Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

OPTIMISING DISCRETE EVENT SIMULATION MODELS USING A REINFORCEMENT LEARNING AGENT

Douglas C. Creighton
Saeid Nahavandi

School of Engineering and Technology
Deakin University
Geelong, Victoria 3200, AUSTRALIA

ABSTRACT

A reinforcement learning agent has been developed to determine optimal operating policies in a multi-part serial line. The agent interacts with a discrete event simulation model of a stochastic production facility. This study identifies issues important to the simulation developer who wishes to optimise a complex simulation or develop a robust operating policy. Critical parameters pertinent to 'tuning' an agent quickly and enabling it to rapidly learn the system were investigated.

1 INTRODUCTION

The field of intelligent agents, using the artificial intelligence learning technique, Reinforcement Learning (RL), has significant potential in advancing parameters and policy optimisation techniques. Sutton and Barto (1998) provide excellent background reading in this field. Comprehensive literature surveys of pre 1996 research have been published by Kaelbling et al. (1996) and Mahadevan (1996).

Production systems vary widely in all areas of manufacturing, as the body of control literature will attest to. Consequently it is extremely difficult to develop a RL agent framework that may be rapidly applied to a variety of industrial situations. A second obstacle is the challenge in interfacing an agent with commercial simulation software.

In this research a Matlab toolbox has been developed to allow an RL agent to be rapidly 'tuned' to optimise a system. The agent may either reside within the Matlab workspace or the simulation software itself. Critical agent operating parameters, that are modeller assigned, have been identified and tools developed to monitor the performance of the agent and compare the effects of different agent parameter settings. The impact of changes to parameter values can be analysed, allowing rapid determination of preferred agent parameters.

The effectiveness of Discrete Event Simulation (DES) model optimisation algorithms is limited by the complexity of real world simulation environments. DES is a powerful tool to perform "What If" analysis of a system, where the programmer may compare selected configurations. However, the best facility design or operating policy may not have been considered or tested by the engineer. Model optimisation algorithms attempt to overcome this problem by searching the solution space according to a set of guiding rules. A high degree of understanding of the system being studied is often required, restricting the solutions available.

Aydin and Oztemel (2000) have successfully applied RL agents to a dynamic job-shop scheduling problem. The agent was trained using a learning stage by coupling it with a simulated environment. Other agent based work in the job scheduling field has also been completed by Jeong (2000), Zhang and Dietterich (1995), Riedmiller and Riedmiller (1999), and Schneider et al. (1998).

Several research groups have recently focused on RL agent applications in manufacturing. Paternina-Arboleda and Das (2001) uses a SMART algorithm on a serial production line and to optimise the preventative maintenance in a production inventory system (Das and Sarkar 1999). Mahadevan et al (1997) used this same algorithm and touched upon the integration of intelligent agents using RL algorithms with commercial DES packages. Mahadevan and Theocharous (1998) also examined a manufacturing application. Currently these methods have been applied to only a few of the potential areas of optimisation in manufacturing and many problems in the application of the algorithms exist.

2 OVERVIEW OF REINFORCEMENT LEARNING

Reinforcement learning is a simulation based optimisation technique. An agent receives information about the state of its environment and selects an action. The action causes the state of the environment to change. The structure of the agent is illustrated in Figure 1.

At predefined decision points, or events, the model passes control to the agent and waits for input before con-

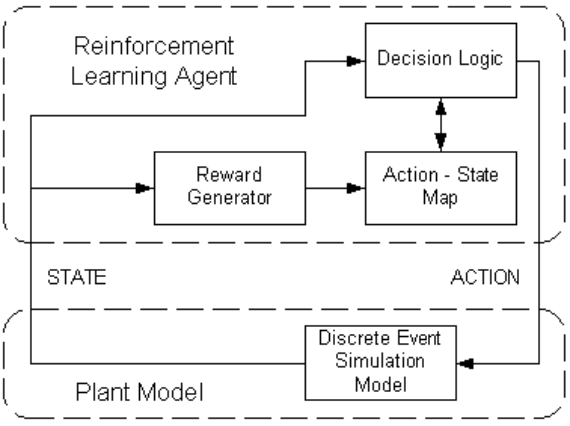

Figure 1: Reinforcement Learning Agent Structure

tinuing. The model outputs its current state and the running cost of the last production epoch, the time period between events requiring an agent decision. The agent must use this information as a basis of future decisions.

