MODELING TEAM PERFORMANCE FOR COORDINATION CONFIGURATIONS OF LARGE MULTI-AGENT TEAMS USING STOCHASTIC NEURAL NETWORKS

by

Jumpol Polvichai

B. Eng. in Computer, KMIT Thonburi, Bangkok, 1991

M. Eng. in Computer, Chulalongkorn University, Bangkok, 1996

M. S. in Electrical and Computer, Carnegie Mellon University, 2001

Submitted to the Graduate Faculty of
the Department of Information Sciences in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2007

UNIVERSITY OF PITTSBURGH

INFORMATION SCIENCES DEPARTMENT

This dissertation was presented

by

Jumpol Polvichai

It was defended on

December 11th 2007

and approved by

Michael Lewis, PhD, Professor

Stephen C. Hirtle, PhD, Professor

Paul W. Munro, PhD, Associate Professor

Katia Sycara, PhD, Professor

Paul Scerri, PhD

Dissertation Advisors: Michael Lewis, PhD, Professor,

Paul Scerri, PhD

MODELING TEAM PERFORMANCE FOR COORDINATION CONFIGURATIONS OF LARGE MULTI-AGENT TEAMS USING STOCHASTIC NEURAL NETWORKS

Jumpol Polvichai, PhD

University of Pittsburgh, 2007

Coordination of large numbers of agents to perform complex tasks in complex domains is a rapidly progressing area of research. Because of the high complexity of the problem, approximate and heuristic algorithms are typically used for key coordination tasks. Such algorithms usually require tuning algorithm parameters to yield the best performance under particular circumstances. Manually tuning parameters is sometimes difficult. In domains where characteristics of the environment can vary dramatically from scenario to scenario, it is desirable to have automated techniques for appropriately configuring the coordination. This research presents an approach to online reconfiguration of heuristic coordination algorithms. The approach uses an abstract simulation to produce a large data set to train a stochastic neural network that concisely models the complex, probabilistic relationship between configurations, environments and performance metrics. The final stochastic neural network, referred as the team performance model, is then used as the core of a tool that allows rapid online or offline configuration of coordination algorithms to particular scenarios and user preferences. The overall system allows rapid adaptation of coordination, leading to better performance in new scenarios. Results show that the team performance model captured key features of a very large configuration space and mostly captured the uncertainty in performance as well. The tool was shown to be often capable of reconfiguring the algorithms to meet user requests for increases or decreases in performance parameters. This work represents the first practical approach to quickly reconfiguring complex sets of algorithms for a specific application.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

This dissertation can be completed with the supports from many people. First, I would like to thank my advisor, Prof. Michael Lewis. He has kindly and patiently given me opportunity, guidance, and all other supports for over the past four years of my doctoral study at the School of Information Sciences. I could not have come this far without his kind supports. I also would like to thank my co-advisor, Dr. Paul Scerri from Carnegie Mellon University for his advise, encouragement, and a great deal of help in my research before and during the dissertation process.

I am also grateful for all the supports and suggestions from my dissertation committee members, Prof. Katia Sycara (Carnegie Mellon), Prof. Paul Munro (School of Information Sciences), and Prof. Stephen Hirtle (School of Information Sciences).

Financially, I owe a debt of gratitude to the Royal Thai Government for the scholarship for my master and part of my doctoral studies in the United States, and to my advisor, Prof. Michael Lewis for the financial support during the time I have been in the Ph.D. program at the School of Information Sciences.

I appreciate the help from Sean Owens in providing the computing resources at Carnegie Mellon that I have used for running the simulations and experiments necessary for my research. I also thank my friend, Kroeksak Tangmankhong, for lending me an additional computer for running my experiments. Having more computers, I can speed-up my experiments and my research.

I am also thankful for my friends, Chirayu Wongchokprasitti and Worasit Choochaiwattana, for their helps, suggestions, and for just being there every time when I practice in preparation for the presentations in the Preliminary Examination, Comprehensive Examination, Proposal Defense, and Dissertation Defense.

I would really like to thank my parents, and my wife's parents in Thailand for their supports and patient wait for over the past eight years.

Finally, I would like to thank my wife, Savinya, and my daughter, Nattamon for their love, support and encouragement. Their companionship has made my study in the United States a pleasure one, from the beginning through my graduation.

1.0 INTRODUCTION

In the field of multi-robot research, it is very challenging for a human operator to effectively coordinate with the robots performing very complex tasks in a large-scale multi-agent system [104]. In a large-scale system, there are usually an extremely large number of parameters, the relationships between which are very complex. This situation prevents human operators from receiving their intended results from the robots. The ultimate goal of research in this area, consequently, is to find an approach that helps humans working with robot teams to be able to obtain results from the robot teams that are as close as possible to the humans' intentions. This dissertation proposes the new idea on the method used to capture the very complex relationships of a large number of parameters, as well as the way to use these captured relationships to make a model to help users to configure the system. The method that has been developed from this research can also generally apply to any complex, nonlinear interacting system of algorithms, or complex, nonlinear interacting physical processes.

1.1 CHALLENGES

In order to best understand the importance of this work, major related challenges in the research area are explored below. The research, when done, will contribute to the solutions to these challenges.

1.1.1 Large Team Coordination Is Very Complex

Sophisticated, complex coordination allows large groups of agents to perform complex tasks in domains such as space [57], the military [36] and disaster response [54]. Cooperation between heterogeneous robot teams is very complex when the tasks cannot be completely divided and assigned a priori and when the robots must work together, dynamically interacting to achieve common goals; for example, see the work of [79]. For a large-sized team [105], the teamwork algorithms include several coordination algorithms, such as task allocation, communication, and planning. The relationships between these algorithms and the behaviors of the team are highly non-linear, stochastic, and extremely complex. Knowing only a part of the system well, a human operator is not likely to understand how the team can be controlled to yield the best performance. Therefore, it is critical for human operators to have a better understanding of the teamwork algorithms [106].

However, this broad understanding cannot easily be obtained by running a few real missions. In multi-robot systems, it is usually impossible to run several physical experiments. Therefore, simulations are used to gain a better understanding of the developing system [24][39][33][65]. Nevertheless, using simulations is more challenging when the system is highly non-linear, stochastic, and extremely complex because a simulation can only provide information on one setting at a time. Fortunately, simulations can be run at fast speeds to create large amounts of data about the relationships between parameters and team performance. To represent these complex relationships, a meta-model of teamwork algorithms [64] was then created from the teamwork simulation. As a result, the team meta-model plays a large role in facilitating the process of gaining a better understanding of the system.

In practice, a software tool with graphic user interfaces can be created based on the team meta-model, making it possible for human operators to quickly explore the relationships between coordination algorithm parameters and team performances. Thus, the developers can perform experiments by tuning the input parameters and observing how the team performance changes. Moreover, it is possible to allow the developers to specify the expected results of team performances and to receive a coordination configuration that best meets these specific performance constraints. This tool is also very helpful during the design process. When developing the team, the developers may have different algorithms for each component of the team architecture. For example, there are several algorithms for task allocation. The team meta-model can determine which algorithm for task allocation is better in a particular configuration setting. When dealing with a new problem domain, users have the opportunity to experiment with different coordination configurations and determine which one is most preferred.

1.1.2 Complex Algorithms Are Required to Work Together

In cooperative control, selecting the right coordination strategy can have a significant impact on the team performance. In many multi-robot applications (e.g., [35], [77], [79], [78]), collaborative robots must coordinate their plans or actions. Selecting the right action, plan, or resource to resolve a problem can have a significant impact on the team performance, particularly in larger multi-robot systems. Unfortunately, selecting the right strategy is difficult. Often a wide diversity of strategies is available, and these strategies can lead to significant variations in the team performance of multi-robot systems.

Due to the high computational complexity of coordination, critical coordination algorithms typically use heuristics that are parameterized and need to be tuned for specific domains to yield the best performance. When several coordination algorithms are used together, the performance of one algorithm will likely affect the performance of the other algorithms. Consequently, the parameters of the individual algorithms must be tuned as a group. Moreover, the relationship of coordination configurations, environmental conditions, and team performance is highly non-linear and extremely complex. Hence, obtaining the best performance from a set of coordination algorithms often involves a complex parametric tuning process that must be performed on a per-problem basis.

The process of tuning control parameters for better team performance is performed not at the level of the coordination algorithm, but also at the level of the individual robot. For example, Parker [77] demonstrated an approach to allow a heterogeneous group of robots to adapt their actions while environmental conditions changed over time. L-ALLIANCE architecture was proposed with a learning capability for controlling parameter adjustments that makes it possible for the robots to adapt their behaviors over time in response to dynamic changes in team capabilities and the environment. These global control parameters, such as threshold of activation, robot impatience, and robot acquiescence, are set according to the task and the different types of robots used in the system. In order to obtain good performance, tuning these control parameters is required. Parker used initial learning for parameter settings before measuring performance. In her work, the approach of dynamic task allocation was demonstrated with real heterogeneous robots on cooperative box-pushing experiments. In [78], she demonstrated a distributed approach using the potential field technique for a cooperative team of robots tracking multiple moving targets by considering actions that keep moving objects under surveillance. The L-ALLIANCE architecture was used to provide mechanisms for fault-tolerant cooperative control and allows robot team members to adjust their actions in response to the actions of teammates. The results show success in achieving the target task. However, in order to determine optimal parameters, the system required considerable tweaking.

Controlling a team of robots with complex coordination architecture requires human operators to compromise when setting algorithm parameters. Each parameter has its own advantages and disadvantages. When considering all parameters simultaneously, selecting each value becomes even more complicated. Although some parameters are fixed, such as domain parameters, some parameters have a wide range of possible values. Some parameters relate to selecting different strategies at the global or local level. Some parameters change over time, such as the number of remaining UAVs. In any given situation, keeping all of these parameters in mind and knowing how each parameter affects the outcomes of the robot team is quite impossible for human operators.

The meta-model of teamwork architecture plays a significant role in solving this problem [64]. This meta-model refers to a model of an abstract teamwork simulation model. The team meta-model can be used to find the best configuration of the team to meet specific performance constraints, e.g., the tradeoff between communication bandwidth and a good allocation of resources. Using the team meta-model in reverse allows users to specify performance tradeoffs and rapidly receive a configuration that best meets those constraints. Reviewing the changeable configuration parameters could yield a configuration that best meets the required performance tradeoffs. Furthermore, the team meta-model could be utilized to allow the team to be reconfigured online by determining appropriate parameters for ongoing situations. When executing a mission, human operators can simply specify the requirements and allow the team meta-model to determine new configurations that best meet those requirements. This approach provides a powerful and effective way to manipulate the team meta-model during execution time and provides an additional mechanism for the supervisory control of executing teams.

1.1.3 Humans Need Intelligible Interactions

When executing a mission, it is essential for human operators to monitor online coordination and preemptively take actions that improve the overall performance. For a review of significant human factors, particularly human workload and situation awareness, see [22]. In the field of human-robot collaboration, many current researchers, including [31], [28], [76], and [70], are focused on implementing interactive control systems that are capable of increasing situation awareness, decreasing human control skill, and balancing human mental workload while yielding an improved performance. It would be helpful for interactive team control if ongoing critical situations could be detected and referred to human operators so that overall team performance could be improved, for an example see [103].

The team meta-model can be utilized as an Intelligent Team Assistance (ITA) system that monitors online situations and gives suggestions or warning messages related to changing configurations. During a mission, the system gathers all available information related to the ongoing situation of the team in the field. When a critical situation occurs, the ITA system uses the team model to provide useful suggestions or warning messages about how the team can change to improve the situation. For example, network communication bandwidth may be reduced, limiting the time-to-live for information tokens to two hops and requiring another team reconfiguration to lessen the degradation in performance. The ITA system may suggest that the user consider one of the following actions: lowering the role threshold, increasing the instantiation rate, or changing the instantiation rule strategy to local. These suggestions have a high probability of improving team performance.

1.1.4 Modeling Very Complex Relationships Is Hard

Many interesting complex systems generally are stochastic and nonlinear. In order to understand such complex systems, much ongoing research is studying how to precisely model uncertainties in such complex systems. To accurately characterize a complex system requires a model of the system as well as a model of the uncertainty impacting the system[3]. That inherent uncertainty leads to uncertainty in the performance of the system. The importance of modeling uncertainty in system performance has been recognized in many fields, such as complex biochemical systems especially in genetic regulatory systems [1, 108, 48], robotic systems performing in real world environment [113], robust control systems [6], and data mining [16]. Notice that for many of these systems, the uncertainty can not be captured as a normal distribution and will often not have been characterized at all. While a range of techniques exist to describe a system and its performance, techniques for capturing the uncertainty in that performance are not as readily available. In order to provide human users with tools for intelligible interactions with complex system, it is necessary to have a novel approach to capturing both the performance of a system and the uncertainty related to that performance in a concise, easy to use form.

1.2 RESEARCH OBJECTIVES

With these research challenges in mind, a new approach is intended to assist human users interacting with a complex system. The goal in the field of multi-agent system is to design a intelligible way for human users to effectively interact with a large-scale multi-agent system. To accomplish this goal, the main objective is *to model the relationship between the configuration parameters of coordination algorithms and the performance measures for large teams.* To complete this main task, the new concept of stochastic neural networks is applied. The next task is to exploit the learned model for assisting human users to successfully cooperate with the actual system. To achieve this task, an interactive user interface that works with the learned model is developed. Initial experiments [86] show that the proposed approach has promisingly helped human users to achieve the intended team performance in a smaller version of a coordination teamwork algorithm.

1.3 RESEARCH STATEMENT

In this research, I have demonstrated that the new approach effectively provides human users with tools for interacting with a large-scale multi-agent system. This means that, with this approach, the human user will be more effective in cooperating with a very complex system. To achieve that, the primary challenge is how to model the very complex, nonlinear relationships. Another challenge is how to develop a tool that helps a user to quickly get coordination configurations for best performance by specifying preferences on output parameters and environmental conditions. In this section, a proposed approach is described through three key ideas.

1.3.1 Approach

Previous approaches to configuring coordination for a team in a particular domain typically required hand-tuning [29] or learning [10] in the domain. Hand-tuning parameters is a time-consuming process that typically requires extensive experience with the algorithms in order to obtain good performance. Learning requires that the team performs many trials under the specific circumstances in which it is used. If the environment can vary dramatically, e.g., the specific characteristics of a disaster response scenario can vary greatly, but the same team will respond to each environment, learning may be infeasible [93]. Thus, previous work does not provide a good solution to the problem of rapid team configuration. This research proposes an approach to configuring coordination algorithms that incorporates the following three key concepts.

1.3.1.1 Creating a Team Performance Model The first concept is very important to create a team performance model that captures the relationships between the environment, team configuration parameters, and performance measures in a way that allows rapid exploration by users. Due to the non-determinism of environments and coordination algorithms, these relationships are highly non-linear and stochastic. To create a team performance model, an abstract simulation of the coordination algorithms with a highly configurable environment is developed first. Because the simulation can be run at high speeds, it can be used to create large amounts of data on the relationships between parameters. These data are treated as training data. Then, to create a precise model of the data, the new concept of neural networks with stochastic features [84] is applied. Then, the team performance model is generated from the method of genetic algorithms. At the end, a team performance model is a result that represents complex relationships between the environment, configuration, and team performance.

1.3.1.2 Searching For the Best Configuration The second concept in this research is to make use of the team performance model to search for the best configuration of the team that meets specific performance constraints, e.g., the tradeoff between enough communication bandwidth and better allocation of resources. Using the team performance model in reverse allows users to specify performance tradeoffs and rapidly receive a configuration that best meets those constraints. Since not all parameters are configurable, e.g., the observability of the domain cannot be changed during execution, an ordinary back propagation method of the neural network cannot be used to find input parameters that meet our output requirements. Instead, a search over the changeable configuration parameters is performed to discover a configuration that best meets the required performance tradeoffs.

1.3.1.3 Allowing Online Reconfiguration The third concept is to allow the team to be reconfigured online, using the team performance model to determine the appropriate parameters for the prevailing conditions. When users have changing preferences or know of changing constraints, they can simply specify these requirements and allow the team performance model to find algorithm parameters that will best meet those requirements. This approach provides a powerful and effective way to manipulate team performance during execution time and provides an additional mechanism for the supervisory control of executing teams.

1.4 RESEARCH QUESTIONS

Designing the approach is intended to solve the following important research questions.

1. How to effectively model a complex, nonlinear system?
2. What is the proper way to collect training data so that they are sufficient to represent a large configuration space and to use in the learning process?
3. How well does the concept of a stochastic neural network represent the complex relationship between the configuration parameters of coordination algorithms and performance measures for large teams?
4. What is an effective learning algorithm to train the stochastic neural networks so that uncertainties in training data are captured?
5. Is this new tool useful for supporting human users for working with a large-scale multi-agent system?

1.5 PROBLEM DESCRIPTION

In this section, the problem model is defined. The problem that we are considering is determining the best coordination configuration according to user constraints. The team performance model is used as a function to map a set of input parameters to a set of output performances. Using the user preference function, the best configuration is identified. However, the process of constructing the team performance model is much more difficult. To model a relationship between the configuration parameters of coordination algorithms and the team performance measures for large teams is not an easy task. The complexity of this relationship is described.

1.5.1 Team Coordination Problem

In this section, some details of the team coordination algorithms for cooperative multi-agent teams used to develop our modeling approach are described. First, we consider the team behavior at a high level. To achieve the high level goal of the team, team members start plans upon detecting particular states/events in the environment. Then, sub-teams are formed to execute those instantiated plans. Later, to maintain accurate mutual belief models guaranteeing cohesive behavior, it is crucial for all sub-team members to update state information relevant to the plans within the sub-teams. In addition, to detect and resolve the possible conflicts between plans, members of sub-teams communicate information about their goals across different sub-teams. Finally, allowing the entire team to leverage the local sensing capabilities of each team member, agents share locally sensed information through an acquaintance network, which connects all agents in the team independently from sub-team relationships. The acquaintance network usually follows a small world network architecture which means any two teammate agents are separated by a relatively small number of neighboring agents.

The general details of team organization and coordination problem are: A team of a large number of agents, $A = \{a_1, a_2, \cdots, a_n\}$, work together to achieve a high level common goal, G. To achieve the ultimate goal G requires achieving a number of sub-goals, $\{g_1, g_2, \cdots, g_i, \cdots\}$. In addition, the entire team will receive a *reward* ($reward_i$), a construct similar to utility used for team control and learning, when any sub-goal g_i is satisfied. For example, in a football team, sub-goals of a high level goal for winning the game might be to score points and prevent another team from scoring. Because of this predefined relationship the football team collects a reward whenever they manage to score points or prevent another team from scoring. In the teamwork algorithms, to satisfy these sub-goals the team utilizes plan templates, $Plan = \{plan_1, plan_2, \cdots, plan_i, \cdots\}$, stored in a team library. Each plan template consists of four elements as $plan_i =< g_i, conditions_i, roles_i, reward_i >$. The first element in the template indicates the sub-goal g_i. The second element specifies the conditions under which the plan is to be executed, $conditions_i = event_1 \cap event_2 \cap \cdots \cap event_l$. The third element lists the possible individual roles, $roles_i = \{r_1, r_2, \cdots, r_k\}$, required to achieve the sub-goal g_i. Each role is specified by its task, $r_i =< task_i, ability_i, resource_i >$, i.e., a explanation of the task, the required capabilities to complete the task and the needed resources to execute the role. The last element indicates the values of $reward_i$ the team will receive on successful satisfaction of the sub-goal g_i.

For example, a possible plan template for a sub-goal (scoring points) can be defined as: <Plan-Score= (Throw a ball for a touchdown), (A free receiver is in the end zone∩Quarterback can throw), $\{r_1, r_2, r_3\}, (120) >$. This plan template can be instantiated if two conditions are present: the quarterback can throw and a free receiver is in the end zone. When this plan is instantiated, there are three roles to be assigned. If this plan succeeds, the team will get a reward 120 reward units. The possible three roles in this template are: throwing a ball, receiving a ball and protecting the one who throws the ball, i.e., $r_1 =<$ (Throwing a ball), (Skillful in throwing a ball), (Gloves) >, $r_2 =<$ (Receiving a ball), (Skillful in receiving a ball), (Gloves) > and $r_3 =<$ (Protecting the one who throws the ball), (Quick and strong in tackling), (Protection gears) >. To perform r_1, an agent is required to be able to throw a football very well and perhaps wear gloves.

The coordination of the team can be configured in many possible way according to the coordination parameters, as shown in Table 1.1. The details of these parameters are described as follows. First, the *Number of Team Members* is a configuration parameter that specifies how many agents are in the team. In this case, the minimum and maximum numbers of agents in the team are 10 and 1000 agents respectively. The *Number of Plan Templates* is used to specify a total number of plan templates that might be instantiated by the team, e.g., if this parameter is set to be 20, the team will have totally of 20 plan templates in the team library. The *Roles Per Team Member* indicates the number of roles a team member can simultaneously perform, e.g., if this parameter is set to be 3, team members have the ability to perform 3 roles at once. The *Total Number of Preconditions* specifies the number of possible preconditions the system can test, e.g., if this parameter is set to be 50, there are a total of 50 possible preconditions that can be used. The *Preconditions Per Plan* specifies the average number of preconditions per plan, e.g., if this parameter is set to be 5, on the average, each plan requires five preconditions for instantiation. The *Roles Per Plan* indicates how many roles the plan requires, e.g., if this parameter is set to be 2, there are two required roles in the plan. The *Plan Template Policy* specifies who has the plan templates, e.g., this parameter is usually set to be 1.0, it means that every team member knows every templates. The *Number of Capability Types* indicates numbers of different capability types in the team, e.g., if this parameter is set to be 5, there are five different capabilities. The *Percent Capable* specifies the percentage of available capabilities that an agent has, e.g., if this parameter is set to be 0.5, an agent will have the ability to fill roles for 50 percent of the capability types. The *New Precondition Rate* indicates the rate at which preconditions are satisfied, e.g., if this parameter is set to be 0.2, preconditions become true twice in 10 steps on the average. The *Precondition Detection Rate* defines the probability that a particular team member locally senses a particular new precondition, e.g., if this parameter is set to be 0.05, an agent might sense a new precondition approximately five times in 100 steps. The *Associate Network Density* specifies an average number of links connecting a team member with its neighbors, e.g., if this parameter is set to be 2, the average connected neighbors of any agent will be 2 links. The *Role Threshold* specifies the minimum value of capability required for performing a role, e.g., if this parameter is set to be 0.8, it requires an agent to have at least 80 percent of all required capabilities to perform the role. The *Instantiation Rule* indicates the policy governing how a team member instantiates a plan. Options are: ALWAYS-whenever it can, LOCAL-instantiate if at least one condition is locally sensed, or PROBABILISTIC-instantiate in a particular step at rate specified by *Instantiate Rate*. The *Instantiate Rate* indicates the probability that a plan will be instantiated if conditions match and *Instantiation Rule* is PROBABILISTIC. Finally, the *Information Token Time To Live* indicates how many steps the information token can be passed among agents before it is removed. Moreover, all measures of performance are listed with their descriptions in Table 1.2.

1.5.2 Problem Model

Table 1.1 and Table 1.2 list an example of system and configurable parameters and measures of performance which are used in particular domain. Other domains may be different or have a larger set. The system parameters are fixed by the environment, but may change

Table 1.1: List of example configuration parameters with their possible ranges and types (S=system, C=configurable).

Configuration Parameters	**Ranges**	**Types**
Number of Team Members (NTM)	*10. . . 1000*	*S*
Number of Plan Templates (NPT)	*1. . . 20*	*S*
Roles Per Team Member (RPT)	*1. . . 5*	*S*
Total Number of Preconditions (TNP)	*20. . . 120*	*S*
Preconditions Per Plan (PPP)	*1. . . 10*	*S*
Roles Per Plan (RPP)	*1. . . 10*	*S*
Plan Template Policy (PTP)	*0.0. . . 1.0*	*S*
Number of Capability Types (NCT)	*1. . . 20*	*S*
Percent Capable (PCP)	*0.0. . . 1.0*	*S*
New Precondition Rate (NPR)	*0.0. . . 1.0*	*S*
Precondition Detection Rate (PDR)	*0.0. . . 1.0*	*S*
Associate Network Density (AND)	*1. . . 16*	*S*
Role Threshold (RTH)	*0.0. . . 1.0*	*C*
Instantiation Rule (IRU)	*0.0. . . 1.0*	*C*
Instantiate Rate (ISR)	*0.0. . . 1.0*	*C*
Information Token Time To Live (ITL)	*1. . . 10*	*C*

Table 1.2: List of example measures of performance.

Performance Parameters	**Description**
Percentage Possible (PP)	*probability that a plan is instantiated by team*
Reward (RE)	*the total reward received by the team*
Messages Per Agent (MA)	*number of information token moves*
Conflicts Detected (CD)	*number of conflicts detected*
Plans Instantiated (PI)	*number of plans created*
Conflict Resolution Messages (CR)	*number of conflict detection messages sent*
Role Allocation Messages (RA)	*number of messages sent to perform role allocation*

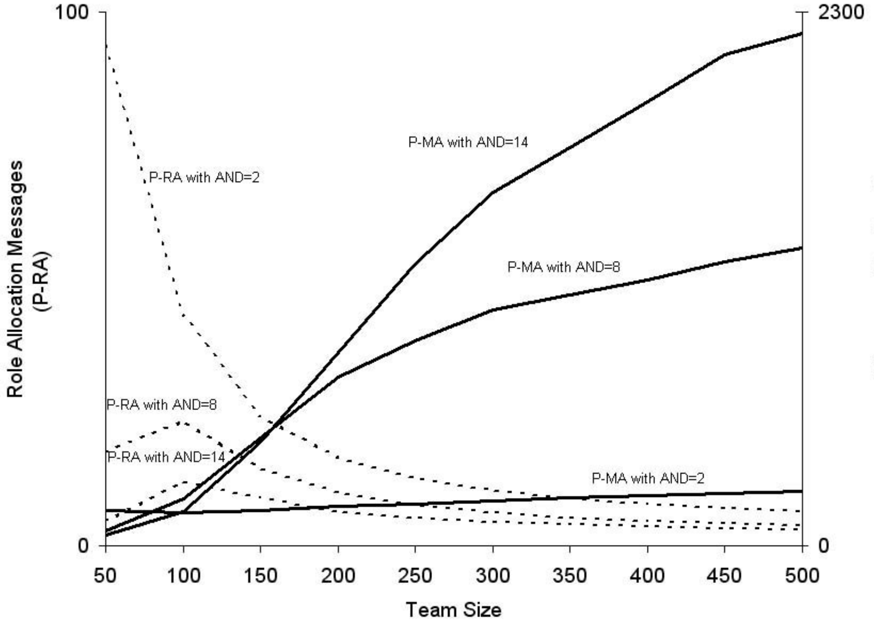


Figure 1.1: An example of relationship between 2 configuration parameters (NTM,AND) to 2 performance measures (RA,MA) in an environment setup.

during the mission, e.g., communication networks might become congested. The configurable parameters are allowed to change. From Table 1.1, we define S and C, which are a set of all possible configurations of twelve system parameters, (NTM, NPT, RPT, TNP, PPP, RPP, PTP, NCT, PCP, NPR, PDR, AND) and a set of all possible configurations of four configurable parameters, (RTH, IRU, ISR, ITL) respectively. Possible ranges of these parameters are listed in table 1.1. Next, let us define P, which is a set of seven system performance measures, (PP, RE, MA, CD, PI, CR, RA). The sets S, C and P will depend on mission domain and coordination algorithms. We define the target function $M : M(s, c) \rightarrow P$ to indicate that this function accepts as a vector of input parameters from the set of environmental parameters ($s \in S$) and the set of configurable parameters ($c \in C$) and produces as a vector of output parameters from the set of system performances P. In other words, M is the team performance model we are trying to find. In addition, we define a user preference function as a linear equation of 7 performance measures, $f(P) \rightarrow w_1$*PP + w_2*RE + w_3*MA + w_4*CD + w_5*PI + w_6*CR + w_7*RA. Finally, with the user preference function, the aim of this work is to find $\arg\max_{c \in C} f(M(s, c))$.

For an example, in a particular system configuration: sp=(200, 20, 1, 100, 5, 5, 1.0, 10, 0.1, 0.3, 0.11, 0.03), assume that we are interested in finding the best configuration from these four configurations as follows: c_1 = (50, 0, 0.3, 3), c_2 = (10, 0, 0.3, 3), c_3 = (50, 1, 0.3, 3), and c_4 = (10, 1, 0.3, 3). In case we have the team performance model, applying the model with these configurations, we get: $M(sp, c_1) \rightarrow$(89, 76, 230, 116, 25, 365, 24), $M(sp, c_2) \rightarrow$ (91, 85, 302, 145, 28, 49, 25), $M(sp, c_3) \rightarrow$(88, 75, 265, 117, 22, 370, 21), and $M(sp, c_4) \rightarrow$ (85, 81, 242, 79, 16, 240, 18). Assume that we are only considering the first measure of performance in set P, doing so by setting w_1 = 1.0 and others are zero, then, the best configurations is c_2=(10, 0, 0.3, 3).

1.5.3 System Complexity

The major challenge to the present research is the complexity of the relationship between the configuration parameters of coordination algorithms and the performance measures for large teams. For example, with a total of 16 input parameters and their possible ranges (as shown in Table 1.1), the combination of these parameters, which imply coordination configurations for this problem, yield more than 10^{25} cases. In addition, not only does each configuration have a high uncertainty in performance caused by non-determinism in algorithms, but also different configurations have different levels of uncertainties in output performances caused by interactions with the domain. Figure 1.1 gives an example of the complex non-linear relationships that occur in coordination systems. On the x-axis the number of agents in the team is varied and on the y-axis two performance variables are measured. The 6 lines, three with solid lines, three with dashed lines, show how changes in two configuration parameters effect average performance as the team size is changed. Clearly, the changes are non-linear and even vary for different values of the same configuration parameter. For the current target coordination algorithms, there are a minimum of 16 important environmental and algorithm parameters implying 10^{25} possible configurations. Each configuration will lead to a distribution of performance, with the nature of the distributions varying from configuration to configuration. For example, some configurations will have normally distributed performance, while others will have multi-modal performance.

To determine whether it was likely to be possible to find relatively simple reconfiguration rules, decision tree induction (C4.5) [91] was used to try to capture the relationship M. With a reduced problem using 14 input attributes, only four distinctive output classes (LOW, MEDIUM, HIGH, VERY HIGH), and 30,000 cases, the result was 573 classification IF-THEN rules. On test data, these rules performed with only 74.2% correct classification. Classification into these coarse output classes is simpler than required here, yet the very large number of rules was unable to even do this reasonably correctly. From this initial experiment, it was concluded that a rich, expressive model was required to capture the complex relationships.

2.0 RELATED WORK

In this section, related research is discussed. This research is classified under different topics related to the proposed research. Most of the topics relate to modeling complex systems, as the proposed research does. First, previous research related to stochastic neural networks is described because this is the main proposed approach. As genetic algorithms are used in the model training process, some works connected with evolutionary neural networks are reviewed. Next, aspects of meta-modeling are reviewed. Then, research related to parameter tuning is surveyed. The relevant research on uncertainty in systems is presented. Finally, some robotic researches that required coordinating as a team in order to achieve their tasks are reviewed.

2.1 STOCHASTIC NEURAL NETWORKS

Stochastic neural networks are very powerful for approximating complex non-linear systems [119]. Recently, this kind of neural network has been utilized as the model of many complex systems. For example, Tian and Burrage [116] used stochastic models of neural networks in representing complicated gene regulatory networks. Krynsky et al. [59] applied the idea of stochastic models in electric distribution networks. Janer et al. [46] investigated stochastic neural networks in logic implementation. Bryant and Miikkulainen [9] introduced stochastic neural networks for multi-agent systems, especially in game control systems.

Supporting evidence for stochastic neural networks is found in the work of Arkin et al. [1]. They have demonstrated that stochastic properties play an important role in cell activities at the molecular level. Although stochastic neural networks are commonly referred to as

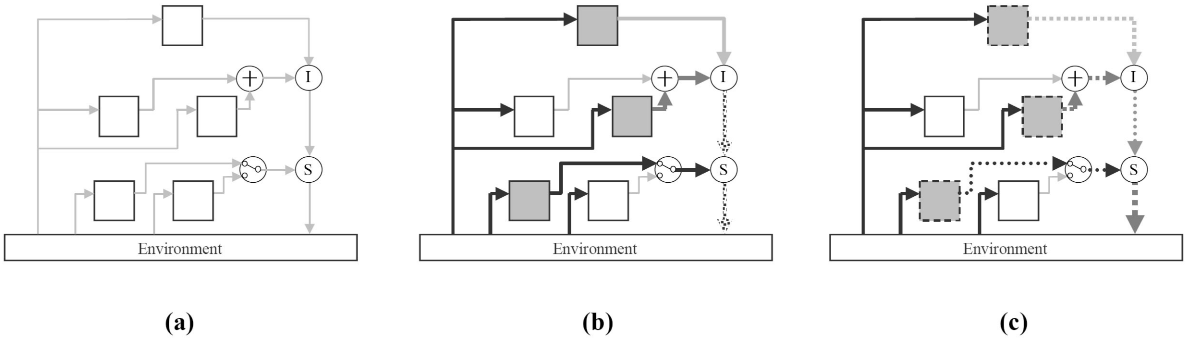


Figure 2.1: Two abstract examples of neural networks are shown. On the left, a traditional network maps input-output pairs as shown in the table below. On the right, a dynamic network maps all input patterns with all possible output patterns with different probabilities to occur. By feeding random signals through additional input nodes, it allows internal active nodes to change stochastically (depicted by dash lines). With the same input pattern, during a period of time, the network internally changes and dynamically produces all possible output patterns, which finally represents a non-deterministic model.

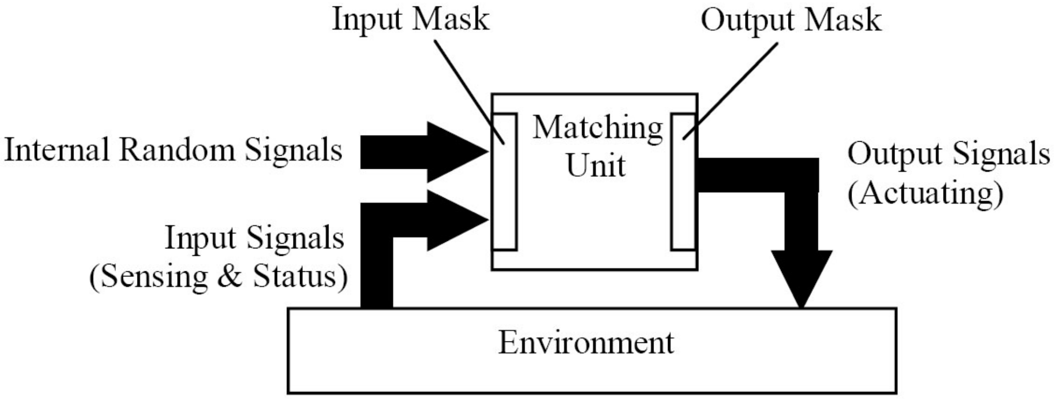


Figure 2.2: Each BM consists of three main parts: input signal unit, output signal unit and matching unit. There are two different input signals: external and internal. The external input signals usually are the sensing and status signals, but the internal signals are binomial random signals. The matching unit consists of an input mask and an output mask. An input mask is used to filter a certain subset of the input signals for matching process. The output mask is used to determine the output signals.

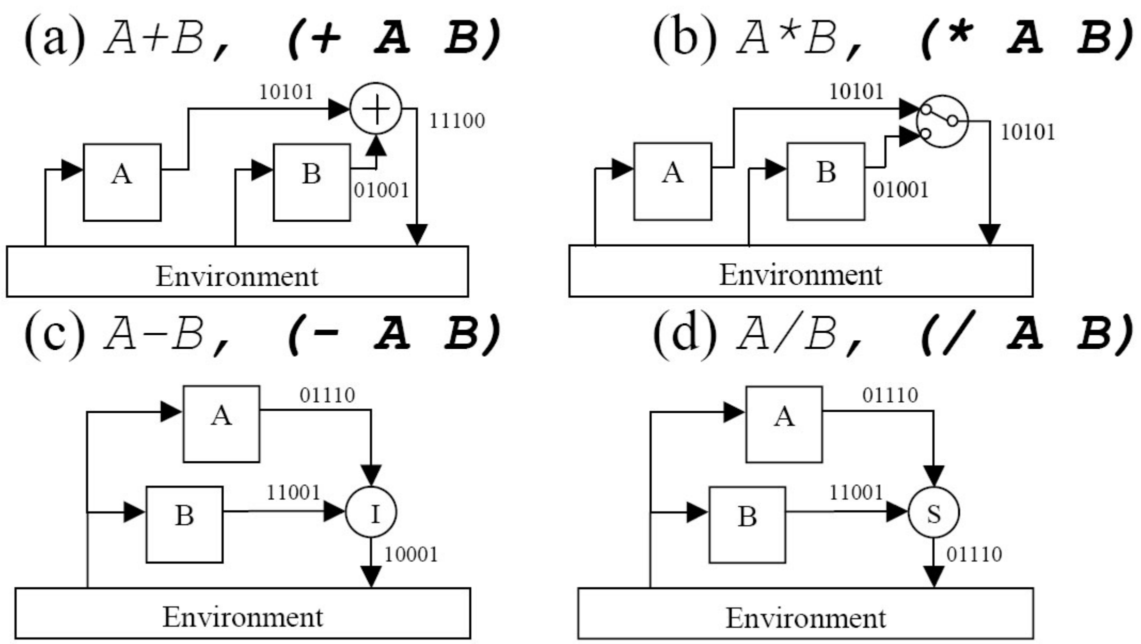


Figure 2.3: There are four BIs. (Note: A and B are represented BMs.) (a) Combination (+): output signals from two BMs are combined (XOR) bit by bit into the resultant signals: (b) Sequence ($*$): the output signals from behavior A or behavior B are passed through the output lines in turn with respect to their active statuses; (c) Inhibition ($-$): the output signals of behavior A inhibit output signals bit by bit from behavior B only when behavior A is active; and (d) Suppression (/): the output signals from behavior A suppress all output signals from behavior B only when behavior A is active. These BIs are expressed as ($+AB$), ($*AB$), (AB) and ($/AB$) respectively.

networks with stochastic components, presenting stochastic elements within neural networks has been implemented in many different ways. The most common approach is using neural networks with stochastic weights. Zhao and Shawe-Taylor [133] [134] represent the weights as a random binary sequence with the input signals. Simulated annealing is then used to train the networks by gradually reducing randomness until the network reaches the optimal state. Kim and Shanblatt [53] also applied this idea at the VLSI logic gate level by using random noise signals that make the weights change stochastically. Turchetti et al. [119] used the random coefficients in a specific kind of neural network, called the Approximate Identity Neural Network. These random coefficients perform a function similar to that of the stochastic weights. Kondo and Sawada [56] proposed the idea of stochastic logic in which random signals are presented as reference thresholds with all weight data. These random thresholds produce the signals of stochastic weights. By using a time periodic signal as a stochastic element, Nobori and Matsui [75] introduced the neural networks with stochastic resonance. In addition, these stochastic neural networks can be trained by the back propagation method, which is extended to stochastic resonance features. Patan and Parisini [80] proposed the idea of inserting a linear dynamic mechanism at the output part of the network. The two stochastic learning methods, which are called Adaptive Random Search (ARS) and Simultaneous Perturbation Stochastic Approximation (SPSA), are introduced to overcome the local minima problem. Pavlovic et al. [81] [82] introduced the stochastic Hopfield neural network by using the random threshold vectors. These random threshold vectors are used as the bias parts in activation functions of the networks. Finally, a very good review on the stochastic features of neural networks can be found in [7] and [8].

Stochastic neural networks can be implemented with any form of networks; for examples, a neural network in Figure 3.5 or a network of Behavioral Modules (BMs)[84][85] in Figure 2.1. In this case, the network of BMs is the appropriate choice to represent the robot controller. Assume that a complex behavior can be decomposed into several simpler behaviors. Therefore the target behavior is first analyzed into a network of simpler behaviors. Then the execution of these simpler behaviors and their systematic interactions will produce the target behavior. Therefore, a complex behavior can be obtained from primitive behaviors by means of composition mechanism. However, there is no general rule for composing complex behaviors from primitive behaviors and determining the needed interactions among behaviors. The concept of *Behavior Programming* (more details in [84][85]) basically consists of a network of Behavioral Modules (BMs) and Behavioral Interactions (BIs), as shown in Figure 2.2 and Figure 2.3 respectively. BMs and BIs are used as terminals and functions in a LISP-like programming language respectively. A program usually represents a structured control system, as shown in Figure 2.1. BMs are designed to represent the simplest controller architectures as primitive behaviors. A BM contains simple structures and processes for linking sensing with actions. Each BM senses the world from its particular point of view and extracts only the information from the sensors of its concern in order to issue actions. BIs are designed to internally link all BMs together. Analogically, if we refer to BMs as variables, the BIs are referred to as operations. As an automatic programming, Genetic Programming methodology is introduced to manipulate these programming structures and carry out target robot controllers in an adaptive way without explicitly programming it. Regarding to the assigned task, complex behavior systems can be created automatically from combinations of a number of simple or primitive behaviors. The terminal set consists of all possible combinations of BMs. All BIs are the members of the function set. With availability of simulation of robotic platform and data recorded from the physical robot, candidate robot controllers are generated. Then, the candidates are tested with the real robot for physical evaluation. Improving the performance of the candidate controllers can be done by updating simulation information and physical data. In addition, to learn complex behaviors, learning process may be built up from simple behaviors to final behaviors hierarchically.

