An experiment management framework for TAC SCM agent evaluation

John Collins
Dept of CSE
University of Minnesota
Minneapolis, MN

Wolfgang Ketter
Rotterdam Sch. of Mgmt.
RSM Erasmus University
Rotterdam, NL

Anuraag Pakanati
Dept of CSE
Michigan State University
East Lansing, MI

Abstract

We describe a web-based system for defining and running TAC SCM experiments, and collecting results. It is a research tool, intended to support experimentation and evaluation of alternate agent configurations. With it, we are able to set up sets of experiments, each of which runs multiple games with a fixed set of competing agents. Game and agent logs are gathered together for later analysis. Experiment sets can be queued, and given enough hardware, multiple games can be run in parallel.

Introduction

Participants in the Trading Agent Competition for Supply Chain Management (TAC SCM) (Collins *et al.* 2005) and other trading agent scenarios are practicing a style of research called "competitive benchmarking." These are challenging domains for autonomous agents, and also challenging domains for research, because they require multiple kinds of rational decision-making, and because there is no *absolute* performance standard by which to judge alternative approaches. Therefore, we create communities around specific problems, like the TAC SCM scenario, as well as TAC Travel (Wellman, Greenwald, & Stone 2007), TAC Ad Auctions (Jordan *et al.* 2009), and other scenarios. Each team builds and enters their own agent, and the competition provides a *relative* performance standard by which to judge different approaches to the various decision problems that agents must solve in order to participate.

Unfortunately, the organized competitions typically do not provide nearly enough data for serious study. Most published papers include data from hundreds to thousands of games. This is necessary for several reasons. First, the TAC SCM game scenario includes multiple sources of variability. Customer demand and supplier capacity can vary over wide ranges, and the relatively small number of agents competing in the scenario creates an "oligopoly" market situation, which makes supply and demand, and therefore prices, quite sensitive to the detailed behaviors of the agents. This variability can make two successive games quite different from each other, and so a large number of games are necessary to evaluate performance over the full range of possible market conditions. Second, a tournament round in the TAC SCM competition typically includes only 18 games, which is not enough to separate subtle differences in performance with any degree of statistical confidence. For example, Jordan *et al.* ran over 12,000 simulations in their study of relative agent performance (Jordan, Kiekintveld, & Wellman 2007). In addition, the development of a viable competition agent typically involves considerable experimentation to determine which of several alternate strategies will give the best performance against a variety of competing agents.

The high degree of variability among successive TAC SCM games forces experimenters to run large numbers of games in order to show statistically significant performance differences. Sodomka *et al.* (2007) showed that by controlling the random number sequences in the simulation server, one could run multiple games with essentially the same market conditions. This allows statistical analysis using a paired-means t-test, which can reduce the number of games required to show statistically significant differences by a large margin. However, the need to run different configurations with the same sets of initial conditions adds significant complexity to the process of running experiments.

A further complication one encounters in running experiments in the TAC SCM domain is that many agents are largely compute-bound for much of the game. This means that running a game with a server and six agents may require seven different machines, for nearly an hour per game. The result is that an experiment with 30 games ties up seven machines for more than 24 hours. Only the most dedicated graduate students are willing to hang around and make sure all the agents are starting correctly in each game for days at a time. The remainder of this paper describes a system developed by the MinneTAC team to solve this problem. We describe the concepts behind the MinneTAC Experiment Manager in the next section. Then we follow with some important details of the design, and some typical results. We conclude with a discussion of the some of the difficulties we have encountered and ideas for improvement, and speculate on the possibility of making this tool publicly available.

Related work

We follow a design-oriented approach, as laid out by (Hevner *et al.* 2004). With the design and implementation of the experiment manager we have create a valid artifact, which is relevant and necessary to solve existing problems and test thoroughly systems in the IS domain.

To our knowledge the Michigan AuctionBot (Wurman, Wellman, & Walsh 1998) is the only other auction platform in the community that allows for the systematic testing of software agents. AuctionBot was used as the underlying platform for the design and implementation of TAC Travel (now called TAC Classic) (Stone & Greenwald 2005). AuctionBot was mainly designed to support research into auctions and mechanism design. Our architecture is designed to support research into agents, auctions, business networks, ontologies, and human-agent interaction.

(Corkan & Lindsey 1992) developed experiment manager software for an automated chemistry workstation, including a scheduler for parallel experimentation. Another interesting agent testing environment is James- A Java Based agent modeling environment for simulation has been developed to support the compositional construction of test beds for multi-agent systems and their execution in distributed environments (Uhrmacher & Kullick 2000).

