Monitoring Progress with Dynamic Programming Envelopes

Robert St. Amant, Yoshitaka Kuwata*, Paul R. Cohen
Computer Science Dept., LGRC
University of Massachusetts
Amherst, MA 01003-4610
stamant@cs.umass.edu, kuwata@rd.nttdata.jp, cohen@cs.umass.edu

Abstract

Envelopes are a form of decision rule for monitoring plan execution. We describe one type, the DP envelope, that draws its decisions from a look-up table computed off-line by dynamic programming. Based on an abstract model of agent progress, DP envelopes let a developer approach execution monitoring as a problem independent of issues in agent design. We discuss the application of DP envelopes to a small transportation planning simulation, and discuss the issues that arise in an empirical analysis of the results.

1 Introduction

An intelligent agent builds plans to achieve its goals. In a complex or uncertain environment, the agent must alter these plans to meet unforeseen circumstances. Adjusting plans to respond to environmental change is an issue addressed by research in reactive planning. There is currently no widely accepted theory of how to generate reactive plans. One approach is suggested by [Beetz92]: "...Most (reactive) plans will be made up of large canned segments retrieved from a library, pasted together, and debugged." In some circumstances, however, one can apply a more principled approach. We have developed a technique for building decision rules, called *envelopes*, that monitor plan execution [Hart90, Cohen92, Hansen92]. In this paper we discuss the role of envelopes in reactive planning, the advantages and disadvantages of their application, and conclusions we have drawn from experimental evaluation.

An envelope monitors a task to determine if and when steps should be taken to improve progress in the task. When we monitor a time-constrained plan, we have the option of waiting until a deadline has been reached, and checking at that point whether a goal has been met; however, we gain a clear advantage by detecting problems earlier. In their original formulation envelopes provided advance warning of plan failure in the form of a simple continue/abort decision. If we extend the range of the envelope's decision choices, we have a form of reactive plan.

One kind of envelope, called a DP envelope, treats execution monitoring as a sequential decision problem [Hansen92]. Dynamic programming (DP) is used to approximate an optimal solution to the problem of when to modify a plan or warn of impending plan failure. Hansen and Cohen have also applied DP to the combined problem of when to monitor as well as modify a plan. In essence, we build a model of the agent and its progress in the environment. Based on the interaction between

To appear in *Proceedings of the Seventh International IEEE Conference on Tools with Artificial Intelligence.*

environmental costs and the agent's actions, we can calculate off-line the best action to take in any situation. This set of actions is embodied in the envelope.

The general envelope concept is a useful tool for implementors of complex planning systems in the following sense. An envelope provides important information to both planner and human designers interested in measuring planner performance. Envelope design and operation can be independent of planner architecture; changing the planner does not necessarily entail changing the envelopes that monitor its behavior. Finally, envelopes can be added to a system and tuned with a relatively small design effort. Thus envelopes can offer an attractive alternative to the development of a hand-built set of monitoring rules.

Much progress has been made in theoretical areas relevant to DP envelopes [Barto93]. Our work aims toward an empirical evaluation of the benefits and limitations of DP envelopes, to some extent in contrast with a heuristic approach. We attempted not only to design envelopes for different environments, but also to explain and if possible predict their behavior under different conditions. In short, our goal was to develop a detailed understanding of the DP envelope as an approach to reactive planning. We posed three questions:

- How easily can we construct envelopes? We assess the design and implementation effort in constructing envelopes. There are built-in limitations associated with almost all dynamic programming approaches. Through a series of experiments we explore techniques by which these limitations may practically be overcome.
- Can we explain the behavior of envelopes? We must be able to understand envelope behavior, or we run the risk of producing incorrect rules and not realizing that they are flawed. More importantly, from a practical standpoint, we must be able to debug envelopes, which is not possible without expectations and explanations. In the same series of experiments we make predictions about envelope behavior and test them against our findings.
- How well do envelopes perform as reactive plans? It is often difficult to define optimal performance, or even acceptable performance. We describe a method by which we can measure the effectiveness of envelopes.

2 Example Domain

Drawing on a background of work in transportation planning [Kuwata92], we designed a simple model of the shipment of cargo between sea ports. The task is to transfer a fixed amount of cargo from a single source to a single destination. All cargo arrives at the source port on a particular date, and must be delivered within a specific window of time, the endpoint being the deadline. A limited number of ships can be allocated, each carrying one piece of cargo at a time. There are no dock/ship or ship/cargo incompatibilities. The task for the envelope is to determine the best number of ships to use at any time during execution of the task.

