ELSEVIER

Artificial Intelligence 72 (1995) 305–327

Artificial Intelligence

The stabilization of environments

Kristian J. Hammond [1], Timothy M. Converse *, Joshua W. Grass [2]

Department of Computer Science, University of Chicago, 1100 E. 58th Street, Chicago, IL 60637, USA

Received June 1992; revised October 1993

Abstract

In planning and activity research there are two common approaches to matching agents with environments. Either the agent is designed with a specific environment in mind, or it is provided with learning capabilities so that it can adapt to the environment it is placed in. In this paper we look at a third and underexploited alternative: designing agents which adapt their environments to suit themselves. We call this *stabilization*, and we present a taxonomy of types of stability that human beings typically both rely on and enforce. We also taxonomize the ways in which stabilization behaviors can be cued and learned. We illustrate these ideas with a program called FIXPOINT, which improves its performance over time by stabilizing its environment.

1. Introduction

The notion of "general purpose intelligence" is somewhat out of fashion these days. Early optimism in artificial intelligence has been tempered by the realization that the other face of generality is intractability. It is easy to find algorithms that solve general problems; the difficulty is to make problems specific enough that their solution is feasible. In AI models of activity, this injunction turns into a task of analysis; the question becomes "How is it possible to do the right thing in this domain?", and any answer that does not depend on the characteristics of that specific domain may well turn out be too general to work [3]. There is reason for skepticism even about assumptions of "domain-independent" intelligence in human beings; we may not be as general as we think we are (see [9]).

* Corresponding author. E-mail: converse@cs.uchicago.edu.

[1] This work was supported in part by DARPA contract number F49620-88-C-0058. E-mail: hammond@cs.uchicago.edu.

[2] E-mail: grass@cs.uchicago.edu.

SSDI 0004-3702(94)00006-M

This special volume of the journal *Artificial Intelligence* calls, among other things, for analyses that take into account how agents are fitted to their environments, and for theories that use such accounts to explain the success of agents, or that help in agent design. Our contribution here is to make an argument for a simple idea: that one way to ensure a good fit between agent and environment is to have the agent change the environment to suit itself. We call this kind of activity *stabilization*.[3]

1.1. Stabilization

We argue that, for an agent to interact successfully with a complex environment, one or more of the following things must be true:

- Either the agent or the environment, or both, must be designed with the other in mind (the "design" of the agent can be either evolutionary or intentional). In this case, agent and environment are well-matched from the start.
- The agent is able to learn to adapt to the particulars of its environment. In this case, through interaction, the agent comes to be better fitted to its environment over time.
- The agent is able to stabilize its environment to make it more hospitable. In this case, due to the action of the agent, the environment comes to be better fitted to the agent as time goes on.

Of course, *all* of the above may be true, and play a role in determining the success of the agent. Take the case of someone moving into a newly rented apartment, and the subsequent changes that occur in both the person and the apartment. The person (wonderfully designed, of course) occupies a new space, which (due to conventions about how buildings are constructed, as well as the particular skill of architects) is designed to make life easy in many ways for such a person. What follows is a process of mutual fitting. The person does quite a bit of learning about the particular space (becomes familiar with the floor plan, location of cabinets, electrical outlets, quirks of the plumbing, and so on). The new tenant also effects considerable change in the apartment (moving and arranging furniture, stocking the refrigerator, deciding where particular objects should be stored, and so on). As a result of *both* of these processes, the person develops a set of routines for interacting with the apartment, and is more effective and more comfortable than on the day of the move.

We use the term "stabilization" for this kind of organizing of the environment, and here is a more specific example of what we mean by it: Most people keep clean drinking glasses in some particular cabinet of their home, and when they want to drink something, they simply use one of them. This means that, at the time of deciding to have a drink of water, they need to do neither a lot of inference about whether they possess a suitable glass, nor a lot of physical search for a glass that meets their needs. The location of clean glasses is *stabilized*. Of course, this only happens due to the good efforts of someone in ensuring that glasses that are used are (at some point or other) cleaned and replaced in the cabinet.

[3] We use the term *stabilization* and *enforcement* (found in earlier papers) interchangeably here; the latter term more precisely captures the idea, but the former sounds less hostile.

In this paper we focus on this sort of stabilization, first of all because it has been neglected (in comparison with learning and more straightforward considerations in agent design), secondly because we believe that the notion can play a central role in understanding the successes of habitual human behavior, and, finally, because it seems like an unexploited research area in the analysis and design of agents that have a long-term interaction with their environments.

For the rest of the paper, we explore the idea of stabilization, and its use in the design of agents and in the analysis of agent–environment interaction. Finally we describe FIXPOINT, a program that implements some of these ideas, and some performance improvement results due to the stabilization of FIXPOINT's environment.

1.2. Models of long-term activity in AI

This paper is an entry in a long-standing debate in AI on appropriate methods for the construction of intelligent agents. Our central concern in this paper is with agents that have a long-term interaction with their environment. In the next two subsections we look very briefly at both classical and situated models, and explain why some concerns we have are addressed in neither tradition.[4] Then we will quickly present our "theory of agency", and explain how questions of stabilization fit into it. The rest of the paper will address stabilization itself.

1.2.1. Classical planning

Research on plan construction in AI can be viewed either as having produced a particular set of techniques for a well-defined computational problem, or as an *approach* that (in addition to the techniques) proposes a theory of action in which intelligent activity is seen as the result of plan construction and execution [2]. Either way, a consensus has arisen in AI that classical techniques for plan construction do not provide a complete and satisfying story about the generation of intelligent action.

There are a number of sources of this discontent: planning's formal intractability, the reliance on complete world descriptions, reliance on complete action models, and the questionable utility of plans produced when the restrictive assumptions are violated.

To this list, we would like to add one more: planning's view of activity is essentially *one-shot*. That is, classical planning techniques produce a plan to satisfy a given goal set, given a particular situation. There is no place for time-extended interaction [1] with an environment: presumably either a planner plans for the whole of time in advance, or is invoked at appropriate times to deal with particular goal sets. The former alternative multiplies the intractability of planning, while the latter requires, at the minimum, a theory of when to invoke the planner. Research on execution monitoring and interleaving of planning and execution can be seen as an attempt to develop such a theory; still, the original technology of this sort of planning was developed for one-shot goal achievement,

[4] These overviews will be so brief as to be caricatures; we apologize, but feel that these two extreme views have framed the debate on these issues to such an extent that we have to identify to the reader where we fit in the landscape defined by them.

and it remains to be seen whether it is appropriate for embedding in a longer-term context.

