner: " + mas[i].getHasPartner() + "\r\n");
    p = mas[i].getPartner();
    outFile.write("Partner ID: " + p + "\r\n");
    if (mas[i].getPerformK())
    {
     nArray = mas[i].getNeighbors();
     outFile.write("Nearest Neighbors:  ");
     for (int k = 1; k <= k_Nearest; k++)
       outFile.write(nArray[k].getID() + "  ");
     outFile.write("\r\n");
    }
    if (p != -1)
    {
     outFile.write("Basic Trust: " + mas[i].getTrustB() + "\r\n");
     outFile.write("Old General Trust In Partner: " + mas[i].getOldTrustG(p) + "\r\n");
     outFile.write("New General Trust In Partner: " + mas[i].getTrustG(p) + "\r\n");
     outFile.write("Situational Trust: " + mas[i].getTrustS() + "     ");
```

المنارة للاستشارات
www.manaraa.com

```
          outFile.write("Risk: " + mas[i].getRisk() + "\r\n");
          outFile.write("Will Cooperate: " + mas[i].getCooperate() + "\r\n");
          outFile.write("Success: " + mas[i].getSuccess() + "\r\n");
          outFile.write("Num Success with Partner: ");
          outFile.write(mas[i].getNumSuccesses(p) + "      ");
          outFile.write("Num Failure: ");
          outFile.write(mas[i].getNumFailures(p) + "\r\n");
        }
        outFile.write("**********************************************\r\n");
        outFile.flush();
      }
    }
  }
  catch (Exception e) {System.err.println(e);}
} //        end method printCooperationLog()



public void printFailures()
{

  /*****************************************************************************/
  /*
  Method printFailures:
    This method formats output that is written to the failure log specified by the variable 'prefix', and
  and a date/time stamp.  Number of failures for a given time step and the failure rate are displayed.
  */
  /*****************************************************************************/

  try
  {
    this.closeOutFiles();

    SimpleDateFormat sdf;
    sdf = new SimpleDateFormat("MMddyyHHmmss");
    String record;
    outFile = new FileWriter(prefix + "failures." + sdf.format(new Date()) + ".txt");
    outFile.write(inFileName + "\r\n\r\n");
    outFile.write("OUTPUT: " + prefix + "failures." + sdf.format(new Date()) + ".txt" +"\r\n\r\n");
    outFile.write("TIMESTEP,");
    outFile.write("NUM ALIVE,");
    outFile.write("NUM ALIVE DECEPTIVE,");
    outFile.write("FAILURES,");
    outFile.write("FAILURE RATE" + "\r\n");

    for (int t = 0; t < maxTime; t++)
    {
      /* outFile.write( t + " : " + tallyF[t] + "\r\n"); */

      record = Integer.toString(t);
      record = record + "," + numAgentsAlive[t];
```

المنارة للاستشارات
www.manaraa.com

```
      record = record + "," + numDecepAlive[t];
      record = record + "," + tallyF[t];
      record = record + "," + new Double(failureRate[t]).floatValue();
      outFile.write(record + "\r\n");
     }
     outFile.flush();
     outFile.close();
    }
    catch (Exception e) {System.err.println(e);}
   } //         end method printFailures()



   public void closeOutFiles()
   {

    /**************************************************************************/
    /*
    Method closeOutFiles:
     This method closes the output stream for the FileWrite object, outFile.
    */
    /**************************************************************************/

    if (debug)
    {
     try
     {
      for (int i = 0; i < mas_Size; i++)
      {
       outFile = out[i];
       outFile.close();
      }
     }
     catch (Exception e) {System.err.println(e);}
    }
   }  //        end method closeOutFiles()

  }  //         End class Kmas
```

المنارة للاستشارات

www.manaraa.com

APPENDIX H

class KmasAgent

```
import java.util.*;
import java.text.*;
import java.io.*;

public class KmasAgent
{

/****************************************************************************/
/*
Class KmasAgent:

Encapsulation of a single intelligent agent with the functionality needed to perform k-Nearest Neighbor,
store and update trust values for known interaction partners, find potential partners as an agent requesting
interaction, decide if cooperation with a selected partner is desired based on situation trust and risk,
determine the results of cooperation as being success or failure, and practice deception if the agent is a
deceptive agent.  Funcationality is also present to allow an agent to act as a selected partner.

1)  If deceptive, receive values for agent level of deception and deceptive threshold through fixed input or
random values.  The choice is based on the variable deceptFlag passed into the class constructor.

2)  If a requester agent (one who selects a partner and initiates interaction), class KMAS will direct the
agent (through KmasAgent methods) to find a potential partner through random selection.  Once the
partner is selected, the partner will be locked into an exclusive partnership and agent ID's will be
exchanged.

3)  Starts cooperation decision logic by using trustworthiness of selected interaction partners.  If partner is
unknown, or exploration is desired, perform k-Nearest Neighbor algorithm using Euclidean distance with
weighted variables age, successful tasks, basic trust, and risk to select k neighbors.

4)  Calculates situational trust to determine if cooperation is warranted.

5)  If cooperation is warranted, cooperate, and store the result of cooperating.  If the agent is an
interaction partner, defect if deception is greater than the deceptive threshold.

6)  If this instance of class KmasAgent is an agent requesting interaction (has selected a partner), this
class uses the result of cooperation to update general trust and cumulative totals of successful or non-
successful (failures) cooperation results as a whole, and also by the interaction partner involved in the
cooperative activity.

*/
/****************************************************************************/
```

www.manaraa.com

```
//class variable declaration

  /*  agent specific */
  boolean alive;              //is agent alive in the MAS?
  int age;                    //age of KMAS agent
  int taskS;                  //number of successfully completed tasks
  int taskU;                  //number of unsuccessful
  double trustB;              //basic trust
  double risk;                //risk threshold
  int agentID;                //numeric agent ID
  int deceptive;              //Is the agent deceptive? 0 (no), or 1 (yes)
  double deception;           //degree of agent deceptiveness
  boolean fixedDeceptionFlag;   //will deceptiveness be fixed or random
  double dThreshold;          //threshold of deception
  double tuRate;              //trust update rate
  int lrnRate;                //learning rate
  boolean explore;            //exploration flag
  Random rand;                //random number generator

  /*  partner specific */
  double trustS;              //situational trust in current partner
  double [] trustArray;       //array of general trust values by agentID
  double [] oldTrustArray;    //array of old general trust values by ID
  int [] taskSArray;          //array of total successful tasks by agentID
  int [] taskFArray;          //array of total failed tasks by agentID

  /*  k-Nearest neighbor specific */
  int numK;                   //value for k in k-nearest
  boolean performK;           //perform nearest neighbor?
  double wA;                  //weight for age attribute;
  double wS;                  //weight for taskS attribute;
  double wT;                  //weight for basic trust attribute;
  double wR;                  //weight for risk attribute;
  double [] edArray;          //agent ordered array of euclidean distances
  KmasAgent [] sArray;        //euclidean distance ordered array of agents

  /*  cooperation specific */
  KmasAgent [] agentArray;    //array of Kmas agents
  int currentPartner;         //current bound cooperation partner
  int numAgents;              //total number of agents in the MAS
  boolean cooperate;          //cooperation flag: Will cooperate with partner?
  public boolean hasPartner;  //Has partner been found?
  boolean success;            //Was cooperation successful?
  boolean requester;          //requesting cooperation?
```

المنارة للاستشارات
www.manaraa.com

```
//class constructor

 public KmasAgent(int id, int decpt, int k, int numA, double wtA,
              double wtS, double wtT, double wtR, double tuRt,
              int lrnRt, boolean deceptFlag)
 {
  agentID = id;
  deceptive = decpt;
  numK = k;
  numAgents = numA;
  wA = wtA;
  wS = wtS;
  wT = wtT;
  wR = wtR;
  tuRate = tuRt;
  lrnRate = lrnRt;
  fixedDeceptionFlag = deceptFlag;