Though this model may seem overly simplistic, uncertainty in the environment makes the task challenging. Ships travel at varying rates; cargo is loaded at varying rates. Simulation activities incur daily costs, accumulating costs, and one-time costs. Following a predetermined ship allocation scheme can lead to poor performance. The solution is to make decisions based on a measurement of progress in the task.

We have built a simulation for this model, based on a more complex simulator called TransSim. Our simulation represents individual ships and pieces of cargo moving between two ports. Costs are recorded as follows: ship allocation cost (per ship), cargo storage cost (cumulative, per unit), lateness penalty (finish date vs deadline), earliness penalty (finish date vs deadline), daily shipping

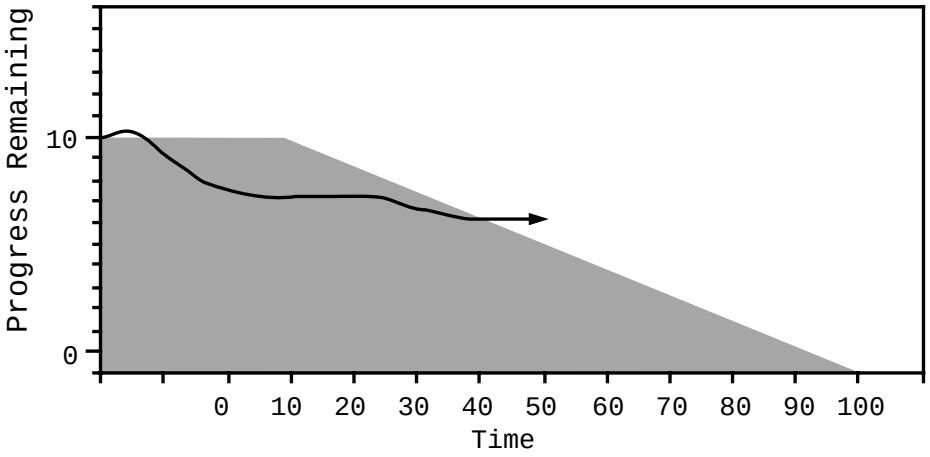

Figure 1: A slack-time envelope

cost (cumulative, per ship in transit), and dock cost (cumulative, per dock in use). All ship and cargo activities proceed autonomously; the key decision required from the envelope is whether to allocate more or fewer ships on each simulated day.

3 Envelope Construction

Envelopes were originally developed in the Phoenix planning testbed as a means of monitoring progress in a task [Cohen89, Hart90]. Imagine a plan that requires an agent to make 10 units of progress (distance toward a goal, for example) in 100 time units. Figure 1 diagrams the progress of the agent toward its goal. The agent begins with 10 units of progress remaining, and must reach 0 before 100 time units elapse. The curved line represents the distance between the agent and its goal at each point in time. The boundary of the shaded region is an envelope, specifically a slack-time envelope, for this task [Cohen92]. When the agent crosses this boundary, the envelope violates, or signals a failure, which typically requires some modification to a plan [Howe93, Howe95]. The slack-time aspect of the envelope refers to an initial period of time during which no failure predictions are made. The agent is thus allowed to fall slightly behind at first, under the assumption that it can make up for lost time later.

The slack-time envelope is based on the determination of a single parameter, the amount of slack, or equivalently the slope of the envelope boundary. The difficult part about generating slack-time envelopes is determining this value. The parameters contributing to this determination are complex, and vary according to the task under consideration. We present, without further explanation, one of the parameters necessary to measure progress in a Phoenix envelope:

$$DigTime = \frac{Perim + SpreadRate(Slack + BDTravel)}{BD\frac{CutRate}{RFactor} - SpreadRate(Ineff + 1)}$$

The point is that calculation is easy—definition of the parameter (and the debugging required for it) is complex. Modification of the plan requires corresponding modifications of the envelope parameters. As the complexity of envelope parameters increases, it becomes difficult to predict the effects of small parameter variations on the performance of the envelope. A more serious difficulty is that we cannot generalize about the applicability of such an envelope to other environments. Because hand-built envelopes are complex, hard to debug, and of unknown generality, we developed an automatic procedure for generating envelopes.

