6.7 Learning to Predict Failures

The problem of learning to predict failures so as to anticipate and avoid them is a problem of figuring out which features of an existing situation have caused a current failure so that these links can be created.

A case-based planner saves memories of failures, indexed by the features that predict them. It uses these memories to anticipate problems when these features arise again in new situations.

When reminded of a failure, the planner needs to recall the token that has been built to represent the failure and to index the plan that avoids it in memory. By having the planner be reminded of the same representation of the failure that was used to index the plan which avoids it in memory, it can use the reminding of the failure directly in searching for the appropriate plan.

CHEF, for example, indexes STRAWBERRY-SOUFFLE by the fact that it avoided a particular problem of too much liquid and too little leavening. A case-based planner that does any sort of causal analysis of a problem in order to repair it has a powerful tool for figuring out which features in a situation have to take the blame for causing a problem. The explanation of why a failure has occurred points out which features are responsible in a current
situation for the problem. These features will be predictive of the problem in later situations. In fact, these features can be generalized, by a process I call generalizing to the level of the rules. This means generalizing an object in an explanation up to the highest level of generality that is possible while still staying within the confines of the rules that explain the failure.

The difference between this notion of explanation and that suggested by other theories of explanation-driven learning [Mitchell et al., 1986; DeJong and Mooney, 1986] lies in what is learned. Within case-based planning, the explanation of why a plan has failed is used to figure out the features that will later predict similar failures. In other theories of explanation-driven learning, explanations of why a plan has been successful are used to weed out the steps that are irrelevant and generalize those that are too specific. Explanation of failures is far more constrained; a task in that only a single anomalous state has to be accounted for and the planner is not trying to generalize the plan it is building at all. It is only trying to generalize its understanding of the circumstances in which that plan can be used.

A case-based planner uses the explanation of a failure to identify the features that will predict it. It generalizes these features to the highest level of description allowed by the rules in the explanation.

In the example of the failed strawberry soufflé plan, where too much liquid caused the soufflé to fall, part of the explanation was that the liquid came from the chopped strawberries. A simple way to avoid this failure in the future would be for the planner to mark strawberries as predictive of it and be reminded of the failure and the repaired plan whenever it is asked to make a strawberry soufflé. But the rule that explains the added liquid as a side effect of chopping the strawberries does not require that the object of the chopping be strawberries at all. It actually says that chopping any fruit will produce this effect. So, instead using STRAWBERRY as the predictive feature, CHEF can use FRUIT (Figure 6.9).

In some cases, the rule that explains a link in the causal chain leading to a failure does have specific requirements. For example, in explaining the soggy broccoli in the BEEF-AND-BROCCOLI plan, CHEF uses a rule that explains that stir frying any crisp vegetable in liquid will make it soggy. In this case, BROCCOLI is too specific a feature to look for, and VEGETABLE is too general. To deal with this, CHEF uses the tests on the rule itself to control the activation of the failure prediction. These tests are associated with the general ingredient type that the rule tests for, VEGETABLE, and tests for the features that are required for the rule to apply, (TEXTURE = CRISP).
Building demons to anticipate failure

Building demon: DEMON2 to anticipate interaction between rules:
"Chopping fruits produces liquid."
"Without a balance between liquids and leavening the batter will fall."

Indexing demon: DEMON2 under item: FRUIT by test: Is the item a FRUIT.

Indexing demon: DEMON2 under style: SOUFFLE

Goal to be activated = Avoid failure of type SIDE-EFFECT:DISABLED-CONDITION:BALANCE exemplified by the failure 'The batter is now flat' in recipe STRAWBERRY-SOUFFLE.

Figure 6.9: Marking FRUIT and SOUFFLE as Predictive of Problems

This means that each time the planner has to deal with stir frying vegetables it will test their texture and partially activates the memory of the failure if they are crisp.

