An Approach to Online Optimization of Heuristic Coordination Algorithms

Jumpol Polvichai
Computer Engineering Dept.
King Mongkut's University of Technology Thonburi
126 Tungkru,
Bangkok, 10140 Thailand
jumpol@cpe.kmutt.ac.th

Paul Scerri
Robotics Institute
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
pscerri@cs.cmu.edu

Michael Lewis
Information Sciences Dept.
University of Pittsburgh
135 North Bellefield Ave.
Pittsburgh, PA 15260
ml@sis.pitt.edu

ABSTRACT

Due to computational intractability, large scale coordination algorithms are necessarily heuristic and hence require tuning for particular environments. In domains where characteristics of the environment vary dramatically from scenario to scenario, it is desirable to have automated techniques for appropriately configuring the coordination. This paper presents an approach that takes performance data from a simulator to train a stochastic neural network that concisely models the complex, probabilistic relationship between configurations, environments and performance metrics. The stochastic neural network is 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.

Categories and Subject Descriptors

I.2.6 [**Learning**]: Connectionism and neural nets; I.6.5 [**Model Development**]: Modeling methodologies

General Terms

Algorithms

Keywords

Stochastic neural network, large scale coordination algorithms, complex system modeling

1. INTRODUCTION

In the near future, large scale automated coordination of humans, robots and software agents will revolutionize the achievement of complex goals in a variety of domains. Exciting work is being performed for domains including disaster response[8], military operations[3] and business[19]. Key characteristics of the environment impacting the coordination, e.g., observability[17], team size[5] and communication bandwidth availability, will vary significantly from scenario to scenario. For effective performance, the coordination algorithms need to adjust to the prevailing scenario, e.g., by communicating less when bandwidth is limited. In some cases, adaptation will involve tuning heuristics, while in other cases it will involve changing algorithms for all or part of the coordination. For example, in a small team when performance must be optimized, an auction may be used for role allocation, but for a bigger team, in a more dynamic environment a token based approach may be better[23].

Unfortunately, few approaches to adapting coordination can be found in the literature. Practical coordination algorithms are too complex for analytic optimization approaches [16]. Some algorithms have an ability to self optimize aspects of their behavior, however this ability is typically limited[11]. For other algorithms, some hand-tuning is possible, although this requires highly skilled operators[2]. Multi-agent learning is an active research area, but most algorithms require many iterations to optimize, making them inappropriate for quickly adapting to a particular scenario[7]. Hence, there is a need for an approach to rapidly optimizing coordination algorithms for a particular scenario.

This paper presents an approach to this problem, relying on three key ideas. First, an abstracted simulation of the coordination algorithms, the possible configuration of those algorithms and the potential characteristics of the environment is used to generate a large volume of data. That data is used to train a neural network model of the relationship between input values representing algorithm configurations and environment circumstances and output values representing aspects of performance. Precise performance of the algorithm will vary given randomness and asynchrony in the environment and algorithm interactions. To deal with this, *stochastic neural networks* are required to allow one input to neural network to have a distribution of possible outputs. Previous approaches to allowing stochastic output to neural networks had a range of drawbacks, so we developed a novel approach that adds a small number of internal, random nodes to the input of a normal neural network.

The stochastic neural network does not produce the same output every time for the same input, hence standard back propagation techniques cannot be used for training. Instead, to train the neural network, a genetic algorithm is used with a novel fitness function for determining whether the output distribution matches the data distribution. To compute the fitness, multiple outputs from the simulation and from the stochastic neural network are required. These are compared

Cite as: An Approach to Online Optimization of Heuristic Coordination Algorithms, Jumpol Polvichai, Paul Scerri and Michael Lewis, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.
Copyright © 2008, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

with a Pearson-Chi-Square test[6] to determine the degree of correlation and hence the fitness. The approach is shown to generate stochastic neural networks with good matches to a range of distributions.