Conceptual model

Many experimental programs involve comparing multiple configurations of a single agent (in our case different configurations of MinneTAC (Collins *et al.* 2008; Collins, Ketter, & Gini 2009)) against a fixed set of competitors, with enough games for each configuration to discover statistically significant effects. The MinneTAC Experiment Manager is designed to support exactly this type of experimental program.

The system is based on a few basic concepts, as shown schematically in Figure 1. At the top level is a work queue whose entries are Experiment Sets. Each Experiment Set consists of one or more Experiments, one for each configuration that is being compared. Experiments are made up of Games. Normally, each Experiment encapsulates the same number of games. The sequence of games in each Experiment are initialized with a common sequence of random-number seeds, so that game $\#k$ in each Experiment has identical market conditions (modulo variations caused by randomness in the competing agents).

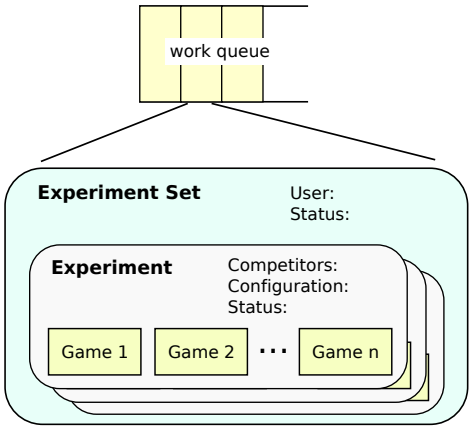

Figure 1: Basic conceptual elements: Experiment Sets, Experiments, and Games

The Experiment Manager is controlled through a web interface. Figure 2 shows the screen for setting up a new experiment.

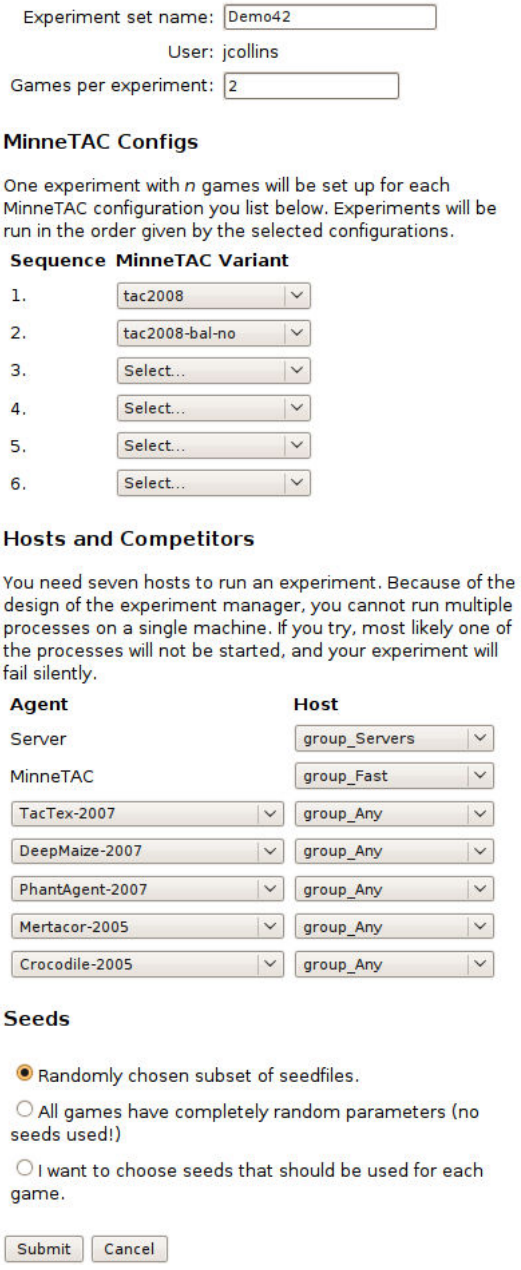

Figure 2: Creating a new Experiment Set

Design details

The Experiment Manager is built in three pieces, a web-based front end, a database, and a back end that actually runs the servers and agents. Running a game requires up to 7 machines. If there is enough hardware available, the system can run multiple games simultaneously. The system is set up as shown in Figure 3. The general scheme is that the