In most situations, it is not a single feature but a set of features that combine to cause a failure. The BEEF-AND-BROCCOLI situation is one of these because the failure is not the result of either stir frying the beef or stir frying the broccoli but of stir frying them together. In these situations, a single feature alone should not activate the prediction of the failure. It is only when all of the features are together, (stir-fry, meat, and crisp vegetable), that the prediction should be made. To deal with this, the activation of a single feature that is linked to a failure only partially activates the memory. It is only when all features send a signal that they are present that the memory of the failure is itself activated and the planner is warned of the impending problem. CHEF does not anticipate a problem when it plans for crisp vegetables alone. It only does so when it is asked to plan for stir frying crisp vegetables along with other goals that will combine with the first to cause the problem again.
When multiple features are required to predict a failure, all of them are linked to the memory of the failure. This memory is not activated unless all of the linked features are present.

In tracing back through the explanation of a failure to the initial causes, a planner passes through states that those original causes have created. These states are the ones that are the more proximate causes of the problems but not the first causes in terms of the recipe. The liquid from chopping the strawberries is the actual cause of the problem with the strawberry soufflé but it is not the initial cause, the goal to include strawberries itself.

Although these states are not the initial causes of the problems that the planner has to deal with, they can still be used to predict the problems in later situations. A planner has to handle these intermediate states in the same way as it handles the ingredients it has to mark as predictive of problems. It generalizes them up to the level of the rules that explain the failure and link them to the token representing the failure. If other conditions are also required for the failure to occur, they are also linked to the memory of the failure so that the one feature alone will not predict the failure when it is inappropriate.

The CHEF program responds to the failure of the strawberry soufflé by marking the goal to include any liquid spice in a soufflé as predictive of that soufflé falling (Figure 6.10). This is implemented by placing a test on the concept SPICE that checks for the texture and partially activates the memory of the failure when the test is true.

Intermediate states that serve as links in the causal chain that led to a failure are also linked to the memory of the failure and can be used to predict it if they arise in a later situation.

When a set of features that combine to predict a failure is present, a case-based planner is reminded of the memory of the failure itself. In CHEF, the request for a soufflé with raspberries in it reminds the planner of the problem of extra liquid in a soufflé and allows it to find and modify the STRAWBERRY-SOUFFLE plan that deals with it. It also allows CHEF to be reminded of it when planning for a kirsch soufflé, in which the added liqueur would have the same effect as the liquid from the chopped fruit. This is a plan that the planner would not have used had it not been for the prediction of the failure. The fact that the ASSIGNED has earlier established these links between features and failures allows it to find and make use of plans that otherwise would have not interesting features in common.
Building demons to anticipate failure.

Building demon: DEMON3 to anticipate interaction between rules:
"Liquids make things wet."
"Without a balance between liquids and leavening the batter will fall."

Indexing demon: DEMON3 under item: SPICE by test: Is the TEXTURE of item LIQUID.

Indexing demon: DEMON3 under style: SOUFFLE

Goal to be activated = Avoid failure of type SIDE-EFFECT:DISABLED-CONDITION:BALANCE exemplified by the failure 'The batter is now flat' in recipe STRAWBERRY-SOUFFLE.

Figure 6.10: Marking LIQUID and SOUFFLE as Predictive of Problems

Before closing the discussion of learning to predict failures, there is one point that has to be made about indexing in a case-based planner. A case-based planner indexes its plans by the fact that they avoid particular problems. It also indexes memories of these problems by the features that predict them. Given this fact, the question that arises is why not just index the plans by the features that predict the problem that they solve directly.

There are two answers to this question. First, because a single class of problems can have many specific causes, it is just more efficient to index the plan under a single characterization of the problem and have the problem be predicted independently instead of indexing the plan by all possible circumstances that might cause the problem. Problems can be predicted with only a few features. There is little interaction between them that would lead to the need for a complex indexing system. But adding these features to the those already used to index plans would increase the complexity of plan indexing dramatically.