There is one line of planning research that is not "one-shot" in this sense: work on planning and learning (cf. the various learning attachments to the PRODIGY planning system [11]) This research concerns itself with planners that improve over time; nonetheless, the sense of performance improvement that is relevant is defined in terms of single tasks, rather than in interaction with an environment over time.

1.2.2. Situated action models

Discontents with planning models have led to an enormous variety of research in recent years—in this section we will focus on "situated action" models [1,4], and leave connections with closely related work to the reader.

This line of work stresses the *interaction* of agents and their environments, and their mutual dependence. It argues that design of an all-purpose agent is not feasible, and that it is incumbent on the designer of an agent to characterize the *dynamics* of an interaction (i.e. patterns of interaction that depend on both agent and environment, without being fully represented in either), and make use of that characterization in design. Chapman and Agre, along with championing this sort of analysis of agent–environment interactions, argue for a methodology of "machinery parsimony" (that is, a preference for the least complicated mechanisms in the agent that will explain an interaction). As illustrations of their theories of activity (and demonstration of sufficiency of simple mechanisms), they wrote programs that play videogames. In these programs, the minimal mapping between perception and action leads to the minimal proposal for a central system: a combinational logic circuit. As the central system diminishes, the peripheral systems grow in complexity. In his work on *Sonja*, for example, Chapman presents a sophisticated implementation of task-directed intermediate vision.

Our central difficulty with the models implied by the programs is that they are *steady-state*. That is, while they do provide an account of long-term interaction with an environment, they do not provide an account of how the interaction can arise (other than by careful characterization and resultant design work). It is difficult (for us, anyway), to see a place for learning in a combinational logic central system. (The only sort of learning that we are aware of that has actually been used with one of these systems is reinforcement learning; we believe that the same combinatoric problems that have dogged classical planning are likely to surface here.)

Of course, these programs were in part just sufficiency demonstrations to begin with—much of Agre's thesis has to do with analyses of how particular agent–environment interactions might arise [1], and our style of analysis here draws heavily from his. Whether our paper should be considered "situated action" research should probably depend on whether the reader will identify that term with programs or theories.

1.2.3. Our view of agency

We would like to have a theory of agency that is neither "one-shot" nor "steady-state", in that the account should explain the agent's extended interaction with an environment, while also telling a story about how that interaction can change and improve over time.

We come at this from a background in case-based planning [5], where the cost of synthetic planning is amortized by attempting to reuse prior plans as much as possible. In more recent work [6,7], we have come to see case-based planning as one part of a framework for the study of long-term agents.

In this framework, extended interaction with the environment consists of the use of a small set of plans[5] that cover the goals that typically arise, where the plans retrieved in response to environmental cues are incrementally debugged in response to failure, and are made as reliable as possible by stabilization of the environment.

Our main concern in this paper is the interaction of plan reuse and stabilization—how stabilization behaviors, external to episodes of using a particular plan, can impact the success of using the plan. To make this more concrete: imagine the "plan" to be knowledge of how to cook a particular dish, and the stabilization to be the set of behaviors that keeps the kitchen organized, cleaned, and stocked with the appropriate food and spices. Our interest is in how the plan use and the stabilization could be coordinated, particularly if the behaviors might be external to any particular episode of using the plan.

The most obvious thing to say about this is that there is clearly a tradeoff between the comprehensiveness of stabilization and the difficulties of plan reuse. For example, if a given plan achieves a given goal in a situation, the plan could obviously be reused later if the situation were *exactly* the same. If the later situation were the same in all respects relevant to the success of the plan, then (by definition) the plan could be reused as well. This will be true regardless of whether the similarity across the episodes is due to the natural stability of the world or due to the agent's efforts in stabilization. The more the "preconditions" of standard plans are stabilized, the less flexible and inventive the use of those plans will have to be.

2. Analysis versus design versus representation

There are several ways in which the concept of stabilization *might* be useful. First of all, it might be useful merely to help in understanding the dynamics that permit a particular agent to succeed in its environment. Secondly, such analysis might help in the *design* of an agent for a particular environment. Finally, such a designed agent might stabilize its environment because it happens to participate in the right dynamics, or (alternatively) because it explicitly represents and reasons about the types of stability it enforces.

As an example, consider an external view (say, by videotape) of a human in extended interaction with a kitchen. Take it as a fact about the kitchen that there is some limited number of drinking glasses, and that they cannot be used again without washing them. The behavior we observe is this: every so often the person goes to a

[5] We are torn between the desire not to use an ambiguous word like "plan", and the desire not to invent a new term when we don't mean anything really new by it. By "plan" we mean: the collection of knowledge used in pursuit of a particular (set of) goals, and that is used *only* in pursuit of those goals. This may or may not consist of a partially ordered set of "primitive actions"; we intend it usually to mean a sketchier representation used by a more flexible executive.

particular cabinet, takes out a glass, drinks from it, and puts it in the sink. At longer intervals, the person washes a number of glasses, dries them, and puts them back in the cabinet.

As a matter of analysis, we can notice that the washing-and-drying behavior supports the drinking behavior by replenishing the glasses (and that, in an odder sense, the reverse is true as well). Of course, we can only speculate on the *design* considerations that led to the behavior. Finally, does the person behave this way because of an explicit awareness that glass-washing supports glass use? Probably, but it is a subtle question, and one that might have a different answer if we wanted to design a robot to do the same task.

2.1. Analysis

We argue that the notion of stabilization is a powerful one, even just in explaining agent–environment interactions. In analyzing the role of stabilization in an interaction, several questions must be answered:

(1) What sorts of stability does the agent depend on?
(2) What sorts of stability are enforced by the agent?
(3) What does the agent *do* to perform the stabilization?
(4) How are the stabilizing behaviors organized and cued?
(5) How is the need for new kinds of stabilization recognized (if at all)?

