A Connectionist model of Planning via Back-chaining Search

Max Garagnani * Lokendra Shastri † Carter Wendelken †

Abstract

Great advances have marked the progress of AI planning research over the past few years. Recent systems can quickly solve problems that are orders of magnitude harder than those tackled by the best previous planners. However, we are still a long way from understanding how humans plan. Understanding how humans plan is important if we are to develop intelligent planning systems capable of dealing with complex real world problems involving uncertainty, incomplete knowledge, limited resources, non-deterministic actions and probabilistic effects.

In this paper we propose a neurally plausible model of cognitive planning that exhibits a state-space search behavior. The system is based primarily on two cognitive functionalities, namely, episodic memory and perception. In spite of its simple structure, the model can search for and execute plans involving an arbitrary number of steps. The model demonstrates how general purpose cognitive faculties such as episodic memory, semantic memory and perception can be harnessed to solve a restricted subclass of planning problems. Although the schema discussed in this paper is limited in a number of ways, we discuss how it can be extended into a more powerful and expressive planning system by incorporating additional control and memory structures.

1 Introduction

Consider a classical planning problem, specified by an initial state, a goal state and a set of operators. A direct approach to solving this problem (assuming that a solution exists) consists of searching the state space to find a 'path' between the initial and final states. Several planning systems adopting this approach in conjunction with the use of (automatically extracted or user-provided) heuristics [5, 4, 1] have shown notable results in terms of efficiency on various planning competition problems.

Although a state space search algorithm is conceptually simple, it is not obvious how the data structures and control mechanisms required for the specification and execution of such an algorithm can be realized in a neurally plausible manner. In this paper we propose a connectionist model that exhibits a state-space search behavior, but that is based primarily on general purpose cognitive functionalities. The proposed model uses only a few simple control structures that work in conjunction with essential cognitive faculties such as episodic memory, semantic memory, and perception to produce the requisite planning behavior.

Episodic memory refers to our ability to remember specific events and situations in our daily lives. Neurological and psychological data strongly suggests that episodic memory is distinct both in its functional characteristics and neural basis from other forms of memories such as semantic memory, memory for common sense knowledge, and procedural knowledge. It has been argued that events and situations in episodic memory are best viewed as *relational instances* that specify a set of bindings between the roles of a relational schema and objects that fill these roles in a given event or situation. We assume that a planning agent is capable of remembering past events such as "performing action A under conditions P lead to consequences C."

We also assume that the planning agent is capable of observing the current state of the world through perception. By this we simply mean that the agent can determine whether or not certain perceptually salient and directly observable relations hold in the world. For example, in the context of a typical blocks world scenario, this assumes that the agent can look at the table and determine whether or not block A is on top of block B.

* Department of Computing, The Open University, Milton Keynes, MK7 6AA, U.K.
† The International Computer Science Institute, Berkeley, CA, 94704 U.S.A.

2 A memory-based schema for planning and execution

Figure 1 shows the abstract structure of the proposed planning schema. Let us examine its behavior and analyse the functionality of each of its components by stepping through a complete loop of activity.

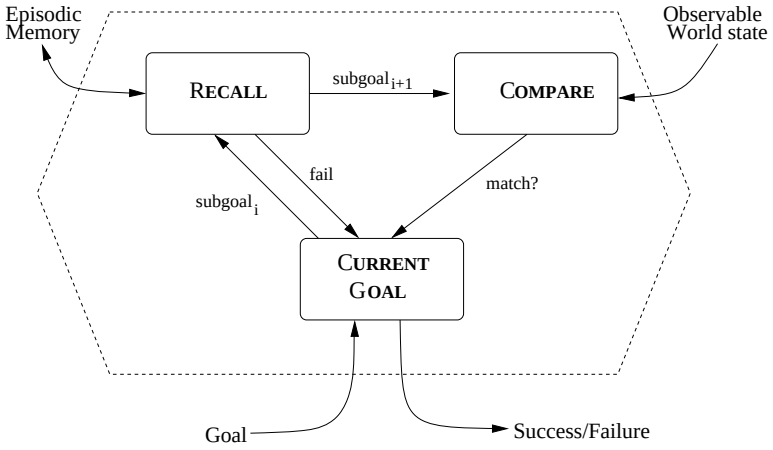

Figure 1: A block diagram showing the basic components of the planning mechanism.

1) Initialize
The activity in the schema is initialized by conveying a new goal G as input to the CURRENT GOAL component. It is assumed that this goal is represented outside the planning schema and communicated to the schema via controlled spreading activation. In the absence of any incoming activity from the COMPARE component, CURRENT GOAL simply passes control along with the goal to the RECALL component. The behavior of the CURRENT GOAL component in the presence of an incoming input from COMPARE is different, and will be described below.