2.2 EVOLUTIONARY NEURAL NETWORKS

To solve non-deterministic problems, using artificial neural network with standard back-propagation technique [71][97] is considered not possible[111]. Stochastic approximation techniques are typically accepted to be more effective to cope with the problems. Especially, evolutionary computation approaches are the most prominent one. Many researchers have shown that using evolutionary algorithms for training or designing neural networks is an effective combination. Gonzalez-Seco [38] and Sun et al. [114] have implemented neural network controllers in which the architectures of the controllers are fixed and the network weights are evolved by genetic algorithms. Sato and Nagaya [100] demonstrated that recurrent neural networks generated by an evolutionary algorithm can be trained to represent behaviors of chaos dynamics.

Many efforts have been applied evolutionary neural networks as tools for decision support system and data mining. A decision support system was developed with the combination of neural networks and genetic algorithms for prediction the future highest and lowest values of Japan's stock exchanges[4]. The system made effective predictions with the simulation experiments. Moreover, in order to extract rules from a neural network, genetic algorithms were used to prune the connections between inside nodes that are not significant toward the outputs of the model[132]. Then, the extracted rules were converted from the pruned neural model. This approach made possible to understand the activities of the networks toward solving the designed problem. As a data mining process to extract rules from databases[130], neural networks were used with genetic algorithms for constructing a feature extraction network. Genetic algorithm was applied for selecting the best sets of features for training a neural network with backpropagation. Then, classification rules were extracted from the trained neural network.

Various works have tried to enhance the effectiveness of evolutionary neural networks by proposing new approaches for very complex problems. A modified three layer neural network[61], in which all hidden nodes are interconnected, was used to learn hand writing patterns for electronic books. This new architecture of neural networks enhances the network abilities for handling various numbers of input data with different hidden neural parts in a network. Because of the complexity this neural network, genetic algorithms were utilized for searching over all possible network parameters for an appropriate network. To predict behaviors of a time series system[74], neural networks were trained with various periodic time series data. The width of repeated time priors was used to determine the input layer of the network. The best neural network was generated by the genetic algorithm process. The final model was used for forecasting the learned data series. An new approach called Hybrid Taguchi Genetic Algorithms[118]was adapted the signal to noise ratio technique for selecting optimum offspring. Results showed that this approach offers more robust and fast converse in tuning the structure and parameters of artificial neural networks.

Several works have tried to improve the efficiency of evolutionary neural networks by altering the standard procedure of genetic algorithms. Srivastava et al. [112] proposed a new genetic operator in which three individuals are selected to perform multi-point crossovers [112]. By increasing the search performance of the evolutionary process, experimental results show that this new operator outperforms traditional operators. To improve the local search capability of genetic algorithms in training neural networks, a Guassian mutation[34]was proposed. This genetic operator diffuses the current values with normal distribution with zero mean. In addition, for effectively coding any artificial neural networks into the form that can be used by genetic algorithms, an optimum binary coding [5] for artificial neural networks was designed allowing various binary operations for generating valid networks. Evolving neural network topology and parameters with hierarchical genetic algorithm was proposed and tested successfully in solving complex problems[128]. This approach encodes a neural network into a chromosome of three-level genes: layer, neural, and parameter genes. These three levels represent the topology of network. The weights in parameter genes are also evolved at the same time. Pujol and Poli[89] used a graph representation for network structure. A chromosome consists of two types of genes: function gene (neural) or terminal gene (variable). However, a work of Siddiqi and Lucas[109] has demonstrated that using directly encoding of neural networks can produce the same effective results.

A lot of works were utilized an evolutionary process only to supplement the implementation of neural network applications. An combination of neural networks and genetic algorithms was implemented to improve a transformer production[26]. First, the neural network was trained for predicting iron loss of transformers. Then, genetic algorithms searched for reducing iron loss and generating an optimization group of transformer cores. The process of genetic algorithm was adapted to help back propagation neural network improving learning performance for recognition of Chinese speeches[62]. Pratap et al. [88] integrated genetic algorithms and neural networks in designing and modeling multi-layer RF passives in two stages. First, the neural networks were utilized in the modeling process. Then, the genetic algorithms were used in the optimization process to determine the preferred electrical characteristics of the circuit boards.

Genetic algorithms were used in many applications for searching over all possible neural network topology and parameters that accurately create an optimal neural network model, e.g. for solving a nonlinear digital driver circuit. [96], for determining an error compensation in welding robot[122] and for estimating of the condition of power transformers[44]. Following are various applications that were applied with evolutionary neural networks. To design a neural network controller for control of flexible-link manipulators [98], modified genetic algorithm processes were validated successfully in simulation studies. To solve the analysis of electronic circuits[125], neural networks were used with genetic algorithms for modeling the PN junction diodes, in which they possess nonlinear and linear characteristics. To create an accurate nonlinear mathematical model of switched reluctance motor[123], evolution neural networks showed quicker convergence than backpropagation neural networks. In a digital image processing task[99], evolutionary process was used to train neural networks to classify objects in images. To predict traffic flow[115], a genetic neural network was applied for generating a simple and accurate forecasting model for short-term traffic flow simulations. To model microwave systems with neural networks[120], genetic algorithms were utilized for determining interconnection models, weights, and transfer functions for helping the design of the final neural networks. For helping doctors in prediction of peritoneal dialysis, a genetic algorithm embedded in neural network (GAENN) trained with experimental data has shown better performance than standard neural networks[131]. In robotic control system, a hybrid process of neural network and genetic algorithms with multi objective optimization techniques were applied effectively in simulations for time-optimal control [13] and friction compensation [14]. To solve the English character identification problem, Hintz and Spofford [42] made use of genetic algorithms in creating neural networks almost from the outset. In their work, nearly every neural network's elements including number of nodes, structures of interconnection, weights, and biases are evolved by a genetic algorithms process. Finally, an approach called Neurogenetic model [50] was effectively applied to model the muscle EMG-torque relation in a human motor control task.

An interesting approach toward challenging complicated tasks, co-evolutionary genetic algorithms [41, 107] have been demonstrated promising results to solve complex problems, see [11]. Co-evolutionary genetic algorithms use either competitive [95, 49, 66] or cooperate [87, 17] interactions of two or more populations to maintain diversity in the populations. These populations usually are different degrees of difficulties. However, resource sharing mechanism between populations is verified to be the main reason for encouraging variety in populations [124].

2.3 META MODELS

To model a complex system is a very difficult task, especially when the system has components and characteristics of which the structure and behavior cannot be described in a single comprehensive format. Meta-models are models that represent classes of models [101]. Although most of meta-models are in formats or structures different from the original models, the meta-model is also considered to be a model [121]. The concept of a meta-model is used in many different fields. Scalzo and Roth [101] used a meta-modeling technique as a framework to improve a process of IEEE standard development. Another framework of meta-modeling is utilized in designing business processes, especially in cooperative decision-making [94]. Caeiro-Rodriguez et al. [12] developed a meta-modeling system as a coordination model in educational learning systems. Finally, as commonly known, XML is a meta-modeling framework in web technology that exists as a neutral model exchange format [121].

A few efforts have been made to utilize neural networks as the meta-modeling method. Meghabghab and Nasr [69] utilized the RBF neural networks as meta-models for computer simulations. These simulation models, designed in industrialized domains, are complex and stochastic. An iterative approach is used as the learning method during training to adjust the RBF networks' weights, centers, and spreads. In addition, two test sets are used for termination and evaluation processes. Cohen and Hudson [21] developed a meta-model of neural networks in a form of meta-knowledge targeting for medical decision systems. However, this system requires that domain experts design the hierarchical structure of the model.

Much research has been done to develop standards and frameworks for facilitating the meta-modeling approach. A selection of these follows. UML and OML (see [2]) are standards of modeling languages, which other work has been based on. AALAADIN [30] is designed to describe task organizations in multi-agent systems. EKD [94] is designed to describe decision-making systems. ATOM3 [121] is designed to describe complex system simulations.

2.4 PARAMETER TUNING

Selecting values for all necessary parameters has been essential in controlling complex systems, especially when the relationship between a parameter change and the corresponding change in performance is non-linear, time-varied, and uncertain. For this reason, manually selecting parameters such that a desired output pattern emerges is quite impractical. Many studies have focused on parameter optimization. Kwong and Jacob [60] used an evolutionary approach to discover the desired swarm formations from a list of a swarm simulation's parameters. Users are allowed to explore a small population of swarm behaviors, each of which represents a particular set of system parameters created by the evolutionary process. By assigning score points to those swarm behaviors, designed swarm behaviors emerge at the end. Santos Coelho et al. [20] used a hybrid learning algorithm for parameter tuning in controlling a robot manipulator. The hybrid approach consists of using an evolutionary algorithm for global optimization and using a simulated annealing algorithm for local optimization. Kim et al. [52] applied an optimization method, called Sequential Quadratic Programming (SQP), for AVR (Automatic Voltage Regulators) parameter tuning.

Several researchers have studied parameter tuning for improving the performance of machine learning algorithms. To solve the traveling salesman problem, Ramos et al. [92] utilized a statistical approach to tuning an evolutionary algorithm's parameters. Yuan and Gallagher [129] applied a meta-learning approach to parameter tuning in evolutionary algorithms. Krink and Ursem [58] proposed a hybrid concept of a multi-agent system and a special kind of evolutionary algorithm. In this learning model, an agent, representing a possible solution, can move along the space of an evolutionary algorithm's parameters such that the parameters are converted into part of the learning process. For improving the learning performance of any induction algorithms, Yang and Wu [126] proposed a way to effectively tune parameters by using the feature elimination approach. Guo and Uhrig [40] used genetic algorithms to select a proper set of input variables that yield good results when these parameters are used with neural networks in monitoring a nuclear power plant.

For determining the optimal initial process parameter settings for plastic injection molding production, an optimization process involved with neural networks and genetic algorithms was used/citeChen2007. First, the neural network was trained and tested by backpropagation method. Then, genetic algorithm process was utilized for searching the best parameter settings. The approach produced successful results for optimizing process parameters.

2.5 UNCERTAINTY MODELING

A complex system generally consists of two types of information: required information for describing the system and needed information for resolving any uncertainty in the system [3]. Belcastro [6] used this general distinction to separate his multi-variable control systems into two models such that an uncertainty model and a nominal model are utilized together for representing nonlinear systems. In the genetic regulatory field, Arkin et al. [1] have demonstrated that the uncertainty of gene expression reactions play an important role in cell activities at the molecular level. In order to understand these complex biochemical systems, many researchers are investigating how to precisely model the systems. Shmulevich et al. [108] used probabilistic Boolean networks to model uncertainty for gene regulatory networks. Smolen et al. [110] and De Jong [48] reviewed many modeling approaches and simulations in gene network modeling. In the problem of autonomous mobile robots, Stone et al. [113] used a robotic system that took uncertainty into account for all components. As a result, running with real robots in dynamic environments yielded effective performances. To model uncertainty in general, but especially in both data mining and statistical aspects, a good fundamental review can be found in Chatfield [16].

2.6 COORDINATION ROBOTIC TEAM

Based on observing police SWAT teams in four training exercises, Jones and Hinds[47] presented a new design of coordinating system for team of robots. They found that the critical basis for coordinating system was establishing common ground among team members. Team leaders have to form overall models of team status and build shared situation understanding among distributed team members. Moreover, using dialogue interaction between team members was also importance. Grabowski et al.[39] presented the design of a team of small, heterogeneous, resource-limited robots that work collaboratively to explore and map unknown areas. Since the knowledge of the position and orientation of each robot is important to achieve accurate mapping and exploration of the environment, a new localization has developed by utilizing dead reckoning and sonar data from multiple robots to compute the position and orientation of each robot. A series of experiments have been conducted to test the effectiveness of a team of Millibots to explore and map a given area. Parasuraman et al.[76] presented empirical evidence for the efficiency of delegation type interfaces to increase the human-robot team performance for controlling multiple robots in the RoboFlag capture-the-flag simulation. This delegation type interfaces allow flexible control for multilevel interface with only small increase in human mental workload. The results showed that the multilevel interaction provided by this flexible control system allowed for effective coordination between human supervisor and robots, as confirmed by numerous successful executions in the experiments. Miller and Parasuraman[70] developed a flexible interaction interface called Playbook, designed to provide the flexibility by providing a supervisor and subordinates the ability to collaborate flexibly about delegating tasks at different levels and instructions to perform those tasks. This interaction presents several advantages for human-robot collaboration such as increased situation awareness, decreased human skill condition, balanced mental workload etc.

Madhavan et al.[32, 68] proposed an approach for cooperative localization and terrain mapping for a team of robots operating in rough outdoor environments. First, depth ranges to a number of objects in the area are calculated from vision depth information. Then, the robots' relative positions are connected with vision depth information. Finally, the terrain map is updated by combination of vision-based depth information and fusing elevation gradient data. Parker et al.[79] proposed a behavior-based control approach for multi-robot systems to solve the site preparation task on Mars. Cooperative control framework called ALLIANCE was used to enable team of robots to automatically perform appropriate actions effectively even in uncertain conditions. The efficiency of this cooperative control approach is established on the concept of motivational behaviors, based from two internal measurements of evaluation its own current task performance and evaluation other robots performance associated with their tasks. Parker[77] demonstrated an approach to allow a heterogeneous group of robots to adapt their actions while environmental conditions are changing over time. L-ALLIANCE architecture was proposed with learning capability for parameter adjustment that makes possible for the robots to adapt their behaviors over time in response to dynamic changing of team capabilities and environment. In this work, this approach of dynamic task allocation was demonstrated with real heterogeneous robots on cooperative box pushing experiments. However, the success of this architecture relied on the presence of global communication between robots, so that the system can precisely exchange knowledge among team members. Parker[78] demonstrated a distributed approach using the potential field technique for cooperative team of robots to track multiple moving targets by considering actions to keep moving objects under surveillance. The ALLIANCE architecture is used to provide mechanisms for fault tolerant cooperative control, and allows robot team members to adjust their actions based upon the actions of their teammates. The results show successful in achieving the target task. However, in order to get a good solution, the system required considerable tweaking of parameters.

Many researches have been focused on developing system architectures and frameworks. Chamberlain et al.[15] presented an initial idea of a control architecture that allows the humans to conduct strategy descriptions based on lower-level strategies. Dixon et al.[24] developed a software framework with graphical user interface that allows the abilities to explore rapid configuration and designing of cooperating teams of heterogeneous robots (real or simulated). This software framework also provides dynamic linking of libraries that allow the same robot code to execute on a simulated or real-world robot. The virtual simulation environments provide realistic mobility characteristics, sensory data, and tactical behaviors for each vehicle. They points out that a significant amount of low-level infrastructure is required to support the communication, control, and simulation of multiple agents in high-fidelity virtual environments. Gerkey and Mataric[35] have proposed a fundamental framework for analyzing any task allocation approaches in multi-robot systems. Taxonomy of multi-robot task allocation problems has been proposed as well, based on three aspects of robot capabilities, task requirements and nature of available information. This taxonomy is quite useful in the way to classify types of task allocation problem. In addition, in order to be able to analyze the task allocation problem, the utility function is introduced by projected from the difference of anticipated quality of task and anticipated resource cost. Fu et al.[33] developed an authoring tool for modeling intelligent agent-based behaviors. Familiar visual representations, which are easy to understand, are used to describe complex behaviors and also allow novices to be part of the process of behavior authoring.

Many researches in human-computer interaction have been studied how effective to include human into the system. A mediation hierarchy of human supervisory in robotic control system has been defined by Adams and Paul[45]. The intention of this hierarchy is to allow the human supervisor to interact with all levels of system, when necessary, in order to maintain the system in a stable state. Four hierarchical levels of supervisory interaction between human and robots have been proposed. The highest level of interaction allows the supervisor to specify the proper set of actions to complete the task. The next level, the supervisor has different ways to interact with robotic agents for controlling, monitoring, or providing information. Next, the process level permits the supervisor to provide assistances when the agent is unable to make a proper action or a decision. The lowest level is the data level, which permits the supervisor to reconfigure the system, when need, and also make sure that the system uses the correct data for the processing. Basically, this hierarchy was designed to provide human with the ways to interact with all levels of a multiagent system while permitting the agents to work autonomously until supervisory interaction is need. Endo et al.[28] conducted a research in a usability evaluation of a robot mission-planning wizard for a hybrid deliberative and reactive control system. By simplifying the user planning process, the results reported that the wizard helped reducing planning time and increasing accuracy of the plan. The guidelines for how to make usability experiments in robot mission planning system also introduced. Crandall et al.[22] demonstrated how estimates of neglect tolerance can be used to help a designer to create a system with multiple robots/tasks that a single human can manage. Basically, neglect tolerance refers to neglect time, time that a robot can be ignored before reaching an unacceptable performance, and interaction time, time that a human must interact with a robot to get maximum performance. This estimate methodology can be used to identify if a particular team configuration is sufficient to maximize performance or predict the performance of a robotic team controlled by a human operator. Fong et al.[31] presented a significant research on human interactions with a team of two autonomous robots by allowing human and robots work together as a collaborative control. Dialogue interactions between human and robots are introduced as a way to naturally get human's attention. In addition, waypoint-based interface has been adapted for controlling robots. This kind of control interface comes with some advantages of an easy to use, small bandwidth, and suitable for time delay. This work was also designed for reducing the burden of task monitoring from human resulted in better performance.

3.0 ALGORITHMS AND APPROACHES

An approach has been developed to facilitate users' understanding and manipulation of the relationship between configurations of coordination algorithms and team performance. This approach has several steps. Figure 3.1 shows the outline of the approach. It begins with the process of collecting large data sets generated by our abstract teamwork simulation. Simulation clients are executed generating training data, in which they are collected centrally at a database server. Then, at the server, collected data is preprocessed so that it is in the right form for use in the learning process. To model the *team performance model*, a novel type of *stochastic neural network* is utilized, in which dynamic features are added to the input layer allowing any non-deterministic system to be modeled. The stochastic neural networks capture randomness from the additional input nodes fed with internal random signals. These random signals, combined with weights between the additional nodes and the hidden nodes, allow stochastic output even though the network is deterministic. To train stochastic neural networks for representing uncertainty of a very complex system, an adapted approach of genetic algorithms is used with a new technique of fitness function called distribution comparison. Next, by means of an evolutionary computation approach, the learning process utilizes the collected data sets to generate team performance models. In addition, team control interface is designed as an intelligible tool allowing users to use team performance models for rapid configuration exploration. The interface is also utilized to connect with the target system for actual interactions. Finally, experiments are performed using the team control interface with the simulation to investigate the performance of the team performance model. In the following sections, each of these steps is described in additional detail.

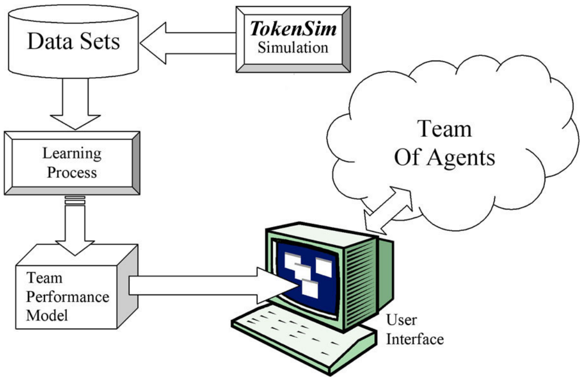
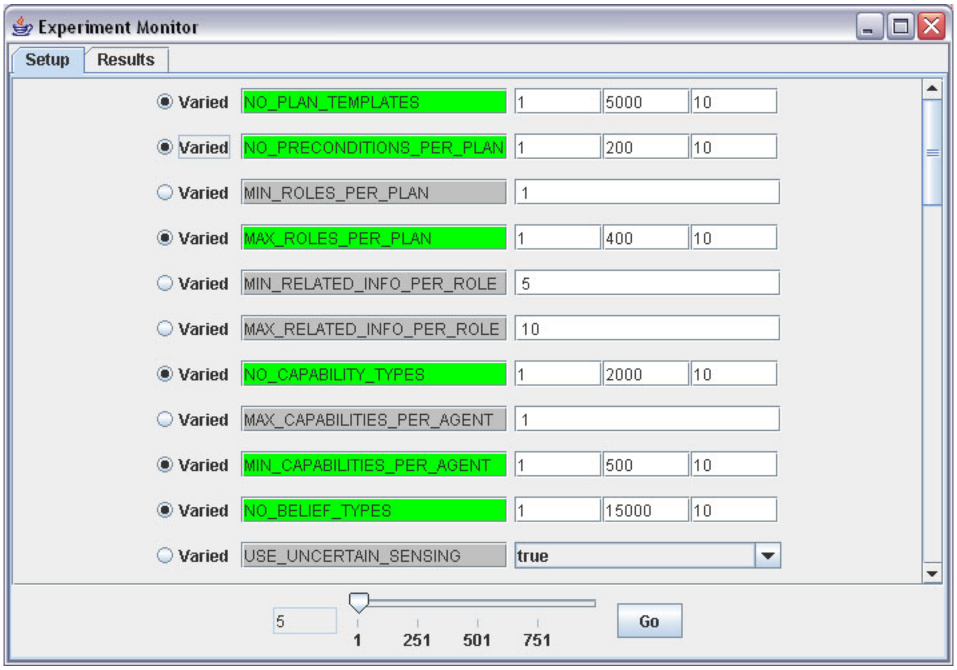


Figure 3.1: High level view of overall process.



● Varied	NO_PLAN_TEMPLATES	1	5000	10
● Varied	NO_PRECONDITIONS_PER_PLAN	1	200	10
○ Varied	MIN_ROLES_PER_PLAN	1		
● Varied	MAX_ROLES_PER_PLAN	1	400	10
○ Varied	MIN_RELATED_INFO_PER_ROLE	5		
○ Varied	MAX_RELATED_INFO_PER_ROLE	10		
● Varied	NO_CAPABILITY_TYPES	1	2000	10
○ Varied	MAX_CAPABILITIES_PER_AGENT	1		
● Varied	MIN_CAPABILITIES_PER_AGENT	1	500	10
● Varied	NO_BELIEF_TYPES	1	15000	10
○ Varied	USE_UNCERTAIN_SENSING	true		

Figure 3.2: Interface of TokenSim.

3.1 ABSTRACT SIMULATION

TokenSim is an abstract teamwork simulation developed by the Intelligent Software Agents Lab at Carnegie Mellon University to investigate coordination algorithms in cooperative multi-agent teams within simulated environments. The capabilities of this simulation consist of the most fundamental features of coordination algorithms including, role allocation, resource allocation, task planning, sensor fusion, and information sharing. TokenSim abstractly omits the effects on environment from consideration; however, it simulates the effects on the team's coordination intervention directly. In brief, an agent acquires knowledge by perceiving through its sensors or communicating with its teammates. In the team, an agent or some agents receive uncertain sensor readings randomly at a configurable rate. At any time step, an agent is only allowed to access a particular simulated resource created specifically for the assigned tasks. Resources can be transferred with no cost and cannot be destroyed or lost. The layout of tasks are generated and distributed randomly in an open environment. In the simulation, all agents move at the same speed so that they are allowed to think and act at each time step and so the effects of their actions are abstractly simulated accordingly. Communication is provided for passing objects between agents. When a task is allocated to an agent, the agent has exclusive access to all required resources and has no cost in moving to the task location. The team receives a *reward* every time an agent is allocated a task. The reward is given while the agent is executing the allocated task, which takes only one time step. Figure 3.2 shows an interface of this simulation. TokenSim allows a number of parameters to be varied and team performance statistics to be recorded. This feature is mainly used in the data collection process. TokenSim has 49 input parameters that can be varied and 10 team performance measures.

TokenSim is the simulation that is used in creating the team performance model. The team performance model is used with the team control interface in order to help users interact with the testing scenario. Before that, a verification test is conducted to confirm how effectively the team performance model predicts team performances using the abstract simulation from which the model was created. TokenSim is used as the target system. A number of scenarios will be set to provide situations that would require automatic reconfiguration assistance in order to reconfigure the team and meet performance targets. Each scenario is executed in various environment settings so that the initial configurations differ. Team performance measures are recorded throughout the execution, providing data for result analysis.

3.1.1 Parameters and Performance Measures

In TokenSim, there are totally 49 input coordination parameters and 10 output performance measures. However, 17 input parameters related with simulation environment setting, unlimited resources, constant time step, and unlimited network bandwidth, are fixed as shown in Table 3.1 along with their preset values. The main reason for allowing every agent in the team to have all needed resources and unlimited network bandwidth is the abstract simulation, TokenSim, was primarily built to investigate how the coordination algorithms work together. Therefore, these environment parameters are abstracted out. As a result, for using in the process of modeling coordination configurations, 32 varied input parameters are available as shown in Table 3.2 along with their possible ranges. However, only 31 input parameters (excluding SIMULATION LENGTH) are used in the team performance model. Because there is a constraint that each configuration should have enough time steps to start its first plan, a flexible simulation length scheme is utilized. This scheme starts SIMULATION LENGTH with 500, and allows increasing the length every 500 steps if the first plan does not start yet until it reaches 5000 steps. As a result, the possible values of SIMULATION LENGTH are 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, and 5000. After 5000 steps, if the first plan does not start yet, no more increasing. To solve a problem that different simulation lengths produce unequal outcomes of the output performance measures, the actual SIMULATION LENGTH is used to normalize the output parameters to be in a form of per-simulation-step parameters (e.g., TOKEN MOVES per-simulation-step). Thus, all output performance measures are recorded in the form of per-simulation-step parameters. In this work, although there are 10 output performance measures as shown in Table 3.3, only 4 output performance measures: TOKEN MOVES, PLANS STARTED, REWARD, and UNIQUE SENSOR FUSION, are exploited. One reason is that measur-

Table 3.1: List of fixed parameters

#	Varied Parameters	Value
1	MIN TASK DURATION	0
2	MAX TASK DURATION	1
3	NO RESOURCES	0
4	MIN REQD RESOURCES	0
5	MAX REQD RESOURCES	0
6	MIN USEFUL RESOURCES	0
7	MAX USEFUL RESOURCES	0
8	RESOURCE TOKEN TYPE	STATIC
9	AVG INTERCHANGEABLE RESOURCES	5
10	GEOGRAPHY TYPE	NONE
11	ENV X DIMENSION	500
12	ENV Y DIMENSION	500
13	AGENT BASE SPEED	1
14	QUIET	TRUE
15	NETWORK BANDWIDTH	-1
16	ROLE TOKEN TYPE	LADCOP
17	TIMESTEP SIZE	1

Table 3.2: List of varied parameters

#	Varied Parameters	Min	Max
1	NO PLAN TEMPLATES	1	50
2	NO PRECONDITIONS PER PLAN	1	5
3	MIN ROLES PER PLAN	1	5
4	MAX ROLES PER PLAN	1	15
5	MIN RELATED INFO PER ROLE	0	10
6	MAX RELATED INFO PER ROLE	1	25
7	NO CAPABILITY TYPES	1	100
8	MIN CAPABILITIES PER AGENT	1	5
9	MAX CAPABILITIES PER AGENT	1	50
10	NO BELIEF TYPES	1	1000
11	USE UNCERTAIN SENSING	FALSE	TRUE
12	BELIEF OCCUR RATE	0.0001	0.01
13	BELIEF SENSE RATE	0.0001	0.5
14	READINGS FOR HIGH CONFIDENCE	1	20
15	READINGS PRODUCED	1	100
16	NO AGENTS	8	500
17	BELIEF TOKEN TTL	1	800
18	SENSOR TOKEN TTL	1	800
19	MESSAGE LOSS RATE	0	0.1
20	LADCOP INITIAL THRESHOLD	0	0.99
21	SIMPLE ROLE TOKEN MOVE LENGTH	1	500
22	ROLE TOKEN SLEEP RULE	NEVER	FIXED
23	ROLE SLEEPS AFTER HOPS	20	1000
24	UNFILLED ROLE SLEEP TIME	20	1000
25	NETWORK TYPE	”0”	”5”
26	NETWORK DENSITY	2	8
27	ROUTING MODEL TYPE	P	RANDOM
28	PROBABILISTIC P CHOICE	FALSE	TRUE
29	REL FOR PRECOND SAME PLAN	0	1
30	REL FOR INFO FOR PRECOND FACTOR	0	1
31	REL FOR ROLES SAME CAPABILITY	0	1
32	SIMULATION LENGTH	500	5000

Table 3.3: List of output parameters

#	Output Parameters
1	TOKEN MOVES
2	BELIEF TOKEN MOVES
3	ROLE TOKEN MOVES
4	JI TOKEN MOVES
5	RESOURCE TOKEN MOVES
6	PLANS STARTED
7	REWARD
8	SENSOR TOKEN MOVES
9	SENSOR FUSION
10	UNIQUE SENSOR FUSION

ing TOKEN MOVES is enough, since TOKEN MOVES represents total sum of BELIEF TOKEN MOVES, ROLE TOKEN MOVES, JI TOKEN MOVES, RESOURCE TOKEN MOVES, and SENSOR TOKEN MOVES.

3.2 DATA COLLECTION

The initial step in modeling a very complex system is the collection of a large volume of data. This data either already exists or needs to be gathered and is stored in various forms. Data preprocessing techniques are then utilized to convert the data into a format which stochastic neural network can accept.

The process of creating and preprocessing training sets for stochastic neural networks is atypical. Because, in the learning process part, distributions of target outputs and actual outputs are required in fitness function, as is described in section 3.4. A set of distributions of target outputs is needed for using in the evaluation process. Subsequently, each set training data is made up of two subsets: a set of every input value and a set of every output distribution. A simple way for creating output distributions may be executed as follow. First, each output is partitioned its all possible range into a number of disjoint subset or slots. However, the partitioning process can be done by any means; however it has to be the same process when partitioning the output of stochastic neural networks. Then, by collecting output data with the same input values for a number of times, the distribution of the outputs is computed by counting how many times the output values fall into each slot. The same procedure is applied when obtaining the distributions of outputs for the stochastic neural networks during learning process.

3.2.1 How Much Data Is Enough?

In this work, we are trying to model team performance for coordination algorithm configuration of large multi-agent teams. The major difficulty is the complexity of interactions between parameter setting in the system. With more than 32 input parameters and their possible ranges, the combination of these parameters, which imply coordination configurations for this problem, are very huge: more than 10^{32} cases. In addition, not only does each configuration have high uncertainty in performance caused by non-determinism in algorithms, but different configurations also have different levels of uncertainties in output performances caused by interactions with the domain. Changing in one parameter may affect in changing in performances non-linearly in various ways.

Because the space of all eligible configurations is very huge and we can only work with relatively small portion of it, it would be nice if we know how many samples are reasonable to statistically represent the system properties. The possible approach is reducing the size of the data in such a way that the reduction does not change the result drastically. Fortunately, working with a smaller set of sample data is applicable, for the reason that roughly optimal solutions are acceptable in many machine learning applications.

The most widely used methods to determine appropriate sample size for given accuracy and confidence parameters use Hoeffding bounds (or the additive Chernoff bounds). The good examples can be found in [55], [117], [19] and [25]. The Hoeffding bounds characterize the deviation between the true errors and observed errors over m independent trials, as

$$P\Big\{Error_{true} > Error_{m-trials} + \varepsilon\Big\} \leq e^{-2m\varepsilon^2} \tag{3.1}$$

This characteristic gives us a way to assure that any consistent learner using a finite hypothesis space H, with probability $1-\delta$, output a hypothesis within error ε of the target concept, we have

$$m \geq \frac{1}{2\varepsilon^2}\left(ln|H| + ln(\frac{1}{\delta})\right) \tag{3.2}$$

However, this equation can not be applied to infinite hypothesis spaces. [72]

For neural networks, [72], especially Perceptron networks containing s Perceptions, each with r inputs, we can bound the number of training examples sufficient to learn (with probability at least $1-\delta$) any target concept within error ε. We have

$$m \geq \frac{1}{\varepsilon}\left(4log_2(\frac{2}{\varepsilon} + 16(r+1)slog_2(es)log_2(\frac{13}{\varepsilon})\right) \tag{3.3}$$

However, this approach fails to apply for Backpropagation networks. [72]

The method that is employed calculating the number of samples that will guarantee a desired accuracy in an estimate $\hat{\sigma_x}$ (calculated based on the samples) of a parameter σ_x called the support threshold. The parameter σ_x is a measure of the prevalance of a pattern given by x in the space of configurations T, in which $T = S \times C$ (S and C are the sets of all possible configurations of the environment, and a set of all possible configurations of the coordination algorithms, respectively.) Where $x \subset t$ for $t \in T$. This is a useful way to measure the accuracy with which a set of samples L represents the configuration space T because we want important patterns to occur in L with the same frequency that they occur in T and hence we want low error in the estimate $\hat{\sigma_x}$.

Formally, Let $t = \{t_1, t_2, t_3, \cdots, t_m\}$ be a set of configuration parameters and T be the set of all possible configurations. Each configuration $t \in T$ is a set of values of all parameters. Given a sub-configuration x and a configuration t, we say that t *contains* x if and only if $x \subseteq t$. We define σ_x as the number of configurations in T that contain sub-configuration x. The probability that a configuration t randomly selected from T contains x is $p_x = \sigma_x/|T|$.

Lee et al use this to determine a sample size[63] $m = |L|$ that will guarantee a desired accuracy in the estimate $\hat{\sigma_x}$ obtained from samples. The analysis of Lee et al. is valid for $\sigma_x < |T| \times sp\%$, where sp is a parameter which gives an upper bound on the prevalence of a pattern x in the space of configurations T.

Let α be a variable used to control the desired accuracy in the estimate of σ_x such that there is a $100(1-\alpha)\%$ chance that the estimate is correct (lay in a confidence interval). Furthermore, if we assume that the error in $\hat{\sigma_d}$ is normally distributed, let $z_{\sigma/2}$ be a critical value of $\hat{\sigma_x}$ such that the area under the error curve beyond $z_{\sigma/2}$ is exactly $\alpha/2$. With the parameters of importance thus described Lee et al. showed that the number of samples to obtain the desired accuracy in $\hat{\sigma_x}$, is given by:

$$2|T|z_{\alpha/2}\sqrt{\frac{sp\%(1-sp\%)}{m}} \leq \frac{|T| \times sp\%}{5} \tag{3.4}$$

or

$$m \geq \frac{(10z_{\alpha/2})^2(1-sp\%)}{sp\%} \tag{3.5}$$

With independent of $|T|$, the sample size can be estimated from a user-specified upper bound support threshold of $sp\%$ and the value of α.

For example, with $sp = 2\%$ and $\alpha = 0.05$, if we randomly select $19,000$ configurations from all possible eligible configurations, so that there is 95% chance that estimation of proportions of any sub-configuration $x \subseteq t$ would lay within the confidence interval (which has the width of upper bound support threshold less than 2% the total possible space). Likewise, with $sp = 5\%$ and $\alpha = 0.05$, if we randomly select $8,000$ configurations from all possible eligible configurations, so that there is 95% chance that estimation of proportions of any sub-configuration $x \subseteq t$ would lay within the confidence interval (which has the width of upper bound support threshold less than 5% the total possible space). Since this number is independent of the size of all possible configurations, a sample set around $19,000$ configurations are big enough to represent the entire system with the preferred accuracy.

Above methods are so called the batch sampling methods, in which the sample size is calculated before execution. Usually, the sample size is overestimated to ensure against the worst case situations. To get a tight sample size, an on-line sampling is proposed by [25]. The samples are obtained adaptively depending on its current situation.

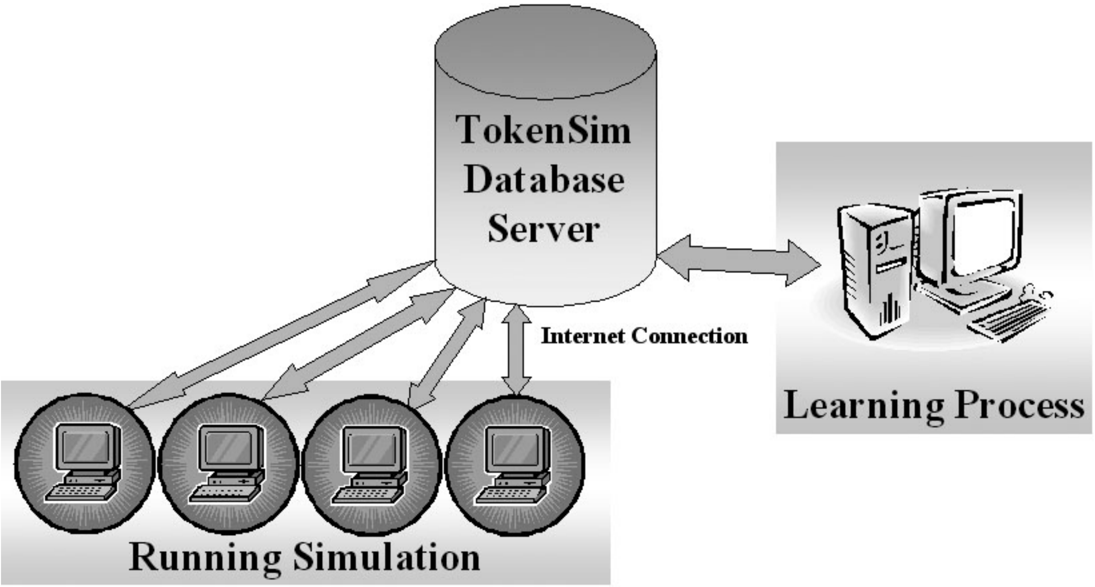


Figure 3.3: A TokenSim database server provides a centralized data storage for data collection and learning process.

3.2.2 Online Database

A database server[1] of TokenSim is designed to facilitate data collecting process. Allowing for running TokenSim clients on remote machines, as shown in Figure 3.3, all results are stored at a central database for further data preprocessing. Statistical analysis and data distributions of output performance are computed at the server before utilizing by learning process machines.

3.2.3 Collecting Procedure

To create a training data set, data are collected by running TokenSim clients on many computers. All results are stored in the central database server. Then, data preprocessing is performed on the server transferring collected data into a right form for the next process

[1] The database server was originally implemented by Gregor Kronenberger. For more details, please visit https://athiri.cimds.ri.cmu.edu/twiki/bin/view/Machinetta/TokenSimDatabase

of model learning. The details of data collection process are described as follows.

1) Create a small set of coordination configurations for executions This process has to be uniformly distributed over all possible parameter space. The method is done by dividing parameters into two groups. The first group consists of all enumerated, boolean, or small number parameters, e.g., USE UNCERTAIN SENSING, ROLE TOKEN SLEEP RULE, NETWORK TYPE, ROUTING MODEL TYPE, PROBABILISTIC P CHOICE. The second group consists of real value or large number parameters, e.g., NO PLAN TEMPLATES, NO BELIEF TYPES, NO AGENTS, BELIEF OCCUR RATE, etc. Next, a small set of configurations is created, relying mainly on the parameters in the first group. The set contains a list of configurations for all combination of the possible values of the first group. This number of configurations is not a big number. For example, if the first group parameters are listed above, there are only 96 possible configurations in the set. Thus, in each configuration in the set, the configuration contains fixed values from combination of the first group and the rest of parameters in the second group are selected by randomly from within their possible ranges. This way, every time the set is created, it contains the same lists of combination in the first group and uniformly random values in the second group.

2) Run a coordination configuration 30 times A coordination configuration in the set is executed with a TokenSim client for 30 times. Each time, all the results are stored at the database server. However, since there is a limited time for data collection process, some configurations that spend a very long time to finish must be terminated. Allowing a very long process to run 30 times is undesirable. Therefore, the process of checking duration is performed on the first execution on every configuration. If the first execution is quicker than a time threshold, the execution proceeds through for 30 times. Otherwise, that configuration is terminated. In this work, depending on the speed of machines, different thresholds are used, e.g., 5 minutes, 10 minutes, and 30 minutes.

3) Do step 2 until the set of coordination configurations is finished If there are more configurations in the set of coordination configurations, the next configuration is executed in step 2.

4) Execute step 1-3 many times until reaching the target number of training data To have an enough training data set, processes in step 1-3 are repeatedly executed. For example, with a small set of 96 configurations with some possible terminated configurations, it normally requires more than 300 iterations to get 20000 eligible training data. This step can be executed in parallel on different running computers.

5) Execute data preprocessing on database server After finishing collecting the data, a data preprocessing routine is called on the database server to transform all 30 recorded data in each configuration to a data distribution form that can be used in the learning process. Generally, each output performance measure is divided into a possible range of 100 slots. Then, the 30 recorded data are distributed among these slots. Finally, frequencies in the slots are used as the output data distributions. These output data distributions are stored on the database server, so that learning machines are able to access and use them in the future.