  //initialize other variables

  rand = new Random();
  alive = false;
  age = 0;
  taskS = 0;
  taskU = 0;
  trustB = rand.nextDouble();
  performK = false;
  this.setPartner(-1);
  cooperate = false;
  rand = new Random ();
  risk = rand.nextDouble();
  trustArray = new double[numAgents];
  oldTrustArray = new double[numAgents];
  taskSArray = new int[numAgents];
  taskFArray = new int[numAgents];

  for (int i = 0; i < numAgents; i++)
  {
   trustArray[i] = 0.0;
   oldTrustArray[i] = 0.0;
   taskSArray[i] = 0;
   taskFArray[i] = 0;
  }
 } //        end constructor
```

المنارة للاستشارات
www.manaraa.com

```
//getter methods
  public int getID()   { return agentID; }
  public boolean getAlive() { return alive; }
  public int getAge()  { return age; }
  public int getDeceptive() { return deceptive; }
  public double getTrustS() { return trustS; }
  public int getTaskS()  { return taskS; }
  public int getTaskU()  { return taskU; }
  public double getTrustB()  { return trustB; }
  public double getRisk()  { return risk; }
  public double getDeception() { return deception; }
  public double getDThreshold() { return dThreshold; }
  public double getTrustG(int agentID) { return trustArray[agentID]; }
  public double getOldTrustG(int agentID) { return oldTrustArray[agentID]; }
  public boolean getHasPartner()  { return hasPartner; }
  public boolean getRequesterVal() { return requester; }
  public int getPartner() { return currentPartner; }
  public boolean getCooperate() { return cooperate; }
  public boolean getSuccess() { return success; }
  public int getNumSuccesses(int agentID) { return taskSArray[agentID]; }
  public int getNumFailures(int agentID) { return taskFArray[agentID]; }
  public boolean getPerformK() { return performK; }
  public KmasAgent [] getNeighbors() { return sArray; }
```

المنارة للاستشارات

www.manaraa.com

```
//setter methods

  public void setAge (int a) { age = a; }

  public void setDeceptive (int d)                    //give a random value for deception if deceptive
  {
    rand = new Random();
    deceptive = d;
    if (deceptive != 0)
      deception = rand.nextDouble();
  }

  public void setDeception(double d) { deception = d; }                    //used for fixed input deception

  public void setDeceptiveThreshold()                    //give a random deceptive threshold
  {
    rand = new Random();
    if (!fixedDeceptionFlag)
      dThreshold = rand.nextDouble();
  }

  public void setDeceptiveThreshold(double dThresh) { dThreshold = dThresh; }     //fixed threshold

  public void setBasicTrust(double bTrust) { trustB = bTrust; }

  public void setRisk(double r) { risk = r; }

  public void setAlive(boolean aVal) { alive = aVal; }

  public void setPartner(int partner)  { currentPartner = partner; }

  public void setCooperate(boolean cVal)  { cooperate = cVal; }

  public void setHasPartner(boolean hpVal)  { hasPartner = hpVal; }

  public void setSuccess(boolean sVal)  { success = sVal; }

  public void setRequester(boolean rVal)  { requester = rVal; }

  public void setKmasAgentArray(KmasAgent [] kA)  { agentArray = kA; }
```

المنارة للاستشارات

www.manaraa.com

```
//refresh methods

 public void defaultCoopVars()
 {
  //refresh cooperation variables to default values before start of cooperation

  this.setPartner(-1);
  this.setCooperate(false);
  this.setHasPartner(false);
  this.setSuccess(false);
  this.setRequester(false);
 }


 public void eraseMemory()
 {
  //erase general trust array, forcing agent to perform K-nearest
  // ******Not used currently

  trustArray = new double[numAgents];
  oldTrustArray = new double[numAgents];
 }



//K-neighbor methods

 public void createEDArray()
 {

  /***************************************************************************/
  /*
  Method createEDArray:
   This method creates an array of agent Euclidean distances, ordered by agent ID.
  */
  /***************************************************************************/

  // local variables
  double ageDiff = 0;
  double taskSDiff = 0;
  double trustBDiff = 0.0;
  double riskDiff = 0.0;
  double attributeSum = 0.0;
  double euclideanD = 0.0;
  edArray = new double[numAgents];    //refresh

  try
  {
   for (int i = 0; i < numAgents; i++)
   {
    ageDiff = wA * (age - agentArray[i].getAge());
```

المنارة للاستشارات
www.manaraa.com

```
      taskSDiff = wS * (taskS - agentArray[i].getTaskS());
      trustBDiff = wT * (trustB - agentArray[i].getTrustB());
      riskDiff = wR * (risk - agentArray[i].getRisk());
      attributeSum = (ageDiff * ageDiff) + (taskSDiff * taskSDiff)
               + (trustBDiff * trustBDiff) + (riskDiff * riskDiff);
      euclideanD = Math.sqrt(attributeSum);
      edArray[i] = euclideanD;
    }
  }
  catch (Exception e)
  {
    System.err.println(e);
    System.err.println("Error in KmasAgent method createEDArray()");
    this.dumpAgentVars();
  }
}  //end method createEDArray()



public void sortAgentsByED()
{

  /*************************************************************************/
  /*
  Method sortAgentsByED:
    This method creates an array of Kmas Agents, sorted by Euclidean distance.
  */
  /*************************************************************************/

  //local variables
  int s = 0;              //index for sorted array
  String [] pickL;          //pick list of agents to sort
  int edSmallest = -1;        //agent ID with smallest euclidean distance

  sArray = new KmasAgent[numAgents];  //refresh sorted agent list
  pickL = new String [numAgents];     //refresh pick list array
  for (int i = 0; i < numAgents; i++)
    pickL[i] = "Not Picked";
  try
  {
    while (s < numAgents)
    {
      for (int i = 0; i < numAgents; i++)
        if (pickL[i] != null)
        {
          if (edSmallest == -1)
            edSmallest = i;
          if (edArray[i] <= edArray[edSmallest])
            edSmallest = i;
        }
      sArray[s] = agentArray[edSmallest];
```

المنارة للاستشارات
www.manaraa.com

```
      pickL[edSmallest] = null;
      edSmallest = -1;
      s++;
    }
  }
  catch (Exception e)
  {
    System.err.println(e);
    System.err.println("Error in KmasAgent method sortAgentsByED()");
    this.dumpAgentVars();
  }
} //end method sortAgentsByED()



public void calcKTrust()
{

  /*****************************************************************************/
  /*
  Method calcKTrust:
    This method calculates the K-Nearest neighbor general trust estimate for an unknown agent, and
  a known agent during periods of exploration.
  */
  /*****************************************************************************/

  //local variables
  double kTrustG = 0.0;
  int nCount = 0;          //num of alive neighbor k contributors
  int kCount = 0;          //num of non zero K contributors

  try
  {
    for (int i = 1; i < numAgents; i++)  //i = 0 is agent performing k
      if (nCount < numK)
        if (sArray[i].getAlive())
        {
          kTrustG += sArray[i].getTrustG(currentPartner);
          if (sArray[i].getTrustG(currentPartner) != 0)
            kCount++;
          nCount++;
        }
    if (kTrustG == 0.0)        //agent unknown by k partners
      kTrustG = 0.5;          //take a chance if risk allows
    else
      kTrustG /= (double) kCount;
    if (!explore)
      trustArray[currentPartner] = kTrustG;
    else
      if (explore && kTrustG < trustArray[currentPartner])
        trustArray[currentPartner] = kTrustG;
```

المنارة للاستشارات
www.manaraa.com

```
    }
    catch (Exception e)
    {
      System.err.println(e);
      System.err.println("Error in KmasAgent method calcKTrust()");
      this.dumpAgentVars();
    }
  }  //end method calcKTrust()



//trust calculation methods

  public void calcSTrust()
  {
    /****************************************************************************/
    /*
    Method calcSTrust:
      This method calculates situational trust for a partner agent.  It is used by an agent that has selected
    a partner, and now wishes to engage in interaction.
    */
    /****************************************************************************/

    //local variables
    trustS = 0.0;              //situational trust

    try
    {
      trustS = trustB * trustArray[currentPartner];
    }
    catch (Exception e)
    {
      System.err.println(e);
      System.err.println("Error in KmasAgent method calcSTrust()");
      this.dumpAgentVars();
    }
  }  //end method calcSTrust()



  public void upDateTrust()
  {

    /****************************************************************************/
    /*
    Method upDateTrust()
      This method updates general trust in a partner after cooperation.  The update equation is based
    on the ratio of total number of cooperation successes with the current partner, divided by total number
    of cooperation attempts with the current partner based on all past time steps.  This ratio is then used
    to calculate a change in trust (deltaT).  The change in trust is used to update general trust.
```

www.manaraa.com

