Dynamic Task Allocation in Mobile Robot Systems using Utility Functions

by

Scott A. Vander Weide
B.S. University of Central Florida, 2000

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the School of Electrical and Computer Engineering
in the College of Engineering & Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2008

Major Professor:
Ladislau L. Bölöni

Abstract

We define a novel algorithm based on utility functions for dynamically allocating tasks to mobile robots in a multi-robot system. The algorithm attempts to maximize the performance of the mobile robot while minimizing inter-robot communications. The algorithm takes into consideration the proximity of the mobile robot to the task, the priority of the task, the capability required by the task, the capabilities of the mobile robot, and the rarity of the capability within the population of mobile robots.

We evaluate the proposed algorithm in a simulation study and compare it to alternative approaches, including the contract net protocol, an approach based on the knapsack problem, and random task selection. We find that our algorithm outperforms the alternatives in most metrics measured including percent of tasks complete, distance traveled per completed task, fairness of execution, number of communications, and utility achieved.

Acknowledgments

The author would like to thank Douglas Richter, of SAIC, and Joseph Swinski, of DiSTI, for the tuition assistance they have so generously provided through out this journey. The author would also like to thank Linus Luotsinen and Majid Khan for the work they performed on the YAES simulation testbed on which the simulations for this thesis were derived. Finally, the author would like to thank his wife, Janet, his parents, Fred and Candy, and his close friends Eddie Coleman, Mike Kosiba, Jim McBrayer, and John Tungseth for all their prayers and support during the writing of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

A squadron of stealthy unmanned aerial vehicles (UAV) are performing a perimeter flight plan above a battlefield. A task is transmitted to the squadron requesting close-air support for a Marine unit that has just come under hostile fire. Without giving any indication of their position through radio communications a UAV must break formation and perform a surgical strike on the enemy position. Which UAV will go? It should be the one most capable of reaching the target in time and most capable of performing the strike. How do the UAVs decide which one is best suited to go without communicating with each other? The focus of this work is to answer that question for this situation and others like it, which require multiple mobile robots in a given area to efficiently perform tasks with little or no inter-robot communications.

This work introduces the use of utility functions to determine what task a given mobile robot attempts to perform. The utility function is based on multiple criteria of a mobile robot and a task in the system, which include proximity of the robot to the task, the priority of the task, the suitability of the robot to perform the task, and the rarity of the capabilities required to complete the task.

A task is an action requiring a specific capability, or set of capabilities, to be performed at a given location. Tasks have a priority, or importance, assigned to them. A task can be generated at any time. A task is no longer active once a mobile robot completes it.

A mobile robot is an autonomous vehicle capable of receiving task information, processing the information, and acting on it without further external stimulation [GMV04]. The mobile robot can be a UAV, an unmanned ground vehicle (UGV), an agent in a simulation, or similar entity. A mobile robot can have one or more capabilities to sense and/or interact with its environment, which can be very common or extremely rare. Mobile robots move at a given speed, can enter the operating area at any time, and have finite energy resources.

The ideas presented in this thesis are evaluated using multi-agent system simulation software. The terms mobile robot and agent, therefore, may be used interchangeably in the rest of this work. They are compared to several other schemes for task allocation including random selection and contract net protocol. In the random selection scheme a mobile robot randomly selects a task to perform from the list of available tasks. It performs the task until completion then randomly selects a new task. The contract net protocol scheme provides a way for agents to negotiate what agent will perform a given task based on the cost of each agent to perform the task [Smi80].

The remainder of this thesis is organized as follows. Chapter 2 is a discussion of works related to task allocation in multi-agent systems. This includes the discussion of task allocations schemes used for comparison to the one presented in this work. It also discusses literature on utility functions and methods for evaluating mobile robot performance. Chapter

3 details the implementation of the utility function-based dynamic task allocation. Chapter 4 discusses the software developed for the simulations that evaluate the dynamic task allocation scheme and their results. Chapter 5 discusses the simulation experiments performed to evaluate the dynamic task allocation scheme. Chapter 6 provides a summary of the task allocation scheme that is presented in this work. Chapter 7 discusses ways to expand the dynamic task allocation scheme, what other considerations and additions may be made, and other future work in this research area.

CHAPTER 2

LITERATURE REVIEW

In this chapter we review works related to the topics of systems of multiple mobile robots and task allocation, task allocation schemes, and ways to evaluate these systems.

2.1 Utility Functions

In this section we review some of the papers that discuss utility functions and their uses in domains similar to that of this thesis.

In [Mon97] Mongin defines the theory of Expected Utility. The theory states that decisions are made by comparing the weighted sums of the utility of outcomes and their probabilities.

In [Gar06] Garces discusses the use of weighted sums for artificial intelligence (AI) engines within games. These weighted sums produce an expected utility for a given situation. Each term of the sum is normalized and is assigned a weight, or importance. The result of the sum is a value in the range [0, 1]. The terms of the sum are dependent on the given situation.

The weights are independent of the given situation and can be cumbersome to determine. Garces proposes several "extensions" to the calculation of expected utility.

The first extension is to define an inertia term for the weighted sums. This additional term provides stabilization when the given situation changes. If the AI first pursues one goal and a second with a higher expected utility enters the situation the AI will immediately switch. The AI could get stuck between the two goals and achieve neither. The inertia term keeps the AI from switching goals when the expected utility of one is only slightly higher than the expected utility of the other.

The second extension is to model one, or more of the terms for the weighted sums as sigmoid functions. This provides a response similar to that of a human, instead of linearly. The sigmoid function has the added bonus of being bounded in the range (0, 1). The sigmoid function is defined in equation 2.1.

$$f(x) = \frac{1}{1 + e^{-2\frac{x-m}{s}}} \tag{2.1}$$

For example, if you have $1 in your wallet and are asked to run an errand for $10 you are more likely to accept the offer than if you had $1000 in your wallet. The expected utility for running the errand is much greater in the situation where you have only $1.

Garces also suggests using functions linearly interpolated along a given set of points. This allows for a well defined curve for the output of the term.

The final extension is to add "clear-cut rules." These rules force the expected utility to one extreme or another based on certain occurrences in a given situation. An example would be if the AI did not have the capabilities to perform the goal the expected utility would be set to zero, regardless of the weighted sums of the terms for the situation.

In [Woo02] Wooldridge discusses the use of utility functions in tasking agents. A real number in the range [0, 1] is assigned as the utility for each state in an environment. The agent is responsible for maximizing its utility to produce the best outcome with a task. "The higher the utility the better."

Wooldridge also discusses the use of utility functions in determining agent preferences. A utility function assigns a real number to each potential outcome of an environment. The higher the number the more preferred the outcome.

2.2 Contract Net Protocol

In this section we review works describing the function of and recent extension to the contract net protocol, which is used for comparison to the algorithm presented in this thesis.

In [Smi80] Smith discusses the development of the contract net protocol, a protocol for the assignment of tasks among nodes in a system. Nodes are distributed logically and physically. A node can take on the role handing out tasks, executing tasks, or performing both roles

simultaneously. Nodes perform a contract negotiation to determine what node performs a given task.

The process for handling a task occurs as follows. A node with a task to be performed announces it to the other nodes. The announcement specifies what is required to complete the task and when the bids must be received. Any node not currently processing another task before the end of the "expiration time" sends a bid to the managing node. The managing node selects a winner and notifies that node to begin processing the task. Finally, the contractor node returns the results of the task to the managing node.

Smith discusses several speed ups and enhancements to the basic contract net protocol. Immediate response bids require a node to respond, even if it is busy with another task. The response may be a standard bid or one of three other types: busy, ineligible, of low ranking. These additional response types give feedback to the managing node to allow for adjustments to the task requirements if no bids are received. Directed contracts allow a managing node to assign a task to a contracting node that is known to be capable of executing the task. There is a mechanism for request-response messages for information gathering. And, there is a mechanism for idle contracting nodes to announce their availability to managing nodes. The managing nodes may then attempt to match a task to the capabilities of the contracting node.

In [San93] Sandholm formalizes the processes for bidding in and awarding tasks with the contract net protocol. He discusses the use of calculating marginal costs in determining the bid for a task and selection of an agent to perform the task. He also discusses the grouping

of interrelated tasks into blocks of tasks for bidding. He then verifies this implementation by running simulations in a vehicle routing application.

In [San98] Sandholm discusses the reallocation of tasks between agents using the contract net protocol. He then discusses extending the contract net protocol with four different types of contracts. The contracts are cluster, swaps, multi-agent, and the original single task. Once the contract types are defined a combined contract type is introduced taking advantage of all four of the described contracts.

The marginal cost of a task to a given agent is the difference between the costs of performing the task itself versus contracting it to another agent. If the cost to perform the task is more than the cost to contract it out the agent will reallocate the task to another agent.

Cluster contracts are contracts that group together a set of tasks and allocate them for a single cost. This is useful when tasks are interdependent, i.e. it would cost more to break up the tasks among multiple agents than it would for one agent to perform all the tasks.

Swap contracts are useful when the cost of two tasks with two agents is lower if the agents were to perform each others tasks.

Multi agent contracts are an extension of swap contracts where more than two agents swap tasks to lower the cost to perform all tasks involved.

The combination of all four contract types into one contract avoids many of the local optima, i.e. it more easily enables the global optima to be achieved.

In [KSF02] the authors discuss an extension to the Foundation for Intelligent Physical Agents (FIPA) contract net protocol specification by adding a confirmation message from the contractor agent to the managing agent once the task has been awarded. The FIPA contract net protocol is itself and extension of the original protocol in which rejection and confirmation messages are added. The confirmation message is used to minimize the over allocation of resources by an agent before it has received a task. These resources would normally be allocated when the bid is sent to the managing agent.

The process works as follows. A managing agent requests bids from contractor agents for a given task. The contractor agents send bids into the managing agents but to not yet allocate resources. The managing agent ranks the bids and selects a winner. The winning contractor agent is notified. The contractor agent then has the option to reject or confirm the task to the managing agent. If it accepts it will allocate resources to perform the task. If it rejects the task the managing agent will notify the next contractor agent in the ranked list of bids.

2.3 Instances of the Knapsack Problem

The knapsack problem is a combinatorial optimization problem that is NP-Hard (nondeterministic polynomial-time hard) [Pis95]. The best known example is that of packing a knapsack with the highest value of items with a given total weight. The problem has many variations. The 0-1 knapsack problem limits the packing of only zero or one of each item.

The bounded knapsack problem allows multiple packings of a given item. The problem is bounded by the fact each item has a weight and so only a certain number of an item, up to the total allowed weight, can be packed. The multiple knapsack problem allows the packing into one or more knapsacks. The knapsacks may have the same or different total weight allowances. There are many other variations, including combinations of variations.

2.4 Task Allocation

The main focus of this thesis is the allocation of task in a multi-agent system. This section discusses others' works in defining and comparing task allocation schemes.

In [CFK97] Cao et al performs a survey of current works in systems of mobile robots. Two points brought up in this paper differentiating multi-robot systems from similar research areas are the multi-robot systems are more capable than individual robots because individual robots are "spatially limited", and multi-robot systems operate in real world environments, which are more difficult to model than the environments of other distributed systems.

In [GMV04] Gage discusses using emotions to recruit robots to perform tasks. He sets up several characteristics of multi-robot teams: unreliable communications, including bandwidth limitations and periodic failures; dynamic team composition, including heterogeneousness and addition/removal of robots; distributed control; task reassignment; and inability to predict future tasking. He proposes using the emotion shame as a model for motivat-

ing robots to help out. A robot sends out a request for help in the form of a new task to be performed. When a robot receives this request it increments its "shame" parameter. The increment value could be based on availability, proximity, ability to complete the task, etc. Gage chooses proximity. Once the "shame" reaches a given threshold the robot accepts the task. The "shame" parameter decays over time. Gage also performs a qualitative comparison of the affective method of task allocation to other methods, including auction, motivation-based, mutual inhibition, team consensus, and no allocation.

In [Kra97] Kraus describes multidisciplinary approaches to achieving cooperation in multi-entity fields, specifically multi agent systems. The paper states that cooperation may improve performance in these systems. Other multi-entity fields may not work to implement multi agent systems but do cover a wide variety of issues important to agent design. There are multiple techniques to choose from for implementing agents. The technique selection is based on several parameters, which are: level of agent cooperation, do the agents have the same goal or are they attempting to maximize individual output; is there an agreed upon protocol for behavior and communication; how many agents the system contains; whether or not the agents need to interact with humans; and the communications and computational limits of the agents.

The paper discusses four fields of study which may be adapted to agent development and gives examples for each. The fields are game theory, classical mechanics, operational research techniques, and behavioral and social sciences.

Game theory is best suited for systems of self-motivated agents attempting to maximize personal benefits. There is an agreed upon protocol. There are few agents (less than 10). The systems do not involve human interaction. The computational power an agent is not likely to be limited. The field deals with the study of conflict and cooperation between people. It involves abstract representations of real life situations. It is the basis for agent interaction protocols. Implementations require substantial computational power.

Classical mechanics may involve hundreds, or more, agents. It is best suited for large sets of joint, common, goals. There is an agreed upon protocol. Direct communications between agents may be too costly or impossible. An example of a system involving classical mechanics is modeling particle dynamics.

In operations research techniques all agents cooperate for a common goal. There are defined protocols. There are few, if any, limitations on communications and computational power. With this technique efficiency decreases with size. An example of the use of this technique is a set partitioning problem.

Behavioral and social sciences techniques are best suited for agents interacting in non-standard environments where the agents are self-motivated and may be required to interact with humans.

In [SSG06] the authors discuss scenarios where multiple UAVs work together to perform search and destroy tasks, where a group of UAVs must "search a region and destroy as many targets as possible within the flight endurance time of the UAVs." In their scenarios UAVs have limited sensor and communications ranges. The authors propose a negotiation scheme

be used between UAVs to determine which one is assigned a given task. They compare this scheme, in two variations, with greedy task assignment. The two variations are with and without target information exchange.

The authors define a mode of operation for an agent. It must perform a search or attack task for every time step. There are four situations that cause an agent to make a decision as to what task it should perform. They are: "No targets and neighbors," which causes the agent to search and move in the same direction; "No targets but has neighbors," which allows the agent to search or attack based on information gathered by the neighbors by negotiating; "Targets are present but no neighbors," which causes the agent to attack the highest valued target; and "Target as well as neighbors present," which allows the agent to search or attack based on information gathered by the neighbors by negotiating.

The negotiation scheme utilized is similar to Rubinstein's model of strategic negotiation. Agents make proposals. These proposals are either accepted or rejected by the other agents within this agent's communications range. This agent performs its proposal depending on what the other agents do. "A coordinated decision by an agent would be one that is not in conflict with the decision of its neighbors. There is no conflict except that which arises due to uncertainty of agent actions."

In [Das06] the authors implement a scheme to use multiple UAVs with limited image processing capabilities to identify and confirm targets in a battlefield scenario. The UAVs are deployed in a given area and each forages the battlefield. The UAVs use ant-like pheromone marking behavior to identify potential targets. Then, other UAVs within communications

range attempt to route to these marked areas to bolster or negate the identification of a target.

In [SS06] the authors discuss two schemes for territory exploration without inter-agent communication. The territory exploration task requires that a set of checkpoints in an environment be visited by n agents simultaneously, $n = 2$ for the experiments in this paper. The two schemes are the social preference mechanism and the pairing mechanism.

