Opportunistic Memory*

Kristian J- Hammond
Department of Computer Science
1100 East 58th Street
The University of Chicago
Chicago, IL 60637

Abstract

In this paper, we present a model of opportunistic planning that uses planning-time rear soning about the opportunities that might arise during plan execution. The model is composed of three parts: a planning-time mechanism that places blocked goals into memory for later activation, an understanding system that activates suspended goals as a by-product of parsing the world, and an execution-time process for evaluating opportunities and merging newly activated goals into the planning/execution agenda. We discuss this model in terms of examples from TRUCKER, a route-scheduling system, and RUNNER, an errand planner.

1 Planning and Acting

Current research in planning has taken a rather dramatic change in course in the past few years. The notion that a planner can exhaustively preplan for a set of goals prior to execution has been largely abandoned. In part, this change is the result of demonstrations that planning for conjunctive goals is undecidable [Chapman, 1987]. A more important factor has been the realization that the dual assumptions of traditional planning (a closed world and complete knowledge) are untenable in any but the simplest domains.

The planning theories that have grown out of this shift in paradigm differ from earlier theories in that they attempt to integrate planning and execution into a single process. The most extreme example of this has been the idea of *situated activity* [Agre & Chapman, 1987], which argues that plan-like behavior rises out of reflexive responses to external cues rather than from the guidance of a declarative plan. Less extreme theories center around the notion that plans have to be refined and repaired at execution time. These theories include:

- Alterman's *Adaptive Planning* [1985], in which execution-time problems are handled by moving between "semantically" similar plan steps.
- Firby's *RAPs* [1987], which allow a hierarchical planner to select between alternative plans on the basis of bottom-up information obtained at execution time.
- Georgeff and Lansky's *Procedural Reasoning System* [1987], in which changing goals and beliefs about the state of the world determine which plans are chosen to be put on the execution stack.
- Hammond's *Case-Based Planning* [1989] and Simmon's and Davis' *Generate, Test, and Debug* [1987], both of which use causal explanations of execution-time failures to choose between a variety of repairs.

Moderate or extreme, each of these theories argues that the world changes, and that a planner must respond to those changes. They define a class of planners which, for lack of a better name, we will refer to as *active planners:* planners that both produce a plan and then actively alter that plan in the face of a changing environment.

Thus far, the stress on integrating planning and execution in this work has been on the issue of recovering from plans that unexpectedly fail. Little work has been done on the corollary concern of exploiting unexpected opportunities. In this paper, we will examine this concern and suggest a memory organization that addresses this issue. We will also describe two planners that are aimed at implementing the theory: TRUCKER, a planner developed at Chicago in the domain of pickup and delivery scheduling, and RUNNER, a new planner under development in the more general domain of errand running.

2 Opportunistic Memory

Our approach in both TRUCKER and RUNNER uses episodic memory to organize, recognize and exploit opportunities. Briefly, the algorithm includes the following features:

- Goals that cannot be fit into a current ongoing plan are considered blocked and, as such, are suspended.
- Suspended goals are associated with elements of episodic memory that can be related to potential opportunities.
- These same memory structures are then used to parse the world so that the planner can make rou-

*This work was supported in part by the Office of Naval Research under contract number N00014-88-K-0295 and by DARPA contract F4962-88-C-058.

tine execution-time decisions.

- As elements of memory are activated by conditions in the world, the goals associated with them are also activated and integrated into the current processing queue.

In this way, suspended goals are brought to the planner's attention when conditions change so that the goals can be satisfied.

Because the planner's recognition of opportunities depends on the nature of its episodic memory structures, we call the overall algorithm presented here *opportunistic memory*.

3 An Example

Before we get into any details, it is important to understand the type of behavior we want to capture. We will do this by looking at a simple example. Although this example is couched in terms of a story, we are interesting in modeling the planning behavior described, not in understanding the text.

This example is taken from the RUNNER domain of errand running:

> On making breakfast for himself in the morning, John realized that he was out of orange juice. Because he was late for work he had no time to do anything about it.
>
> On his way home from work, John noticed that he was passing a Seven-Eleven and recalled that he needed orange juice. Having time, he stopped and picked up a quart and then continued home.

There are a number of interesting aspects to this example. First of all, the planner is confronted with new goals during execution as well during planning. This makes complete preplanning impossible. Second, the planner is able to stop planning for a goal before deciding exactly how to satisfy it. In effect, he is able to say "I don't have all the information or the time to completely integrate a plan for this goal into my current agenda." Using Schank's vocabulary, we call this the ability to *suspend* a goal [Schank & Abelson, 1977]. And third, although the goal is suspended, the planner is able to recognize the conditions that potentially lead to its satisfaction.[1]

There is a final element to this example that does not lie quite so close to the surface: in order to decide to suspend planning for the goal to possess orange juice, John has to do some reasoning about what a plan for that goal entails. That is, he has to see that the goal is blocked by lack of time to go to the store. As a result, he has a clear idea, at planning time, as to what an execution-time opportunity would look like.

*One could argue that in this example the planner does completely preplan for this goal and that the plan is to get the orange juice on the way home. But this is begging the question, in that we can take any version of a plan designed to satisfy this goal and still argue that opportunities to satisfy it in other ways should be exploited. For example, if John passes by someone giving away free samples of orange juice on the way to work, we would certainly want him to recognize that this is a chance to satisfy a currently suspended goal. The point is that we want a planner to exploit opportunities to satisfy goals, whether or not it has already planned for them.

4 Opportunistic Planning

The idea of opportunistic memory builds on two views of opportunism in planning—that of Hayes-Roth and Hayes-Roth [1979], and that of Birnbaum and Collins [1984].

Hayes-Roth and Hayes-Roth presented the view that a planner should be able to shift between planning strategies on the basis of perceived opportunities, even when those opportunities are unanticipated. Their model, which they called *opportunistic planning,* consisted of a blackboard architecture [Lesser *et* a/., 1975] and planning *specialists* that captured planning information at many levels of abstraction. These specialists included domain-level plan developers *(e.g.,* specialists that know about routes, stores, or conditions for specific plans) as well as more strategic operators *(e.g.,* specialists that would look for clusters of goals and goals with similar preconditions). The planner could jump between strategies as different specialists "noticed" that their activation conditions were present. For example, in scheduling a set of errands, a specialist with knowledge of clustering of errands by location could be invoked while another specialist was scheduling them by goal priority. In this way, the planner could respond to opportunities noticed at planning time.

Unfortunately, there are some problems with this view of opportunism. Primarily, it includes no model of execution. As with many planners, all planning is done in the absence of the ability to execute the plan and thus respond to the effects of that execution. It is a model of opportunism at *planning time* rather than *execution time.* It fails to capture the behavior we are interested in modeling.

More recently, Birnbaum and Collins [1984] presented a view of opportunism that does include a role for execution. Under their model, goals are viewed as independent processing entities that have their own inferential power. When a goal is suspended because of resource constraints, it continues to examine the ongoing flow of objects and events that pass by the agent. If circumstances that would allow for the satisfaction of the goal arise, the goal itself recognizes them and projects a plan into the current action agenda.