Building on the concise representation of the mapping between algorithm parameters, environment and performance, a tool was developed that allows a user to specify preferences on output parameters and prevailing environmental conditions and be rapidly presented with coordination configurations for best performance. A fast genetic algorithm is required to search for optimal configurations, because not all inputs or outputs of the neural network are free to vary, thus making back-propogation impossible. The user is presented with a small set of options that meet their preferences but have either different performance tradeoffs or have different performance distributions. The tool can even be used online to reconfigure a running team to changing or newly understood environment conditions or changed user preferences. For example, early on the user might want the coordination to act quickly to bring a situation under control, using as much bandwidth as required, but later focus on quality at the expense of some time while also attempting to free up bandwidth for other purposes.

2. PROBLEM STATEMENT

We define S and C, as the sets of all possible configurations of the environment, and a set of all possible configurations of the coordination algorithms, respectively. $s \in S$ describes the key features of a particular environment in which the multi-agent system is operating. For example, s will capture the observability of the environment and the size of the team being coordinated. $c \in C$ describes how the coordination algorithms are configured. For example, the vector c may include components for which task allocation algorithm to use and parameters impacting communication decisions. The precise components of these vectors will depend on the coordination algorithms. For the token based algorithms which are the focus of this paper, $|s|$ and $|c|$ are at least 12.

We define P as the set of performance measures. For example, performance measures will include probability of goal achievement, communication bandwidth used and resources utilized. The function $M : S \times C \rightarrow P$ maps from an environment and set of configuration parameters to a measure of performance. We refer to M as the *Team Performance Model*. The team performance model is typically not known, nor can it be analytically constructed. At a particular time, the user will have a preference for tradeoffs with respect to P. For example, they may be willing to sacrifice some performance to keep communication bandwidth use to a minimum. In general, we capture this via a function $f : P \rightarrow \mathcal{R}$. Typically, we consider f to be a weighted sum over P, i.e., $f(P) = \sum_{p_i \in P} w_i p_i$. Thus, the aim of this work is to find:

$$\arg\max_{c \in C} f(M(s, c))$$

The major challenge to this research is the complexity of the relationship between the configuration parameters of coordination algorithms and the performance measures for large teams. Figure 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

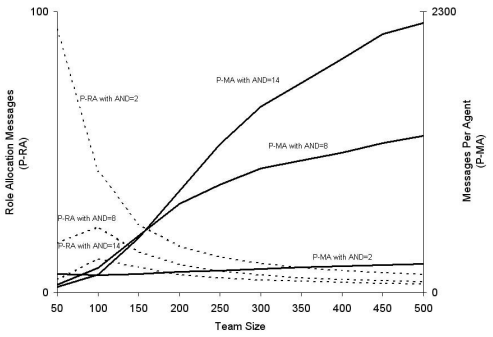


Figure 1: Example of the complex relationships between parameters and performance.

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) [18] 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 a complex relationship.

3. OVERALL PROCESS

Figure 2 shows the outline of the approach. Data is collected from a simulation environment that is sufficiently high fidelity to capture key algorithm and performance features but sufficiently fast to be able to perform a large number of simulation runs. This is often feasible for large scale coordination algorithms simply by abstracting away the environment and using internal message passing for communication, since typically these are the most time intensive components. Next, a evolutionary learning process utilizes the collected data to generate the *Team Performance Model*. Finally, the team performance model is used as the core of a tool that allows users to optimize team performance prior or during a scenario. In the following sections, each of these steps is described in additional detail.

4. DATA COLLECTION

The initial step of the process is the collection of a large volume of data. This data either already exists, e.g., from

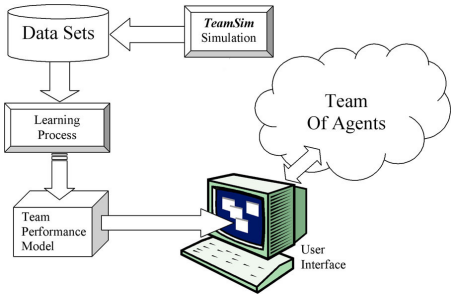


Figure 2: High level view of overall process.

many training exercises, or needs to be gathered. In this paper, data is collected from an abstracted simulation environment. Several computer years were used to generate simulation data, with each run taking between 1s 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 is made up of two parts: the input vectors, s and 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.

