Configuration and Reconfiguration of Large Teams

Paper ID: 320

Abstract

Coordination of large numbers of agents for performing 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 of algorithm parameters to get the best performance in particular circumstances. This tuning of parameters is something of a black art. In this paper, we use a dynamic neural network to model the way a coordination algorithm will work under particular circumstances. The neural network model can be used to rapidly determine an appropriate configuration of the algorithm for a particular domain. A user can also specify required tradeoffs in algorithm performance and use the neural network to find the best configuration for those tradeoffs. Reconfiguration can even be performed online to improve the performance of an executing team as situation changes. We present results showing the approach facilitating users to configure and control a large team executing sophisticated teamwork algorithms.

1. Introduction

Sophisticated, complex coordination allows large groups of agents to perform complex tasks in domains such as space [6], the military [4] and disaster response [5]. Due to the high computational complexity of coordination, critical coordination algorithms typically use heuristics which are parameterized and need to be tuned for specific domains for best performance. For example, different coordination configurations might be required for different rates of change in the world, individual failure rates or communication bandwidth availability. A coordination configuration specifies parameter values for a team's coordination algorithms. When several coordination algorithms are used together, e.g., algorithms for task allocation, communication and planning, the performance of one algorithm will likely affect the performance of the other algorithms, thus tuning parameters of the individual algorithms must be performed together. As we show in Section 2, the relationship between coordination configuration, e.g., the number of roles that each team member can handle, environmental conditions, e.g., the observability of the environment, and performance, e.g., the number of messages that will be sent during coordination, is highly non-linear and extremely complex. Hence, getting best performance from a set of coordination algorithms often involves a complex parametric tuning process, that must be performed on a per problem basis.

Previous approaches to configuring a team for a particular domain typically either required hand-tuning [3] or learning [1] in the domain. Hand-tuning of parameters is a time consuming process that typically requires extensive experience with the algorithms for good performance. Learning requires that the team perform many trials in the specific circumstances it is to be used. If the environment can vary dramatically, e.g., the specific characteristics of a disaster response scenario can vary greatly from disaster to disaster but the same team should respond to each one, learning may be infeasible.[11] Thus, previous work does not provide a good solution to the problem of rapid team configuration.

We have developed an approach to configuring coordination algorithms that incorporates three key ideas. The first idea is to create a *team performance model* that captures the relation between the environment and team configuration parameters and measures of performance in a way that allows rapid exploration by users. We have developed an abstract simulation of the coordination algorithms and a highly configurable environment to test these ideas. Because the simulation is fast it can be used to create large amounts of data containing relationships between parameters. Due to the non-determinism of environments and coordination algorithms and the sensitivity of performance to circumstances these relationships are highly non-linear. They are also highly variable even for the same configuration. To create a concise model of the data we use genetic algorithms to learn a dynamic neural network.[8] A neural network with two hidden layers is sufficiently powerful [7] to capture the complexity of these data and thus provides a rapid mapping from environment and configuration parameters to performance parameters. The aim of this work is to remove some of the art from configuring a team for a particular environment. The team performance model captures

the complex relationship between the environment, configuration and performance of the team.

The second idea in this work is to use the team performance model to find the best configuration of the team to meet specific performance constraints, e.g., the tradeoff between communication bandwidth and good 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, we cannot simply use back propagation of the neural network to find input parameters that meet our output requirements. Instead we perform a search over the changeable configuration parameters to find a configuration that best meets the required performance tradeoffs.

The third idea is to allow the team to be reconfigured online, using the team performance model to determine appropriate parameters for the prevailing conditions. The reconfiguration can either be prompted manually by a human or initiated autonomously by an agent monitoring team performance. 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 for manipulation of team performance during execution time and provides an additional mechanism for the supervisory control of executing teams.

We have implemented and tested this approach with an abstract teamwork simulation, TeamSim as well as with the fully distributed Machinetta proxy [13][12] architecture in a domain. The abstract teamwork simulation was also used to provide a large amount of data from which the dynamic neural network model was created. The model was shown to accurately capture the behavior of the coordination algorithms across a wide range of configurations. The model was then used to configure teams and change configurations online with results as predicted by the model.