```
  */
  /*****************************************************************************/

  //local variables
  double ratioS;              //ratio of successful to total tasks
  double deltaT;              //difference of ratioS and old general trust


  try
  {
   if (taskSArray[currentPartner] + taskFArray[currentPartner] == 0)
     ratioS = 0;
   else
     ratioS = (double) taskSArray[currentPartner] / (taskSArray[currentPartner] +
                                 taskFArray[currentPartner]);
   deltaT = ratioS - trustArray[currentPartner];
   trustArray[currentPartner] += deltaT * (1 - deltaT) * tuRate;
   if (trustArray[currentPartner] > 1.0)
     trustArray[currentPartner] = 1.0;
   if (trustArray[currentPartner] < 0.0)
     trustArray[currentPartner] = 0.001;
  }
  catch (Exception e)
  {
   System.err.println(e);
   System.err.println("Error in KmasAgent method upDateTrust()");
   this.dumpAgentVars();
  }
 }  //end method upDateTrust()



//cooperation methods

 public void findPartner()
 {

  /*****************************************************************************/
  /*
  Method findPatner:
   This method allows an agent to select a potential interaction partner among the active agents
  in the system.  Partners are randomly selected.
  */
  /*****************************************************************************/
```

المنارة للاستشارات
www.manaraa.com

```
 //local variables
 Random partner = new Random();
 int i = 0;

 try
 {
  while (! hasPartner)
  {
   i = Math.abs(partner.nextInt()) %numAgents;
   if ( ! agentArray[i].getHasPartner() && i != agentID
                                   && agentArray[i].getAlive())
   {
    agentArray[i].setHasPartner(true);
    agentArray[i].setPartner(agentID);
    this.setHasPartner(true);
    this.setPartner(i);
    this.setRequester(true);
   }
  }
 }
 catch (Exception e)
 {
  System.err.println(e);
  System.err.println("Error in KmasAgent method findPartner()");
  this.dumpAgentVars();
 }
} //end method findPartner()



public void startCooperation()
{

 /******************************************************************************/
 /*
 Method startCooperation:
  This method allows a requester agent to decide whether or not to cooperate with a chosen
 interaction partner.  If this life cycle is a period of exploration, or if the partner is unknown,
 k-Nearest Neighbor is performed.  Situational trust is calculated, and the requester agent will
 cooperate if the situational trust is greater than risk.  The results of cooperation are retrieved (partner
 did or did not defect).  Arrays recording cooperation failures or successes are updated.  Trust is
 updated.
 */
 /******************************************************************************/

 performK = false;
 explore = false;
```

المنارة للاستشارات
www.manaraa.com

```
//start agent cooperation execution

if (!requester)
  return;
try
{
  oldTrustArray[currentPartner] = trustArray[currentPartner];
  if(trustArray[currentPartner] == 0.0)        //unknown partner
  {
    performK = true;
    this.createEDArray();
    this.sortAgentsByED();
    this.calcKTrust();
    oldTrustArray[currentPartner] = trustArray[currentPartner];
  }
  else                                         //exploration for known
    if (lrnRate != 0 && (age % lrnRate) == 0)
    {
      explore = true;
      performK = true;
      this.createEDArray();
      this.sortAgentsByED();
      this.calcKTrust();
      oldTrustArray[currentPartner] = trustArray[currentPartner];
    }

  this.calcSTrust();
  if (this.willCooperate())
  {
    this.setCooperate(true);
    this.getCoopResult();
    if (success)
    {
      taskSArray[currentPartner] += 1;
      taskS++;
    }
    else
    {
      taskFArray[currentPartner] += 1;
      taskU++;
    }
    this.upDateTrust();
  }
}
catch (Exception e)
{
  System.err.println(e);
  System.err.println("Error in KmasAgent method startCooperation()");
  this.dumpAgentVars();
}
} //end method startCooperation()
```

المنارة للاستشارات

www.manaraa.com

```
public boolean willCooperate()
{
 //requester agent will cooperate if situational trust is greater than risk

 if (trustS >= risk)
   return true;
 else
   return false;
}  //end method willCooperate()



public void getCoopResult()
{
 //retrieve cooperation result from current partner, results in success or failure

 this.setSuccess(agentArray[currentPartner].returnCoopResult());
}  //end method getCoopResult()



public boolean returnCoopResult()
{

 /*******************************************************************************/
 /*
 Method returnCoopResult:
   This method returns the cooperation result to the caller.  During execution, the caller is a
 requester agent using this method to see whether or not a selected partner will cooperate
 successfully or defect.  The functionality of this method is executed by the selected partner.
 */
 /*******************************************************************************/

 //return cooperation result to requester agent
 //boolean result for caller success variable

 this.setDeceptiveThreshold();
 if (deception > dThreshold && deceptive ==1)
   return false;
 else
   return true;
}  //end method returnCoopResult()
```

www.manaraa.com

```
public void dumpAgentVars()
{

 /*************************************************************************/
 /*
 Method dumpAgentVars:
   This method dumps agent variables in case of expections to an output file.  The output file is
 named using a combination of "DMPA", agentID, and date/time stamp.
 */
 /*************************************************************************/

 FileWriter out;
 SimpleDateFormat sdf;

 sdf = new SimpleDateFormat("MMddyyHHmmss");
 try
 {
  out = new FileWriter("DMPA" + agentID + "_" + sdf.format(new Date()) + ".txt");
  out.write("id: " + agentID + "\r\n");
  out.write("age: " + age + "\r\n");
  out.write("MAS size: " + numAgents + "\r\n");
  out.write("K value: " + numK + "\r\n");
  out.write("successes: " + taskS + "\r\n");
  out.write("failures: " + taskU + "\r\n");
  out.write("basic trust: " + trustB + "\r\n");
  out.write("risk: " + risk + "\r\n");
  out.write("deceptive: " + deceptive + "\r\n");
  out.write("deception: " + deception + "\r\n");
  out.write("dThreshold: " + dThreshold + "\r\n");
  out.write("has partner? : " + hasPartner + "\r\n");
  out.write("partner: " + currentPartner + "\r\n");
  out.write("situational trust: " + trustS + "\r\n");
  out.write("requester agent? : " + requester + "\r\n");
  out.write("perform K-N? : " + performK + "\r\n");
  out.write("cooperate? : " + cooperate + "\r\n");
  out.write("cooperation successfull? : " + success + "\r\n");
  out.write("\r\n");
  out.write("-------------------------------------------------------------\r\n");
  out.write("Agent Array:");
  out.write("\r\n");
  out.write("contents: " + agentArray + "\r\n");
  out.write("\r\n");
  for (int i = 0; i < numAgents; i++)
   out.write("agentArray[" + i + "] : " + agentArray[i].getID() + "\r\n");
  out.write("-------------------------------------------------------------\r\n");
  out.write("Trust Array:");
  out.write("\r\n");
  out.write("contents: " + trustArray + "\r\n");
  out.write("\r\n");
  for (int i = 0; i < numAgents; i++)
  out.write("trustArray[" + i + "] : " + trustArray[i] + "\r\n");
  out.write("-------------------------------------------------------------\r\n");
```

المنارة للاستشارات
www.manaraa.com

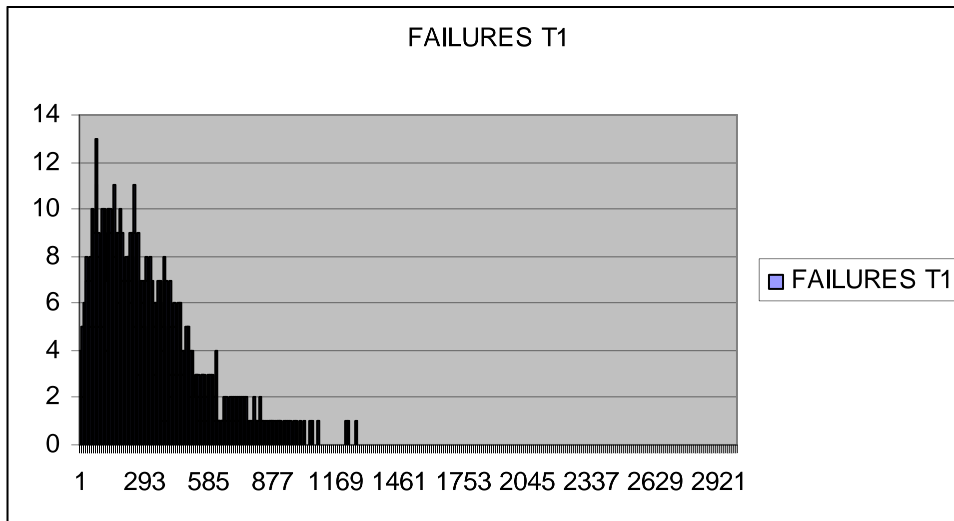
```
      out.write("Old Trust Array:");
      out.write("\r\n");
      out.write("contents: " + oldTrustArray + "\r\n");
      out.write("\r\n");
      for (int i = 0; i < numAgents; i++)
      out.write("oldTrustArray[" + i + "] : " + oldTrustArray[i] + "\r\n");
      out.write("--------------------------------------------------------\r\n");
      out.write("Successful Tasks Array:");
      out.write("\r\n");
      out.write("contents: " + taskSArray + "\r\n");
      out.write("\r\n");
      for (int i = 0; i < numAgents; i++)
        out.write("taskSArray[" + i + "] : " + taskSArray[i] + "\r\n");
      out.write("--------------------------------------------------------\r\n");
      out.write("Failed Tasks Array:");
      out.write("\r\n");
      out.write("contents: " + taskFArray + "\r\n");
      out.write("\r\n");
      for (int i = 0; i < numAgents; i++)
        out.write("taskFArray[" + i + "] : " + taskFArray[i] + "\r\n");
      out.write("--------------------------------------------------------\r\n");
      out.write("Euclidean Distance Array:");
      out.write("\r\n");
      out.write("contents: " + edArray + "\r\n");
      out.write("\r\n");
      for (int i = 0; i < numAgents; i++)
        out.write("edArray[" + i + "] : " + edArray[i] + "\r\n");
      out.write("--------------------------------------------------------\r\n");
      out.write("Sorted Neighbor Array:");
      out.write("\r\n");
      out.write("contents: " + sArray + "\r\n");
      out.write("\r\n");
      for (int i = 0; i < numAgents; i++)
        out.write("sArray[" + i + "] : " + sArray[i].getID() + "\r\n");
      out.flush();
      out.close();
    }
    catch (Exception e)
    {
      System.err.println(e);
      System.err.println("Error in Agent " + agentID + " dump");
    }
  }  // end method dumpAgentVars()