They present a simple yet compelling example of the behavior that interests them in a description of an agent trying to obtain both food and water in the wild. In their example, the agent suspends the goal to find water while trying to satisfy the goal to find food. While searching for food, however, the agent jumps over a stream and is able to recognize that the stream affords an opportunity to satisfy a suspended goal.

Birnbaum and Collins argue that this is managed by giving the suspended goal the ability to examine the current situation and inference directly off of it. They argue that this must be the case, since there can be no way to

decide, at the time of suspending a goal, the exact conditions under which it should be activated. Any planner that has to store and then activate suspend goals will miss opportunities that it could not anticipate.

Birnbaurn [1986] has argued further that indexing of suspended goals by descriptions of the conditions that allow their satisfaction is an unworkable approach. Not only will the indices be too complex, but the planner will have to constantly compare the current state of the world to the features used to index suspended goals in what amounts to a memory of unsatisfied tasks. As Birnbaurn says, this is hardly opportunism.

We share Birnbaum's and Collins' philosophical stance of trying to explain complex opportunistic behavior (even up to the subtle form of opportunism exhibited in Freudian slips). However, we disagree that this behavior results from goals constantly monitoring the world. We believe that indexing suspended goals is a far better explanation.

Birnbaum's arguments against indexing apply to models that separate a memory of goals from the planner's memory and understanding of the world. Under our model, however, there is no such distinction between types of memory. There is only one memory that is used to understand the world and to store suspended goals. Thus the act of recognition is the same as the act of indexing.

5 The Assumptions

The ideas of opportunistic memory in this paper exist in the context of a more general theory of planning and memory called *case-based planning* [Hammond, 1989; Kolodner *et a*/., 1985]. Case-based planning views planning as a memory task. Memories of past successes are used as prototypes for new plans, memories of past failures are used to avoid repeating the failures, and memories of past plan modifications are used to tailor old plans to new situations. This contrasts with the view of planning as a task of composition in which large plans are constructed piece-by-piece out of primitive actions.

There are certain assumptions that we make in case-based planning: first, we do not have a closed world. This limits the usefulness of preplanning and projection. Second, we have an incomplete and imperfect model of the world. This is assumed of not just the domain in general but also of the steps in plans that we use on a regular basis. In practice, this means that a planner cannot fully trust either its knowledge of the world's physics or its understanding of the world's current state. Third, we cannot do projection. That is, we cannot run complete simulations of our plans in order to tease out problems due to step interactions. This follows from our first and second assumptions. While we can project on the basis of *what the planner knows,* there is no guarantee that the projection will match what really happens.

We do not make these assumptions because we want to. We make them because we have been forced to. Such is the nature of real-world domains. But human planners are able to plan in complex domains, often with little knowledge of the true physics of those domains. This makes us believe that it is possible for a planner to plan and do where it cannot understand.

Strangely enough, the need for opportunistic reasoning and the tools for developing it both grow out of these assumptions. Having assumed that the planner cannot completely model the effects of its own actions or predict those of other agents, we must provide it with some sort of mechanism that will allow it *react* to the world as well as *act* in it.

Likewise, because the planner cannot completely model the world, we must assume a mechanism for parsing or understanding the world that provides the planner with enough information to make execution-time decisions. In effect, the lack of a perfect world model requires that any planner must watch what it is doing as it is doing it. We will use this understanding process as the core of our opportunistic reasoning mechanism.

There are other ramifications to the use of a case-based planner, having to do with what constitutes a planning step and the nature of projection. The most important point we want to make in this section, however, is that a planner must understand and interact with the world in which it executes plans.

6 TRUCKER and RUNNER

Our opportunistic memory algorithm is only part of an overall approach to planning. Because of this, we must include some discussion of the two planners TRUCKER and RUNNER. Both planners are case-based in design and *active* in the sense that they test their plans through interaction with complex simulated worlds.

6.1 TRUCKER

TRUCKER is a University of Chicago planner that interacts with a simulated world in order to test out its plans and learn from both failure and success. Its domain is a UPS-like pickup and delivery task in which new orders are received during the course of a day's execution. Its task is to schedule the orders and develop the routes for its trucks to follow through town.

TRUCKER creates new plans using two means: a map and a memory. TRUCKER uses its map when it initially builds a route for an area. Once it has built a new route and verified it by running it in the world, it stores the route in a case memory, indexed by its literal endpoints as well as by neighborhood and part-of-town descriptors. When it is able to find a plan for an order in memory, the plan is expanded and placed at the end of TRUCKER's action agenda. TRUCKER never tries to optimize over multiple goals unless it already has a plan in memory that does so.

TRUCKER optimizes its planning for multiple goals only when it notices an opportunity to do so during execution. If TRUCKER notices an opportunity to satisfy a goal that is scheduled later in the queue, it stops and reasons about the utility of merging the later plan with the steps it is currently running. If it is able to construct a plan that is significantly better than one which treats the plans independently, it uses the new plan. It also stores the new plan in memory, indexed by each of the separate goals. When either goal reoccurs, TRUCKER

searches its action queue for for the partner goal and uses the plan that it has created for the pair.

Even when a goal is placed on the action queue, TRUCKER treats it as though it were blocked. That is, it establishes the conditions that would allow TRUCKER to satisfy the goal and then associates the goal with the memory structures that would be active during the recognition of those conditions. For example, while planning for a pickup at the Sears Tower later in the day, TRUCKER associates the goal with its internal representation of the Tower. This allows it to activate and then satisfy the goal if it recognizes the Sears Tower earlier in the day.

6.2 RUNNER

The RUNNER project is the direct descendent of TRUCKER. RUNNER'S domain is similar to the errand-running domain used by Hayes-Roth and Hayes-Roth to study opportunistic planning [1979]. Unlike TRUCKER, RUNNER has only a single agent to control, but RUNNER'S agent is capable of a wider range of activity, and its domain allows for a richer set of goal interactions than does TRUCKER's. This change of domain was motivated by our desire to study a wider range of issues in both opportunism and execution-time failure recovery.

RUNNER's plans are MOP structures [Schank, 1982] that organize steps by their functions: establishing preconditions, side conditions, goal satisfaction, or postcondition cleanup. For each new goal being planned for, RUNNER checks for prototypical plan interactions that would allow it to merge preconditions, piggyback plans, or subsume them under single plans. For example, RUNNER looks for plans that have similar precondition steps in order to splice them together.

Like TRUCKER, RUNNER is designed to put off optimization until it obtains cues during execution that inform it of positive plan interactions. So RUNNER ends up suspending many of its goals in much the same way that TRUCKER does. The content of RUNNER's goals and plans, however, is more than just the simple pickup and delivery structures used by TRUCKER. As a result, RUNNER must do more of an analysis of what might constitute an opportunity to satisfy a goal than TRUCKER was forced to do.