2. Problem Model

We define S and C, which are a set of system parameters and a set of configurable parameters respectively. The system parameters are fixed by the environment, but may change during the mission, e.g., communication networks might become congested. The configurable parameters are allowed to change. Next, let us define P, which is a set of system performances measures. The sets S, C and P will depend on mission domain and coordination algorithms. Table 1 and Table 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 bigger sets. 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. Finally, with user preferences function, $f(P) \rightarrow Value$, the aim of this work is to find $\arg\max_{c \in C} f(M(s, c))$.

Configuration Parameters	Range	Type
Number of Team Members	*10 ... 1000*	S
Number of Plan Templates	*1 ... 20*	S
Roles Per Team Member	*1 ... 5*	S
Total Number of Preconditions	*20 ... 120*	S
Preconditions Per Plan	*1 ... 10*	S
Roles Per Plan	*1 ... 10*	S
Plan Template Policy	*0.0 ... 1.0*	S
Number of Capability Types	*2 ... 20*	S
Percent Capable	*0.0 ... 1.0*	S
New Precondition Rate	*0.0 ... 1.0*	S
Precondition Detection Rate	*0.0 ... 1.0*	S
Associate Network Density	*1 ... 16*	S
Role Threshold	*0.0 ... 1.0*	C
Instantiation Rule	*0.0 ... 1.0*	C
Instantiate Rate	*0.0 ... 1.0*	C
Information Token (Time To Live)	*1 ... 10*	C

Table 1. List of configuration parameters with their possible ranges and types. (S=system, C=configurable)

Performance Parameters
Percentage Possible
Reward
Messages Per Agent
Conflicts Detected
Plans Instantiated
Conflict Resolution Messages
Role Allocation Messages

Table 2. List of measures of performance.

2.1. System Complexity

The major difficulty of this work is the complexity of interactions in multi-agent system. From Table 1, with the total of 16 input parameters and their possible ranges, the combination of these parameters, which imply coordination configurations for this problem, are very huge: more

NO_TEAM_MEMBERS > 900
PRECONDITIONS_PER_PLAN <= 1
NEW_PRECONDITION_RATE > 0.208264
PRECONDITION_DETECT_RATE <= 0.0541394
ASSOCIATES_NW_DENSITY > 5
ASSOCIATES_NW_DENSITY <= 8
————-> class V.Good [87.1%]

Table 3. An example of rule generated by Decision Tree induction (C4.5).

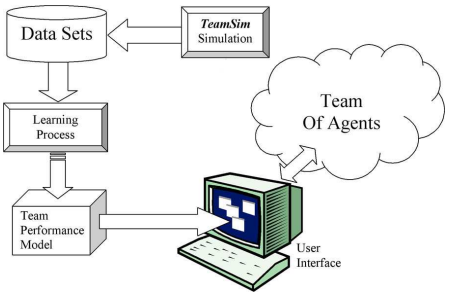


Figure 1. Process diagram.

than 10^{25} 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.

Previously we have investigated modeling the system by using decision tree induction (C4.5)[10]. With 14 input attributes, only 4 distintive output classes and 30000 cases (much smaller than the actual possible problem space we use in this work), we ended up having 573 classification IF-THEN rules. On test data, these rules performed with only 74.2% correct classification. However, these rules were far from being able to solve our configuration problem, because the C4.5 decision tree only classified into small classes making optimization difficult. An example of one of the generated rules is shown in Table 3. This rule basically says that if you have as follows: total number of members of the team is more than 900 members, average number of preconditions per plan is not more than 1, chance of a particular precondition coming true at any step is more than 20%, chance that a particular team member locally senses a particular new pre-condition is very low, and average number of links each team member creates between 5 and 8, then the performance will be very good with 87% confidence.

3. Algorithm and Approach

We have developed an approach to facilitate users in understanding and manipulating the relationship between configurations of coordination algorithms and measures of their performance. The approach has several steps. It starts with the collection of large data sets generated by our abstract teamwork simulation. Then, an evolutionary computation approach is used to learn team performance models. An input/output model of artificial neural networks with dynamic features is used to represent the team performance model. Finally, mission experiments with our abstract teamwork simulation and a network of Macinetta proxies were used to investigate the performance of the team performance model. Figure 1 shows the outline of our approach.