2) Recall
The second component in the loop, RECALL, is activated when it receives the current goal state as input from CURRENT GOAL. The task of this component is to query the agent's episodic memory, and retrieve the trace of an *event* wherein the current goal was achieved by performing some specific *action* (say, A) under some specific *preconditions* (say, P). If the memory of such an event is found, action A and preconditions P are activated, and control is transferred to the COMPARE component. If there is no episodic memory such that its consequence C matches the goal specified by CURRENT GOAL, the activity of the schema halts and signals a *failure* (i.e., indicates that no plan was found).

Since each episodic memory trace of the type mentioned above consists of triples of the form *Precondition, Action, Consequence*, we will refer to such episodic memories (or events) as PAC memories (or events). Note that each precondition and consequence may involve multiple assertions.

3) Compare
If the query to episodic memory returns a positive answer, the set of preconditions P becomes the current (sub) goal and the focus of the agent's attention, since in order to achieve the original goal G, it suffices to achieve P and execute action A. The COMPARE block now compares P with the current world-state which is assumed to be observable through the perceptual system. If P is true in the current state, COMPARE returns a positive outcome to CURRENT GOAL. If not, COMPARE returns a negative outcome to CURRENT GOAL. After the comparison, CURRENT GOAL takes control of the activity and reacts as explained below.

4) Repeat
If CURRENT GOAL receives a positive input from COMPARE, the schema terminates returning *Success*. When this happens, the control is automatically given to the appropriate Action schema that will carry out the action currently

active in memory (A). If after the successful execution of an action, the resulting state does not satisfy the original goal, the planning schema is reinvoked.

If CURRENT GOAL receives a negative result from COMPARE, the following happens:

- The original goal G ceases to be the agent's focus of attention and is temporarily ignored. Consequently, it no longer drives the activity of the schema.
- The precondition P becomes the (new) current goal and is passed by the CURRENT GOAL to RECALL.

At this point, one loop is completed and the procedure repeats from step (2) with P as the current goal. In order to resume the activity after the complete execution of an action, the schema is re-initialized and starts executing with the initial goal G (though with a different world state).

In order to keep the model simple, we are not using a working memory that would allow the system to memorize goals and sub-goals during the search process. We are also assuming that the planning schema can entertain only *one* PAC event at a time. Because of these simplifications, the proposed system is forced to 'rediscover' parts of the same plan every time an action is executed. This process is represented in Figure 2, where the dashed lines indicate the order in which the states are (re)visited by the schema. However, as shown by the following example, these limitations do not infringe upon the soundness of the model.

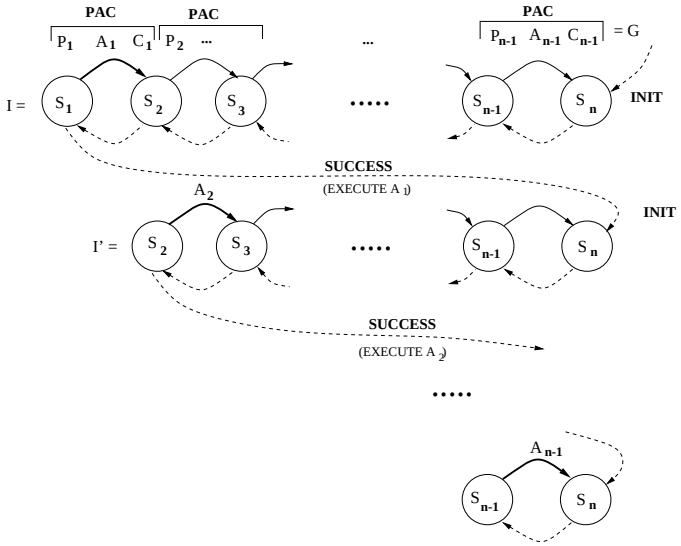

Figure 2: A diagram showing the order in which states are visited by the schema. Arrows in bold indicate action execution. Dashed lines indicate the search flow through the space.

2.1 A simple example of state space search

Let us illustrate the functioning of the planning schema through a simple example in the blocks-world domain. Assume that the initial goal state G is `On(A,B)` and the current world state is `On(B,A)`. Let us also assume that the agent's episodic memory contains the following PAC events:

1. In state P_1={ `On(B,A)` },
   action A_1= `Unstack(B,A)` led to
   state C_1={ `OnTable(A),OnTable(B)` };
2. In state P_2={ `OnTable(A),OnTable(B)` },
   action A_2= `Stack(A,B)` led to
   state C_2={ `On(A,B)` }.