} //           End Class KmasAgent
```

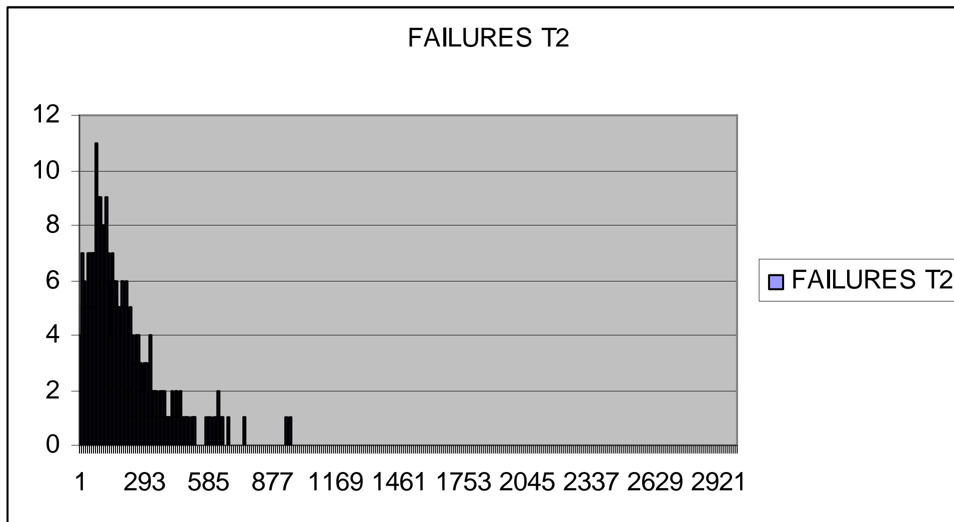
المنارة للاستشارات
www.manaraa.com

APPENDIX I

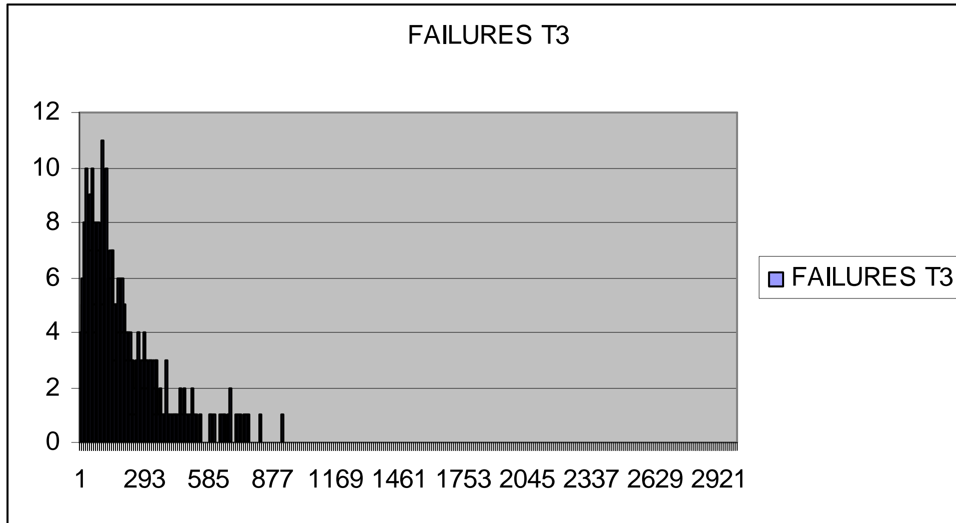
Experiment: 1 Failures over time

EXPERIMENT 1 GROUP A



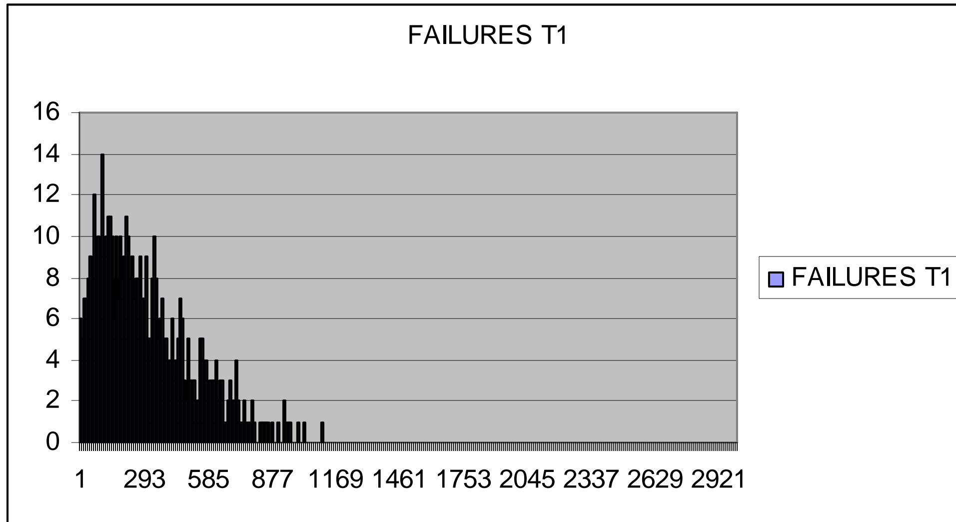


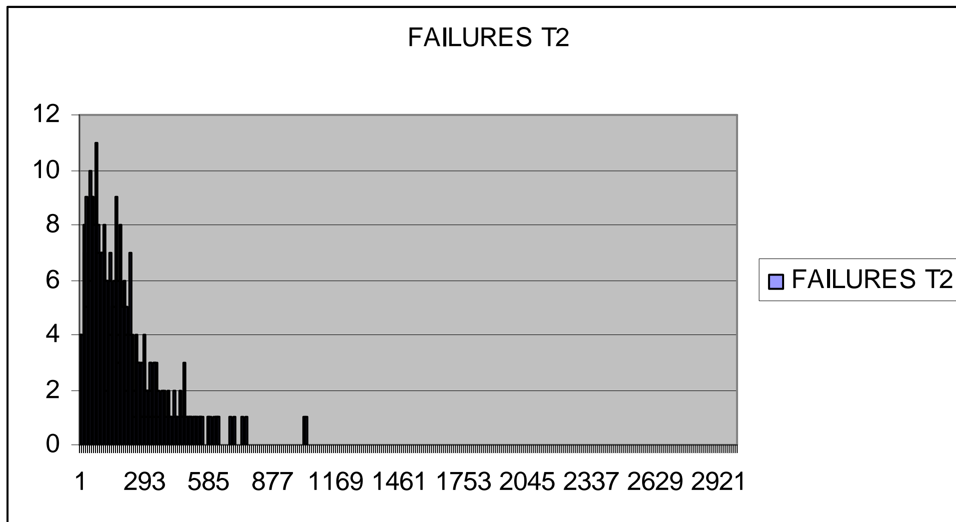
المنارة للاستشارات
www.manaraa.com



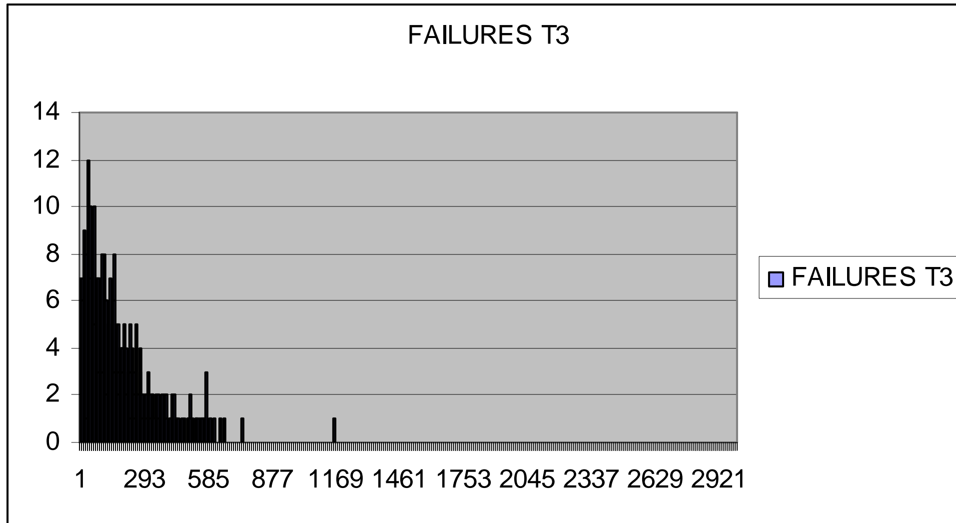
www.manaraa.com

EXPERIMENT 1 GROUP B