3.1. Data sets

TeamSim is an abstract teamwork simulation developed to investigate coordination algorithms in cooperative multi-agent teams within simulated environments. Running the simulation provided a huge training data set. This data set was used to learn models of team performance. To create a precise team performance model, very large amounts of data were needed to capture the complex relationships between parameters. As we show in section 2, the complete configurable space is very large. Providing full coverage of all possible configurations was impossible. To overcome this difficulty we collected simulation data using two methods: non-random sampling and random sampling. The non-random sampling method gathered data by specifying cases chosen to roughly to cover the possible ranges of input parameters. Although this data set roughly represents the entire configuration space it has sparse coverage. The random sampling method was designed to collect data from throughout the space but without guarantees of coverage. It took several weeks to collect the approximately five hundred thousand data entries used in this study. Each data entry consists of all environmental and configurable parameters and output performances from a run.

3.2. Neural Network Models

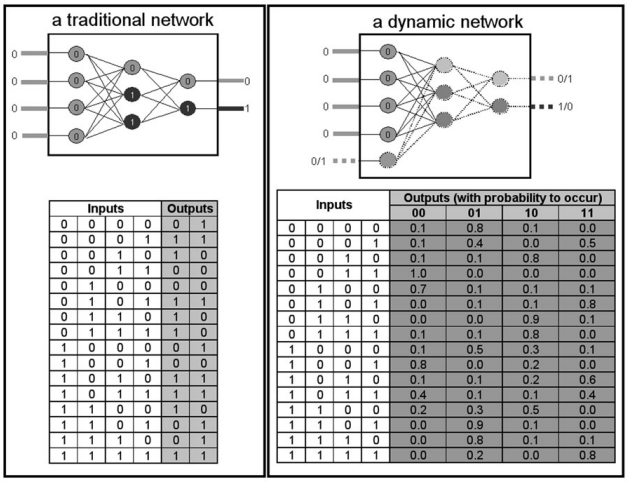
To represent the highly non-linear relationship between the environment, configuration and performance of the team, we used multilayer feed-forward neural networks. The network topology consists of sixteen nodes in the input layer (one input node representing each configurable or system parameter), sixteen nodes in the first hidden layer, eight nodes in the second hidden layer, and seven nodes in the output layer. The two hidden layers used sigmoid units, and the output layer use linear units. This

three-layer feed-forward network is capable of representing any arbitrary function.[7]

Peter Eggenberger et al [2] introduced the idea of dynamic rearrangement of biological nervous systems to accommodate learning in nonstationary 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 is changed.

With inspirations from the dynamic rearrangement idea [2], we use the concept, called *Dynamic Networks* [8][9], which allows all internal nodes in the network to act stochastically and independently even though all external input data remain unchanged. Figure 2 shows an abstract example of a dynamic network compared with a regular network. Dynamic networks capture randomness from the additional input nodes fed with internal random signals. These random signals along with weights between the additional nodes and the hidden nodes bring dynamic states into internal nodes within the network and output nodes. Changing the weights results in changing behaviors of the dynamic networks. This kind of network enlarges the capability to deal with non-deterministic problems. Moreover, this stochastic-ness adds robustness and flexibility to the network. If a target system has high variation in output even for the same input configuration, the dynamic network can adapt the weights to match this variance. If the target system were deterministic, these weights would adapt to zero.

Because team coordination algorithms are dynamic and non-deterministic a Dynamic Network provides a good model. Dynamic networks were applied into our multilayer neural network by adding four special nodes into the input layer, so that the total number of input nodes becomes twenty. These special nodes insert random values between 0.0 and 1.0 into the network.



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 2. 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 every input with all possible output patterns with different probabilities of each output one to occur. By allowing all active nodes to turn on/off stochastically (depicted by dash lines), with the same input pattern, during a period of time, the network changes internally and produces dynamically all possible output patterns, which finally represent a non-deterministic control system.