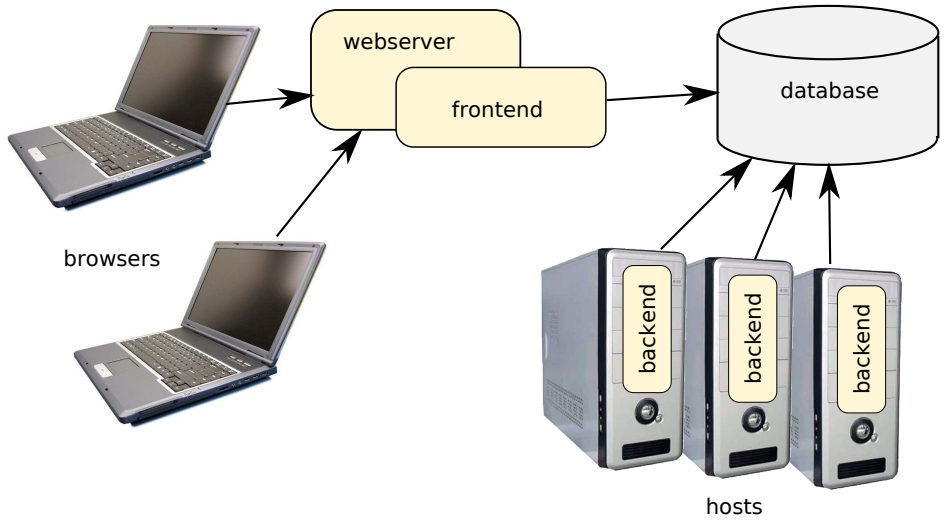

Figure 3: Deployment architecture of the MinneTAC Experiment Manager

front end adds work items to a queue in the database, and the back end runs independently on each available machine, looking in the database for work to do.

The front end is a straightforward web application that allows users to inspect the work queue, add new experiment sets to the queue, modify existing sets that have not yet been completed, and inspect those that have been completed. Other functions allow users to set up and manage competitor agents, set up and manage MinneTAC variants, and determine which host machines are available for work.

The database contains the work queue and status of in-process experiment sets, along with the status of each available host, agent configurations, and records of completed experiments. Since the system is intended to support multiple users, some of these data sets (agent configurations and records of completed experiment sets) are partitioned by user. This is important because each experimenter needs to be able to control their own configurations and interact with their own experiments.

The back end consists of two types of "cron" jobs, one that manages games, and one on each host that manages its host. Every 5 minutes, the game manager checks to see whether there are games that need to be run, and hosts available to run them. If so, it assigns resources and makes a game active. Each host manager wakes up every two minutes, and checks to see whether its host is currently involved in a game (running a server or an agent for a game that has started and not yet completed). If not, the host manager looks to see whether it has been assigned new work by the game manager. If there is new work, then the host manager starts up the necessary process and updates its status in the database.

This high-level description implies that once hosts have work to do, it is just a matter of starting up the necessary processes. This simple view ignores three important problems:

- Many of the hosts we use are general-purpose desktop workstations, and we occasionally experience failures due to user activity on a host. This means that if a game fails to complete, it may be necessary to re-start the game using a different set of hosts.
- There is a correct order for starting processes. For each game, the server must be running before agents can be started.
- The server and many of the agents generate large quantities of log data. Our computing environment uses the Network File System (NFS) for shared directories, and we experience serious contention problems when 18 agents running on 18 different machines are all trying to write voluminous log entries every 15 seconds.

The problem of starting processes in order is easily solved by introducing multiple states to the host management process. The contention problem is much more difficult. The solution we have arrived at is to run each agent from a host-local filesystem, and then copy the logs back to the shared area when a game completes. But this is not quite enough. There is no fixed association between hosts and agents, and the host-local filesystems are not backed up. Also, agents can change, so each game must be sure to use the current "master" copy of each of its agents. This means that agents must be copied to the local filesystems prior to each game, which creates more contention for the shared filesystem. The result is some uncertainty about the time required to set up the environment for a game, which complicates the host managers and creates a requirement to communicate among them, because once the first agent joins a game, all other agents must join within a limited time (typically about 90 seconds) in order to have every agent ready to start on the first day.

Sample results

There are two ways to evaluate the power of our approach. The first is off-line, running the model on saved data and

comparing its estimates and predictions to the actual data. The second is online, integrating the model into a working agent, and letting the agent play against other competitive agents. Here we describe a set of online experiments.

In this set of experiments we compared two different configurations of the MinneTAC agent. We concentrated on the sales side and compared different agent configurations for determining the probability of customer offer acceptance, and the prediction of future prices.