RUNNER analyzes blocked plans and goals using two knowledge sources: precondition information associated with each plan and knowledge of opportunities with which it should be concerned. For example, in the case of John realizing that he cannot go to the grocery store to get orange juice, RUNNER would examine the preconditions *(e.g.,* the planner has money, is at the store, and has time) and look for specific conditions that might lead to an opportunity *(e.g.,* the planner has a missing resource, the planner is at the appropriate location or the planer has a time window). Each potential opportunity is evaluated as to how normative it is—with non-normative features ranking as better opportunities—as well as whether or not it is actually blocked in the current situation.

In our example, having money, being at a grocery store, and having time are all preconditions for buying orange juice. But there is a difference between them, in that having money is a normative condition and as such does not constitute an opportunity, while being near a store is a non-normative precondition and as such does constitute an opportunity.

7 Opportunism in TRUCKER and RUNNER

In both TRUCKER and RUNNER, blocked goals are associated with the elements of memory that will be active when opportunities to satisfy those goals arise. The main difference between the two planners is that establishing these conditions is relatively trivial in the TRUCKER domain—goals are of the same basic type. While the details of goal blockage and the analysis of opportunity differs between these planners, the basic approach to dealing with blocked goals is the same.

While these two planners are different, both require the ability to halt planning on a goal, suspend it, and then recall it when opportunities for execution present themselves. The approach used by TRUCKER and RUNNER to suspend and recall blocked goals has three basic parts.

First, the planner suspends the blocked goals by associating them with the elements of memory that describe potential opportunities. This requires that the planner have access to a vocabulary that differentiates between the different types of planning problems *(e.g.,* resource limitations, time constraints, and the planner's limitations).

Next, the planner executes the plans for its active goals. During execution, it has to monitor the ongoing effects of its plan as well as the effects of the plans of others in its world. The representational elements used to do this parsing are the same elements with which suspended goals have been associated. As a result, the planner's general recognition of a situation that constitutes an opportunity can immediately activate any goals that have previously been associated with that situation.

Finally, any activated goals are integrated into the current set of scheduled steps, and the plan is executed. This requires reasoning about resources and protections, as well as the effects of actions.

In our example of John and the orange juice, these steps translate into:

- John's goal to possess orange juice is blocked by lack of time to run the default plan. He decides, on the basis of the preconditions on his plan to possess orange juice, that being at a store would constitute an opportunity to get the orange juice.[2] As a result, he links the suspended goal to the condition of being near a store.
- While coming home, he sees and recognizes a Seven-Eleven. This activates the goal to obtain orange juice that he associated with this condition earlier in the day.

[2]Other preconditions are noted as well *(e.g.,* having money and time) but these conditions are discarded because the first is a normative condition of the planner and the second is difficult to recognize.

- He then tests the preconditions on the plan and merges it into his current agenda.

7.1 Suspending blocked goals

When either TRUCKER or RUNNER finds a goal blocked, that goal is suspended. In TRUCKER, this requires associating a request with the memory token for the locations it involves. For RUNNER, this requires somewhat deeper reasoning about the various conditions that might provide opportunities to satisfy a blocked goal.

When TRUCKER receives a new request for a pickup and delivery, it attempts to satisfy the order using a variety of methods. First it checks all active requests on its truck's agendas for one that has a known positive interaction with the new request. If this fails, TRUCKER attempts to find a truck that is currently idle to take up the order. If this also fails, TRUCKER searches its "desktop" for a suspended request that might be usefully combined with the new order. If all else fails, TRUCKER is forced to place the request on a queue of orders waiting for idle trucks.

When this is done, TRUCKER considers the goal blocked, and thus suspends it. To suspend a goal, TRUCKER marks its representation of the goal's pickup and delivery points with an annotation that there is a goal related to those locations. Because TRUCKER plans for only one type of goal, it doesn't have to do any more reasoning than this to identify good opportunities to satisfy the suspended goals.

RUNNER's domain and the goals it must plan for are more complex than TRUCKER's. As a result, it must reason far more than TRUCKER about the conditions that might constitute opportunities to satisfy a blocked goal.

In general, opportunities to run plans can be derived from the preconditions on each of the steps of a plan. A planner could, given time, move through a plan step by step and collect the preconditions that have to obtain at that point in the plan. But this would require the examination of many conditions that are not particularly useful in the context of opportunism. Some preconditions for obtaining orange juice—having money, having time, and being able to carry the carton—are not useful if we are looking for the features that will allow us to recall the suspended goal at the appropriate time. This is because there are more constraints on "opportunities" than on simple preconditions. These constraints include features such as ease of recognition, likelihood of occurrence, and predictiveness.

For example, having money is a strong precondition for buying orange juice, but it is also a normative condition. As a result, it is a bad predictor of an opportunity to satisfy the goal to have orange juice. If the suspended goal is tied to having money, the planner will be reminded of the goal far too often.

Rather than test all preconditions of a plan for these constraints, RUNNER uses a taxonomy of *opportunity types* to derive the conditions that will serve as opportunities to satisfy the plan. This taxonomy guides RUNNER's search through the plan for appropriate preconditions. Once a plan has been analyzed in terms of this taxonomy, it is annotated with pointers to the features associated with opportunities to run it. Features are removed from this list if they cause the goal and plan to be recalled at inappropriate times.

This taxonomy takes the form of a set of tests or questions that RUNNER asks of a plan:

- Is there a special[3] resource that is needed to run the plan?
  If so, associate the goal with the resource.
- Is there a special tool that is needed to run the plan?
  If so, associate the goal with the tool.
- Is there a special location associated with the plan?
  If so, associate the goal with the location.
- Is there a special agent or skill associated with the plan?
  If so, associate the goal with the agent or skill.
- Is there a specific time constraint associated with the plan?
  If so, associate the plan with the time.

There are also special-purpose rules for suspending particular goals. Possession goals, for example, are associated with the object of the possession.

Using these rules, RUNNER associates the blocked goal to possess orange juice with the location, GROCERY-STORE, and the object itself, ORANGE-JUICE. This association takes the form of a *SUSPEND* link from the representations of these items to the suspended goal itself. RUNNER associates the goal with the least likely conditions with the hope that most of the other conditions will obtain when the suspended goal is activated. The other conditions are checked when the goal is recalled, but they are not linked to the suspended goal in memory.

7.2 Recalling suspended goals

Both TRUCKER and RUNNER tie execution of actions to locations, landmarks and addresses that they recognize in the world. Thus they must parse and interpret the objects in the world. It is during this parse that they both recognize and recall previously suspended goals.

A typical TRUCKER plan, when fully expanded, is a route in the form of a list of the turns that have to be made, described in terms of street names and compass directions. So the plan step (GOTO (920 E-55th)) after a pick-up at (5802 S-V00DLAWN) expands into:

```
(START NORTH (5802 W00DLAWN))
(TURN EAST E-57TH)
(TURN NORTH S-CORNELL)
(TURN EAST E-55TH)
(STOP (920 E-55TH))
```

As TRUCKER moves through its world, it parses the objects at its current location and responds to any

[3] A feature is "special" if it does not predictably reoccur as a product of the planner's policy decisions.

changes that the tokens it has recognized suggest: turning, for example, when it recognizes the 5700 block of Woodlawn. But TRUCKER does more than this when it recognizes an individual token. It also checks the token for any annotation of a goal that might be associated with it. If one is found, TRUCKER activates the suspended goal and attempts to integrate it into the current schedule. This allows TRUCKER to easily and effectively activate suspended goals when the opportunities to satisfy them arise. Further, the overhead on this activation is trivial, in that all that TRUCKER has to do is look for a specific type of link on each of the objects it recognizes.

