Learning to Envision: An Intelligent Agent for Ship Damage Control

Vadim V. Bulitko
Beckman Institute
Univ. of Illinois at Urbana-Champaign
405 N. Mathews Ave.
Urbana, IL 61801, USA
phone: (217) 398-3149
fax: (217) 244-8371
e-mail: bulitko@uiuc.edu

David C. Wilkins
Beckman Institute
Univ. of Illinois at Urbana-Champaign
405 N. Mathews Ave.
Urbana, IL 61801, USA
phone: (217) 333-2822
fax: (217) 244-8371
e-mail: dcw@uiuc.edu

Abstract

This paper describes an Intelligent Agent for real-time crisis decision making, called Minerva-DCA, which improves its performance by compiling the results of a first-principles simulator. The agent is blackboard based and uses envisionment to schedule its actions. This is necessary because the complexity and chaos associated with ship damage control don't allow the range of necessary behaviors to be easily captured in a ruleset of reasonable size. The first principles envisionment is slow because it involves a simulation of physics of the spread of fires and flooding. In our approach the learning module compiles the knowledge generated by the slow envisionment process and thus allows for real-time performance on subsequent trials. In a large exercise involving 500 ship crises scenarios, Minerva-DCA showed a 76% improvement over Navy officers by saving 89 more ships.

INTRODUCTION

This paper describes an Intelligent Agent for real-time crisis decision making, called Minerva-DCA, which improves its performance by compiling the results of a first-principles simulator. The agent is blackboard based and uses envisionment to schedule its actions. This is necessary because the complexity and chaos associated with ship damage control don't allow the range of necessary behaviors to be easily captured in a ruleset of reasonable size. A typical problem-solving cycle within the blackboard framework [Larsson&Hayes-Roth96, Carver&Lesser94, Park91, Nii89, Bulitko98b] consists of the following three steps: during the deliberation step the knowledge sources are consulted and a set of proposed actions is posted on the blackboard; at the scheduling step the proposed actions are evaluated using some utility-type metrics and the best action is selected; and finally at the execution step the selected action is passed to the actuators/environment and the blackboard is updated to reflected the new state of the environment. The cycle is then repeated [Bulitko98b, Najem93]. This paper focuses on the scheduling step. A detailed treatment of the deliberation stage and its properties relevant to scheduling is given in [Bulitko98b, Bulitko98a].

In this paper we propose a novel model-based scheduling approach. The scheduler represents the environment model as an Extended Petri Net (EPN) model. The reward values come from a static state evaluator that evaluates EPN-predicted states. Previous model-based scheduling approaches include: decision-theoretic dynamic programming [Russell95], qualitative reasoning [Kuipers94], and numerical simulation [Heath96].

There are several strong motivations for the use of Extended Petri Nets and a static board evaluator for blackboard scheduling including:

1) EPNs have good representational and reasoning power for domains with complex causality and time-structure function models. They are capable of handling context (i.e. variables), timing information, and procedural knowledge. The experiments described in this paper suggest that EPNs scale up to practically useful domains such as the ship damage control domain.

2) A variety of previously developed machine learning techniques and corresponding off-the-shelf software packages could be used to synthesize the static board evaluator. We have successfully used artificial neural networks and decision rulesets (C5.0) for this task.

OVERALL DESIGN

At a high level the main idea of our approach is rather simple: predict the state(s) of the environment given the action being evaluated; then evaluate the states using a static state evaluator [Winston84]. The reward values are combined probabilistically to come up with a single utility value for each action. The actions are sorted by that value and the best one is executed. Figure 1 presents a high level operation of the Minerva-DCA expert system that uses this approach. Further details on Minerva-DCA can be found in [Bulitko99, Bulitko98a].

Our approach is model-based [Russell95]. That is, unlike in Q-learning an explicit model of the environment is used. An environment model has the following two main functions: (1) it envisions the future states of the environment given the current state and the action the agent is about to take; and (2) it envisions the reward the agent will get if it takes the action. While in decision theory such a model is often represented as the transition probability and reward functions, in our case the environment is modeled as an Extended Petri Network (EPN). The EPN itself does not contain any reward information. The reward values come from a static state evaluator [Bulitko98a].