3.3. Genetic Algorithms

Genetic Algorithms (GA) is a search technique loosely based on the mechanism of natural selection and genetics. In relation to problem domains, structures of a possible solution are represented in string formats. Given an environment and a goal formulated as a fitness function, an initial population and selecting genetic operators are generated at random. The new generation of possible solutions is generated using three common genetic operators, namely, reproduction, crossover and mutation. A number of processes of generating new populations based on prior populations are repeatedly executed until the termination condition is met. The solution of the problem is found in the final population.

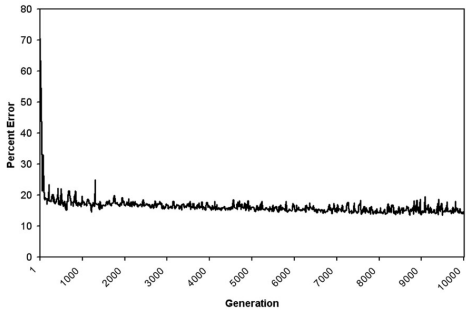


Figure 3. A graph shows the best fitness of the GA after each generation.

Each generation of the population was set to have a thousand individuals. Each individual represented a neural network. The chromosomes of each individual were the weights of the neural network. All weights are randomly generated at the start. After evaluation with the training data set, the first 500 best performances are kept and then used to produce the new 500 individuals randomly by genetic operations (i.e. crossover, mutation). The fitness function was the average of square error between target output and actual output, $\sum_{d \in D} (O_t^d - O_a^d)^2 / sizeof(D)$. The lower this score is, the better. The training data was sampled from the data set, so that different training data was used in each generation. The size of each training data set was 5000. The termination conditions were either reaching the maximum iteration (10000) or reaching the minimum error of the network output (0.05). When the termination condition was met, the best performing individual was chosen to be our team performance model.

We learned our multilayer feed-forward neural network using an evolutionary algorithm because the relationship between variables was not only non-linear, but also highly dynamic and non-deterministic, which is demanding for backpropagation which must overcome a huge number of local minima. Secondly, in GA, the unit of adaptation is not an individual agent, but a population of agents, which is excellent for dealing with very huge and noisy training data set. Figure 3 shows how quickly the GA converged. The learning process converged to 20 percent error quickly and slowly converged to 15 percent error after that.

3.4. User Interface

Our user interface is shown in Figure 4. The user 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 parameters and performance. The offline mode has two features: *input-to-output feature* and *output-to-input feature*.

Input-to-Output Feature Using the team performance model in forward mode, a user can experiment with changing input parameters and observing how output parameters change. Although System parameters are part of the team performance model they cannot be changed from the interface. Users can change Configurable parameters. When the user sets a configuration a full model with all input parameters is used to estimate the performance parameters displayed on the interface.

Output-to-Input Feature Using the team performance model in backward mode, the interface allows a user to change output parameters and receive a configuration that best meets some specific performance constraints

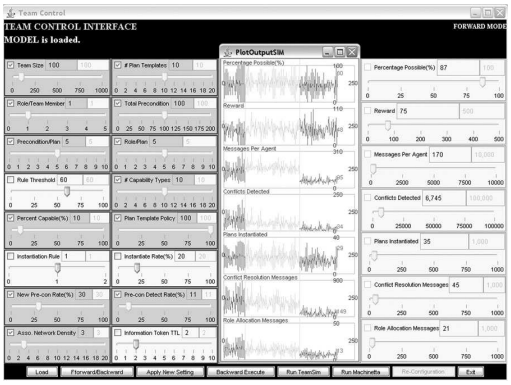


Figure 4. User interface is designed to use with the team performance model. When the model is loaded, input parameters are shown on the left side. Performance measures are shown on the right side. The user can switch between forward mode and backward mode when working offline. In online mode, a plotting window is shown in the middle.

both in input and output. The user specifies domain parameters by selecting check boxes. In order to find input parameters that meet output requirements, the interface performs a search over the changeable configuration parameters using the team performance model to find a configuration that gives the required performance tradeoffs. The search space covers all changeable configuration parameters and the search constraints are all output performance parameters.