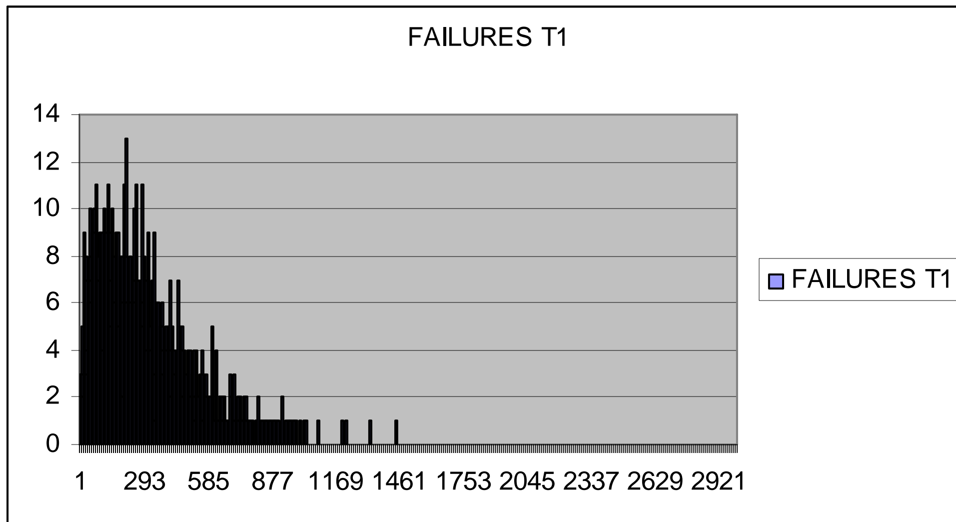


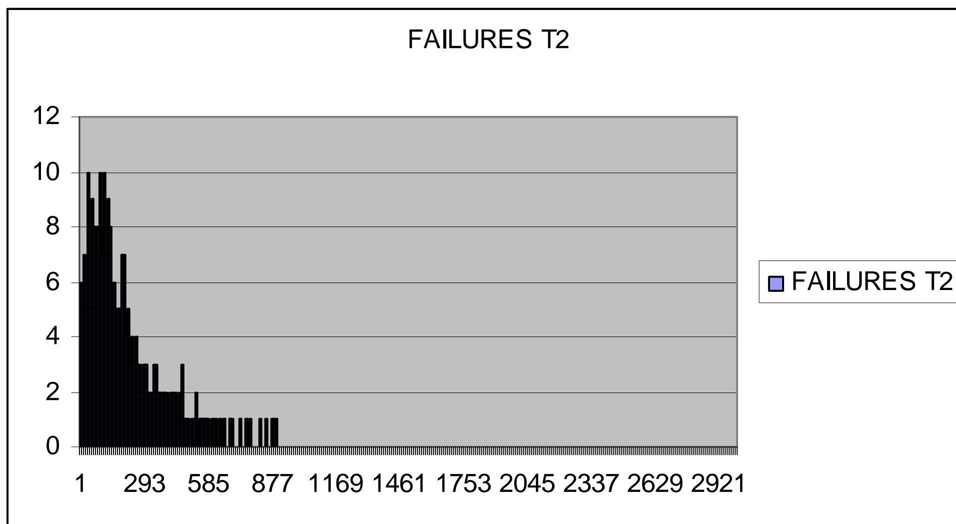
المنارة للاستشارات
www.manaraa.com



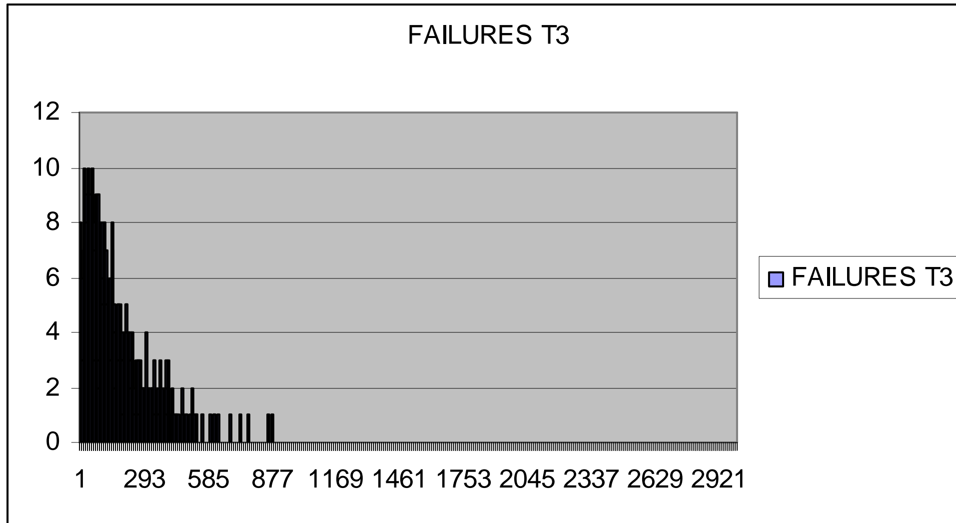
www.manaraa.com

EXPERIMENT 1 GROUP C





المنارة للاستشارات
www.manaraa.com

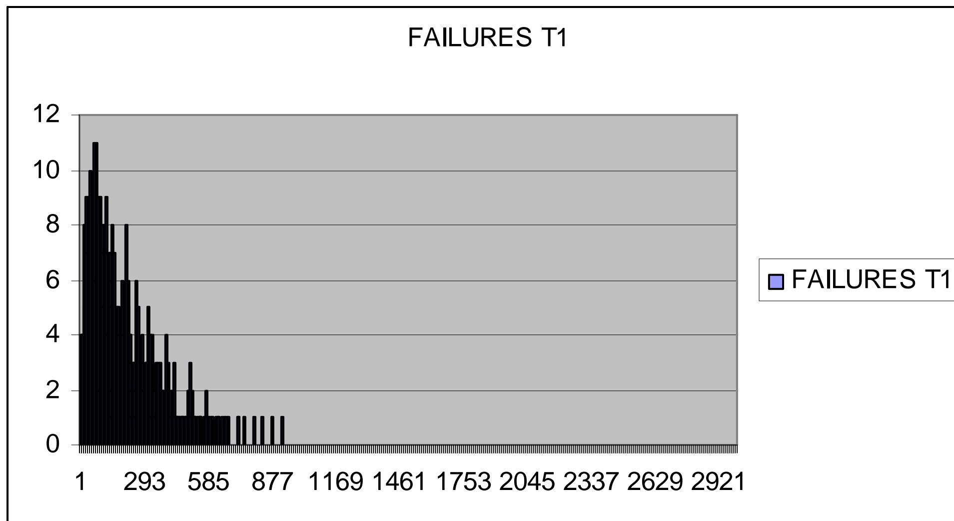


المنارة للاستشارات
www.manaraa.com

APPENDIX J

Experiment: 2 Failures over time

EXPERIMENT 2 GROUP A



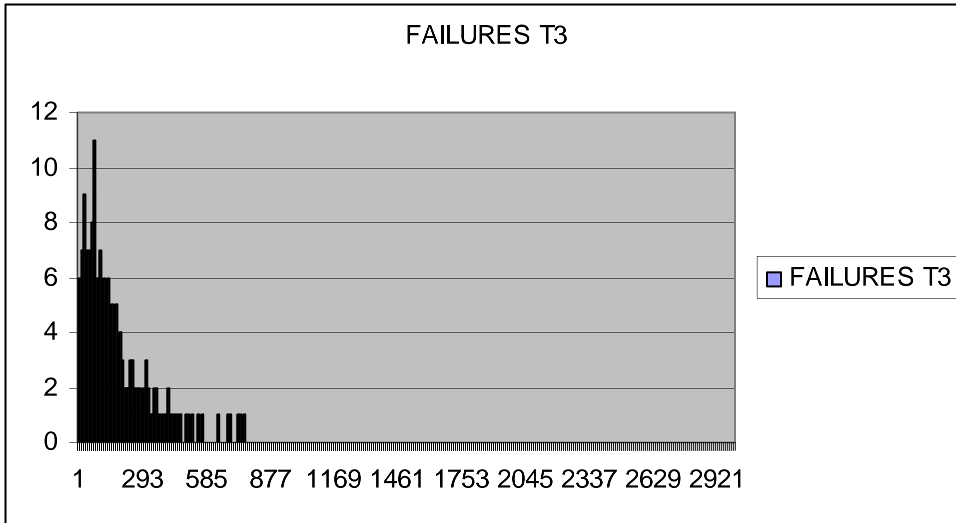
FAILURES T2

12
10
8
6
4
2
0