RUNNER has a wider variety of planning options for any one goal. RUNNER can pick up orange juice at *any* grocery store, not just a particular one. As a result, we are using a much more general and realistic approach to parsing than we used in TRUCKER.

To deal with this, RUNNER is designed to use the DMAP parser [Martin &, Riesbeck, 1986] as its recognition system. DMAP uses MOPs [Schank, 1982] to represent *concept sequences* to recognize concepts by recognition of their parts. DMAP uses a smart marker-passing algorithm in which two types of markers are used to *activate* and *predict* concepts in an ISA and PART-OF network. *Activation markers* are passed from primitive features up an abstraction hierarchy. In RUNNER, these features include type descriptions such as "road", "wall", "window", and "sign" but no tokens such as "the Seven-Eleven on Cornell and 47th". When any PART-OF a concept is active, *prediction markers* are spread to its other parts. When a predicted concept is handed an activation marker, it becomes active. Likewise, when all parts of a concept are activated, the concept itself is activated.

For our uses, we add a new type of link to DMAP. This link associates suspended goals with concepts that represent opportunities to achieve them. Pointing from concepts to goals, this SUSPEND link is traversed by any activation marker that is placed on the concept. So, the activation of a concept also activates any suspended goals associated with it.

In our example, the suspended goal to get the orange juice is associated with the concept representing 'the planner is at a grocery store". As the world is parsed, a Seven-Eleven is recognized as a sequence of "parking lot", "building", and "Seven-Eleven sign". Because DMAP is passing activation markers up ISA links, the Seven-Eleven is recognized as a particular Seven-Eleven, an instance of Seven-Elevens in general, a CONVENIENCE-STORE, a GROCERY-STORE, and a STORE. While the suspended goal is not directly associated with the concept "Seven-Eleven", it is associated with GROCERY-STORE. So the recognition of the Seven-Eleven causes the activation of the suspended goal. In general, RUNNER uses this property of DMAP to recall goals associated with general characterizations of opportunities through the recognition of specific situations.

In both planners, the basic approach is the same. In order to execute a plan, the low-level features must be disambiguated into tokens representing specific objects in the world. As this is done, each token is checked for an associated goal. And if any goal is found, it is considered a candidate for immediate satisfaction.

7.3 Exploiting the opportunities

Once a suspended goal is reactivated, it has to be evaluated for integration into the current execution agenda.

Here again, the TRUCKER approach uses special-purpose techniques tailored to the domain. When a suspended goal is recalled by the planner, it attempts to find the best placement in the current route for the awakened request. Scheduling the pickup is trivial, in that a truck is at the pickup location. The difficulty lies in scheduling the delivery. TRUCKER does this by stepping through each location already scheduled and finding the section of the route that will be the least altered by the insertion of the delivery. This can be done even before the exact routes are selected, by using the map and simple rules of geometry.

Because this optimization is fairly time-consuming, TRUCKER saves the resulting route, so that it can reuse it when the same conjunct of goals arises again. We see the recognition of execution-time opportunity as a special case of expectation failure [Schank, 1982] and treat it as an indication of a gap in TRUCKER's knowledge base.[4]

RUNNER deals with a wider variety of goals than does TRUCKER. As a result, the special-purpose techniques used in TRUCKER are not applicable. Still, RUNNER's plan-merging techniques are not as general as those used by most planners. RUNNER falls back on the same techniques for merging plans that it uses for plan construction.

Steps in RUNNER's plans are specifically labeled as to function (PRECONDITION, GOAL-SATISFACTION, POST-CONDITION, CLEAN-UP). RUNNER uses these labels to search for specific ways to merge plans. Although RUNNER goes through the same process during preplanning, the task is somewhat easier when applied to a newly activated goal. Just as TRUCKER knows that it is already at the pickup location, RUNNER knows that the conditions associated with the activation of the suspended goal are already satisfied. If they were not, the suspended goal would not have been noticed in the first place.

In our orange-juice example, the steps required to get the planner to a store can be ignored, in that being *at the store* is the condition that activated the goal in the first place. But the planner can also ignore other steps. In particular, the steps that are used to "recover" from the precondition of being at the store once the plan is over can be ignored. Because the planner did not need to *run* the steps in the GROCERY-STORE plan to get to the store, it will not have to run the steps in that plan that will get it away from the store. In general, precondition/cleanup pairs can be canceled together. The planner knows that the running plan must include the

[4] For a more detailed discussion of this type of learning, see [Hammond *et* a/., 1988].

same pair, and need not be concerned with the part of the recalled plan that includes it.

The remaining steps—going into the store, buying the orange juice, and exiting—have to be integrated in a fairly traditional way. The planner checks the preconditions not set by the activation conditions, and it notes the use of resources and their interactions with existing protections. The final product is a small change in the overall plan that takes the planner into the store for a moment before resuming his trip home.

In both TRUCKER and RUNNER, the task of integrating recalled goals is essentially the same as the task of creating an initial plan. The only difference is that both planners have additional information about the conditions that currently hold and, as a result, are able to avoid consideration of the steps that establish those conditions. RUNNER is able to go beyond this and avoid consideration of steps involving postcondition cleanup.

8 Conclusions

In this paper, we have argued the need for an execution-time ability to recognize and exploit planning opportunities. Our goal was to present a model of opportunistic planning that provides this ability with little overhead.

We argue that our model of *opportunistic memory* does exactly that. By associating blocked goals with the same structures used to represent the planner's world, we are able to get activation of suspended goals as a by-product of the understanding process.

The process, implemented in TRUCKER and currently being expanded upon in RUNNER, requires three basic steps. Suspended goals are associated with the elements of memory that are related to potential opportunities. These memory structures are then used to parse the world during execution. As elements of memory are activated by conditions in the world, any goals associated with them are also activated and integrated into the current planning queue.

This combination of planning-time suspension and execution-time activation gives both TRUCKER and RUNNER the ability to halt consideration of a goal with the assurance that the goal will be brought back to mind when conditions change to allow its satisfaction.

9 Acknowledgements

I'd like to thank Mitch Marks and Tim Converse for their work on TRUCKER as well as Jeff Berger, Neil Hurwitz, and Greg Hajek for their comments and conversation. I'd also like to thank Larry Birnbaum and Gregg Collins for letting me steal these ideas in the first place.

10 References

[Agre *k* Chapman, 1987] Agre, P.E. *k* Chapman, D. Pengi: An implementation of a theory of activity. In: *Proceedings of the Sixth National Conference on Artificial Intelligence,* Seattle, Washington, July 1987.