The authors started with a set of "timeout agents." These agents perform a random walk in search of check points. Once found, an agent waits a predefined amount of time for another agent to arrive at the check point, allowing the checkpoint to be cleared. If no second agent arrives before the timeout then that agent continues its random walk. An agent will always move to the closest checkpoint.

The authors then move to the social preference mechanism. Instead of moving to the closest checkpoint the agent now moves to the closest checkpoint with another agent waiting at it. This minimizes the number of agents waiting for another agent to allow a checkpoint to be cleared.

The final implementation is the pairing mechanism. Here, agents form pairs so that two agents simultaneously reach a checkpoint, clearing it without any wait time. An agent starts the task by searching for other agents and ignoring checkpoints. Once paired, one agent is the leader and the other a follower. The leader then searches for checkpoints.

In [WBH06] the authors develop a multi agent system consisting of Automatic Guided Vehicles (AGV) to perform load transportation tasks within an industrial environment, a warehouse. The system uses field gradients to direct AGVs to tasks. Each task produces an attractive force on the AGV while competing AGVs repulse each other. An AGV follows the gradient of the summation of all the forces acting on it.

The authors performed tests in an AGV simulator using real maps and parameters of industrial environments. The metrics measured when evaluating the system were average wait time for a task to be executed, throughput (number of tasks executed), and number of messages transmitted. The field-based task assignment scheme is compared to the contract net protocol scheme.

In [Gri05] the author introduces Multi-Dimensional Trust (MDT) as a solution to the problem of mitigating risk of failure and/or poor performance with cooperative agents in a robust system. Autonomous agents may choose to cooperate, at what level of effort to perform a cooperative task, and when/if to rescind an agreement to cooperate. The trust is used as a factor when deciding how to "appropriate cooperative partners." The author's domain for application of this trust is a set of heterogeneous, self-interested, agents with specific tasks that, at times, requires cooperation with other agents, of different capabilities, to complete.

Multi-Dimensional Trust evaluates the trustworthiness of an agent based on multiple distinct values, which are used as weights in calculating the decision to assign an agent a

task. The trust values are not based on based on similar situations, but on general evaluations of past performance.

The paper models trust based on these dimensions: success, cost, timeliness, and quality. Success is the likelihood the agent will complete the task. Cost is the likelihood the cost "will be no more than expected," i.e. the executing agent will not blow its budget. Timeliness is the likelihood the agent will finish within the time it said it would. Quality is the ability of the agent to meet expectations.

In [AL05] the authors explore the use of mediation in assigning tasks, and sub-tasks, to different agents of a multi-agent system. Mediation consists of decomposing a task into sub-tasks, finding suitable agents to execute the sub-tasks, and negotiating with those agents to get the task completed. Th focus is to provide an expansion of the Semi Markov Decision Process that utilizes the actions the mediator may use that change from time to time.

In [OVM05] the authors describe a system of robots that perform a search and rescue mission within interior environments. The system does not have a priori knowledge of the layout of the environment. The mission of the system is to map the environment, find an object of interest, then establish a perimeter to protect the object of interest.

Once the system has defined the environment, during the mapping phase, a set of tasks are defined. These tasks implement the second and third parts of the mission, search, and protection. A distributed dispatcher doles out these tasks. A robot informs the dispatcher it needs tasking. The dispatcher responds in one of two ways. It may assign a task to the robot based on the robot's position and battery level. Or, it may send the robot a list of

tasks and have the robot bid on the tasks. The robot ranks the list of tasks based on its position and battery level. The rankings are submitted to the dispatcher, which then assigns tasks to the robots.

2.5 Evaluation Criteria

In this section we review works on evaluating task allocation schemes, what metrics to calculate, and what are preferred characteristics of simulation environments.

In [DJ03] the authors discuss the comparison of multi-agent systems that use dynamic resource allocation. They propose the evaluation of systems based on general qualitative criteria and problem space specific criteria. They have identified a way of classifying architectures based on the level of coordination between individual agents and how synchronous the agents' interactions are. They perform testing with specific agents within an intelligent networks simulation.

The authors identify nine attributes for comparing the performance of different agent architectures operating in the same type of system. Reactivity is how long it takes for an architecture to respond to a request. Load-balancing is how well tasks are distributed among agents. Fairness is a measure of how fair each task is treated within the system. Utilization of resources is a measure of how active resources are kept and whether all resources available are being used. Responsiveness is how long it takes for an agent to return a result for a

given task. Communications overhead is amount of time required to send and receive tasks among agents and clients. Robustness is a measure of how well an architecture handles agent failures and communications problems. Modifiability is an abstract quality of how easy an architecture's implementation is to change once initial development is completed. Scalability is a measure of how well an architecture handles increased demand or load.

In [Nij04] the author expands on the work presented in [DJ03] by comparing the contract net protocol, coalition formation, and socially responsible agent architectures. He reviews the criteria presented by Davidsson and Johansson. He then explains these criteria within the context of multi-agent task allocation. Reactivity is the time it takes, from task creation, for a task to be assigned to an agent. Load balance is the measure of how equitable the assignment of tasks to agents is. Fairness is a measure of how evenly tasks are treated. The author also dismisses responsiveness as not being agent architecture dependent. The reactivity and communications overhead are part of the responsiveness. Once the two attributes are removed all that is left is a measure of how well a resource handles task execution. The resources are independent of the architecture used to allocate tasks.

In [LEF06] the authors present a comparison of twelve agent control schemes within a simulated environment. The environment mimics a biological system and is titled the Feed-Flee-Multiply game. It is represented by a two-dimensional map with open spaces and obstacles. Food randomly appears on the map. The agents are to move about the map, find food, multiply, and, if necessary, fight. This environment has been chosen to allow "multiple

paths to success" so that no one scheme should have an advantage or disadvantage compared to the others.

Multiple criteria for the success of an agent have been defined. They include amount of food gathered, number of agents killed, total number of agents of a given type surviving, etc.

CHAPTER 3

TASK ALLOCATION ALGORITHM

This chapter discusses the algorithm developed for dynamically assigning tasks to mobile robots within a scenario.

There are many ways to determine which mobile robot in a group is assigned a given task. A user or central application may choose the task assignments for each mobile robot. The mobile robots may collaborate amongst themselves to determine task allocation. The mobile robots may select tasks autonomously. Autonomous task allocation may be performed with or without the knowledge of the other mobile robots in the group.

We have developed an algorithm that involves each mobile robot computing its own utility function for each task it has received from a central task broadcasting service. The utility function is comprised of four components: proximity, priority, suitability, and rarity. The task broadcasting service sends tasks to all mobile robots in the scenario and mobile robots communicate back notification of task completion.

3.1 Problem Formulation

A task is an action requiring a specific capability, or set of capabilities, to be performed at a given location. Tasks have a priority, or importance, assigned to them, can be generated at any time, and are no longer present in the system once completed. Equation 3.1 defines the task mathematically.

$$t = t < x_t, y_t, c_t, p_t > \tag{3.1}$$

A mobile robot is an autonomous vehicle capable of receiving task information, processing the information, and acting on it without further external stimulation [GMV04]. The mobile robot can be a UAV, an unmanned ground vehicle (UGV), an agent in a simulation, or similar entity. It can have one or more capabilities to sense and/or interact with its environment, which can be very common or extremely rare. They move at a given speed and have finite energy sources. Equation 3.2 defines the mobile robot mathematically.

$$a = a < x_a, y_a, e_a, \{c_a^1, c_a^2, ...\} > \tag{3.2}$$

3.2 Computing Utility

Expected Utility theory states that decisions are made by comparing the weighted sums of the utility of outcomes, which are normalized, and there probabilities [Mon97]. The closer the result of the utility function is to one for a given task the more likely the mobile robot will choose to execute it. Our utility function is divided into four components. It is presented in equation 3.3.

$$U(a,t) = \begin{cases} w_d \frac{1}{1+e^{\frac{-2}{s} \times \left[D_{max} - \sqrt{(x_a - x_t)^2 + (y_a - y_t)^2} - m\right]}} + w_p \frac{1}{p_t} + w_s S_{a_c,t_c} + w_r R_{a_c} & S_{a_c,t_c} \neq -1 \\ 0 & S_{a_c,t_c} = -1 \end{cases} \tag{3.3}$$

The various pieces of the utility function are:

- $U(a,t)$ is the utility of a to perform t.
- D_{max} is the maximum distance a mobile robot will respond to a task.
- m, s are parameters of the sigmoid.
- S_{a_c,t_c} is the suitability of a to perform t, based on capabilities.
- R_{a_c} is the rarity of the capability of a.
- w_d, w_p, w_s, and w_r are weights to be determined.

The sum of the weights w_d, w_s, w_p, and w_r is 1, as illustrated in equation 3.4.

$$w_d + w_p + w_s + w_r = 1 \tag{3.4}$$

See section 5.4 for a discussion on the selection of the utility weights' values.

3.2.1 Proximity

The closer a mobile robot is to a given task the more likely it is to perform that task. This is formalized as the proximity component. The proximity is calculated using the euclidean distance formula, in two-dimensions.

There is a physical range limit for a mobile robot, D_{max}, the maximum distance an agent should respond to a task. This value may be the agent's communications range or its sensor range. This value may also be the agent's travel range based on current energy level. The values d and D_{max} are used to calculate the proximity utility function, illustrated in equation 3.5.

$$D(d) = \frac{1}{1 + e^{\frac{-2}{s} \times \left[(D_{max} - \sqrt{(x_a - x_t)^2 + (y_a - y_t)^2}) - m \right]}} \tag{3.5}$$

The sigmoid function will always provide a value in the range [0, 1]. The value of m affects the center point of the sigmoid, the point at which the sigmoid returns the value 0.5, along the x-axis. Increasing m moves the center closer to zero while decreasing m moves the center closer to D_{max}. The value of s affects the rate at which the value returned by

the sigmoid increases or decreases. The greater the value of s the more vertical the center portion of the sigmoid is. See section 5.5 for a discussion on the selection of the sigmoid function and the values of m and s and Appendix C for graphs used in determining the values of m and s. Figure 3.1 illustrates the sigmoid chosen for the proximity component.

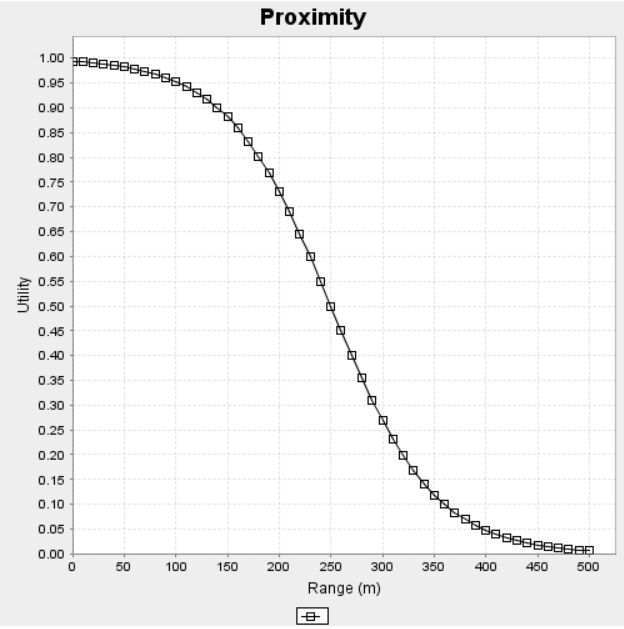

Figure 3.1: Proximity Sigmoid Graph

3.2.2 Priority

The priority of a task conveys the importance of a given task relative to other tasks in the system. It is a value from one to five, p, where one is the highest priority. Mobile robots attempt to perform tasks with higher priorities before those with lower priorities. The priority function is illustrated in equation 3.6.

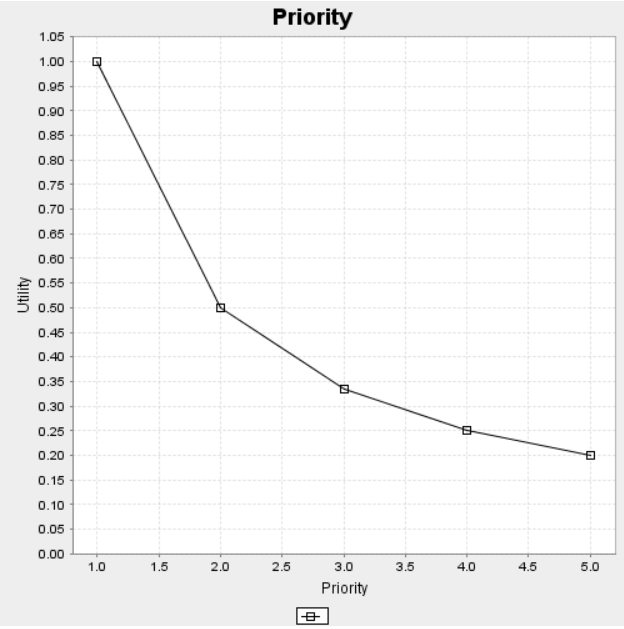

Figure 3.2: Priority Graph

$$P(t) = \frac{1}{p_t} \tag{3.6}$$

The function computes the inverse of the priority value. For a priority of one the priority function results in one. As the priority decreases, p increases, the result of the priority function diminishes linearly. But, since the priority function never results in zero there is always an opportunity for a task with a low priority to be executed. Figure 3.2 displays a graph of the priority equation.

3.2.3 Suitability

Every mobile robot has a set of capabilities derived from installed sensors, weapons, etc. A task requires one type of capability. The suitability of a mobile robot to perform a task is how well the mobile robot's capabilities match the capability required by the task. There are some limited equivalencies between capabilities. For instance, a task requiring missiles can also be accomplished with a bomb, though not necessarily as well. However, the same task can not be accomplished with a synthetic aperture radar. There are also types of capabilities that can not be replaced with any alternative, such as the radiation sensor.

The suitability matrix, presented in table 3.1, is a formal representation of these equivalencies. The suitability values are in the range [-1, 1]. If the capability required by the task matches exactly the capability of the mobile robot then the suitability is one. As the capabilities of the task and mobile robot diverge the suitability diminishes. If the mobile robot is completely incapable of performing the task, i.e. the mobile robot's capability is completely incompatible with the required capability of the task, the suitability is -1. This value is necessary to prevent the mobile robot from attempting to perform tasks it is incapable of performing. Further explanation of the use of the -1 suitability is discussed in section 3.2.5. Note that the matrix is not symmetrical, some capabilities are inherently better at performing certain functions than others. For example, electro-optical and infra-red cameras both produce human interpretable imagery, but infra-red cameras can sense through clouds while electro-optical cameras cannot.

Table 3.1: Suitability Matrix

	EO Camera	IR Camera	SA Radar	Chemical Sensor	Radiation Sensor	Bomb	Missile
EO Camera	1	0.25	0.25	-1	-1	-1	-1
IR Camera	0.5y	1	0.5	-1	-1	-1	-1
SA Radar	0.5	0.5	1	-1	-1	-1	-1
Chemical Sensor	-1	-1	-1	1	-1	-1	-1
Radiation Sensor	-1	-1	-1	-1	1	-1	-1
Bomb	-1	-1	-1	-1	-1	1	0.25
Missile	-1	-1	-1	-1	-1	0.25	1

EO = Electro-optical, IR = Infra-red, SA = Synthetic Aperture

3.2.4 Rarity

A mobile robot with a rare capability should be used sparingly within a scenario to preserve it for tasks requiring its capability. The rarity of a capability is determined by the probability of that capability not being available on a mobile robot. The probabilities were calculated from a survey of data published on currently fielded Unmanned Aerial Vehicles by the United States Armed Forces and other United States government agencies, including the Department of Homeland Security and the National Aeronautics and Space Administration.