3.1 Dynamic Programming Envelopes

Dynamic programming is a mathematical technique for solving problems that decompose into related subproblems. Simple applications include matrix chain multiplication and longest common subsequence search, i.e., problems that contain identical recurring subproblems. Monitoring can be treated as a sequential decision problem[Hansen92], in that an optimal monitoring policy must take into account the future ramifications of each of its decisions. Dynamic programming is well-suited to such problems.

The basic purpose of a slack-time envelope and a dynamic programming (DP) envelope is the same: to decide whether to take action to improve progress in a task. The difference is that slack-time envelope decisions are based on an arbitrarily complex combination of relevant variables; DP envelope decisions are based on an explicit model. There are three aspects of interest in the DP envelope: the underlying model, how a policy of action is calculated from the model, and how the policy is applied in an environment.

We represent the state of the system with just three variables: cargo remaining (C), ships in use (S), and time remaining until deadline (T): $State = \langle C, S, T \rangle$. Progress is measured by the amount of cargo delivered. Transition from one state to another thus depends on ship arrivals at the destination port. Since individual ships are not modeled explicitly, the ship arrival rate must be approximated. We calculate the transition probabilities by assuming that ship arrivals follow a binomial distribution, with p being the inverse of round trip travel time, and $N = S$, the number of ships. For example, if the travel time is 10 days and 2 ships are in use, then the mean number of ships to arrive on any given day, Np, is 0.2, with variance, $Np(1-p)$, of 0.16. Though a very rough approximation, this turns out to behave well in practice. We can describe our model then as follows:

$$
\begin{array}{lll}
Reachable(\langle C, S, T \rangle) = & \text{nil} & \text{if } C = 0 \\
 & \langle C, S, T-1 \rangle & S = 0 \ldots MaxS \\
 & \langle C-c, S, T-1 \rangle & S = 0 \ldots MaxS
\end{array}
$$

When there is no cargo remaining, there are no states reachable from the current state. With S ships and C units of cargo remaining, the system can move to other states in which the number of ships may change, but no cargo is delivered, or the number of ships may change and c units of cargo are delivered. Legal actions allow the number of ships to range from zero to $MaxS$, a fixed parameter.

$$Legal(\langle C, S, T \rangle) = 0 \ldots MaxS$$

The system is charged a cost for each action it takes, including the 'null' action. In the case in which the number of ships remains unchanged, the cost includes a storage cost for cargo, a shipping and docking cost for active ships, and late or early costs determined by task completion. When the number of ships changes from S to R, an additional cost is assessed for allocation or deallocation of those ships.

$$
\begin{array}{ll}
Cost(\langle C, S, T \rangle, S) = & \text{estimated incremental cargo cost} + \\
 & \text{estimated shipping cost} \\
Cost(\langle C, S, T \rangle, R) = & \text{estimated incremental cargo cost} + \\
 & \text{estimated shipping cost} + \\
 & \text{(de)allocation cost for } R - S \text{ ships}
\end{array}
$$

Finally progress, in terms of the probability of moving from one state to the next, is defined as a binomial function, as described earlier.

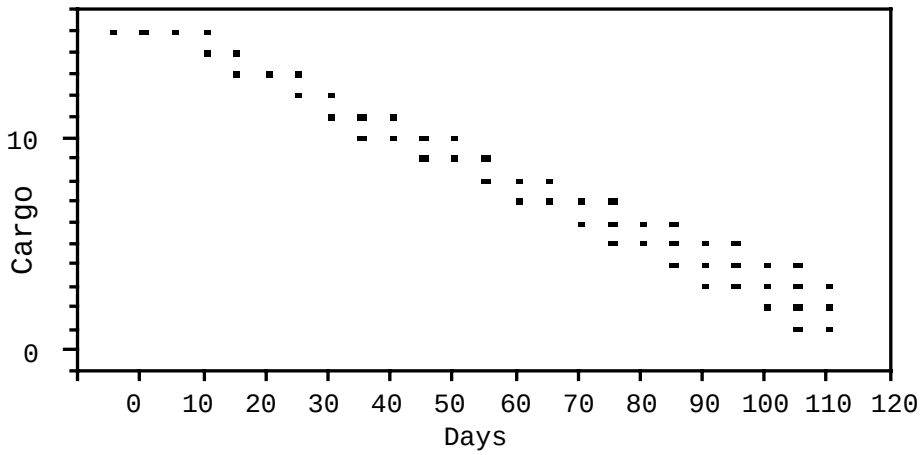

Figure 2: Cross-section of a DP envelope for S = 5 ships

$$Pr(\langle C, S, T\rangle, \langle C - c, S, T - 1\rangle) = BinomialPDF(p, S).$$

Details concerning boundary conditions and cost parameters have been omitted for the sake of presentation. The main points in the model are that on each day cargo is reduced with a particular probability, and the costs incurred depend on the amount of cargo remaining and the number of ships in use.