1 293 585 877 1169 1461 1753 2045 2337 2629 2921

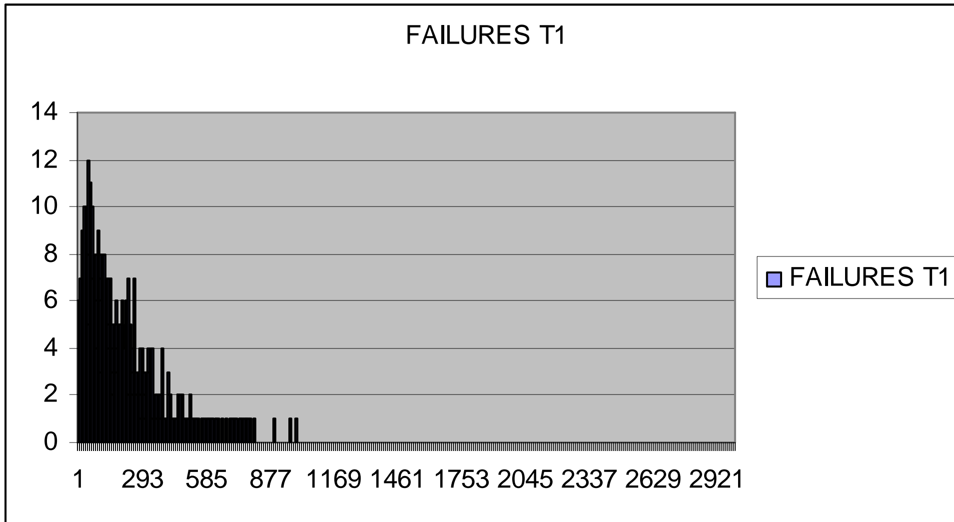
FAILURES T2

المنارة للاستشارات
www.manaraa.com



المنارة للاستشارات
www.manaraa.com

EXPERIMENT 2 GROUP B

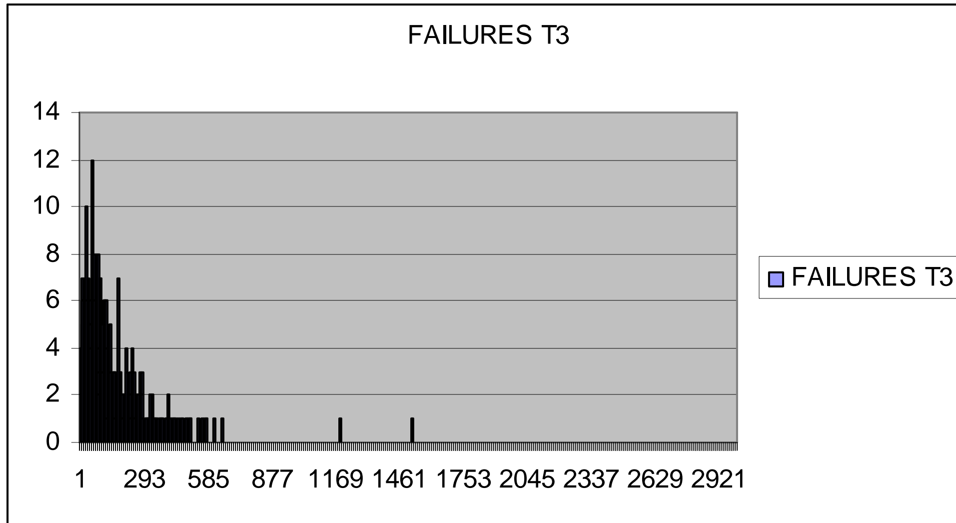


FAILURES T2

12
10
8
6
4
2
0
1 293 585 877 1169 1461 1753 2045 2337 2629 2921

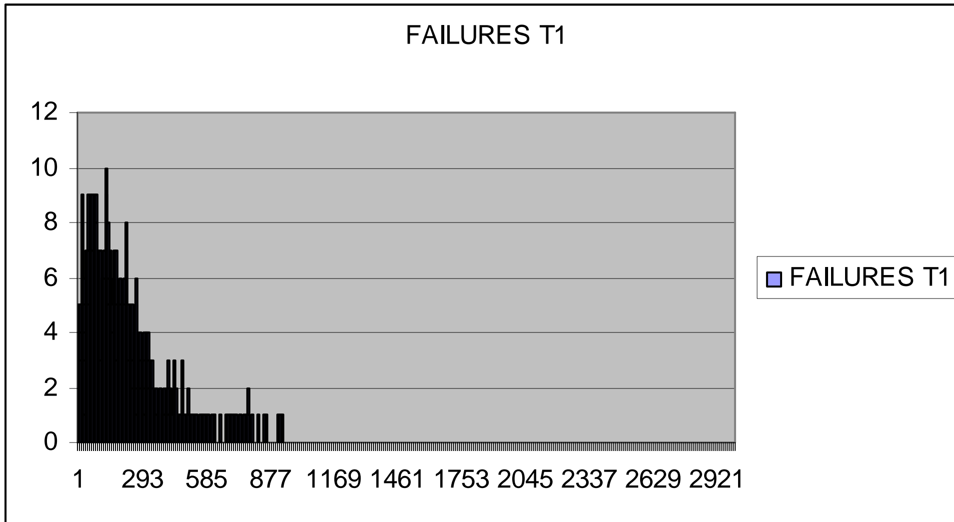
FAILURES T2

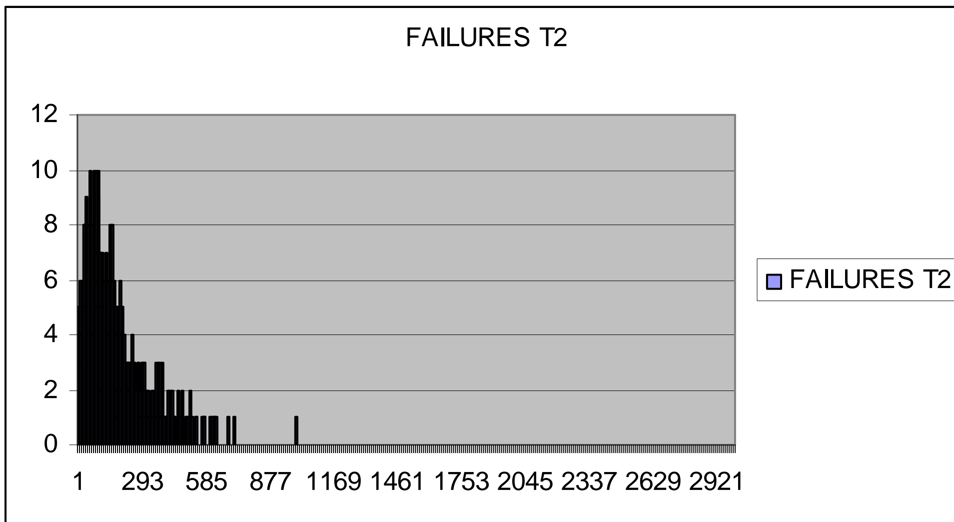
المنارة للاستشارات
www.manaraa.com