Training Data Requirements

Since the space of possible configurations is very large, training data can only be collected for a small subset of the space. Fortunately, statistical techniques exist which bound the data collection requirements, while providing guarantees on coverage.

The most widely used methods to determine appropriate sample size for given accuracy and confidence parameters use Hoeffding bounds (or the additive Chernoff bounds)[9, 21]. The Hoeffding bounds characterize the deviation between the true errors and observed errors over n independent trials, as

$$P\{Error_{true} > Error_{n-trials} + \varepsilon\} \leq e^{-2n\varepsilon^2} \quad (1)$$

This characteristic gives us a way to ensure 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

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

However, this equation can not be applied to infinite hypothesis spaces[12], thus are unapplicable here. For neural networks, particularly Perceptron networks containing s Perceptions, each with r inputs, we can bound the number of training examples sufficiently to learn (with probability at least $1-\delta$) any target concept within error ε[12]. We have

$$m \geq \frac{1}{\varepsilon}\left(4log_2(\frac{2}{\varepsilon} + 16(r+1)slog_2(es)log_2(\frac{13}{\varepsilon})\right) \quad (3)$$

However, this approach fails to apply for backpropagation networks[12]. The method that we employ calculates 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[10] $m = |L|$ 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:

$$m \geq \frac{(10z_{\alpha/2})^2(1-sp\%)}{sp\%} \quad (4)$$

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 confident interval (which has the width less than 0.04% the total possible space). For this work, we collected 21000 samples.

.

5. STOCHASTIC NEURAL NETWORKS

To capture the team performance model, with the nonlinearities, output distributions and 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 (SNNs)*.

Previous approaches have shown significant potential for representing uncertainty as a form of stochastic neural networks. A special kind of stochastic neural network, called

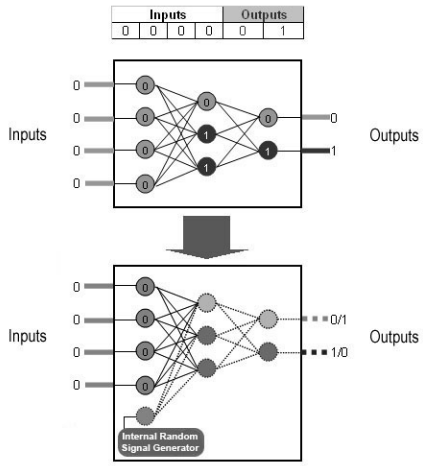
Approximate Identity Neural Networks (AINN) [22], is very powerful and effective in approximate, complex, non-linear systems. However, implementation of an AINN is very complicated and becomes significantly more difficult as the number of input parameters grow. Alternatively, neural networks with *stochastic resonance* were introduced, using a time periodic signal as a stochastic element[14]. 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 represent complicated gene regulatory networks using Poisson random signals[20]. 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.

With inspiration from the dynamic rearrangement [1], we developed a new type of stochastic neural networks that has the properties required to capture the team performance model. As shown in Figure 3, a normal artificial neural 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 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 three random input nodes will effectively capture even the most complex distributions. 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



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: A neural network can be transformed into a stochastic neural network by adding extra input nodes feed with. These random signals are stochastically changed every time the network is executed.

configuration and system performance instances.

Team Performance SNN

To capture the highly non-linear relationship between environment, configuration and team performance an SNN was constructed. A three-layer feed-forward network is was chosen since it is capable of representing any arbitrary function [15]. The network topology consists of 31 nodes in the input layer, 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. In the next section, the training of this network is described.