Given a model, the standard DP approach entails calculating a policy based on the model. For each possible state and action, we compute the expected value of taking that action in that state and store the result in a lookup table. Calculation proceeds from the deadline backward to the start time. At any point during the calculation, all future actions from the current point to the deadline have been calculated. The calculation at each point thus can account for all possible future occurrences, and evaluate decisions based on this knowledge. We can use the completed lookup table to implement a policy under which we take in each state the action with the lowest expected cost.

The lookup table contains as many dimensions as the size of the state representation. Our model, with its three state variables, takes the form of a three-dimensional vector space. For any values of cargo remaining, ships in service, and time remaining, the lookup table provides a value containing the proper number of ships to allocate or deallocate.

We can visualize the three-dimensional structure of a DP envelope through two-dimensional slices. In this form a DP envelope shows a resemblance to the abstract slack-time envelope presented earlier. Figure 2 shows a projection of the envelope for a small shipping scenario. The projection is over cargo (16 units) and time (120 days), with the ship allocation level constant at 5. The plotted points indicate those situations in which the recommended action is to change allocation from 5 ships to 6. The upper boundary of the points is where allocation should change from 5 to 7. A DP envelope can be thought of as a stacked set of such boundaries, comparable to slack-time envelope boundaries, which mark the points at which different actions should be taken.

3.2 Discussion

DP envelopes have a number of advantages over slack-time envelopes. Decisions are no longer based on an arbitrarily complex combination of parameters, but rather on a fixed model. Adapting an envelope to a specific environment is much simplified. Rather than adjusting and testing a

possibly large number of parameters, we simply change the initial state variables and recalculate the envelope. In addition the action recommended for each state is optimal.

If we view the design of a DP envelope from an abstract level, we find a strong similarity to building a heuristic reactive plan. We begin with a domain model, design a knowledge base/DP model, and iteratively refine it until evaluation shows its performance to be acceptable. One difficulty we encountered can be attributed to this similarity between heuristic rule base and model development. As we incrementally develop a model, it becomes difficult to ensure that we introduce no errors in the tuning process. For example, we found early in development that the envelopes behaved plausibly under some conditions, but they were unreasonably sensitive to changes in the round trip travel time between the two ports. We had difficulty explaining the behavior: we weren't sure whether the behavior arose from the assumptions of the model or from an error in our implementation. The behavior turned out to be due to an error in the calculation of the transition probabilities. Though development of DP envelopes is more principled than a heuristic approach, it is still a nontrivial procedure.

We encountered a further difficulty that is characteristic of dynamic programming applications. Dynamic programming entails a relatively expensive form of search. If our state representation requires s variables, each of which takes on v values, then the time complexity of a DP algorithm may be as high as v^s. As we improve our model, adding detail and precision, the cost of calculating the policy grows steeply. This aspect of DP has come to be known as "the curse of dimensionality."

In our model this meant that we needed to abstract particular elements of the environment. It is impractical, for example, to model the travel rates of independent ships, when the number of ships may run into the hundreds or even thousands. In calculating accumulating costs, such as cargo storage costs, we needed to estimate costs to be incurred and amortize them over the course of the task. The benefits of adding details to a model must be weighed against the cost of increasing the size of the model to an impractical level.

Practically speaking, calculation time for a DP envelope far exceeds application time. In our implementation, simulating a single TransSim trial takes on the order of seconds, while the time to calculate an envelope beforehand takes on the order of fifteen or twenty minutes. On the other hand, once an envelope is calculated, it can be used and re-used in similar environments essentially for free. When there is sufficient time to pre-calculate a DP envelope, and when the agent/environment can be modeled succinctly, then we can use DP envelopes.

4 Measuring Envelope Performance

Deciding on an objective performance measure is a difficult problem. A DP envelope uses an optimal policy; however, the policy is based on a model that can only approximate the actual environment. During informal testing we found that the DP envelope matched or exceeded the performance of heuristic rules designed for specific scenarios. It is always possible, though, to devote further effort toward improving a heuristic rule base so that the reverse would hold. Here we present a more robust way to demonstrate the effectiveness of DP envelopes, relative to heuristic rules.