In online mode, the user interface is connected with TeamSim and allows the user to display system 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. When the user changes the configuration online, a mark is indicated on the plotting window. During execution in online mode, the user is allowed to use offline interface features such as output-to-input to aid in selecting reconfiguration parameters.

4. Experiments and Results

In this section we present evidence that our approach achieves our primary goals which are: 1) capturing the effects of configuration parameters on performance measures in the team performance model 2) demonstrating the correspondence between predicted and obtained changes in performance using the output-to-input feature of the team performance model 3) demonstrating that relations be-

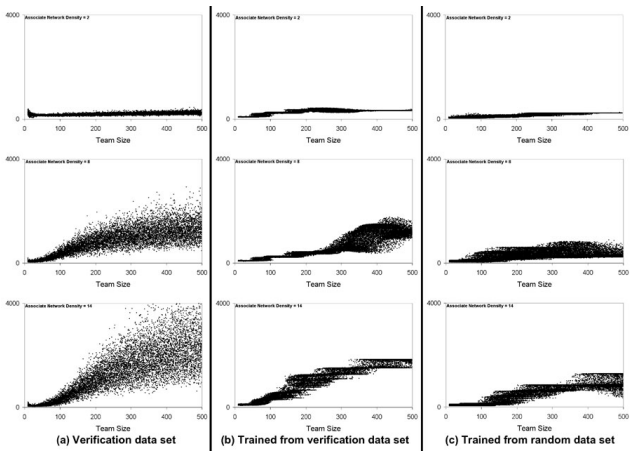


Figure 5. These graphs are plotted between number of messages per agent and number of team members. Each row shows graphs for different numbers of communicating neighbors. In column (a), three graphs are plotted from the data in the verification data set. After learning process, graphs in cloumn (b) and (c) are plotted from the team performance model learned from verification data set and learned from random data set respectively.

tween configuration parameters of team performance hold for higher fidelity team simulations

4.1. Team Performance Model Verification

The objective of the team performance model is to capture the relation between a vast space of possible system and configuration parameters and the performance measures. To obtain high resolution for comparisons, we created a small reference data set consisting of data obtained from running our abstract simulation by varying only 3 input parameters: *Number of Team Members*, *Associate Network Density*, *Plan Template Policy*, leaving other input parameters fixed. Some of these data entries are plotted in column (a) of Figure 5. A second data set of much lower resolution was created by generating the same number of observations using random configurations over all the input parameters. These two data sets were used to learn new team performance models which in turn generated graphs which were compared with those plotted from the original reference data set. As Figure 5 shows there is close correspondence between the team performance models and data set in the relation between messages per agent and number of agents (columns) for each level of network density (rows).

		1	2	3	4
Team Configuration	Number of Team Members	200	200	200	200
	Number of Plan Templates	10	10	10	10
	Roles Per Team Member	1	1	1	1
	Total Number of Preconditions	100	100	100	100
	Preconditions Per Plan	5	5	5	5
	Roles Per Plan	5	5	5	5
	Plan Template Policy	100%	100%	100%	100%
	Number of Capability Types	10	10	10	10
	Percent Capable	10%	10%	10%	10%
	New Precondition Rate	30%	30%	30%	30%
	Precondition Detection Rate	11%	11%	11%	11%
	Associate Network Density	3%	3%	3%	3%
	Role Threshold	50	40	40	10
	Instantiation Rule	ALWAYS	ALWAYS	ALWAYS	LOCAL
	Instantiate Rate	30%	70%	40%	90%
	Information Token Time To Live	3	9	2	2
Performance	Percentage Possible	89	91	88	85
	Reward	76	85	75	81
	Messages Per Agent	230	302	265	242
	Conflicts Detected	116	145	117	79
	Plans Instantiated	25	28	22	16
	Conflict Resolution Messages	365	490	370	240
	Role Allocation Messages	24	25	21	18

Figure 6. Four team configurations are set during the mission according to occurred situations, and along with performance measures predicted by the user interface.

As expected the correpondence is closest for the high resolution model yet even the low resolution model captures the increases in messaging with team size and the flattening of the curves as network density decreases. While these comparisons are qualitative they support our approach by demonstrating that team performance models can learned for large parameter sets that perform well even when examined closely over a small range of settings.