See Appendix B for the compiled data and probability calculations for the capabilities. Table 3.2 lists the capabilities we are interested in and their relative rarities.

Table 3.2: Rarity Probabilities

Capability	Rarity
EO Camera	0.0000
IR Camera	0.0135
SA Radar	0.9291
Chemical Sensor	0.9865
Radiation Sensor	0.9865
Bomb	0.5439
Missile	0.9662

EO = Electro-optical, IR = Infra-red, SA = Synthetic Aperture

3.2.5 Rule-based Adjustments

In [Gar06] the author discusses extending the effectiveness of utility functions by "clear-cut rules." These rules force the expected utility to one extreme or another based on certain occurrences in a given situation.

The rule for handling a negative suitability value is simple. If the suitability of a mobile robot to perform a task is -1 then the utility of that mobile robot to perform that task is zero regardless of the values of the other components of the utility function.

3.3 Utility Function Implementation

The utility function mobile robot receives and stores all broadcast tasks. Every time the mobile robot enters its execution method the following steps occur. The mobile robot checks its current location to the location of its current task. If they are the same the task is considered completed. A message is broadcast to all mobile robots and the task broadcaster. The mobile robot recalculates its utility to execute each of the tasks in its list. The task with the highest calculated utility is selected to be executed. This may preempt the task that is currently being executed. Only tasks with suitabilities greater than zero are considered for execution. The mobile robot then moves toward its current task. If there is no task to execute, because the mobile robot is not suitable to execute any available tasks or there are no tasks currently in the scenario, the mobile robot will not move during this execution cycle. Algorithm 3.1 illustrates the utility function task selection algorithm.

Algorithm 3.1: Utility Function Algorithm

```
if new task received then
    Store task in task_list

if energy > 0 then

    if currentTask location = mobile robot location then
        Stop moving

        Save task completion time and current mobile robot energy level (final energy)

        Remove currentTask from task_list

        Broadcast task complete message for currentTask

        has task = false
    foreach task in task_list do

        if suitability of task ≠ -1 then
            Compute task utility
    Sort task_list by computed utility, highest to lowest

    previousTask = currentTask

    if task_list length > 0 and utility of task_list[0] > 0 then
        currentTask = task_list[0]

        if currentTask ≠ previousTask then
            Save task start time and current mobile robot energy level (initial energy)

        has task = true
    if has task = true then
        Move toward currentTask one speed unit
```

CHAPTER 4

SIMULATION IMPLEMENTATION

This chapter discusses the development of the simulation software created to evaluate the algorithm presented in Chapter 3. The simulation software is divided into applications for generating scenario and configuration files and simulation execution, and is developed in Java.

4.1 Scenario Generation

The scenario generation application creates multiple sets of scenario files and associated simulation configuration files. A scenario contains a group of mobile robots, with specified capabilities, operating in a specific area to perform certain tasks with specified required capabilities and priorities. Each set of scenarios includes 50 individual scenario files created randomly with the same set of initial conditions. The configuration file specifies the type of mobile robots to run in the simulation, the functions to use when calculating the utility, utility weights, and other environmental parameters. The scenario generation algorithm is outlined in algorithm 4.1.

Algorithm 4.1: Scenario Generation Algorithm

foreach *number of tasks* **do**

foreach *number of mobile robots* **do**

for $0 \leq i \leq 50$ **do**

Create sorted Map of strings, *steps*, indexed by start time

for $0 \leq j \leq$ *number of tasks* **do**

create random start *time*, *location*, *task capability*

if *capability* = ***ANY*** *or* ***MULTI*** **then**

create random *priority*

else

priority = 1

Write string containing *time*, *location*, *priority*, *task capability* to

steps[*time*]

for $0 \leq j \leq$ *number of mobile robots* **do**

create random start *time*, *location*

if *capability equals* ***ANY*** *or* ***MULTI*** **then**

create random *mr capability*

else

mr capability = *capability*

Write string containing *time*, *location*, *priority*, *mr capability* to *steps*[*time*]

for $0 \leq j \leq$ *number of mobile robots* + *number of tasks* **do**

Write *steps*[*j*] to file

The initial conditions for the generation process are passed into the application as a set of command line parameters. The command line parameters are the number of tasks, the number of mobile robots, the capability(ies) of the mobile robots, and whether the mobile robots should start at a common location or be distributed randomly. The number of tasks is specified by a comma separated list of integers, without spaces, identifying the number of tasks to be generated for each scenario. Including more than one number of task values in the list creates more than one set of scenarios differing by the number of tasks generated in each. The number of mobile robots is specified by a comma separated list of integers, without spaces, identifying the number mobile robots to be generated for each scenario. Including more than one number of mobile robot values in the list creates more than one set of scenarios differing by the number of mobile robots generated in each. The capability(ies) of mobile robots parameter is one of the capabilities enumerations, the keyword ANY, or the keyword MULTI. See appendix B for information on the capabilities. If one of the capabilities enumerations is specified then each mobile robot in the generated scenarios has the same capability, i.e. it is a homogeneous multi-agent system. If the word ANY is specified then each mobile robot in a scenario is assigned a randomly generated capability. The capability selection is weighted by the rarity of the capability. If the word MULTI is specified then each mobile robot has two randomly selected capabilities. These capabilities will not be identical. The final parameter is a flag to turn on and off the central starting location for the mobile robots. This option is either 1 or 0 and is optional. By default, or when the flag is 0, the mobile robots are placed at random locations throughout the scenario. When the flag is 1

the mobile robots all begin at the center of the scenarios, as if they were all departing from a base of operations. Here is an example command line.

```
ScenarioSetGenerator 1,2,5,10,25,50,100 1,2,5,10 BOMB 0
```

When a capabilities enumeration is passed for the mobile robot capability then tasks are generated with capabilities enumeration and the priority is always one. This decision was made to facilitate the comparison of the utility function based mobile robot to the knapsack based one. See section 5.3 for more details. The priority is randomly selected, weighted on a bell curve and the capability is randomly selected, otherwise. The bell curve causes priority three tasks to be most prevalent in the scenario.

As the scenario generation application executes it writes the scenario files to a hard-coded location with a name that includes the initial conditions and a time stamp so that the files do not overwrite each other. An example file name would be:

```
tasks_2_mrs_5_caps_BOMB_1211002584953.scenario
```

This scenario contains three tasks and five mobile robots capable of bombing.

Once all the scenario files are written to disk the configuration file is created. It contains a hard-coded text string containing a default set of configurations parameters. The list of scenario files, with fully qualified file paths, is then appended to the end of the configuration file. The configuration file is then written to disk. The configuration file name is based on the capabilities of the scenario set and a time stamp so that it does not overwrite an existing configuration file.

4.2 Simulation Execution Software

The simulation execution software has been written to be highly configurable and extensible. There are dynamically loadable classes for each of the task allocation algorithms. This allows one or more of the mobile robot types to be loaded for a given simulation run. There are also dynamically loadable classes for each utility function component. This allows for easy testing and comparison of different functions for each of the utility function components, i.e. proximity, priority, suitability, and rarity. There are dynamically loadable methods for generating a multitude of graphs. This allows for zero or more different types of graphs to be generated during each simulation run. The methods used to define the graphs are part of the Main simulation execution class. All of the dynamically loadable entities are referenced by name in the simulation configuration file. A class diagram for the simulation execution software is found in figure 4.1.

Execution begins in the Main class. There, the configuration file is read. The class then begins to create FirstSimulation based runs for each scenario enumerated in the configuration. It parses the scenario file, executes each step of the scenario at the appropriate simulation time, processing new tasks and mobile robots when they enter or appear in the scenario, and updates each of the tasks and mobile robots until the simulation completes. The processing of new tasks and mobile robots is handled by a task broadcaster object. It sends new tasks to existing mobile robots. It sends all current tasks to new mobile robots. It tracks the completion of tasks. Once complete, the Main class collects the SimulationOutput

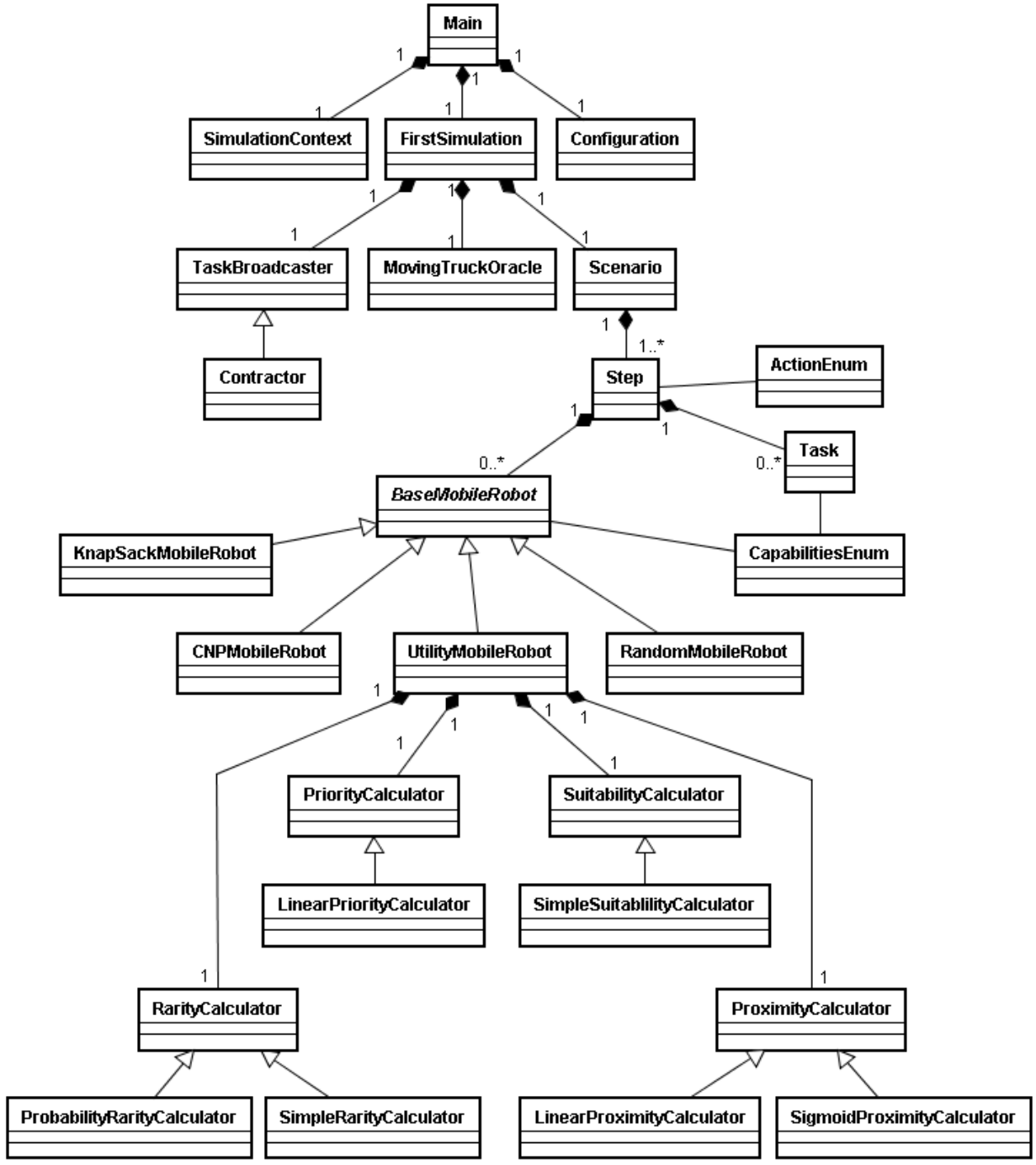

Figure 4.1: Simulation Software Class Diagram

data from each FirstSimulation object. Finally, the data is processed, written to a comma separated text file and graphed using the graphing methods specified in the configuration file.

Algorithm 4.2: Main Execution Algorithm

```
Read configuration file

foreach scenario in the configuration file do
    Create SimulationOutput object

    begin                                   /* Execute FirstSimulation object */
        Read scenario file

        while currenttime < maxexecutiontime do
            Get steps for currenttime

            foreach current steps do

                if step type is ENTERS then
                    Process new mobile robot

                else if step type is APPEARS then
                    Process new task
            Update tasks

            Update mobile robots
    end

    Write data to SimulationOutput object
Graph results
```

4.2.1 Mobile Robots

Along with the utility function algorithm presented in this thesis we have also implemented several other algorithms for the purpose of comparison, which are the random task selection, the contract net protocol, and the knapsack problem based algorithm. These three algorithms are described here.

Each mobile robot algorithm is contained in its own class. These algorithm classes are all derived from the BaseMobileRobot class. The BaseMobileRobot class contains methods for sending and receiving messages, members for holding lists of tasks, and data gathered about task execution, including start and completion time, initial and final energy readings, and number of messages sent. The mobile robot classes also implement several interfaces for simulating movement, position, and communications along with displaying the position of a mobile robot on a graphical display, provided by the YAES simulation framework [BT05], [YAE05].

4.2.1.1 Random Task Selection Implementation

The random task selection mobile robot receives and stores all broadcast tasks. When the mobile robot needs a new task to execute it randomly selects one of the tasks in its list. If it is suitable to perform the selected task it begins. If it is not suitable to perform the

selected task then it randomly selects a new task until it selects one it is suitable to perform. Algorithm 4.3 illustrates the random task selection algorithm.

Algorithm 4.3: Random Task Selection Algorithm

```
if new task received then
    Store task in task_list

if energy > 0 then

    if Task location reached then
        Stop moving

        Save data

        has task = false
    if has task = false then

        repeat
            Randomly select task from list

        until mobile robot suitable to perform task

    if has task = true then
        Move one step closer to task
```

4.2.1.2 Contract Net Protocol Implementation

The Contract Net Protocol (CNP) implementation of task allocation comes from the work of Smith [Smi80], with extensions described by [KSF02]. This simplified version of CNP has only one centrally located contacting agent instead of Smith's distributed contractors that are also task executioners. The contracting agent is represented by an object of type

Contractor, which is derived from task broadcaster. It handles the processing of new tasks and new mobile robots.

Algorithm 4.4: Contract Net Protocol Bid Algorithm

```
Receive request for bid
if not currently executing another task then
    if mobile robot suitable to perform task then
        Generate expected utility for this task
        Send bid for task to Contractor
```

When a new task is introduced the Contractor broadcasts a request for proposal to all the mobile robots. It then waits a fixed amount of simulation cycles for bids, the auction open period. Mobile robots perform the algorithm illustrated in algorithm 4.4 to determine whether to bid and what the value of the bid will be. The bid value is calculated using the same utility function as the utility function based mobile robots, discussed in section 3.3. Once the auction is closed the contractor sends a contract award message to the mobile robot with the highest bid. If that mobile robot is not currently executing another task it sends an accept message to the contractor. The contractor then responds with a begin task execution message. If the mobile robot is currently executing a task it sends a reject task message to the contractor. The contractor then sends the award message to the next highest bidder. This continues until the award is accepted or there are no bidders available. If there are no available bidders the task is reprocessed, i.e. a new request for proposals is broadcast

and the auction open period is reset. See figure 4.2 for a graphical representation of the bid process.