We begin by drawing a parallel between a heuristic knowledge base and a DP envelope. An envelope contains a policy decision for every state. Such a policy is effectively a (large) rule base, for which each possible state corresponds to the left hand side of a rule, with the policy decision the right hand side. Each envelope rule is extremely simple:

```
IF cargo remaining = C
   AND ships in use = S
   AND time remaining = T
```

`THEN Allocate/Deallocate N ships.`

A DP envelope is then a rule base consisting of a different rule for each different set of conditions. A rule base can be constructed using techniques other than dynamic programming. We can easily map a set of higher-level heuristic rules into this representation, for example. Because the rule base can then be used in the same way as a DP envelope, we will call the result of any such mapping an envelope as well.

Often a small set of heuristic rules can provide good performance for a small range of parameter values. Experience has shown us, however, that building a robust set of rules is a difficult undertaking. A DP envelope ordinarily outperforms heuristic rules and shows greater robustness. We would like to show that a DP envelope calculated for a particular environment provides the best decisions for that environment. (For convenience we call the DP envelope for a particular environment the true envelope for that environment.) While a DP envelope *should* converge to the optimal strategy, the abstractions required in modeling the environment may cause this not to be the case. We wish to show empirically that the true envelope has the best performance by comparing its performance against that of other envelopes (DP or otherwise) in the environment. For many reasons a direct comparison between the DP and heuristic approach is impractical, but we can offer evidence with the following experiment.

In this experiment we held all environmental parameters constant except the one-way travel time between ports. We let travel time range from 6, 8, 10, 12, 14, 16, to 18 days. We thus had 7 different environments. For each of these environments we generated an envelope. We then tested each envelope in all the different environments, that is, in its own environment as well as all the others.

The results for total cost are displayed in Figure 3. The marker in each portion of the figure represents the performance of the true envelope in its own environment. The dotted line marks the performance of other envelopes in the same environment. We see that each envelope performs best, or nearly best, in its own environment.

This shows that the true envelope is about as good as or better than other envelopes, and that the magnitude of difference in performance depends on the particular environment of the true envelope. We can take the analysis further. In a single environment we see a small difference between the performance of the true envelope and the performance of other envelopes designed for similar environments. For example, in the first environment, with travel time of 6, the envelope that most closely matches the true envelope in performance is the second envelope, built for a travel time of 8. We notice that the greater the difference in the environmental parameters, the greater the difference in performance. We can account for this relationship by a comparison of the internal structure of envelopes.

We can measure structure similarity between envelopes very simply, by comparing the policy for each state in an envelope against that of another envelope (or equivalently, the 'rules' of one against the 'rules' of another) and summarize the differences. If under the same conditions envelope E_1 recommends a specific action while E_2 recommends a different action, we count this as a different rule. Suppose we choose a measure such as the sum of squared differences between two envelope policies, formalized as

$$\text{Structural Difference}(E_1, E_2) \equiv SSD(E_1, E_2) = \sum_{c,s,t} \left(P_1(c,s,t) - P_2(c,s,t)\right)^2$$

where E_1 and E_2 are envelopes and P_i denotes the policy of envelope E_i.