The agents used in our experiments were obtained from the TAC SCM agent repository[1]. In addition to MinneTAC, we selected four other finalists from the 2006 competition, and an agent from the 2005 competition. The agents are:

1. MinneTAC – University of Minnesota
2. DeepMaize – University of Michigan
3. Maxon – Xonar, Inc.
4. PhantAgent – Politechnica University of Bucharest
5. RationalSCM – Australian National University; competed in 2005.
6. TacTex – University of Texas; winner TAC SCM 2006

For our experiments we use a controlled server (Sodomka, Collins, & Gini 2007) to run N_G games, each with a different pseudo-random sequence, with MinneTAC and the five other agents, and then run N_G games with the *same market factors* (the same set of N_G pseudo-random sequences) with a modified MinneTAC and the same set of competing agents. In other words, all the pseudo-random sequences, as well as the set of agents competing with our test agent, from the first set of N_G games are repeated in the second set of N_G games. For our tests, $N_G = 23$. This method removes the profit variability due to the agents seeing different market conditions, and at the same time it removes the possibility of being hindered by unwanted interactions that can occur when multiple copies of agents under test are run against each other.

We use two different versions of our MinneTAC agent, each using different models for for strategic decisions (price and price trend prediction) and for tactical decisions (order probability calculation).

For strategic decisions we used two different price prediction methods. The first is a price-follower method (an exponential smoother predicts future prices, without using a regime model), while the second uses regimes with Markov prediction as described in (Ketter *et al.* 2007; 2009) (called "Regime-M" in Table 1).

For tactical decisions we used two methods to calculate the order probability. The first is a simple linear interpolation between the smoothed minimum and maximum prices, the second uses the regime model and makes predictions using the exponential smoother (called "Regime-E" in Table 1).

We used the Wilcoxon signed rank test (Gibbons 1986; Hollander & Wolfe 2000) to assess statistical significance among these three experiments. This is a *non-parametric* test of the difference between the medians of two samples

[1] http://www.sics.se/tac/showagents.php

Experiment	1	2
Strategic:	Follower	Regime-M
Tactical:	Linear	Regime-E
Agent	Mean Profit/Std. Dev. (in $M)	
TacTex	8.75/5.68	9.21/5.39
DeepMaize	8.84/4.63	8.32/4.18
PhantAgent	8.05/5.42	8.17/5.44
Maxon	4.24/4.52	4.02/4.18
MinneTAC	1.35/3.70	2.12/3.76
Rational	0.74/4.91	1.31/4.53

Table 1: Experimental results with repeated market conditions and two variations of MinneTAC for order probability, price and price trend predictions. Mean profit and standard deviation results are based on 23 games. Regime-M uses the regime model with Markov prediction process, and Regime-E uses the regime model with exponential smoother lookup process.

Test #	1: Exp 2 - Exp 1		
$\alpha = 0.05$	p	h	$srank$
All	0.0138	1	57
Positive	0.0054	1	13
Negative	0.4258	0	15

Table 2: Wilcoxon signed rank test of equality of medians. The tests were performed at a significance level of $\alpha = 0.05$ based on 23 data points. p represents the p-value, h is the result of the hypothesis test, $srank$ gives the value of the signed rank statistic.

that does not require the samples to come from normal (or even the same) distribution. The test is used to determine whether the median of a symmetric population is 0. First, the data are ranked without regard to sign. Second, the signs of the original observations are attached to their corresponding ranks. Finally, the one sample z statistic (mean / standard error of the mean) is calculated from the signed ranks.

Table 2 shows the results of the Wilcoxon test on our three experimental setups. The null hypothesis can be rejected if the medians from the two different samples are different. p is the probability of observing a result equally or more extreme than the one using the data (from both samples) if the null hypothesis is true. If p is near zero, this casts doubt on the null hypothesis. The field "srank" contains the value of the signed rank statistic.

We performed the tests on (1) the set of all the games, (2) only the positive profit games, and (3) only the negative profit games. We find significant differences between the outcome of experiments (2) and (1). We are able to reject the null hypothesis of equal median for the set of all games and the set of all positive games, but not for the set of negative games. The most likely reason why we are not able to reject the null hypothesis of equal median for the set of negative games is that in negative games an agent is more concerned with controlling cost than making profit and so the differences between the configuration are less apparent.

We are also able to show significance between experiments (2) and (1) for the set of all and all positive games. We are able to reject the null hypothesis for the set of all games at $\alpha = 0.05$ significance level. It is possible that the test would show significance with a larger sample size. We have shown statistical significance between the original configuration and the regime/regime configuration.