For instance, in trying to account for what makes it possible to use clean drinking glasses at arbitrary times, the answers might be something like this:

(1) The agent depends on a wide array of types of stability including: the physics of the world, the fact that most household objects don't move unless someone moves them, and so on.
(2) Among other things, the agent enforces the fact that there is always at least one clean drinking glass in the cabinet.
(3) To ensure this, the agent periodically collects used glasses, washes them, dries them, and replaces them in the cabinet.
(4) The question of how the glass-washing behavior is organized and cued is by far the most interesting one here, and, in the case of a human being, is a question for psychologists and anthropologists rather than for us. We can, however, talk about some ways in which the behavior *might* be effectively cued:
    - Glasses could be washed immediately after use.
    - Glasses could be washed whenever a certain number of them collect in the sink.
    - All glasses could be washed every so often, say, once a day.

    These are only a few of the possibilities, and we will have more to say about this in a later section.
(5) Finally, there is the question of how the need for stabilization is recognized in the first place. This is somewhat beyond the scope of our example (which had to do with a successful "steady-state" pairing of behaviors), so we will postpone discussion of this until later in this section.

2.2. Design

Let us continue with the same example, but change our project from psychology to engineering; rather than speculating about how it is that people manage to get their dishes done, let's imagine that we have the task of designing a robot butler who must serve drinks in clean glasses at a moment's notice. How should our robot ensure that glasses are available when necessary?

Let's assume that the robot has the ability both to fill glasses and wash glasses—our concern will be with how those behaviors are linked. Of course, if the robot must always respond immediately to a request for a drink, then it is easy to see that there is the possibility of failure: all it would take would be simultaneous requests for more drinks than there were glasses in the household. As designers, we would want to ensure that, when possible, the robot did not rely on the assumption of clean glasses when none were clean, and also that the robot ensured that there were clean glasses when it was possible for it to do so.

The main source of possible failure here is that glass use will outstrip replenishment of glasses. In designing an agent to avoid that failure, there are a number of strategies to employ:

(1) Have the robot maintain an internal count of glasses used. When this count exceeds some threshold, insist that the robot collect and wash the glasses.
(2) Have the robot put used glasses into the sink, and scan the sink periodically. Whenever the number of glasses in the sink exceeds some threshold, insist that the robot wash the glasses.
(3) Assume some upper limit on the rate at which glasses are used, and insist that the robot wash glasses periodically (say, by using an internal timer).
(4) Have the robot wash glasses whenever it *notices* a dirty glass (whatever that would mean).
(5) Have the robot use glasses until they have all been used. Upon encountering a failure situation, where no clean glasses are available, have it wash all the glasses.

These suggestions vary along a number of dimensions: where information is carried (internally versus in the world), assumptions about future demands, assumptions about perceptual abilities, and the cost of occasional failure. The main point that we would like to make here is that there can be multiple ways to design an agent to enforce a single type of stability.

2.3. Representation

As we have seen, it is at least conceivable that an agent could be designed that

(1) enforced a certain kind of stability in the world,
(2) relied on that stability, and
(3) had no internal representation of either the stability or of its reliance on it.

For example, one might be able to design a robot that did three things: served drinks in glasses when asked, deposited used glasses in the sink, and washed glasses when enough glasses accumulated in the sink. Cleaning would be linked to use *only* via the

external representation of glasses standing for themselves. Assuming that the robot were able to notice the accumulation of glasses, and that glass use did not vary unpredictably (as it might, say, during a party), then it is possible that this design strategy would be effective in the absence of any internal connection between use and cleaning.

Having said that, we have to say that we don't believe in that sort of strategy for the design of stabilizing agents, primarily because it requires too much precision from the designer. The *steady-state* behavior of our robot is plausible—relying on environmental cues to tell it when to wash glasses and when to use them. But it is a lot to ask of a designer to anticipate all of the dynamics that the designed agent might participate in, particularly if the design task includes ruling out unlikely sources of failure. If our robotic butler has no representation of the connection between washing glasses and their use, then what is it to do in the case when it runs out of glasses?

3. Types of stability and stabilizing behaviors

In our discussion of so far we have focused on a single example of stabilization. In this section we give a broader categorization of a number of types of stability and stabilizing behaviors that occur in daily life, and offer some intuitive arguments for their adaptiveness. At the end of the section, we discuss the problem of recognizing the need for a novel type of stabilization.

3.1. Types of stability

Here are some types of stability that people typically enforce. In addition to the common-sense examples, we offer reasons why it might be functional to perform this sort of stabilization.

3.1.1. Stability of location

The most common type of stability that arises in everyday activity relates to the location of commonly used objects. Our drinking glasses end up in the same place every time we do dishes. Our socks are always together in a single drawer. Everything has a place and we enforce everything ending up in its place.

Enforcing STABILITY OF LOCATION, then, serves to optimize a wide range of processing goals. First of all, the fact that an often used object or tool is in a set location reduces the need for any inference or projection concerning the effects of standard plans on the objects or the current locations of objects.[6] Second, it allows plans that rely on the object's locations to be run without explicit checks (e.g., no need to explicitly determine that the glasses are in the cupboard before opening it). Third, it removes the need at execution time for a literal search for the object.

[6] This strategy happens to be spatial, and provides stability for activity in the long term. See [8] (in the companion volume) for a more general treatment of use of the spatial world to support activity. Our concerns are similar, and to some extent the taxonomies are orthogonal.

3.1.2. Stability of schedule

Another common form of stability involves the construction of standard schedules that persist over time. Eating dinner at the same time every day or having preset meetings that remain stable over time are two examples of this sort of stability. The main advantage of this sort of stability is that it allows for very effective projection in that it provides fixed points that do not have to be reasoned about. In effect, the fixed nature of certain parts of an overall schedule reduces that size of the problem space that has to be searched.

A second advantage is that fixed schedules actually allow greater optimization of the plans that are run within the confines of the stable parts of the schedule. Features of a plan that are linked to time can be removed from consideration if the plan is itself fixed in time. For example, by going into work each day at 8:30, an agent might be able to make use of the traffic report that is on the radio at the half-hour. Because the schedule is stable, however, he doesn't actually have to reason about the times that the report is on the air to be assured of hearing it.