4.2. User Interface Verification

In this subsection, we examine the performance of the output-to-input feature of the team performance model and its use through the user interface.

We use TeamSim, our abstract simulation, as the target system. The interface is connected directly with TeamSim, so that users can set team configurations and monitor team performance measures online. The user configures the team at the start of the mission. Performance measures from the simulation are graphically displayed on the user interface at every time step. When performance changes are requested the offline features of the team performance model are used to find suitable reconfigurations

The user interface and reconfiguration assistance were evaluated over 10 scenarios. Scenarios were selected to provide situations that would require users to reconfigure their team in order to meet performance targets. For example, in a mission involving a very large team of 300 agents the user might be requested at some point in the mission to reduce the number of messages per agent or increase the number of plans instantiated. Performance measures are recorded throughout the execution. Each scenario was run for 250 time steps, with each step taking 5 seconds. The data presented here represents 4 hours of runtime with a user in the loop. One scenario with a team of 200 agents is shown in

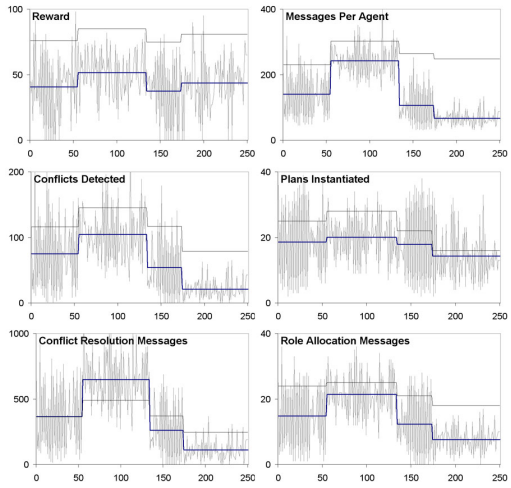


Figure 7. Six performance measures recorded from TeamSim are ploted during the mission with 3 times of reconfiguration. Thick lines show the average values of actual performance measures of each configuration setting. Thin lines are the predicted values by the user interface.

Figure 6. At step 2, the user is asked to increase level of rewards obtained by the team disregarding other performance measures. Using the output-to-input feature of the team performance model the user finds a new coordination configuration that increases reward performance and reconfigures the team. At step 3 network communication bandwidth is reduced limiting the time-to-live for information tokens to 2 hops requiring another team reconfiguration to lessen the degradation in performance. At step 4, the user is again asked to reconfigure to improve reward performance. Results for six of the performance measures are shown in Figure 7. The bold lines show average values for the configured system while the lighter lines indicate the values predicted by the output-to-input model. The jagged lines show the moment to moment variation in the actual performance measures. Despite the high variablity of team performance measures the model accurately predicts the effects of reconfiguration on average performance values across all six measures. By demonstrating the team performance model's effectiveness for predicting the effects of team configurations these tests demonstrate the potential of our approach for both the initial configuration of teams and supervisory control of executing teams.

4.3. Experiments with Machinetta proxies

In this section we demonstrate that changes in a team simulation parameter can produce changes in performance for a more realistic team simulation. While a team performance model has not yet been learned for this environment, this demonstration suggests the feasibility of such a model. We are integrating our software with Machinetta[13][12], a proxy-based team coordination infrastructure for teams of robots, agents, and people (RAPs). In Machinetta, each RAP is represented in team coordination by a proxy, implemented as a lightweight Java process. These proxies exchange coordination information, and each can also communicate with its own RAP. The flexibility and sophistication of Machinetta proxies will allow us to test our approach in complex, dynamic settings.

For this test we treated our user interface as a RAP and provided a proxy for it. This proxy periodically sent data collection messages, which circulated through the team and collected statistics on the team status and performance. These messages were returned to the originating proxy, where they are processed and updated statistics displayed on the user interface. The user reconfigured the team by sending reconfigure messages through the interface proxy which were circulated through the team, informing proxies (and hence the team members) of the change.