Conclusion and future work

Autonomous agents participating in market simulations must typically make their decisions in environments containing high complexity, high variability, and incomplete information. While such complexity accurately represents the problems real businesses face on a daily basis, researchers can take advantage of the fact they are working in a simulated environment to systematically control factors that would otherwise add noise to the model output.

We have created a tool which allows researchers to methodically control market conditions in the TAC SCM environment. We have used this tool to measure the amount of variability caused by uncontrollable stochastic agents, and have demonstrated how researchers can determine which variables are most important in affecting the specified output.

We are planning to implement analyses routines which are called upon completion of an experiment set to simplify and automate the data analyses, i.e. parse agent log and game server files and store results in Matlab data files.

Currently the experiment manager has a PHP/JavaScript frontend and a Python backend communicating through a MySQL database. We are currently working on to replace the PHP/JavaScript frontend with a Python implementation so that we only have to maintain one code base with a good object-oriented design. Through this refactoring the code should be much more modular and understandable.

After these efforts are done we are planning to release the experiment manager as a testing tool to the TAC SCM community with full documentation. The will enable all the teams in the community to thoroughly test their agents under the same conditions and make much stronger claims about their agents and experimental results.

Acknowledgments

We would like to acknowledge the help from TAC SCM researchers who made this work possible by making their agents available for others to use.

References

Collins, J.; Arunachalam, R.; Sadeh, N.; Ericsson, J.; Finne, N.; and Janson, S. 2005. The supply chain management game for the 2006 trading agent competition. Technical Report CMU-ISRI-05-132, Carnegie Mellon University, Pittsburgh, PA.

Collins, J.; Ketter, W.; Gini, M.; and Agovic, A. 2008. Software architecture of the MinneTAC supply-chain trading agent. Technical Report 08-031, University of Minnesota, Department of Computer Science and Engineering, Minneapolis, MN.

Collins, J.; Ketter, W.; and Gini, M. 2009. Flexible decision control in an autonomous trading agent. *Electronic Research Commerce and Applications* in publication.

Corkan, L., and Lindsey, J. 1992. Experiment manager software for an automated chemistry workstation, including a scheduler for parallel experimentation. *Chemometrics and intelligent laboratory systems* 17(1):47–74.

Gibbons, J. D. 1986. Nonparametric statistical inference. *Technometrics* 28(3):275.

Hevner, A.; March, S.; Park, J.; and Ram, S. 2004. Design science in information systems research. *Management Information Systems Quarterly* 28(1):75–106.

Hollander, M., and Wolfe, D. A. 2000. Nonparametric statistical methods. *Journal of the American Statistical Association* 95(449):333.

Jordan, P. R.; Cassell, B.; Callender, L. F.; and Wellman, M. P. 2009. The ad auctions game for the 2009 trading agent competition. Technical report, University of Michigan, Department of Computer Science and Engineering.

Jordan, P. R.; Kiekintveld, C.; and Wellman, M. P. 2007. Empirical game-theoretic analysis of the tac supply chain game. In *Proc. of the Sixth Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, 1188–1195.

Ketter, W.; Collins, J.; Gini, M.; Gupta, A.; and Schrater, P. 2007. A predictive empirical model for pricing and resource allocation decisions. In *Proc. of 9th Int'l Conf. on Electronic Commerce*, 449–458.

Ketter, W.; Collins, J.; Gini, M.; Gupta, A.; and Schrater, P. 2009. Detecting and Forecasting Economic Regimes in Multi-Agent Automated Exchanges. *Decision Support Systems* in publication.

Sodomka, E.; Collins, J.; and Gini, M. 2007. Efficient statistical methods for evaluating trading agent performance. In *Proc. of the Twenty-Second National Conference on Artificial Intelligence*, 770–775.

Stone, P., and Greenwald, A. 2005. The first international trading agent competition: Autonomous bidding agents. *Electronic Commerce Research* 5(1):229–64.

Uhrmacher, A. M., and Kullick, B. G. 2000. "plug and test": software agents in virtual environments. In *WSC '00: Proceedings of the 32nd conference on Winter simulation*, 1722–1729. San Diego, CA, USA: Society for Computer Simulation International.

Wellman, M. P.; Greenwald, A.; and Stone, P. 2007. *Autonomous Bidding Agents*. MIT Press.

Wurman, P.; Wellman, M.; and Walsh, W. 1998. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Proceedings of the second international conference on Autonomous agents*, 301–308. ACM New York, NY, USA.