[Alterman, 1985] Alterman, R. Adaptive planning: Refitting old plans to new situations. In: *Proceedings of the Seventh Annual Conference of the Cognitive Science Society,* Irvine, California, August 1985.

[Birnbaum, 1986] Birnbaum, L. *Integrated processing in planning and understanding.* Yale Technical Report #480, 1986.

[Birnbaum *k* Collins, 1984] Birnbaum, L. *k* Collins, G. Opportunistic planning and Freudian slips. In: *Proceedings of the Sixth Annual Conference of the Cognitive Science Society,* Boulder, Colorado, June 1984.

[Chapman, 1987] Chapman, D. *Planning for Conjunctive Goals.* Technical Report TR 802, MIT Artificial Intelligence Laboratory, 1985.

[Firby, 1987] Firby, R.J. An investigation into reactive planning in complex domains. In: *Proceedings of the Sixth National Conference on Artificial Intelligence,* Seattle, Washington, July 1987.

[Georgeff & Lansky, 1987] Georgeff, M.P. *k* Lansky, A.L. Reactive reasoning and planning. In: *Proceedings of the Sixth National Conference on Artificial Intelligence,* Seattle, Washington, July 1987.

[Hammond, 1989] Hammond, K. *Case-based planning: Viewing planning as a memory task.* Academic Press, Cambridge, Massachusetts, 1989.

[Hammond *et al,* 1988] Hammond, K., Converse, T., *k* Marks, M. Learning from opportunities: Storing and reusing execution-time optimizations. In: *Proceedings of the Seventh National Conference on Artificial Intelligence,* St. Paul, Minnesota, August 1988.

[Hayes-Roth *k* Hayes-Roth, 1979] Hayes-Roth, B., *k* Hayes-Roth, F. A cognitive model of planning. *Cognitive Science* 2:275-310.

[Kolodner *et al,* 1985] Kolodner, J.L., Simpson, R.L., *&* Sycara-Cyranski, L. A process model of case-based reasoning in problem solving. In: *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* Los Angeles, California, August 1985.

[Lesser *et al,* 1975] Lesser, V.R., Fennell, R.D., Erman, L.D., *k* Reddy, D.R. Organization of the Hearsay-II speech understanding system. In: *IEEE Transactions on Acoustics, Speech and Signal Processing.* ASSP-23, 1975, 11-23.

[Martin *k* Riesbeck, 1986] Martin, C, *k* Riesbeck, C. Uniform parsing and inferencing for learning. In: *Proceedings of the Fifth National Conference on Artificial Intelligence,* Philadelphia, Pennsylvania, August 1986.

[Schank, 1982] Schank, R. *Dynamic memory: A theory of reminding and learning in computers and people.* Cambridge University Press, Cambridge, England, 1982.

[Schank *k* Abelson, 1977] Schank, R. *k* Abelson, It. *Scripts, plans, goals and understanding.* Lawrence Erlbaurn Associates, Hillsdale, New Jersey, 1977.

[Simmons *k* Davis, 1987] Simmons, R., *k* Davis, R. Generate, test, and debug: Combining associational rules and causal models. In *Proceedings of the Tenth Internation Joint Conference on Artificial Intelligence,* Milan, Italy, August 1987.

An Adaptive Model of Decision-Making in Planning

Gregg Collins[1]
University of Illinois
Dept of Computer Science
Urbana, Illinois
61801
Collins@p.cs.uiuc.edu

Lawrence Birnbaunv
Yale University
Dept. of Computer Science
New Haven, Connecticut
06520-2158
Birnbaum@yale.arpa

Bruce Krulwiclv
Yale University
Dept. of Computer Science
New Haven, Connecticut
06520-2158
Krulwich@yale.arpa

Abstract

Learning how to make decisions in a domain is a critical aspect of intelligent planning behavior. The ability of a planner to adapt its decision-making to a domain depends in part upon its ability to optimize the tradeoff between the sophistication of its decision procedures and their cost. Since it is difficult to optimize this tradeoff on *a priori* grounds alone, we propose that a planner start with a relatively simple set of decision procedures, and add complexity in response to experience gained in the application of its decision-making to real-world problems. Our model of this adaptation process is based on the explanation of failures, in that it is the analysis of bad decisions that drives the improvement of the decision procedures. We have developed a test-bed system for the implementation of planning models employing such an approach, and have demonstrated the ability of such a model to improve its procedure for projecting the effects of its moves in chess.

1 Introduction

The ability to plan effectively involves many different forms of reasoning: projecting the effects of actions in a current or hypothetical situation, deciding which goal to pursue from among the many that might be pursued at any given time, constructing sequences of actions that can achieve a given goal, determining whether to execute such a sequence, and so on. By and large, most research on planning in AI has concentrated on those aspects of planning that involve reasoning about actions, and more specifically, on how sequences of actions can be constructed to achieve particular and well-defined goals. Considerably less attention has been devoted to those aspects of planning that we might term *decision-making:* Deciding whether (or when) to pursue a given goal, or whether to use a given plan for the goal, or for that matter whether even to consider the goal in the first place.

To make such decisions correctly, a planner must know a great deal about the particular environment in which they arise. For example, a person's decision about whether to turn around and look if he hears footsteps behind him depends crucially on the situation in which he finds himself: His decision is likely to be quite different on a moderately populated suburban street in the middle of the day than in an empty city street late at night. The relevant characteristic of the environment, obviously, is the rate at which threats can be expected to arise, and in particular the threat of being victimized by a criminal.

One of the major issues confronting any planner is determining how much effort to expend on evaluating a given decision. Decision-making in general is subject to a tradeoff between the sophistication of the procedure employed to make the decision, and the time and effort required to perform the analysis. A more sophisticated decision-making procedure will, generally speaking, either consider more alternatives, consider each alternative in more depth, or do both. This requires more reasoning, and is correspondingly more expensive. Whether a more sophisticated procedure is worth using depends on whether the gain from its superior performance outweighs this extra cost. Consider, for example, the tradeoffs involved in deciding which goal to pursue: The planner must first determine which of a myriad possible goals—including goals to acquire resources, protect desired states, or improve the current situation, among others—are reasonable candidates, and then compare the expected benefits from each of these candidates. Obviously, the more goals the planner considers as reasonable candidates, the more likely it will be to find a better goal to pursue. On the other hand, the more goals

[1]This research was supported in part by the Defense Advanced Research Projects Agency, monitored by the Air Force Office of Scientific Research, under contract number F49620-88-C-0058, and in part by the Office of Naval Research under contract number N0014-85-K-010.

the planner must compare in making its decision, the more this procedure will cost.

The key point about such tradeoffs in the sophistication of the decision-making process is that, like the decisions themselves, they are highly dependent upon the particular planning environment. For example, in a highly time-limited situation (e.g., basketball or some other fast-paced game), a planner simply cannot afford to consider very many candidate goals when trying to determine what it should do next. Thus, when confronted with a novel planning environment, a planner must be prepared to *adapt* its decision-making processes to that environment, balancing the tradeoff between their sophistication and their cost.