The agent uses a state-action map to associate actions with their affects on the model. The focus of the RL community has been on the use of neural nets to store the accumulated knowledge of the agent. Neural net techniques are potentially more efficient then tables for efficient data storage, providing both a generalization technique and compact structure, however they, in particular multi-layer perceptrons, are comparatively time consuming to train (Harp and Samad 2000). Consequently a state space aggregation technique was used in this study.

A reward is maximised, or a cost minimised, depending on the goal function of the agent. The serial-line plant model exported a cost function based on inventory storage cost, die set up costs, production costs and downstream shortage costs to measure the relative advantage of a decision. Many researchers have considered the best method to attribute reward for a selected action. These methodologies may be broadly broken into average and discount rewards (Sutton and Barto 1998).

The decision logic module must select the most appropriate action for the model to take. During the learning phase its goal is to the explore the model response to actions and populate the action - state map to maximise the long term reward of the agent.

3 THE PRODUCTION FACILITY

A stochastic serial line production facility was simulated, with breakdowns that could be repaired on or off line. An example of such as facility is an automotive stamping line, operating under an abort-resume production policy.

Simple production lines have been modelled as Markov processes (Ashkin and Standridge 1993), however the necessity of a state probability transition matrix to predict the next state distribution, limits the application of dynamic programming techniques. Large, real life problems rapidly become computationally intractable.

A system state defines the status of the system. In the facility studied the system states are defined as the inventory levels of all parts in the system, adjusted for depletion rates and set up times.

4 SOFTWARE IMPLEMENTATION

The initial serial-line plant model was implemented in Matlab, as a series of inventory level functions. To develop and test the agent strategy a set of mathematical rules and probability functions were used to calculate the cost of a particular production run with stochastic breakdowns.

A reinforcement learning agent is not required to optimise such a problem which is modelled by a series of mathematical functions, as the probability transition matrices can be estimated, either mathematically or empirically (Campbell et al. 2001). The power of RL techniques is in their application to SMDP problems. Consequently once the methods were evaluated using this test bed, the agent and toolbox was linked to a commercial discrete event simulation package. To enable communication between Matlab and the simulation package, Quest, a Visual Basic server was implemented. Figure 2 illustrates the system architecture of the Reinforcement Agent Simulation Environment (RASE).

The model was generated using Batch Control Language (BCL) commands at run time, to allow flexibility in the models when evaluating the agent's performance. Once the Visual Basic server had established a link with Quest and generated the model, it opened a server to listen for a socket connection from the executing Quest model. It also launched the agent in the Matlab workspace and communicated using a DDE connection.

5 TUNING THE AGENT

By analysing the internal components of an agent individually, it is possible to tune the agent parameters to improve performance. Comparing actual agent progress against the expected behaviour resulting from a parameter set allows identification of problems or limitations that the agent is experiencing. A heuristic technique is more effective then guessing changes to the parameters and decision logics based on the agent's action-state map.

Components important in tuning the agent include the "Reward Generator," "Decision Module Logic," "Boundary Guides," "Model Action Set," and state space aggregation factors such as the state space mapping technique method and parameters.