3.3 STOCHASTIC NEURAL NETWORK MODELS

A standard artificial neural network is sometimes referred to as a black box, which is able to forecast an output pattern from a given input pattern. Normally, the generated output patterns are static, which means that unless the observed input patterns change, the output patterns does not change at all. Therefore, a traditional neural network usually limits its applications to deterministic problems, where the outcomes of any possible input pattern can be calculated with certainty. However, most complex problems are non-deterministic, in which the outcomes are uncertain even with the same input patterns. To make it possible for a traditional neural network to represent non-deterministic relationships, the network is augmented with extra input nodes supplied with internal stochastic signals. As shown in Figure 3.4 a, a simple neural network consists of five input nodes, two hidden nodes, all interconnection weights, and an output node. This neural network looks like any traditional neural network, except that there are three special input nodes feeding with three different random signals generated uniformly between 0.0 and 1.0, named R1, R2 and R3. All these random numbers are generated internally so that their values are changed stochastically in every execution. While the first two input values are static, at 0.8 and 0.1, when the neural network is executed for 100 times without all extra input nodes, the outcomes are certain (around 0.3), as shown in Figure 3.4 b. When the network is executed for 100 times with only one extra node, the outcomes are 0.3 and 0.4 for 16 and 84 times respectively, as shown in Figure 3.4 c. In Figure 3.4 d and e, the produced outcomes are more uncertain, when the network is executed for 100 times with two and three extra input nodes respectively. With different combinations and connection weights of extra input nodes, the outcomes of the networks can be generated with different shapes of distributions as shown in Figure 3.4 c, d, and e. In summary, the new concept allows outcomes to be uncertain even while input is held constant and internal nodes execute deterministically.

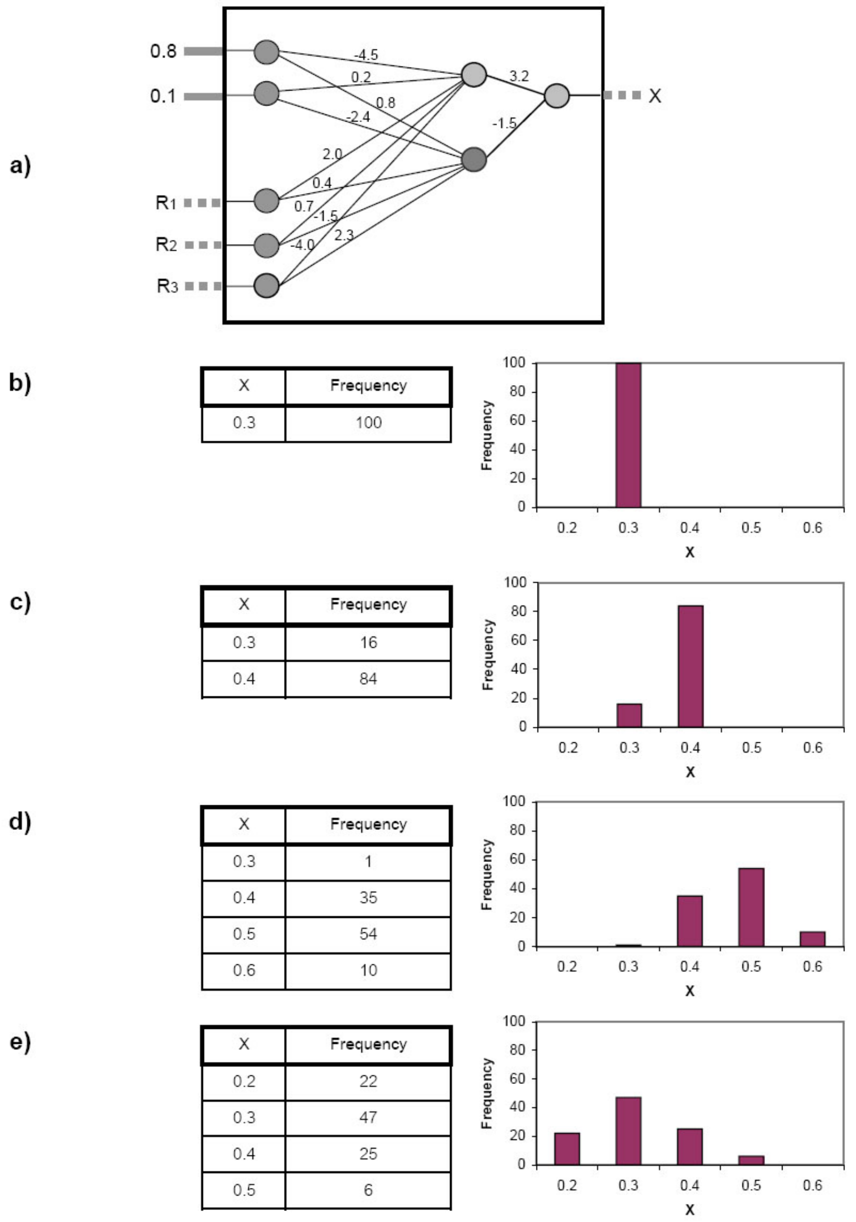
To capture the team performance model, with the non-linearities, output distributions and a very large size, a concise, flexible representation was required. Neural networks were chosen because of the particularly concise way they can capture and reproduce an arbitrarily complex function. However, typical neural networks map precisely from input to output variables and do not allow distributions of possible outputs. To make neural networks appropriate for capturing the team performance model, it was necessary to develop *Stochastic Neural Networks.*

In this section, details of its abstract model and its capabilities are described. At the end, an initial result for attempting to represent team performance models is shown.

3.3.1 Conceptual Idea

Eggenberger et al. [27] introduced the idea of the dynamic rearrangement of biological nervous systems to accommodate learning in non-stationary environments. Their approach allows neural networks an additional mechanism to dynamically change synaptic weight modulations and neuronal states during execution. This capability of changing the modulation types allows the control networks to change their structures when the environment changes.

With inspiration from the dynamic rearrangement [27], a concept of Stochastic Neural Networks (called Dynamic Networks in [84], [85] and [86]) is presented here. This concept allows output nodes to act stochastically even while input is held constant and internal nodes act deterministically. In general, an artificial neural network is an interconnected, layer by



b)

X	Frequency
0.3	100

c)

X	Frequency
0.3	16
0.4	84

d)

X	Frequency
0.3	1
0.4	35
0.5	54
0.6	10

e)

X	Frequency
0.2	22
0.3	47
0.4	25
0.5	6

Figure 3.4: An example of a simple neural network is shown in a, consisting five input nodes, two hidden nodes, all interconnection weights, and an output node. R1, R2 and R3 are extra input signals with different random numbers generated uniformly between 0.0 and 1.0. When the network is executed 100 times without R1, R2, and R3, possible outputs are shown in b. When the network is executed 100 times with one (R1 only), two (R1 and R2), three(R1, R2, and R3) extra input nodes, possible outputs are shown in c, d, and e, respectively.

layer, chain of simple processing nodes, where each node receives a number of inputs and sends an output to other nodes. Each node is deterministic in that its output is based entirely on its input values. To make output nodes act effectively stochastic, an internal stochastic component is included in the network, which is responsible for producing random signals. These internal random signals are simply treated as additional input signals of the artificial neural network, i.e., they are connected to every internal node by adjustable weights. Changing weights result in changing the behaviors of the stochastic networks. Thus, it makes possible to manipulate the neural network's stochastic behavior by adjusting the weights inside the network.

If a target system has a high variation in outputs even for the same input configuration, in non-deterministic cases, the stochastic neural network adapts their weights to match these variances by being stochastic. If the target system is deterministic, which means the outputs are static, the stochastic neural network adapts the weights to ignore stochastic components. Figure 3.5 shows two examples of networks. On the left, a traditional artificial neural network is used to map input-output pairs as shown in the table below it. On the right, a stochastic neural network is used to map all input patterns with all possible output patterns with different probabilities of occurrence. The stochastic neural network captures uncertainly from the additional input nodes fed with internal random signals.

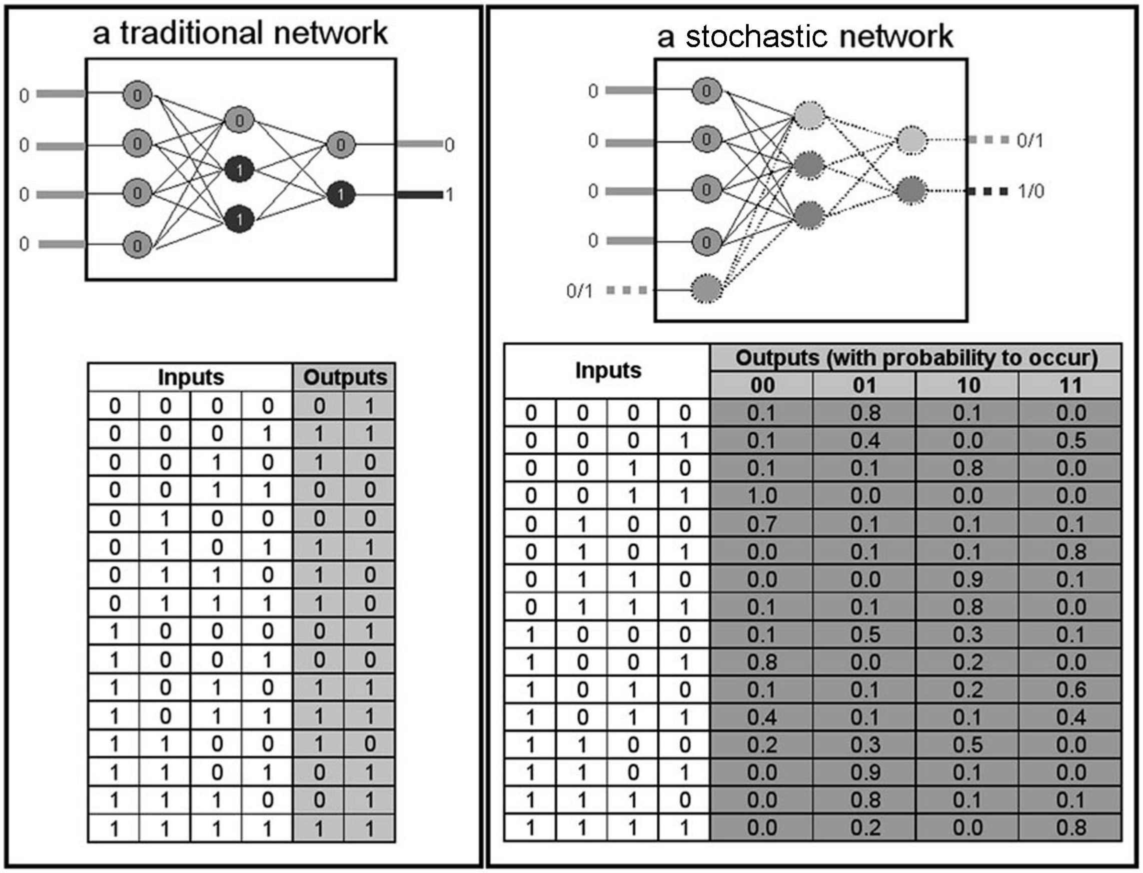
As shown in Figure 3.6, a typical artificial neural network network can be transformed into a stochastic neural network by adding extra input nodes to the input layer and feeding them with internal random signals. These random signals are stochastically changed every time the network is executed. Executing the network for a number of times, a distribution of possible outputs can be generated. The random signals that we normally use are random numbers internally generated from a uniform distribution between 0 and 1. All nodes in the network use sigmoid units. In addition, a three-layer feed-forward network is typically preferred since it is capable of representing any arbitrary function [83].

The concept allows output nodes to act stochastically even while input is held constant and internal nodes act deterministically. In general, an artificial neural network is an interconnected, layer by layer, chain of simple processing nodes, where each node receives a number of inputs and sends an output to other nodes. Each node is deterministic in that its output is based entirely on its input values. The internal random signals are simply treated as additional input signals to the artificial neural network, i.e., they are connected to every internal node by adjustable weights. Changing weights result in changing the distributions of values on the output nodes. Thus, it makes possible to manipulate the neural network's stochastic behavior by adjusting the weights inside the network. If the target system is deterministic, which means the outputs are always the same for the same input, the stochastic neural network adapts the weights to ignore stochastic components.

Effectively, the network maps a given input and range of random values to output values proportional to how often that output occurs in the output distribution. For example, in a simple case, if input 1 corresponded to output 1 20% of the time and to output 2 80% of the time, the neural network would map input 1 and 20% of the possible random values to an output of 1 and map an input of 1 and 80% of the possible random values to an output of 2. Notice that externally, it will not be known which range of random values map to which output value. Arbitrarily complex output distrbutions simply correspond to more complex mappings from input and random ranges to output values.

More than one random input node may be required to capture particularly complex uncertainty distributions, but our results show that approximately three random input nodes will effectively capture even the most complex distributions. However, more random nodes can speed up the learning process, apparently by providing more options for constructing a mapping. In contrast to previous approaches, this technique is straightforward to implement and can be trained simply with pairs of system configuration and system performance instances.

To represent the highly non-linear relationship between environment, configuration, and team performance, stochastic neural networks are used. The network topology consists of 31 nodes in the input layer (one input node representing each varied parameter, see Table 3.2), 5 random signal nodes, 16 nodes in the first hidden layer, 8 nodes in the second hidden layer, and 4 nodes in the output layer.

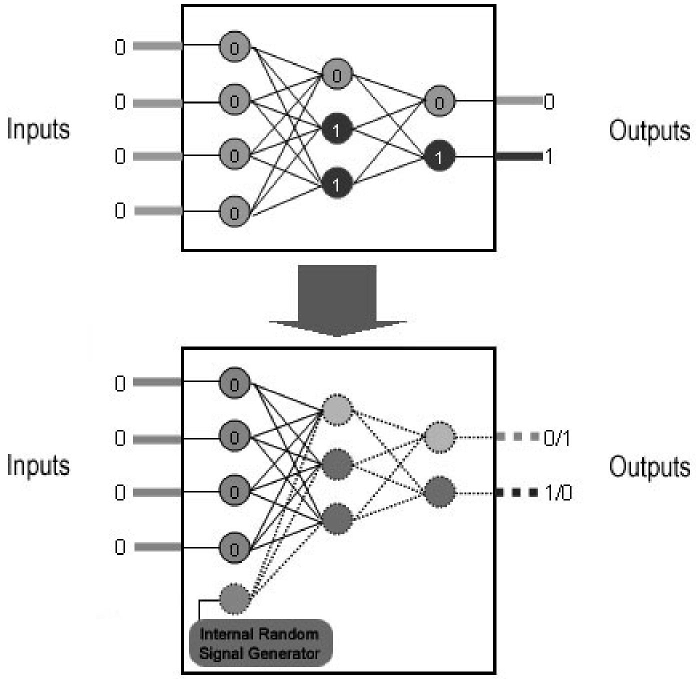


Inputs				Outputs	
0	0	0	0	0	1
0	0	0	1	1	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	0	1	1	1
0	1	1	0	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	0	0
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	1	1

Inputs				Outputs (with probability to occur)			
				00	01	10	11
0	0	0	0	0.1	0.8	0.1	0.0
0	0	0	1	0.1	0.4	0.0	0.5
0	0	1	0	0.1	0.1	0.8	0.0
0	0	1	1	1.0	0.0	0.0	0.0
0	1	0	0	0.7	0.1	0.1	0.1
0	1	0	1	0.0	0.1	0.1	0.8
0	1	1	0	0.0	0.0	0.9	0.1
0	1	1	1	0.1	0.1	0.8	0.0
1	0	0	0	0.1	0.5	0.3	0.1
1	0	0	1	0.8	0.0	0.2	0.0
1	0	1	0	0.1	0.1	0.2	0.6
1	0	1	1	0.4	0.1	0.1	0.4
1	1	0	0	0.2	0.3	0.5	0.0
1	1	0	1	0.0	0.9	0.1	0.0
1	1	1	0	0.0	0.8	0.1	0.1
1	1	1	1	0.0	0.2	0.0	0.8

Figure 3.5: Two examples of artificial neural networks are shown. On the left, a traditional network maps input-output pairs as shown in the table below. On the right, a dynamic network maps all input patterns with all possible output patterns with different probabilities of occurrence.

Inputs				Outputs	
0	0	0	0	0	1



Inputs				Random	Outputs	
0	0	0	0	[0.0, 0.1)	0	0
0	0	0	0	[0.1, 0.2)	1	0
0	0	0	0	[0.2, 0.3)	0	1
0	0	0	0	[0.3, 0.4)	0	1
0	0	0	0	[0.4, 0.5)	0	1
0	0	0	0	[0.5, 0.6)	0	1
0	0	0	0	[0.6, 0.7)	0	1
0	0	0	0	[0.7, 0.8)	0	1
0	0	0	0	[0.8, 0.9)	0	1
0	0	0	0	[0.9, 1.0]	0	1

Figure 3.6: A traditional neural network can be transformed to a stochastic neural network by adding extra input nodes and feeding them with internal random signals to the input layer. These random signals are stochastically changed every time the network is executed. By executing the network a number of times, a distribution of possible outputs can be generated.

3.3.2 What Is New?

To accurately characterize a complex system requires a model of the system as well as a model of the uncertainty impacting the system [3]. That inherent uncertainty leads to uncertainty in the performance of the system. The importance of modeling uncertainty in system performance has been recognized in many fields, such as complex biochemical systems especially in genetic regulatory systems [1, 108, 48], robotic systems performing in real world environment [113], robust control systems [6], and data mining [16]. Notice that for many of these systems, the uncertainty can not be captured as a normal distribution and will often not have been characterized at all. While a range of techniques exist to describe a system and its performance, techniques for capturing the uncertainty in that performance are not as readily available. This work presents a novel approach to capturing both the performance of a system and the uncertainty related to that performance in a concise, easy to use form.

Previous approaches have shown significant potential for representing uncertainty as a form of stochastic neural networks. A special kind of stochastic neural network, called Approximate Identity Neural Networks (AINN) [119], is very powerful and effective in approximate, complex, non-linear systems. However, implementation of an AINN is very complicated and becomes significantly more difficult as the number of input parameters grow. Alternatively, neural networks with *stochastic resonance* were introduced, using a time periodic signal as a stochastic element[75]. Although such stochastic neural networks are trained with an extended back propagation method, their performance is limited to very simple tasks. Recently, stochastic models of neural networks were used to representing complicated gene regulatory networks by using Poisson random signals[116]. However, such stochastic neural networks only capture Poisson and Normal uncertainty distributions. Thus, although promising neural networks have no effective way of handling arbitrary uncertainty distributions.

This work presents a new approach to stochastic neural networks that incorporates three key ideas. The first idea is to expand traditional artificial neural networks by adding extra input nodes to the input layer and feeding them with internal random signals. These random signals are uniformly varied between 0 and 1. The rest of the artificial neural network remains unchanged and deterministic. Effectively, values of the random signal are mapped to possible output values for the system configuration represented by the normal input nodes. More than one random input node may be required to capture particularly complex uncertainty distributions, but our experiments show that approximately three random input nodes will effectively capture even the most complex distributions. In contrast to previous approaches, this technique is straightforward to implement and can be trained simply with pairs of system configuration and system performance instances.

The second key to this technique is to train the stochastic neural networks with a genetic algorithm. An evolutionary process is utilized to shape a population of stochastic neural networks, generation by generation, in accordance with fitness function. Genetic algorithms can overcome a huge number of local minima when relationship between variables is highly non-linear as is the case with many complex systems [38]. In addition, the unit of adaptation is not an individual point, but a population of individual points, which is appropriate for dealing with large and noisy training data. However, significant care is required in the design and use of the fitness function, because it is infeasible to compare an stochastic neural network against every training datum to determine its fitness. Fortunately, not all examples to be seen for a distribution to be accurately captured[119].

The third key to this approach is to allow stochastic neural networks to generate an uncertainty distribution, once it has been trained. The stochastic neural network captures system performance and uncertainty, hence when the system configuration is provided on the input nodes of the stochastic neural network, *one* possible performance case will be present on the output nodes. The output nodes will represent a system performance with probability proportional to the probability of getting the performance in the system being modeled. By executing a stochastic neural network a number of times, a distribution of possible outputs can be generated.

3.3.3 An Initial Result

The objective of the team performance model is to capture the relationship between a vast space of possible system and configuration parameters and the team performance measures.

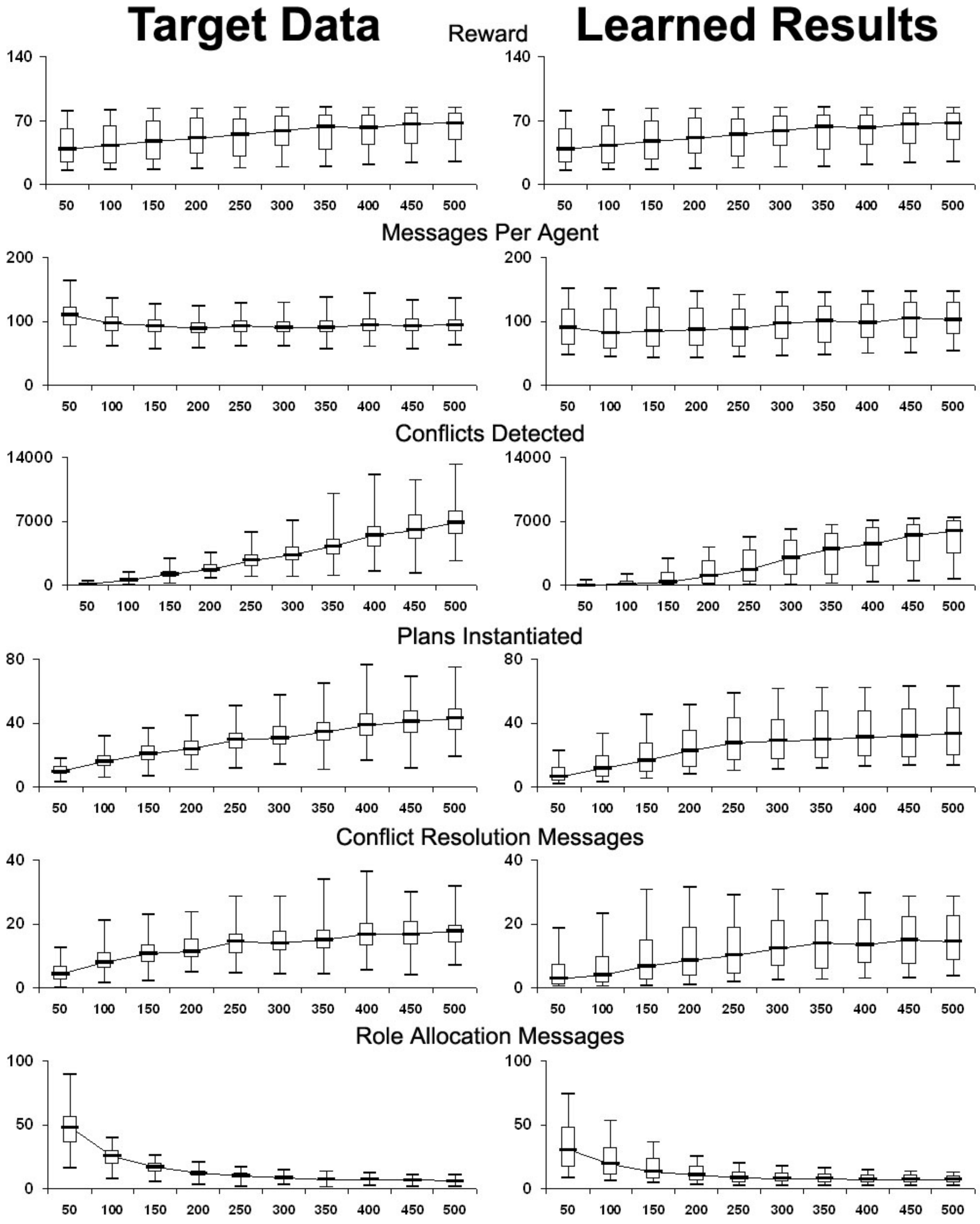


Figure 3.7: Example plots show comparisons between target data and learned results of six output performance measures from a particular setting. The error bars can be viewed as the distributions of output data. The x-axis represents the number of agents.

To obtain high resolution for comparisons, a small reference data set was created, consisting of data obtained from running our previous abstract simulation (called *TeamSim*) by varying only three input parameters leaving other input parameters fixed. The data sets were used to train new team performance models, which in turn generated graphs that were compared with those plotted from the original reference data set.

As shown in Figure 3.7, graphs are plotted between six performance measures and the number of team members. In each row, two graphs are plotted. The graphs on the left are plotted from the reference data set. The graphs on the right are plotted from the team performance model yielded by the reference data set. By using fitness function in Equation 3.9, in which actual and target distributions are compared, the team performance model has good performance in capturing uncertainty of the target system, as shown in Figure 3.7. After running for 1000 generations (about 2 hours), the learned results are approximately the same as the target data, especially the means. There are close correspondence between the team performance models and reference data set in all performance measures and number of agents. However, there still need more time and data for improvement.

Although these comparisons are qualitative, they support our approach by demonstrating that team performance models can be learned for large parameter sets that perform well, even when examined closely over a small range of settings.

3.4 LEARNING PROCESS

Our main objective is to create an input-output model that approximately represents the very complex relationship between coordination algorithm parameters and performance measures for large scale multi-agent teams. Because it is infeasible to collect sufficient data from such a system in real time, an abstract simulation has been created that simulates only the coordination aspects of team behavior. In simulating the behavior of a UAV team, for example, the abstract simulation would still instantiate plans, allocate roles, pass information tokens, etc. but would eliminate the computationally expensive steps of searching the environment to match preconditions or simulating actual execution. Thus, running the abstract

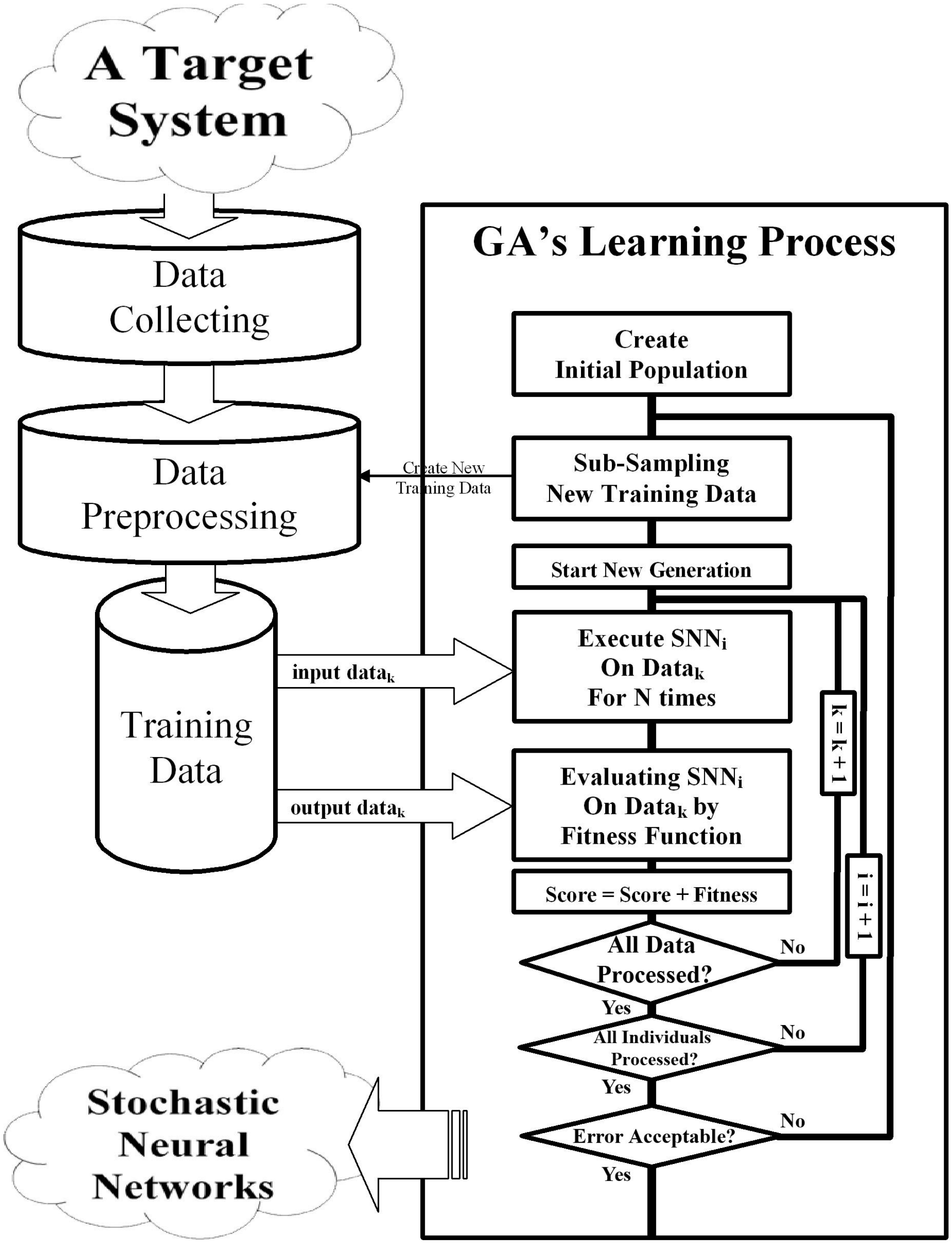


Figure 3.8: Overall learning process diagram

simulation makes it possible to quickly collect data representing target relationships. Even though the target relationships are very huge and very complex, we have found ways to collect sample data that is statistically adequate for use as training data to create a model, we have called *team performance model*. We have, additionally, developed a new neural network architecture, *stochastic neural networks*, that is capable of representing arbitrary complex relationships for this model. Although this team performance model is capable of representing the target relationships, it is not compatible with conventional learning methods such as back propagation, so separate training methods are needed to fit the model. In this section, we describe our training method. The learning process of genetic algorithms is adapted as a learning process to train stochastic neural networks. First, a population of stochastic neural networks is randomly created as the possible candidates for the final model at the end of the learning process. An important and new idea that makes this possible is the new fitness function that utilizes a method of comparing target output distributions and actual output distributions. In order to capture uncertain outcomes, we should not match only one specific output (such as the mean), but rather match a distribution of various possible outputs, so the idea of fitness function for distributions was developed in this work. In addition, a technique, called *the hall of fame approach*, was created and applied to help speed up the process by selecting and retaining good candidates in a *hall of fame* to be used to generate the new generations of possible candidates in the learning process. After the final team performance model has been trained, a graphic user interface, called *team control interface*, is designed to exploit the team performance model for rapid coordination configuration exploration and online reconfiguration.

In this section, details of the modified genetic algorithm are described, especially its adaptation, learning procedure and genetic operations. The ”hall of fame” approach is introduced and its contribution to learning speed-up discussed.

3.4.1 An Adapted Genetic Algorithm

A genetic algorithm[43][37] is a search technique loosely based on the mechanism of natural selection and genetics. Given an environment and a goal formulated as a fitness function, an initial population is generated at random and a set of genetic operators defined. New generations of individuals are generated using three common genetic operators: reproduction, crossover, and mutation. This process repeats until either a sufficiently fit individual is found or time has expired. The solution of the problem is found in the final population.

For this work, each generation of the population contained 2,000 individuals. The chromosomes of each individual defined the weights of the stochastic neural network connection. All weights were randomly generated at the start. After evaluation of the training data set, the 100 best performers were kept and used to produce the 1900 new individuals, via genetic operations. Genetic algorithms were chosen for training because the relationship between input variables was not only non-linear, but also stochastic, which is problematic for back propagation methods due to the large number of local minima.

Genetic algorithms utilize a fitness function which measures, in this case, the accuracy of the stochastic neural network to the target data in order to determine which individuals to propagate to the next generation. Determining fitness fits more technically challenging than for vanilla neural networks because for a stochastic neural network the random internal node causes the output to be different each time the input is presented. To evaluate each individual in every generation, every stochastic neural network is required to execute with the same input values (excluding all stochastic signals) a number of times, so that distributions of actual outputs can be measured. These actual output distributions are compared in the evaluation process against the target output distributions from the training sets via the goodness-of-fit test or Pearson Chi-square (χ^2) test [51] :

$$\chi^2 = \sum_{i=1}^{n} \left(\frac{(f_s - f_s^*)^2}{f_s^*} \right), n = n_a + n_t, \tag{3.6}$$

where n denotes the number of all slots, n_a denotes the number of slots in actual distribution, n_t denotes number of slots in target distribution, f_s denotes observed frequency in a particular slot s, and f_s^* denotes predicted frequency in a particular slot s. The predicted frequency in a particular slot s can be calculated from:

$$f_s^* = \frac{t_o \cdot t_s}{T}, \tag{3.7}$$

where T denotes the total number of observations, t_o denotes the total number of observations in actual or target distributions, and t_s denotes the total number of observations in the same slots combined. From [51], we can get an index of the strength for the relation between these two distributions, by using the formula:

$$V = \sqrt{\frac{\chi^2}{N(k-1)}}, \tag{3.8}$$

where N is the total frequency and $k = 2$ in this case. Then, this index is used to estimate the percentage of error between actual and target distributions. In addition, we need to add an admissible constraint, which is an absolute difference between means of actual and target distributions, to the fitness function for directing the learning process to convert. Thus, the fitness function of individual i in population is:

$$Fitness_i = \frac{\sum\limits_{d \in D} \sum\limits_{p \in P} \left(\chi^2_{d,p} + |\mu_a^{d,p} - \mu_t^{d,p}| \right)}{|D| \cdot |P|}, \tag{3.9}$$

where D is the set of training data ($d \in D$), P is the set of output nodes ($p \in P$), $\chi^2_{d,p}$ is the χ^2 of the p^{th} output of the data entry d, $\mu_a^{d,p}$ is a mean of actual distribution of the p^{th} output of the data entry d, $\mu_t^{d,p}$ is a mean of target distribution of the p^{th} output of the data entry d, $|D|$ is the size of training data and $|P|$ is the number of output nodes.

To compute fitness value for a stochastic neural network$_i$ in a particular generation, the process is as follows:

```
1) for each $data_k$ in training data set,
   a. execute a stochastic neural network$_i$ with the same input values from $data_k$ $t_a$ times
   b. set $Score = 0$
   c. for each $output_j$:
      i. compute distributions of actual $output_j$ and its actual mean ($\mu_a^{k,j}$)
      ii. use target $output_j$ distribution from $data_k$ to compute the target mean ($\mu_t^{k,j}$)
      iii. compute predicted frequency $f_s^*$ in every slot $s$ (Equation 3.7)
      iv. compute $\chi^2_{k,j}$ (Equation 3.6)
      v. compute $Score = Score + \chi^2_{k,j} + |\mu_a^{k,j} - \mu_t^{k,j}|$
```

d. $Score = Score/|P|$

2) $Fitness_i = Score/|D|$

Then, $Fitness_i$ is assigned to the i^{th} stochastic neural network in the current population. These values are used in a part of ranking process for generating the new population of stochastic neural networks in the next generation.

3.4.2 Learning Procedure

The outline of the learning procedure is shown in Figure 3.8. The details are described as follows.

```
Create an Initial Population
generation = 0
do
    Sub-sampling New Training Data
    Start New Generation
    Execute All Individual Stochastic Neural Networks
    Evaluating All Individual Stochastic Neural Networks
    Compute Ranking
    generation = generation + 1
until Terminate The Process
return
```

Create an Initial Population : An initial population of stochastic neural networks is created. If it is at the beginning of the learning procedure without previous saved stochastic neural networks, all individual stochastic neural networks are generated from totally random numbers. Otherwise, the previous saved stochastic neural network files are loaded. Usually, the best 50 stochastic neural networks are stored in files periodically during the process.

That means if the process was terminated and started again, the first 50 stochastic neural networks are loaded from saved files and the rest of stochastic neural networks are randomly created.

Sub-sampling New Training Data : Since there is a potentially huge volume of collected data, it is infeasible to allow stochastic neural networks to train with all the data, so instead a smaller set is employed for training stochastic neural networks in each generation. In every new generation, a small set of new training data is randomly picked from the total collected data. This ensures the stochastic neural networks are tested in an unpredictable way, which improves robustness in dealing with uncertainty. Our criteria for selecting how many data sets are needed is dependent on execution time of the learning process in a generation, since diversity of stochastic neural networks generated between generations is the most important factor for the learning process. These training data consists of two data, input parameter data and their attached output performance measure data. There is an option that the same training data are used repeatedly for a specific number of generations before a new training data are selected again.

Start New Generation : The new generation of stochastic neural networks is generated based on the performance of all stochastic neural networks in the previous generation. First, the 5 percent top performance stochastic neural networks are reproduced to be in the new generation. Then, the rest of the population is created by crossover operations. In addition, mutation operations are applied occasionally with probability 0.001.

Execute All Individual Stochastic Neural Networks : By using input parameter data from each training datum, each individual stochastic neural network is executed for a prespecified number of times, ~ 30. As results, output distributions are computed. In addition, the means of output distributions are also determined.

Evaluating All Individual Stochastic Neural Networks : Each stochastic neural network is evaluated with the fitness function in (Equation 3.6). Basically, the fitness function is computed from actual output distributions, target output distributions, means of actual output distributions, and means of target output distributions.

Compute Ranking : After fitness values of all individual stochastic neural networks are computed, these values are converted to each individual performance score. Then, all stochastic neural networks are ranked by their performance scores. This ranking information is mainly used for generating new generation.

Terminate The Process : The learning procedure is terminated when a low percent for error of the best individual stochastic neural network is reached. In other cases, the process is terminated if maximum generation is reached or by user.

3.4.3 Genetic Operations

As the common Genetic Algorithms' process [43][37], there are three genetic operations: reproduction, crossover, and mutation. The reproduction is a simple operation for duplicating an individual in the previous population into a new population of the next generation. The crossover is an operation of recombining randomly chosen parts of two of the previous individuals. The mutation is an infrequent operation that randomly changes a part or some parts of an individual. In this work, even though an individual presents a network structure of neural nodes and values of weights that link neural nodes together, the crossover and mutation are designed to operate only on the values of weights, since the structure of the network is unchangeable.

There are two types of crossover operations. The first crossover randomly chooses a link between any two nodes in the network and then uses this link position as a point to swap two neural networks, referred as one point crossover, as shown in Figure 3.9(a). The second crossover, referred as multi-point crossover [112], as shown in Figure 3.9(b), operates on three cutting points, in which each point is in three different weight layers. The swaps are also done separately in all three layers.

The mutation is an operation with a low probability to happen with any individual. When an individual is chosen to mutate, the mutation scans the neural network inside and randomly selects some weights to change their values. In this work, there are three ways to change the values of weights. The first way, the new value is replaced with a new random value between -0.5 and 05. The second way, the new value is an addition between the old value and a random value between -0.5 and 0.5. The third way, the new value is an addition between the old value and a Gaussian noise value. These three ways of value changing are

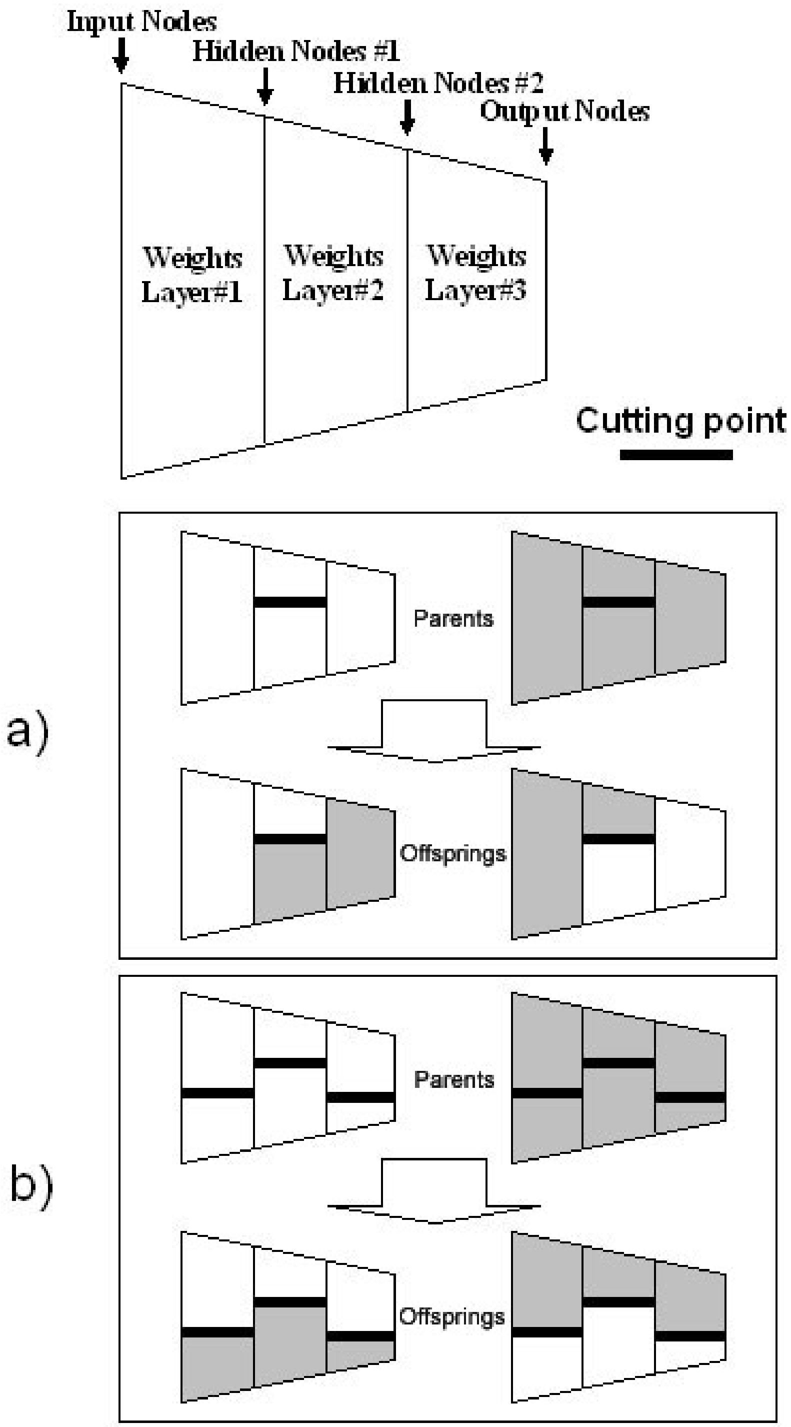


Figure 3.9: Abstract model of a neural network is shown (above) with three weight areas and four columns of nodes. Abstract examples of two types of crossover operation are shown. In (a), the crossover operates with a cut. In (b), the crossover operates with three cuts in three different weight layers.

randomly chosen to apply with an individual when the mutation occurs.