5.1 Genetic Algorithm Training

A genetic algorithm is a search technique loosely based on the concept of natural selection[4]. 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 are generated using three common genetic operators: reproduction, crossover, and mutation with fitter individuals more likely to be chosen as a basis for the genetic operators. This process repeats until either a sufficiently fit individual is found or time expires. The solution of the problem is found in the final population. 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 SNN to the target data in order to determine which individuals to propogate to the next generation. Determining fitness fis more techni-

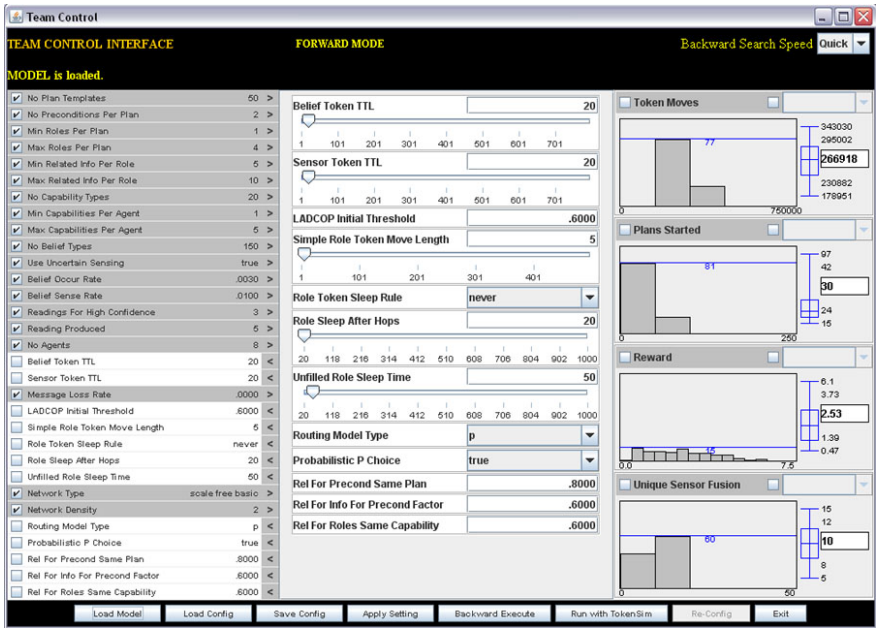


Figure 4: Team control interface displays all input parameters on the left side, performance measures on the right side, selected editable parameters in the middle, and command buttons in the bottom.

cally challenging than for vanila neural networks because or an SNN 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 Pearson Chi-square (χ^2) test [6]:

$$\chi^2 = \sum_{i=1}^{n} \left(\frac{(f_s - f_s^*)^2}{f_s^*} \right), n = n_a + n_t \tag{5}$$

where n denotes the total number of slots, n_a denotes the number of slots in actual distribution, n_t denotes number of slots in target distribution. Typically, these values will be equal. 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:

$$f_s^* = \frac{t_o \cdot t_s}{T} \tag{6}$$

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 [6], we can get an index of the strength for the relation between these two distributions, using the formula:

$$V = \sqrt{\frac{\chi^2}{N(k-1)}} \tag{7}$$

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:

$$Fitness_i = \frac{\sum_{d \in D} \sum_{p \in P} \left(\chi^2_{d,p} + |\mu_a^{d,p} - \mu_t^{d,p}| \right)}{|D| \cdot |P|} \tag{8}$$

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.

Training process

To compute fitness value for a stochastic neural network$_i$ in a particular generation, the process is described in Algorithm 1. $Fitness_i$ 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. To enable the possibility of presenting the user with options of different configurations, the genetic algorithm keeps track of the ten best individuals. Notice that while this process is marginally more complex than computing the fitness for a normal genetic algorithm, since neural networks are so fast, it is not significantly slower than the normal process.