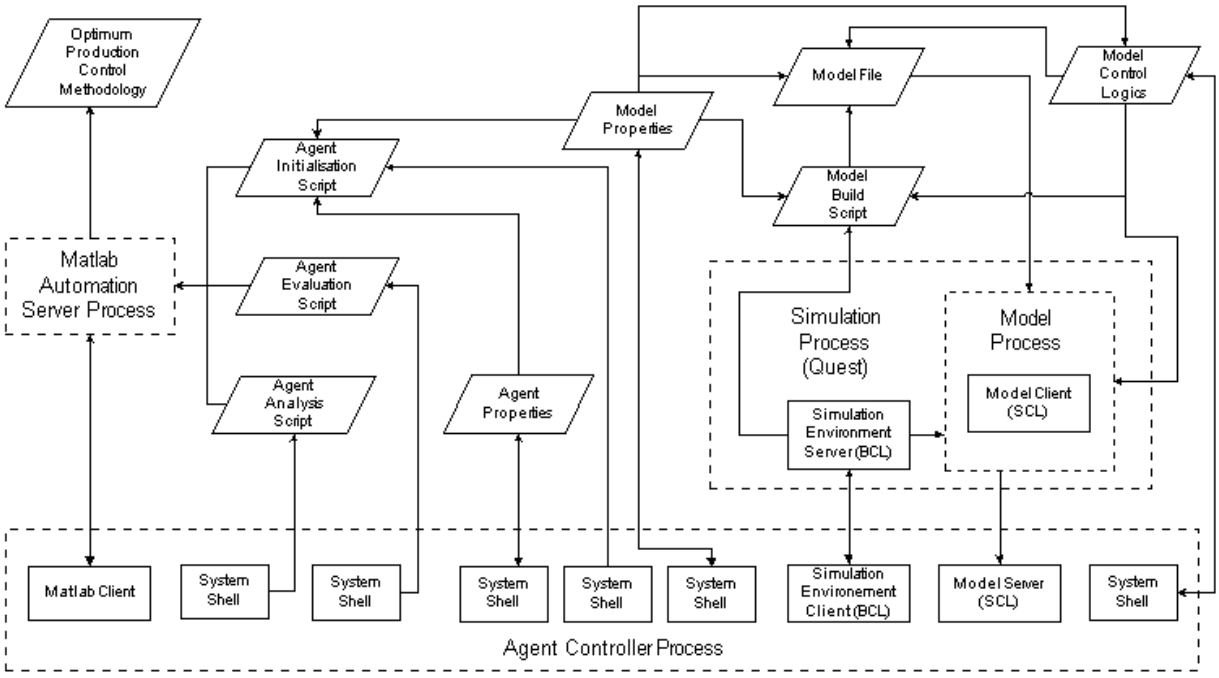

Figure 2: RASE Architecture

Although the best measure of success is the long term reward or operating cost achieved by the agent, it is informative to observe the effects of individual actions or mapping techniques on the agents performance. During the tuning phase of the agent a complete set of data, specify all information received and exported from the agent is recorded. The agent's speed is reduced, but this is a necessary step to tune the agent to effectively interact with complex systems.

Most of the analysis functions in the Matlab RL agent analysis toolbox use colour to differentiate between outcomes of different actions. This is useful to allow the user to observe the affect of specific actions in an action set.

5.1 Agent Reward Generator

The key issue in reward generator design is termed the "Temporal Credit Assignment Problem." This involves the assignment of responsibility of the current state of the system on previous actions taken by the agent.

The reward generator utilises a table to store previous run data. After each iteration all data is shifted up one position and the latest run information in added to the end of the list. Once the data reaches the top of the table the average cost of the action is calculated using all run data listed in the table and some weighting function. The length of the table and the weight function may be customised depending on the system.

A tool was developed to guide the user by predicting the effect of such customisations. The state space exploration pattern is influenced by different reward function weightings. The user identifies better possible options and then needs to test the agent performance on the model.

The costs of producing a large batch size in a system state with part inventories close to shortage may initially produce a good cost advantage. However the subsequent part shortages and higher long term costs will result in a higher long term cost function. The tool generates a plot to show the expected reward accumulated by alternate reward estimation methods. This allows a visual assessment of the effectiveness of different the methodologies. It shows, for example, how many decision epochs should be considered in a reward calculation.