3.4.4 A "Hall of Fame" Approach

As Genetic Algorithms method is used to train the team performance model, a problem arises when dealing with complex and uncertain training data. Since each generation different training data are used for guiding discovery promising models, any good models in a particular generation might not be good enough to be kept in a next few generations but might be useful in the future. It is very importance to find a way to keep these candidate models for future benefit. In [127], a special population of local good individuals is employed to keep only better performance individuals in one place. These good individuals are also utilized in the process of creating new generation as well.

A hall of fame approach is an additional procedure designed to keep the promising models for future improvement during a learning process. The basic idea is that the best performance model in every generation is considered to be kept. However, the best one in every generation can not be reserved, the hall of fame should contain with a small number of good ones not another way round. A possible method is simply using probability approach. For example, the best model out of fifty generations is randomly selected to be in the hall of fame. Using some additional criterions for selection consideration is possible; however they may contain biases that guide into wrong learning directions.

In addition, being in the hall of fame, all kept models are considered to be out of action but useful. Therefore, these reserved individuals are not directly used as any current individuals in any generation during learning process. The only way to be able to use these individuals is by selecting them to be available candidates for possible crossover operations during the process of creating new generation. In each time, some reserved ones from the hall of fame are randomly picked and applied with the mutation operations, and then they are listed as accessible candidates for breeding.

The hall of fame approach provides many additional features for the learning process, as follows.

Preserving Good Individuals : This feature is a basic feature of the approach. Randomly keeping the best performance in each generation preserves good models that can be used in the future.

Keeping Progress Tracking : The progression of the learning process is monitored by evaluating reserved individuals in the hall of fame. Using information embedded with the kept models, it is possible to track the learning performance. Moreover, if there are increasing numbers of members in the hall of fame, the members that have been outperformed by recent newcomers can be removed from the list.

Running with Different Settings : With many options for learning parameter settings, the hall of fame approach allows the learning process to run with different settings at the same time. By keeping the best performances of all settings together, every running can take advantages from other settings' runs. If a particular run has achieved very good results toward solving the problem, other operations will get benefits and learn from those results as well.

Interrupt-able Learning : By keeping the up-to-date good performances, the learning process can be stopped and started again at any time. When the learning process starts the members in the hall of fame provide a head start for the process to pick up from the point where it stopped.

Speeding Up Learning Process : As a result of all above features, the hall of fame approach is intended to speed up the learning process. Keeping good performances saves time for learning those ones again. Running different approaches in the learning process makes possible of parallel searching for possible solutions, which they can be executed on different computer machines or different processes.

3.5 TEAM CONTROL INTERFACE

The stochastic neural network does not solve the reconfiguration problem on its own, it simply forms the core of a tool which the user can use to reconfigure the team. The user specifies the environment and preferences over output parameters and gets options for possible configurations. The user can then request that the tool sends messages to each member of the team to arrange the reconfiguration. The exact mechanism for doing this is dependent on the specific coordination algorithms.

The team control interface is designed to be used with the team performance model. The team control interface is shown in Figure 3.10. The top part of the interface is used for displaying messages. On the right of the top part is a choice of backward search speeds. The bottom part is a place for command buttons, e.g. load model button, load configuration button, backward execute button, exit button, etc. Between the top and bottom parts, there are three display areas. The left side is a list of all input parameters with their current values. Each input parameter uses a check box to indicate whether it is a system (shaded background) or configurable (white background) parameter. The parameters listed only display their current values. To change these values, arrow buttons (> or <) next to the values are used to select which parameters are displayed in the middle of the screen where user can change their values. However, only configurable parameters are able to be changed, therefore if the system parameters need to be changed, they have to switch to be configurable parameters first. Then, later after their values are changed, they have to switch back to the system parameters.

The middle panel gives the user the ability to explore various configurations. They simply select values on the sliders and the expected performance of that configuration is shown on the right hand side. This usage of the model, referred to as “forward mode” because the underlying stochastic neural network is used in a normal feed forward manner, allows an expert user to explore various options very quickly without using either a real team or even the abstract simulator.

Four output performance measures are shown on the right side. These outputs are displayed in two formats: data distributions and box plots. The data distributions are histograms of output values when the current team performance model is executed for one hundred times. The box plots indicate output minimum values, lower quartile, median, upper quartile, and maximum values. These values show statistical distributions of the outputs generated from the current model. There are two check boxes for each performance measure. The left check box is used to switch between displaying a result distribution with all available slots or with only non zero slots. The right check box is used to specify backward search constraints identifying whether which performance measures are chosen to be used as constraints for backward searching. When this check box is checked, pull down menus are also used to specify the constraints whether to increase or to decrease those selected performance measures. When a model is loaded, current input parameters are shown on the left side and output results are instantly presented on the right side. If any parameter in the middle is changed the updated results are displayed instantaneously.

3.5.1 Features

The team control interface has many features that allow users to utilize team performance models toward better understanding and controlling the coordination configuration of large multi-agent teams. All features are described as follows.

Loaded Model : A team performance model can be loaded only one model at a time to the team control interface. All models generated from the learning process are eligible for use with the interface. If there is no model presented, the team control interface will not operate any features.

Load and Save Configuration : A preset configuration can be loaded into the team control interface from a configuration file. The configuration file contains not only all values of every parameters but also information about which parameters are system or configurable parameters. When a user is exploring the team control interface, the current working configuration can be saved into a file and used again later.

In addition, the team control interface can be used in two modes: *offline* and *online*. Offline mode is designed for interacting with the team performance model to help users develop a better understanding of the relationship between coordination configuration parameters and team performance. The offline mode has two features, which are described as follows.

Input-to-Output Feature : Using the team performance model in this forward mode, users can experiment with changing input parameters and observing how output parameters change instantaneously. In this mode, only configurable parameters, which are selected for displaying in the middle of the screen, are allowed to change values. If system parameters have to to be changed, they have to switch to be configurable parameters first. Users can type in new values, use sliders, or use pull down menus for changing parameter's values, see Figure 3.10.

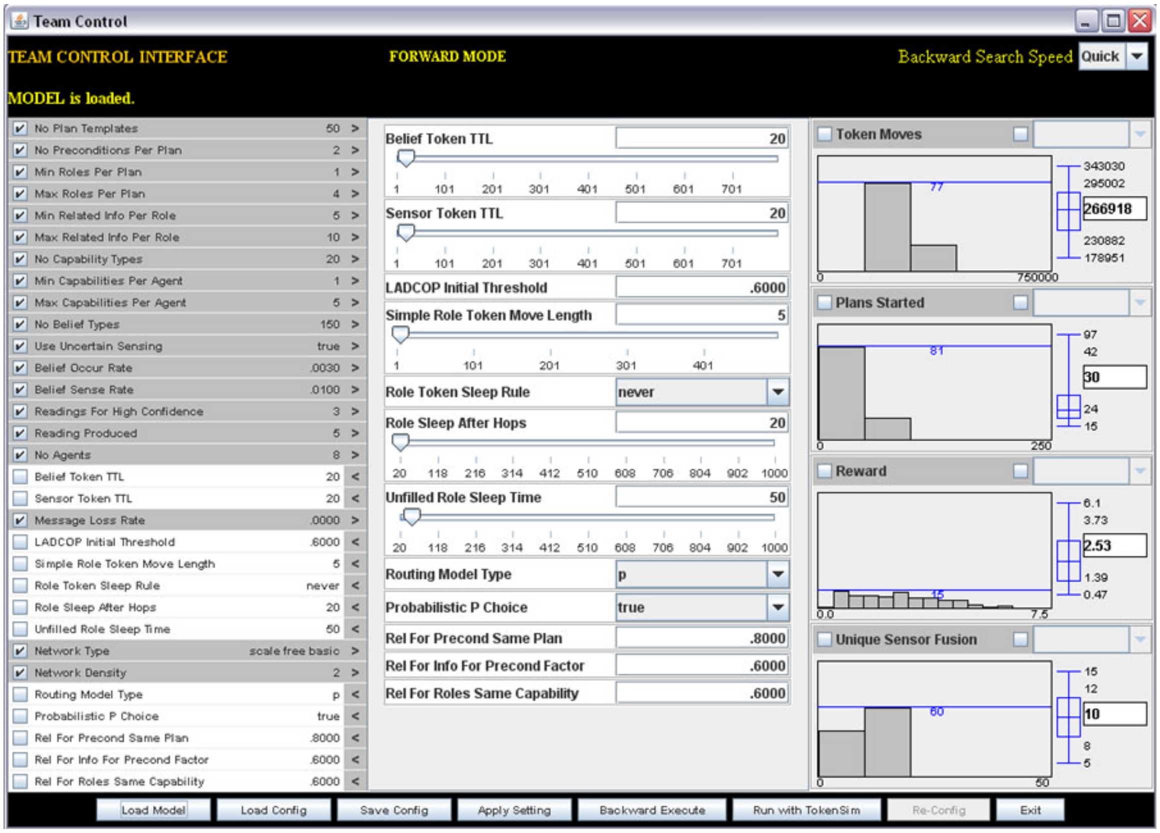
Output-to-Input Feature : Using the team performance model in backward mode, the team control interface allows the user to choose constraints on output parameters (either increase or decrease) and receive a list of configurations that meets specific performance constraints, as shown in Figure 3.11.

Backward Execution : When the "backward execution" button is clicked, the team control interface performs a search over the changeable configuration parameters using the team performance model to find a configuration that yields the required performance trade-offs. The search space covers all changeable configuration parameters and the search constraints are all output performance parameters. Guided by a sensitivity analysis technique [67] on the team performance model, the searching process performs very quickly. The user can choose three different speeds of backward searching, as shown in Figure 3.12. This option indicates how many times the backward search is used for finding a list of good matches. The search result window, in Figure 3.11, contains a list of ten best matches with different configurations in which users can examine which one they would like to use.

Apply Setting : A configuration from the list can be selected and applied to the interface by using an "apply setting" button. Then, the current configuration is replaced with the selected configuration and the outputs are updated with the new setting, as shown in Figure 3.13.

Run with TokenSim : In online mode, the team control interface is connected with TokenSim. The interface allows the user to display team performances and to change configuration during execution. When running in this mode, an additional window is used to display the online output performance parameters in graphical plots, as shown in Figure 3.14. In each plot, an anticipated box plot is presented showing the distribution range of values predicted by current team performance model.

Re-Configuration : During execution in online mode, the user is allowed to use offline interface features, such as output-to-input, to aid in selecting new configuration parameters. By using the "re-config" button, new configurable parameters are sent to replace the old values in TokenSim, as shown in Figure 3.15.



Team Control

TEAM CONTROL INTERFACE

FORWARD MODE

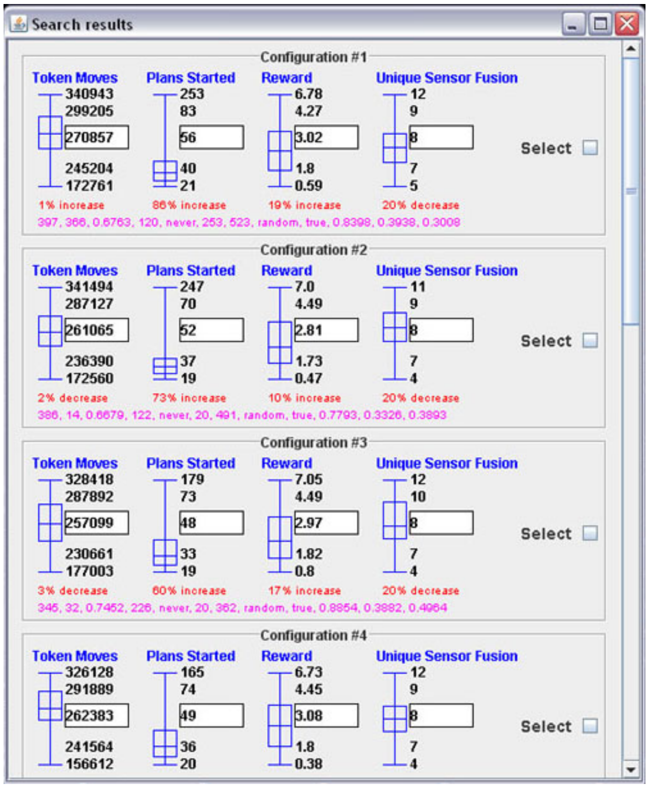
Backward Search Speed: Quick

MODEL is loaded.

☑	No Plan Templates	50	>
☑	No Preconditions Per Plan	2	>
☑	Min Roles Per Plan	1	>
☑	Max Roles Per Plan	4	>
☑	Min Related Info Per Role	5	>
☑	Max Related Info Per Role	10	>
☑	No Capability Types	20	>
☑	Min Capabilities Per Agent	1	>
☑	Max Capabilities Per Agent	5	>
☑	No Belief Types	150	>
☑	Use Uncertain Sensing	true	>
☑	Belief Occur Rate	.0030	>
☑	Belief Sense Rate	.0100	>
☑	Readings For High Confidence	3	>
☑	Reading Produced	5	>
☑	No Agents	8	>
□	Belief Token TTL	20	<
□	Sensor Token TTL	20	<
☑	Message Loss Rate	.0000	>
□	LADCOP Initial Threshold	.6000	<
□	Simple Role Token Move Length	5	<
□	Role Token Sleep Rule	never	<
□	Role Sleep After Hops	20	<
□	Unfilled Role Sleep Time	50	<
☑	Network Type	scale free basic	>
☑	Network Density	2	>
□	Routing Model Type	p	<
□	Probabilistic P Choice	true	<
□	Rel For Precond Same Plan	.8000	<
□	Rel For Info For Precond Factor	.6000	<
□	Rel For Roles Same Capability	.6000	<

Belief Token TTL	20
Sensor Token TTL	20
LADCOP Initial Threshold	.6000
Simple Role Token Move Length	5
Role Token Sleep Rule	never
Role Sleep After Hops	20
Unfilled Role Sleep Time	50
Routing Model Type	p
Probabilistic P Choice	true
Rel For Precond Same Plan	.8000
Rel For Info For Precond Factor	.6000
Rel For Roles Same Capability	.6000

Figure 3.10: Team control interface displays all input parameters on the left side, performance measures on the right side, selected editable parameters in the middle, and command buttons in the bottom.



```
Search results

Configuration #1
Token Moves   Plans Started   Reward   Unique Sensor Fusion
340943        253             6.78     12
299205        83              4.27     9
270857        56              3.02     8                       Select □
245204        40              1.8      7
172761        21              0.59     5
1% increase   86% increase    19% increase   20% decrease
397, 366, 0.6763, 120, never, 253, 523, random, true, 0.8398, 0.3938, 0.3008

Configuration #2
Token Moves   Plans Started   Reward   Unique Sensor Fusion
341494        247             7.0      11
287127        70              4.49     9
261065        52              2.81     8                       Select □
236390        37              1.73     7
172560        19              0.47     4
2% decrease   73% increase    10% increase   20% decrease
386, 14, 0.6679, 122, never, 20, 491, random, true, 0.7793, 0.3326, 0.3893

Configuration #3
Token Moves   Plans Started   Reward   Unique Sensor Fusion
328418        179             7.05     12
287892        73              4.49     10
257099        48              2.97     8                       Select □
230661        33              1.82     7
177003        19              0.8      4
3% decrease   60% increase    17% increase   20% decrease
345, 32, 0.7452, 226, never, 20, 362, random, true, 0.8854, 0.3882, 0.4964

Configuration #4
Token Moves   Plans Started   Reward   Unique Sensor Fusion
326128        165             6.73     12
291889        74              4.45     9
262383        49              3.08     8                       Select □
241564        36              1.8      7
156612        20              0.38     4
```

Figure 3.11: A window of search results is shown a list of possible configurations along with their possible outcomes. Users might select one of these configurations to apply into the interface.

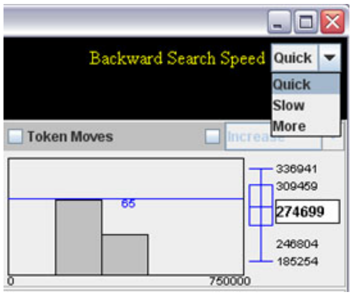
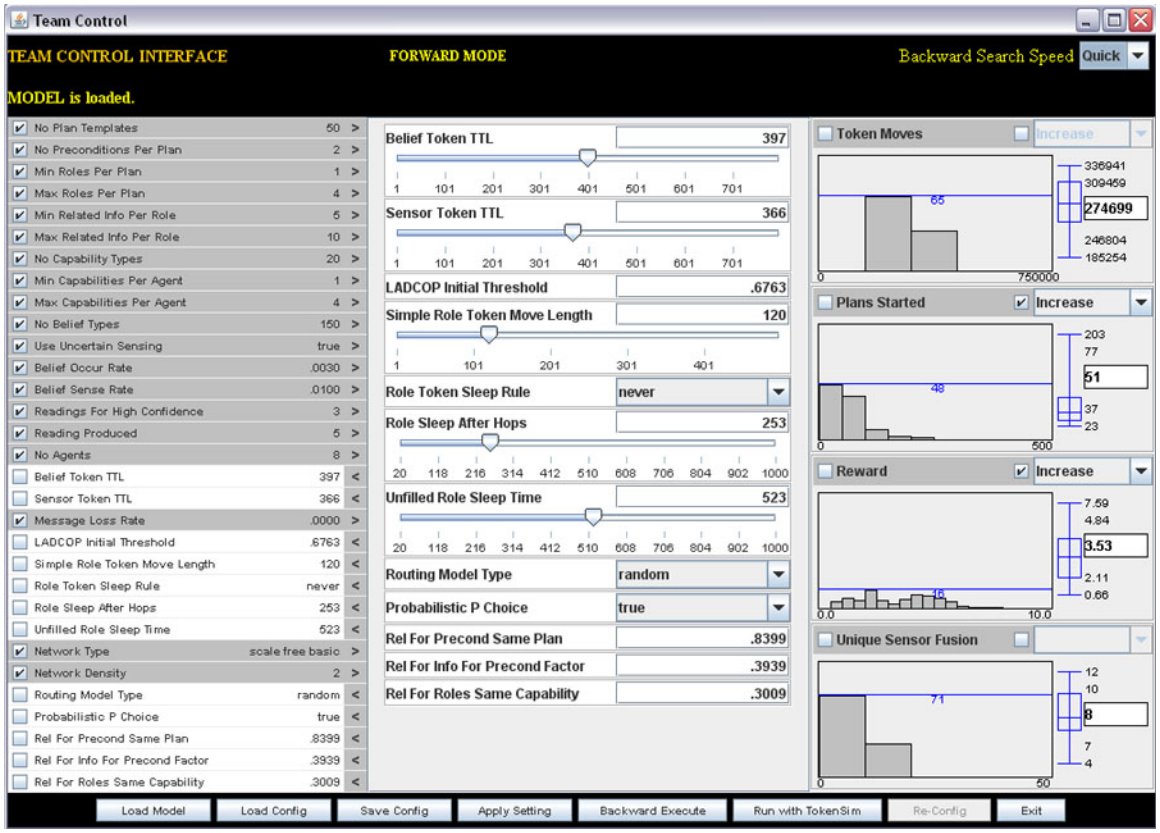


Figure 3.12: A pull down menu has three choices of backward search speeds.



Team Control

TEAM CONTROL INTERFACE

FORWARD MODE

Backward Search Speed Quick

MODEL is loaded.

☑	No Plan Templates	50	>
☑	No Preconditions Per Plan	2	>
☑	Min Roles Per Plan	1	>
☑	Max Roles Per Plan	4	>
☑	Min Related Info Per Role	5	>
☑	Max Related Info Per Role	10	>
☑	No Capability Types	20	>
☑	Min Capabilities Per Agent	1	>
☑	Max Capabilities Per Agent	4	>
☑	No Belief Types	150	>
☑	Use Uncertain Sensing	true	>
☑	Belief Occur Rate	.0030	>
☑	Belief Sense Rate	.0100	>
☑	Readings For High Confidence	3	>
☑	Reading Produced	5	>
☑	No Agents	8	>
□	Belief Token TTL	397	<
□	Sensor Token TTL	366	<
☑	Message Loss Rate	.0000	>
□	LADCOP Initial Threshold	.6763	<
□	Simple Role Token Move Length	120	<
□	Role Token Sleep Rule	never	<
□	Role Sleep After Hops	253	<
□	Unfilled Role Sleep Time	523	<
☑	Network Type	scale free basic	>
☑	Network Density	2	>
□	Routing Model Type	random	<
□	Probabilistic P Choice	true	<
□	Rel For Precond Same Plan	.8399	<
□	Rel For Info For Precond Factor	.3939	<
□	Rel For Roles Same Capability	.3009	<

Belief Token TTL 397

Sensor Token TTL 366

LADCOP Initial Threshold .6763

Simple Role Token Move Length 120

Role Token Sleep Rule never

Role Sleep After Hops 253

Unfilled Role Sleep Time 523

Routing Model Type random

Probabilistic P Choice true

Rel For Precond Same Plan .8399

Rel For Info For Precond Factor .3939

Rel For Roles Same Capability .3009

□ Token Moves □ Increase

□ Plans Started ☑ Increase

□ Reward ☑ Increase

□ Unique Sensor Fusion □

Figure 3.13: A new configuration is updated from the results of backward search.

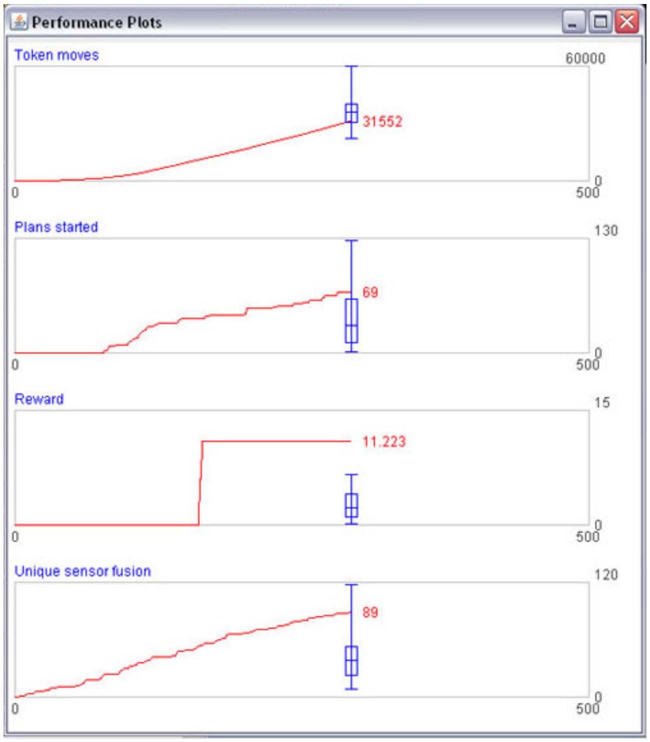


Figure 3.14: A plotting window displays four output values when it is running with TokenSim.

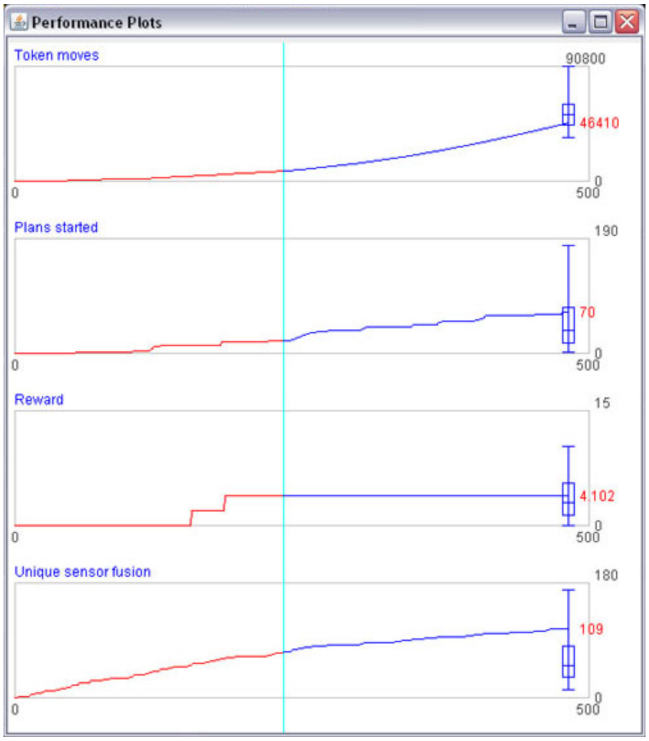


Figure 3.15: When the re-configuration command is issued, the plotting display changes to a new color and draws a line indicating a point where a new configuration has been executed.

3.5.2 Backward Search Procedure

In this work, the backward search utilizes a Genetic Algorithms search technique. The procedure is described as follows.

```
Create an Initial Population
generation = 0
do
    Evaluate All Individual Configurations with Current Team Performance Model
    Generate New Population
    generation = generation + 1
repeat Search for a Specific Number of Generations
return List the Top Ten
```

Create an Initial Population : Initial population of possible distinctive configurations is created partly guided by a sensibility analysis technique [67] on the team performance model.

Evaluate All Individual Configurations with Current Team Performance Model : The current team performance model is applied with each individual configuration to generate distributions of performance measures. According to output constrains specified by the user, the fitness function is calculated. Thus, all individual are ranked in proportion to their fitness values.

Generate New Population : A new population is created by randomly perturbing[18] the top ten best scores. In addition, the newly generated configurations have to be distinctive from other members in the population.

Search for a Specific Number of Generations : The search proceeds for a specific number of generations. The user chooses the speed of backward search from the interface. Each speed has a different number of generations to repeat. In this work, the quick, slow, and more options are assigned with 100, 500, and 1000 generations respectively.

List the Top Ten : When the search is finished, the final top ten distinctive configurations are revealed as the results for selection.

4.0 EXPERIMENTS AND RESULTS

This section, experiments were conducted to verify the proposed approach. First, a preliminary experiment was carried out for demonstrating the capability of stochastic neural networks to represent the model beta distribution, indicating the potential to represent uncertainty. Then, a main experiment was performed for verifying the approach by modeling team performance models, in which they represent the relationships between configuration parameters of coordination algorithms and performance measures for large multi-agent teams. Details of experiment settings and procedures are described. Finally, results are presented with discussions.

4.1 MODELING A BETA DISTRIBUTION

A beta distribution is a versatile distribution that has been commonly used for modeling data with uncertainty in many applications [23, 73]. The probability density function of the beta distribution can be calculated with the following equation [73] :

$$f(x;\alpha,\beta) = \frac{1}{Beta(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \tag{4.1}$$

$$0 < x < 1,$$

where $\alpha > 0$, $\beta > 0$ and $Beta(\alpha,\beta)$ denote the beta function defined by

$$Beta(\alpha,\beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt.$$

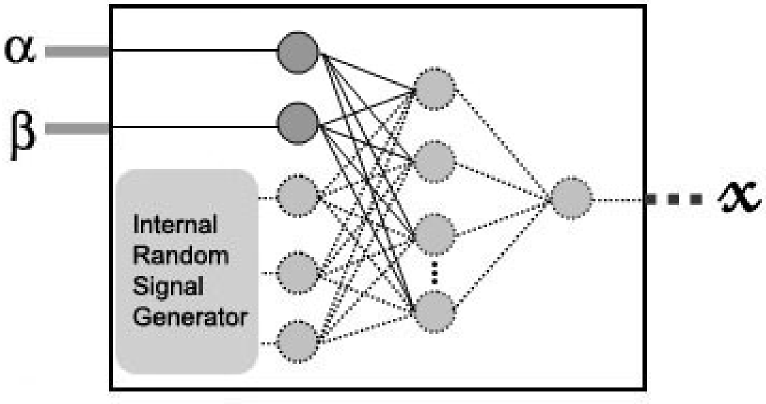


Figure 4.1: The network architecture for the first experiment consists of three layers. The input layer has α, β and internal random signals. The output layer has only a variable x. The hidden layer usually has two layers of nodes.

The key feature of a beta distribution is that different density function shapes can be obtained by changing α and β. For instance, uniform distribution can be obtained when $\alpha = 1$ and $\beta = 1$ and when $\alpha = 1/2$ and $\beta = 1/2$, a U-shape distribution called arc-sine distribution is produced [23]. When $\alpha = \beta > 1$, the distribution generates a symmetric normal distribution. When $\alpha \neq \beta$, $\alpha > 1$ and $\beta > 1$, an asymmetric distribution is generated.

4.1.1 Experiment Setup

To model the beta distribution, multilayer feed-forward neural networks were used as shown in Figure 4.1. The network topology consists of five nodes in the input layer (two input nodes representing α and β parameters, and three stochastic signal nodes), sixteen nodes in the first hidden layer, eight nodes in the second hidden layer, and one node in the output layer. The ranges of α and β are limited to $0 < \alpha \leq 10$ and $0 < \beta \leq 10$. All stochastic signals are uniformly distributed between 0.0 and 1.0. All nodes in the networks are sigmoid units.

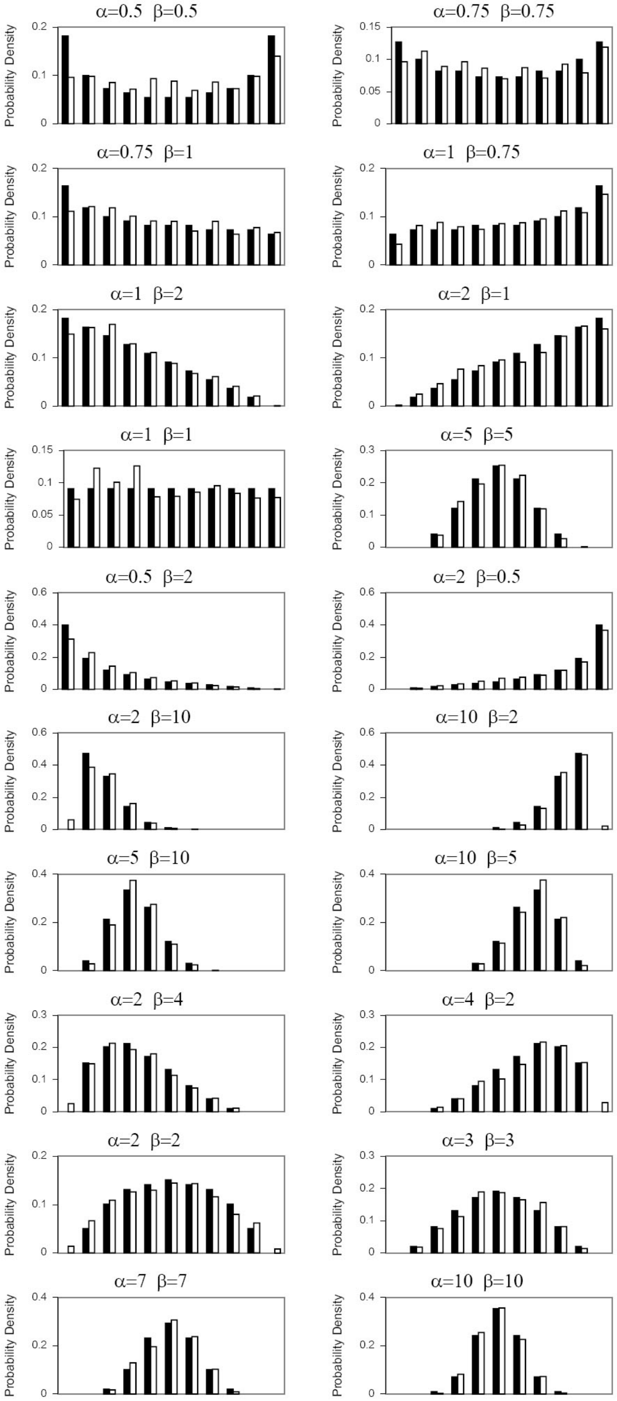


Figure 4.2: A set of 20 plots with different values of α and β was produced by one of the best generated stochastic neural networks comparing with target distributions. The target distributions are shown in solid black bars and actual distributions are shown in white bars. There is an average of less than 10 % error.

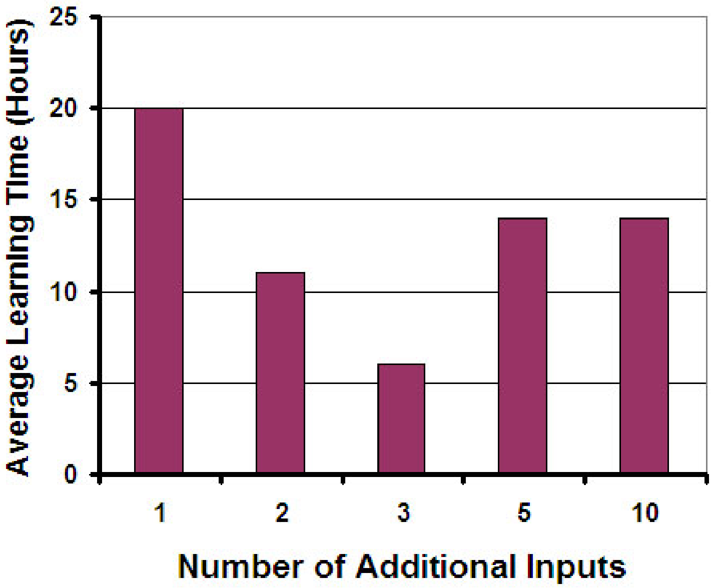


Figure 4.3: Average learning times (to reach 25 percent of distribution error) of stochastic neural networks with different numbers of random inputs.

To create training sets, data were collected by using Equation 4.1. The data collection process is described as follow.

1) First, to get a target distribution of x, we divided variable x into 11 slots, where $x = 0.0, x = 0.1, x = 0.2, \cdots, x = 1.0$.

2) Values of α and β were randomly generated.

3) Using Equation 4.1, the values of α and β were used with values of x in all 11 slots to get its target distribution.

4) If not done, go to step 2.

In this experiment, the training set consisted of 20 data records, which were randomly generated. The training set was changed to a new set in every generation. The learning process terminated when the best individual achieved less than 10 % error.

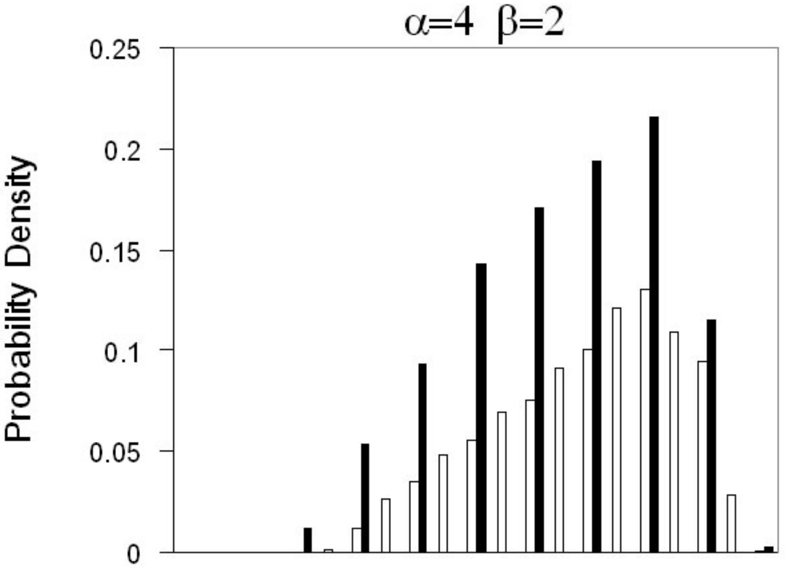


Figure 4.4: After trained using data with 11 possible output slots, stochastic neural network can be expanded from 11 slots (black bars) to 21 slots (white bars) by modifying process of output discretization.

4.1.2 Results and Discussions

After running about 2000 generations (approximately 20 hours, using PC-Intel Celeron 1.4GHz, 480MB), the results are shown in Figure 4.2. The target distributions are shown in solid black bars and actual distributions are shown in white bars. These plots are generated from one stochastic neural network by changing values of α and β. Each plot is a result of executing the network 110 times and counting frequencies of 11 slots of the output node x. It is clear stochastic neural network can handle symmetric normal distributions and asymmetric distributions well (when $\alpha > 1$ and $\beta > 1$). Uniform distributions (when $\alpha = \beta = 1$) and U-shape distributions are more difficult. It was found that the learning process adapted quickly to match the symmetric normal distributions and asymmetric distributions, but took much more time to adapt to the more difficult ones.

A second experiment was run to determine the optimal number of additional random inputs that were needed to solve the problem. We found that without any additional random input, the stochastic neural networks learned only to represent the means of the target distributions. Having only one additional random input present the stochastic neural network took a very long time to reach the error threshold whereas having two or five present the threshold could be met in about 2000 generations (approximately 20 hours) on average. Figure 4.3 compares the time taken in hours to reach a 25 percent of distribution error threshold and shows the optimal time was achieved with three additional random inputs. Having too few inputs makes it difficult to convert into complex input-output relationships while having too many means more time is required to adjust more weights. This number must be determined on a per problem basis. As shown in Figure 3.4, it suggests that more numbers of random input nodes might effectively lead to a Gaussian input being fed to the internal nodes, so that the more the output signal is approximating a Gaussian, the more multiple random input nodes will be useful and the more is it uniform or something else, the more a single random input will be appropriate, perhaps even with non-uniform distribution. This number of extra random inputs must be determined on a per problem basis.

An interesting feature of stochastic neural networks is that the networks can modify the density of possible outputs and still get the same distributions. As shown in Figure 4.4, a trained stochastic neural network was modified its output discretization by doubling its possible output ranges and still get the same shape of distribution. The size of distribution is smaller because there are more possible outcomes.

4.2 MODELING TEAM PERFORMANCE MODELS

The stochastic neural network approach was applied to create a *team performance model* that captures the relationships between the environment, team configuration parameters, and performance measures in a way that allows rapid exploration by users. Due to the non-determinism of environments and coordination algorithms, these relationships are highly non-linear and stochastic. To create a team performance model, an abstract simulation of the coordination algorithms with a highly configurable environment is developed first. Because the simulation can be run at high speeds, it can be used to create large amounts of data on the relationships between parameters. That data can then be used as training data.

Then, the team performance model is used to search for the best configuration of the team that meets specific performance constraints. Using the team performance model in "reverse" allows users to specify performance tradeoffs and rapidly receive a configuration that best meets those constraints. Since not all parameters are configurable, a search over the changeable configuration parameters is performed to discover a configuration that best meets the required performance tradeoffs.

In this section evidences are presented that the approach achieves the primary goal which is to capture the effects of configuration parameters on performance measures in the team performance model.

4.2.1 Training Data

After almost 2 months of data collection with more than 10 running computers, training data have been collected, containing 20767 configurations. While attempting to represent a very large space of possible configurations, these data were sampled randomly and designed to use all possible ranges of parameters equally. The sampling distributions of input parameters are shown in Figure 4.5 and 4.6.