Specific Values

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 initialized. After evaluation of the training data set, the 100 best performers were kept

Algorithm 1: Training process

(1) **foreach** $data_k$ in training data set
(2) Execute SNN_i with the same input values from $data_k$ t_a times
(3) $Score \leftarrow 0$
(4) **foreach** $output_j$
(5) compute distributions of actual $output_j$ and its actual mean ($\mu_a^{k,j}$)
(6) use target $output_j$ distribution from $data_k$ to compute the target mean ($\mu_t^{k,j}$)
(7) compute predicted frequency f_s^* for each s (Eqn. 6)
(8) compute $\chi^2_{k,j}$ (Eqn. 5)
(9) compute $Score = Score + \chi^2_{k,j} + |\mu_a^{k,j} - \mu_t^{k,j}|$
(10) $Score \leftarrow Score/|P|$
(11) $Fitness_i \leftarrow Score/|D|$

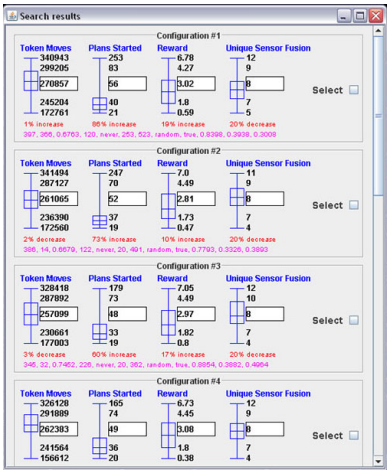


Figure 5: Dialog box for presenting four options to the user for reconfiguration.

and used to produce the 1900 new individuals, via genetic operations. Other genetic algorithm values were used, but none were found that performed significantly better.

6. USER INTERFACE

The SNN 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 interface is shown in Figure 4. At the top right of the interface is a choice of backward search speeds, the slower the speed the more chance the genetic algorithm finds a good configuration. The middle of the interface has the three key panels. On the left is a list of input parameters, with their current values. The user sets system values, either before the scenario or during the scenario as the values become apparent. Parameters with a white background are ones that can be changed by the reconfiguration process. In some cases, the user can determine whether or not a value can be changed. For example,

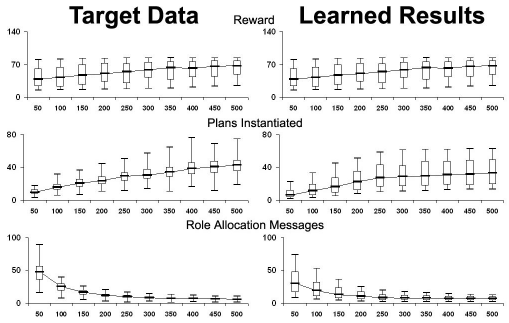


Figure 6: Representative comparison between data from simulator and model learned by SNN. This represents a tiny fraction of the overall model.

in some domains, the number of robots might be fixed, while in others the user may be able to vary the number of robots.

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 (see below). This usage of the model, referred to as "forward mode" because the underlying SNN 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.

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 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 which performance values the user wants increased or decreased. 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.

Once the user selects which parameters they want increased or decreased, they click "backward execute" and are presented with a window showing ten configurations offering different tradeoffs with their distribution. Figure 5 shows a dialog box offering four different choices (the user needs to scroll to see additional choices). The user may be able to select between configurations with a higher uncertainty but higher average or with lower uncertainty but less desirable averages. After a decision is made the message goes out to the team to get the change made.