5.2 Decision Module Logic

Real systems have an inherent tendency to become unstable. Such a situation is analogous to training a small robot to search for an item on the table top. If the robot makes a series of incorrect decisions it will end up on the floor and unable to return to the table and complete its task.

An inexperienced agent may take a series of decisions that put the system in a non-recoverable position. Options to guide the agent's decisions include both an internal structure as a part of the agent's decision logics or an external secondary agent observing the agents operations. Either agent may make decisions based on either the state information of the system, or the previous action sequence selected by the agent.

Current research has highlighted the inability to ensure the model remains bounded when randomly selecting actions during the exploration phase. A series of poor decisions results in the inventories running empty, and the agent is unable to build a stable production schedule. Simple guides to limit the agent from making inappropriate decisions in boundary cases overcame most observed problems. A specialised behaviour based agent, working in collaboration with the RL agent, provided a more robust solution to this problem. The technique enabled the agent to keep the serial-line plant facility within an operating region where recovery from shortages were possible.

One goal of toolbox development was to build a simple set of tools that allow a simulation developer to quickly implement and perform complex analysis of DES models. If each possible region of instability must be identified, and guiding rules provided to the agent, the problem will rapidly become too difficult for complex DES models.

5.3 State-Action Map

The mapping technique should not reduce the reachability of regions of the state space. In the case of non uniform state space mapping, the certain system states may be visited less frequently. The mapping should be structure in a way that maximises the reachability of critical states.

Paternina-Arboleda and Das's (2001) study of a multi-machine serial line made several assumptions to reduce the state space, including defining total WIP, rather then individual buffer levels behind machines, as the system's state. The method generated an operating policy that proved to be better then current heuristics production control policies, including CONWIP, Kanban, EKCS and Basestock. Further optimisation of the policy would require analysis of WIP at each buffer stage, increasing the state space by orders of magnitude.

5.4 Agent Action Selection

A wide range of actions potentially gives the agent an opportunity to reach all parts of the state space. After a simulation run a more refined action set may be selected. A plot of the rewards received in each state, by action, is invaluable in understanding the parameter interactions.

When multiple similar actions, or batch sizes, were tested the agent did not readily select the best action. Instead it selected a stable operating policy utilising a combination of actions. The agent proved effective in finding a 'ball park' batch sizes, but could not differentiate between minor variations in parameter. This was a consequence of the variability in the reward due to the stochastic nature of the system.

With limited alternate actions available the agent selected a single best action. However when a second simulation is run to "zoom in" on this region the agent failed to stabilise the system. A broad range of actions was required during the learning phase to allow it to get the agent out of trouble following a poor decision. Selection of a continuous range of q values simplifies the construction of the agent of the changing of values, but reduces the ability to "zoom in" whilst maintaining an optional large q value to recover system stability.

5.5 Exploration Technique

The primary goal of the agent, and therefore the exploration engine of the decision logic, is to generate an accurate action - state map in order to maximise or minimise a cost function. The secondary goal is to achieve this task efficiently. A common methodology is to reduce the rate of exploration over time. This technique shows limited success in situations where repeated poor decisions result in the system entering an unrecoverable unstable state. Once in such an unrecoverable state all further data collected is irrelevant. The risk of reaching a poor state is reduced by initialising the model into a state away from unstable boundary areas and good selection of the agent's action set.

By reducing the exploration rate periodically, as illustrated in Figure 3, such that the agent uses acquired knowledge to shift the system into a preferred state before commencing exploration again, the agent performance is increased. The optimal period of the cycle must be determined by experimentation.

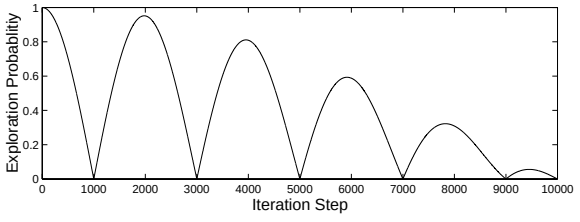

Figure 3: Cooling of Agent Exploration Rate

This technique relies on the agent having collected a minimal amount of knowledge during the early stages of exploration. An advantage is that it will explore the important regions, those most likely to yield optimal solutions of the state space, more than the less important regions.