But efficiency is not the only argument. A more compelling argument is that indexing the plans that solve problems by the features that predict them, without marking those features in any way, doesn’t work. As was pointed out
earlier in this chapter it is often the case that the best plan for a situation is not the one that has the closest match in terms of low level features.

In CHEF, for example, the prediction of the “soggy vegetable” failure allowed the program to find the beef and broccoli plan that avoided it. Without this prediction, however, the planner would have to rely on finding a plan using only the features of the initial goals. Because the planner does have a plan for CHICKEN-AND-PEANUTS in memory, this would be taken as the base-line plan. But this plan, when modified for the current goals, will lead to soggy snow peas while a modified BEEF-AND-BROCCOLI will not.

The fact that a case-based planner indexes plans by the problems they solve and then predicts these problems for use in search allows it to recover them in situations where the low level features of the current goals will lead it to less appropriate plans.

6.8 Learning Critics

Aside from storing plans and failures, a case-based planner also stores some of the repairs that it makes to plans so that they can be used again. These repairs, stored in the form of critics, allow a planner to repair plans it knows to be faulty before it runs them. A planner would run into this situation when it predicts a problem, but cannot find a plan of the proper type to deal with that problem. This can happen if the only plans in memory that solve this problem are too different from what the input goals ask for the modification rules to handle. Remember that modification rules are not all-purpose complete planning rules. When this occurs, the case-based reasoner uses a plan that it knows to be faulty and then change it with the same patch it used to repair another faulty plan in the past.

A case-based planner stores some of the repairs that it makes, indexed by the problems that they solve.

As with learning plans, the task of the learner is not to build the repair that is going to be stored, but only to decide how it is going to be indexed so that it can be accessed at the right time. There is a difference, however, in that not all repairs can be saved. Some repairs, because they involve interactions between many parts of a plan are too complex to transfer to new problems. Others are also linked to the type of plan that is being built, so there is never a possibility that the problem is predicted but no plan of the dish type needed can be found. Aside from deciding where to store the repair, the learner also has to decide which repairs can be saved at all.
The decision to save a repair as a critic is based on the repair strategy that is used to build the specific repair and on the cause of the failure in the first place. Some failures are the product of multiple ingredients interacting, so no one ingredient can be blamed for the problem, and no one ingredient can be given the necessary critic. The first test for turning a repair into a critic, then, is whether it relates to a single ingredient. The second test depends on the complexity of the repair itself. Some strategies, such as ALTER-PLAN:PRECONDITION, ALTER-PLAN:SIDE-EFFECT, and SPLIT-AND-REFORM depend on specific steps being present in a plan and their changes cannot reliably be reused. On the other hand, strategies such as REORDER and REMOVE-FEATURE create simple changes that can be added to most plans. For example, CHEF created one recipe with duck that was too greasy. This was repaired by the REMOVE-FEATURE strategy which suggested removing the fat from the duck before cooking. This repair can be added to any plan involving duck. CHEF created another recipe that marinated shrimp before shelling, making the shelling process very difficult. This was repaired by the REORDER strategy which suggested shelling the shrimp first. This repair can be applied to any case where two steps are misordered.

A case-based planner can only save those repairs that can be transferred to any plan in which the problem that they repair arises.

In turning a repair into critic, a case-based planner has to store the specific change suggested by the strategy under the ingredient that it relates to. This is again so the prediction of a problem can lead to finding the repair that will fix it. By doing this, the past repair can be suggested when and only when the problem it relates to is noticed or predicted.

In CHEF, for example, a repair that the planner creates that adds the step of removing the fat from the duck is stored under the concept duck itself. Unlike other repairs that take the form of critics, however, these new critics are indexed by the problem that they solve. To activate them, then, CHEF has to anticipate the problem and then fail to find a plan that solves it. This restriction prevents CHEF from applying a repair to a plan that has already been fixed. It only applies the new critic when the features predicting the problem are anticipated and the plan that is used by the planner fails to deal with it.