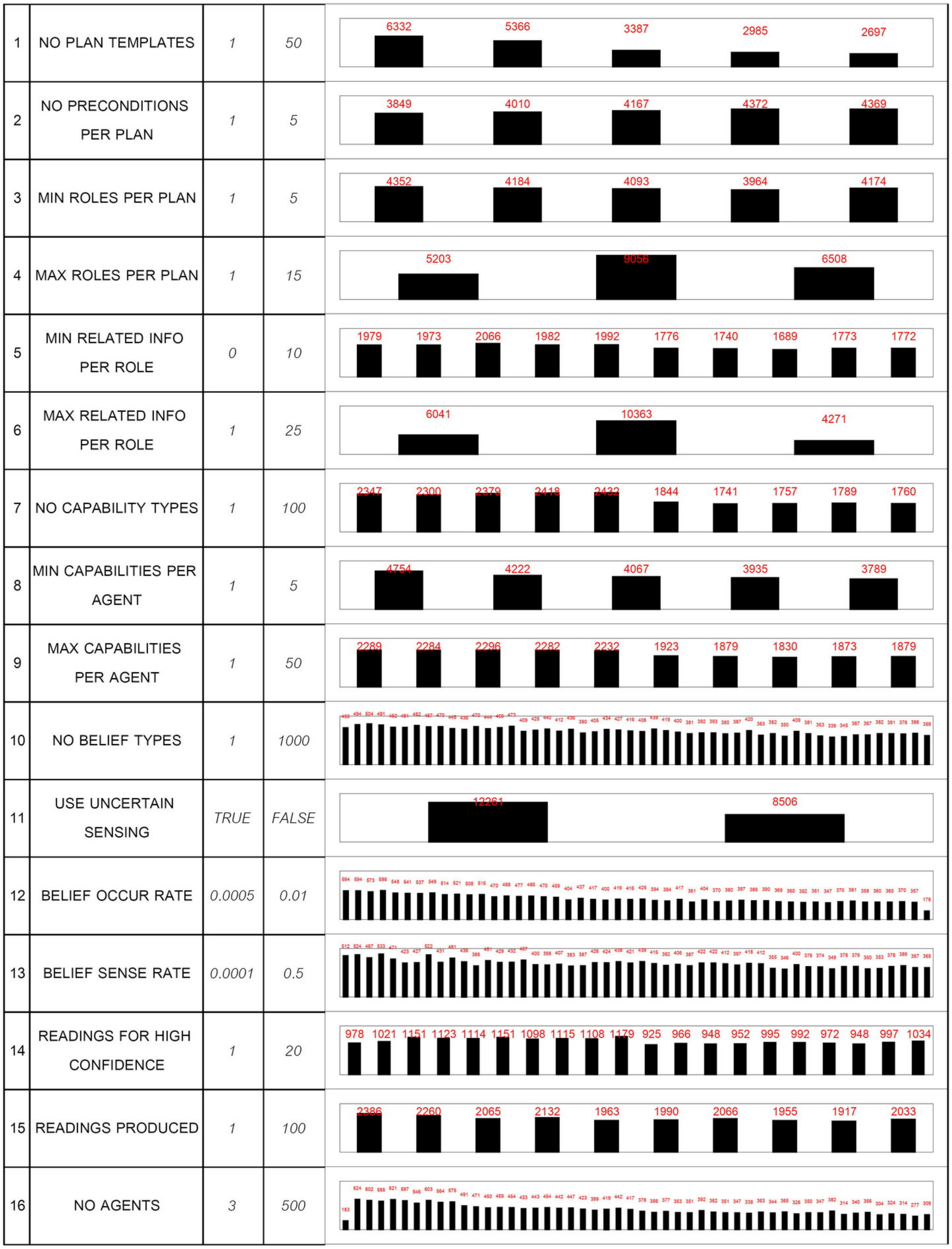
4.2.2 Number of Training Instances

An initial experiment has been done to investigate how many training instances are a sufficient amount for using in the learning process. This number is a tradeoff between precision and speed of the process. With a small number of training instances the learning process can be very fast, but training instances in each generation might be very different. At one time, training instances might contain almost the same training patterns, which do not represent the whole training data that have numerous different patterns. As shown in Figure 4.7, learning with 30 or 50 training instance sample levels to low variety in learning. However, using 50 training instances can achieve more precision. When considering elapsed time, as shown in Figure 4.8, learning from 30 training instances is twice as quick as using 50 training instances. Results indicate that using 30 training instances the learning process is quicker but less precise than using 50 training instances. Fortunately, in this work, using of the hall of fame technique, the learning process could use 30 training instances at start for rapid filtering and use 50 training instance samples at the end to obtain good precision.

4.2.3 Examination of The Hall of Fame

To investigate the usefulness of the hall of fame approach, an experiment was performed. In this experiment, short start up learning processes were executed with the same learning settings with a single exception in runs with or without the existence of the hall of fame. Before the experiment started, preliminary quick experiments were conduced for to collect good performance models. Then, these models were selected for the hall of fame.

All learning processes began with randomly generated initial values. Only a short period of learning times was recorded. Results were measured from ten executions for both two settings. After running all learning process, the learning graphs of both settings are shown in Figure 4.9. From the graphs, the normal learning processes start to perform with high percent of distribution error and then quickly drop and stay at roughly 50 percent of distribution



1	NO PLAN TEMPLATES	*1*	*50*	6332 5366 3387 2985 2697
2	NO PRECONDITIONS PER PLAN	*1*	*5*	3849 4010 4167 4372 4369
3	MIN ROLES PER PLAN	*1*	*5*	4352 4184 4093 3964 4174
4	MAX ROLES PER PLAN	*1*	*15*	5203 9056 6508
5	MIN RELATED INFO PER ROLE	*0*	*10*	1979 1973 2066 1982 1992 1776 1740 1689 1773 1772
6	MAX RELATED INFO PER ROLE	*1*	*25*	6041 10363 4271
7	NO CAPABILITY TYPES	*1*	*100*	2347 2300 2379 2418 2432 1844 1741 1757 1789 1760
8	MIN CAPABILITIES PER AGENT	*1*	*5*	4754 4222 4067 3935 3789
9	MAX CAPABILITIES PER AGENT	*1*	*50*	2289 2284 2296 2282 2232 1923 1879 1830 1873 1879
10	NO BELIEF TYPES	*1*	*1000*	455 494 504 491 462 461 482 467 470 445 436 470 444 456 473 409 425 440 412 436 390 405 434 427 416 408 439 419 400 381 392 393 380 387 420 363 382 350 409 381 353 339 345 367 367 382 381 378 386 359
11	USE UNCERTAIN SENSING	*TRUE*	*FALSE*	12261 8506
12	BELIEF OCCUR RATE	*0.0005*	*0.01*	594 594 573 598 548 541 537 549 514 521 508 515 470 488 477 485 470 459 404 437 417 400 419 416 425 394 384 417 361 404 370 380 397 388 390 369 360 352 361 347 370 381 358 360 360 370 357 178
13	BELIEF SENSE RATE	*0.0001*	*0.5*	512 524 487 533 471 423 427 522 431 481 436 386 461 429 432 467 400 398 407 383 387 426 424 438 421 439 415 392 406 387 422 422 412 397 418 412 355 346 400 376 374 349 378 379 350 353 378 389 367 365
14	READINGS FOR HIGH CONFIDENCE	*1*	*20*	978 1021 1151 1123 1114 1151 1098 1115 1108 1179 925 966 948 952 995 992 972 948 997 1034
15	READINGS PRODUCED	*1*	*100*	2386 2260 2065 2132 1963 1990 2066 1955 1917 2033
16	NO AGENTS	*3*	*500*	183 624 602 588 621 597 546 603 564 575 491 471 450 459 454 433 443 454 442 447 423 399 419 442 417 378 356 377 353 351 392 362 351 347 338 363 344 365 326 350 347 382 314 340 356 304 324 314 277 309

Figure 4.5: Sampling distributions of input parameters in training data (Part 1)

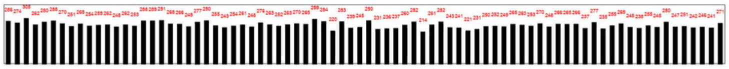
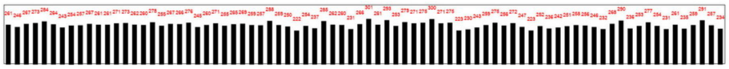
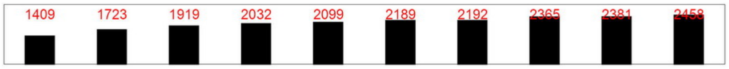
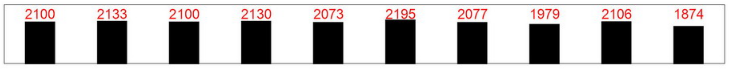
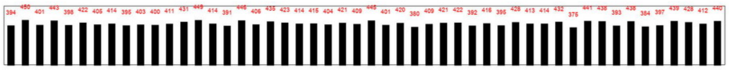
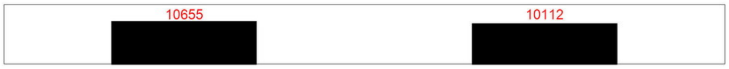
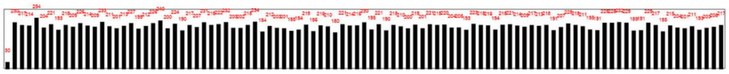
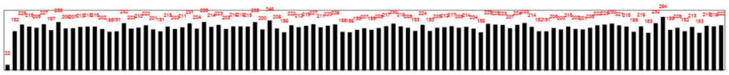
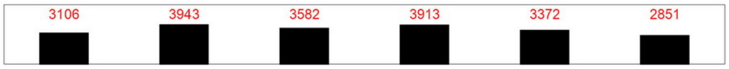
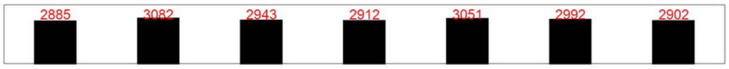
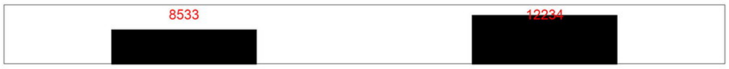
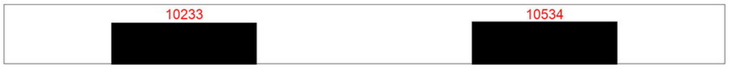
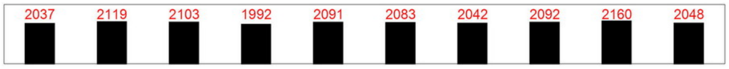
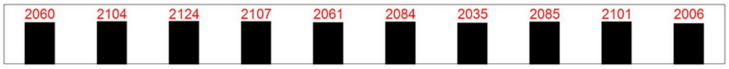
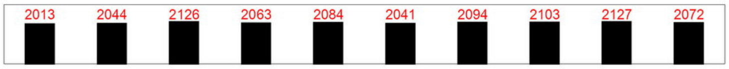
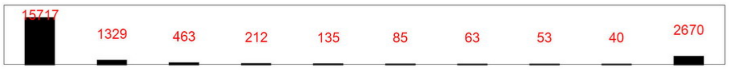
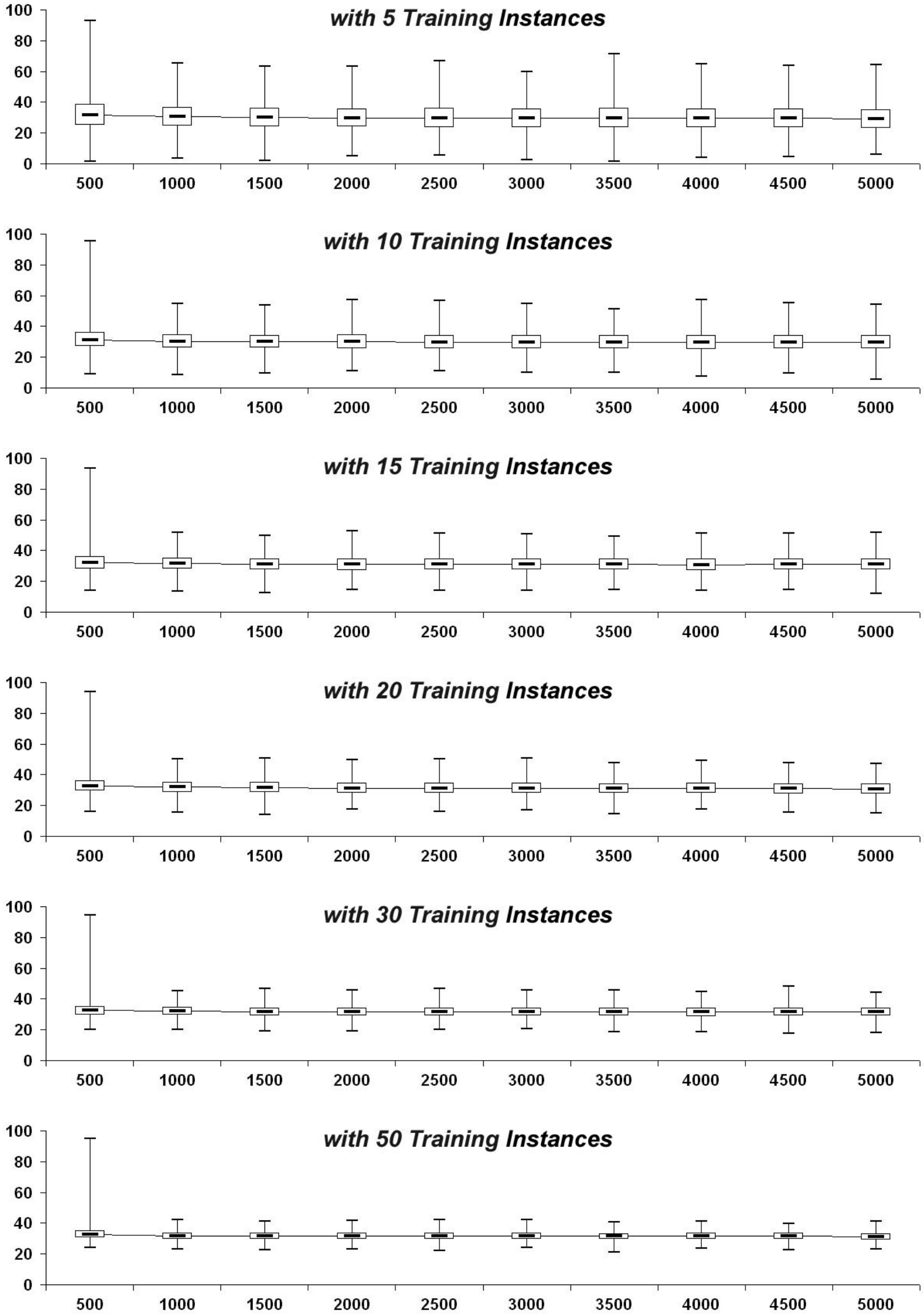
17	BELIEF TOKEN TTL	*1*	*800*	
18	SENSOR TOKEN TTL	*1*	*800*	
19	MESSGAE LOSS RATE	*0*	*0.1*	
20	LADCOP INITIAL THRESHOLD	*0*	*0.99*	
21	SIMPLE ROLE TOKEN MOVE LENGTH	*1*	*500*	
22	ROLE TOKEN SLEEP RULE	*NEVER*	*FIXED*	
23	ROLE SLEEP AFTER HOPS	*20*	*1000*	
24	UNFILLED ROLE SLEEP TIME	*20*	*1000*	
25	NETWORK TYPE	*Complete, Random, Scale-Free, Small-World, RCM, RCL*		
26	NETWORK DENSITY	*2*	*8*	
27	ROUTING MODEL TYPE	*P*	*Random*	
28	PROBABILISTIC P CHOICE	*TRUE*	*FALSE*	
29	REL FOR PRECOND SAME PLAN	*0*	*1*	
30	REL FOR INFO FOR PRECOND FACTOR	*0*	*1*	
31	REL FOR ROLES SAME CAPABILITY	*0*	*1*	
32	SIMULATION LENGTH	*500*	*5000*	

Figure 4.6: Sampling distributions of input parameters in training data (Part 2)



with 5 Training Instances

with 10 Training Instances

with 15 Training Instances

with 20 Training Instances

with 30 Training Instances

with 50 Training Instances

Figure 4.7: Plots display a comparison of learning results with different numbers of training instances using in each generation. Values in y-axis are learning performance (fitness values as Chi-square errors) and values in x-axis are numbers of generations.

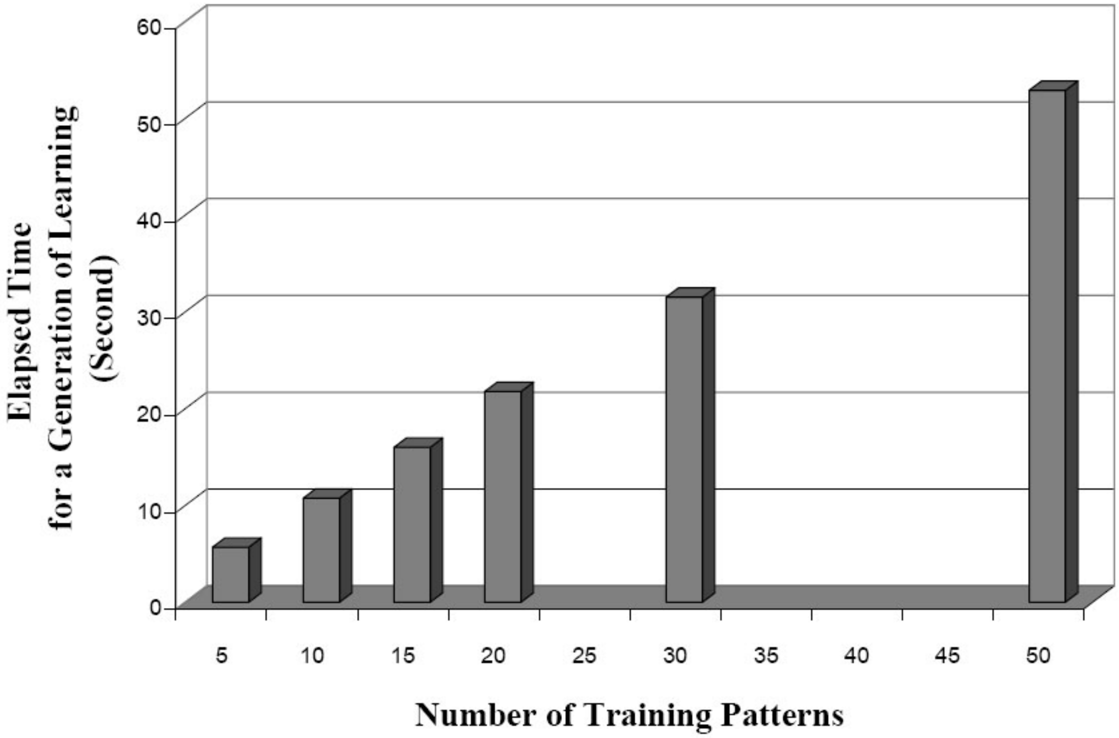


Figure 4.8: A plot display elapsed time of learning process in a generation when using with different numbers of training data.

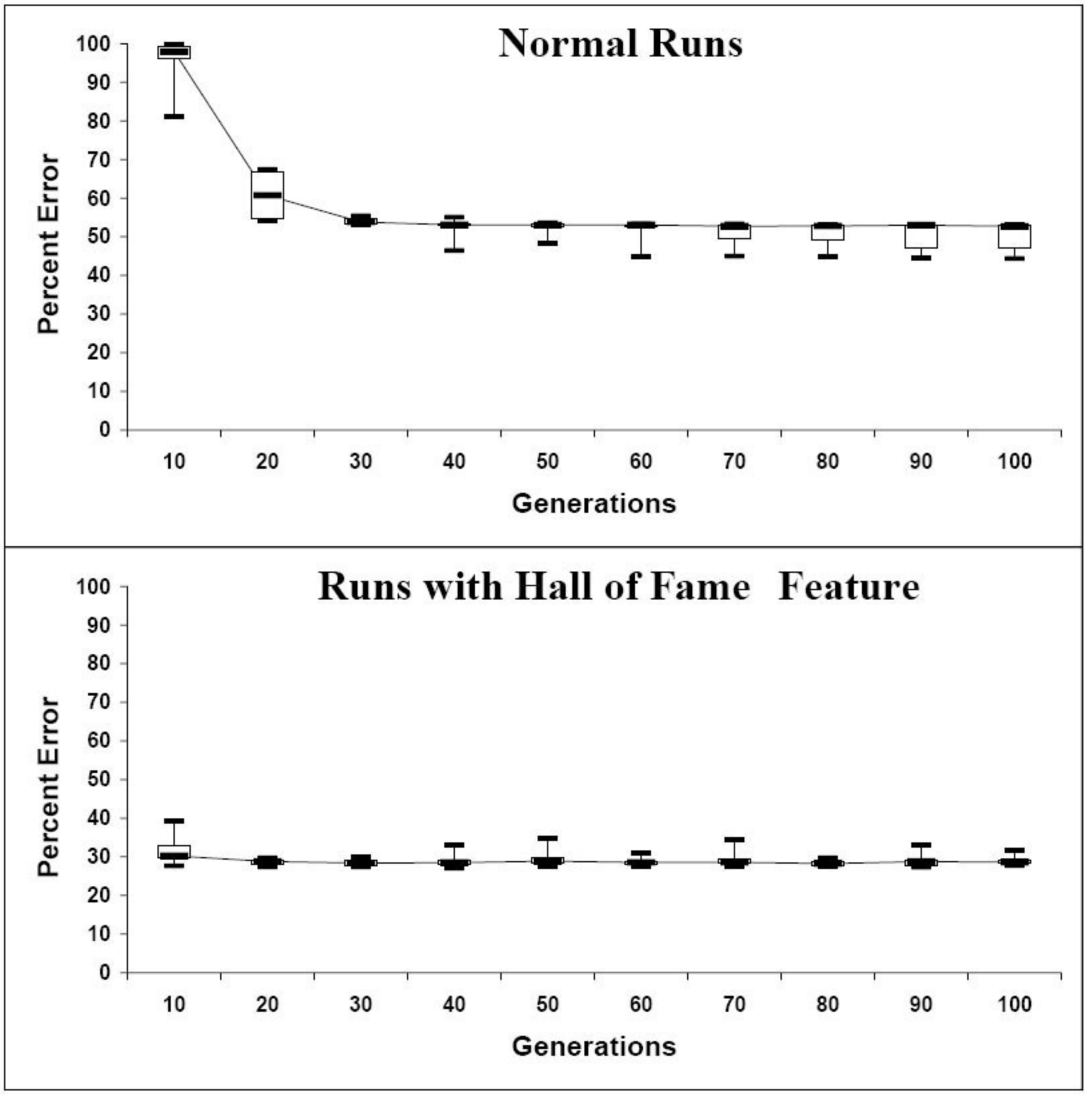


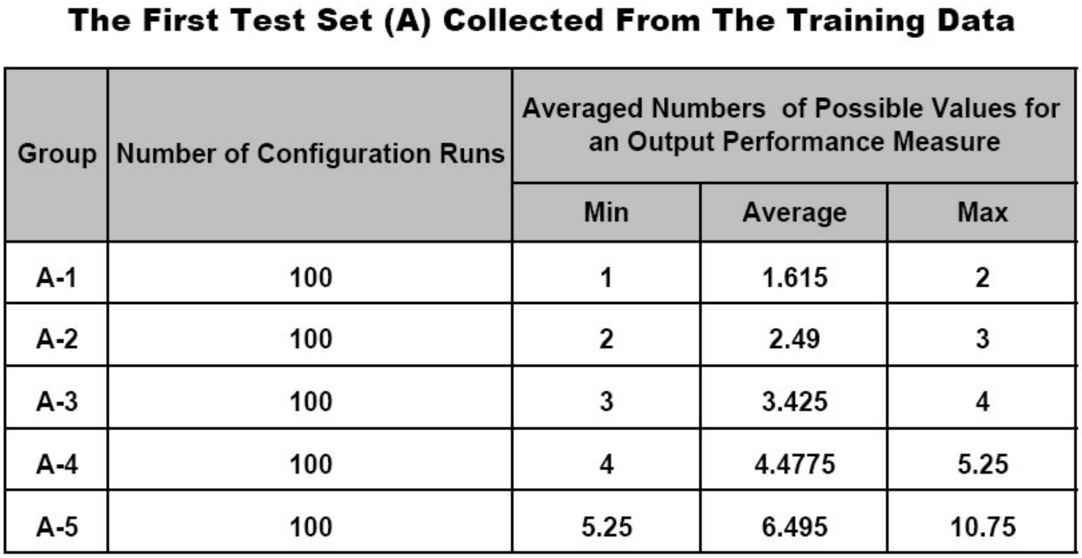
Figure 4.9: Two learning graphs are shown in which learning runs with the hall of fame feature (below) much improved performances than normal runs (above) have. The values were measured from 1200 executions for generating actual distributions.

error. At the same time, with the hall of fame present, the learning processes start to perform at about 30 percent of distribution error and drop a little to 28 percent of distribution error, taking advantage the models in the hall of fame that have performances around 26-34 percent of distribution error. These results show that the hall of fame helps guiding the learning process to quickly start at about the same level of the reserved high performance models.

4.2.4 Test Sets

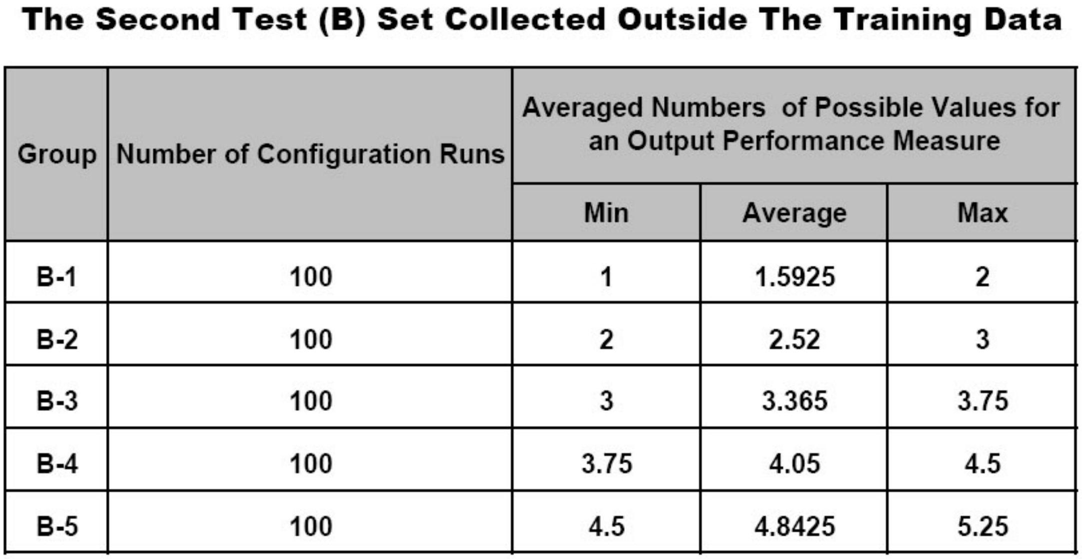
For evaluation of the team performance model, two test sets were created. The first test set is a set of 500 different configuration runs that were randomly picked from the training data. The second test set was a set of another 500 different configuration runs randomly generated outside the training data. For convenience, the first and the second test sets are referred as test set A and test set B respectively. Each configuration run consisted of input parameters for a configuration and its output distributions. In order to investigate how the team performance model handles uncertainty of output performance measures, each test set was separated into 5 groups of 100 configuration runs ranking from most stable outputs to most unstable outputs, as shown in Table 4.1. Sub test sets A-1 and B-1 are considered to have most stable outputs in their test sets for the reason that their numbers of possible values of any output in average are 1.62 and 1.59 respectively. On the other hand, sub test sets A-5 and B-5 are considered to have most unstable outputs in their test sets for because their averaged numbers of possible values of any particular output are 6.49 and 4.84 values respectively.

Table 4.1: Two tables show numbers of possible values for an output performance measure of two test sets (A and B). These numbers are used to separate the test sets into 5 groups of 100 configuration runs.



The First Test Set (A) Collected From The Training Data

Group	Number of Configuration Runs	Averaged Numbers of Possible Values for an Output Performance Measure		
		Min	Average	Max
A-1	100	1	1.615	2
A-2	100	2	2.49	3
A-3	100	3	3.425	4
A-4	100	4	4.4775	5.25
A-5	100	5.25	6.495	10.75



The Second Test (B) Set Collected Outside The Training Data

Group	Number of Configuration Runs	Averaged Numbers of Possible Values for an Output Performance Measure		
		Min	Average	Max
B-1	100	1	1.5925	2
B-2	100	2	2.52	3
B-3	100	3	3.365	3.75
B-4	100	3.75	4.05	4.5
B-5	100	4.5	4.8425	5.25

4.2.5 Team Performance Model Verifications

After executing the learning process for some time (about two months), some good team performance models have been collected. At the end, the best model from this collection was chosen to verify the performance for representing the target system, coordination configurations of large multi-agent teams. In this experiment, there are two main verification processes as follows.

4.2.5.1 Examination With Test Data In this verification, there are two examinations. First, an examination was performed on a test set A, a set of 500 different configuration runs collected from the training data. Second, an examination was performed on a test set B, a set of 500 different configuration runs collected outside the training data. Details of test sets are on section 4.2.4. These two verification processes evaluated the team performance model in term of percent of distribution error. The percent of distribution error was computed by the Equation 3.8. To get this percent of distribution error, the team performance model was executed for a number of executions to generate actual output distributions. Then, the percent of distribution error was calculated from these actual output distributions and the target output distribution from test sets. Because the more numbers of executions, the more accuracy of output distributions, the results are shown in different number of executions.

4.2.5.2 Verification With Team Control Interface In this verification, because of time limitation, only 10 percent of each test set were used to verify the results in the previous verification by using the online testing feature of team control interface. First, the team control interface was loaded with the team performance model. Then, configuration data were loaded for generating statistical information of output performance measures. The statistical information consists of the smallest value (MIN), lower quartile (Q1), median (M), upper quartile (Q3), and largest value (MAX) from 100 executions. This information was then used in the "*run with TokenSim*" feature to measure prediction errors. To compute the prediction error, a special method was used as follows.

```
Run With TokenSim For 500 Steps
Determine The Final Values of All Four Output Performance Measures
Compute The Prediction Error
  $TotalError$ = 0
  for each $Value_{i,o}$ of four output performance measures do
    $Error_o$ = 0
    if ($Value_{i,o}$ >= $Q1_o$)and($Value_{i,o}$ <= $Q3_o$),
        then $Error_o$ = $Error_o$+0
    else, if ($Value_{i,o}$>=$MIN_o$)and($Value_{i,o}$ <= $MAX_o$),
        then $Error_o$ = $Error_o$+1
    else, if ($Value_{i,o}$>=$MIN_o - (MAX_o - MIN_o)$)and($Value_{i,o}$ < $MIN_o$),
        then $Error_o$ = $Error_o$+1+24($MIN_o - Value_{i,o}$)/($MAX_o - MIN_o$)
    else, if ($Value_{i,o}$<=$MAX_o + (MAX_o - MIN_o)$)and($Value_{i,o}$ > $MAX_o$),
        then $Error_o$ = $Error_o$+1+24($Value_{i,o} - MAX_o$)/($MAX_o - MIN_o$)
    else, $Error_o$ = $Error_o$ + 25
    $TotalError$ = $TotalError$ + $Error_o$
  end for
```

In each configuration in a test set, the prediction error measurement started by running each configuration with TokenSim for 500 steps. The reason for running the simulation 500 steps is because the team performance model was trained by training data that were recorded proportionally based on the simulation length of 500 simulation steps. Thus, the simulation length of 500 simulation steps was used to measure the prediction error. After finishing the run, the final values of all four output performance measures were recorded as $Value_{i,o}$, where $i = 1, 2, 3, \ldots, N$ and N denotes the size of a test set, o indicates the output performance measure, $o = 1, 2, 3, 4$. Then, by using the team performance model, five statistical information, consisting of MIN_o, $Q1_o$, M_o, $Q3_o$, and MAX_o, were generated for all four output performance measures by executing the model for 100 times. At the end, the $Value_{i,o}$ was used to compare with MIN_o, $Q1_o$, M_o, $Q3_o$, and MAX_o for computing the prediction error for each output performance measures. Basically this method measures the percent of prediction error by computing how far the actual executions with TokenSim were from the prediction of the team performance model. If the final value ($Value_{i,o}$) was in between the lower quartile ($Q1_o$) and upper quartile ($Q3_o$), it was considered no error. If the value missed that inner range but still was in the bound of minimum (MIN_o) and maximum (MAX_o) values, the error was considered to be a prediction error. Then, if the value missed the total prediction range, the error was computed in proportion to how far it was from the bound of the prediction range. In worse case, if the value was too far, more than the width of the range ($MAX_o - MIN_o$), the error ($Error_o$) is totally 25 percent of prediction error. Because there are four output values to measure, each output has maximum error as 25 percent of prediction error. Totally, the final prediction error ($TotalError$) is summed up to 100 percent.

4.2.6 Reconfiguration Validation

In this validation, the performance of the team performance model and its use through the team control interface were examined. TokenSim, the abstract simulation, was used as the target system. The team control interface was connected directly with TokenSim so that a user could set team configurations and monitor team performance measures online. The user configured the team at the beginning of the mission. Team performances measured from the simulation were graphically displayed on the user interface at every time step. When performance changes were requested, the backward searching feature of the team performance model was used to find suitable reconfigurations.

The team control interface and reconfiguration assistance were evaluated over eight configurations. These configurations were selected from Table A1 to provide situations that would require a user to reconfigure the team in order to meet performance targets. From Table A1, the first 16 input parameters (1-16) are system parameters that could not change. Thus, only the next 15 parameters (17-31) are considered changeable. The backward searching feature utilizes these changeable parameters to find new configurations.

4.2.7 Results and Discussions

In this section, results are presented to validate three aspects of the approach. First, in the first experiment, results are presented that show how the stochastic neural network is able to model a wide range of distributions of outputs. Second, results are provided that show how well the stochastic neural network was able to model the team performance data. Finally, results show how the user interface tool is able to reconfigure a running team.

To create the team performance mode, the learning processes were executed on a number of machines at the same time. Even though the speed of learning was very slow, the progress of learning was persistent. An example of a learning plot is shown in Figure 4.10. In this plot, the performances of generated team performance models were measured with test set A using 1200 executions for generating actual distributions. The results demonstrated that the performance errors were gradually decreasing over time from about 26 to 21 percent of distribution error.

To measure the efficiency of generated models, chi-square errors were used for assessing the fit between the actual output distributions and target output distributions. The smaller this number is the better. However, because this number is not in the form of percentage, the number is converted to percentage form by equation 3.8. Because the number of executions to generate output distributions was small (30 executions each), the reported percent of distribution error is approximate. To present a more accurate picture of likely error, results are shown in five values calculated from five different numbers of executions for generating actual output distributions.

For verification of team performance model with test data, results are shown in Table 4.2. As shown in Table 4.2, results are presented with five different numbers of executions of the model both in test set A and B. When using 30 executions for generating actual output distributions, the percent of distribution error between the actual and target output distributions for test set A and B are 47.74% and 48.46% in average, respectively. When using 300 executions for generating actual output distributions, the percent of distribution error between the actual and target output distributions for test set A and B are 31.42% and 31.89% in average, respectively. When using 1200 executions for generating actual output distributions, the percent of distribution error between the actual and target output distributions for test set A and B are 20.76% and 20.81% in average, respectively. Results show that the more uncertainty in the test sets, the more errors. Sub test set A-1 and B-1 got lowest numbers of percent of distribution error. Likewise, sub test set A-5 and B-5 got highest numbers of percent of distribution error. These results indicate that the model has difficulty capturing high uncertainty. The 21 percent of distribution error produced after 1200 executions is quite good compared to 50 percent of distribution error found from a model producing only average values of outputs without distributions. In fact, if further learning time was allowed, the performance of models at this level might improve beyond 20 percent of distribution error.

Verification results from the team performance model with the team control interface are shown in Table 4.3. Ten configurations were randomly picked from each sub-group for two test sets for testing with the team control interface. Every configuration in this test was performed 30 times to get these averaged results. From 100 configurations from test set A and B, the final assessments of the team performance model were averaged 15.38 and 15.55 percent of prediction error respectively. On average, each sub test set made approximately the same prediction errors. In test set A, the maximum error was 28.77% and minimum error was 5.01%. In test set B, the maximum error was 32.47% and minimum error was 3.33%. In this case, the prediction errors of 25-30% were mostly caused by the completely wrong prediction of one output performance measure. For 15%, generally, the model was correct in all output values, but sometimes the values were unpredictable in one output, especially the reward. The reward values were the most uncertain output in this work. The prediction errors below 10% indicated that the predictions were almost correct in all outputs. In this experiment, there are about 22% (10 and 12 cases in test set A and B respectively). These results are good evidence for verifying the efficiency of team performance model toward prediction of values of output performance measures.

A number of graphic examples of these verification experiments are shown in Appendix A. By using a selected set of example configurations, several plots of target output distributions and actual output distributions and their values of percent of distribution error are revealed. In addition, a number of graphs from the verification experiments with team control interface are shown as well.

For validation of reconfiguration, first, examinations with one constraint were conducted. Each configuration with its four output parameters was reconfigured eight times by backward execution fixing a single constraint for increasing and decreasing each output parameter. A total of 64 reconfiguration cases were performed. Results were averaged from five to ten executions for each case. These averaged results are shown in Table 4.4. Highlighted results are the ones for which the backward searching failed to find new configurations achieving the constraints. Results show that 12 cases out of 64 cases are unsuccessful. When the user asked some performance measures to increase, it did in 25 cases out of 32 cases, increasing on average 435 percent (however, without one outlier, the average increase was 73 percent). When the performance measure did not increase as requested, the decline was only 24 percent (excluding one outlier only 20 percent). When the user asked for a decrease in a single performance measure, a decrease was observed in 28 cases out of 32 cases. Only 3 of the 4 times a requested decrease was not observed were for the same configuration, hinting that its performance measures may have already been very low. When performing correctly, the average decrease was 35 percent and when performing incorrectly, the measure went up on average 115 percent (removing one outlier change this to 29 percent). These test show that the tool generally has the ability to meet simple user requests, although there is significant room for improvement.

Those results that have percent of changes lower than 20 percent (except 0 percent) were considered toss up situations, in which the results were unpredictable. Those results with higher percent of changes were easy to find a new configuration with the backward execution.

Second, examinations with two or more constraints were conducted. Eight cases of different conditions with more constraints were set for searching new configurations. With the same eight start configuration, 64 reconfigurations were performed. As show in Table 4.5, results show that outcomes of 19 reconfigurations were wrong, 10 reconfigurations were not applicable, and 35 reconfigurations were correct. Because all 10 not-applicable results are also predicted by the team performance model, they were considered to be correct predictions. However, in 141 constraints out of the 160 constraints, the change requested by the user was observed correctly. These results show how the system handles more complex requests as well as simple requests.

Notice that in both experiments, in some cases, it may have been impossible to meet the requested change, e.g., a value cannot be increased beyond it maximum, and in other cases, on average the new configuration might have given the correct result, but uncertainties in the outputs led to the observation. Since the configuration space is so large, it is infeasible to determine what percentage of the failures above were due to these reasons.

Finally, examinations with constraints in sequence were conducted. However, these examinations were not for accuracy assessment but demonstrations of the reconfiguration processes. Four examples of reconfigurations are shown in appendix B. All results demonstrate the usefulness of team control interface toward facilitating the user in finding proper configurations in accordance with user requests.

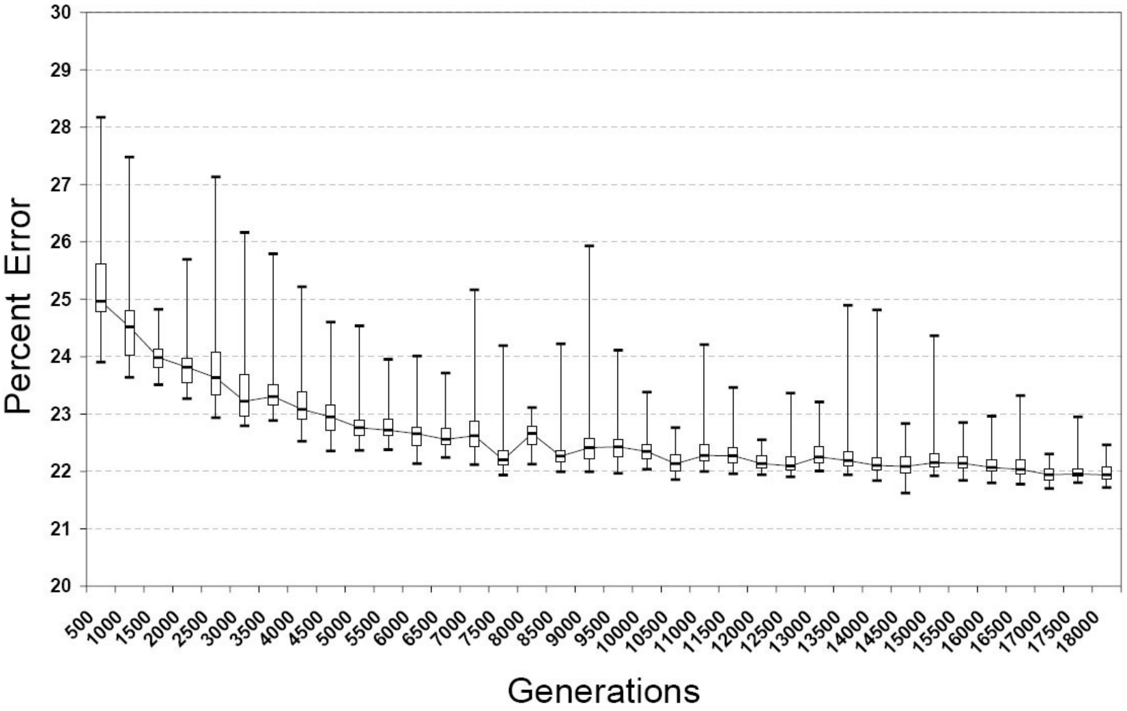
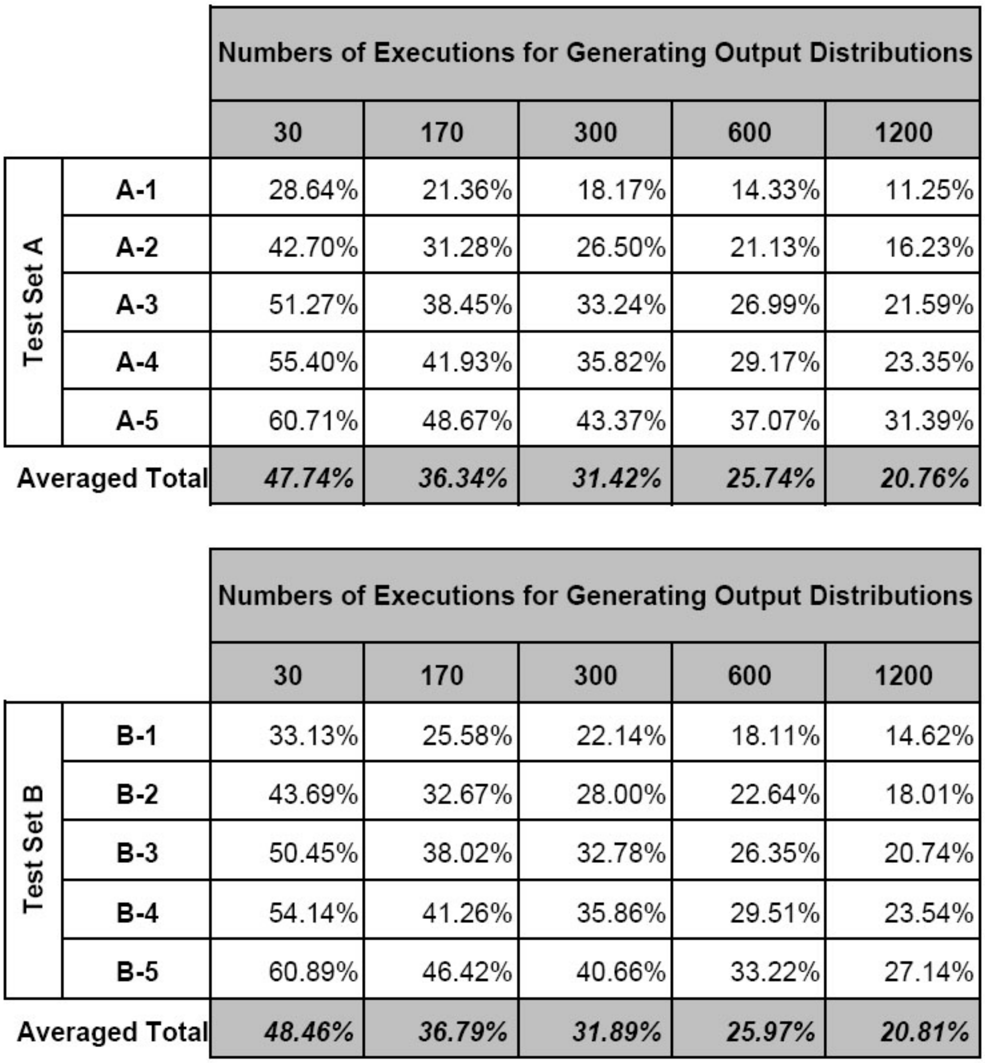


Figure 4.10: A learning plot of a continuous learning execution with the hall of fame approach is shown. The values were measured from 1200 executions for generating actual output distributions. At start, models in the hall of fame have their performance around 26-30 percent of distribution error. This learning process took about three weeks of execution time.