The state evaluator has several inputs representing the state of the environment and several outputs representing the severity of the state with regard to the problem-solving tasks (Figure 2). In the test-bed domain of ship damage control the environment is the ship (either real or simulated) from a perspective of the Damage Control Assistant (DCA). A DCA is the officer in charge of handling crises aboard a ship. The DCA's functions include recognizing crises and properly responding to them by giving orders to repair stations and the like. Thus, the input of the state evaluator is a state of the ship: status of compartments and vital equipment with regard to fire, flood, etc. The output is severity of the ship state and is proportional to the time till a predicted major

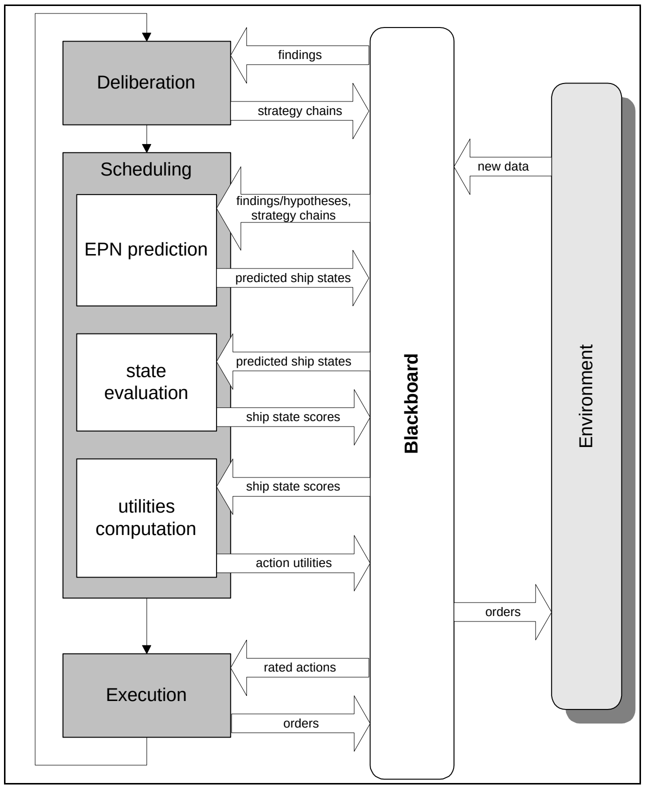

Figure 1. Minerva-DCA Operation: the blackboard framework and problem-solving cycle stages

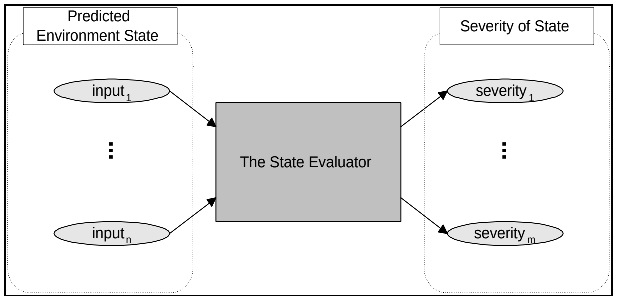

Figure 2. Static state evaluator as a black-box with state attributes as inputs and state severity values as outputs. The predicted environment state is obtained from the environment model (the predictor) while the multi-dimensional severity of state constitutes a score of the state with regard to our decision-making task. A damage control instantiation of this diagram is given later in the paper.

disaster (also known as "time till kill point") [Bulitko98a]. Artificial neural networks [Haykin94] and decision trees [Quinlan86, Quinlan93] are used as the internals of the state evaluator. The subsequent sections provide more detail.

SCHEDULER

This section is central to the paper as it presents us with the scheduler design. The two main components of the scheduler: the EPN envisionment module and the state evaluator will be considered in turn.

EPN Envisionment Module

In this section we will first introduce the Extended Petri Networks and then step through a simple example showing an EPN in action.

Petri Nets is a formalism proposed by C. A. Petri with some early applications to modeling communication protocols and distributed software systems [Murata89]. Petri Nets have traditionally been viewed as being best suited for performance evaluation and communication protocol analysis in the areas of distributed software, database, and communication systems, programmable logic and VLSI design, formal languages, and logic programs [Murata89, Peterson81]. Later on higher-order Petri Nets extended with color and time information have been developed and applied to modeling and analysis of complex real-world systems such as production plant operations [vanderAalst94, vanderAalst93, Jensen92, Merlin74, Merlin76]. Attractive properties of Petri Nets include the solid and well developed mathematical foundation, a number of theoretical and practical analysis methods, and a great variety of supporting software packages. Petri Nets continue to be widely used and there are several periodic conference exclusively devoted to their development [e.g. ICATPN-99]. Unfortunately the classical Petri Nets are inconvenient to use for realistic scheduling due to their propositionality, instantaneous transitions, lack of hierarchies, and uncertainty support. In our approach we propose a set of extensions to the classical Petri Net framework (hence Extended Petri Networks) to make Petri Nets suitable for scheduling in AI systems. A detailed discussion of the proposed extensions is given in [Bulitko98a]. Here an example might serve the best to demonstrate the key extensions of our framework.