The length of the total exploration cycle was determined through experimentation. Once the exploration rate is cooled to zero the agent and model may reach a stable interaction, where the agent selects a cyclic series of actions. A stable production system resulting from the agent exploiting its acquired knowledge of the system indicates the agent has developed a possible operating policy. In the system studied the state cycle period was up to 20 states for a 5 part inventory system.

If the system is dynamic an agent needs to be able to track the process evolution to maintain an optimal operating policy. Ongoing low level of exploration throughout the operation may assist if slow changing state-maps do not result in a timely evolution.

It was observed that there are multiple control policies that result in a stable production system, with a range in the number of actions in a cycle, for the facility studied. The overall measure of success was the operating cost during the stable time period. However for agents to be used as a control mechanism the total cost required to reach this equilibrium is important. In systems which may occasionally become unstable or whose control parameters drift slowly requiring ongoing agent learning. There is also further work here to determine an effective learning algorithm to track system drift. That is, does the agent simply maintain a small level of experimentation or use a strict greedy policy which is maintained until an alternative action becomes more cost effective. (the agent must continuously collect data even during its production process in this case.) For long time runs such small changes may be difficult to reflect if using cost averages as a state which has been visited a large number of times will be effected much less by an extreme value then a state which has only be visited occasionally.

Future enhancements to an intelligent agent might allow it to determine the point at which to cease or make constant the level of exploration. Also different 'flavoured' agents that possess some extra knowledge or guide to allow them to have input into the direction of exploration might be implemented. The most effect agent would then be selected from the pool for further tuning.

6 RESULTS

Figures 4 and 5 indicate the possible variation in agent performance as a result of parameter selection. The average cost of actions taken during the agent's learning phase. A large state space results in slow transitions to lower cost operating regions. The exploration-exploitation curve, illustrated in Figure 3, is also observed in the data shape.

Both runs identified a stable operating policy, however the wide band of costs toward the end of the agent's iterations indicates a greater variation in return costs. The erratic high cost returns in Figure 5 are the result of the production facility entering unstable regions of the state space. A good action set and boundary rules resulted in robust agent performance, with recovery to a stable region.

7 CONCLUSIONS

An RL agent effectively identified optimal operating policies of a real production facilities with a large state space. The Matlab RL agent analysis toolbox was successfully utilised to select the agent's action set, decision and state space

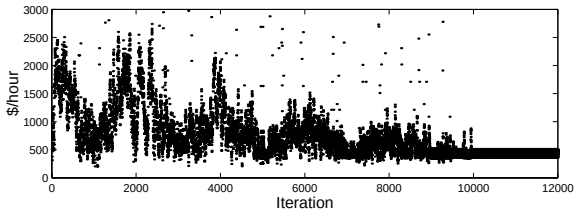

Figure 4: Average Agent Action Cost - Stable States

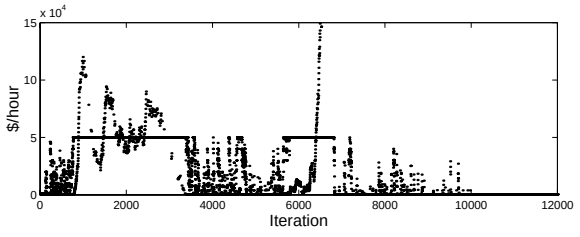

Figure 5: Average Agent Action Cost - Boundary States

abstraction parameters. This reduced the overall time to create and implement an agent. The simulation developer, armed with these tools, can rapidly gain an understanding of the dependancies between agent parameters and tune a RL agent to optimal performance, with minimal knowledge of the model itself.

The RASE architecture allows interfacing between RL agents and commercial simulation products. It has potential to be applied to any programable simulation software, including many commercial packages.