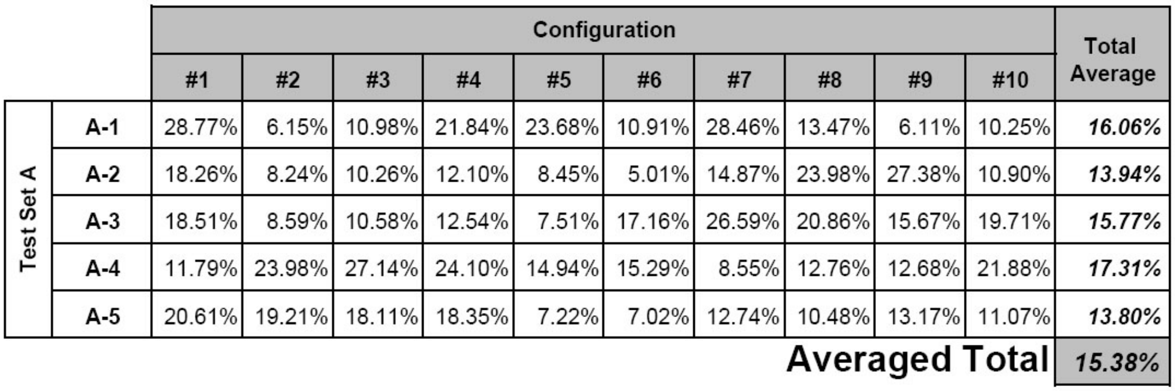
Table 4.2: Two tables show efficiencies (in term of percent of distribution error) of team performance model when they were evaluated with two test sets using different numbers of executions for generating output distributions. The top table shows results from the test set A. The bottom table shows results from the test set B.



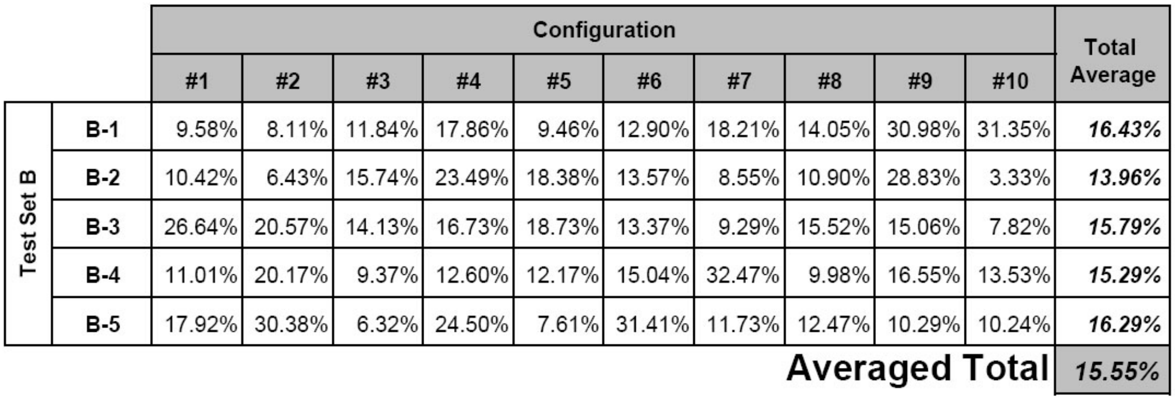
		Numbers of Executions for Generating Output Distributions				
		30	170	300	600	1200
Test Set A	A-1	28.64%	21.36%	18.17%	14.33%	11.25%
	A-2	42.70%	31.28%	26.50%	21.13%	16.23%
	A-3	51.27%	38.45%	33.24%	26.99%	21.59%
	A-4	55.40%	41.93%	35.82%	29.17%	23.35%
	A-5	60.71%	48.67%	43.37%	37.07%	31.39%
Averaged Total		***47.74%***	***36.34%***	***31.42%***	***25.74%***	***20.76%***

		Numbers of Executions for Generating Output Distributions				
		30	170	300	600	1200
Test Set B	B-1	33.13%	25.58%	22.14%	18.11%	14.62%
	B-2	43.69%	32.67%	28.00%	22.64%	18.01%
	B-3	50.45%	38.02%	32.78%	26.35%	20.74%
	B-4	54.14%	41.26%	35.86%	29.51%	23.54%
	B-5	60.89%	46.42%	40.66%	33.22%	27.14%
Averaged Total		***48.46%***	***36.79%***	***31.89%***	***25.97%***	***20.81%***

Table 4.3: Results from verification experiments with team control interface are shown in two tables. The top table shows results in percent of prediction error from the test set A. The bottom table shows results in percent of prediction error from the test set B.

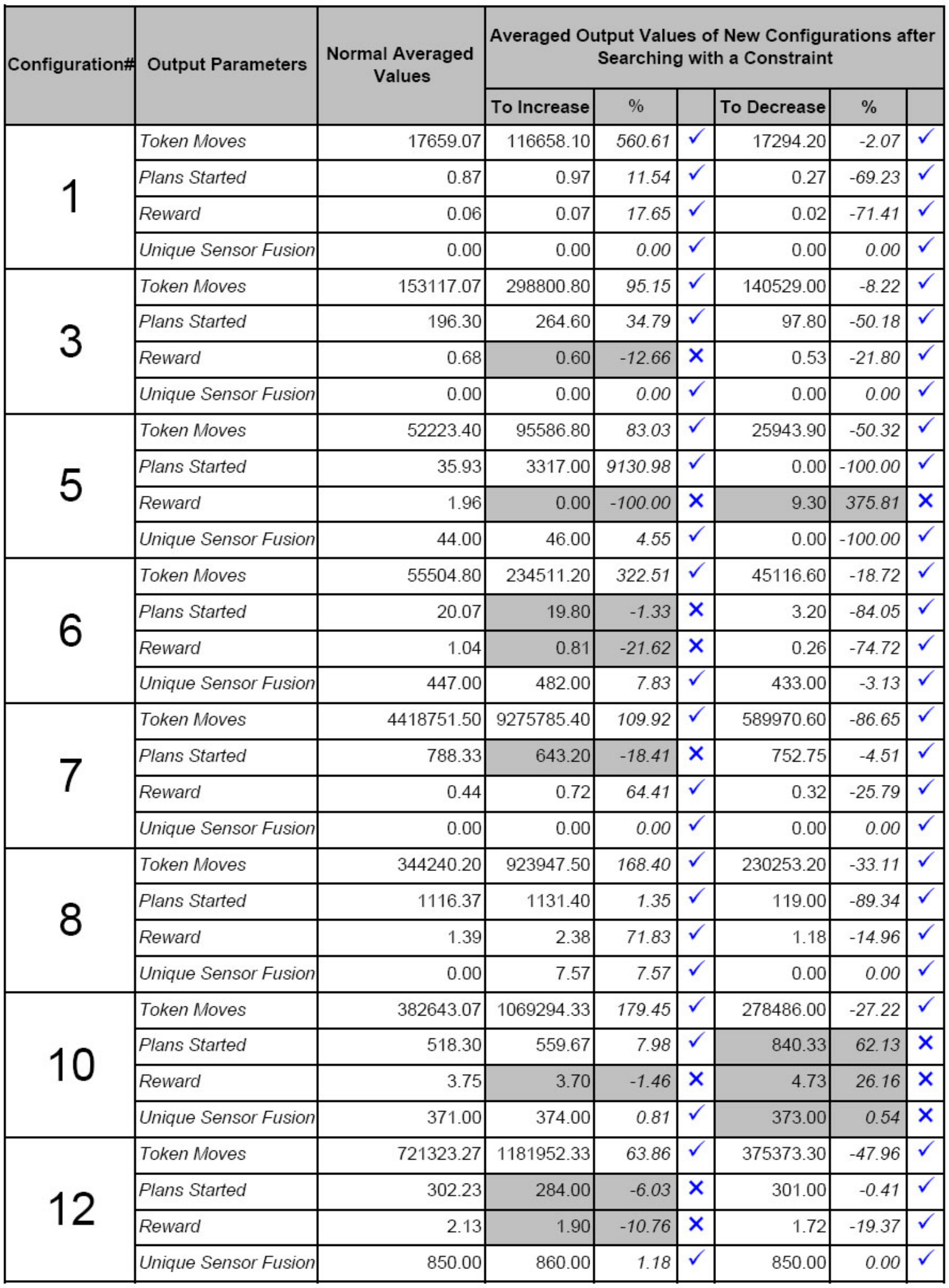


		Configuration										Total Average
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	
Test Set A	A-1	28.77%	6.15%	10.98%	21.84%	23.68%	10.91%	28.46%	13.47%	6.11%	10.25%	***16.06%***
	A-2	18.26%	8.24%	10.26%	12.10%	8.45%	5.01%	14.87%	23.98%	27.38%	10.90%	***13.94%***
	A-3	18.51%	8.59%	10.58%	12.54%	7.51%	17.16%	26.59%	20.86%	15.67%	19.71%	***15.77%***
	A-4	11.79%	23.98%	27.14%	24.10%	14.94%	15.29%	8.55%	12.76%	12.68%	21.88%	***17.31%***
	A-5	20.61%	19.21%	18.11%	18.35%	7.22%	7.02%	12.74%	10.48%	13.17%	11.07%	***13.80%***
											Averaged Total	***15.38%***



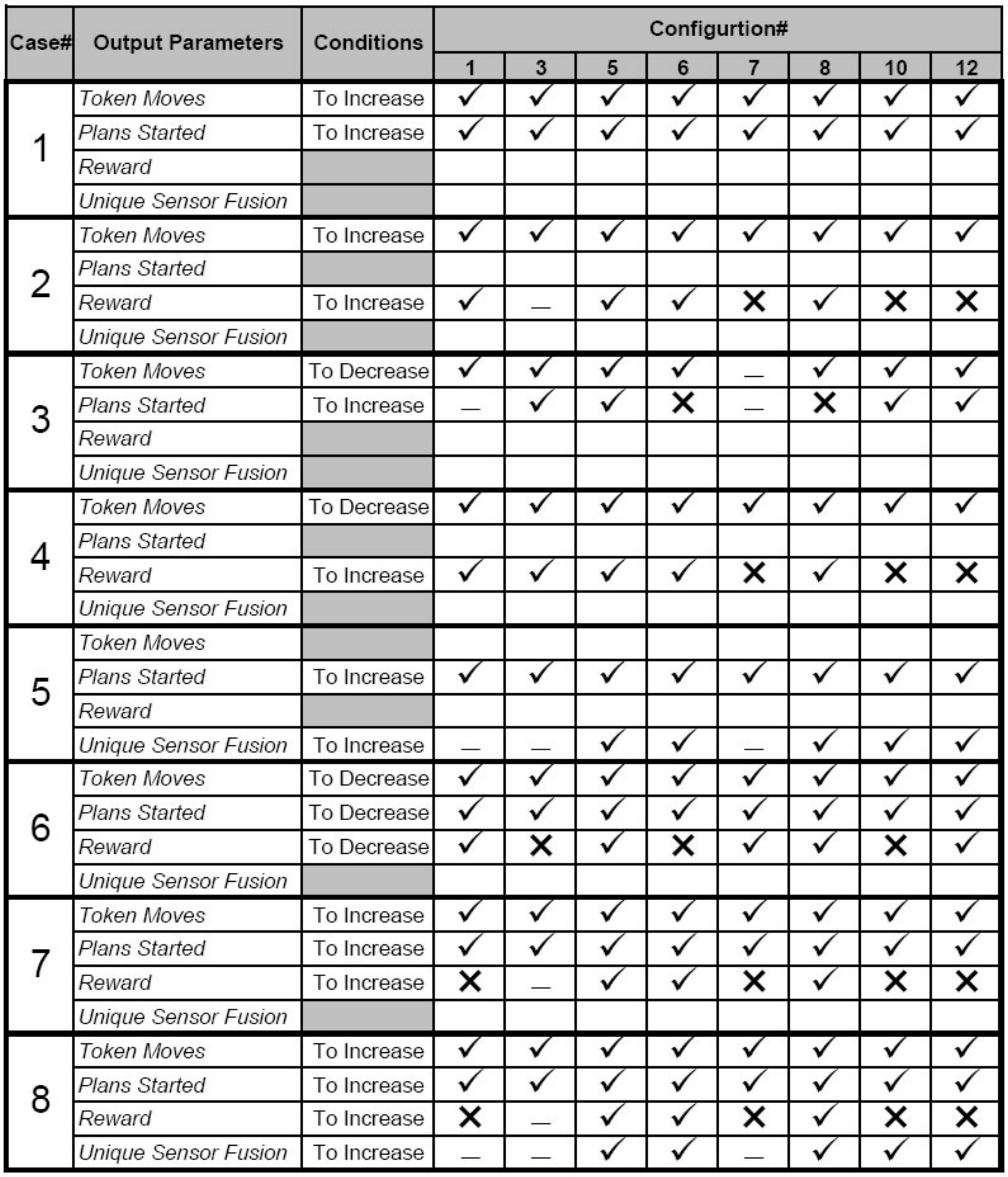
		Configuration										Total Average
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	
Test Set B	B-1	9.58%	8.11%	11.84%	17.86%	9.46%	12.90%	18.21%	14.05%	30.98%	31.35%	***16.43%***
	B-2	10.42%	6.43%	15.74%	23.49%	18.38%	13.57%	8.55%	10.90%	28.83%	3.33%	***13.96%***
	B-3	26.64%	20.57%	14.13%	16.73%	18.73%	13.37%	9.29%	15.52%	15.06%	7.82%	***15.79%***
	B-4	11.01%	20.17%	9.37%	12.60%	12.17%	15.04%	32.47%	9.98%	16.55%	13.53%	***15.29%***
	B-5	17.92%	30.38%	6.32%	24.50%	7.61%	31.41%	11.73%	12.47%	10.29%	10.24%	***16.29%***
											Averaged Total	***15.55%***

Table 4.4: Eight configurations were reconfigured by backward executions with one constraint searching, either to increase or to decrease. Averages results are shown comparing with the normal averaged values, percent of changes, and prediction results ($\surd$ = correct or $\times$ = wrong).



Configuration#	Output Parameters	Normal Averaged Values	Averaged Output Values of New Configurations after Searching with a Constraint					
			To Increase	%		To Decrease	%	
1	*Token Moves*	17659.07	116658.10	*560.61*	✓	17294.20	*-2.07*	✓
	Plans Started	0.87	0.97	*11.54*	✓	0.27	*-69.23*	✓
	Reward	0.06	0.07	*17.65*	✓	0.02	*-71.41*	✓
	Unique Sensor Fusion	0.00	0.00	*0.00*	✓	0.00	*0.00*	✓
3	*Token Moves*	153117.07	298800.80	*95.15*	✓	140529.00	*-8.22*	✓
	Plans Started	196.30	264.60	*34.79*	✓	97.80	*-50.18*	✓
	Reward	0.68	0.60	*-12.66*	×	0.53	*-21.80*	✓
	Unique Sensor Fusion	0.00	0.00	*0.00*	✓	0.00	*0.00*	✓
5	*Token Moves*	52223.40	95586.80	*83.03*	✓	25943.90	*-50.32*	✓
	Plans Started	35.93	3317.00	*9130.98*	✓	0.00	*-100.00*	✓
	Reward	1.96	0.00	*-100.00*	×	9.30	*375.81*	×
	Unique Sensor Fusion	44.00	46.00	*4.55*	✓	0.00	*-100.00*	✓
6	*Token Moves*	55504.80	234511.20	*322.51*	✓	45116.60	*-18.72*	✓
	Plans Started	20.07	19.80	*-1.33*	×	3.20	*-84.05*	✓
	Reward	1.04	0.81	*-21.62*	×	0.26	*-74.72*	✓
	Unique Sensor Fusion	447.00	482.00	*7.83*	✓	433.00	*-3.13*	✓
7	*Token Moves*	4418751.50	9275785.40	*109.92*	✓	589970.60	*-86.65*	✓
	Plans Started	788.33	643.20	*-18.41*	×	752.75	*-4.51*	✓
	Reward	0.44	0.72	*64.41*	✓	0.32	*-25.79*	✓
	Unique Sensor Fusion	0.00	0.00	*0.00*	✓	0.00	*0.00*	✓
8	*Token Moves*	344240.20	923947.50	*168.40*	✓	230253.20	*-33.11*	✓
	Plans Started	1116.37	1131.40	*1.35*	✓	119.00	*-89.34*	✓
	Reward	1.39	2.38	*71.83*	✓	1.18	*-14.96*	✓
	Unique Sensor Fusion	0.00	7.57	*7.57*	✓	0.00	*0.00*	✓
10	*Token Moves*	382643.07	1069294.33	*179.45*	✓	278486.00	*-27.22*	✓
	Plans Started	518.30	559.67	*7.98*	✓	840.33	*62.13*	×
	Reward	3.75	3.70	*-1.46*	×	4.73	*26.16*	×
	Unique Sensor Fusion	371.00	374.00	*0.81*	✓	373.00	*0.54*	×
12	*Token Moves*	721323.27	1181952.33	*63.86*	✓	375373.30	*-47.96*	✓
	Plans Started	302.23	284.00	*-6.03*	×	301.00	*-0.41*	✓
	Reward	2.13	1.90	*-10.76*	×	1.72	*-19.37*	✓
	Unique Sensor Fusion	850.00	860.00	*1.18*	✓	850.00	*0.00*	✓

Table 4.5: Eight cases with two or more constraints, either to increase or to decrease, were applied to eight configurations for reconfigurations. Results are shown if the backward execution is able to suggest new configurations that meet the specific output conditions. (Note, $\surd$ = Correct Predictions, $\times$ = Wrong Preditions, $-$ = Not Applicable)



Case#	Output Parameters	Conditions	Configurtion#							
			1	3	5	6	7	8	10	12
1	*Token Moves*	To Increase	✓	✓	✓	✓	✓	✓	✓	✓
	Plans Started	To Increase	✓	✓	✓	✓	✓	✓	✓	✓
	Reward									
	Unique Sensor Fusion									
2	*Token Moves*	To Increase	✓	✓	✓	✓	✓	✓	✓	✓
	Plans Started									
	Reward	To Increase	✓	—	✓	✓	✗	✓	✗	✗
	Unique Sensor Fusion									
3	*Token Moves*	To Decrease	✓	✓	✓	✓	—	✓	✓	✓
	Plans Started	To Increase	—	✓	✓	✗	—	✗	✓	✓
	Reward									
	Unique Sensor Fusion									
4	*Token Moves*	To Decrease	✓	✓	✓	✓	✓	✓	✓	✓
	Plans Started									
	Reward	To Increase	✓	✓	✓	✓	✗	✓	✗	✗
	Unique Sensor Fusion									
5	*Token Moves*									
	Plans Started	To Increase	✓	✓	✓	✓	✓	✓	✓	✓
	Reward									
	Unique Sensor Fusion	To Increase	—	—	✓	✓	—	✓	✓	✓
6	*Token Moves*	To Decrease	✓	✓	✓	✓	✓	✓	✓	✓
	Plans Started	To Decrease	✓	✓	✓	✓	✓	✓	✓	✓
	Reward	To Decrease	✓	✗	✓	✗	✓	✓	✗	✓
	Unique Sensor Fusion									
7	*Token Moves*	To Increase	✓	✓	✓	✓	✓	✓	✓	✓
	Plans Started	To Increase	✓	✓	✓	✓	✓	✓	✓	✓
	Reward	To Increase	✗	—	✓	✓	✗	✓	✗	✗
	Unique Sensor Fusion									
8	*Token Moves*	To Increase	✓	✓	✓	✓	✓	✓	✓	✓
	Plans Started	To Increase	✓	✓	✓	✓	✓	✓	✓	✓
	Reward	To Increase	✗	—	✓	✓	✗	✓	✗	✗
	Unique Sensor Fusion	To Increase	—	—	✓	✓	—	✓	✓	✓

5.0 CONCLUSIONS

In this section, final conclusions of this research are discussed in greater detail. Then, several aspects of the error measurements used in the experiments are discussed. At the end, possible future work in this research area are recommended.

5.1 RESEARCH CONCLUSIONS

Large scale coordination has become increasingly important as robots and agents become more effective and less expensive. The coordination algorithms controlling the teams need to be heuristic since several key coordination problems are known to be NP-Complete or harder. Thus, to get effective performance in a particular scenario, the algorithms must be carefully tuned. This research presented an approach to online reconfiguration of heuristic coordination algorithms. The approach uses a simulation to produce a large data set which is learned by a stochastic neural network that concisely models the complex, probabilistic relationship between configurations, environments and performance metrics. The final stochastic neural network, referred to as *the team performance model*, is then used as the core of a user interface tool that allows rapid online and offline configuration of coordination algorithms to particular scenarios and user preferences. The overall system allows rapid adaptation of coordination, leading to better performance in new scenarios. Results show that the team performance model captured key features of a very large configuration space and mostly captured the uncertainty in performance well. In addition, the user interface tool with the learned model was shown to be often capable of reconfiguring the algorithms to meet user requests for increases or decreases in performance parameters. In summary, this work represents the first practical approach to quickly reconfigure complex sets of algorithms for a specific application.

The performance of the approach has been demonstrated by two verification experiments on the learned team performance model. First, on the measurement of how well the team performance model captures target output distributions, the model approximated target distributions with an average error of only 20 percent for a very large configuration space (more than 10^{32} possible configurations). This level of performance is quite good when compared with an early attempt to model this complex relationship with decision tree induction (C4.5). The decision tree induction was used with a reduced problem using 14 input attributes, only four distinctive output classes (LOW, MEDIUM, HIGH, VERY HIGH), and only 30,000 cases, to generate 573 classification IF-THEN rules. On test data, these rules performed with only 74.2% correct classification. Even though the task was very simple, the generated tree was considered useless for any further applications. Furthermore, considering the huge variability of the data, having 20 percent of distribution error was quite significant compared with the level of 50 percent of distribution error measured from a particular model that generates only average values of outputs without variation. Basically, this level of performance signifies that the team performance model was capable of capturing most of outputs with normal uncertainty but had trouble in capturing high variation especially for outputs with irregular behaviors. From the collected training data, a number of irregular cases were found in this research. These difficulties may result from the inadequacy of neural network architecture[61]. With a traditional topology of multi-level feed forward neural networks as used in this research, the network has to struggle in representing unsmooth (irregular) relationships, e.g., in a case when one of all parameters is changed by a little bit, but the new corresponding results are very different from the previous results. An experiment for modeling the beta distribution was conducted to show the stochastic neural network's basic ability for representing uncertainty. This beta distribution is a very special mathematical model that is widely considered an ideal tool for uncertainty modeling. Because the process of collecting training data is quick and it has a smooth relationship between input parameters, the experimental setup offered a good ground truth of verification of the proposed approach. From the experiment, the learning process adapted quickly to match the symmetric normal distributions and asymmetric distributions, and later took more time to adapt to the more irregular ones. These results show evidence of those distributions that are commonly found in nature are learned very fast. The final model achieved very good results, less than 10 percent distribution error. Finally, results from this initial experiment validated the basic feasibility of the approach, that it is feasible for stochastic neural networks to represent any real world system with regular relationships.

Second, to measure how accurate the team performance model was in predicting the final output performance measures, the final assessments of the team performance model were 15.465 percent of prediction error on average. This level of performance, was quite excellent because 15 % performance indicates that the team performance model made predictions correctly in almost all output performance measures, but that occasionally one output value was incorrect. The most uncertain output performance measure that was difficult to predict was the reward. In the experiment, there about 22 % of the test results indicate the percent of prediction errors lower than 10 %, which means that the predictions were almost correct in all output performance measures. Moreover, only 12 % of the test results indicates a percent of prediction error in between 25-33 %. These prediction errors were mostly caused by the completely wrong prediction of one output performance measure while the others were fine. These results are excellent evidence for the efficiency of the team performance model for prediction of values of output performance measures.

At the end, the significance of team performance model was verified in the reconfiguration experiments, in which the team control interface was used as a tool for facilitating users in controlling the system. The team control interface was designed to exploit the team performance model for rapidly exploring and providing users with possible configurations that make the target system achieve preferred outcomes. In the team control interface, the team performance model was an important key, so that all features in the interface operated by making use of the team performance model. If we could demonstrate that the team control interface and its functions were useful, the team performance model would have demonstrated its significance. The results of these experiments provide evidence for valuable assessment of team performance model as the backward execution has shown effective outcomes for finding new possible preferred configurations. The results were 81.25 % and

88.125 % corrections for one constraint reconfigurations and multiple constraint reconfigurations respectively. Moreover, the results of sequential reconfigurations with constraints demonstrated the generality of the approach. The only problem was that when the results indicated that the differences of the change were lower than 20 percent, the outcomes might not be as predicted. Since this problem occurred only when the differences are small, they might be caused by system uncertainty. In summary, if the search results indicated more than 20 percent of changes both increasing cases and decreasing cases, most of them will likely be observed.

To achieve this level of performance, the learning process took about 2 months with five computing machines (most of them are PC-Intel Celeron 1.4GHz, >526MB, with Windows or Linux OS) running in parallel independently. Because the proposed learning process requires a lot of computing resources for each generation, the next challenge is how to speed up the learning process. For this reason, it is crucial to find ways to allow good potential models to emerge as fast as they can. First, the number of training instances used in each generation was examined. The study found that even though, with concern about elapsed time, using 50 training instances were determined to give a good trade off between learning performance precision and elapsed time, other lower numbers of training instances also proved useful for identifying initial team performance models and stepping up the learning process as well. An additional technique, called a "*hall of fame*" approach (for details please read section 3.4.4), was used for speeding up the learning process and made it possible to run on different machines allowing possible parallel training processes. Running a number of learning processes at the same time, sometimes with different settings, provides a huge variety of possible models generated from every run. In general, the hall of fame was used to collect good learned team performance models from each learning step, so that every learning process could get benefit from other processes. In the future, to expedite the learning process even more, it would be a great improvement if a parallel computer or a quantum computing machine were available because the learning process used in this research was designed to take advantage of different computing machines running in parallel and Genetic Algorithms process can use quantum properties in order to evaluate each individual model more quickly and effectively.

To model the very complex relationship for coordination configurations of large multi-agent teams, it is a very challenging task to obtain enough training data from a huge parameter space within a limited time. Referred from section 3, the initial step of the over all process is the collection of a large volume of training data. This training data either already exists, e.g., from many training exercises, or needs to be gathered. In this research, training data was collected from an abstracted simulation environment. Several computer years were used to generate simulation data, with each run taking between 1 second and ten minutes, depending on the complexity of the configuration. The process of creating and preprocessing training sets for the learning process is atypical, because distributions of outputs are required. Each training datum was made up of two parts: the input vectors (system parameters, s and configurable parameters, c) and a set of output vectors, p. For the purposes of computing a fitness function during the evolutionary algorithm, the output values are discretized into some number of slots. In this work, 100 slots were typically used. Furthermore, these training data have to be able to represent the entire relationship in some aspect. Unfortunately, it is impossible to collect perfect training data because it takes too long to collect them all. Fortunately, with a statistical analysis, described in section 3.2.1, a smaller number of training data can be collected that can be statistically shown to represent the entire relationship within a confidence interval. With the additional assumption that this complex configuration relationship forms a smooth surface, training data can be simply sampled randomly if the intention is to use all possible ranges of parameters equally. For this work, training data were collected in roughly 21000 samples.

In summary, this work has demonstrated that the new approach effectively provides human users with tools for interacting with a large-scale multi-agent system. The new approach consisting of three key ideas has been successfully implemented and verified. First, the proper training data collection has been designed and implemented to assemble a number of training data that are statistically sufficient to represent a large configuration space and are used in the learning process. The concept of a stochastic neural network has been effectively used to represent the complex relationship between the configuration parameters of coordination algorithms and performance measures for large teams. An effective learning algorithm to train the stochastic neural networks has been developed so that uncertainties in training data are captured. A team performance model has been created for representing the relationships between the environment, team configuration parameters, and performance measures in a way that allows rapid exploration by human users. Then, a team control interface has been designed to use with the team performance model in searching for possible best configurations. Finally, the interface has been used successfully allowing online reconfigurations when human users are working with a large-scale multi-agent system. The main contribution of this research is the development of a new idea for stochastic neural networks that concisely models the complex, probabilistic relationship between configurations, environments and performance metrics. A genetic algorithm is used with a new fitness function for determining whether the output distribution matches the data distribution. The stochastic neural network is then used as the core of the tool that allows rapid online or offline configuration of coordination algorithms to particular scenarios and user preferences. As the results, the overall system allows rapid adaptation of coordination, leading to better performance in new situations. Importantly, the approach that has been developed from this research can also generally apply to any complex, nonlinear interacting system of algorithms, or complex, nonlinear interacting physical processes.

5.2 ERROR MEASUREMENT DISCUSSION

In this work, two error measurements were used for verifying the team performance model. The first method is the percent of distribution error computed by using Pearson Chi-square test [51]. This test is utilized for measuring statistical differences between two output distributions, in this case, actual output distributions and target output distributions. The actual output distributions were obtained from executing the team performance model for a specific number of times. On the other hand, the target output distributions were obtained from the collected training database, in which they were collected by executing TokenSim 30 times per configuration. To get the percentage of difference error, equation 3.8 was used. This equation mainly depends on the values of Chi-square test (χ^2) and total number of frequencies from the two output distributions ($N = n_a + n_t$). In a case that the actual output distributions are made from executing team performance model for 30 times, i.e., n_a is 30, since n_t is always 30, and then N is 60. In another case that the actual output distributions are made from executing team performance model for 300 times, i.e., n_a is 300, and then N is 330. In general, if the output distributions are made from high numbers of executions, better precision is expected. Unfortunately, while actual output distributions can be obtained from executing team performance model many times to gain better precision, target output distributions cannot. As a result, the results of verification in Table 4.2 are shown with different numbers of executions for the actual output distributions. Results show that the larger the sample of actual distributions the less of percentage of error. While this method is a statistical assessment for the team performance model toward representing the target system, it is not certain which value is the real error assessment for the team performance model, since the target output distributions have very small samples for comparison. However, in this research, I mainly used the actual output distributions produced after 1200 executions considering error precision and execution time. Overall results show excellent potential of the approach for representing a very complex system. However, from the experiments showed there was a training difficulty in that the learning process took a long time to capturing high uncertainty outputs. Even though the final averaged percent of distribution error was around 21 and 16 percent for output distributions produced after 1200 and 2500 executions respectively, if further learning time were allowed, the performance of models might improve significantly.

In addition, the second error assessment was implemented as the percentage of prediction error. This method measures how accurate the team performance model was in predicting the actual final output performance measures in a form of possible ranges of possible output parameters. The predictions are based on the statistical information of possible output performance measures generated by executing the team performance model for 100 times with test coordination configurations, i.e., the smallest value (MIN), lower quartile (Q1), median (M), upper quartile (Q3), the largest value (MAX). The measure utilizes a reasonable plausible measurement to get the percentage of prediction error, as described in section 4.2.5.2. In general, if the final performance outputs are in the ranges predicted by the team performance model, the errors are very small. If the final performance outputs fall outside the ranges, the errors are computed from their distances from the bounds of the range. Figure 5.1 depicts how to assess the percent of prediction error from an error bar. The idea behind this method is to use the range between Q1 and Q3 as the primary target prediction and the range between MIN and MAX as the secondary target prediction, so that the difference between primary and secondary target prediction is determined to be 1% prediction error. Other than that, the percent of prediction error is determined from the distance between the final performance outputs and the boundary of the error bar. However, if the distance is far beyond the length of MAX-MIN, the maximum prediction error is applied. Normally, a prediction error is computed from all output performance measures independently. Since there are totally four output performance measures for each configuration, each one has the maximum 25 percent of prediction error so that the total percent of prediction error of all four outputs are summed up to 100 percent of prediction error. The decision to use summation of all fours prediction errors for representing the total percent of prediction error is to give a better impression of the relative results across four performance outputs. For example, see Figure A4 in Appendix A, configuration♯5 got 6.82 percent of prediction error, in which the number gives an impression that all final performance outputs were just a bit off. On the other hand, configuration♯6 got 26.0 percent of prediction error, in which the number provides an impression that one performance output was completely wrong and the others were fine. In Figure A5, configuration♯10 got 33.75 percent of prediction error, in which the number presents an impression that two performance outputs were somewhat off. While this method is an ad hoc measurement, it provides a good assessment for the usefulness of the team performance model toward searching possible configurations. Results, in Table 4.3, show that the team performance model is reliable for predicting the possible ranges of output performance measures. From the experiments, the team performance model performed about 15% prediction error on average, in which the model was correct in all output values, but sometimes the values were unpredictable in one output.

To give a better impression of the difference between these two error assessments, Table 5.1 shows a comparison among six different team performance models. These six team performance models were picked during the learning process, so that they have different levels of presentation for representing the target system. The first team performance model was picked at the early stage of learning process, so that it has a very poor representation model of the target system. The second, third, fourth and fifth team performance models are generated from the learning process accordingly. Finally, the sixth team performance model was the best model generated from the learning process, i.e., it has a best representation model of the target system so far. From the Table 5.1, it is noticeable that all team performance models are in order of their levels of representation in which the model♯1 is the worse and the model♯6 is the best. First, considering the first error assessment, with different number of executions, the values of percent of distribution error are decreased accordingly when better team performance models were used. When using 30 executions for generating actual output distributions, the values of percent of distribution error were stuck at 47 percent. This effect is probably caused by having a small number of frequencies from the two output distributions. When using more numbers of execution, e.g., 300, 1200 or 2500 executions, the values of percent of distribution error are improved as the number of execution increases. Each of them gives different values of percent of distribution error, in which it was not sure which one is the real error assessment since the target distributions are based on small samples. However, the results in this research were shown mainly by using 1200 executions. Furthermore, the values of percent of prediction error are very close to percent of distribution error generated from 2500 executions, except when the values of percent of distribution error were higher than 50% the percent of prediction errors were nearly 100%. The possible reason is that the good team performance models normally have their means of actual output distributions very close to the means of target output distributions, and when using more numbers of executions (2500 executions), the widths between the minimum and maximum values of output distributions spread out wider, resulting in getting better percent of distribution error and lowering the percent of prediction error. On the other hand, the models, which have percent of distribution error lower than 50%, normally produce the prediction ranges very far from the actual values, so that the percent of prediction errors were nearly 100%. Interestingly, even though these two error assessments were used differently for verification of team performance model in this research, they are connected when using high number of executions (2500) for computing percent of distribution error results were much closer to the results computed from percent of prediction error.

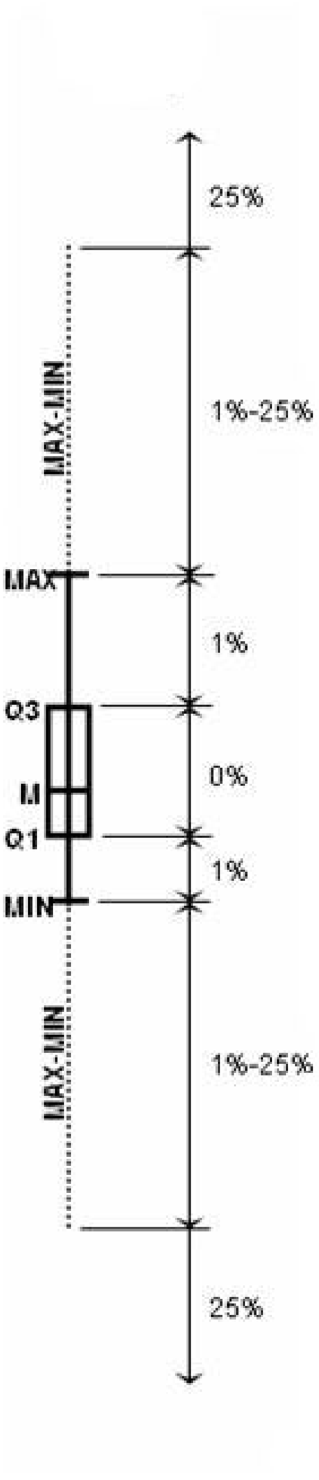
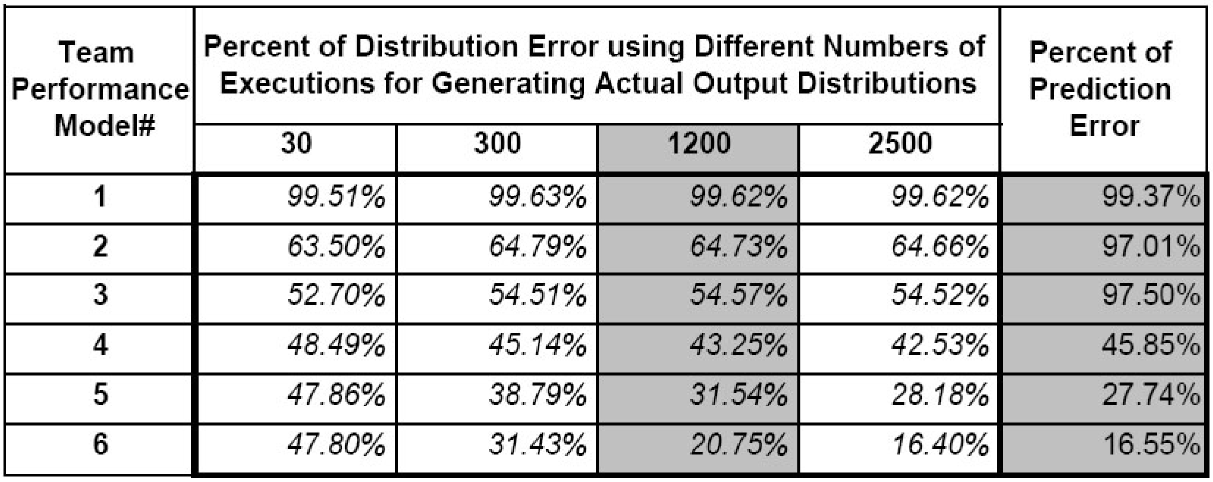


Figure 5.1: Percent of prediction error is computed from the error bar generated from team performance model.

Table 5.1: Six different team performance models are shown with their error assessments, both percent of distribution error with different numbers of executions and percent of prediction error. Columns with shaded highlight show two error assessments that were mainly used in this research.