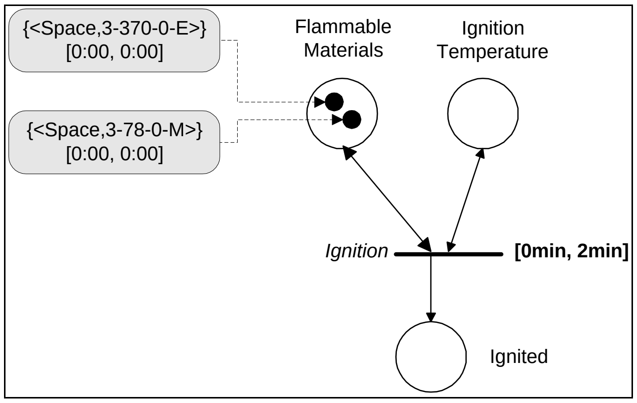

Figure 3. Step 0: the initial configuration

We will start with the an EPN modeling ignition. Figure 3 presents the initial configuration where place "Flammable Materials" is filled with two tokens labeled with compartment (room) names. Both tokens have their time intervals set to [0:00,0:00] meaning that the compartments have been flammable since the beginning of scenario. The other enabling place is labeled "Ignition Temperature" and contains compartments that have reached a sufficiently high temperature. In step 0 that place has no tokens in it meaning that all compartment are cool enough. The two places are enabling places for the "Ignition" transition representing the event of ignition. The transition is labeled with the time delay interval. The output label of the transition is naturally labeled "Ignited" representing a compartment being ignited.

In Step 1 a new token is posted in place "Ignition Temperature". However, the token has a label {<Space, 4-22-0-L>,[2:37,2:45]} indicating that the passageway 4-22-0-L became hot sometime between 2:37 and 2:45 scenario time. This news has no effect on the subnet since the passageway doesn't contain flammable materials (as far as we are aware).

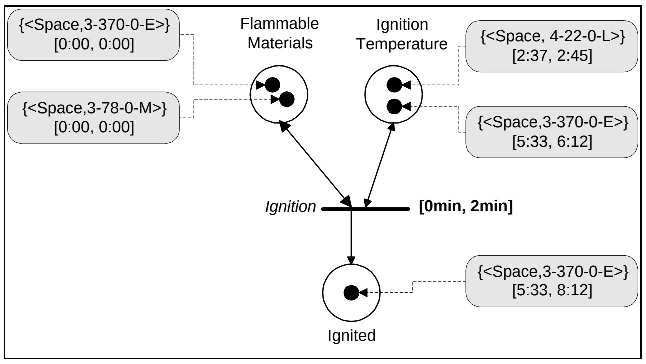
{<Space,3-370-0-E>} [0:00, 0:00]
Flammable Materials
Ignition Temperature
{<Space, 4-22-0-L>} [2:37, 2:45]
{<Space,3-78-0-M>} [0:00, 0:00]
{<Space,3-370-0-E>} [5:33, 6:12]
Ignition
[0min, 2min]
{<Space,3-370-0-E>} [5:33, 8:12]
Ignited

Figure 4. Step 2: generator room gets ignited

In Step 2 the generator room 3-370-0-E becomes hot and a new token is posted in place "Ignition Temperature". This time the space labels match and the transition fires resulting a token in place "Ignited" and reflecting upon the event of generator room ignition (Figure 4).

There are several observations to make: (1) the enabling places retained their tokens as the enabling arcs were double-ended; (2) the time-interval of the new token got affected by the delay interval of the transition.

State Evaluator

The EPN envisionment process can the system with a forecast of what the environment will probably be like in the nearest future. From the scheduling point of view, this information by itself is nearly useless since without evaluating the predicted states we won't be able to know what is good and what is bad, and thus we will have less information to decide what actions to prefer at the moment. In commonly used reinforcement learning terminology we would say that the state evaluator generates reward values while the EPN predictor represents the environment model.

Various classification-type frameworks can serve the task of evaluating a state of the environment. In our experiments we have used artificial neural networks with backpropagation learning, Kohonen maps [Haykin94], and decision rulesets generated by C5.0 [Quinlan93]. In case of the connectionist approach the state of damage control relevant portion of the ship was encoded as a vector of floating point numbers. We have used distributed encoding to represent the time till ship is lost. In case of the decision rulesets all the triggered rules would fire and suggest possibly various values of the state severity (time till kill-point). The one with a maximum confidence factor would be taken as the single output. The next sections provides more details on the scheduler implementation in Minerva-DCA.