7. RESULTS

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

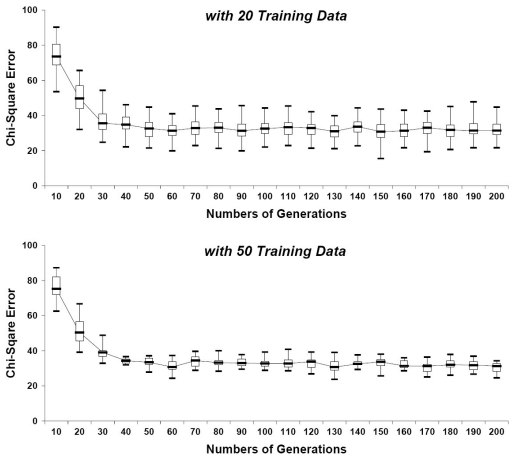


Figure 7: Plots display a comparison of learning results with different numbers of training data using in each generation. Values in y-axis are Chi-Square errors and values in x-axis are numbers of generations.

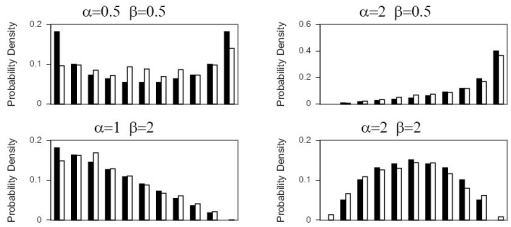


Figure 8: A set of 4 plots with different values of α and β. The target distributions are shown in solid black bars and actual distributions are shown in white bars. There is an average of less than 10 % error.

7.1 Model Accuracy

Figure 6 shows a small subset of data from the simulator compared with results from the trained model. These graphs are representative of the type of correlation that was found between data and model. Notice that general trends in the data were reasonably captured by the SNN. Trends in the distributions were generally followed, however the absolute amount of uncertainty was not always well captured. Wider distributions in the data were often over estimated. Next we show how the genetic algorithm learned the model. Figure 7 shows cases where 20 and 50 training samples were used for evaluating fitness (see Algorithm 1). The more data used the more computation required, but there is a corresponding increase in performance. Notice the Chi-square error gets to about 20% which is approximately the amount of error shown in Figure 6.

7.2 Capturing Output Distributions

A beta distribution is a versatile distribution that has been commonly used for modeling data with uncertainty in many applications [13]. The probability density function of the beta distribution is calculated as follows:

$$f(x;\alpha,\beta) = \frac{1}{Beta(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \qquad (9)$$

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

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

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. 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.

Figure 8 shows the results of tests where an SNN was trained to match different beta distributions. Notice that the beta distribution is the uncertainty for a particular input and hence must be captured by the stochastic component of the SNN. Black bars show the target distribution and white bars show the distribution produced with the SNN. For each of the different distribution types, there is a good match with the SNN, although the 'U'-shape is most problematic. These results are representative of a larger set of tests done with 20 different α and β values. The 'U'-shape was the poorest match of all the 20 cases.

7.3 Reconfiguration Validation

In the final experiment, the performance of the team performance model and its use through the team control interface are examined. The team control interface is connected directly with the abstract simulator used to generate the original data so that a user can set team configurations and monitor team performance measures online. The user configures the team at the beginning of the mission. Team performance as measured from the simulation is graphically displayed on the user interface at each time step. When performance changes are requested, the backward search feature of the team performance model is used to find suitable reconfigurations. The changed team performance is displayed from that time step onwards. The simulation is artificially slowed down to allow the user the chance to request configuration changes. Notice that in this case the interface tool is connected to the simulation and not to a real team. This makes running tests and measuring performance easier, but tests with a real team are planned for the future.

The team control interface was evaluated with eight randomly selected configurations. In the first set of tests, the user requested that single performance measures were either increased or decreased. This resulted in 64 cases, each of which was run 5 times. As shown in Table 1, in 52 of the 64 cases performance increased or descreased as the user requested; in 12 of the 64 cases it did not. When the user asked some performance measure to increase, it did in 25 of 32 cases, increasing on average 435% (however, without one outlier, the average increase was 73%). When the performance measure did not increase as requested, the decline was only 24% (excluding one outlier only 12%). When the user asked for a decrease in a single performance measure, a decrease was observed in 28 out 32 cases. 3 of the 4 times

Table 1: Reconfiguration Results

	One Constraint	More Constraints
Success	52 (81.25%)	141 (88.125%)
Fail	12 (18.75%)	19 (11.875%)

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% and when performing incorrectly, the measure went up on average 115% (removing one outlier changes this to 29%). These tests show that the tool generally has the ability to meet simple user requests, although there is significant room for improvement.