Building a critic out of a repair requires that the planner only put together the pieces that have already been built or identified. The explanation that is used to repair the plan in the first place identifies where the critics should be stored. The critic itself is the actual repair that was built by the strategy to patch the plan in the first place.
CHEF builds new critics out of current repairs and then indexes them by the description of the problem being repaired.

Applying \textsc{top} \rightarrow \textsc{side-feature:goal-violation} to failure The bunch of dumplings is now greasy. in recipe \textsc{bad-duck-dumplings}

Implementing plan \rightarrow After doing step: Bone the duck do: Clean the fat from the duck. Suggested by strategy \textsc{remove-feature}

Figure 6.11: Repairing \textsc{duck-dumplings} with \textsc{remove-feature}

Building critic to avoid failure: The bunch of dumplings is now greasy. caused by condition: The duck is now fatty.

Critic =
After doing step: Bone the duck do: Clean the fat from the duck because: The duck is now fatty.
Storing critic under \textsc{duck}
Indexing by \textsc{side-feature:goal-violation}

Figure 6.12: Storing New Critic under \textsc{duck}

An example of this notion arises when the CHEF program has to confront the problem of the grease from duck fat. In dealing with the fat from the duck making the dish greasy, CHEF repairs the plan using the strategy \textsc{remove-feature} (Figure 6.11). Because the repair is a simple one having to do with only one ingredient, it can then go on to store it as a critic under \textsc{duck}, indexed so that the prediction of the problem will activate the critic for use in a different plan (Figure 6.12). When it later has to make a duck pasta dish and cannot use its \textsc{duck-dumplings}, it is in the position of predicting a problem that it does not solve with a complete plan. During modification, it activates the critics and adds the step that removes the fat from the duck.

6.9 How Learning Works

Learning for a case of the planner decreases the likelihood of learning new critics. The strength of learned and how it affects the plan for the problem is in the absence of new structures but can use in an active manner. Learning is defined to organize the rest of memory of experience.

As with the overall learning new critic stress is on how appropriate time. If it will fail, it may be a feature that predicts a run.

Learning is the knowledge that learning is declarative. It is used again. In the planner's results.

Because a case of the most na...
6.9. **HOW LEARNING FROM PLANNING IS DIFFERENT**

(Figure 6.13). Even though it could not find a complete plan to deal with the problem then it is able to avoid it before running its new plan by using its earlier repair.

*Problem predicted – SIDE-FEATURE:GOAL-VIOLATION*

*Considering critic:*
*After doing step: Bone the duck*
*do: Clean the fat from the duck*
*because: The duck is now fatty. – Critic applied.*

*Figure 6.13: Using New DUCK Critic*

As with the other aspects of a case-based planner’s learning, the stress in learning new critics is not on the construction of the critics themselves. The stress is on how they are to be stored so that they can be accessed at the appropriate time. By indexing the patches to a plan under the prediction that it will fail, a case-based planner can activate those patches only when the features that predict the failure are present and avoid them before the plan is run.

### 6.9 How Learning from Planning is Different

Learning for a case-based planner is a by-product of planning, so the needs of the planner determine what is going to be learned, when it is going to be learned and how it is going to be learned. Other learners that learn categories or plans in the absence of a use for what they learn are concerned with creating new structures but have little interest in how they are stored and managed for use in an active memory. A learner associated with a planner, on the other hand, does not do any building on its own. It is instead concerned with how to organize the results, both positive and negative, of its planner in a dynamic memory of experience that the planner can use.

Learning is the organization of experience. As a result, the core issues of learning are memory organization and indexing. The most important problem in learning is deciding how to describe and store an experience so that it can be used again. In a planner that learns, this means finding a way to organize the planner’s results so they can be used again in the appropriate situations.