Finally, if the schedule is stabilized with regard to a pre-existing norm, (e.g., always have lunch at noon) coordination between agents is also facilitated.

Here we see an instance of a tradeoff between enforcement and planning flexibility. While an enforced schedule allows for optimization of search and execution for recurring goals, it often reduces the flexibility required to incorporate new goals into the preset agenda. As with any heuristic that reduces the combinatorics of a search space, there will be times when an optimal plan is not considered.

It is important to realize that the schedule enforced is optimized over the goals that actually do tend to recur. Thus, an agent who is enforcing this sort of stability is able to deal with regularly occurring events with far greater ease than when it is forced to deal with goals and plans outside of its normal agenda. This sort of tradeoff in which commonly occurring problems are easier to solve than less common ones seems to be an essential by-product of stabilizing an environment.

3.1.3. Stability of resource availability

Many standard plans have a consumable resource as a precondition. If the plans are intended to be used frequently, then availability of the resource cannot be assumed unless it is enforced. A good result of this sort of enforcement is when attempts to use a plan that depends on it will usually succeed. The ideal result is when enforcement is effective enough that the question of availability need not even be raised in connection with running the plan.

3.1.4. Stability of satisfaction

Another type of stability that an agent can enforce is that of the goals that he tends to satisfy in conjunction with each other. For example, people living in apartment buildings tend to check their mail on the way into their apartments. Likewise, many people will stop at a grocery store on the way home from work. In general, people develop habits that cluster goals together into compact plans, even if the goals are themselves unrelated. The reason that the plans are together is more a product of the conditions associated with running the plans than the goals themselves.

An important feature of this sort of stability is that the goals are recurring and that the plan associated with the conjunct is optimized with respect to them. Further, the goals themselves must be on loose cycles and robust with regard to over-satisfaction.

The advantage of this sort of STABILITY OF SATISFACTION is that an optimal plan can be used that is already tuned for the interactions between individual plan steps. Second, it can be run habitually, without regard to the actual presence of the goals themselves. As in the case of STABILITY OF LOCATION in which a plan can be run without explicit checks on the locatio: of objects, STABILITY OF SATISFACTION allows for the execution of plans aimed satisfying particular goals, even when the goals are not explicitly checked.

A way to enforce this sort of stability is associate the plan with a single cue—either a goal or a feature in the world—and be in execution of that plan whenever the cue arises. In this way, the habitual activity ca be started even when all of the goals that it satisfies are not present.

3.1.5. Stability of plan use

We often find ourselves using familiar plans to satisfy goals even in the face of wide-ranging possibilities. For example, when one of us travels to conferences, he tends to schedule his flight in to a place as late as he can and plans to leave as late as he can on the last day. This optimizes his time at home and at the conference. It also allows him to plan without knowing anything about the details of the conference schedule. As a result, he has a standard plan that he can run in a wide range of situations without actually planning for them in any detail. It works, because it already deals with the major problems (missing classes at home and important talks at the conference) as part of its structure.

The major advantage here in enforcing the STABILITY OF PLAN USE is that the plan that is used is tuned to avoid the typical interactions that tend to come up. This means, of course, that the plans used in this way must either be the result of deep projection over the possible problems that can come up in a domain or be constructed incrementally. A further advantage is that little search through the space of possible plans for a set of goals needs to be done in that one plan is always selected.

3.1.6. Stability of cues

One effective technique for improving plan performance is to improve the proper activation of a plan rather than improve the plan itself. For example, placing an important paper that needs to be reviewed on his desk before going home improves the likelihood that an agent will see and read it the next day. Marking calendars and leaving notes serves the same sort of purpose.

One important area of enforcement is related to this use of visible cue in the environment to activate goals that have been suspended in memory. The idea driving this type of enforcement is that an agent can decide on a particular cue that will be established and maintained so as to force the recall of commonly recurring goals. One example of this kind of enforcement of STABILITY OF CUES is leaving a briefcase by the door every night in order to remember to bring it into work. The cue itself remains constant

over time. This means that the agent never has to make an effort to recall the goal at execution time and, because the cue is stabilized, it also never has to reason about what cue to use when the goal is initially suspended.

The advantage of this sort of enforcement is that an agent can depend on the external world to provide a stable cue to remind it of goals that still have to be achieved. This sort of stability is suggested when an agent is faced with repeated failures to recall a goal and the plan associated with the goal is tied to particular objects or tools in the world.

3.2. *Types of enforcement*

To some extent the question of which sorts of stability an agent can profit from is separate from the question of how to ensure that stability. We now categorize some methods of ensuring stability, which differ from each other partly in what actions are taken and partly in how and when those actions are cued.

3.2.1. *One-time change*

It is often possible to make a single change to the environment which will persist without further effort on the agent's part. If this is a desirable state that facilitates normal activity, it may be worthwhile to perform.

A good example of this is rearrangement of furniture, say, to remove a couch from a frequently-traveled path. Once the change has been made, it can be forgotten about, and taken as a normal fixed part of the environment. But at the same time, the world has been made more hospitable to the normal activity of the agent.

3.2.2. *Policy*

Another type of enforcement is what McDermott calls "policy" [10]. For example, everyone always carries money. This is because we always need it for a wide variety of specific plans.

Enforcement of POLICY requires the generation of specific goals to satisfy the policy state whenever it is violated. In terms of policies such as always having money on hand, this means that the lack of cash on hand will force the generation of a goal to have cash, even when no specific plan that will use that cash is present.

Many policies have to do with ensuring resource availability. Here again, the advantage is that plans can be run without explicit reference to many of the conditions that must obtain for them to be successful. An agent can actually assume conditions hold, because he has a POLICY that makes them hold.

3.2.3. *Plan modification*

Enforcement of POLICY requires detecting when the desired state is being infringed upon. Another strategy for enforcing similar types of stability is to modify all the plans that normally disturb the stable state to include its re-establishment. This strategy is only possible when the state can only be disturbed by the agent, and there is a small set of plans that are implicated.