Team Performance Model#	Percent of Distribution Error using Different Numbers of Executions for Generating Actual Output Distributions				Percent of Prediction Error
	30	300	1200	2500	
1	99.51%	99.63%	99.62%	99.62%	99.37%
2	63.50%	64.79%	64.73%	64.66%	97.01%
3	52.70%	54.51%	54.57%	54.52%	97.50%
4	48.49%	45.14%	43.25%	42.53%	45.85%
5	47.86%	38.79%	31.54%	28.18%	27.74%
6	47.80%	31.43%	20.75%	16.40%	16.55%

5.3 RECOMMENDATIONS FOR FUTURE RESEARCH

While this research shows the potential for using stochastic neural networks to model complex coordination behavior, significant work remains. One question is how to obtain greater precision in presenting the complex relationships within such large dynamic models. One possible way to enhance the capability of stochastic neural networks is to modify the neural network as proposed in [61]. In [61], the new architecture of neural networks enhances the network abilities for handling various numbers of input data with different hidden neural parts in a network. This architecture consists of a three layer neural network, in which all hidden nodes are interconnected, making it possible for the network to learn relationships with a non smooth surface. Consequently, in theory, any complex system could be modeled with this new stochastic neural network. However, the technique to train this network still is a big question. Even though using an adapted genetic algorithms approach with distribution comparisons has shown promising results in this work, it does not guarantee optimal solutions. In fact, to model a very complex system with uncertainty present, it is difficult to guarantee optimal solutions. One possible improvement is to speed up the learning process so that it is practical to implement in real applications, e.g., genetic regulatory systems[48] and prediction of trading values in stock markets [4]. Another issue is how well results or even modeling approaches may generalize between application domains. The approaches should find an effective way to adapt when target algorithms have been changed so that the entire process does not start from the beginning again. For example, a small group of five mobile robots has been trained to effectively explore and create a map for a particular area. Later, the robots have added an additional capacity to use video for searching for injured people. The approach should be able to use the first learned model to adapt for the second task more easily. In this case, new additional input nodes and output nodes could be added to the model connecting to the nearest hidden layer with zero weights. The new model would preserve the functioning of the old model before training with new training data, so that the new task can be learned faster. While the results have qualitatively demonstrated that this approach produced a model that behaves as we believe it should, quantitative measures and comparisons with alternate approaches are needed to support this claim with greater certainty. By using complex common tasks that have been used in literatures, e.g., forecasting of the sunspot numbers[118, 74], hand writing interpretation[61], the approach can be applied and made quantitatively comparisons with other approaches.

To evaluate the results of this work with a high fidelity simulation, a scenario involving a large-scale multi-agent system could be used. This testing scenario is a military simulation, which runs on the Machinetta [90] platform. The proxies provide the general purpose coordination infrastructure that connects the assets in the environment, the human commander, and the terrain analysis agent. The environment simulation, called Sanjaya [102], is a constructive simulation supporting the simulation of ground, air, and unmanned aerial vehicles. This simulation can be designed to present the uncertainty and confusion that can occur in large-scale coordination in real-world environments. The human commander is a member of the team represented by a proxy. The proxy handles the coordination for the teamwork, but a domain-specific interface is required between the commander and his proxy to ensure that the proxy represents the commander's intentions effectively during the coordination of team activities. The terrain analysis agent is a special member of the team who provides a tactical analysis of the terrain to the rest of the team. With this multi-agent infrastructure, a challenging scenario can be designed for evaluating the proposed approach with a human commander. A user study can be conducted, investigating aspects of how the system helps users to achieve improved task performance.

In addition, evaluation of the design of interactive user interface that facilitates intelligible interactions between the user and the multi-agent system is an interesting issue for future development.

APPENDIX A

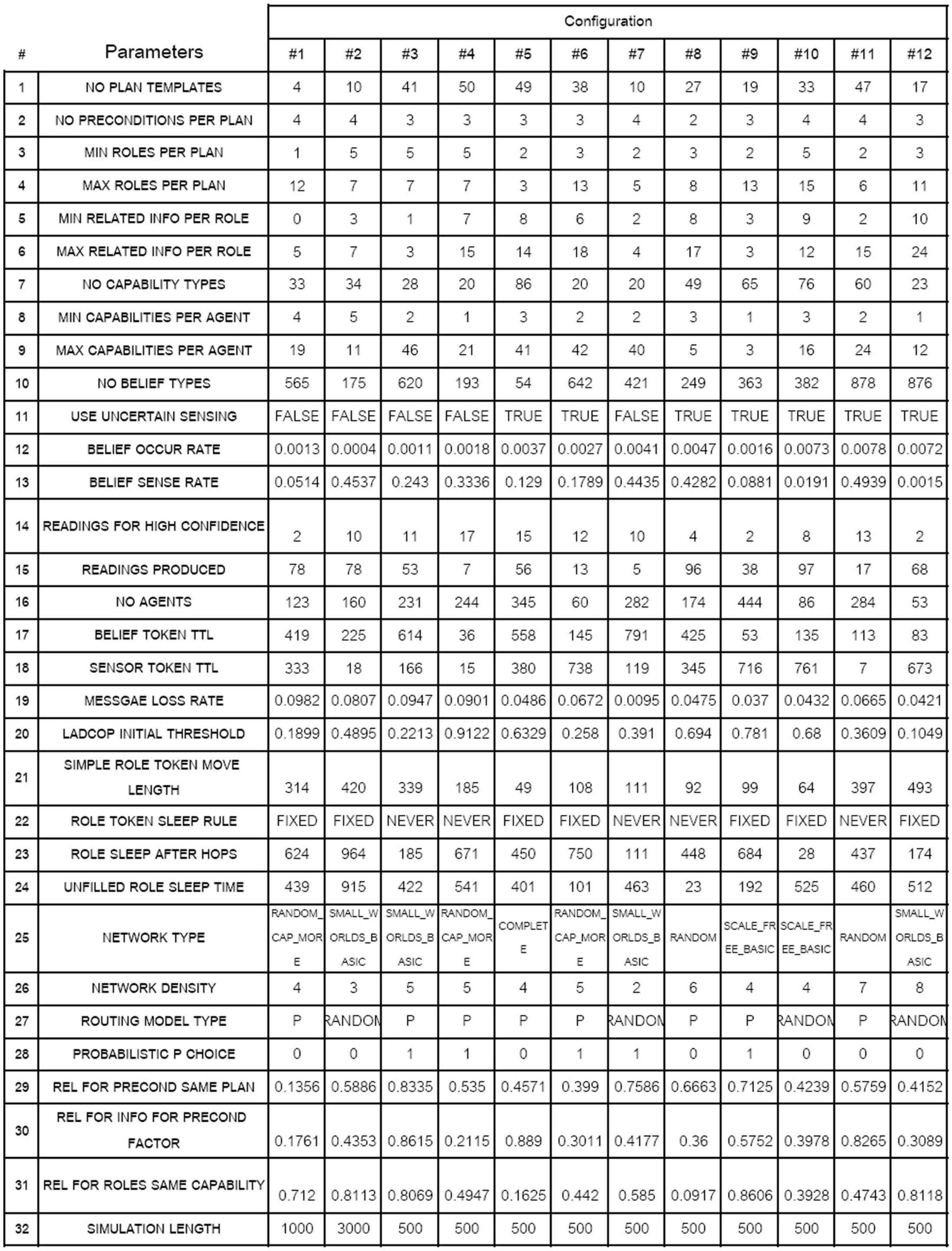
EXAMPLES OF TEAM PERFORMANCE MODEL VERIFICATIONS

While quantitative results have been shown in the experiments and results section, to get an idea what the results look like visually, examples are shown in this section. Twelve configurations were selected, as shown in Table A1, representing all levels of learning difficulty. Configuration 1 was recognized as the easiest one and configuration 12 was recognized as the hardest one.

As shown in Figure A1 and A2, all twelve example configurations are plots with their four output distributions. Target distributions are displayed as arrow poles. Actual distributions generated by team performance model are displayed as diamond poles. Out1, out2, out3, and out4 represent four output performance measures; *token moves*, *plant started*, *reward*, and *unique sensor fusion*, respectively. Figure A1 shows the first half of twelve example configurations that receive good learning performance with lower percent of distribution error. On other hand, Figure A2 demonstrates the second half of twelve example configurations that receive poor learning performance with higher percent of distribution error.

For verification experiments with team control interface, examples are shown in Figure A3, A4, and A5. In Figure A3, the first four configurations (1-4) are shown with their prediction error assessments. Then, the next four configurations (5-8) and the last four configurations (9-12) are shown with their prediction error assessments in Figure A4 and Figure A5, respectively. The total prediction errors were summed up from four prediction errors of output performance measures. The box plots represent the output predictions at current step of execution.

Table A1: Twelve configurations are listed for using as examples for verifications.



#	Parameters	Configuration											
		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
1	NO PLAN TEMPLATES	4	10	41	50	49	38	10	27	19	33	47	17
2	NO PRECONDITIONS PER PLAN	4	4	3	3	3	3	4	2	3	4	4	3
3	MIN ROLES PER PLAN	1	5	5	5	2	3	2	3	2	5	2	3
4	MAX ROLES PER PLAN	12	7	7	7	3	13	5	8	13	15	6	11
5	MIN RELATED INFO PER ROLE	0	3	1	7	8	6	2	8	3	9	2	10
6	MAX RELATED INFO PER ROLE	5	7	3	15	14	18	4	17	3	12	15	24
7	NO CAPABILITY TYPES	33	34	28	20	86	20	20	49	65	76	60	23
8	MIN CAPABILITIES PER AGENT	4	5	2	1	3	2	2	3	1	3	2	1
9	MAX CAPABILITIES PER AGENT	19	11	46	21	41	42	40	5	3	16	24	12
10	NO BELIEF TYPES	565	175	620	193	54	642	421	249	363	382	878	876
11	USE UNCERTAIN SENSING	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
12	BELIEF OCCUR RATE	0.0013	0.0004	0.0011	0.0018	0.0037	0.0027	0.0041	0.0047	0.0016	0.0073	0.0078	0.0072
13	BELIEF SENSE RATE	0.0514	0.4537	0.243	0.3336	0.129	0.1789	0.4435	0.4282	0.0881	0.0191	0.4939	0.0015
14	READINGS FOR HIGH CONFIDENCE	2	10	11	17	15	12	10	4	2	8	13	2
15	READINGS PRODUCED	78	78	53	7	56	13	5	96	38	97	17	68
16	NO AGENTS	123	160	231	244	345	60	282	174	444	86	284	53
17	BELIEF TOKEN TTL	419	225	614	36	558	145	791	425	53	135	113	83
18	SENSOR TOKEN TTL	333	18	166	15	380	738	119	345	716	761	7	673
19	MESSGAE LOSS RATE	0.0982	0.0807	0.0947	0.0901	0.0486	0.0672	0.0095	0.0475	0.037	0.0432	0.0665	0.0421
20	LADCOP INITIAL THRESHOLD	0.1899	0.4895	0.2213	0.9122	0.6329	0.258	0.391	0.694	0.781	0.68	0.3609	0.1049
21	SIMPLE ROLE TOKEN MOVE LENGTH	314	420	339	185	49	108	111	92	99	64	397	493
22	ROLE TOKEN SLEEP RULE	FIXED	FIXED	NEVER	NEVER	FIXED	FIXED	NEVER	NEVER	FIXED	FIXED	NEVER	FIXED
23	ROLE SLEEP AFTER HOPS	624	964	185	671	450	750	111	448	684	28	437	174
24	UNFILLED ROLE SLEEP TIME	439	915	422	541	401	101	463	23	192	525	460	512
25	NETWORK TYPE	RANDOM_CAP_MORE	SMALL_WORLDS_BASIC	SMALL_WORLDS_BASIC	RANDOM_CAP_MORE	COMPLETE	RANDOM_CAP_MORE	SMALL_WORLDS_BASIC	RANDOM	SCALE_FREE_BASIC	SCALE_FREE_BASIC	RANDOM	SMALL_WORLDS_BASIC
26	NETWORK DENSITY	4	3	5	5	4	5	2	6	4	4	7	8
27	ROUTING MODEL TYPE	P	RANDOM	P	P	P	P	RANDOM	P	P	RANDOM	P	RANDOM
28	PROBABILISTIC P CHOICE	0	0	1	1	0	1	1	0	1	0	0	0
29	REL FOR PRECOND SAME PLAN	0.1356	0.5886	0.8335	0.535	0.4571	0.399	0.7586	0.6663	0.7125	0.4239	0.5759	0.4152
30	REL FOR INFO FOR PRECOND FACTOR	0.1761	0.4353	0.8615	0.2115	0.889	0.3011	0.4177	0.36	0.5752	0.3978	0.8265	0.3089
31	REL FOR ROLES SAME CAPABILITY	0.712	0.8113	0.8069	0.4947	0.1625	0.442	0.585	0.0917	0.8606	0.3928	0.4743	0.8118
32	SIMULATION LENGTH	1000	3000	500	500	500	500	500	500	500	500	500	500

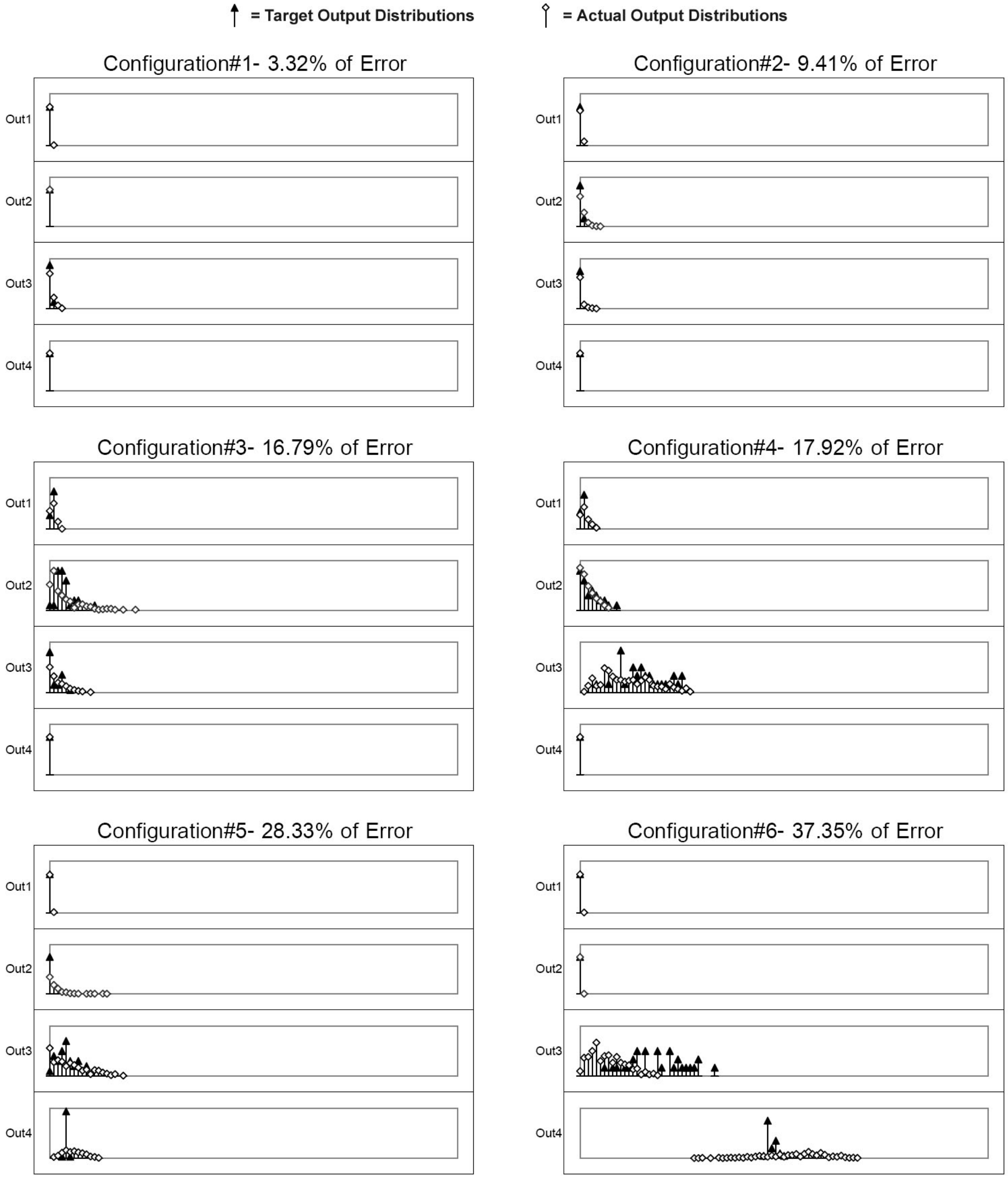


Figure A1: Six plots of the first six example configurations are shown with their percent of distribution error. Each configuration consists of four output distribution plots. Target distributions are displayed as arrow poles. Actual distributions generated by team performance model are displayed as diamond poles.

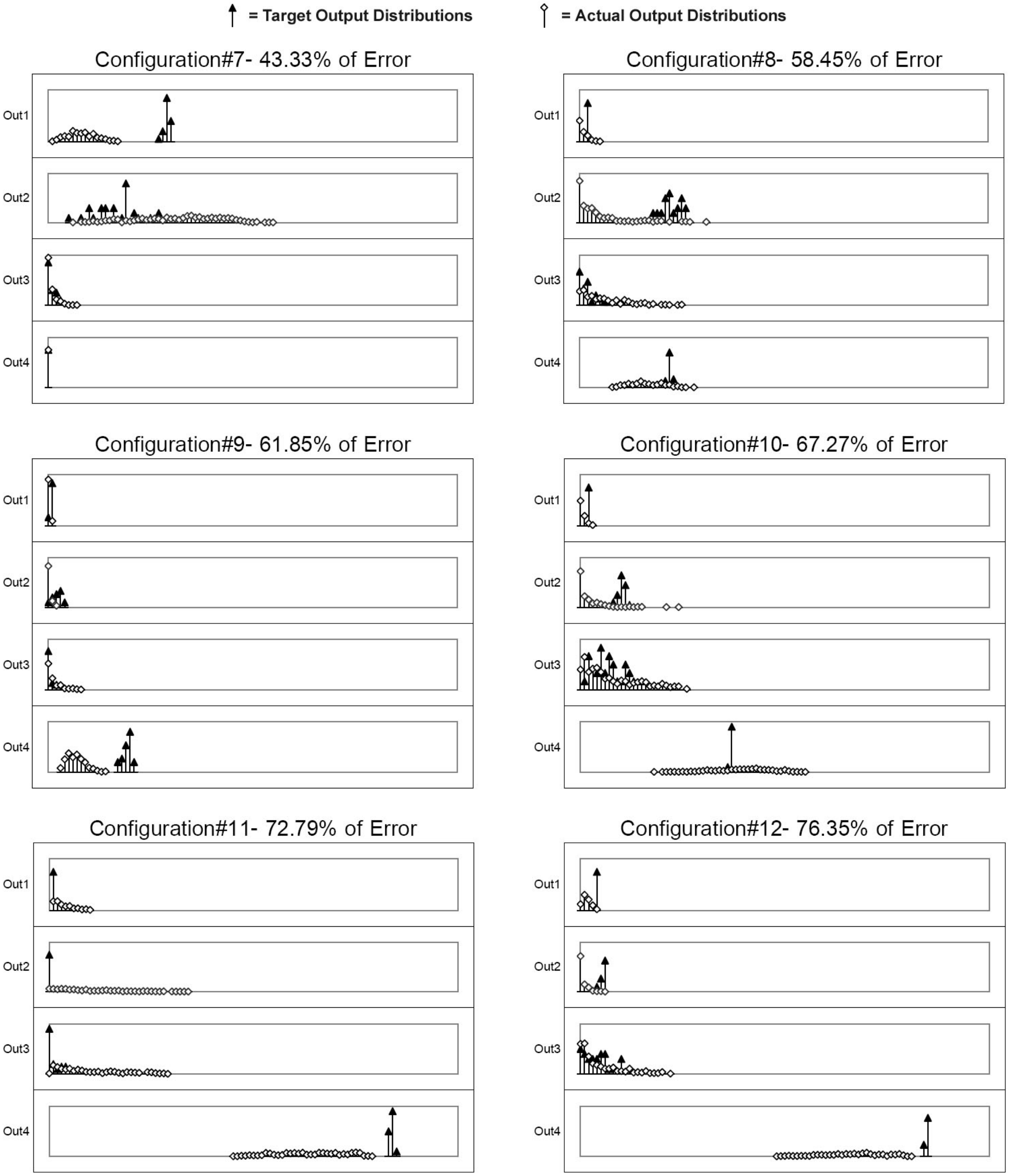


Figure A2: Six plots of the last six example configurations are shown with their percent of distribution error. Each configuration consists of four output distribution plots. Target distributions are displayed as arrow poles. Actual distributions generated by team performance model are displayed as diamond poles.

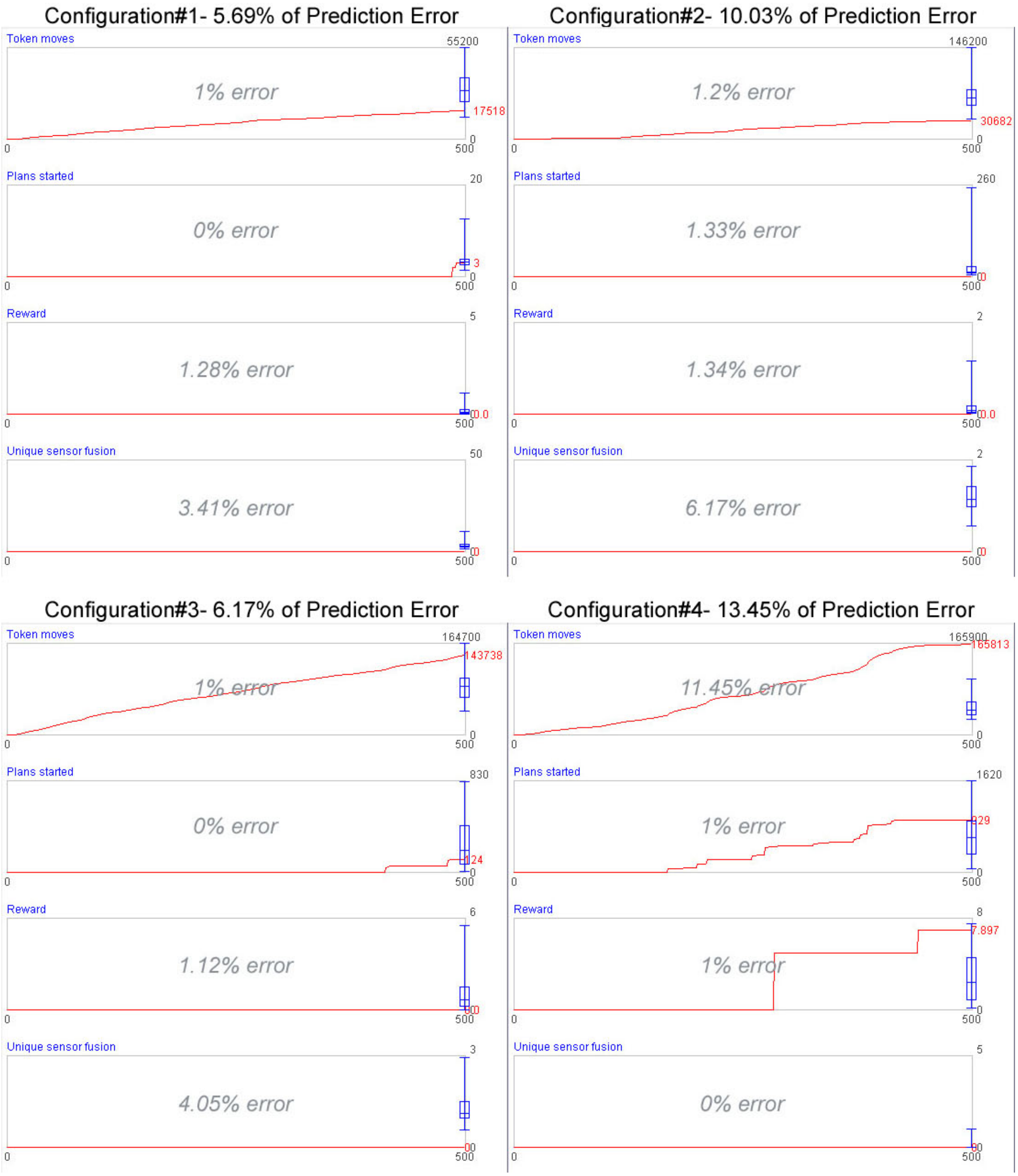
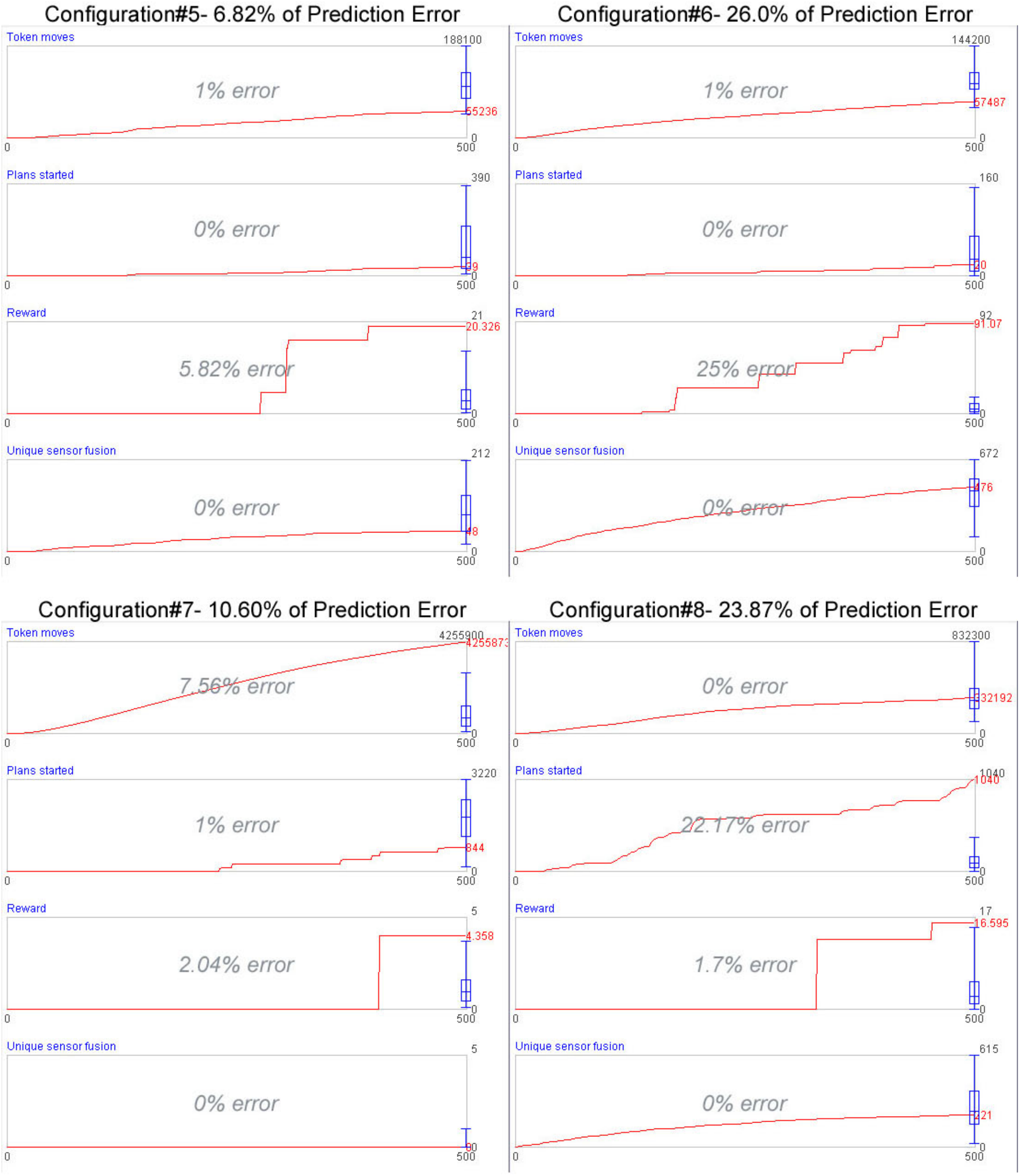


Figure A3: Examples of the first four configurations (1-4) are displayed when they are verified with team control interface.



Configuration#5- 6.82% of Prediction Error

Configuration#6- 26.0% of Prediction Error

Configuration#7- 10.60% of Prediction Error

Configuration#8- 23.87% of Prediction Error

Figure A4: Examples of the next four configurations (5-8) are displayed when they are verified with team control interface.

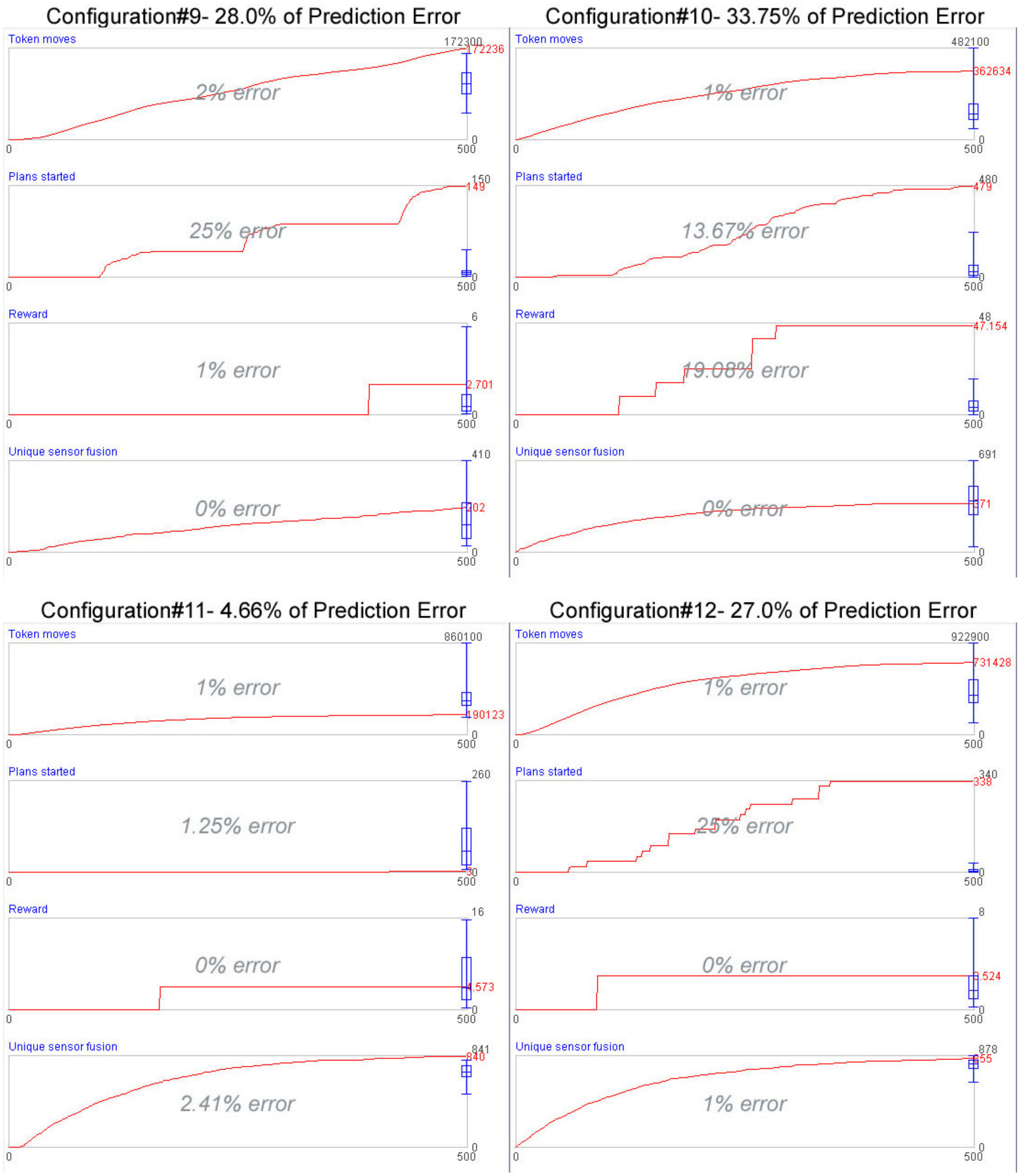


Figure A5: Examples of the last four configurations (9-12) are displayed when they are verified with team control interface.

APPENDIX B

EXAMPLES OF RECONFIGURATION VALIDATIONS

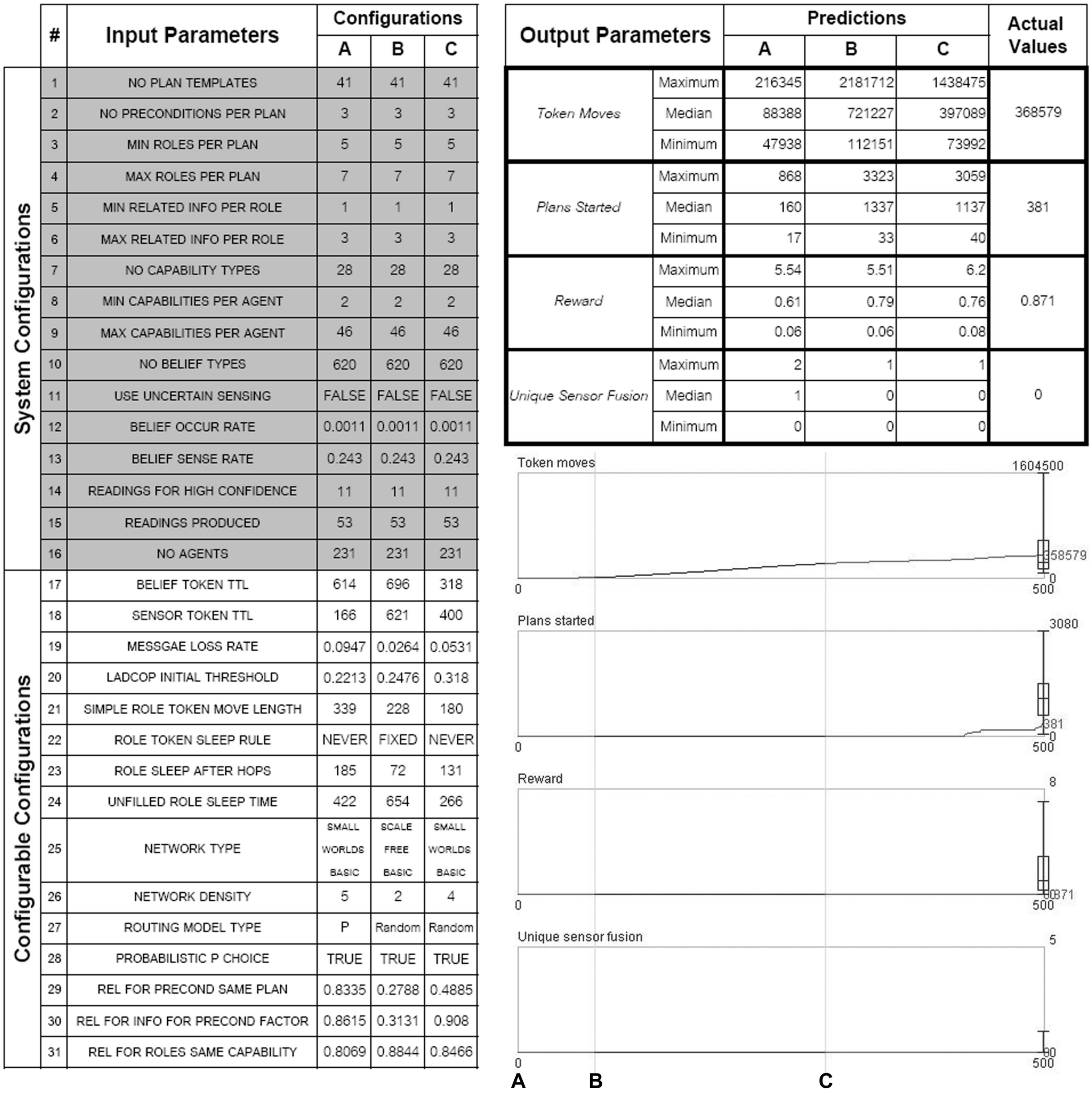
In this section, four examples of reconfigurations were demonstrated with constraints in sequence. In each example, there were two points of reconfiguration. At start, at point A, a configuration was set, then the first reconfiguration was executed at point B, and the final reconfiguration was performed at point C. Thus, there were three configurations assessed. Each point of assessments all input parameters and predicted output parameters were recorded. At the end, the actual output performance measures were recorded. In all figure, on the left, there configurations are shown with all input parameters at point A, B, and C, respectively. The first 16 input parameters are system configurations that can not be changed. The last 15 input parameters are configurable configurations that can be changed during execution. On the top right, output parameters are shown. The prediction of four output values at point A, B, and C are listed for their maximum, median, minimum. The final actual output values are shown as well. On lower right, four plots of four output performance measures are displayed. At point A, it is a start point. At point B, and C, there are lines indicating the point of reconfigurations. The maximum step of execution is 500. The scales of the plots are automatically adjusted to display all information. The maximum output value of each plot is shown on the upper right side of the plot. The box plots represent the output predictions at current step of execution.

The first example is shown in Figure B1. At point A, the start configuration was set with the third configuration in Table A1. At point B, with intention to increase the number of token moves, a new configuration was searched and reconfigured by the team control interface. Then, with intention to increase the number of plans started, the team control interface was used to find a new configuration. At point C, the new configuration was reconfigured. At the end, comparing with the original configuration, the results showed high number of token moves and plans started. In addition, other outputs were very close to predictions.

The second example is shown in Figure B2. At point A, the start configuration was set with the fifth configuration in Table A1. At point B, with intention to increase the number of token moves and plans started, a new configuration was searched and reconfigured by the team control interface. Then, with intention to decrease the number of token moves but still increase the number of plans started, the team control interface was used to find a new configuration. At point C, the new configuration was reconfigured. At the end, comparing with the original configuration, the results showed that the number of token moves did increase not much even it started to increase a lot at point B, but it slowly increased after point C. The number of plans started was relatively increased from the start. In addition, other outputs were very close to predictions.

The third example is shown in Figure B3. At point A, the start configuration was set with the sixth configuration in Table A1. At point B, with intention to increase the number of token moves, a new configuration was searched and reconfigured by the team control interface. Then, with intention to increase the number of plans stared, the team control interface was used to find a new configuration. At point C, the new configuration was reconfigured. At the end, comparing with the original configuration, the results showed the number of token moves started to increase at point B and got a high number. The number of plans started was not increased at all before point C, and started to increase after point C. In addition, other outputs were very close to predictions.

The fourth example is shown in Figure B4. At point A, the start configuration was set with the eighth configuration in Table A1. At point B, with intention to increase the number of token moves, a new configuration was searched and reconfigured by the team control interface. Then, with intention to increase the number of reward, the team control interface was used to find a new configuration. At point C, the new configuration was reconfigured. At the end, comparing with the original configuration, the results showed the number of token moves started to increase a lot at point B and got slower increase after point C as the effect of the third configuration. The final number of token moved was still high. The number of reward was not increased at start until the start of point C, and then it started to increase and got the range of predicted values. The number of plans started was not increased at all before point B, and started to increase after point B. Even though this number of plans started was too high than the predicted values, other outputs were very close to predictions.



	#	Input Parameters	Configurations		
			A	B	C
System Configurations	1	NO PLAN TEMPLATES	41	41	41
	2	NO PRECONDITIONS PER PLAN	3	3	3
	3	MIN ROLES PER PLAN	5	5	5
	4	MAX ROLES PER PLAN	7	7	7
	5	MIN RELATED INFO PER ROLE	1	1	1
	6	MAX RELATED INFO PER ROLE	3	3	3
	7	NO CAPABILITY TYPES	28	28	28
	8	MIN CAPABILITIES PER AGENT	2	2	2
	9	MAX CAPABILITIES PER AGENT	46	46	46
	10	NO BELIEF TYPES	620	620	620
	11	USE UNCERTAIN SENSING	FALSE	FALSE	FALSE
	12	BELIEF OCCUR RATE	0.0011	0.0011	0.0011
	13	BELIEF SENSE RATE	0.243	0.243	0.243
	14	READINGS FOR HIGH CONFIDENCE	11	11	11
	15	READINGS PRODUCED	53	53	53
	16	NO AGENTS	231	231	231
Configurable Configurations	17	BELIEF TOKEN TTL	614	696	318
	18	SENSOR TOKEN TTL	166	621	400
	19	MESSGAE LOSS RATE	0.0947	0.0264	0.0531
	20	LADCOP INITIAL THRESHOLD	0.2213	0.2476	0.318
	21	SIMPLE ROLE TOKEN MOVE LENGTH	339	228	180
	22	ROLE TOKEN SLEEP RULE	NEVER	FIXED	NEVER
	23	ROLE SLEEP AFTER HOPS	185	72	131
	24	UNFILLED ROLE SLEEP TIME	422	654	266
	25	NETWORK TYPE	SMALL WORLDS BASIC	SCALE FREE BASIC	SMALL WORLDS BASIC
	26	NETWORK DENSITY	5	2	4
	27	ROUTING MODEL TYPE	P	Random	Random
	28	PROBABILISTIC P CHOICE	TRUE	TRUE	TRUE
	29	REL FOR PRECOND SAME PLAN	0.8335	0.2788	0.4885
	30	REL FOR INFO FOR PRECOND FACTOR	0.8615	0.3131	0.908
	31	REL FOR ROLES SAME CAPABILITY	0.8069	0.8844	0.8466

Output Parameters		Predictions			Actual Values
		A	B	C	
Token Moves	Maximum	216345	2181712	1438475	368579
	Median	88388	721227	397089	
	Minimum	47938	112151	73992	
Plans Started	Maximum	868	3323	3059	381
	Median	160	1337	1137	
	Minimum	17	33	40	
Reward	Maximum	5.54	5.51	6.2	0.871
	Median	0.61	0.79	0.76	
	Minimum	0.06	0.06	0.08	
Unique Sensor Fusion	Maximum	2	1	1	0
	Median	1	0	0	
	Minimum	0	0	0	

Figure B1: The first example of reconfiguration. At point B, a new configuration was used to increase the number of token moves. At point C, a new configuration was used to increase the number of plans started.