When a new mobile robot is introduced the Contractor sends it all tasks currently being bid on. This gives the new mobile robot the opportunity to immediately begin attempts to obtain tasks to execute.

Algorithm 4.5 illustrates the CNP task selection algorithm.

Algorithm 4.5: Contract Net Protocol Algorithm

```
if new BEGIN task message received then
    Store task in task_list

if energy > 0 then

    if has task = false then
        Select available task

    if has task = true then

        if Task location reached then
            Stop moving

            Save data

            has task = false
        else
            Move one step closer to task
```

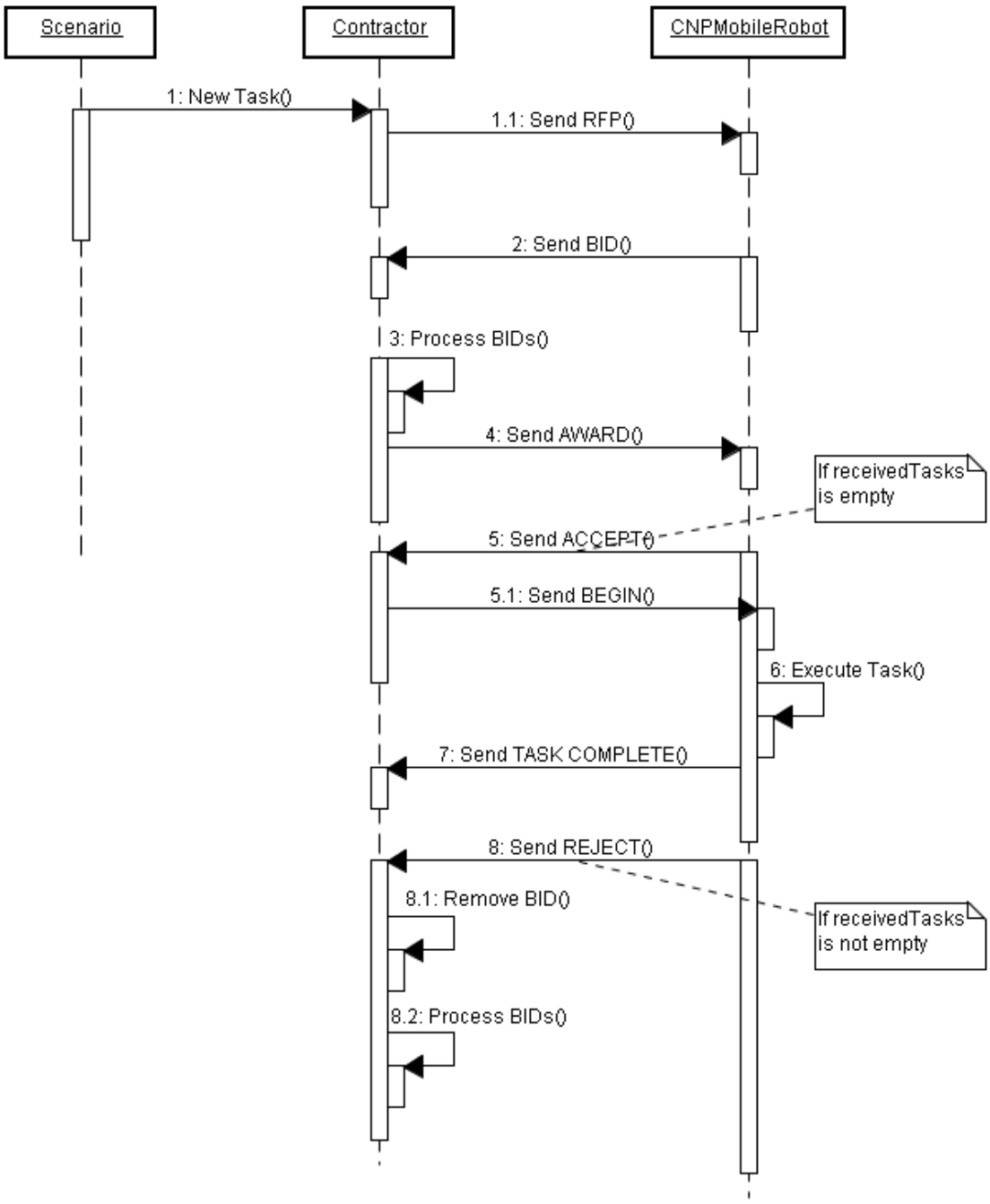

Figure 4.2: Contract Net Protocol Bid Process

4.2.1.3 Knapsack Implementation

The knapsack implementation of the mobile robot works considerably differently than the other mobile robot implementations. The tasks are pre-selected for each mobile robot in a scenario before simulation execution begins. This is because the algorithm for the knapsack problem is computationally intensive and would not be able to run in simulation time to select tasks. The algorithm must also know of all mobile robots and tasks in a scenario for proper computation regardless of the time the mobile robot or task enters or appears. The knapsack implementation that divides the tasks among the mobile robots is called the oracle, because it is able to "see into the future."

The oracle must first parse an entire scenario storing the tasks and mobile robots into lists. It then assigns each task to a mobile robot. Algorithm 4.6 illustrates the oracle's logic. During simulation execution when a mobile robot enters the oracle gives it its pre-defined list of tasks. When a new task appears the task is broadcast to all the mobile robots. During the mobile robot's execution step it selects one of its pre-selected tasks to execute. The task it selects must already have appeared in the simulation. Algorithm 4.7 illustrates the knapsack implementation's execution of tasks.

Algorithm 4.6: Oracle Algorithm

Get all tasks from scenario file

Get all mobile robots from scenario file

Initialize data structures for mobile robots

Initialize array of packed flags for tasks

while *There are tasks to assign and mobile robots with energy* **do**

for $0 \leq j \leq numberofmobilerobots$ **do**
$itemindex = -1$

for $0 \leq i \leq numberoftasks$ **do**

if *mobile robot j has energy to execute task i* **then**
$itemindex = i$

Break

if $itemindex \neq -1$ **then**
Assign task $itemindex$ to mobile robot j

Remove the energy required for the task from the mobile robot's capacity

Set the packed flag for task $itemindex$ to *true*

4.2.2 Tasks

Tasks are represented by a class, Task, that holds simulation data and has code to display a graphical representation of the task. Tasks contain a location in two-dimensional space, a required capability necessary to execute the task, and a priority, how important a task is. Tasks store data generated during simulation execution including task creation, start, and

Algorithm 4.7: Knapsack Algorithm

```
if new task received then
    Store task in task_list

if energy > 0 then

    if Task location reached then
        Stop moving

        Save data

        Has task = false
    if has task = false then

        if available tasks & tasks assigned by oracle then

            foreach available task do

                foreach task assigned by oracle do

                    if available task = task assigned by oracle then
                        Assign task to execute

                        Has task = true
    if has task = true then
        Move one step closer to task
```

end times, what mobile robot executed the task, and the initial and final energies of that mobile robot.

4.2.3 Data Collection

Data is collected by the FirstSimulation object once the simulation for a given scenario is completed. For each task that is completed in the scenario the time to complete the task, the distance traveled completing the task, the reactivity of the mobile robot to the task, and the utility achieved by the mobile robot completing the task are accumulated. The total number of tasks in the scenario, the number of tasks completed, and the total number of messages sent are recorded, and the overall difficulty of the scenario is computed.

After all the scenarios enumerated in a simulation configuration file are executed the data is collected and written to a comma separated text file and any specified graphs are generated.

CHAPTER 5

EXPERIMENTS AND SIMULATION RESULTS

This chapter discusses the experiments performed with the task allocation algorithms. Section 5.1 describes the experiments performed to verify our task allocation algorithm. Section 5.2 explains the criteria for evaluation of the different task assignment algorithms are explained. Section 5.3 explains the tests performed to determine the hardness of our problem. Sections 5.4 and 5.5 discuss the selection of utility function components and weights. Section 5.6 explains the tests performed to compare the task allocation algorithms. Section 5.7 discusses the results of the algorithm comparisons.

5.1 Utility Function Verification

The following scenarios have been pre-defined for use in evaluating the utility function-based dynamic task allocation scheme. $c, c_0..c_n$ represent the capabilities of mobile robots or the capabilities required by tasks. C represents a constant used for logical coordinate offsets. $p, p_0..p_n$ represents the priority of a task, with lower numbered indexes representing higher priorities. $t_0..t_n$ represent simulation time, with lower numbered indexes representing

earlier times. $r, r_0..r_n$ represent the rarities of capabilities in the scenarios. Each scenario is scripted, stored in a text file, and loaded into simulations at simulation initialization time. An explanation of how scenarios are scripted can be found in Appendix A.

In each section there is a description of the initial conditions of the test followed by a description of the expected behavior of the mobile robot(s) executing in the test's scenario. For all tests the observed behavior of the mobile robots matched the expected behavior identified in each section.

Along with these formal tests extensive observation of the mobile robots executing tasks were made using the visual display functionality of YAES. Initial executions of the utility function implementation were also heavily instrumented with print statements and file logging. These investigations were used to assure the utility function implementation performed as designed and that the design responded similarly to how a human operator tasking mobile robots might respond.

5.1.1 Basic Scenario

This scenario allows the mobile robot to select between two equally weighted tasks. A mobile robot, MR1, is in an area at position (x, y) at time t_0 and being capable of c. Two tasks, T1 and T2, appear in the area at positions ($x - C$, $y - C$) and ($x + C$, $y + C$), respectively, at the same time, t_1, with the same priority, p, both requiring the same capability, c.

The expected behavior of MR1 is to perform either one of T1 or T2 then perform the other. T1 is listed first in the scenario file so the mobile robot will actually perform it first.

5.1.2 Temporal Selection Scenario

This scenario allows the mobile robot to select between two equally weighted tasks appearing at different times in the simulation. A mobile robot, MR1, is in an area at position (x, y) at time t_0 and being capable of c. Two tasks, T1 and T2, appear in the area at positions ($x - C$, $y - C$) and ($x + C$, $y + C$), respectively, with the same priority, p, both requiring the same capability, c. Task T1 appears at time t_1. Task T2 appears at time t_2.

The expected behavior of MR1 is to perform T1 then T2.

5.1.3 Capability Scenario

This scenario tests the capability component of the utility function. A mobile robot, MR1, is in an area at position (x, y) at time t_0 and being capable of c_0. Two tasks, T1 and T2, appear in the area at positions ($x - C$, $y - C$) and ($x + C$, $y + C$), respectively, at the same time, t_1, with the same priority, p. Task T1 requires capability c_0 while task T2 requires capability c_1.

The expected behavior of MR1 is to perform T1 then T2.

5.1.4 Priority Scenario

This scenario tests the priority component of the utility function. A mobile robot, MR1, is in an area at position (x, y) at time t_0 and being capable of c. Two tasks, T1 and T2, appear in the area at positions $(x - C, y - C)$ and $(x + C, y + C)$, respectively, at the same time, t_1, both requiring the same capability, c. Task T1 has priority p_0. Task T2 has priority p_1.

The expected behavior of MR1 is to perform T1 then T2.

5.1.5 Proximity Scenario

This scenario tests the proximity component of the utility function. A mobile robot, MR1, is in an area at position (x, y) at time t_0 and being capable of c. Two tasks, T1 and T2, appear in the area at positions $(x - C, y - C)$ and $(x + C + \epsilon, y + C + \epsilon)$, respectively, at the same time, t_1, with the same priority, p, both requiring the same capability, c.

The expected behavior of MR1 is to perform T1 then T2.

5.1.6 Rarity Scenario

This scenario tests the rarity component of the utility function. A mobile robot, MR1, is in an area at position (x, y) at time t_0 and being capable of c_0, with rarity r_0, and c_1, with rarity

r_1. Two tasks, T1 and T2, appear in the area at positions $(x-C, y-C)$ and $(x+C, y+C)$, respectively, at the same time, t_1, with the same priority, p. Task T1 requires capability c_0 while task T2 requires capability c_1.

The expected behavior of MR1 is to perform T1 then T2.

5.1.7 Duplicate Work Scenario

This scenario is a situation where two mobile robots at the same location receive the same two identically weighted tasks at the same time. Two mobile robots, MR1 and MR2, are in an area at position (x, y) at time t_0 and being capable of c. Two tasks, T1 and T2, appear in the area at positions $(x - C, y - C)$ and $(x + C, y + C)$, respectively, at the same time, t_1, with the same priority, p, both requiring the same capability, c.

The expected behavior of MR1 is to perform T1 then T2. The expected behavior of MR2 is to perform T1 then T2. MR1 will arrive at and perform T1 and T2 first.

5.1.8 Two Mobile Robot Capability Scenario

This scenario tests the suitability component of the utility function with two mobile robots and two tasks. Two mobile robots, MR1 and MR2, are in an area at position (x, y) at time t_0 and being capable of c_0 and c_1, respectively. Two tasks, T1 and T2, appear in the area

at positions $(x - C, y - C)$ and $(x + C, y + C)$, respectively, at the same time, t_1, with the same priority, p. Task T1 requires capability c_0 while task T2 requires capability c_1.

The expected behavior of MR1 is to perform T1. The expected behavior of MR2 is to perform T2.

5.2 Evaluation Criteria

To evaluate the performance of our task allocation algorithm we collect the following metrics: reactivity, percent of tasks complete, distance traveled per completed task, utility achieved, number of communications, and fairness.

Reactivity is the time it takes, from task creation, for a task to be assigned to an agent. This provides information on how efficient an implementation is at assigning tasks to agents. Reactivity is measured in number of simulation cycles, s. It is calculated by subtracting the time the task was received by the task broadcaster for assignment, $t_{c,T}$, from the record time an agent begins a given task, $t_{b,T}$. See equation 5.1. The reactivity metric is only calculated for completed tasks.

$$R(T) = t_{b,T} - t_{c,T} \tag{5.1}$$

The percent of tasks complete for a given scenario is the number of tasks complete divided by the total number of tasks in that scenario. It is a general measure of efficiency of an agent implementation. See equation 5.2.

$$P_T(S) = \frac{T_{complete,S}}{T_{total,S}} * 100 \tag{5.2}$$

The distance traveled per completed task, $d_{A,T}$, is a measure of a mobile robot's efficiency. It is measured in simulation logical distance units, m, from the location the mobile robot started the task to the location the mobile robot completed the task.

Utility achieved is a measure of how useful the agents are in performing tasks. The higher the priority of a completed task the higher the utility achieved by completing the task. The shorter the distance traveled in completing a task the higher the utility achieved by completing the task. It is calculated by dividing the priority of a completed task by the distance traveled completing it. It is measured in priorities per m. See equation 5.3.

$$U_a(A, T) = \frac{p_T}{d_{A,T}} \tag{5.3}$$

The number of communications in a scenario is the sum of all communications sent from the task broadcaster and all the communications sent from each mobile robot in a scenario. One of the goals of this thesis is to define a task allocation implementation that minimizes the number of communications utilized in the execution of a scenario.

Fairness is a measure of how evenly tasks are treated and is based on task priority. If an implementation executes all priority one tasks, some of the priority two tasks and none of the priority three tasks it is exceedingly more fair than an implementation that executes all of the priority three tasks without regard to any of the higher priority tasks. This metric is measured by summing up the values of the priorities of all completed tasks then subtracting the sum of the priorities of tasks completed with a lower priority when there were tasks with a higher priority left incomplete. Fairness is unitless. Algorithm 5.1 illustrates how fairness is calculated.