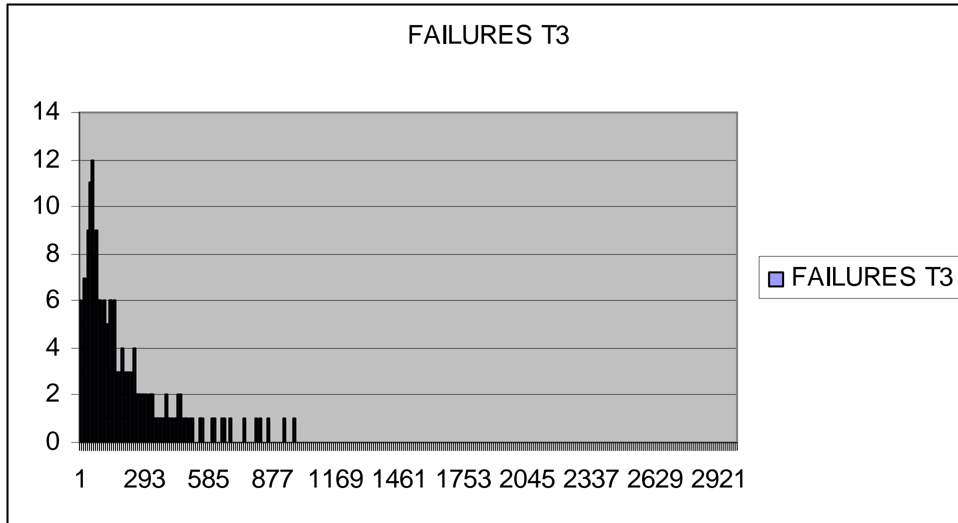
www.manaraa.com

EXPERIMENT 2 GROUP C





المنارة للاستشارات
www.manaraa.com

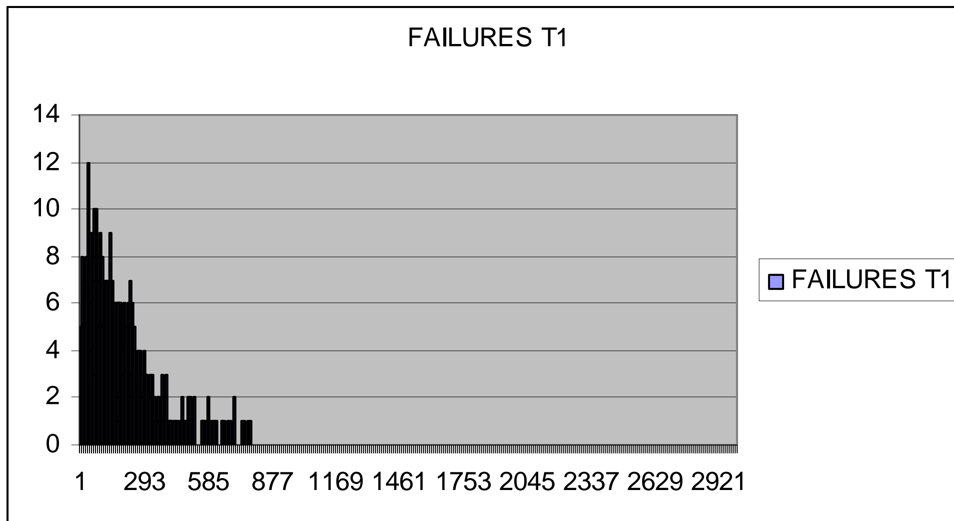


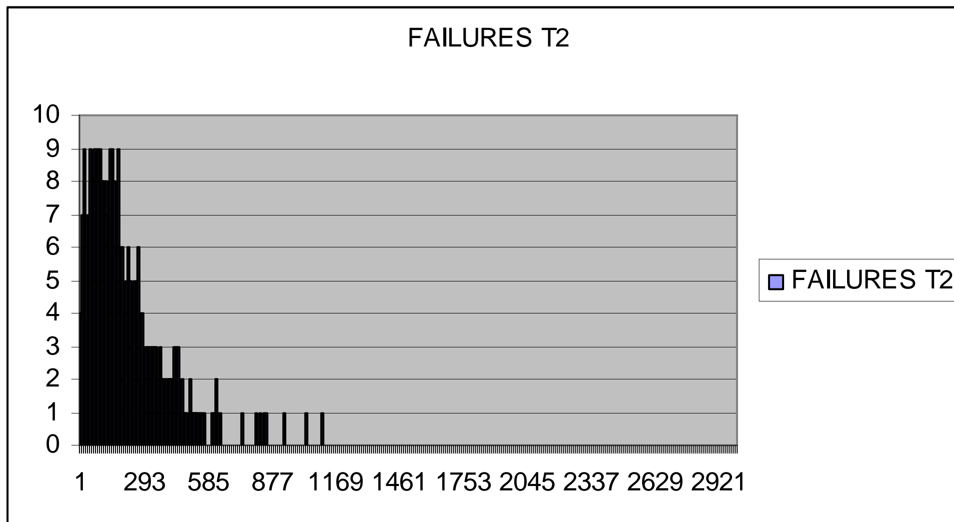
www.manaraa.com

APPENDIX K

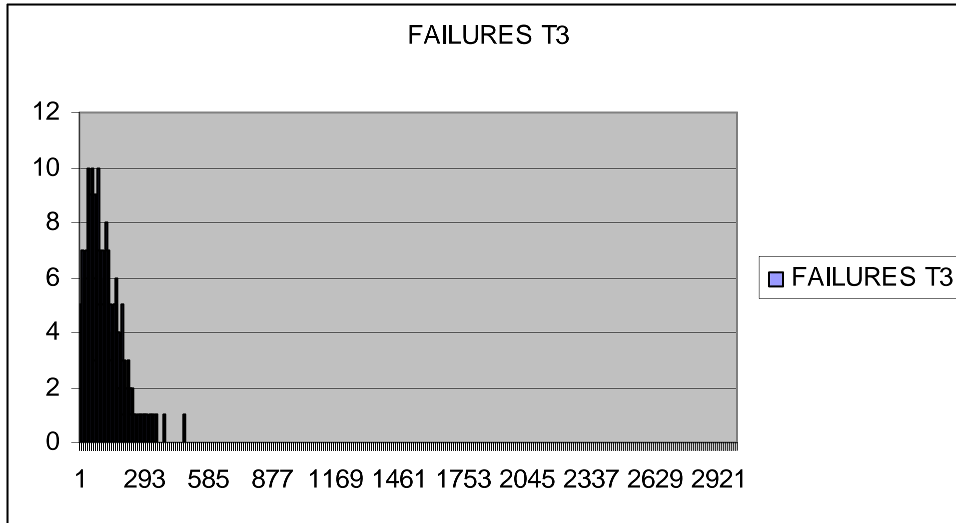
Experiment: 3 Failures over time

EXPERIMENT 3 GROUP A



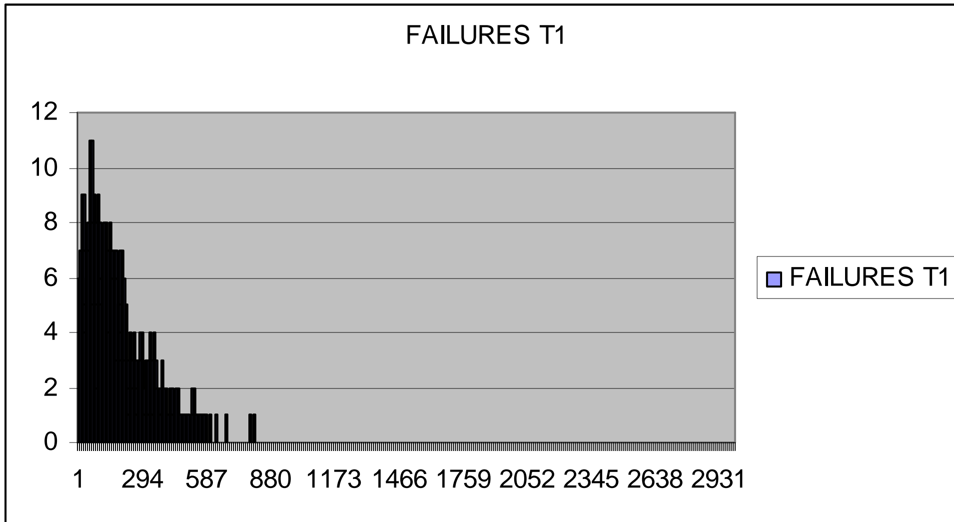


المنارة للاستشارات
www.manaraa.com



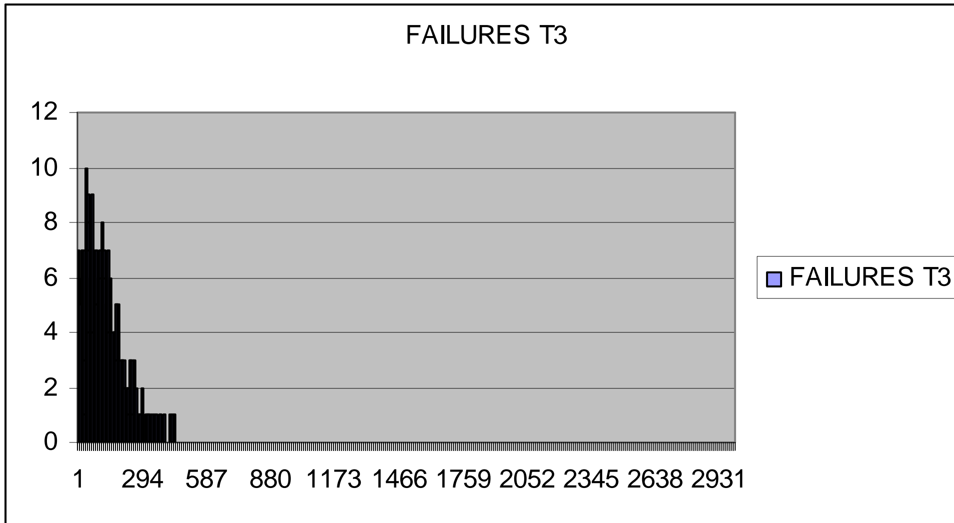