DESIGNING A EPN-BASED SCHEDULER

In this section we will go through a possible process of designing a system that uses EPNs for model-based scheduling. An instantiation of this process for the ship damage control domain is presented in the next section together with the experimental results.

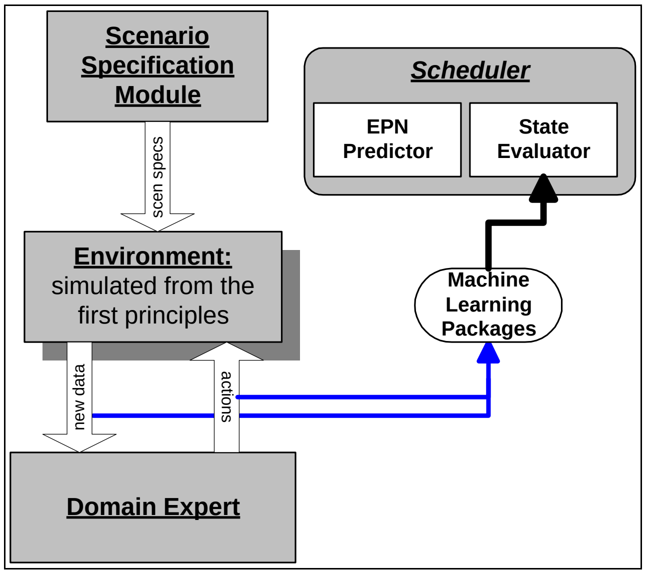
Scenario Specification Module
Scheduler
EPN Predictor
State Evaluator
scen specs
Environment: simulated from the first principles
Machine Learning Packages
new data
actions
Domain Expert

Figure 5. Designing a state evaluator: a number of scenarios are being run with a "teacher" (i.e. a human domain expert or another expert system) and the environment states and the agent's actions are being logged and used to train the static board evaluator.

The first step is to construct an EPN that represents the physical causality and timings in the domain. The places would typically correspond to states of the environment and the transitions would reflect the state change information and timings. The arcs represent the causality information and procedural knowledge (through the operators attached to them). This is currently the most labor-intensive step since it is done by hand eliciting knowledge from a domain expert. Automating this step is an area of our current research.

The second step is to design a state evaluator to work with the EPN predictor. One of the important decisions to make is the choice of state severity scale and state attributes. Once those two are selected one can turn to designing the evaluator's internals. Our design decisions have been to utilize a static state evaluator using artificial neural networks or decision rules as the internals. Fortunately, both formalisms allow for automated synthesis as follows. A number of training traces can be collected (either off actual or simulated system logs as shown in Figure 5. The simulation could be done with some existing, possibly slow, off-line simulator). Each state in the traces is to be represented with the attributes chosen and to be annotated with the severity values as per the severity metric chosen. Finally the annotated traces could be used as the training samples to synthesize the ANN or a decision tree. Commercial packages could be used for this substep.

Once trained the system could be used as described in the previous section and outlined in Figure 1. As demonstrated in the experimental evaluation section, the system trained in this manner can indeed outperform its teacher.

EXPERIMENTAL EVALUATION

Implementation in Minerva-DCA

The EPN framework has been tested as a part of Minerva-DCA blackboard expert system that has been written for the ship damage control domain [Wilkins97]. Below is a description of the EPN implementation within Minerva-DCA while the most detailed treatment is given in [Bulitko98a].

The deliberation module of Minerva-DCA generates a number of feasible actions while solving a crisis. Each action corresponds to a high-level goal (e.g. process hypothesis, findout finding, etc.). Example: action "*fight fire in X*" might correspond to hypothesis "*fire in compartment X*".

Minerva-DCA has a handcoded EPN. Each deliberated action is fed into the scheduler to assess utility of the action. Simply put, *overall utility* of an action is the sum of its *operational* and *goal* utilities. Operational utility is the difference between the value of predicted ship state if the action is taken and the value of predicted ship state of the action is not taken. Intuitively, operational utility reflects how much good an action is going to do us if taken now. Goal utility reflects how important the goal, that the action being evaluated is pursuing, is. It is defined as the difference between the value of the predicted ship state assuming that the goal hypothesis holds and the value of the predicted ship state assuming it doesn't hold. Examples: operational utility of action "*investigate compartment for fire*" is low since it doesn't improve state of the ship, however, it is goal utility is high since a [potential] fire is an important issue. On the other hand, action "*fight fire*" will have both operational and goal utilities high.

We limited the lookahead to 10 minutes. In other words, anything predicted further into the future was ignored. The output of the EPN was fed into a state evaluator. The single top-ranked action was carried out.

The state of the ship was encoded in a single vector of real values with each compartment state encoded as a single value (Figure 6).