A slightly more complex test involved the user specifying two or more performance measures to be increased or decreased simultaneously. This experiment involved 160 cases, again averaged over 5 runs. As shown in Table 1, in 141 of the 160 cases, the change requested by the user was observed. This shows 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 performance cannot be increased beyond its maximum, and in other cases, on average the new configuration might have given the correct result, but uncertainty in the coordination resulted in it not being seen. Since the configuration space is so large, it is infeasible to determine accurately what percentage of the failures above were due to these reasons.

8. CONCLUSIONS

Large scale coordination will become increasingly important as robots and agents become more effective and less expensive. The coordination algorithms controlling the teams will necessarily 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 paper 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 then used as the core of a user interface tool. Results show that the model captured key features of a very large configuration space and mostly captured the uncertainty in performance well. The tool was shown to be often capable of reconfiguring the algorithms to meet user requests for increases or decreases in performance parameters. We believe this represents the first practical approach to quickly reconfiguring a complex set of algorithms for a specific scenario.

Immediate future work is focused on interacting with an actual distributed team, determining the current situation and autonomously reconfiguring to improve performance. While the model is generally accurate enough for the specific purpose it is being used for, more accuracy is desirable. One direction for investigation is adaptive selection of training examples, to focus better on areas with more complex relationships and higher uncertainty. Finally, we would like to find ways to bring down the very large amount of computation required to produce the training data and train the models. While Moore's law will make the problem somewhat less critical, it will not always be possible to make very fast simulations of the target algorithms.

9. REFERENCES

[1] P. Eggenberger, A. Ishiguro, S. Tokura, T. Kondo, and Y. Uchikawa. Toward seamless transfer from simulated to real worlds: A dynamically-rearranging neural network approach. In *8th EWLR*, 1999.

[2] R. Falcone and 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.

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

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

[5] G. Kaminka and M. Bowling. Towards robust teams with many agents. Tech. Report CMU-CS-01-159, CMU, 2001.

[6] F. N. Kerlinger and H. B. Lee. *Foundations of Behavioral Research.* Harcourt College Publishers, 4th edition, 2000.

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

[8] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjoh, and 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*, volume VI, 1999.

[9] J. Kivinen and H. Mannila. The power of sampling in knowledge discovery. In *In Proc. of Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium, Principles of Database System*, 1994.

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

[11] E. Malville and F. Bourdon. Task allocation: A group self design approach. In *ICMAS'98*, 1998.

[12] T. M. Mitchell. *Machine Learning*, chapter Computational Learning Theory, McGraw-Hill Higher Education, 1997.

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

[14] T. Nobori and N. Matsui. Stochastic resonance neural network and its performance. In *IEEE-INNS-ENNS Int. J. Conf. on Neural Networks*, 2000.

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

[16] D. Pynadath and M. Tambe. Multiagent teamwork: Analyzing the optimality and complexity of key theories and models. In *AAMAS'02*, 2002.

[17] D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. In *JAIR*, 2002.

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

[19] C. Rolland, S. Nurcan, and G. Grosz. A unified framework for modeling cooperative design processes and cooperative business processes. In *Thirty-First Hawaii International Conference on System Sciences*, 1998.

[20] T. Tian and K. Burrage. Stochastic neural network models for gene regulatory networks. In *Congress on Evolutionary Computation*, 2003.

[21] H. Toivonen. Sampling large databases for association rules. *22nd International Conference on Very Large Data Bases*, 1996.

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

[23] Y. Xu, P. Scerri, K. Sycara, and M. Lewis. Comparing market and token-based coordination. In *AAMAS'06*, 2006.