www.manaraa.com

EXPERIMENT 3 GROUP B



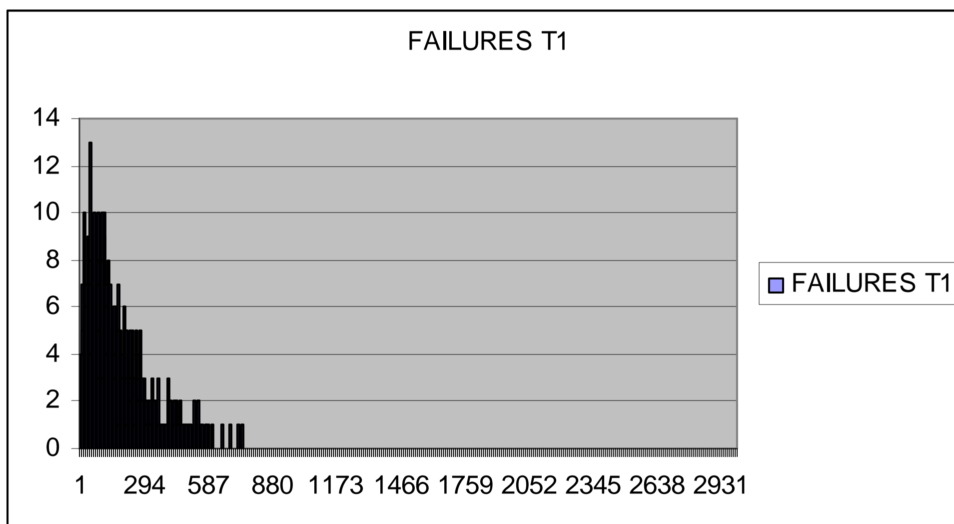
FAILURES T2
12
10
8
6
4
2
0
1 294 587 880 1173 1466 1759 2052 2345 2638 2931
FAILURES T2

المنارة للاستشارات
www.manaraa.com



www.manaraa.com

EXPERIMENT 3 GROUP C

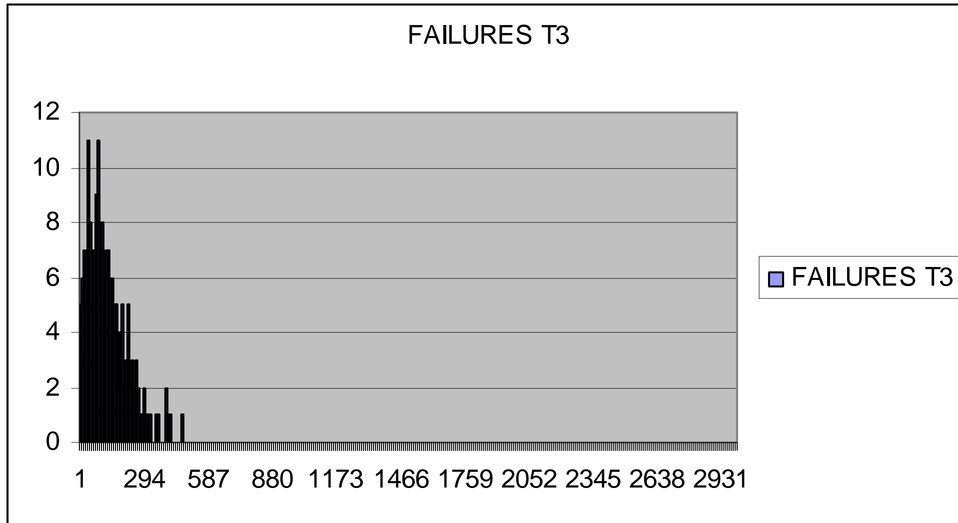


FAILURES T2

12
10
8
6
4
2
0
1 294 587 880 1173 1466 1759 2052 2345 2638 2931

FAILURES T2

المنارة للاستشارات
www.manaraa.com



المنارة للاستشارات
www.manaraa.com