Because a case-based planner makes use of existing plans to create new ones, the most natural thing for it to be learning is new plans so that it can
adapt them for later use. A case-based planner plans by finding the "best match" between new goals and old plans that satisfy them so it also makes sense that the most important issue in learning plans is indexing them so that they can be found at the appropriate time. A case-based planner must predict problems so that it can find plans that avoid them, so it makes sense for it to learn the features that predict the problem it encounters. Because it is possible to predict a problem that cannot be handled with an existing plan, it also makes sense for a planner to to learn specific fixes that can be applied to repair the plan it has to use. The needs of the planner decide what is going to be learned: plans, problems, and repairs.

Planning is a test of the planner's knowledge. When that knowledge fails, the planner can see through its own experience that it needs to learn some new way to discriminate between the situation it thought it was in and the one it is actually in.

Because a planner is able to test its knowledge of the world by building new plans, its planning experiences can tell it when it has to learn something from those experiences. When a case-based planner builds a successful plan, it knows to store it for later use. Because it knows in what sense the plan is successful, in that it knows the goals it satisfies and the problems it avoids, it also knows how to index it in memory. When a case-based planner fails, it knows to mark the features that caused the failure as predictive of it so that it can anticipate it in the future. The failure itself tells the planner that its knowledge has a gap, and that the gap has to be filled with the prediction of a problem that the planner was not able to anticipate before. And when a case-based planner repairs a plan, it knows that this repair is a patch that fixes a failure created by its own modification process. In response, the planner adds this patch to the knowledge used by that process. The planner determines when the learner is turned on: when the planner succeeds, fails, and repairs.

Finally, the planner itself provides the learner with the content of what is learned. It builds the plans that are stored in memory. It builds the explanation that is used to assign blame for a failure. It even builds the patch that is stored as a new ingredient critic. The learner does not create what is learned, the planner does. In every case, the task of the learner is the task of collecting the features that should be used to index the planner's work and then store that work away for later use.

The planner does the reasoning. The learner examines that reasoning and decides how it should be stored in memory such that it can be recalled again when needed. Unlike systems that do nothing but learn and as a result do nothing with what they learn, a case-based planner that learns is managing a dynamic memory of experience in service of planning.
6.10 How Learning from Planning is Better

A case-based planner must build functional categories that allow it to anticipate problems and then react to them. This is as opposed to programs that build definitional categories that are no more than lists of necessary and sufficient features. The notion of learning not only a plan, but the features that it should be indexed by is an improvement over other learners that are aimed at plan learning. A planner is concerned with the reuse of its plans while other systems are not. It is not satisfied with just having the plan in hand, it has to understand when it should be used and where it should be stored.

A case-based planner can use its knowledge as a planner to guide the credit assignment decisions it makes in marking features as predictive of problems so it does not run into the difficulties associated with inductive learning algorithms. It only has to see one instance of a problem to learn the features that predict it. It ignores extraneous features. It uses the explanation of a situation to determine the level of generality that those features will be pushed to. And it does not have to rely on a tutor to hand it the correct set of examples for generalization. By using the knowledge of what it is learning to guide it, a case-based planner can replace credit assignment through repetition with an more powerful credit assignment through relevance.

A planner knows when to learn. Unlike learning systems that do nothing but learn, a planner actually has a motivation for learning: the improvement of its planning abilities. Because it learns in response to the needs of the planner, it learns when and only when the planner requires it to. Because it learns in order to improve the planner, its learning is constrained to be only that which will help the planner with later efforts. The stress of the learning is on the use of what is learned instead of on the moment of learning itself.

Learning from planning is an improvement over other types of learning in that it uses the knowledge of the planner to determine what it learns, how to index it and when to learn it at all.

All of these improvements come for the basic idea that learning is not separate from planning. Learning is the management of the planner's memory.
of its own experience so that it can plan more effectively and avoid the problems that it has encountered before. Learning does not just involve memory, learning is memory.