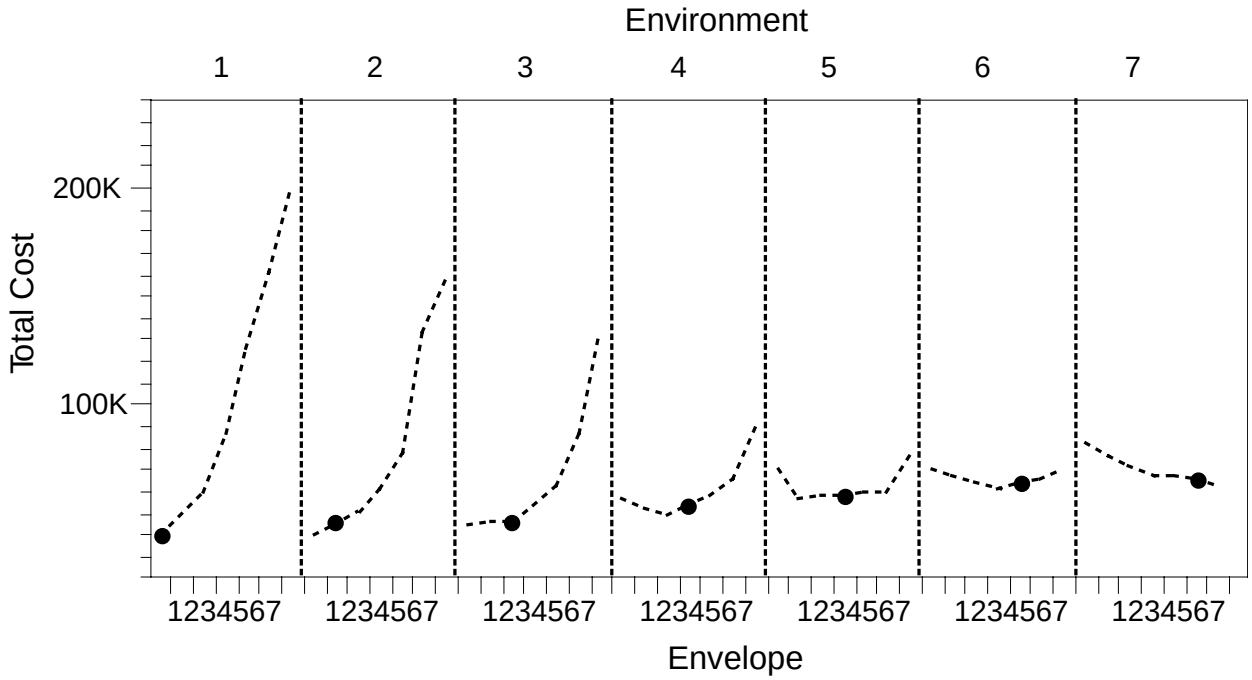

Figure 3: Results of the cross environment experiment: The performance of eight envelopes in eight different environments was measured. The best (lowest) performance in a specific environment is most often associated with the envelope designed specifically for that environment.

Besides measuring the difference between the structure of two envelopes, we can also measure the difference between their performances. For any environment V_i we can calculate a 'best' performance by its true envelope E_i. The performance of any other envelope E_j in environment V_i can be measured in terms of the degradation from the best performance. Thus for any pair of environments, we can calculate two performance degradation values. We call these values Δ.

$$\Delta : \begin{cases} \Delta(E_1, E_2|V_1) = |Cost(E_2, V_1) - Cost(E_1, V_1)| \\ \Delta(E_1, E_2|V_2) = |Cost(E_1, V_2) - Cost(E_2, V_2)| \end{cases}$$

Given these two measures, we can analyze the relationship between the structural similarity of two envelopes and the similarity of their performance. In Figure 4 we see a graph of structural difference versus performance difference for a pairwise comparison of the envelopes in the experiment above. Along the x-axis we see a log transform of the structural difference between every pair of envelopes E_1 and E_2. Along the y-axis, for each pair of envelopes we see the two symmetrical values.

We interpret this graph as follows: For a given environment V and an envelope E calculated for V, the greater the structural difference between E and another envelope F, the poorer the performance of F in environment V. In simpler terms, if we use an envelope in an inappropriate environment it is possible that it will perform much worse than the appropriate envelope. We noticed earlier that the performance difference between two envelopes depends to some extent on the environment in which they are used. Thus two structurally different envelopes may perform similarly in a specific environment; however, in general greater structural differences correspond to greater performance differences.

To summarize, we have shown evidence that a DP envelope, in the environment for which it was