	#	Input Parameters	Configurations		
			A	B	C
System Configurations	1	NO PLAN TEMPLATES	49	49	49
	2	NO PRECONDITIONS PER PLAN	3	3	3
	3	MIN ROLES PER PLAN	2	2	2
	4	MAX ROLES PER PLAN	3	3	3
	5	MIN RELATED INFO PER ROLE	8	8	8
	6	MAX RELATED INFO PER ROLE	14	14	14
	7	NO CAPABILITY TYPES	86	86	86
	8	MIN CAPABILITIES PER AGENT	3	3	3
	9	MAX CAPABILITIES PER AGENT	41	41	41
	10	NO BELIEF TYPES	54	54	54
	11	USE UNCERTAIN SENSING	TRUE	TRUE	TRUE
	12	BELIEF OCCUR RATE	0.0037	0.0037	0.0037
	13	BELIEF SENSE RATE	0.129	0.129	0.129
	14	READINGS FOR HIGH CONFIDENCE	15	15	15
	15	READINGS PRODUCED	56	56	56
	16	NO AGENTS	345	345	345
Configurable Configurations	17	BELIEF TOKEN TTL	558	373	210
	18	SENSOR TOKEN TTL	380	506	614
	19	MESSGAE LOSS RATE	0.0486	0.0273	0.0345
	20	LADCOP INITIAL THRESHOLD	0.6329	0.781	0.8849
	21	SIMPLE ROLE TOKEN MOVE LENGTH	49	25	21
	22	ROLE TOKEN SLEEP RULE	FIXED	NEVER	NEVER
	23	ROLE SLEEP AFTER HOPS	450	245	255
	24	UNFILLED ROLE SLEEP TIME	401	280	158
	25	NETWORK TYPE	COMPLETE	COMPLETE	RANDOM
	26	NETWORK DENSITY	4	5	3
	27	ROUTING MODEL TYPE	P	P	P
	28	PROBABILISTIC P CHOICE	FALSE	TRUE	Ture
	29	REL FOR PRECOND SAME PLAN	0.4571	0.3185	0.1602
	30	REL FOR INFO FOR PRECOND FACTOR	0.889	0.9428	0.9458
	31	REL FOR ROLES SAME CAPABILITY	0.1625	0.1652	0.3958

Output Parameters		Predictions			Actual Values
		A	B	C	
Token Moves	Maximum	190040	386815	277576	70692
	Median	97254	144638	126226	
	Minimum	48010	78197	74911	
Plans Started	Maximum	376	500	487	52
	Median	143	278	279	
	Minimum	8	30	52	
Reward	Maximum	15.02	17.55	18.67	15.185
	Median	3.7	5.08	6.47	
	Minimum	0.3	0.27	0.59	
Unique Sensor Fusion	Maximum	227	214	227	46
	Median	72	73	70	
	Minimum	6	8	11	

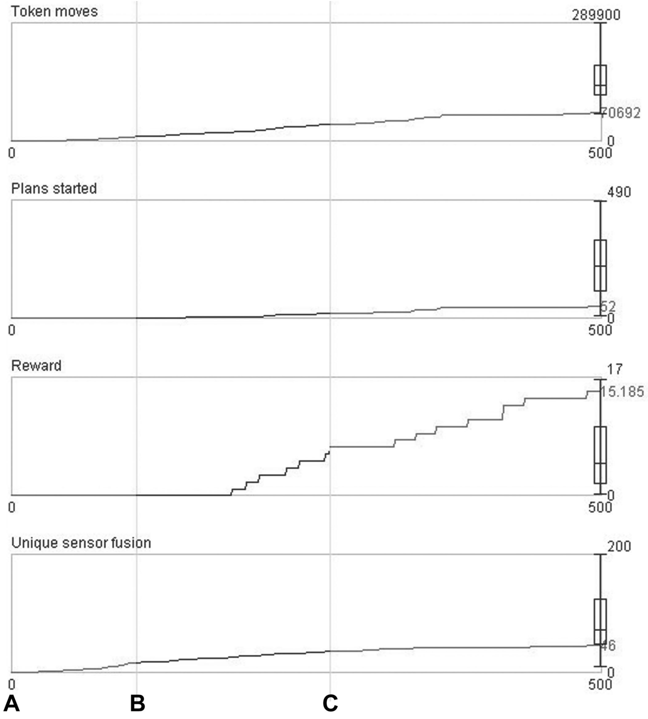


Figure B2: The second example of reconfiguration. At point B, a new configuration was used to increase the number of token moves and plans started. At point C, a new configuration was used to decrease the number of token moves and still increase the number of plans started.

	#	Input Parameters	Configurations		
			A	B	C
System Configurations	1	NO PLAN TEMPLATES	38	38	38
	2	NO PRECONDITIONS PER PLAN	3	3	3
	3	MIN ROLES PER PLAN	3	3	3
	4	MAX ROLES PER PLAN	13	13	13
	5	MIN RELATED INFO PER ROLE	6	6	6
	6	MAX RELATED INFO PER ROLE	18	18	18
	7	NO CAPABILITY TYPES	20	20	20
	8	MIN CAPABILITIES PER AGENT	2	2	2
	9	MAX CAPABILITIES PER AGENT	42	42	42
	10	NO BELIEF TYPES	642	642	642
	11	USE UNCERTAIN SENSING	TRUE	TRUE	TRUE
	12	BELIEF OCCUR RATE	0.0027	0.0027	0.0027
	13	BELIEF SENSE RATE	0.1789	0.1789	0.1789
	14	READINGS FOR HIGH CONFIDENCE	12	12	12
	15	READINGS PRODUCED	13	13	13
	16	NO AGENTS	60	60	60
Configurable Configurations	17	BELIEF TOKEN TTL	145	153	78
	18	SENSOR TOKEN TTL	738	750	538
	19	MESSGAE LOSS RATE	0.0672	0.0359	0.0777
	20	LADCOP INITIAL THRESHOLD	0.258	0.4799	0.5692
	21	SIMPLE ROLE TOKEN MOVE LENGTH	108	287	83
	22	ROLE TOKEN SLEEP RULE	FIXED	FIXED	NEVER
	23	ROLE SLEEP AFTER HOPS	750	814	779
	24	UNFILLED ROLE SLEEP TIME	101	422	107
	25	NETWORK TYPE	RANDOM CAP MORE	SCALE FREE BASIC	RANDOM CAP MORE
	26	NETWORK DENSITY	5	5	3
	27	ROUTING MODEL TYPE	P	P	P
	28	PROBABILISTIC P CHOICE	TRUE	FALSE	TRUE
	29	REL FOR PRECOND SAME PLAN	0.399	0.5757	0.2042
	30	REL FOR INFO FOR PRECOND FACTOR	0.3011	0.6323	0.5744
	31	REL FOR ROLES SAME CAPABILITY	0.442	0.5992	0.5202

Output Parameters		Predictions			Actual Values
		A	B	C	
Token Moves	Maximum	143546	251783	147855	62168
	Median	84340	157586	83693	
	Minimum	43112	64091	45052	
Plans Started	Maximum	138	134	147	11
	Median	24	19	28	
	Minimum	1	1	4	
Reward	Maximum	16.46	15.39	16.26	7933
	Median	4.75	3.24	5.05	
	Minimum	0.17	0.18	0.2	
Unique Sensor Fusion	Maximum	660	775	607	465
	Median	405	573	400	
	Minimum	109	120	85	

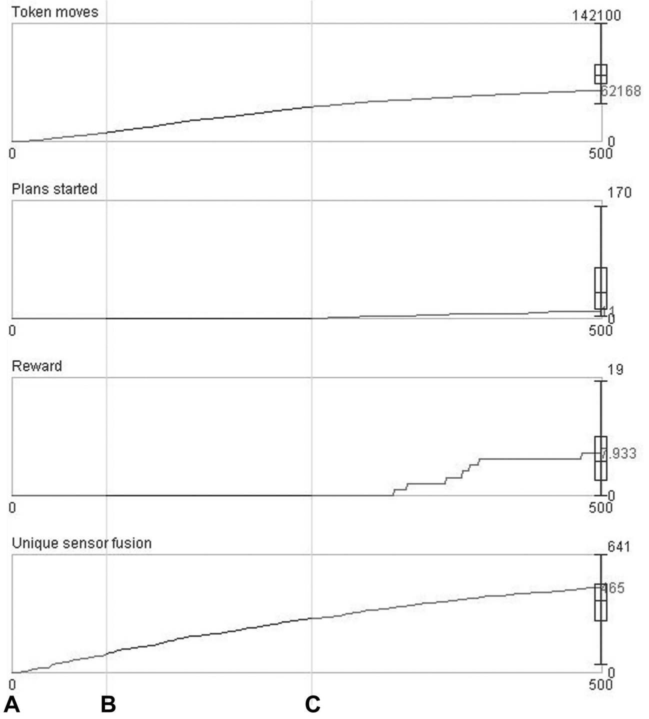
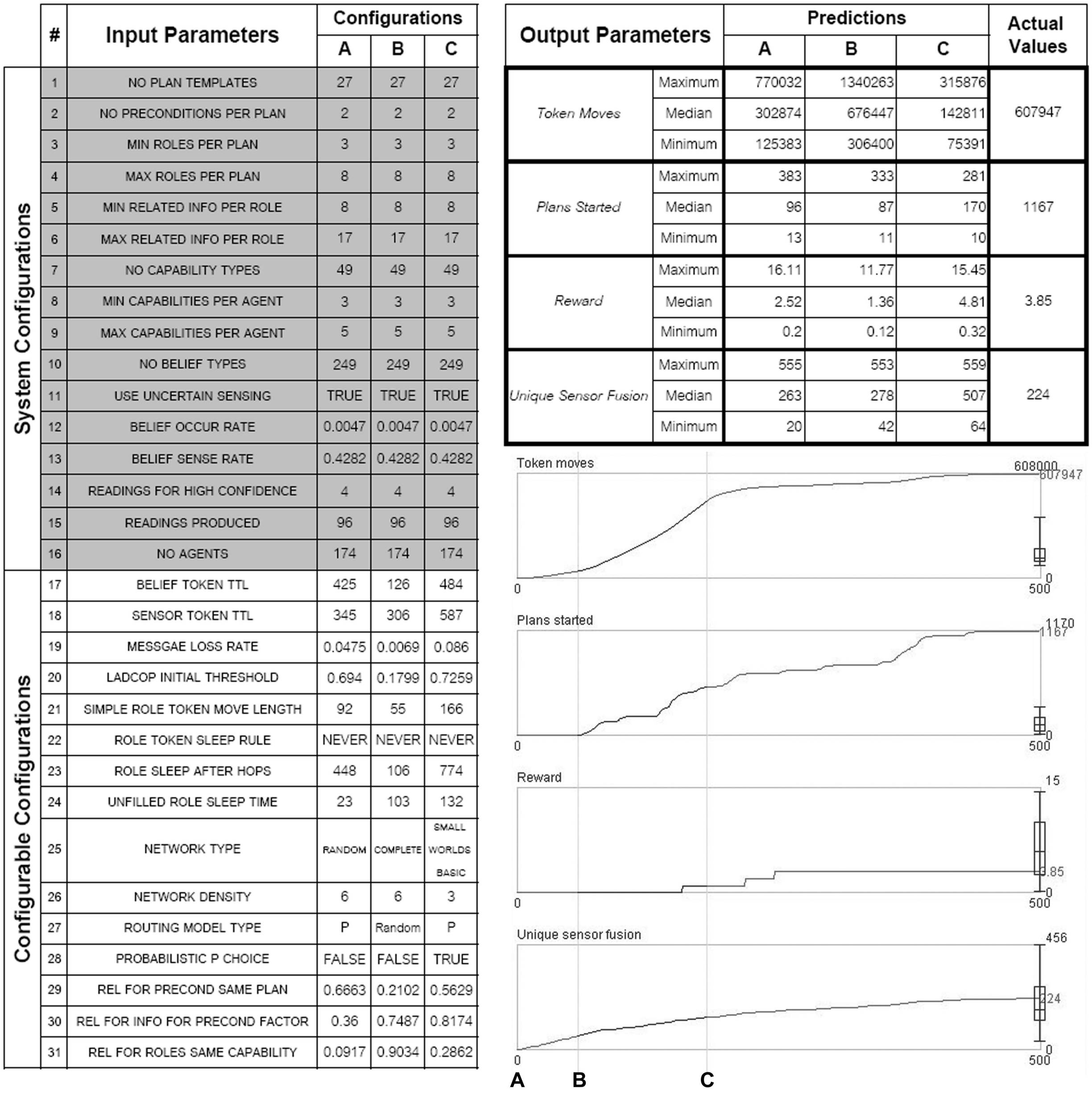


Figure B3: The third example of reconfiguration. At point B, a new configuration was used to increase the number of token moves. At point C, a new configuration was used to increase the number of plans started.



	#	Input Parameters	Configurations		
			A	B	C
System Configurations	1	NO PLAN TEMPLATES	27	27	27
	2	NO PRECONDITIONS PER PLAN	2	2	2
	3	MIN ROLES PER PLAN	3	3	3
	4	MAX ROLES PER PLAN	8	8	8
	5	MIN RELATED INFO PER ROLE	8	8	8
	6	MAX RELATED INFO PER ROLE	17	17	17
	7	NO CAPABILITY TYPES	49	49	49
	8	MIN CAPABILITIES PER AGENT	3	3	3
	9	MAX CAPABILITIES PER AGENT	5	5	5
	10	NO BELIEF TYPES	249	249	249
	11	USE UNCERTAIN SENSING	TRUE	TRUE	TRUE
	12	BELIEF OCCUR RATE	0.0047	0.0047	0.0047
	13	BELIEF SENSE RATE	0.4282	0.4282	0.4282
	14	READINGS FOR HIGH CONFIDENCE	4	4	4
	15	READINGS PRODUCED	96	96	96
	16	NO AGENTS	174	174	174
Configurable Configurations	17	BELIEF TOKEN TTL	425	126	484
	18	SENSOR TOKEN TTL	345	306	587
	19	MESSGAE LOSS RATE	0.0475	0.0069	0.086
	20	LADCOP INITIAL THRESHOLD	0.694	0.1799	0.7259
	21	SIMPLE ROLE TOKEN MOVE LENGTH	92	55	166
	22	ROLE TOKEN SLEEP RULE	NEVER	NEVER	NEVER
	23	ROLE SLEEP AFTER HOPS	448	106	774
	24	UNFILLED ROLE SLEEP TIME	23	103	132
	25	NETWORK TYPE	RANDOM	COMPLETE	SMALL WORLDS BASIC
	26	NETWORK DENSITY	6	6	3
	27	ROUTING MODEL TYPE	P	Random	P
	28	PROBABILISTIC P CHOICE	FALSE	FALSE	TRUE
	29	REL FOR PRECOND SAME PLAN	0.6663	0.2102	0.5629
	30	REL FOR INFO FOR PRECOND FACTOR	0.36	0.7487	0.8174
	31	REL FOR ROLES SAME CAPABILITY	0.0917	0.9034	0.2862

Output Parameters		Predictions			Actual Values
		A	B	C	
Token Moves	Maximum	770032	1340263	315876	607947
	Median	302874	676447	142811	
	Minimum	125383	306400	75391	
Plans Started	Maximum	383	333	281	1167
	Median	96	87	170	
	Minimum	13	11	10	
Reward	Maximum	16.11	11.77	15.45	3.85
	Median	2.52	1.36	4.81	
	Minimum	0.2	0.12	0.32	
Unique Sensor Fusion	Maximum	555	553	559	224
	Median	263	278	507	
	Minimum	20	42	64	

Figure B4: The fourth example of reconfiguration. At point B, a new configuration was used to increase the number of token moves. At point C, a new configuration was used to increase the number of reward.

BIBLIOGRAPHY

[1] A. Arkin, J. Ross, H. McAdams, Stochastic kinetic analysis of developmental pathway bifurcation in phage-infected escherichia coli cells, Genetics 149 (1998) 1633–1648.

[2] C. Atkinson, Meta-modeling for distributed object environments, in: Proceedings of the First International Enterprise Distributed Object Computing Workshop, 1997.

[3] B. Ayyub, G. Klir, Uncertainty Modeling And Analysis in Engineering And the Sciences, 1st ed., Chapman & Hall-CRC, 2006.

[4] N. Baba, Y. Yanjun, Utilization of neural networks & genetic algorithms for constructing a reliable decision support system which deals nikkei-225, in: Proceedings of the 41st SICE Annual Conference, vol. 3, 2002.

[5] D. Barrios, D. Manrique, J. Porras, J. Rios, Optimum binary codification for genetic design of artificial neural networks, in: Proceedings. Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies, vol. 2, 2000.

[6] C. Belcastro, Parametric uncertainty modeling: an overview, in: Proceedings of the 1998 American Control Conference, vol. 2, 1998.

[7] B. Brown, H. Card, Stochastic neural computation. i. computational elements, IEEE Transactions on Computers 50 (9) (2001) 891–905.

[8] B. Brown, H. Card, Stochastic neural computation. ii. soft competitive learning, IEEE Transactions on Computers 50 (9) (2001) 906–920.

[9] B. Bryant, R. Miikkulainen, Evolving stochastic controller networks for intelligent game agents, in: IEEE Congress on Evolutionary Computation- CEC 2006, 2006.

[10] H. H. Bui, S. Venkatesh, D. Kieronska, A framework for coordination and learning among team members, in: Proceedings of the Third Australian Workshop on Distributed AI, 1997.

[11] L. Bull, On coevolutionary genetic algorithms, Soft Computing - A Fusion of Foundations, Methodologies and Applications 5 (3) (2001) 201–207.

[12] M. Caeiro-Rodriguez, M. Llamas-Nistal, L. Anido-Rifon, Meta-modeling for computer-supported group-based learning design, in: Proceedings of the 5th International Conference on Intelligent Systems Design and Applications, 2005.

[13] N. Chaiyaratana, A. Zalzala, Hybridisation of neural networks and genetic algorithms for time-optimal control, in: Proceedings of the 1999 Congress on Evolutionary Computation, CEC99, vol. 1, 1999.

[14] N. Chaiyaratana, A. Zalzala, Hybridisation of neural networks and a genetic algorithm for friction compensation, in: Proceedings of the 2000 Congress on Evolutionary Computation, vol. 1, 2000.

[15] L. Chamberlain, J. Tang, M. Watugala, J. A. Adams, M. Babish, A behavioral architecture for strategy execution in the roboflag game, in: Proceedings of the American Control Conference, 2003.

[16] C. Chatfield, Model uncertainty, data mining and statistical inference, Journal of the Royal Statistical Society, Series A (Statistics in Society) 158 (3) (1995) 419–466.

[17] H. Chen, X. Wang, Cooperative coevolutionary approach and its promising applications in power system, in: Sixth International Conference on Advances in Power System Control, Operation and Management, ASDCOM 2003, vol. 2, 2003.

[18] P. Chongstitvatana, Using perturbation to improve robustness of solutions generated by genetic programming for robot learning, Journal of Circuits, Systems and Computer 9 (1 and 2) (1999) 133–143.

[19] C. Clifton, Using sample size to limit exposure to data mining, Journal of Computer Security 8 (4) (2000) 281–307.

[20] L. S. Coelho, A. R. Coelho, R. Krohling, Parameters tuning of multivariable controllers based on mimetic algorithm: fundamentals and application, in: Proceedings of the 2002 IEEE International Symposium on Intelligent Control, 2002.

[21] M. Cohen, D. Hudson, Meta neural networks as intelligent agents for diagnosis, in: Proceedings of the 2002 International Joint Conference on Neural Networks, vol. 1, 2002.

[22] J. W. Crandall, M. A. Goodrich, D. R. O. Jr, C. W. Nielsen, Validating human-robot interaction schemes in multi-tasking environments, IEEE Transactions on Systems, Man and Cybernetics, Part A 35 (4) (2005) 438449.

[23] F. Cribari-Neto, K. L. P. Vasconcellos, Nearly unbiased maximum likelihood estimation for the beta distribution, Journal of Statistical Computation and Simulation 72 (2) (2002) 107–118.

[24] K. Dixon, J. Dolan, W. Huang, C. Paredis, P. Khosla, Rave: A real and virtual environment for multiple mobile robot systems, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 1999.

[25] C. Domingo, R. Gavalda, O. Watanabe, Adaptive sampling methods for scaling up knowledge discovery algorithms, Data Mining and Knowledge Discovery 6 (2002) 131–152.

[26] N. Doulamis, A. Doulamis, Optimal distribution transformers assembly using an adaptable neural network-genetic algorithm scheme, in: IEEE International Conference on Systems, Man and Cybernetics, vol. 5, 2002.

[27] P. Eggenberger, A. Ishiguro, S. Tokura, T. Kondo, Y. Uchikawa, Toward seamless transfer from simulated to real worlds: A dynamically-rearranging neural network approach, in: Proceeding of 1999 the Eighth European Workshop in Learning Robots (EWLR-8), 1999.

[28] Y. Endo, D. C. MacKenzie, R. C. Arkin, Usability evaluation of high-level user assistance for robot mission specification, IEEE Transactions on Systems, Man and Cybernetics, Part C 34 (2004) 168–180.

[29] R. Falcone, C. Castelfranchi, Tuning the collaboration level with autonomous agents: A principled theory, in: The IJCAI-01 Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents, 2001.

[30] J. Ferber, G. Uma, A meta-model for the analysis and design of organizations in multiagent systems, in: Third International Conference on Multi-agent systems, Paris, 1998.

[31] T. Fong, S. Grange, C. Thorpe, C. Baur, Multi-robot remote driving with collaborative control, in: IEEE Intl. Workshop on Robot-Human Interactive Communication, 2001.

[32] K. Fregene, R. Madhavan, L. E. Parker, Incremental multiagent robotic mapping of outdoor terrains, in: In Proceedings of IEEE International Conference on Robotics and Automation, 2002.

[33] D. Fu, R. Houlette, R. Jensen, O. Bascara, A visual, object-oriented approach to simulation behavior authoring, in: The Interservice/Industry Training, Simulation and Education Conference, 2003.

[34] Q. Gao, K. Qi, Y. Lei, Z. He, An improved genetic algorithm and its application in artificial neural network training, in: Fifth International Conference on Information, Communications and Signal Processing, 2005.

[35] B. Gerkey, M. J. Mataric, A formal analysis and taxonomy of task allocation in multi-robot systems, International Journal of Robotic Research 23 (9) (2004) 939–954.

[36] D. Glade, Unmanned aerial vehicles: Implications for military operations, Tech. Rep. Occasional Paper No. 16, Center for Strategy and Technology Air War College (2000).

[37] D. Goldberg, Genetic Algorithms in search, optimizaton, and machine learning, MA: Addison-Wesley, 1989.

[38] J. Gonzalez-Seco, A genetic algorithm as the learning procedure for neural networks, in: International Joint Conference on Neural Networks-IJCNN, vol. 1, 1992.

[39] R. Grabowski, L. E. Navarro-Serment, C. J. Pareidis, P. Khosla, Heterogeneous teams of modular robots for mapping and exploration, Autonomous Robots Special Issue on Heterogeneous Multi-Robot Systems 8 (2000) 293–308.

[40] Z. Guo, R. Uhrig, Using genetic algorithms to select inputs for neural networks, in: International Workshop on Combinations of Genetic Algorithms and Neural Networks, COGANN-92, 1992.

[41] H. H, T. Horiuchi, O. Katai, T. Kaneko, T. Konishi, M. Baba, Coevolutionary ga with schema extraction by machine learningtechniques and its application to knapsack problems, in: Proceedings of the 2001 Congress on Evolutionary Computation, vol. 2, 2001.

[42] K. Hintz, J. Spofford, Evolving a neural network, in: Proceedings of the 5th IEEE International Symposium on Intelligent Control, 1990.

[43] J. Holland, Adaptation in natural and artificial systems, Ann Arbor, MI: University of Michigan Press, 1975.

[44] Y.-C. Huang, Condition assessment of power transformers using genetic-based neural networks, in: IEE Proceedings-Science, Measurement and Technology, vol. 150, 2003.

[45] R. P. J. A. Adams, Human management of a hierarchical control system for multiple mobile robots, in: Proceedings of the 1994 IEEE International Conference on Systems, Man and Cybernetics, 1994.

[46] C. Janer, J. Quero, L. Franquelo, Fully parallel summation in a new stochastic neural network architecture, in: IEEE International Conference on Neural Networks, vol. 3, 1993.

[47] H. Jones, P. Hinds, Extreme work groups: Using swat teams as a model for coordinating distributed robots, in: In Proceedings of the ACM 2002 Conference on Computer Supported Cooperative Work (CSCW 2002), 2002.

[48] H. D. Jong, Modeling and simulation of genetic regulatory systems: A literature review, Journal of Computational Biology 9 (1) (2002) 67–103.

[49] H. Juille, J. B. Pollack, Dynamics of co-evolutionary learning, in: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, MIT Press, 1996.

[50] L. Kent, A neurogenetic model of muscle emg to torque, in: International Joint Conference on Neural Networks, IJCNN '99, vol. 5, 1999.

[51] F. N. Kerlinger, H. B. Lee, Foundations of Behavioral Research, 4th ed., Harcourt College Publishers, 2000.

[52] J. M. Kim, S. I. Moon, J. Lee, A new optimal avr parameter tuning method using on-line performance indices of frequency-domain, IEEE Power Engineering Society Summer Meeting 3 (2001) 1554–1559.

[53] Y. C. Kim, M. Shanblatt, Random noise effects in pulse-mode digital multilayer neural networks, IEEE Transactions on Neural Networks 6 (1) (1995) 220–229.

[54] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjoh, S. Shimada, Robocup rescue: Searh and rescue in large-scale disasters as a domain for autonomous agents research, in: Proc. 1999 IEEE Intl. Conf. on Systems, Man and Cybernetics, vol. VI, Tokyo, 1999.

[55] J. Kivinen, H. Mannila, The power of sampling in knowledge discovery, in: In Proceedings of Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium, Principles of Database System, ACM Press, 1994.

[56] Y. Kondo, Y. Sawada, Functional abilities of a stochastic logic neural network, IEEE Transactions on Neural Networks 3 (3) (1992) 434–443.

[57] D. Kortenkamp, D. Keirn-Schreckenghost, R. P. Bonasso, Adjustable control autonomy for manned space flight, in: IEEE Aerospace Conference, 2000.

[58] T. Krink, R. Ursem, Parameter control using the agent based patchwork model, in: Proceedings of the 2000 Congress on Evolutionary Computation, vol. 1, 2000.

[59] J. Krynsky, E. Janecek, P. Janecek, A new stochastic model of the electric distribution network, in: International Conference on Probabilistic Methods Applied to Power Systems, 2004.

[60] H. Kwong, C. Jacob, Evolutionary exploration of dynamic swarm behavior, in: The 2003 Congress on Evolutionary Computation, CEC '03, vol. 1, 2003.

[61] H. Lam, F. Leung, Digit and command interpretation for electronic book using neural network and genetic algorithm, IEEE Transactions on Systems, Man and Cybernetics, Part B 34 (2004) 2273 – 2283.

[62] M.-L. Lan, S.-T. Pan, C.-C. Lai, Using genetic algorithm to improve the performance of speech recognition based on artificial neural network, in: First International Conference on Innovative Computing, Information and Control, ICICIC '06, vol. 2, 2006.

[63] S. D. Lee, D. Cheung, B. Kao, Is sampling useful in data mining ? a case in the maintenance of discovered association rules, Data Mining and Knowledge Discovery 2 (3) (1998) 233–262.

[64] M. Lewis, J. Polvichai, K. Sycara, P. Scerri, In N. Cooke (Ed.), The Human Factors of Remotely Piloted Vehicles, chap. Scaling-up Human Control for Large UAV Teams, New York: Elsevier, 2006, pp. 237–250.

[65] M. Lewis, J. Wang, S. Hughes, Usarsim : Simulation for the study of human-robot interaction, Journal of Cognitive Engineering and Decision Making 1 (1) (2007) 98–120.

[66] J. D. Lohn, W. F. Kraus, G. L. Haith, Comparing a coevolutionary genetic algorithm for multiobjective optimization, in: CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress, IEEE Computer Society, Washington, DC, USA, 2002.

[67] M. Lu, S. M. AbouRizk, U. H. Hermann, Sensitivity analysis of neural networks in spool fabrication productivity studies, Journal of Computing in Civil Engineering 15 (4) (2001) 299–308.

[68] R. Madhavan, K. Fregene, L. E. Parker, Distributed cooperative outdoor multirobot localization and map-ping, Autonomous Robots, Special Issue on Analysis and Experiments in Distributed Multi-Robot Systems 17 (1) (2004) 2339.

[69] G. Meghabghab, G. Nasr, Iterative rbf neural networks as metamodels of stochastic simulations, in: Proceedings of the Second International Conference on Intelligent Processing and Manufacturing of Materials, vol. 2, 1999.

[70] C. Miller, R. Parasuraman, Designing for flexible interaction between humans and automation: Delegation interfaces for supervisory control, Human Factors.

[71] T. M. Mitchell, Machine Learning, chap. Artificial Neural Networks, McGraw-Hill Higher Education, 1997, pp. 81–127.

[72] T. M. Mitchell, Machine Learning, chap. Computational Learning Theory, McGraw-Hill Higher Education, 1997, pp. 201–229.

[73] S. Nadarajah, A. K. Gupta, Characterizations of the beta distribution, Communications in Statistics - Theory and Methods 33 (12) (2004) 2941 – 2957.

[74] J. Neves, P. Cortez, An artificial neural network-genetic based approach for time series forecasting, in: the IVth Proceedings Brazilian Symposium on Neural Networks, 1997.

[75] T. Nobori, N. Matsui, Stochastic resonance neural network and its performance, in: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks - IJCNN 2000, vol. 2, 2000.

[76] R. Parasuraman, S. Galster, P. Squire, H. Furukawa, C. Miller, A flexible delegation-type interface enhances system performance in human supervision of multiple robots: Empirical studies with roboflag, IEEE Transactions on Systems, Man, and CyberneticsPart A: Systems and Humans 35 (4) (2005) 481–493.

[77] L. E. Parker, Lifelong adaptation in heterogeneous multi-robot teams: Response to continual variation in individual robot performance, Autonomous Robots 8 (2000) 239–267.

[78] L. E. Parker, Distributed algorithms for multi-robot observation of multiple moving targets, Autonomous Robots 12 (3).

[79] L. E. Parker, Y. Guo, D. Jung, Cooperative robot teams applied to the site preparation task, in: Proceedings of 10th International Conference on Advanced Robotics, 2001.

[80] K. Patan, T. Parisini, Stochastic learning methods for dynamic neural networks: simulated and real-data comparisons, in: Proceedings of the 2002American Control Conference, vol. 4, 2002.

[81] V. Pavlovic, D. Schonfeld, G. Friedman, Enhancement of hopfield neural networks using stochastic noise processes, in: Proceedings of the 2001 IEEE Signal Processing Society Workshop-Neural Networks for Signal Processing XI, 2001.

[82] V. Pavlovic, D. Schonfeld, G. Friedman, Stochastic noise process enhancement of hopfield neural networks, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 52 (4) (2005) 213–217.

[83] L. I. Perlovsky, Neural Networks and Intellect: Using Model-Based Concepts, Oxford University Press, 2001.

[84] J. Polvichai, P. Khosla, An evolutionary behavior programming system with dynamic networks for mobile robots in dynamic environments, in: Proceedings of 2002 IEEE/RSJ International Conference on Intelligent Robots and System, vol. 1, 2002.

[85] J. Polvichai, P. Khosla, Applying dynamic networks and staged evolution for soccer robots, in: Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and System, vol. 3, 2003.

[86] J. Polvichai, M. Lewis, P. Scerri, K. Sycara, Using a dynamic neural network to model team performance for coordination algorithm configuration and reconfiguration of large multi-agent teams, in: Intelligent Engineering Systems Through Artificial Neural Networks, Smart Engineering System Design, ASME Press Series, vol. 16, 2006.

[87] M. A. Potter, K. A. D. Jong, Cooperative coevolution: An architecture for evolving coadapted subcomponents, Evolutionary Computation 8 (1) (2000) 1–29.

[88] R. Pratap, S. Sarkar, S. Pinel, J. Laskar, G. May, Modeling and optimization of multilayer ltcc inductors for rf wireless applications using neural network and genetic algorithms, in: Proceedings of the 54th Electronic Components and Technology Conference, vol. 1, 2004.

[89] J. F. Pujol, R. Poli, Evolving neural networks using a dual representation with a combined crossover operator, in: The IEEE International Conference on Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence, 1998.

[90] D. V. Pynadath, M. Tambe, An automated teamwork infrastructure for heterogeneous software agents and humans, Journal of Autonomous Agents and Multi-Agent Systems, Special Issue on Infrastructure and Requirements for Building Research Grade Multi-Agent Systems.

[91] J. Quinlan, C4.5: Programs for machine learning, Morgan Kaufmann, 1993.

[92] I. Ramos, M. Goldbarg, E. Goldbarg, A. Neto, Logistic regression for parameter tuning on an evolutionary algorithm, The 2005 IEEE Congress on Evolutionary Computation 2 (2005) 1061–1068.

[93] P. Riley, M. Veloso, An overview of coaching with limitations, in: Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems, 2003.

[94] C. Rolland, S. Nurcan, G. Grosz, A unified framework for modeling cooperative design processes and cooperative business processes, in: Proceedings of the Thirty-First Hawaii International Conference on System Sciences, vol. 5, 1998.

[95] C. D. Rosin, R. K. Belew, Methods for competitive co-evolution: Finding opponents worth beating, in: Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[96] M. Roumbakis, B. Mutnury, S. Ulrich, J. Ratcliffe, D. de Araujo, M. Cases, Neurogenetic models in modeling nonlinear digital i/o buffer circuits, in: Proceedings. 55th Electronic Components and Technology Conference, vol. 2, 2005.

[97] D. E. Rumelhart, R. Durbin, R. Golden, Y. Chauvin, Backpropagation: theory, architectures, and applications, chap. Backpropagation: the basic theory, Lawrence Erlbaum Associates, 1995, pp. 1–34.

[98] J. Sandeep, P. Pei-Yuan, A. Tzes, F. Khorrami, Neural network designs with genetic learning for control of a single link flexible manipulator, in: American Control Conference, vol. 3, 1994.

[99] Y. Sasaki, H. de Garis, P. Box, Genetic neural networks for image classification, in: Proceedings. 2003 IEEE International Geoscience and Remote Sensing Symposium, IGARSS '03, vol. 6, 2003.

[100] Y. Sato, S. Nagaya, Evolutionary algorithms that generate recurrent neural networks for learning chaos dynamics, in: Proceedings of IEEE International Conference on Evolutionary Computation, 1996.

[101] R. Scalzo, H. Roth, A meta-model for fault management, in: Proceedings of WORDS 94., First Workshop on Object-Oriented Real-Time Dependable Systems, 1994.

[102] P. Scerri, S. Owens, R. Glinton, B. Yu, K. Sycara, Synergistic integration of agent technologies for military simulation, in: Joint LSMAS/MMAS Workshop at AAMAS'06, 2006.

[103] P. Scerri, K. Sycara, M. Tambe, Adjustable autonomy in the context of coordination, in: AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit, 2004, invited Paper.

[104] P. Scerri, R. Vincent, R. Mailler, Challenges of Large Scale Coordination, chap. Comparing Three Approaches to Large Scale Coordination, Springer, 2005.

[105] P. Scerri, Y. Xu, E. Liao, J. Lai, K. Sycara, Scaling teamwork to very large teams, in: Proceedings of AAMAS'04, 2004.

[106] N. Schurr, P. Scerri, M. Tambe, Impact of human advice on agent teams: A preliminary report, in: In AAMAS'03 Workshop on Humans and Multiagent Systems, 2003.

[107] F. Seredynski, A. Y. Zomaya, Parallel and distributed computing with coevolutionary algorithms, in: 16th Annual Int. Parallel and Distributed Processing Symposium (IPDPS), Workshop 14: Workshop on Bio-Inspired Solutions to Parallel Processing Problems (BIOSP), IEEE, 2002.

[108] I. Shmulevich, E. Dougherty, S. Kim, W. Zhang, Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks, Bioinformatics 18 (2002) 261–274.

[109] A. Siddiqi, S. Lucas, A comparison of matrix rewriting versus direct encoding for evolving neural networks, in: The IEEE International Conference on Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence, 1998.

[110] P. Smolen, D. Baxter, J. Byrne, Mathematical modeling of gene networks, Neuron 26 (2000) 567–580.

[111] J. C. Spall, Multivariate stochastic approximation using a simultaneousperturbation gradient approximation, IEEE Transactions on Automatic Control 37 (1992) 332–341.

[112] A. Srivastava, S. Srivastava, K. Shukla, Genetic evolution of neural network based on a new three-parents crossover operator, in: Proceedings of IEEE International Conference on Industrial Technology, vol. 2, 2000.

[113] P. Stone, M. Sridharan, D. Stronger, G. Kuhlmann, N. Kohl, P. Fidelman, N. Jong, From pixels to multi-robot decision-making: A study in uncertainty, Robotics and Autonomous Systems. Special issue on Planning Under Uncertainty in Robotics 54 (11) (2006) 933–943.

[114] X. Sun, C. Ren, Y. Wu, L. Ning, J. Wang, The design and application of neural network controller based on genetic algorithms, in: International Conference on Machine Learning and Cybernetics, vol. 3, 2003.

[115] J. Tao, P. Qingle, L. Xinyun, Study of traffic flow forecasting based on genetic neural network, in: Sixth International Conference on Intelligent Systems Design and Applications, ISDA '06, vol. 1, 2006.

[116] T. Tian, K. Burrage, Stochastic neural network models for gene regulatory networks, in: The 2003 Congress on Evolutionary Computation - CEC '03, vol. 1, 2003.

[117] H. Toivonen, Sampling large databases for association rules, in: T. Vijayaraman, A. P. Buchmann, C. Mohan, N. L. Sarda (eds.), VLDB'96, Proceedings of 22nd International Conference on Very Large Data Bases, Morgan Kaufmann, Seattle, 1996.

[118] J.-T. Tsai, J.-H. Chou, T.-K. Liu, Tuning the structure and parameters of a neural network by using hybrid taguchi-genetic algorithm, in: IEEE Transactions on Neural Networks, vol. 17, 2006.

[119] C. Turchetti, M. Conti, P. Crippa, S. Orcioni, On the approximation of stochastic processes by approximate identity neural networks, IEEE Transactions on Neural Networks 9 (6) (1998) 1069–1085.

[120] M. Vai, S. Prasad, Applications of neural networks optimized by the genetic algorithm to microwave systems, in: IEEE Antennas and Propagation Society International Symposium, vol. 4, 1999.

[121] H. Vangheluwe, J. D. Lara, Meta-models are models too, in: Proceedings of the Winter Simulation Conference, vol. 1, 2002.

[122] D.-S. Wang, X.-H. Xu, Genetic neural network and application in welding robot error compensation, in: Proceedings of 2005 International Conference on Machine Learning and Cybernetics, vol. 7, 2005.

[123] X. Wang, Y. Zhang, H. Liu, H. Huang, Nonlinear modeling of switched reluctance motor based on combination of neural network and genetic algorithm, in: Proceedings of the Eighth International Conference on Electrical Machines and Systems, ICEMS 2005, vol. 1, 2005.

[124] J. Werfel, M. Mitchell, J. Crutchfield, Resource sharing and coevolution in evolving cellular automata, IEEE Transactions on Evolutionary Computation 4 (4) (2000) 388–393.

[125] M. Yakout, A. Abdelfattah, A. Elbazz, Analysis of circuits containing nonlinear elements using neural networks and genetic algorithm, in: Seventeenth National Radio Science Conference, 17th NRSC '2000, 2000.

[126] Y. Yang, X. Wu, Parameter tuning for induction-algorithm-oriented feature elimination, IEEE Intelligent Systems 19 (2) (2004) 40–49.

[127] Z. Yanming, K. Youan, A genetic algorithm with sharing and its application in musicmethod, in: Fourth International Conference on Signal Processing Proceedings, ICSP '98, vol. 1, 1998.

[128] G. Yen, L. Haiming, Hierarchical genetic algorithm based neural network design, in: IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, 2000.

[129] B. Yuan, M. Gallagher, A hybrid approach to parameter tuning in genetic algorithms, The 2005 IEEE Congress on Evolutionary Computation 2 (2005) 1096–1103.

[130] Z. Yuanhui, L. Yuchang, S. Chunyi, Combining neural network, genetic algorithm and symbolic learning approach to discover knowledge from databases, in: IEEE International Conference on Systems, Man, and Cybernetics,: Computational Cybernetics and Simulation, vol. 5, 1997.

[131] M. Zhang, Y.-M. Hu, T. Wang, Optimal predicting method of peritoneal fluid absorption rate using genetic algorithm embedded in neural network, in: Proceedings of 2005 International Conference on Machine Learning and Cybernetics, vol. 7, 2005.

[132] Z. Zhang, Y. Zhou, Y. Lu, B. Zhang, Extracting rules from a ga-pruned neural network, in: IEEE International Conference on Systems, Man, and Cybernetics, vol. 3, 1996.

[133] J. Zhao, J. Shawe-Taylor, Stochastic connection neural networks, in: Fourth International Conference on Artificial Neural Networks, 1995.

[134] J. Zhao, J. Shawe-Taylor, A recurrent network with stochastic weights, in: IEEE International Conference on Neural Networks, vol. 2, 1996.