Algorithm 5.1: Computing Fairness

```
for 1 ≤ i ≤ 5 do
    fairness += number of completed tasks * priority i

    remaining = number of incompleted tasks of priority i

    if remaining > 0 then
        total = 0

        for i + 1 ≤ j ≤ 5 do
            total += number of completed tasks * priority j

        if total > remaining then
            fairness -= remaining * priority i

        else if total > 0 then
            fairness -= (remaining - total) * priority i
```

5.3 Proof of Hardness

Within the scope of the 0-1 multi-knapsack problem there are n knapsacks that need to be filled with items. Each item has a weight and a value. The objective is to fill the knapsacks with as many items as possible, up to a maximum given weight, while maximizing the value of the stored items. There is only one instance of each item.

The problem we solve in this thesis is at least as hard as the 0-1 multi-knapsack problem. To prove this we have solved the 0-1 multi-knapsack problem, which is NP-Hard (nondeterministic polynomial-time hard), using our problem solution. Each mobile robot in a scenario becomes a knapsack. The tasks become items to store (assigned to a mobile robot), each one being stored at most once. The initial distance to travel from the mobile robot to a task is the weight of the task to be packed. The priority and capability required by a task are fixed so that the effective profits of all tasks are equal.

We perform tests to compare the performance of a dynamic programming solution of the knapsack problem to the utility function based solution presented in this thesis. Table 5.1 gives a summary of the test results. The reactivity of the dynamic programming solution is much better because the implementation uses an oracle algorithm to assign the tasks. They are ready to be executed as soon as they appear in a scenario. The percent of tasks complete, distance traveled, and utility achieved heavily favor the utility function implementation. This is, at least in part, because using the distance from the initial location of the mobile robot

to compute a weight is inaccurate compared to computing it from where the mobile robot is currently.

Table 5.1: Knapsack to Utility Function Implementation Comparison

	Reactivity (s)	% of TC	DT on TC (m)	UA (priorities/m)	NOC	Fairness
Utility Function	259.1075	79.5276	143.8640	16.2538	1331.0467	0.8411
Knapsack	96.4802	43.8908	323.7067	4.3952	730.5660	0.6616

TC = Tasks Complete

DT = Distance Traveled

UA = Utility Achieved

NOC = Number of Communications

Figures 5.1 and 5.2 show the results of one set of tests for the two implementations. The graphs display the number of tasks completed for each scenario in a set amount of time, averaged over the 50 scenarios with a given set of initial conditions, versus the total number of tasks in a scenario. The greater the number of completed tasks the better the performance of the mobile robots.

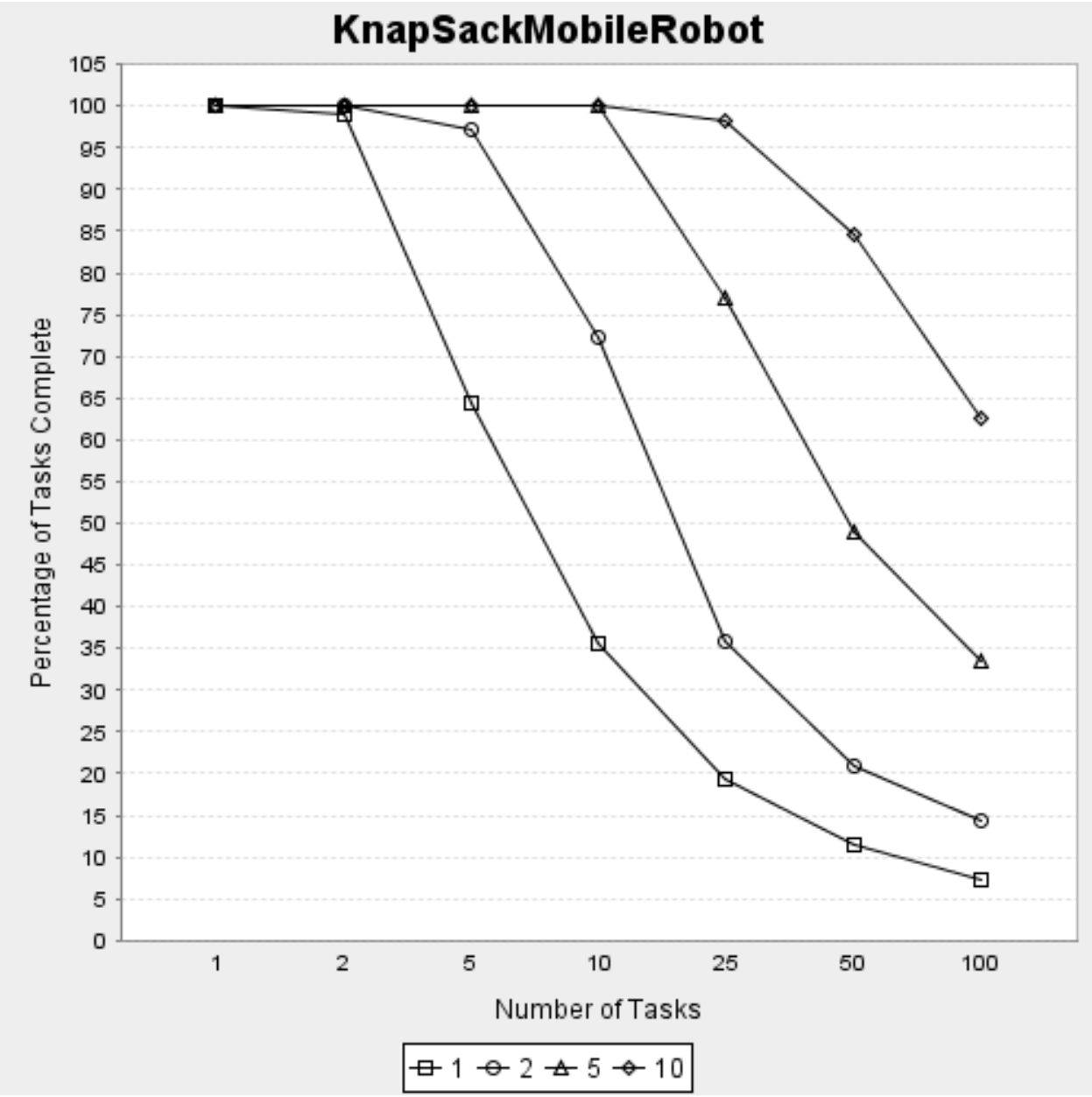

Figure 5.1: Percentage of Tasks Complete for Knapsack Implementation Execution

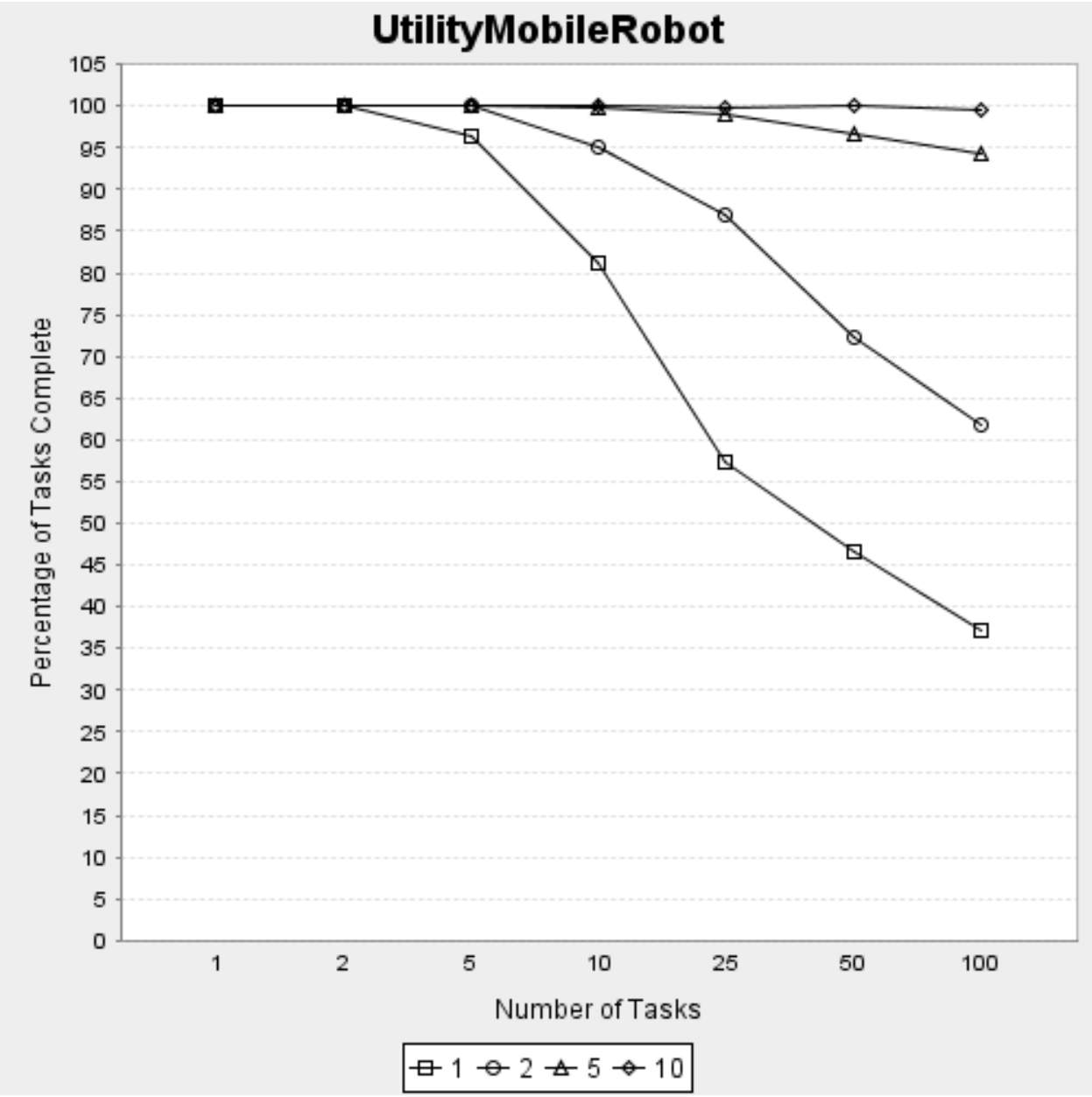

Figure 5.2: Percentage of Tasks Complete for Utility Function Implementation Execution

5.4 Utility Weights Selection

The utility function used in our implementation requires four weights, one each for the priority, proximity, suitability, and rarity components of the function. We have selected several reasonable sets of choices for these weights and compared them by running a comprehensive set of scenarios for each set of weights. In the first set all weights are equal. Remember from 3.2 the sum of the weights must equal one. Therefore, each weight is 0.25. In each of the other four sets of weights tested one of the weights is 0.5 and the rest of the weights are equal, having a value of 0.17. The weights are unitless.

Table 5.2 compares the results of each of the sets of weights. In all evaluation criteria, except one, the set of weights with the proximity weighted 0.5 performed better than the other sets. The one criterion it did not perform better is the reactivity, the time from task creation to the time that task execution begins. The average for the reactivity is 167.0354 and its standard deviation is 3.8406. The reactivity for the proximity weighted 0.5 set is within one standard deviation. These results lead to the selection of the weights set with proximity weighted 0.5, priority weighted 0.17, suitability weighted 0.17, and rarity weighted 0.16 for the mobile robot implementations comparison tests.

Table 5.2: Utility Weights Performance Comparison Chart

	Reactivity (s)	% of TC	DT on TC (m)	UA (priorities/m)
d25 p25 s25 r25	166.9361	54.7157	196.0050	3.2828
d50 p17 s17 r16	168.8076	58.3456	186.0150	3.5097
d17 p50 s17 r16	168.4581	50.3725	203.7821	3.1374
d17 p17 s50 r16	160.5362	53.9040	204.0264	3.1404
d17 p17 s16 r50	170.4393	51.5602	200.5371	3.1954

TC = Tasks Complete, DT = Distance Traveled, UA = Utility Achieved

d** = the proximity weight (** / 100)

p** = the priority weight (** / 100)

s** = the suitability weight (** / 100)

r** = the rarity weight (** / 100)

5.5 Proximity Function Selection

There are two function types considered for the proximity component of the utility function, linear and sigmoidal. With the linear function the proximity result is directly proportional to the distance from the mobile robot to the task being evaluated. With the sigmoid function the proximity result follows an s-shaped curve. The slope of the curve is initially low. As the distance from the mobile robot to the task being evaluated decreases the slope increases then

decreases again as the distance approaches zero. The sigmoid function has two parameters that effect its shape. These are s and m. See section 3.2.1 for a discussion of these parameters. We have selected several reasonable sets of choices for these parameters and compared them by running a comprehensive set of scenarios for each set of parameters and for the linear function implementation.

Table 5.3 compares the results of each of the sets of parameters and the linear function implementations. There is no difference in performance between the simulations run with the different sets of parameters. The sigmoid function implementation performed better than the linear function implementation. There is a 0.0035% improvement in task completion and a 0.1564m reduction in distance traveled. Therefore, the sigmoid function has been chosen for the proximity component of the utility function.

Since there is no difference among the sigmoid function results for the differing s and m parameters an educated guess must be made on what values to use for them. Based on an examination of the the graphs presented in Appendix C the values $m = 250$ and $s = 100$ are chosen. With $m = 250$ the center of the sigmoid coincides with the middle of the mobile robot's constraint of performing tasks only if they are within 500 units of the task. With $s = 100$ the sigmoid function's value when the distance from the mobile robot to the task is 500 is exactly 0 and exactly 1 when the distance is 0, i.e. the function spans the full range of acceptable values for the proximity component.

Table 5.3: Proximity Function Comparison Chart

	Reactivity (s)	% of TC	DT on TC (m)	UA (priorities/m)
Linear Proximity	204.7318	79.3010	150.4250	17.4202
$m = 125$, $s = 50$	204.6632	79.3045	150.5814	17.4238
$m = 125$, $s = 100$	204.6632	79.3045	150.5814	17.4238
$m = 125$, $s = 150$	204.6632	79.3045	150.5814	17.4238
$m = 166.67$, $s = 50$	204.6632	79.3045	150.5814	17.4238
$m = 166.67$, $s = 100$	204.6632	79.3045	150.5814	17.4238
$m = 166.67$, $s = 150$	204.6632	79.3045	150.5814	17.4238
$m = 250$, $s = 50$	204.6632	79.3045	150.5814	17.4238
$m = 250$, $s = 100$	204.6632	79.3045	150.5814	17.4238
$m = 250$, $s = 150$	204.6632	79.3045	150.5814	17.4238

TC = Tasks Complete, DT = Distance Traveled, UA = Utility Achieved

5.6 Random Scenario Generation

The tests performed to compare the contract net protocol and random selection implementations to our implementation are broken up into four parts, homogeneous scenarios, heterogeneous scenarios with randomly placed mobile robots (at simulation start time), heterogeneous scenarios with centrally located mobile robots (at simulation start time), and heterogeneous scenarios with multi-capable mobile robots. Each scenario has a set of initial conditions, number of tasks, number of mobile robots, mobile robot capability (specific, random, or multiple), and whether or not to centrally locate mobile robots. Each combination of initial conditions constitutes a set. Within a set there are 50 individual, randomly generated scenarios that allow for evaluation using monte carlo methods. Results presented in this section represent an averaging of recorded values across the 50 scenarios in a set. The number of tasks and number of agents initial conditions are common to all tests. The number of tasks generated are 1, 2, 5, 10, 25, 50, and 100. The number of mobile robots generated are 1, 2, 5, and 10. These multiple numbers of tasks and mobile robots exercise the scalability of each implementation.