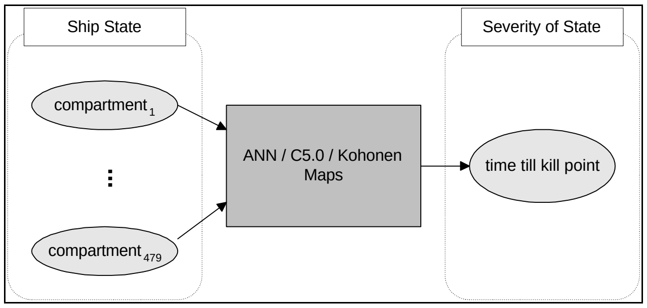

Figure 6. Minerva-DCA State Evaluator with the compartmental state of the ship and the time till ship lost as the severity of the state

Table 1 shows the encoding we used.

Encoding	Compartment State
-0.5	Flooded
0.0	Intact
f([0.1,0.5), t)	High-temp alarm went off *t* minutes ago
f([0.5,1.0),t)	Fire was reported *t* minutes ago
1.0	Destroyed

Table 1. Compartment state encoding. The monotonically increasing function f incorporates the temporal information into the encoding. We used which maps *[0,¥)* to *[a,b)*.

The encoding utilized allowed us to represent both the physical aspect (fire/flood/intact/etc.) and the temporal aspect of the compartment state. The latter is clearly important for assessing severity of a state since there is a significant difference between a compartment just engulfed and the one that has been on fire for over half an hour. Such an encoding resulted in a significant state space size: even if we assume that there are no more than *100* different compartment states (i.e. discretize *[-0.5,1.0]* to *1…100*) we will still end up with $100^{500} = 10^{1000}$ states important for scheduling.

The output (time till ship lost) was represented either in distributed form (i.e. the ANN had 60 output nodes and the final value being the index of the node with the maximum value) or as the right-hand side of the decision rules in case of C5.0. Several rules could fire at once each contributing

```
Rule 1: (cover 4)
    ('4-394-0'. 'JP-5 tank') > 0.2
       ->  class 1  [0.833]

Rule 2: (cover 3)
    ('-1-220-2'. 'Odd station No. 2') > 0
       ->  class 1  [0.800]

Rule 3: (cover 2)
       ('01-188-2'. 'Filter cleaning shop') > 0.7
       ->  class 2  [0.750]

Rule 4: (cover 4)
    ('4-394-0'. 'JP-5 tank') <= 0.2
       ('2-220-0'. 'Access trunk') > 0
       ->  class 2  [0.667]

Rule 5: (cover 2)
    ('4-394-0'. 'JP-5 tank') > 0
       ('2-220-0'. 'Access trunk') <= 0
       ->  class 2  [0.500]

Rule 6: (cover 5)
       ('01-188-2'. 'Filter cleaning shop') > 0.6
       ('01-188-2'. 'Filter cleaning shop') <= 0.7
       ->  class 3  [0.857]
```

Figure 7. A Fragment of Minerva-DCA decision ruleset generated by C5.0. The left-hand sides of the rules have compartments and the encoded status while the right-hand sides specify the time-till kill-point (class) and the confidence level of the conclusion (given in the brackets).

its confidence level to the predicted value. The value with the maximum combined confidence level was taken as the final output.

Following the design process outlined in the previous section we had first handcoded an EPN to model fire and flood related phenomena that take place aboard a DDG-51 class destroyer. We had then used Minerva-4 equipped with a handcoded rule-based scheduler [Bulitko98b] to run damage control scenarios automatically and collect the training data. The training data was used to generate a decision rule board evaluator using C5.0 package. The scenarios were run within the DC-Train ship damage control simulator used at the Surface Warfare Officer School (SWOS) in Newport, R. I.

The three classifier frameworks we have used (ANN, K-maps, and C5.0 rulesets) have show generally comparable performance. However, on the data sets we have had the decision rules were the fastest and exhibited the highest cross validation accuracy [Bulitko98a] as shown in Figure 8.