It seems difficult to imagine how an agent might find the roughly optimal level of decision-making sophistication on the basis of *a priori* reasoning alone. This suggests that planners should be prepared to alter their decision-making processes on the basis of experience within a new planning environment. The approach we have taken, therefore, is to start with a relatively simple model of decision-making—one that ignores many potential alternatives, and many of the factors that might bear on deciding among them—and to progressively make it more sophisticated in response to poor decisions, i.e., decisions to pursue goals and plans that ultimately fail. In other words, our model of how a planner adapts its decision-making to new planning domains is *failure-driven* (see, e.g., Sussman, 1975; Schank, 1982; Hayes-Roth, 1983; Minton, 1984; Hammond, 1986). In such an approach, when a plan fails, the goal of preventing a similar failure in the future leads to an attempt to explain the current failure.[1] The explanation of the failure, in turn, suggests ways in which similar failures might subsequently be avoided.

We have elsewhere presented our model of failure-driven learning in planning domains (Birnbaum and Collins 1988;Collinsand Birnbaum, 1988a and 1988b). In our approach, the planner encodes a description of the intended functioning of its plans in explicit *justification structures.* When the expectation that a plan will succeed is violated, the planner can trace back through the justification structure to identify the culprit among the initial assumptions (see, e.g., deKleer, *et ah,* 1977; Doyle, 1979). By identifying which assumptions have failed, the planner is able to arrive at a characterization of how the particular si tuation brought about this failure, which can then serve as the basis for a rule that suggests what to do when similar situations arise subsequently.

[1]**Thus the approach can also be viewed as a form of explanation-based learning (see, e.g., Dejong and Mooney, 1986; Mitchell, Keller, and Kedar-Cabelli, 1986).**

This paper describes the application of our model to the progressive adaptation of decision-making procedures to new planning environments. Such an application requires spelling out the assumptions involved in the expectation that simple decision procedures will suffice, how those assumptions can fail, and how the procedures can be improved upon the diagnosis of such failures. We first discuss how these issues arise in a case study of the chess fork. We next briefly describe a program which implements some of our ideas, and then discuss its application to a second case study, concerning the development of a planner's ability to predict its opponent's move in deciding its own move in chess.

2 Case study: The fork

As our first case study in improving decision-making in planning, we will consider how a novice chess planner can learn to deal with a fork—a situation in which the planner's opponent confronts it with an attack on two pieces simultaneously. Unless a move can be made that will defend both pieces at once—or alternatively one piece can be moved or defended in a way that at the same time poses a strong counter-threat to the opponent—one of the pieces in question will surely be captured. The fork is something that all chess players eventually learn about, usually as a result of being victimized by it. The question is, what does a novice learn about the fork from this experience?

A planner victimized by an opponent's tactic should learn two things: how to avoid being victimized in the future, and how to use the same tactic to victimize others. We will concentrate here on the first of these, the question of how a planner learns to avoid the fork. Since the fork takes advantage of the planner's plan for defending its pieces, we must begin by considering what the planner knows about this plan, and in particular, what he knows about the assumptions upon which its efficacy depends.

The plan that almost every novice chess player seems to have for defending his pieces is this: Before each turn, check to see if any of the opponent's pieces is in position to move to the location of one of your pieces. If this is the case, then you can prevent the execution of the threat by moving the threatened piece, or by guarding it with another piece. This plan makes the assumption that a one-move warning of a threat to a piece is sufficient to to allow it to be defended, and this is generally the case. It is this assumption, however, that the fork takes advantage of: When two imminent threats are detected on the same turn, there is not time to block both of them.

Given that a novice planner understands that its plan depends upon this assumption, a plausible explanation

for how it comes to understand the fork can be constructed. The fork results in the failure of the plan to protect materiel, since it results in the capture of a piece. This failure can be traced to the violation of an underlying assumption of the plan, namely that a one-move warning of any threat would be sufficient.[1] The violation of this assumption was, in turn, brought about by the opponent's positioning of a piece so that it attacked two pieces simultaneously. The key point here is this: Such an analysis not only explains the fork, it suggests the way to guard against it. The failure of the assumption that a one-move warning is sufficient implies the need to detect at least one of the two threats earlier. Moreover, the fact that the failure was brought about by the positioning of a piece to attack two pieces at once means that efforts at earlier detection can be limited to situations where the opponent can, with a single move, produce a threat to two pieces. Thus, understanding how the fork violates its assumptions enables a planner to determine how it can avoid being victimized by the fork in the future: It must modify its plan for scanning the board to include a check for opponent's pieces that can be moved into position to attack two unguarded pieces of its own.

This explanation of how a novice planner learns from experiencing the fork leaves us with a question, however: Why would the planner have assumed that a one-move warning was enough in the first place? The most obvious answer is that the planner was simply unable to imagine a circumstance in which this would not be the case. Unfortunately, this explanation does not hold up upon further reflection. The idea that multiple threats pose a problem for defenses that involve detecting and blocking individual threats is well known to all human planners: This is the fundamental reason, for example, why "ganging up" allows a superior force to be defeated by a collection of inferior forces in any competitive situation. It seems likely that any human planner would have experienced this phenomenon, and grasped its significance, long before learning to play chess. But given the knowledge that detection-based plans for blocking threats are vulnerable to an overload of threats, plus the lack of any particular reason to believe that this problem would not arise in chess, it seems unlikely that the planner would nevertheless conclude that a one-move warning would always be enough. To argue this would be to conclude that vulnerability to the fork is the result of a mistake—a mistake made consistently by almost every person who ever learns the game of chess. It would be difficult to explain why such an error in reasoning should be made by nearly everyone.

In fact, the answer lies not in the logic of the situation, but in its economics. The reason for not detecting threats two moves in advance is that it would cost too much to do so: The planner would be swamped by the need to respond to a massive number of threats, since in two moves most of the opponent's pieces would be able to move to at least one square currently occupied by one of the planner's pieces. Most of these "threats" would never develop, and the planner would at best waste a great deal of time that could better be used plotting strategy. There is, in other words, a tradeoff: By detecting threats early, the planner provides more time to deal with multiple threats, but at the cost of being forced to consider an enormous number of possible—but unlikely—threats. By detecting threats later, the planner runs the risk of not having enough time to deal with multiple threats, but avoids being swamped with too many low probability reports. In this case, the tradeoff clearly is on the side of later detection, and this is likely to be true in many domains.

A key characteristic of the appropriate defense against the fork is that it avoids a blow-up in the number of detected threats by looking only for cases in which one piece can be moved so as to attack two pieces at once. In other words, by looking for forks in particular, rather than detecting all threats two moves in advance, the planner focusses its attention narrowly enough to make the extra effort worthwhile. So this is what the planner really learns from the fork: Not that one move's warning might not be enough—it already knew that—but a characterization of the circumstances in which it is not that is precise enough to allow them to be detected efficiently.

We can now provide a coherent account of how the fork is learned. The planner begins with the understanding that threats to pieces will arise periodically in chess. The standard strategy for dealing with threats in any domain is this: First, construct methods for detecting individual instances of threats in advance. Second, construct methods for blocking threats that can be executed when a threat is spotted. And third, ensure that all threats will be detected in time for a blocking routine to be carried out.