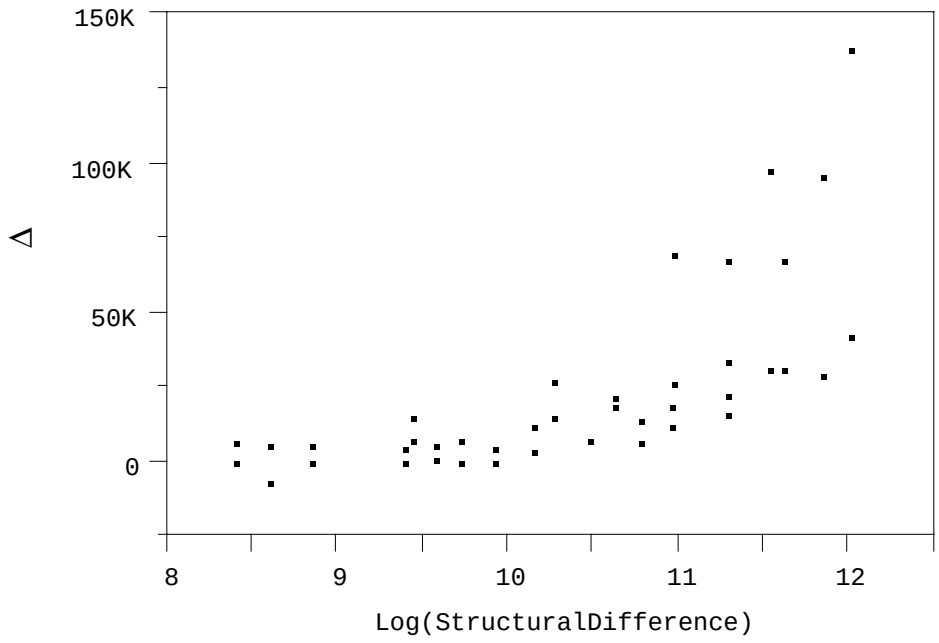

Figure 4: Performance difference versus structural difference: As the difference in policy between two envelopes increases, the difference in performance in a given environment increases as well.

designed, performs as well as or better than other envelopes in that environment. The difference in performance is proportional to the internal differences between the envelopes. This internal difference can be measured by the differences in policy between the envelopes.

Now we return to the equivalence we described earlier between the construction of an envelope policy and a knowledge base. Suppose we build a heuristic knowledge base for a particular environment. We build a DP model for this environment as well. We test the performance of both approaches, and redesign until we are satisfied with the performance. Now we change the environment. To adjust the DP envelope we simply recompile it. Adjusting the heuristic knowledge base may be much more expensive.

We also note that a single rule may subsume a great number of policy decisions in the envelope. However, while the effort required to adjust the DP envelope is constant, we can make no guarantees about the effort to change the knowledge base. It may be small, but it may also be very large. Using an envelope in a situation where costs may change can be attractive for this reason.

5 Related Work

This work has been most strongly influenced by the results of Hansen and Cohen [Hansen92], who showed that the envelope problem was amenable to dynamic programming. In further work [Hansen93], they showed that envelopes can be learned using reinforcement learning.

Reactive planning has been considered in many other contexts as well. Dean and Wellman [Dean91] outline a relationship between reactive planning and control theory. Dynamic programming is described in the context of stochastic control of dynamical systems. Dean has also recently outlined a scheme for using a related notion of envelopes for planning in uncertain environments [Dean93].

Barto and others [Barto93] address the area of learning reactive plans automatically. A machine

	Heuristic Reactive Plan	DP Envelope
Building rules	From experience	By modeling
Debugging	Trial and error	Parameter adjustment
Changing behavior	Redesign rules	Change parameters or redesign model
Accuracy	Depends on rules	Guaranteed within the model
Scalability	Depends on rules	Limited
Calculation	Execution time	Pre-compiled
Parameters	Input by hand	Input by hand (or learned)
Explanation of Rules	Easy	Hard

Table 1: Comparing heuristic reactive plans and DP envelopes

learning approach can solve many of the problems in constructing envelopes. It is possible, for example, to explore the search space selectively, which allows us to increase the complexity of the model. We can also learn a policy without an explicit model [Barto93, Watkins92]. These and other variations require that we change the basic approach and address other issues.

The DP envelopes approach to reactive planning can also be compared to Schopper's universal plans. Tradeoffs between deliberation at runtime and universal plans have been discussed elsewhere, in particular in [Ginsberg89a, Chapman89, Ginsberg89b]. A DP envelope is not a universal plan, however, in that envelopes may not be solely responsible for the decisions they make, but may act in a larger context.

6 Conclusion

We return now to our three questions. First, the initial design of an envelope is no more difficult than the design of a heuristic reactive plan. Modification of an existing design can be much easier. Experimentation showed us that the computational expense of the dynamic programming approach can be alleviated, with some possible degradation in performance. More importantly, the cost of recalculating an envelope is directly and easily measurable. The cost of refining a good heuristic plan, on the other hand, remains to a large extent hidden, even unmeasured, in the modification of the knowledge base.

Second, envelopes behave predictably. In the experiments we were able to both predict and explain changes in envelope behavior due to changes in envelope structure and environment parameter values. This was a concern because of the relative difficulty in interpreting a DP model in comparison to heuristic rules.