For example, one of us typically carries a transit pass in his wallet. There is only a single plan that requires taking it out of the wallet. If that plan includes the step of putting it back, then stability of location is effectively enforced, and the assumption that it is "always true" can be made.

Whether policy or plan modification is preferable depends also on the costs and utilities of establishing the state. For example, one method for ensuring having cash might be to add a trip to an automatic teller to every plan that uses cash, thereby ensuring that it is always replenished. It so happens that the trip is costly and the violation is easy to detect, so a policy makes more sense in this case.

3.2.4. Clean-up plans

One difference between PLAN MODIFICATION and POLICY is how the actions that re-establish a desirable state are cued. The first associates the actions with detecting the violation, while the second associates them with use of the plans that disturb the state. Another alternative is to have explicit plans that look for a certain category of states that need to be re-established, and then to use the plans in response to reliable cues.

For example, most people keep their kitchens stocked with food by some mixture of noticing depletion (policy) and periodically doing more exhaustive checking for what needs to be replenished (a clean-up plan). Similarly, people often maintain stability of location in their living spaces by a mixture of "putting things back" when done with them, and "cleaning up". The fact that clean-up plans are often dissociated from the plans that violate desired states as well as from recognition of the violation means that there must be other cues that indicate when it is time to employ them. For example, it is common to have a standard routine for leaving a home or office, cued by the activity of leaving, that involves looking for various standard states that need to be re-established.

3.3. Detecting the need for stabilization

In this section we are concerned with the question of how novel stabilizing behavior might be evolved. This can be slightly difficult to distinguish from the question of how enforcement behaviors are *cued*, which occupied us in the last section. Our suggestions here are also more tentative than the previous two taxonomies; the problem of learning when to change stabilization behaviors is difficult, and probably requires exactly the sort of deep reasoning that stabilization itself is designed to avoid.

3.3.1. Plan failure

Probably the central way to recognize the need for a particular kind of stabilization is to encounter the failure of a particular plan. If this plan is to be reused frequently, there are a number of alternative responses to the failure:

- Repair the plan.
- Substitute an alternate plan for the same goal that is not subject to the failure.
- Determine the circumstances in the world that are responsible for the failure and only use the plan when those conditions do not hold.
- Determine the circumstances responsible for the failure, and arrange to stabilize them so that the plan can always be used.

This categorization is quite abstract, so as an example: imagine that you have made coffee intending to drink it with milk, but find out that in fact there is no milk in your refrigerator. What should you do differently in the future?

- Repair—you could resolve to run out to the store to get milk whenever you encountered this problem in the future.
- Substitution—decide to have tea in the future instead.
- Selective use—decide that you should check for availability of milk before having coffee, and only have coffee when milk is available.
- Stabilization—decide that the problem is in the world, rather than in your specific plan for making coffee. Resolve that there will always be milk in the refrigerator in the future. (This, of course, leaves open the question of how the stabilization is to be accomplished and cued, which we discussed in the previous section.)

Of course, this example was constructed in such a way as to make stabilization the most attractive alternative. Also, the explanatory stance is a bit disingenuous, since it is unlikely that the notion of having to buy milk would be entirely novel to someone sophisticated enough to successfully make coffee. Still, we believe that plan failure is a good indication of the need for more subtle kinds of stabilization, and that it can also indicate the need for better tuning of stabilization behaviors that already exist. Encountering the failure in the above story might well indicate the need to buy milk more frequently or consistently, even for someone who had a well-tuned set of habits.

3.3.2. Critic application

Outright failure is not the only reason to be discontented with plans or patterns of activity—substandard results, inefficiency, or wasteful use of resources can indicate the need for learning, and possibly the need to learn stabilizing behaviors.

The problem is that inefficiency, for example, is difficult to recognize, and even more difficult to assign blame for. Depending on the action model and the expressiveness of a plan representation, it may (or may not) be possible to trace outright failure to the failure of a particular step or assumed precondition. But it is difficult to detect when a plan is taking "more time than it should", and even harder to diagnose what is wrong and what should be done.

One way that classical planners increased the efficiency of plans was by looking for certain patterns of steps and precondition relationships in the current versions of plans; for example, if two different steps relied on the establishment of identical preconditions, it indicated that the plan might be transformed by establishing the precondition once for both steps, eliminating one of the establishing steps.

This sort of critic application might be useful for pointing out the need for stabilization—rather than indicating a possible transformation to the plan, the critic would point out a change in the world that would make an improved version of the plan workable. For example, if a particular plan repeatedly established a particular condition, the corresponding stabilization might be an action external to the plan that ensured that the condition was always true. As we will see, FIXPOINT operates in part by means of something quite like this kind of plan criticism.

Of course, critic application depends on having a symbolic representation, either of a proposed plan or of a sequence of actions that had been performed in the past.

3.3.3. Profiling

A standard technique in software optimization is to *profile* the program—to study the amount of time the program spends in different function calls, in hopes of finding out where the bulk of execution time is being spent. This can be done either exhaustively or statistically (by periodically sampling the stack). Once the profiling has been done, efforts in optimization can be directed to the parts of the program that are actually responsible for most of the execution time. This may turn out to be fruitless, since it may well be that those parts of the program are already as fast as they can be made. But as a development strategy it makes sense to try to speed up the functions that account for most of the time.

One problem with critic application as an optimization strategy, or as an indicator of the need for stabilization, is that it relies on a representation of a sequence of actions (whether in the future or in the past). This is fine for a linear plan execution system, which must possess such a representation anyway, but may be restrictive for more flexible action systems. As the most speculative part of this paper, we would like to suggest that something like profiling might serve as an initial focusing mechanism to indicate where stabilization could be useful. Where plan critics might require, for example, a representation of a sequence of back-and-forth travel steps before suggesting a transformation to eliminate them, a profiling approach might detect (say, by sampling) that a large portion of the time spent in pursuit of a certain goal was spent in traveling. While there would be no associated transformation to suggest, stabilizations that reduced the need for travel could then be actively sought.

We now abandon these speculations, and turn to the FIXPOINT program, and what we have learned from it.

4. The FIXPOINT program

FIXPOINT is a computer program (written by Grass) which demonstrates some of the benefits of stabilization in a simple simulated domain. We make no great claims for it, either in the completeness with which it embodies what we have talked about or in the generality of its approach to stabilization. In particular, (as we will see) the stabilization it performs occupies just one point in the space defined by our earlier taxonomies.