We tested this mechanism and the effect of a reconfiguration on performance using a team running Machinetta proxies in a disaster simulation in which fire trucks (each represented by a proxy) fight fires in a city. In our scenario, no fires were present at the start of the simulation, but ignited randomly as the simulation progressed. In addition, fires spread to neighboring areas unless extinguished. When a fire truck sensed a fire, it formed a plan to extinguish it. However, a fire truck could only accept a role in a plan to extinguish a fire if its distance to the fire was less than a threshold parameter. Otherwise, it would pass the information to a teammate.

In our experiment, 300 Machinetta proxies were distributed over five networked computers running Linux. All communication between proxies was carried out over the network. We simulated a city of size 200 by 200, where fire trucks had an initial threshold of 200 (i.e., a fire truck had to be within 200 blocks of a fire, as measured by Manhattan distance, to accept a role to extinguish it). Each run of the simulation ran for 200 steps. Eight independent runs in which a reconfigure command was issued by the user interface around step 90, and 7 control runs in which no such command was issued were collected. The reconfiguration called for thresholds to be reduced to 10. The averaged number of fires extinguished are shown in figure 4.3.

The teams in control runs extinguished more fires than

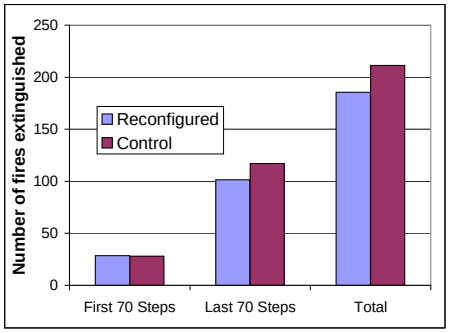


Figure 8. Number of fires extinguished before and after the threshold was lowered, and in total.

the teams in reconfigured runs. This was expected, because the lowered threshold reduced the likelihood that an fire truck would accept a role of extinguishing a fire. It is particularly instructive to see when control teams put out more fires. During the first 70 steps of the simulation, before the reconfigure command was issued, the teams in both cases extinguished nearly equal numbers of fires. However, during the last 70 steps of the simulation, after the reconfigure command had been issued, the reconfigured team extinguished fewer fires. This illustrates the effectiveness of our reconfiguration mechanism for Machinetta and the ability of configuration parameters to affect performance in more realistic team settings.

5. Conclusions and Future Work

In this paper, we have presented a preliminary approach to building a tool that makes it possible for users to quickly explore relationships between configuration parameters of coordination algorithms and performance measures for large teams. Initial experiments showed that the approach was able to aid users configuring and controling a large team executing complicated coordination teamwork algorithms.

While this preliminary work suggests that the parameters of team work algorithms may be effectively controlled to manage team performance in much the same way that individual control parameters have been used to manage robotic behaviors much research remains to be done. One question is how to obtain greater precision in presenting the complex relationships within such large dynamic models. Another issue is how well results or even modeling approaches may generalize between domains. While we have qualitatively demonstrated that our methods 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. Finally, the extension of this work to supervisory control of large teams will require human experimentation and integration of configuration control with other forms of control such as mission planning, redirection, and goal manipulation available for controlling teams.

References

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

[2] 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 *Proceeding of 1999 the Eighth European Workshop in Learning Robot (EWLR-8)*, pages 44–60, 1999.

[3] 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.

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

[5] 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, pages 739–743, Tokyo, October 1999.

[6] D. Kortenkamp, D. Keirn-Schreckenghost, and R. P. Bonasso. Adjustable control autonomy for manned space flight. In *IEEE Aerospace Conference*, 2000.

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

[8] J. Polvichai and 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*, volume 1, pages 978–983, 2002.

[9] J. Polvichai and P. Khosla. Applying dynamic networks and staged evolution for soccer robots. In *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and System*, volume 3, pages 3016–3021, 2003.

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

[11] P. Riley and M. Veloso. An overview of coaching with limitations. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1110–1111, 2003.

[12] P. Scerri, D. V. Pynadath, L. Johnson, P. Rosenbloom, N. Schurr, M. Si, and M. Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *The Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 2003.

[13] P. Scerri, Y. Xu, E. Liao, J. Lai, and K. Sycara. Scaling teamwork to very large teams. In *Proceedings of AAMAS'04*, 2004.