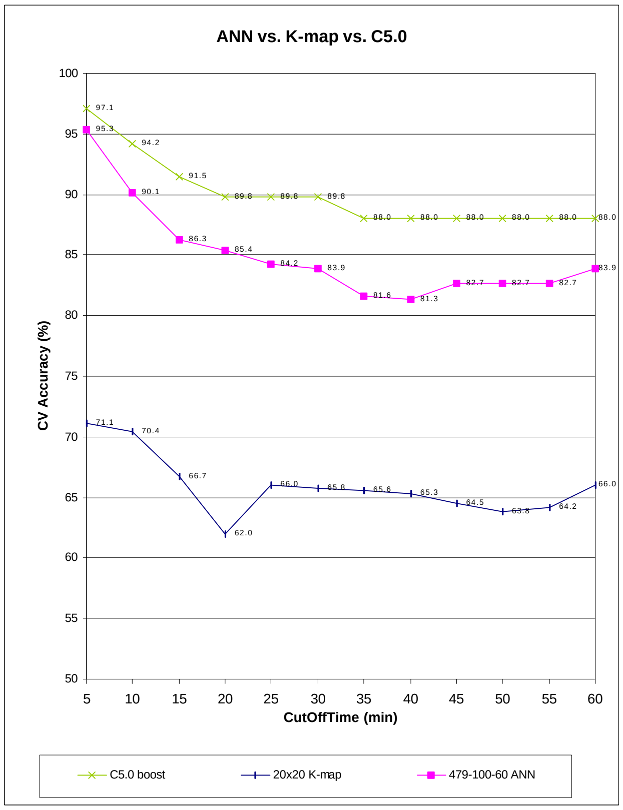

Figure 8. Cross-validation accuracy of the three classifier approaches attempted for the state evaluator. The cut-off time specifies the prediction range. All training exemplars with a kill-point more distant than the cut-off time were considered to have no kill-point at all.

Figure 7 shows a fragment of the decision rule-set we used in Minerva-DCA.

DC-Train 1.0 Runs. To evaluate the performance 500 scenarios have been run within the DC-Train ship damage control simulator routinely we developed that is used at the Navy officers school in Newport, R. I. We have then compared Minerva-DCA using the EPN scheduler to Minerva-4 that had a *handcoded rule-based scheduler* [Bulitko98b] and to Navy officers at SWOS. The results are presented in Figure 9.

Any scenario could have had one of the three outcomes: "ship lost" meaning that a kill point (ship loss) was reached (i.e. a major disaster such as a missile compartment explosion); "ship possibly saved" meaning that at 25 minutes scenario time the ship was still alive yet there were active crises; and "ship saved" means there were no active crises at the 25-minute mark.

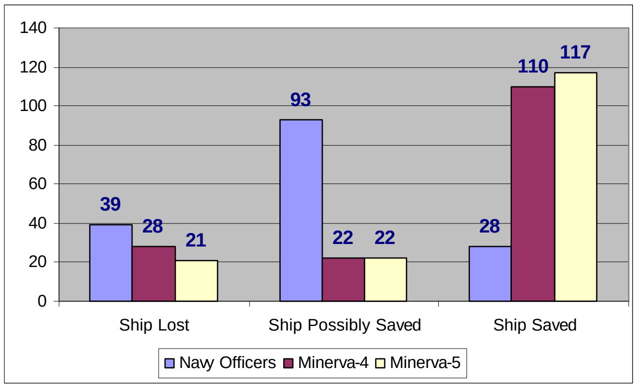

Figure 9. Minerva-DCA vs. Minerva-4 vs. Navy officers experimental comparison

The use of the EPN based scheduler resulted in 21 ships being lost. This is a 25% improvement over the use of a handcoded rule-based scheduler wherein 28 ships were lost. And it is a 46% improvement over Navy officers where 39 ships were lost.

RELATED WORK IN SCHEDULING

Model-free Approaches

Q-learning. In this section we will briefly introduce the main concepts following the notation from [Russell95, Mitchell97, Dietterich98]. In the field of Q-learning and reinforcement learning in general, the learning problem is often described as a Markov Decision Process as follows. The agent/system could be in one of the finite number of states $s_1,\ldots,s_n$. In every state the agent can take one of the *m* actions: $a_1,\ldots,a_m$. The environment is represented via transition probability *P* and reward *R* functions. Transition probability function $P(s'|s,a)$ represents the conditional probability of getting into the state *s'* if the agent takes action *a* being in state *s*. The reward function $R(s'|s,a)$ represents the numerical reward the agent will receive if it takes action *a* in state *s* and gets into state *s'*. The goal of the agent is to learn (choose) a policy *p* that maximizes the value function $V^p(s)=E[R(s'|s, p(s))+gV^p(s')]$ for each state *s*. The optimal (best) policy is typically denoted as $p^*=argmaxV^p(s)$ where *max* is taken over the set of all policies.

Given the notation above, the Q-learning algorithms choose (learn) a [scheduling] policy through a *model-free* Q-function as follows: $p^*(s)=\arg\max_a Q(s,a)$ where the Q-function is defined as: $Q(s,a)=E[R(s'|s,a)+gV^*(s')]$ (here *V** is the value function corresponding to policy *p**; *g* is the decay factor). Different learning methods could be used to learn the Q-function but all of them are model-free in that they don't explicitely refer to the environment's *R* and *P* functions. Most Q-learning systems learn the Q-function by starting with a random policy *p* and updating it iteratively based on the observed rewards. Such approaches include the value iteration, temporal difference learning, hierarchical Q-learning, and others [Russell95, Mitchell97, Dietterich98].