4.1. The domain

The agent in the FIXPOINT program works in a simulated woodshop,[7] and its task is to construct little wooden boats. This involves several pieces of wood, and a number of operations using tools such as lathes, band-saws, and so on.

[7] The simulation was built on top of Firby and Hanks' Truckworld simulator.

The agent inhabits a world made up of six rooms, which contain woodworking machines, storage bins, and pieces of wood of various shapes and sizes. At any time, a given piece of wood may be in a machine, in a bin, or just in a room (as though it were lying on the floor).

The agent has nine basic actions available to it: The agent can move in a particular direction, to get from one room to another. The agent can grab the first available object of a given type in a room with one of its hands (i.e. it can pick up a random piece of wood in a room). With a more specific version of this action, it can also find and pick up a piece of wood of given dimensions (if any such are in the room). The agent can drop anything in a given hand, can transfer the contents of a hand to the internals of a particular machine, and can take an object from a machine (transferring it to a hand). Similarly, the agent can put objects into specific bins and take them out again. Finally, the agent can *operate* the various machines, which transforms or combines the objects inside the machine in a manner dependent on the type of machine.

These actions take varying amounts of time to execute, and in some cases expand into multiple actions in execution. For example, execution of the size-specific version of "grab" involves picking up and examining pieces of wood in a room until one of the right size is found, and so can take time proportional to the number of objects in a room.

FIXPOINT's agent's job is to use these actions to make as many toy boats as it can.

4.2. *Planning and execution in* FIXPOINT

FIXPOINT's agent has a single goal (to make toy boats), which it must repeatedly pursue over the course of a long-term interaction with its environment. The environment is different each time the goal is satisfied, largely due to the agent's own actions: the agent moves objects around, depletes resources, replenishes resources, and so on.

The agent uses a single (handcrafted) plan to satisfy its goal (see appendix). This plan is in the form of a sequence of STRIPS operators, augmented with descriptions of preconditions that are necessary for the subsequent chunk of the plan to execute successfully. In addition, the agent has the background tasks of ordering more wood when necessary, and "cleaning up".

4.2.1. *What* FIXPOINT *does*

The execution cycle for FIXPOINT is as follows:

(1) If the amount of available wood is insufficient to make a certain minimum number of boats, then more wood is "ordered". This means that the ordered wood will appear after a substantial delay.

(2) If there is sufficient wood to make a boat, then the agent attempts to use its standard plan to do so. Execution consists of stepping through the plan (which is a mix of actions and precondition statements). Preconditions are checked, and when they are false, a STRIPS planner is invoked to create a plan that will make them true. This "patch plan" is executed first, and then execution returns to the main plan.

(3) After creation of the patch plan, FIXPOINT examines it for evidence that the patch would have been unnecessary if prior stabilization had been done. In the current implementation, this can happen in two ways:
- If the patch plan involves moving wood from one room into another where it is needed, FIXPOINT makes an annotation that wood of that type "should" be in the room where it was used.
- If the patch plan involves moving wood to a room, then FIXPOINT also makes an annotation that, in the future, that type of wood should be put in a bin, since this reduces the physical search necessary to find it and use it.

(4) Finally, whenever ordered wood arrives, FIXPOINT's agent delivers it to the appropriate rooms and bins, according to the annotations derived from trying to use its standard plan.

4.2.2. *World models in* FIXPOINT

In some sense there is no interesting difference between planning and execution in FIXPOINT. FIXPOINT maintains an accurate internal model of the current state of the world, and plans that FIXPOINT constructs based on that model always succeed.

FIXPOINT spends its time in one of a small number of modes. Either it is executing a part of its main plan, or it is trying (by means of invoking STRIPS and using the resulting patch plan) to establish a precondition statement of the main plan, or it is cleaning up, or it is ordering more wood. In addition FIXPOINT may be analyzing patch plans to see where wood should be stored.

We are not advocating the assumption of complete world models, nor the use of STRIPS planners to establish preconditions in plan reuse. If FIXPOINT were operating exclusively by constructing and then executing plans, then there would be no need for a simulation (since the execution of plans would give no information that could not be gleaned from their construction). As it is, though, FIXPOINT combines construction and execution of very short plans with flexible reuse of a very long standard plan (in addition to extracurricular activities like ordering wood). Although the effect of any given action is predictable, the effect of use of the main plan, or the effect of ordering and restocking, is not projected by FIXPOINT. We intend the use of short constructed plans to establish preconditions as a stand-in for improvisation to establish them. The difference-reduction approach of STRIPS is not a bad stand-in, if the plans produced are very short.

4.3. *An annotated trace*

What follows is a set of excerpts from a long trace of the FIXPOINT program at work, with some explanatory comments.

At the beginning of the trace, FIXPOINT's agent begins to try to execute its standard plan for making boats. To negotiate the first part of the plan, the agent must start out in a particular room with a piece of wood of a particular size in its hand. As it turns out, the agent is in a different room with nothing in its hand.

```
Checking precondition ---
(AND (LEFT-SIZE 2 6 ?Z) (LOC ROBOT ROOM2))
Precondition failed. Attempting to replan and patch...
```

At this point, a conventional STRIPS planner is invoked to create a "patch plan" that can be run to make the desired initial conditions hold:

```
Working on: (AND (LEFT-SIZE 2 6 ?Z) (LOC ROBOT ROOM2))
  Trying to reduce differences.
    Differences =
      ((LOC ROBOT ROOM2) (LEFT-SIZE 2 6 ?Z))
[..]
The Plan is:

(LEFT-GRAB WOOD 2 6 24 (:RECT))
(MOVE ROOM1 ROOM2)
Executing patch ---
(LEFT-GRAB WOOD 2 6 24 (:RECT))
(MOVE ROOM1 ROOM2)
```

As it happens, there is wood of the right type in the current room, but no such wood in the room that contains the needed machine. So the result of invoking STRIPS is simply to grab the requisite wood and carry it to the room where it will be used.