REFERENCES

Ashkin, R. G., and C. R. Standridge. 1993. *Modelling and analysis of Manufacturing Systems* Cambridge, MA: The MIT Press.

Aydin, M. E., and E. Öztemel. 2000. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems* 33: 169–178.

Campbell, P., P. Hodgson, and M. Cardew-Hall. 2001. Optimisation of Production Decisions with Stochastic Process Failures. *Proceedings of the 16th International Conference On Production Research*.

Das, T. K., and S. Sarkar. 1999. Optimal preventitive maintenance in a production inventory system. *IIE Transactions* 31: 537–551.

Harp, S. A., and T. Samad. 2000. Adaptive agents and artificial life: Insights for the power industry. *Soft Computing and Intelligent Systems: Theory and Applications*. eds. Sinha, N. K., and M. G. Madan, San Diego: Academic Press.

Jeong, K. 2000. Conceptual frame for development of optimized simulation-based scheduling systems. *Expert Systems with Applications* 18: 299–306.

Kaelbling, L. P., M. L. Littman, and A W. Moore. 1996. Reinforcement learning: a survey *Journal of Artificial Intelligence Research* 4: 237–285.

Mahadevan, S. 1996. Average reward reinforcement learning: foundations, algorithms, and empirical results. *Machine Learning* 22 (1): 159–195.

Mahadevan, S., N. Marchalleck, T. K. Das, and A. Gosavi. 1997. Self-improving factory simulation using continuous-time average-reward reinforcement learning. *Proceedings of the 14th International Conference on Machine Learning* 202–210.

Mahadevan, S., and G. Theochaurus. 1998. Optimizing production manufacturing using reinforcement learning. *Proceedings of the 11th International FLAIRS Conference* 372–377.

Paternina-Arboleda, C. D., and T. K. Das. 2001. Intelligent dynamic control policies for serial production lines. *IIE Transactions* 33: 65–77.

Reidmiller, S., and M. Reidmiller. 1999. A neural reinforcement learning approach to learn local dispatching policies in production scheduling. *Proceedings of the 16th International Joint Conference on Artificial Intelligence*.

Schneider, J. G., J. A. Boyan, and A.W. Moore. 1998. Value function based production scheduling. *Machine Learning: Proceedings of the Fifteenth International Conference*.

Sutton, R. S., and A. G. Barto. 1998. *Reinforcement learning: an introduction*. Cambridge: MIT Press.

Zhang, W., and T. G. Dietterich. 1995. A reinforcement learning approach to job-shop scheduling. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. 1114–1120.

AUTHOR BIOGRAPHIES

DOUGLAS C. CREIGHTON is a researcher in the School of Engineering and Technology at Deakin University. The industrial focus of his work has been made possible through a collaborative research program between Deakin University and Ford Motor Company, called FAST. His research interests are discrete event simulation, intelligent agent technology, simulation optimisation techniques, and the design and modelling of manufacturing systems. He received a BE (Honours) in Systems Engineering and a BSc in Physics from the Australian National University, where he attended as a National Undergraduate Scholar. Doug Creighton also has several years of engineering and computing experience, working with the Australian Department of Defence and in Federal Parliament House, and more recently as a simulation consultant. His e-mail address is <dcreight@deakin.edu.au>.

SAEID NAHAVANDI received BSc (Hons), MSc and a PhD in Automation from Durham University (UK). In 1991 he joined Massey University (NZ) where he taught and led research in robotics. In 1998 he became an Associate Professor at Deakin University and the leader for the Intelligent Systems research group and also Manager for the Cooperative Research Center for CAST Metals Manufacturing. Dr. Nahavandi has published over 100 reviewed papers and delivered several invited plenary lectures at international conferences. He is the recipient of four international awards, best paper award at the World Automation Congress (USA) and the Young Engineer of the Year award. Dr. Nahavandi is the founder of the World Manufacturing Congress series and the Autonomous Intelligent Systems Congress series. Dr. Nahavandi is a Fellow of IEAust and IEE.