The ability to learn without representing/knowing the environment model doesn't come for free. Q-learing is generally slow even in small domains. Various powerful extensions have been proposed to address this issue, such as hierarchical decomposition as in MAXQ-learning. Unfortunately even state-of-the-art systems still require *many thousands* of policy iterations to converge on a good policy in small artificial domains [Dietterich98].

Recursive Heuristic Classification. The approach of recursive heuristic classification (RHC) [Park93] considers the scheduling task as a classification problem on its own. Thus a dedicated expert system is designed and used to handle this task. In fact it turns out that certain expert system designs make it is possible to use the same expert system for both tasks (hence the name *recursive*). In experiments with Minerva-2 [Park93], the domain instantiation of the expert system shell (called Minerva-MEDICINE) deliberated the actions and then called the scheduling instantiations of itself (called Minerva-SCHED) to schedule the actions. Due to domain-independently expressed strategy knowledge the only extra knowledge layer needed for such setup was the domain knowledge of Minerva-SCHED. Furthermore it has been shown that this scheduling domain knowledge layer could be automatically induced through apprenticeship learning methods [Park93, Donoho93]. While being a great scheduling method for medical diagnosis it has several cons for real-time decision making:

- ♦ the induced scheduling knowledge is model-free and thus domain independent. It is clear to us that in real-time decision making context of an action has a crucial effect on its scheduling time. Thus, we argue that an efficient scheduler has to have a domain model and thus be domain dependent. It is less clear how to induce domain-dependent scheduling knowledge automatically;
- ♦ a significant slow down relatively to a handcoded scheduler has been reported in [Park93].

Rule-based Systems. Several system (including [Bulitko98b]) have exploited rule-based schedulers. While theoretically given a flexible rule format rule-based schedulers could be shown to be as powerful as any other schedulers, there are several complications in practice. Domain-independent rule-based schedulers are subject to the same considerations as RHC systems. Domain-dependent rule-based schedulers can take the context into account, however common rule base problems (conflict resolution mechanism, uncertainty reasoning, rule base creation, rule base revision/debugging/maintenance, rule base visualization, etc.) arise. Overall, rules don't appear to be the best formalism for blackboard scheduling. An interesting comparison between EPN-scheduling and the rule-based scheduling could be found in [Bulitko98a] and summarized in the section on performance evaluation given above.

Model-based Approaches

Three alternative model-based approaches that we considered are dynamic programming, numerical simulation, and quantitative reasoning. We will discuss each of these in turn.

Dynamic Programming. A dynamic programming formulation of a scheduling problem explicitly represents all the states, and can typically manage problems with in the order of 50,000 states [Dietterich98, Mitchell97, Russell95]. This method seems unsuitable for the blackboard scheduling problem associated with the ship damage control decision making task because there are on the order of 10^{1000} states in our current test-bed system (refer to the previous sections for the size analysis).

Numerical simulation methods [Heath96] can explicitly model an environment by simulating it from first-principles. In fact, Minerva-DCA works in conjunction with a numerical simulator for training the decision making system, but it can't be used for scheduling because it is approximately three orders of magnitude too slow.

Qualitative Reasoning methods [Kuipers94] are another major method of explicitly modeling a complex environment. Their major strength is the ability to simulate a domain qualitatively. Extended Petri Nets and the state evaluator framework has been found to be useful because their strong formal foundation permits many types of automation in the problem analysis, such as reachability and deadlock analysis.

DISCUSSION

This paper has presented the formalism of Extended Petri Nets that was created for the Blackboard scheduling domain. An EPN-based scheduler should be able to be used as an environment model in a variety of systems. Combined with a state evaluator it provides both envisioned states and associated rewards. Another major contribution of the paper is the learning framework which helps automate the scheduler synthesis.

The proposed approach has a number of strengths including high prediction speed, variable lookahead time, well-studied formalism of Petri Nets, and automatic induction of the state evaluator. It also opens a number of directions for future research as follows.

FUTURE RESEARCH DIRECTIONS

The state evaluator easily allows for a number of extensions such as enriching the input and outputs of the evaluator and adding costs to actions. In our test-bed domain of ship damage control we are currently extending the state description with equipment states and personnel availability as well as enriching the output with the "time till crisis is over" (which allows for fine-tuned scheduling when no kill-point is likely in the near future).