After the patch plan is executed successfully, it is examined for optimizations in the world that could make the preconditions of the main plan easier to establish.

```
Patch successful, returning to main plan.
**************************************************
Looking at what was needed in patch and planning to
change the environment so the precondition is met
**************************************************
Looking for...
  Optimization of location...
    Location optimizations are:
      (ROOM2 WOOD 2 6 24 (:RECT))

  Optimization of bins...
    Bin optimizations are:
      (ROOM2 (1 2 6 24 (:RECT) WOOD 1))
```

Essentially, the program examines the patch plan to see whether it involves transporting wood from one room to another. If so, it makes an annotation to itself that wood of that type "should" be in that room. In addition, an annotation is made that wood of that type should be placed in a distinguished bin, since it is less time-consuming to get wood from a known bin than it is to physically search a room for it.

Execution of the main plan proceeds in this way, with precondition statements being established by creating and executing very short plans. Sometimes the patch plans in-

volve nothing but moving the agent to an appropriate room, and (since the program has no knowledge of stabilizations that can ensure that the agent is always in a particular location) no annotations are made. Eventually, after about seventeen minutes of simulated time, with the help of five machines in different rooms, FIXPOINT's agent has constructed its first boat. In addition, it has made annotations about which rooms and bins should contain different types of wood:

```
Wood-room allocation:
  (ROOM2 WOOD 2 6 24 (:RECT))
  (ROOM2 WOOD 2 4 24 (:RECT))
  (ROOM2 WOOD 0.5 0.5 36 (:DOWEL))
  (ROOM2 WOOD 0.4 0.4 36 (:DOWEL))
  (ROOM1 WOOD 0.5 0.5 4 (:DOWEL))
  (ROOM1 WOOD 1 4 6 (:RECT (:HOLE :MEDIUM 0.5)
                           (:HOLE :LARGE 0.5)))
  (ROOM1 WOOD 0.4 0.4 3 (:DOWEL))
  (ROOM1 WOOD 2 6 10 (:SLANTED :POINTED :RECT))

Wood-bin allocation:
  (ROOM1 (1 1 WOOD 0.5 0.5 4 (:DOWEL))
         (2 1 WOOD 1 4 6 (:RECT (:HOLE :MEDIUM 0.5)
                                (:HOLE :LARGE 0.5)))
         (3 1 WOOD 0.4 0.4 3 (:DOWEL))
         (4 1 WOOD 2 6 10 (:SLANTED :POINTED :RECT)))
  (ROOM2 (1 1 WOOD 2 6 24 (:RECT))
         (2 1 WOOD 2 4 24 (:RECT))
         (3 1 WOOD 0.5 0.5 36 (:DOWEL))
         (4 1 WOOD 0.4 0.4 36 (:DOWEL)))
```

After completing its first boat, FIXPOINT's agent has depleted its stock of wood enough that it needs to order more. After ordering, FIXPOINT enters a period of "clean-up" (removing scrap wood, and putting useful pieces of wood into appropriate bins according to the annotations it has made). Then, when wood has arrived, the agent distributes the wood to appropriate rooms and bins.

```
+++ Alarms have fired:
    ((2 9 19 49)
     (ORDER-ARRIVED
      (3 (1 (:RECT) 2 4 24)
         (1 (:RECT) 2 6 24)
         (3 (:DOWEL) 0.5 0.5 36)
         (3 (:DOWEL) 0.4 0.4 36))))

(1 (:RECT) 2 4 24)...is arriving. [..]
```

After the wood has been distributed, FIXPOINT begins to use its main plan again to build a second boat. Although the main plan is the same as it was when the first boat

was made, it is being used in slightly different circumstances: the agent happens to be in a different location, and wood has been stored according to the annotations made on the first use of the plan.

```
Checking precondition ---
(AND (LEFT-SIZE 2 6 ?Z) (LOC ROBOT ROOM2))
Precondition failed. Attempting to replan and patch...

Working on: (AND (LEFT-SIZE 2 6 ?Z) (LOC ROBOT ROOM2))
  Trying to reduce differences.
    Differences =
      ((LEFT-SIZE 2 6 ?Z))
```

The agent is in the right room, but is not holding wood of the right type. Because the restocking of wood was done using the annotations made during the first use of the plan, there is now wood of the correct dimensions stored in a bin in the room it will be used in. The "patch plan" created in this case is simply to take a piece of wood from the bin.

```
The Plan is:

(LEFT-GET-FROM-BIN BIN5 1)
Executing patch ---
(LEFT-GET-FROM-BIN BIN5 1)
[..]
Patch successful, returning to main plan.
**************************************************
Looking at what was needed in patch and planning to
change the environment so the precondition is
**************************  ********************
Looking for...
  Optimization of location...
    No location optimizations.

  Optimization of bins...
    No bin storage optimizations.
```

In this way, FIXPOINT's agent constructs the second boat using the same plan as the first time around, but needs to perform fewer actions to make the various parts of the plan executable. Most of these actions that are omitted the second time involve either travel between rooms or search within a room for an object.

As a result, while it takes nearly eighteen minutes to construct the first boat (17:44), the second boat takes less than seventeen minutes (16:50). This is a small improvement, but is directly attributable to the stabilization performed. It is also larger than it seems, since more time is spent physically searching for wood in the second execution, simply because there is more wood in the rooms after the restocking.

4.4. Discussion

Now that we have discussed FIXPOINT, and have presented our taxonomy of stabilities and stabilizing behaviors, we are finally in a position to treat the first in terms of the second.

4.4.1. Types of stabilization in FIXPOINT

FIXPOINT has an inflexible commitment to stability of plan use; i.e. it has no choice but to construct boats using its standard plan. In the service of plan reuse, FIXPOINT enforces stability of location—by critiquing its patch plans, it makes annotations that enable it to ensure that the preconditions of its standard plans are either true or easily achieved. FIXPOINT tracks the use of resources internally, and periodically ensures their availability and distribution by use of a standard clean-up plan (ordering and distribution). FIXPOINT also enforces stability of schedule by tuning the minimum quantities of wood in stock that it will tolerate before ordering more (this capability was not covered in the trace). Finally, FIXPOINT displays some learning of stabilization of the location of objects: the annotations made during execution of the main plan persist, and are used during ordering. As a result, FIXPOINT decreases the total time necessary for execution of its standard plan.