In chess, first, the obvious method for detecting threats is to determine which of the opponent's pieces could move to locations occupied by the planner's pieces in some particular number of moves. Second, the obvious method for blocking these threats is to move the threatened piece out of harm's way; a slightly more sophisticated plan is to guard the piece with another. And third, since any piece can generally either be moved or guarded in a single turn, and since—all other things being equal—it is best to detect threats as late as possible, a planner can conclude that one move's warning will generally be enough. However, the planner cannot *prove* that one move will always be enough, and so is forced to make the assumption that this will generally be the case. By

[1] See Collins and Birnbaum, 1988b, for a discussion of how this is accomplished.

making this assumption explicit, and by ensuring that it is monitored whenever a threat arises, the planner assures itself of being alerted to cases in which the assumption fails, as it does in the fork. So, while the planner is forced to accept the cost of a few failures as the price of an efficient threat-detection strategy, by monitoring the assumptions it is forced to make, and analyzing the failures that occur, it is able to gradually improve its performance by dealing with problematic cases one at a time.

In a sense, this explanation argues that a planner is deliberately courting failure as a shortcut to a better plan. Viewed this way, the alternative would have been for the planner to reason out, *a priori,* the circumstances in which multiple threats could occur, rather than waiting for those circumstances to arise in practice. In effect, this would mean inventing the fork before actually playing the game. While this is possible in principle, in practice the computational cost is likely to be extremely high—high enough that it is almost certainly cheaper to wait for the fork to happen. We believe that similar explanations underlie the acquisition of a wide range of decision-making strategies employed by planners.

3 An implementation and a second case study

In order to explore more concretely our failure-driven approach to the adaptation of decision-making processes in planning, we have constructed a test-bed—comprising mechanisms for inference and rule application, justification maintenance, expectation handlers, failureexplanation, and rule patching—and implemented within it a simple model of decision-making and planning for turn-taking games. This model, in turn, has been used for exploring rudimentary planning and learning in tic-tac-toe and in chess.

```
(def-brule proj-factor-2
   (world (move move) (player player) result (weight integer)
          (opp-move move) (time time)
          (better-move move) (better-goal goal)
          (better-value integer) (orig-value integer) )
 (project-factor proj-factor-1.5 world move player result weight)
 (and (= world (world-at-time time))
      (decide (player-opponent player)
      (possible-moves-at-time (player-opponent player) time)
      world simple-dec-factors opp-move)
      (= result (world-after-move opp-move (world-after-move
          move world)))
      (= weight 1)
   )
 [Justification here!)
```

Figure 1: Initial Projection Method

The planning model is implemented in 14 fairly general rules, concerning the detection of threats and opportunities, making choices, forming and checking expectations, and the like; some typical rules are shown later in this section. In addition, there are 22 chess-specific reasoning rules. All of these rules, general as well as specific, have associated justification structures, as do all particular facts and expectations contained in the system's data base. These justification structures are constructed and utilized by the rule applier, the failure explanation algorithm, and the rule patching mechanism. We devote the rest of this section to describing the program's application to a second case study, concerning the development of a planner's ability to predict its opponent's move in deciding its own move in chess.

The decision-making process within our planning model is composed of three interleaved components: a *decider,* which determines what move to make, a *projector,* which projects the results of a possible move in a given situation, and an *evaluator,* which evaluates a situation from a given player's perspective. Our planner starts with extremely primitive methods for accomplishing each of these decision-making tasks. First, to decide what move to make it simply projects the results of each move available to it, evaluates each resulting situation, and chooses the move that yields the best result. To project the results of making a given move, the planner simply assumes that its opponent will behave as it would in any given situation; the projection method that implements this is shown in figure 1. Finally, to evaluate a given situation, the planner computes the difference between the total values of the two player's pieces remaining on the board.

Consider now how a planner endowed with these primitive decision-making methods would behave in the situation shown in figure 3. To such a planner, moving the rook to take the knight looks like the best move. This decision is based on the erroneous expectation that its opponent will take the planner's knight. It forms this expectation because the projection method shown in figure 1 simply evaluates the opponent's options in the *original* situation, i.e., without taking into account the effects of the planner's own move. The use of such an unsophisticated procedure considerably simplifies the projection problem, because it obviates the need to recompute the opponent's response individually for each move contemplated by the planner. However, its validity, and hence the validi ty of theexpectationsthe planner generates concerning the opponent's moves, depends on an assumption that nothing will occur that will enable the opponent to make a better move than those currently available to it. This assumption, is an instance of what is sometimes referred to as an *inertia assumption,* namely an assumption that things will stay as they are as much as possible. In our planner, this assumption is itself justified by a conjunction that says roughly "Nothing will happen to give him a better move because he can't do anything to give himself one, I won't do anything to give