Another research direction that we are currently exploring is Learning EPNs. Currently the Extended Petri Networks are manually designed. While being a reasonable approach in some domains, handcoding might be too costly in other domains. The consistency might become an issue as well. This branch of research can take advantage of machine learning methods to automatically extract the training data and generate the EPNs based on it.

ACKNOWLEDGEMENTS

Discussion with KBS members and Dr. V.K.Bulitko was of a great help. The research has been supported in part by ONR Grant N00014-95-1-0749, ARL Grant DAAL01-96-2-0003, and NRL Contract N00014-97-C-2061.

REFERENCES

van der Aalst, W.M.P. 1993. Interval Timed Coloured Petri Nets and their Analysis. In M. Ajmone Marsan editor, Application and Theory of Petri Nets, volume 691 of Lecture Notes in Computer Science, pp. 453 - 472. Springer-Verlag, Berlin.

van der Aalst, W.M.P. 1994. Putting Petri nets to work in industry. *Computers in Industry*, 25(1):45--54.

Bulitko, V.; Wilkins. D.C. 1999. Automated Instructor Assistant for Ship Damage Control. In the Proceedings of The Eleventh IAAI Conference. (accepted for publication).

Bulitko, V. 1998a. Minerva-5: A Multifunctional Dynamic Expert System. MS Thesis. Department of Computer Science, University of Illinois at Urbana-Champaign.

Bulitko, V.; Wilkins, D.C. 1998b. Minerva: A Blackboard Expert System for Real-Time Problem-Solving and Critiquing. Tech. Report UIUC-BI-KBS-98-003, University of Illinois at Urbana-Champaign.

Carver, N.; Lesser, V. 1994. The Evolution of Blackboard Control Architectures. In Expert Systems with Applications--Special Issue on the Blackboard Paradigm and Its Application, Volume 7, Number 1, pp. 1-30, Liebowitz. New York, Pergamon Press.

Dietterich, T. 1998. The MAXQ Method for Hierarchical Reinforcement Learning. In Proceedings of ICML'98 conference.

Donoho, S. 1993. Similarity-based learning for recursive heuristic classification. Master’s thesis. University of Illinois at Urbana-Champaign.

Haykin, S. 1994. *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing Company.

Heath, M. 1996. *Scientific Computing: An Introductory Survey*. McGraw Hill Text.

ICATPN-99. *The International Conference on Application and Theory of Petri Nets*. The 20th annual meeting will be held in June 1999.

Jensen, K. 1992. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Volume 1, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag.

Kuipers, B. 1994. *Qualitative Reasoning*. The MIT Press.

Larsson, E.; Hayes-Roth, B.; Gaba, D. 1996. Guardian: Final Evaluation. Knowledge Systems Lab, Stanford University. TechReport KSL-96-25.

Merlin, P. 1974. A Study of Recoverability of Computer Systems. Ph.D. Thesis. University of California, Irvine.

Merlin, P.; Faber, D.J. 1976. Recoverability of Communication Protocols. *IEEE Transactions on Communication*, 24, pp. 1036-1043.

Mitchell, T. 1997. *Machine Learning*. WCB/McGraw-Hill.

Murata, T. 1989. Petri Nets: Properties, Analysis, and Applications, in the proceedings of the IEEE 77, 541-580.

Najem, Z.H. 1993. A Hierarchical Representation of Control Knowledge For A Heuristic Classification Shell. Ph.D. Thesis. Department of Computer Science. University of Illinois at Urbana-Champaign.

Barr, A.; Cohen, P.R.; Feigenbaum, E.A. eds. 1989. *The Handbook of Artificial Intelligence*. Volume IV, Chapter XVI by Nii, H.P. Addison-Wesley.

Quinlan, J.R. 1986. Induction of Decision Trees. *Machine Learning*. 1(1), 81-106.

Quinlan, J.R. 1993. *C4.5: Programs for Machine Learning*, San Mateo, California: Morgan Kaufmann.

Park, Y.T; Tan, K.W.; Wilkins, D.C. 1991. Minerva 3.0: A Knowledge-based Expert System Shell with Declarative Representation and Flexible Control. UIUC, Department of Computer Science.

Peterson, J.L. 1981. *Petri Nets Theory and Modeling of Systems*. Prentice-Hall, Inc.

Russell, S.; Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc.

Wilkins, D.C.; Sniezek, J.A. 1997. An Approach to Automated Situation Awareness for Ship Damage Control. KBS Tech. Report UIUC-BI-KBS-97-012. Beckman Institute. University of Illinois at Urbana-Champaign.

Winston, P.H. 1984. *Artificial Intelligence*. The second edition. Addison-Wesley Publishing Company.