4.4.2. What changes in FIXPOINT?

One possible objection the reader might have at this point is that FIXPOINT does not really change its environment—instead it changes its plans (i.e. learns), and any performance improvement should be viewed as the result of learning rather than stabilization, with change in the world a mere side-effect.

This is true in a sense, but the distinction is a subtle one. Any type of stability that needs to be actively maintained by an agent (i.e., that is not "one-time change" in our taxonomy) can be viewed as the direct product of the agent's plans and intentions. Improvement at stabilization, then, is a type of learning, and (in the long run) the changes in the world are simply the result of that learning.

The distinction (if one can be made) between improvement through stabilization and improvement through more straightforward learning depends in part on the way in which we segment the behavior we are trying to analyze. As an example of this, imagine that a person who cooks frequently learns grocery-shopping practices that ensure that the cupboard is always fully stocked with the spices typically used in that cooking. As a result, there may be fewer cooking failures, and in some sense we would want to say that the person had learned to be a better cook. To the extent that shopping is viewed as a separate activity from cooking, though, we might prefer to say that the same person now cooks in a better kitchen. The latter view may be preferable if the main connections between the activities are via this relatively stable environment that they both impact, rather than through internal representations.

4.4.3. Shortcomings and future work

The program trace we presented illustrates a particular type of stability, maintained in a particular way. In some sense, the program falls into a particular subcategory of each

of the taxonomies presented in Section 3, rather than demonstrating them all at once.

The program is mainly intended to be illustrative of the idea of stabilization; we make no particular other claims for it, and don't want to be held to its representational commitments. Two shortcomings, however, are interesting to address, if only because they suggest directions for future work.

First of all, although the particular stabilization instances that FIXPOINT learns to enforce are not built into the program, the methods for detecting the need for them and the methods for enforcing those particular stabilizations are hand-coded. One direction for future research is to try to generalize these methods so that more of the work of making stabilization decisions can be made by the program. This work is underway in the RUNNER project [7].

Secondly, as we have said, there are no interesting differences between the (symbolic) internal model that FIXPOINT has of its world, and the (symbolic) simulation that is that world. In addition to the acknowledged implausibility of such perfect world models, the fact that FIXPOINT has such a model decreases the utility of stabilizing the location of objects. An extension of this implementation would be to have the program more thoroughly substitute learned stabilization policies for its models; in other words, have the program's knowledge of the world depend less on tracking it with a world model, and more on the program's knowledge of what the world should be like (due to its own efforts).

5. Conclusions

Agents involved in long-term interactions with their environments shape those environments, in addition to being shaped by them. In this paper we've presented the idea of *stabilization*, discussed its use in the analysis and design of agents, and categorized some of the different forms it can take. Finally, we presented FIXPOINT, a program that implements many of the ideas presented here, and showed how it achieves performance improvements by actively stabilizing its environment.

Acknowledgments

We thank Mark Roller for his work with Josh Grass on the implementation of FIXPOINT. We also thank Jim Firby for asking us questions that led us to thinking about the issues in Section 2, and an anonymous reviewer for detailed and helpful comments. Finally, we thank Phil Agre for not letting us get away with not writing this.

Appendix A. Plan representation

Both to give a sense of what FIXPOINT's plan representation is like, and as an aid in understanding the trace, we reproduce the first few statements of the standard plan used by FIXPOINT.

```
'( (precondition (and (left-size 2 6 ?z)
                      (loc robot room2)))
(put-in-machine 'left)
(operate-machine '(:z 10))
(get-from-machine 'left 'used)
(get-from-machine 'right 'scrap)
(drop 'right)
  (precondition (and (left-size 2 6 10)
                     (loc robot room5)))
(put-in-machine 'left)
(operate-machine '(:pointed))
(get-from-machine 'left 'used)
  (precondition (and (left-shape (:pointed :rect))
                     (left-size 2 6 10)
                     (loc robot room3)))
(put-in-machine 'left)
(operate-machine '(:slanted))
(get-from-machine 'left 'used)
  (precondition (loc robot room1))
(drop 'left)
  (precondition (and (left-size 2 4 ?z) (loc robot room2)))
(put-in-machine 'left)
(operate-machine '(:x 1))
(get-from-machine 'left 'scrap)
(drop 'left)
(operate-machine '(:z 6))
(get-from-machine 'left 'used)
(get-from-machine 'right 'scrap)
(drop 'right)
...)
```

References

[1] P.E. Agre. The dynamic structure of everyday life, Ph.D. Thesis, Technical Report 1085, MIT Artificial Intelligence Laboratory, Cambridge, MA (1988).

[2] P.E. Agre and D. Chapman, What are plans for? in: *Designing Autonomous Agents: From Biology to Engineering and Back* (MIT Press, Cambridge, MA, 1991).

[3] D. Chapman, On choosing domains for agents, Position paper prepared for the Workshop on Benchmarks and Metrics, NASA Ames (1990).

[4] D. Chapman, *Vision, Instruction, and Action* (MIT Press, Cambridge, MA, 1991).

[5] K.J. Hammond, *Case-Based Planning: Viewing Planning as a Memory Task*, Perspectives in Artificial Intelligence **1** (Academic Press, San Diego, CA, 1989).

[6] K.J. Hammond, T.M. Converse and C. Martin, Integrating planning and acting in a case-based framework, in: *Proceedings AAAI-90*, Boston, MA (1990).

[7] K.J. Hammond, M. Marks and T.M. Converse, Planning in an open world: a pluralistic approach, in: *Proceedings 11th Annual Meeting of the Cognitive Science Society*, Ann Arbor, MI (1989).

[8] D. Kirsh, The intelligent use of space, *Artif. Intell.* **73** (1995), to appear.

[9] J. Lave, *Cognition in Practice* (Cambridge University Press, New York, 1988).
[10] D. McDermott, Planning and acting, *Cogn. Sci.* **2** (1978) 71–109.
[11] S. Minton, Learning effective search-control knowledge: an explanation-based approach, Technical Report 133, Carnegie-Mellon University, Department of Computer Science, Pittsburgh, PA (1988).