Finally, envelopes perform well as reactive plans. In examining their performance we found a way to measure the cost of applying an incorrect reactive plan in an environment. This kind of evaluation is especially useful for environments with probabilistic parameters or parameters that change value frequently. Producing a robust reactive plan by hand, using heuristic rules perhaps, can be difficult. An alternative, producing a less robust heuristic reactive plan and modifying the knowledge base whenever an evaluation measure changes, is both time consuming and difficult. If we rely on a DP envelope, on the other hand, we can adapt to new environment parameters simply by recalculating the envelope. While we cannot claim in general that one approach is better than the other, we can argue that in many cases it is much simpler to adjust a DP model than to modify a knowledge base. In the table below we summarize our comparison.

Acknowledgments

This work was supported by ARPA/Rome Laboratory Contract F30602-93-C-0100, and by NTT Data Communication Systems Corp. The U.S. government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

References

[Barto89] Barto A. G., Sutton R.S. and Watkins C.J.C.H, 1989. Learning and Sequential Decision Making, Dept. of Computer Science Technical Report 89-95, University of Massachusetts, Amherst.

[Barto93] Barto, A.G., Bradtke, S.J., Singh, S.P., 1993. Learning to Act using Real- Time Dynamic Programming. Dept. of Computer Science Technical Report 93-02. University of Massachusetts, Amherst.

[Beetz92] Beetz, and McDermott, D., 1992. Declarative Goals in Reactive Plans. In Artificial Intelligence Planning Systems. Morgan Kaufmann. 3-12.

[Bertsekas87] Bertsekas, 1987. Dynamic Programming: Deterministic and Stochastic Models. Prentice-Hall.

[Boddy91] Boddy, M., 1991. Any Time Problem Solving Using Dynamic Programming. In Proceedings AAAI-91. AAAI.

[Chapman89] Chapman D., 1989. Penguins Can Make Cake. AI Magazine, Winter 1989, 45-50.

[Cohen89] Cohen, P.R.; Greenberg, M.L.; Hart, D.M.; and Howe, A.E., 1989. Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. AI Magazine, 10(3):32-48.

[Cohen92] Cohen, P.R.; St. Amant, R. and Hart, D.M., 1992. Early warnings of plan failure, false positives and envelopes: Experiments and a model. In Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society. Lawrence Erlbaum Associates. 773-778.

[Cormen90] Cormen, Leiserson, and Rivest, 1990. Introduction to Algorithms. Morgan Kaufmann.

[Dean91] Dean, T.L., and Wellman, M.P., 1991. Planning and Control. Morgan Kaufmann.

[Dean93] Dean, T.L., Kaelbling, L.P., Kirman, J., and Nicholson, A., 1993. Planning With Deadlines in Stochastic Domains. In Proceedings AAAI-93. AAAI.

[Ginsberg89a] Ginsberg M.L., 1989a. Universal Planning: An (Almost) Universal Bad Idea. AI Magazine, Winter 1989. 40-44.

[Ginsberg89b] Ginsberg M.L., 1989b. Ginsberg Replies to Chapman and Schoppers Universal Planning Research: A Good or Bad Idea ? AI Magazine, Winter 1989. 61-62.

[Hansen92] Hansen, E.A. and Cohen, P.R., 1992. Learning a decision rule for monitoring tasks with deadlines. Dept. of Computer Science Technical Report 92-80, University of Massachusetts, Amherst.

[Hansen93] Hansen, E. and Cohen, P.R., 1993. Learning Monitoring Strategies to Compensate for Model Uncertainty. Working Notes of the AAAI-93 Workshop on Learning Action Models. AAAI.

[Hart90] Hart, D.M.; Anderson, S.D.; and Cohen, P.R., 1990. Envelopes as a Vehicle for Improving the Efficiency of Plan Execution. In Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control. K. Sycara (Ed.). Morgan Kaufmann. 71-76.

[Howe93] Howe, Adele E., 1993. Accepting the Inevitable: The Role of Failure Recovery in the Design of Planners. PhD thesis, Dept. of Computer Science, University of Massachusetts, Amherst.

[Howe95] Howe, Adele E. and Cohen, P.R., 1995. Understanding Planner Behavior. AI Journal. To appear.

[Kuwata92] Kuwata, Y., Hart, D.M. and Cohen, P.R., 1992. Steering the Execution of a Large-Scale Planning and Scheduling System. Working Notes of the AAAI-92 SIGMAN Workshop on Knowledge-Based Production Planning, Scheduling, and Control. AAAI. 62-72.

[Watkins92] Watkins, C.J.C.H. and Dayan, P., 1992. Q-learning. Machine Learning 8, 279-292.