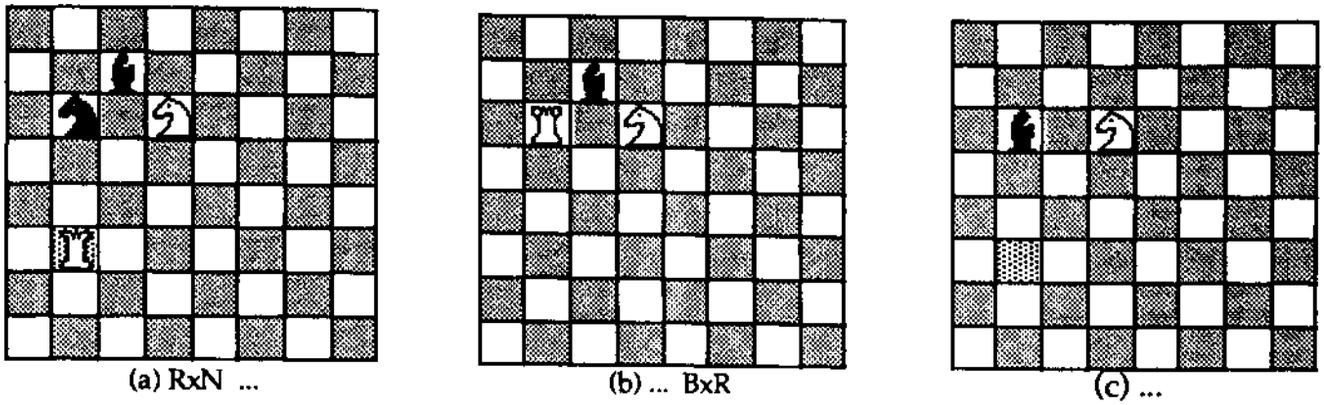

Figure 3: Game Board Sequence

him one, and no outside forces will give him one." Because the planner's decision to take the knight is justified by its projection that its opponent will take a knight in response, it monitors the status of this expectation. The expectation, in turn, is justified by the assumptions underlying the projection method used to produce it.

```
Applying rules in NEW-EXPECTATION-FAILURE-RULES
—Expectation failure:
  item=(move-to-make (move opponent move-take pawn
                                (ro>loc 2 3) (rc->loc 3 4) knight)
                     opponent (goal-unknown) 2)
Traversing...
Checking fact
#(fact 1785: (project (move computer move-take rook
                      (move opponent move-take   pawn
                            (rc->loc 2 3} (rc->loc 3 4) knight)

Checking b-rule #{b-rule proj-factor-2)
—> Valid:
(not (exists (c event)
        (move-to-make ?e.216 opponent ?better-goal.209 1)))
—> Valid:
(not (exists (e event)
        (and (extraneous-event ?e.2l7)
         (event-enables-event ?e.217 ?better-move.208))))
—> Found a bug:
(no (and (move-enables-move ?move.210 ?better-move.208)
        (move-possible ?better-move.208 ?time.207)
        (move-legal ?better-move.208)
        (evaluate (world-after-move ?better-move.208
                     (world-after-move ?move.210 ?world.211)
                  eval-factors
                  (player-opponent ?player.212)
                  ?better-value.213)
        (evaluate (world-after-move ?opp-move.214
                     (world-after-move ?move.210 ?world.21D)
                  eval-factors
                  (player-opponent ?player.212)
                  ?orig-value.215)
        (> ?better-valuc.213 ?orig-value.215)))
  *** Returning a BUG ***
```

Figure 4: Traversing an Expectation's Justification

Unfortunately, as we can see, the opponent is not going to make the move that the computer expected, and the move that he *will* make—namely taking the planner's rook—is a better move than the one the planner expected. When this happens, the planner detects that its expectation that the opponent will take the knight has failed. This then causes the planner to traverse the justification structure for that expectation, checkingthe assumptions upon which it depends.

In particular, the planner will discover that its assumption that the opponent will not have a better move has been violated. A partial transcript of this process is shown in figure 4, but it should be clear that the assumption in the justification that will fail is the inertia assumption that no event will occur that will give the opponent a better move than we expect him to have, and in particular the assumption that no action of the program's will give the opponent a better move.

As can be seen in figure 4, the system knows more than just the identity of the assumption that failed: It also knows *how* that assumption failed. The program's domain knowledge allows it to determine that the opponent's move was in fact a legal one, and a comparision of the justification for this belief with the program's prior expectation reveals that the event that enabled the opponent to make a better move than expected was in fact the computer's own move. Once the program discovers this, it must modify its reliance on the inertia assumption to take the problem into account, using the information provided in its justification structures and its analysis of the failure. The program uses standard goal regression techniques (see, e.g., Dejong and Mooney, 1986; Mitchell *etal.,* 1986) todetermine just those aspectsof the situation that caused the expectation to fail.

The resulting generalized explanation for the failure is used to patch the method that predicts the opponent's response, as utilized by the projection component, so that it checks for opportunities presented to the opponent by the planner's own move, thus ensuring that this mistake will be avoided in the future (see figure5). The new rule says roughly the following: 'To see what the world will be like after making move M, see what you would do in the opponent's situation at the current time, and assume

that he will make that move, as long as your move M doesn't enable a better move for him." When the same example is run with the new rule antecedant, the computer does not make the same mistake.

```
(def-brule proj-factor-3
  (world (move move) (player player) result (weight integer)
         (opp-move move) (time time)
         (better-move move) (better-goal goal)
         (better-value integer) (orig-value integer) )
         (project-factor proj-factor-1.5-mod world move player
              result weight)
 (and (= world (world-at-time time))
      (decide (player-opponent player)
        (possible-moves-at-time (player-opponent player) time)
      world simple-dec-factors opp-move)
      (= result (world-after-move opp-move
                   (world-after-move move world)))
      (= weight 1)
      ; here's the new stuff:
      (no (and (move-enables-move move better-move)
               (move-possible better-move (current-time))
               (move-legal better-move)
               (evaluate result eval-factors
                  (player-opponent player) orig-value)
               (evaluate (world-after-move better-move
                            (world-after-move move world))
                  eval-factors (player-opponent player)
                  better-value)
               (> better-value orig-value) ))))
```

Figure 5: The Modified Projection Method

4 Conclusion

Learning how to make decisions in a domain is a critical aspect of intelligent planning behavior. The ability of a planner to adapt its decision-making to a domain depends in part upon its ability to optimize the tradeoff between the sophistication of its decision procedures and their cost. Since it is difficult to optimize this tradeoff on *a priori* grounds alone, we propose that a planner start with a relatively simple set of decision procedures, and add complexity in response to experience gained in the application of its decision-making to real-world problems. Our model of this adaptation process is based on the explanation of failures, in that it is the analysis of bad decisions that drives the improvement of the decision procedures. We have developed a test-bed system for the implementation of planning models employing such an approach, and have demonstrated the ability of such a model to improve its procedure for projecting the effects of its moves in chess.

References

Birnbaum, L. and Collins, G. 1988. The transfer of experience across planning domains through the acquisition of abstract strategies. In J. Kolodner, ed., *Proceedings of a Workshop on Case-Based Reasoning,* Morgan Kaufmann, San Mateo, CA, pp. 61-79.

Collins, G. and Birnbaum, L. 1988a. An explanation-based approach to the transfer of planning knowledge across domains. *Proceedings of the 1988 AAAI Spring Symposium Series: Explanation-Based Learning,* Stanford, CA, pp. 107-111

Collins, G. and Birnbaum, L. 1988b. Learning strategic concepts in competitive planning: An explanation-based approach to the transfer of knowledge across domains. Research report no. UIUCDCS-R-88-1443, University of Illinois, Dept. of Computer Science, Urbana, IL, 1988.

Dejong, G. and Mooney, R. 1986. Explanation-based learning: An alternative view. *Machine Learning,* vol. 1, pp. 145-176.

deKleer,J.,Doyle,J.,Steele,G.andSussman,G..AMORD: Explicit control of reasoning. *Proceedings of the ACM Symposium on Artificial Intelligence and Programming Languages,* Rochester, NY, pp. 116-125.

Doyle, J. 1979. A truth maintenance system. *Artificial Intelligence,* vol. 12, pp. 231-272.

Hammond, K. 1986. Case-based planning: An integrated theory of planning, learning, and memory. Research report no. 488, Yale University, Dept. of Computer Science, New Haven, CT.

Hayes-Roth, F. 1983. Using proofs and refutations to learn from experience. In R. Michalski, J. Carbonell, and T. Mitchell, eds., *Machine Learning: An Artificial Intelligence Approach,* Vol. 1, Tioga, Palo Alto, CA, pp. 221 -240.

Minton,S. 1984. Constraint-based generalization: Learning game-playing plans from single examples. *Proceedings of the 1984 AAAI Conference,* Austin, TX, pp. 251-254.

Mitchell,T., Keller, R. and Kedar-Cabelli,S.1986. Explanation-based generalization: A unifying view. *Machine Learning,* vol. 1, pp. 47-80.

Schank, R. 1982. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People.* Cambridge University Press, Cambridge, England.

Sussman, G. 1975. *A Computer Model of Skill Acquisition.* American Elsevier, New York.