5.6.1 Homogeneous Scenarios

In the homogeneous scenarios tests all tasks require the bombing capability and have a priority of 1, and all mobile robots are capable of bombing. This test case gives a baseline

performance evaluation of the three mobile robot implementations being compared. It isolates the proximity component of the utility function used for the task selection of the utility function implementation and the bid computed for the contract net protocol implementation.

A summary of the results of these tests can be found in Table 5.4.

Table 5.4: Homogeneous Scenarios Performance Comparison

	Reactivity (s)	% of TC	DT on TC (m)	UA (priorities/m)	NOC	Fairness
UF	259.1075	79.5276	143.8640	16.2538	1331.0467	0.8411
RS	279.2934	41.3541	280.6480	6.1699	693.9473	0.6545
CNP	209.3537	30.6193	300.3920	4.9387	60948.5307	0.5660

TC = Tasks Complete, DT = Distance Traveled, UA = Utility Achieved, NOC = Number of Communications

UF = Utility Function, RS = Random Selection, CNP = Contract Net Protocol

5.6.2 Heterogeneous Scenarios

In the heterogeneous scenarios tests task required capabilities and priorities, and mobile robot capabilities are randomly generated. The randomness for capabilities is weighted

Table 5.5: Randomly Placed Heterogeneous Scenarios Performance Comparison

	Reactivity (s)	% of TC	DT on TC (m)	UA (priorities/m)	NOC	Fairness
UF	263.7346	63.1246	167.6047	3.6982	1062.8853	0.6271
RS	265.6561	38.8608	284.0433	1.8773	670.0873	0.4982
CNP	212.0441	31.0106	306.5173	1.5482	60448.9253	0.4757

TC = Tasks Complete, DT = Distance Traveled, UA = Utility Achieved, NOC = Number of Communications

UF = Utility Function, RS = Random Selection, CNP = Contract Net Protocol

based on the rarities of the capabilities. The randomness for priorities is weighted based on a standard distribution.

5.6.2.1 Randomly Place Mobile Robots

The heterogeneous scenarios tests with randomly placed mobile robots simulate groups of mobile robots already dispatched from a base performing tasks or in a loiter situation awaiting commands.

A summary of the results of these tests can be found in Table 5.5.

Table 5.6: Centrally Located Heterogeneous Scenarios Performance Comparison

	Reactivity (s)	% of TC	DT on TC (m)	UA (priorities/m)	NOC	Fairness
UF	319.2429	50.7169	188.1093	2.7889	857.6933	0.5583
RS	247.8778	40.4389	276.7987	1.6575	696.4993	0.5082
CNP	200.2856	32.2396	275.3173	1.5482	59429.4960	0.4846

TC = Tasks Complete, DT = Distance Traveled, UA = Utility Achieved, NOC = Number of Communications

UF = Utility Function, RS = Random Selection, CNP = Contract Net Protocol

5.6.2.2 Centrally Located Mobile Robots

The heterogeneous scenarios tests with centrally located mobile robots simulate groups of mobile robots being commanded from a base or operations area. These tests help to identify problems with multiple mobile robots attempting to perform the same task at the same time by removing the initial distance to tasks as a major contributor to utility function results.

A summary of the results of these tests can be found in Table 5.6.

Table 5.7: Heterogeneous, Multiple Capability Scenarios Performance Comparison

	Reactivity (s)	% of TC	DT on TC (m)	UA (priorities/m)	NOC	Fairness
UF	294.5923	64.0141	160.6460	4.0439	1058.7333	0.6981
RS	279.5001	40.8514	279.1160	2.0561	693.5867	0.5470
CNP	215.0414	31.3197	298.5253	1.5885	60478.5353	0.4845

TC = Tasks Complete, DT = Distance Traveled, UA = Utility Achieved, NOC = Number of Communications

UF = Utility Function, RS = Random Selection, CNP = Contract Net Protocol

5.6.2.3 Mobile Robots with Multiple Capabilities

The heterogeneous scenarios tests with multi-capable mobile robots are similar to the first heterogeneous scenarios tests except each robot has two randomly selected capabilities. These tests most accurately depict currently fielded and future unmanned aerial vehicles, which often have a visual or infra-red sensor, or both, along with a weapons capability. See Appendix B for a survey of currently fielded unmanned aerial vehicles and their capabilities. These tests allow for the exercising of the rarity component of the utility function and the bid calculation.

A summary of the results of these tests can be found in Table 5.7.

5.7 Discussion

From the results of the random scenario generation tests the utility function based implementation out performs the random task selection and contract net protocol implementations in percent of tasks complete, distance traveled per completed task, utility achieved, number of communications and fairness. The rest of this section discusses the results of the tests, broken out by evaluation criterion.

5.7.1 Reactivity

The reactivity results should and do favor the contract net protocol. While it does have overhead associated with the assignment of a task, due to the bidding process, a mobile robot will begin execution of a task as soon as it is received. With the random task allocation and utility function task allocation methods the task is immediately broadcast to all mobile robots in the scenario but may remain in a mobile robot's list of received tasks for some time before being randomly selected of having the highest utility to be selected.

Table 5.8 displays the summary of reactivity results for the three task allocation implementations and the four test cases.

Table 5.8: Reactivity Metric Summary

	UF	RS	CNP
Homogeneous Scenarios	259.1075	279.2934	209.3537
Heterogeneous, Randomly Placed	263.7346	265.6561	212.0441
Heterogeneous, Centrally Located	319.2429	247.8778	200.2856
Heterogeneous, Multi-Capable	294.5923	279.5001	215.0414

UF = Utility Function, RS = Random Selection, CNP = Contract Net Protocol

5.7.2 Percent of Tasks Complete

Consistently high values across all tests for the percent of tasks complete metric would be considered ideal for a fielded multi-agent system. The random task allocation and contract net protocol implementations have consistent performance across all tests, but both fail to complete even half of the tasks in a scenario, on average. The utility function based implementation has a wide range of values for percent of task completion, from 79.5871% down to 50.7028%. The variability is inherent in the design of the implementation.

In the homogeneous scenarios tests, the test case with the best results for the utility function based implementation, the only component contributing to the utility function values is proximity. If a mobile robot is always going to the closest task it will easily finish more tasks than if it had other considerations for what task to execute. The random task allocation implementation does not consider proximity to a task at all when selecting a task.

Table 5.9: Percent of Tasks Complete Metric Summary

	UF	RS	CNP
Homogeneous Scenarios	79.5276	41.3541	30.6193
Heterogeneous, Randomly Placed	63.1246	38.8608	31.0106
Heterogeneous, Centrally Located	50.7169	40.4389	32.2396
Heterogeneous, Multi-Capable	64.0141	40.8514	31.3197

UF = Utility Function, RS = Random Selection, CNP = Contract Net Protocol

The contract net protocol implementation considers proximity when bidding, but what tasks are available to bid on limits how effective this contribution can be.

In the heterogeneous scenarios tests with centrally located mobile robots, the test case with the worst results for the utility function based implementation, the utility function based implementation tends to exhibit a follow the leader effect. If two mobile robots have the same capability and are each closest to a particular task they both will attempt to execute that task. One mobile robot will complete the task before the other, wasting energy and time the second mobile robot could have been using to execute another task.

Table 5.9 displays the summary of percent of tasks complete results for the three task allocation implementations and the four test cases.

Table 5.10: Distance Traveled per Completed Task Metric Summary

	UF	RS	CNP
Homogeneous Scenarios	143.8640	280.6480	300.3920
Heterogeneous, Randomly Placed	167.6047	284.0433	306.5173
Heterogeneous, Centrally Located	188.1093	276.7987	275.3173
Heterogeneous, Multi-Capable	160.6460	279.1160	298.5253

UF = Utility Function, RS = Random Selection, CNP = Contract Net Protocol

5.7.3 Distance Traveled per Completed Task

The distance traveled per completed task shows how efficient an implementation is. The utility function based implementation is the most efficient because it most often allocates the closest task to the mobile robot. The ability for the utility function based implementation to interrupt the currently executing task of a mobile robot assures the close proximity of executing tasks regardless of when a task appears in a scenario. There is no guarantee, or expectation, that a randomly selected task will be the closest to the mobile robot. The contract net protocol cannot adequately handle a task appearing later in a simulation that may have a higher utility for a mobile robot than the one it is currently executing.

Table 5.10 displays the summary of distance traveled per completed task results for the three task allocation implementations and the four test cases.

5.7.4 Utility Achieved

The utility achieved metric is a measure of how useful a mobile robot is, i.e. how well it selects and how efficiently it executes the most useful tasks. The utility function based implementation achieves the highest utility because it most directly selects tasks based on their priority, which is our measure of how important a task is. The contract net protocol implementation has the same priority component in its bid computation as that used in the utility function base implementation. The bidding process and the implementation's inability to preempt tasks obscure the priority component.

The utility function based implementation also scores high with this metric because of its better scoring with the distance traveled per completed task metric. The better performance of the utility function based implementation with the utility achieved metric is not purely due to its better performance with the distance traveled per completed task metric. The utility function based implementation performs 202% better than the random selection method with the distance traveled metric, but 284% better with the utility achieved metric. It performs better 216% better than the contract net protocol implementation with the distance traveled metric, but 355% better with the utility achieved metric.

Table 5.11 displays the summary of utility achieved results for the three task allocation implementations and the four test cases.

Table 5.11: Utility Achieved Metric Summary

	UF	RS	CNP
Homogeneous Scenarios	16.2538	6.1699	4.9387
Heterogeneous, Randomly Placed	3.6982	1.8773	1.5482
Heterogeneous, Centrally Located	2.7889	1.6575	1.5482
Heterogeneous, Multi-Capable	4.0439	2.0561	1.5885

UF = Utility Function, RS = Random Selection, CNP = Contract Net Protocol

5.7.5 Number of Communications

The utility function based implementation performs similarly to the random selection implementation, with regards to number of communications. They both implement a broadcast message to all mobile robots and the task broadcasting center when a task is complete. The higher number of communications required by the utility function based implementation is due to the fact that this implementation completes more tasks.

The contract net protocol has the added communications overhead of the bidding process. The contractor broadcasts a request for bids message. Any suitable, unoccupied mobile robots respond with a bid message. The contractor sends an award message to the winning bidder. The winning bidder sends an accept or reject message. The contractor will then either send a begin message in response to an accept message or send out a new award message out to the next highest bidder. This reject/award sequence is repeated until an accept message is received by the contractor or all the bidders have rejected the accept

Table 5.12: Number of Communications Metric Summary

	UF	RS	CNP
Homogeneous Scenarios	1331.0467	693.9473	60948.5307
Heterogeneous, Randomly Placed	1062.8853	670.0873	60448.9253
Heterogeneous, Centrally Located	857.6933	696.4993	59429.4960
Heterogeneous, Multi-Capable	1058.7333	693.5867	60478.5353

UF = Utility Function, RS = Random Selection, CNP = Contract Net Protocol

message. If no bids are received or no bidders accept the award then the process starts again with a new request for bids broadcast.

Table 5.12 displays the summary of number of communications results for the three task allocation implementations and the four test cases.

5.7.6 Fairness

The utility function based implementation performs the fairest of them all. This is due to its better handling of a task's priority than the random selection and contract net protocol implementations. See the discussion of the utility achieved metric in section 5.7.4. The utility function based implementation executes more of the higher priority tasks before executing the lower priority tasks. The total number of tasks complete, indicated by the percent of

Table 5.13: Fairness Metric Summary

	UF	RS	CNP
Homogeneous Scenarios	0.8411	0.6545	0.5660
Heterogeneous, Randomly Placed	0.6271	0.4982	0.4757
Heterogeneous, Centrally Located	0.5583	0.5082	0.4846
Heterogeneous, Multi-Capable	0.6981	0.5470	0.4845

UF = Utility Function, RS = Random Selection, CNP = Contract Net Protocol

tasks complete metric in section 5.7.2, may also have some effect on this metric, but no direct correlation can be determined.

Table 5.13 displays the summary of fairness results for the three task allocation implementations and the four test cases.

CHAPTER 6

CONCLUSIONS

This thesis has introduced the use of utility functions to determine what task a given mobile robot attempts to perform. The utility function is based on multiple criteria of a mobile robot and a task in the system. The criteria considered in the utility function include proximity of the robot to the task, the priority of the task, the suitability of the robot to perform the task, and the rarity of the capabilities required to complete the task. This allocation scheme has been implemented to utilize communications sparingly.

This thesis introduces two concepts, which, to our knowledge, are not covered in any previous literature. These concepts are suitability and rarity.

The suitability of a mobile robot to perform a task has been introduced to help distinguish tasks of similar, but distinct, capabilities and make the tasks assigned to mobile robots better match the capabilities of them. The suitability is represented as a two-dimensional matrix of mobile robot capabilities to tasks' required capabilities. The entries range from 1, an exact match, to -1, completely unsuitable, where values between 1 and 0 representing limited equivalencies between capabilities.

The rarity of a mobile robot's capabilities has been introduced to allow for the conservation of rare resources. A mobile robot with a rare capability should be used sparingly within a scenario to preserve it for tasks requiring its capability. The rarity values were calculated based on the percentage of real world fielded UAVs with a given capability.

The following assumptions have been made during the development of this thesis. The specific case of the mobile robot as a UAV is used to develop the implementations and simulation. A straight line path to a task is the best choice while a mobile robot proceeds to a task, because, with UAVS, the time to reach a task is proportional to the distance to the task. Mobile robots are always in communications range of each other and the task broadcaster. Communications take no time to send or process, and consume no energy. Tasks take no time and consume no energy to execute once the mobile robot is at the task location.

Our implementation requires a minimum of messages for communications. One message is sent out per task generated to each mobile robot. One message is sent out per completed task to each mobile robot. There are no other inter-robot communications.

Based on the results presented in section 5.7 the utility function based implementation of task allocation out performs the random selection and contract net protocol implementations. The utility function based implementation is able to perform a higher percentage of tasks in a given scenario in a given amount of time. It performs tasks more fairly, giving the proper attention to tasks based on their priority. It generates a higher achieved utility. It travels shorter distances, on average, per task to complete a given task.

The specific type of scenario has considerable influence on the performance of the utility function task allocation implementation. Our implementation has consistently good performance with one or multiple capabilities. It perform better, overall, when the mobile robots in the simulation are distributed throughout the scenario before simulation start, as opposed to being centrally located. Adding a launch and loiter system to position mobile robots before task allocation begins would benefit a real world implementation of this work.

CHAPTER 7

FUTURE WORK

There have been several areas left unexplored or under explored during the creation of this thesis. Failures could be implemented. Communications systems could be better modeled. Task could have more or different criteria or requirements. Energy usage could be explored. Utility weights could be determined differently. More task allocation schemes could be implemented for comparison purposes.

One of the attributes explored in [Nij04] is robustness, or how well failures are handled within a system. In this thesis mobile robots are incapable of failure and all communications are always received. It is possible to script a failure into the scenario file to occur at a given time. This would allow the failures to be repeatable for testing. The YAES simulation framework contains communications models that can induce dropped messages. These two changes to the simulation would allow for testing, and implementation improvements, based on situations more closely resembling the function of real mobile robots.

Two features of many tasks in real world unmanned aerial vehicle applications are deadlines and time on target. A deadline requires that a task be completed no later than a given time. This would require a new component in the utility function to give increased

importance to tasks that need to be completed sooner than other tasks. Time on target defines a specific time that a mobile robot must arrive at a task by. This feature would require changes in the way all mobile robots in the simulation behave. Currently mobile robots move at fixed speed. To implement time on target a mobile robot would need to be able to vary its speed. The utility function would also need to take the time on target into account when computing the utility of a task. This may be implemented as an increased importance, similar to that for deadlines, or an increased importance on all tasks near the task with a time on target requirement. With the proximity importance the mobile robot can still complete other tasks while waiting for the appropriate time to complete the task requiring a specific time on target instead of varying speed to slow the mobile robot's arrival at the task.

Other features of tasks could be the need for more than one mobile robot to complete them, the requiring of more than one capability, the requiring of time to complete them once a mobile robot arrives at them, the requirement of a specific sequence of task completion, or the generation of new tasks upon task completion.

Tasks may not have a location, only an appointment time or duration. Such tasks may include communications relaying. This could be done by adding a task duration, as mentioned above. For these tasks the proximity could be set to 1, meaning the mobile robot has already arrived at the task, when computing their utility.

In the currently implemented simulation energy consumption is directly related to speed. The simulation could be expanded to allow the mobile robot to vary its speed. The simulation

could also change the relationship between speed and energy consumption. It can continue to be linear or it could be changed to be quadratic, exponential, or a piecewise function based on real world data.

Neither the communications system nor the capabilities in the simulation currently cause any energy consumption. The energy consumption for communications is part of the communications modeling expansion discussed above. The capabilities' energy consumption would require some research into the physical devices that provide the capabilities. Once such a survey is complete and the energy consumption of the capabilities is defined, preferably in units energy per simulation cycle, the energy consumed by a capability could be accounted for when the completion of a task is recorded by a mobile robot.

A more comprehensive way of determining the utility weights could be implemented. We selected five reasonable choices for the set of utility weights then ran the simulation with each set. The best performing set of weights was selected after reviewing the simulation results. Genetic algorithm techniques could be used to determine the weights. This method could find an optimal set of weights through iterative processes.

There are many more task allocation schemes that could be selected for comparison to the utility function based implementation. A scheme in which each mobile robot determines its own tasks for execution would be a good next step. This type of scheme would execute similarly to the utility function based implementation. The contract net protocol could be reimplemented with distributed task allocation, better resembling what Smith proposed [Smi80].

APPENDIX A

SIMULATION CONFIGURATION

A.1 Simulation Configuration File

A.1.1 Configuration File Format

The configuration file provides configuration information to the YAES-based simulation for each execution, which may contain multiple scenario executions with differing parameters. The original configuration file format and Java code for reading and interpreting the file was developed by L.J. Luotsinen and M.A. Khan. The configuration file contains sets of name/value pairs. The pairs are read into a static class that is accessed by the simulation code. There are several pairs of note.

the MobileRobotClassPath pair describes where to find the Java classes defining the mobile robots used in the simulation.

The PlotMethodNames.n pairs provide a list of plot method names to execute. There can be any number of plot method names, n, starting at zero. The plot methods must be defined in the Main class of the simulation.

The MobileRobotType.n pairs provide a list of mobile robot class names to execute. Each scenario identified in this configuration file is executed with each mobile robot type listed. There can be any number of mobile robot class names, n, starting at zero.

The MobileRobot.ProximityWeight, MobileRobot.PriorityWeight, MobileRobot.SuitabilityWeight, and MobileRobot.RarityWeight pairs define the utility function weights for each of the four

components of the utility function. These parameters are only used when the UtilityMobileRobot is being simulated.

The MobileRobot.ProximityCalculatorClassName, MobileRobot.PriorityCalculatorClassName, MobileRobot.SuitabilityCalculatorClassName, and MobileRobot.RarityCalculatorClassName define what version of each of the four utility function components to use to calculate them. The value of the pair is a class name that must be defined and located in the Java package associated with that type of function. These parameters are only used when the UtilityMobileRobot is being simulated.

The MobileRobot.SigmoidProximityMValue and MobileRobot.SigmoidProximitySValue pairs define the s and m parameters of the SigmoidProximityCalculator. These parameters are only used when the UtilityMobileRobot is being simulated with the SigmoidProximityCalculator.

The MobileRobot.Endurance pair defines the total distance each mobile robot can travel.

The ScenarioFileName.n pairs identify what scenario files to load when executing this simulation. There my be one or more scenarios loaded, n, starting at zero. Each file is executed sequentially and its results tallied in the simulation.

A.1.2 Example Configuration File

The following is the configuration file for the rarity scenario of the utility function verification tests.

```
# IMPORTANT! Make sure AppResourcesPath is correct and
# that AppLogPath is where you want to store your log files and statistics.


#=== General Node setup ===
MobileRobotClassPath = dta.nodes.


PlotMethodNames.0 = createPercentTasksCompleteVsNumTasksChart
PlotMethodNames.1 = createTotalEnergyExpendedVsNumTasksChart
PlotMethodNames.2 = createUtilityAchievedVsNumTasksChart
PlotMethodNames.3 = createNumCommunicationsVsNumTasksChart
PlotMethodNames.4 = createEnergyExpendedPerTaskVsNumTasksChart
PlotMethodNames.5 = createPercentTasksCompleteVsDifficultyChart
PlotMethodNames.6 = createTotalEnergyExpendedVsDifficultyChart
PlotMethodNames.7 = createUtilityAchievedVsDifficultyChart
PlotMethodNames.8 = createNumCommunicationsVsDifficultyChart
PlotMethodNames.9 = createEnergyExpendedPerTaskVsDifficultyChart
```

```
PlotMethodNames.10 = createPercentTasksCompleteVsNumTasksChartMRSeries

PlotMethodNames.11 = createTotalEnergyExpendedVsNumTasksChartMRSeries

PlotMethodNames.12 = createUtilityAchievedVsNumTasksChartMRSeries

PlotMethodNames.13 = createNumCommunicationsVsNumTasksChartMRSeries

PlotMethodNames.14 = createEnergyExpendedPerTaskVsNumTasksChartMRSeries

PlotMethodNames.15 = createPercentTasksCompleteVsDifficultyChartMRSeries

PlotMethodNames.16 = createTotalEnergyExpendedVsDifficultyChartMRSeries

PlotMethodNames.17 = createUtilityAchievedVsDifficultyChartMRSeries

PlotMethodNames.18 = createNumCommunicationsVsDifficultyChartMRSeries

PlotMethodNames.19 = createEnergyExpendedPerTaskVsDifficultyChartMRSeries


MobileRobotType.0 = UtilityMobileRobot

#MobileRobotType.1 = KnapSackMobileRobot

#MobileRobotType.2 = RandomMobileRobot

#MobileRobotType.3 = CNPMobileRobot


#=== UtilityMobileRobot Utility Function Weights ===

MobileRobot.ProximityWeight = 0.16

MobileRobot.PriorityWeight = 0.17

MobileRobot.SuitabilityWeight = 0.51

MobileRobot.RarityWeight = 0.13
```

```
MobileRobot.ProximityCalculatorClassName = SigmoidProximityCalculator

MobileRobot.PriorityCalculatorClassName = LinearPriorityCalculator

MobileRobot.SuitabilityCalculatorClassName = SimpleSuitabilityCalculator

MobileRobot.RarityCalculatorClassName = SimpleRarityCalculator


MobileRobot.SigmoidProximityMValue = 125.0

MobileRobot.SigmoidProximitySValue = 50.0


MobileRobot.Speed = 0.1

MobileRobot.MaxTaskRange = 500


#=== MobileRobot energy setup ===

MobileRobot.Endurance = 1500


#=== General Application Settings ===

AppName = Dynamic Task Allocation (YAES)


#AppMode (0 = Live, 1 = Replay Engine)

AppStartMode = 0

AppResourcesPath = src/dta/resources/
```

```
AppImageFile = simple_800_600.jpg

AppIconFile = icon.gif

AppWindowWidth = 800

AppWindowHeight = 600

AppSceneWidth = 400

AppSceneHeight = 300


AppMaxCycles = 10000

AppDrawGUI = True


AppFps = 0.1f

AppPathPlannerTimeout = 100

#For fastest execution set this to false and FPS to 0

AppDrawAgentStatus = True

AppDrawAgentSpeech = True

AppGUIRepaintRate = 1

AppConsolePrint = True

AppConsolePrintProgress = 10


# logging and statistics

AppLogPath = src/dta/log/
```

```
AppLogData = True

AppLogStatisticsClassPath = dta.statistics.

AppLogExecutionFile = execution.log

AppLogStatisticsFile = stats.log

AppLogStatistics.0 = FinalEnergyStatistic


ScenarioFileName.0 = C:/Documents and Settings/svanderweide/My Documents/Thesis/
  Dissertation/SVanderWeide_MSc/scenarios/verification/
  tasks_2_mrs_1_caps_ALL_rarity.scenario
```

A.2 Scenario Definition File

A.2.1 Scenario File Format

A text-based file format has been developed to describe a single scenario involving one or more mobile robots and one or more tasks in a given area. Each line of the file represents one command to the simulation to be executed at the specified time. The commands are enters, appears, generates, fails, and transmitted to. Currently only enters and appears are implemented in the simulation for this work. The others are defined for future work. The enters command commands the simulation to create a new mobile robot with the given

parameters. The appears command commands the simulation to create a new task with the given parameters.

Below is the definition of the commands for the scenario file format.

```
t: UAV<n> enters at pt<x,y> heading<h> with speed<v> and capability<a,b,...>
   and rank<r>.
t: T<n> appears at pt<x,y> with priority<p> requiring capability<a,b,...>.
t: UAV<n> generates T<m> at pt<x,y> with priority<p> requiring capability<a,b,...>.
t: UAV<n> fails.
t: T<n> transmitted to UAV<m> from user pt<x,y> with priority<p>
   requiring capability<a,b,...>.
```

Where:

- t is the time of command generation.
- n and m are ordinal values used to identify a mobile robot or task, starting at zero.
- r is an ordinal value used to designate the order that mobile robots with a given capability enter the simulation, ranging from zero to the maximum number of mobile robots with a given capability.
- x is the X coordinate the mobile robot or task originates at, from 0.0 to 800.0.
- y is the Y coordinate the mobile robot or task originates at, from 0.0 to 600.0.

- h is the starting heading of mobile robot, in the range [0.0, 360.0).
- v is the initial velocity of the mobile robot, greater than zero.
- a and b are text values identifying the capabilities of the mobile robots or required by the task.

The capabilities a, b, etc. can be one of:

- ELECTRO_OPTICAL
- INFRA_RED
- SAR_RADAR
- CHEMICAL
- RADIATION
- BOMB
- MISSILE

A.2.2 Example Scenario File

The following scenario file is for the rarity scenario of the utility function verification tests.

```
1.0: UAV<0> enters at pt<200.0, 200.0> heading<0.0> with speed<1.0>
```

```
     and capability<BOMB,MISSILE> and rank<0>.

2.0: T<0> appears at pt<100.0, 100.0> with priority<1> requiring capability<BOMB>.

2.0: T<1> appears at pt<300.0, 300.0> with priority<1> requiring capability<MISSILE>.
```

APPENDIX B

CAPABILITIES

The capabilities used for the tasks in this thesis were derived from a survey of unmanned aerial vehicles (UAV) currently fielded or under development. The capabilities identified are electro-optical camera, infra-red camera, synthetic aperture radar, missiles, bombs, radiation sensors, and chemical agent sensors. Table B.1 displays a list of UAVs, there capabilities, and the number of vehicles identified as being fielded.

Based on the number of UAVs and their capabilities we derive the rarity for each capability as one minus the number of UAVs with that capability divided by the total number of UAVs surveyed. The total number of UAVs surveyed is 296. Equation B.1 illustrates how the rarity for each capability was calculated.

$$r_C = 1 - \frac{n_C}{n_{total}} \tag{B.1}$$

Table B.2 lists the capabilities, the number of UAVs with that capability, and their calculated rarities.

Table B.1: Survey of UAV Capabilities

Vehicle	Capabilities	Number in Use	Reference
MQ-1 Predator	EO, IR, M	125	[Pre08]
RQ-6 Outrider	EO, IR	20	[Out08]
RQ-3 Darkstar	EO, SAR	4	[Dar08]
RQ-5 Hunter	EO, IR	56	[Hun08]
RQ-7 Shadow	EO, IR	43	[Sha08]
MQ-8 Firescout	EO, IR	17	[Fir08]
RQ-4 Global Hawk	EO, IR, SAR	17	[Glo08]
SAIC Vigilante	EO, IR, N, C	4	[Vig08]
MQ-9 Reaper	EO, IR, M, B	10	[Rea08]

EO = Electro-optical Camera

IR = Infra-red Camera

SAR = Synthetic Aperture Radar

M = Missile

B = Bomb

N = Radiation Sensor

C = Chemical Agent Sensor

Table B.2: Rarity Probabilities

Capability	Count n_c	Rarity r_c
EO Camera	296	0.0000
IR Camera	292	0.0135
SA Radar	21	0.9291
Chemical Sensor	4	0.9865
Radiation Sensor	4	0.9865
Bomb	10	0.5439
Missile	135	0.9662

EO = Electro-optical, IR = Infra-red, SA = Synthetic Aperture

APPENDIX C

PROXIMITY SIGMOID FUNCTION

Figures C.1, C.2, and C.3 illustrate the proximity functions and allow for a visual comparison of the linear proximity function with the sigmoid proximity function. The sigmoid function is displayed with all of its tested parameters.

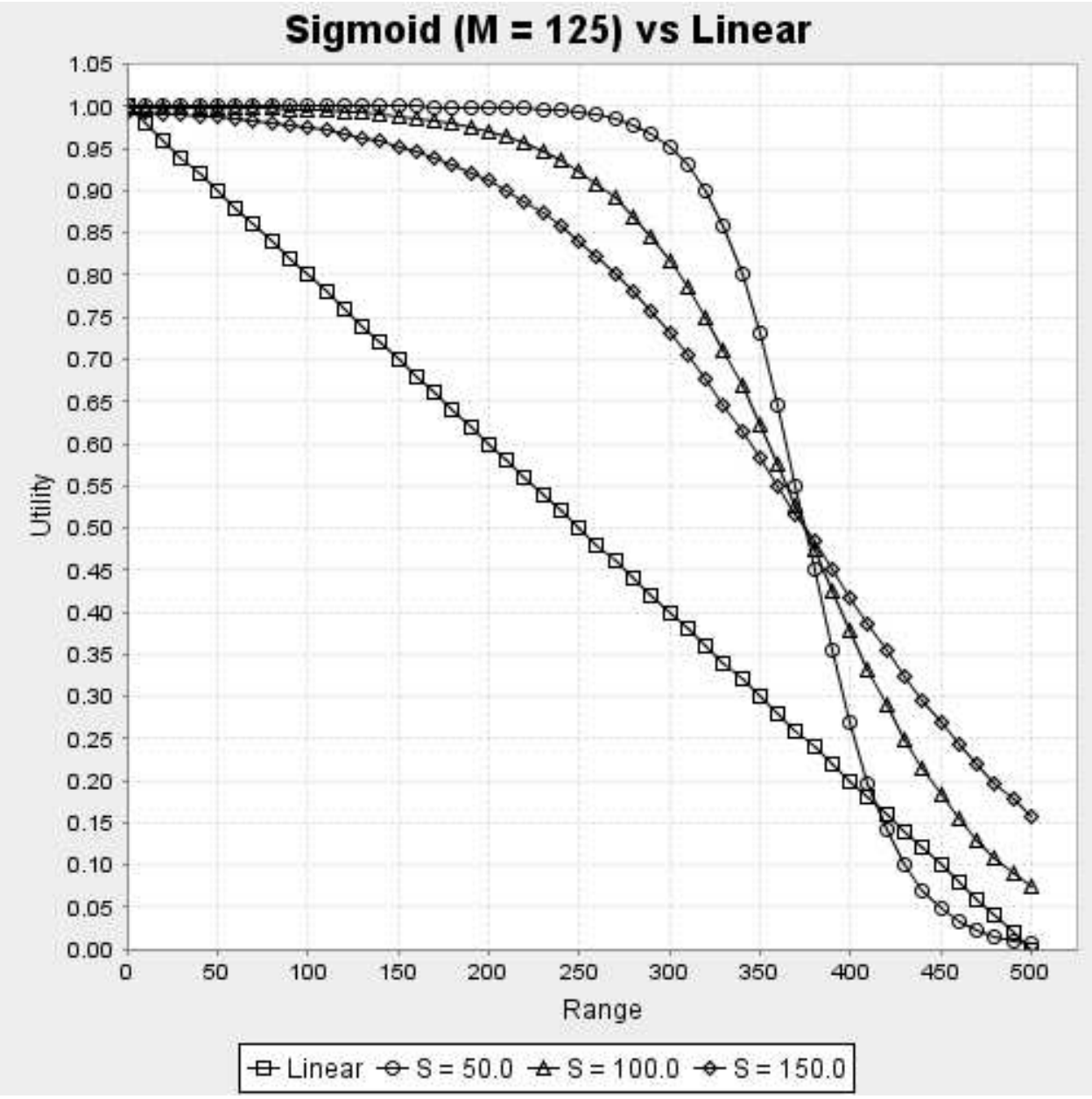

Figure C.1: Graph of Sigmoid Functions with M=125 vs Linear Function

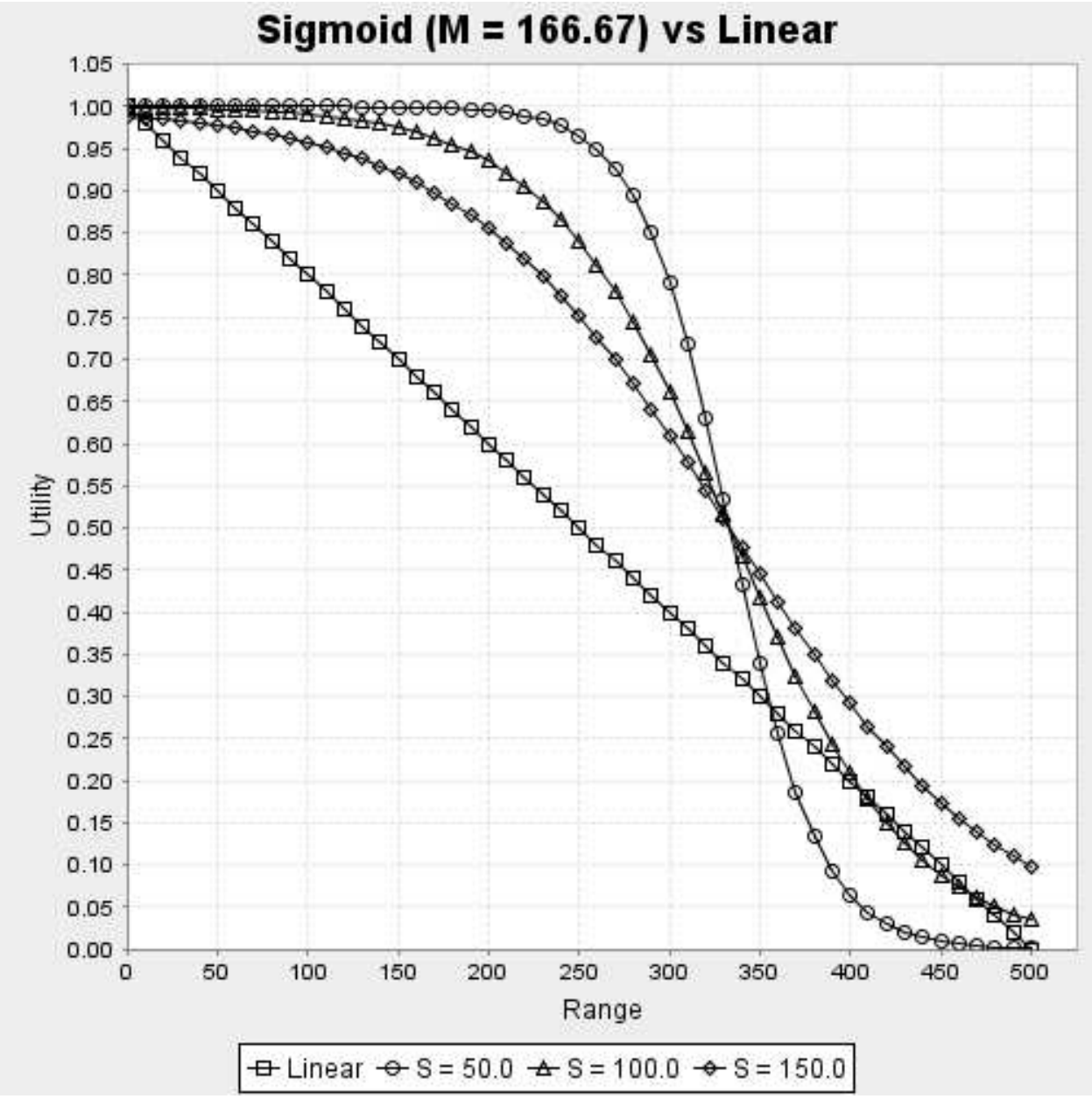

Figure C.2: Graph of Sigmoid Functions with M=166.67 vs Linear Function

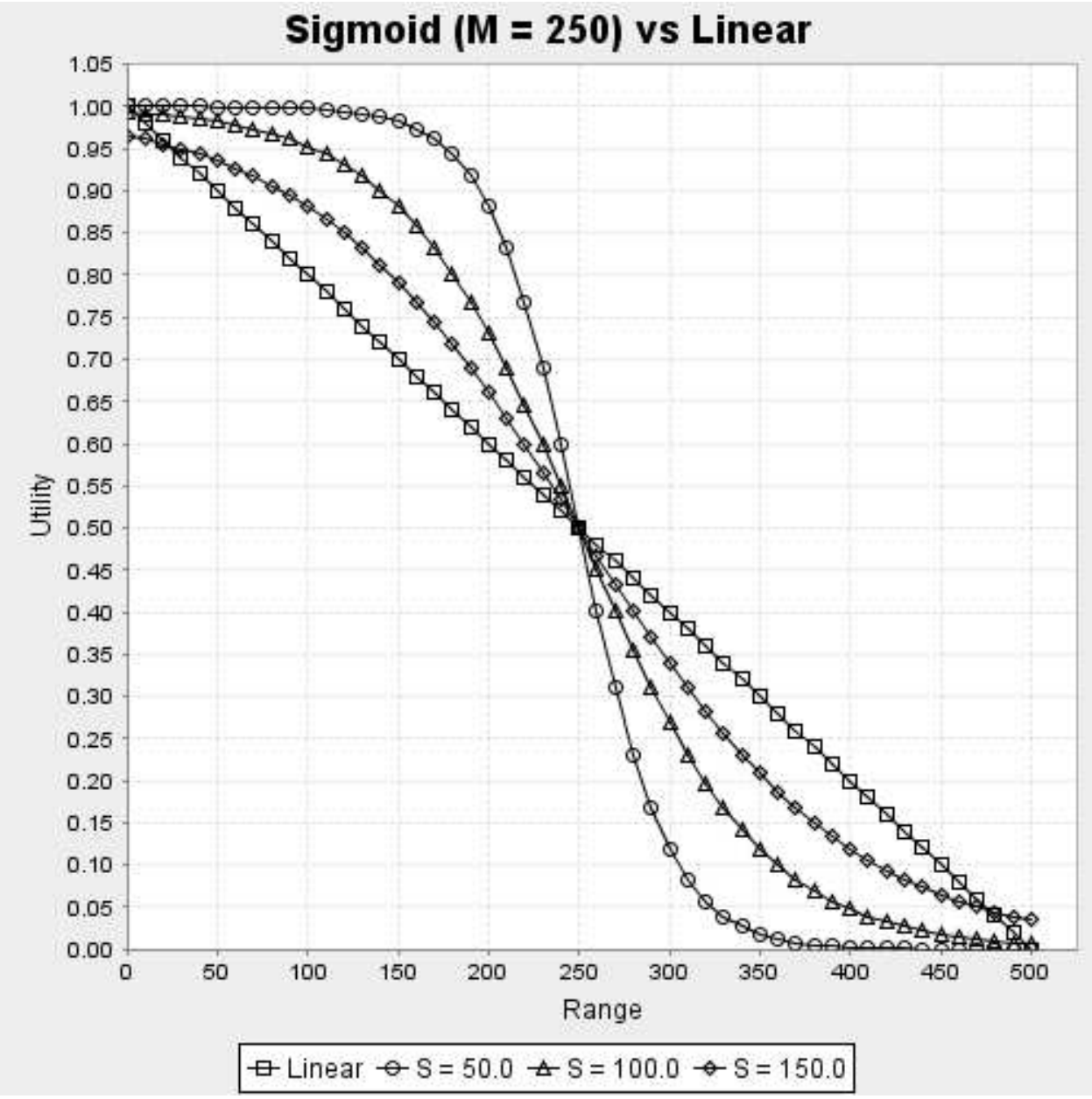

Figure C.3: Graph of Sigmoid Functions with M=250 vs Linear Function

LIST OF REFERENCES

[AL05] Sherief Abdallah and Victor Lesser. "Modeling task allocation using a decision theoretic model." In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pp. 719–726, New York, NY, USA, 2005. ACM.

[BT05] Ladislau Bölöni and Damla Turgut. "YAES: a modular simulator for mobile networks." In *MSWiM '05: Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pp. 169–173, New York, NY, USA, 2005. ACM.

[CFK97] Y. Uny Cao, Alex S. Fukunaga, and Andrew B. Kahng. "Cooperative Mobile Robotics: Antecedents and Directions." *Autonomous Robots*, **4**(1):7–23, March 1997.

[Dar08] URL http://www.dfrc.nasa.gov/Gallery/Photo/Tier3-/index.html, 2008.

[Das06] Prithviraj Dasgupta. "Distributed automatic target recognition using multi-agent UAV swarms." In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 479–481, New York, NY, USA, 2006. ACM.

[DJ03] P. Davidsson and S. Johansson. "Evaluating multi-agent system architectures: A case study concerning dynamic resource allocation." *Lecture Notes in Computer Science*, **2577**:170–183, 2003.

[Fir08] URL http://www.is.northropgrumman.com/, 2008.

[Gar06] Sergio Garces. *Extending Simple Weighted-Sum Systems*, pp. 331–339. Charles River Media, Boston, MA, USA, 2006.

[Glo08] URL http://www.spacewar.com/reports/
Last_Block_10_Global_Hawk_Arrives_For_Check_Flights.html, 2008.

[GMV04] Aaron Gage, Robin Murphy, Kimon Valavanis, and Matt Long. "Affective Task Allocation for Distributed Multi-Robot Teams." Technical Report CRASAR-TR2004-26, Center of Robot Assisted Search and Rescue, University of South Florida, 2004.

[Gri05] Nathan Griffiths. "Task delegation using experience-based multi-dimensional trust." In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pp. 489–496, New York, NY, USA, 2005. ACM.

[Hun08] URL http://www.defense-update.com/products/h/hunter.htm, 2008.

[Kra97] Sarit Kraus. "Negotiation and Cooperation in Multi-Agent Environments." *Artificial Intelligence*, **94**(1-2):79–97, 1997.

[KSF02] T. Knabe, M. Schillo, and K. Fischer. "Improvements to the FIPA contract net protocol for performance increase and cascading applications." In *In International Workshop for Multi-Agent Interoperability at the German Conference on AI*, 2002.

[LEF06] L.J. Luotsinen, J.N. Ekblad, T.R. Fitz-Gibbon, C. Houchin, J. Key, M.A. Khan, J. Lyu, J. Nguyen, R. Oleson, G. Stein, S. Vander Weide, V. Trinh, and L. Bölöni". "Comparing apples with oranges: evaluating twelve paradigms of agency." In R.H. Bordini, M. Dastani, J. Dix, and A.F. Segrouchni, editors, *Fourth international Workshop on Programming Multi-Agent Systems (PROMAS-2006)*, pp. 51–65, May 2006.

[Mon97] Phillipe Mongin. *Expected Utility Theory*, pp. 342–350. Edward Elgar, London, 1997.

[Nij04] T.C. Nijmeijer. "Comparing Multi-agent System Architectures." In *1st Twente Student Conference on IT*, Enschede, The Netherlands, June 2004.

[Out08] URL http://www.designation-systems.net/dusrm/app2/q-6.html, 2008.

[OVM05] Charles L. Ortiz, Régis Vincent, and Benoit Morisset. "Task inference and distributed task management in the Centibots robotic system." In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pp. 860–867, New York, NY, USA, 2005. ACM.

[Pis95] David Pisinger. *Algorithms for Knapsack Problems.* PhD thesis, University of Copenhagen, 1995.

[Pre08] URL http://www.vectorsite.net/twuav.html, 2008.

[Rea08] URL http://www.ga-asi.com/products/predator_b.php, 2008.

[San93] T. W. Sandholm. "An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations." In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pp. 295–308, Hidden Valley, Pennsylvania, 1993.

[San98] T. Sandholm. "Contract types for satisficing task allocation: I theoretical results." In *In AAAI 1998 Spring Symposium: Satisficing Models*, 1998.

[Sha08] URL http://www.globalsecurity.org/intell/systems/shadow.htm, 2008.

[Smi80] Reid G. Smith. "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver." *IEEE Transactions on Computers*, **C-29**(12):1104–1113, March 1980.

[SS06] Paul Schermerhorn and Matthias Scheutz. "Social coordination without communication in multi-agent territory exploration tasks." In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 654–661, New York, NY, USA, 2006. ACM.

[SSG06] P. B. Sujit, A. Sinha, and D. Ghose. "Multiple UAV task allocation using negotiation." In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 471–478, New York, NY, USA, 2006. ACM.

[Vig08] URL www.advancedtechnologiesinc.com/uav/pdfs/vigilante_sept04.pdf, 2008.

[WBH06] Danny Weyns, Nelis Boucké, and Tom Holvoet. "Gradient field-based task assignment in an AGV transportation system." In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 842–849, New York, NY, USA, 2006. ACM.

[Woo02] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley and Sons Ltd, West Sussex, England, 2002.

[YAE05] "YAES: Yet Another Extensible Simulator." URL http://netmoc.cpe.ucf.edu/Yaes/Yaes.html, 2005.