When the planning schema is initialization, goal G is in the current focus and is therefore passed on to RECALL. RECALL queries the episodic memory, and since the consequences of the second PAC event, (P_2, A_2, C_2), match the current goal, this PAC event gets retrieved. In effect, the agent remembers that it had achieved `On(A,B)` by performing `Stack(A,B)` in a situation where both blocks were lying on the table. As a consequence of this recollection, the precondition P_2=`{On(A,B)}` necessary to perform the action `Stack(A,B)` is compared with the current state of the world (i.e., the initial state `{On(A,B)}`). Since the result of this comparison is negative, no action is performed, and P_2 becomes the new (sub)goal. This shifts the focus of attention from G to P_2; G is ignored, and P_2 is passed on to RECALL in its place.

The schema now queries the episodic memory by asking whether { `OnTable(A),OnTable(B)` } has been achieved in the past. The first PAC event (P_1, A_1, C_1) matches the query, since C_1 = P_2; the agent remembers that it had achieved {`OnTable(A),OnTable(B)`} from {`On(B,A)`} by performing the action `Unstack(B,A)`. As before, the precondition P_1=`{On(B,A)}` becomes the focus of attention, and is compared with the current world state. In this case, the comparison returns a positive outcome. Hence, a chain of two PAC events connecting the goal to the initial state has been found, and the planning problem has been (potentially) solved. Unfortunately, since the schema is not able to dynamically store goals and sub-goals during the search process, the first part of the chain has already been forgotten; all the agent can remember at this point is the last PAC event recollected. However, the connecting path going from the goal back to the initial state terminated exactly with such an event. Thus, the currently active action A_1=`Unstack(B,A)` can and *should* be executed, since this will get the current situation one step closer to the solution of the problem. This is what happens after COMPARE returns a positive outcome to CURRENT GOAL: the schema terminates returning '*success*', and control is passed to the 'Action' schema; the agent carries out the currently active action `Unstack(B,A)`, and the new state of the world becomes {`OnTable(A),OnTable(B)`}.

After the action has been completed, the agent is 're-exposed' to the initial goal G, which has not been achieved yet, and the planning schema is re-initialized. The flow of activity following this second initialization is identical to the first part of the loop described before, except that when P_2=`{OnTable(A),OnTable(B)}` reaches the COMPARE block, it matches the current state of the world, and thus action A_2= `Stack(A,B)` is executed. This leads to the achievement of the original goal G, and the schema is no longer invoked; the planning problem has been solved.

Notice the similarities between the two PAC events of the example and the STRIPS operator schemata `Stack`(x, y) and `Unstack`(x, y). The two PAC events can be seen as *ground instances* of operators, except for the absence of the 'Delete' list and of other 'secondary' effects, which can be *deduced* from the current context [1]. In other words, the agent can be seen as having solved a *relaxed* problem (cf. [4, 5]) in which non-relevant facts are ignored and (possible) negative interations between actions are taken care of only in a later phase of the cognitive planning process.

3 The connectionist planning schema

The planning mechanism described above has been implemented using the representational machinery of SHRUTI, a neurally plausible structured connectionist architecture that demonstrates how a network of neuron-like elements can encode a large body of structured knowledge and perform a variety of inferences within a few hundred milliseconds [16, 11, 18].

SHRUTI suggests that the encoding of relational information (frames, predicates, and schemas) is mediated by neural circuits composed of *focal-clusters*, and that the dynamic representation and communication of relational *instances* involves the transient propagation of *rhythmic* activity across these clusters. A role-entity binding is represented in this rhythmic activity by the *synchronous* firing of appropriate cells. Rules are encoded by links that enable the propagation of rhythmic activity across focal clusters, and a fact in long-term memory is a temporal pattern matching circuit.

In the past, SHRUTI's representational machinery has been used to encode commonsense knowledge [16], causal models [18], entities and types [12], as well as action schemas and reactive plans [17] and decision-making [24].

[1] For example, achieving `On(B,A)` from `OnTable(A,B)` also implies – given a two-block world context – `Clear(B)`, `Not(OnTable(B))`, etc.

3.1 A memory-based proto-planner

Consider the network structure depicted in Figure 3(a). This network fragment consists of two "control" focal-clusters `ACHIEVE` and `RECALL`, two predicate focal-clusters `On` and `OnTable`, an action focal-cluster `Unstack`, two entity focal-clusters `A` and `B`, and a type focal-cluster `Block`. Typically, a focal-cluster contains several *control* and *role* nodes. For example, the focal-cluster `ACHIEVE` contains control nodes +, -, and ?, and role nodes I and G (the entity and type focal-clusters do not contain role nodes).

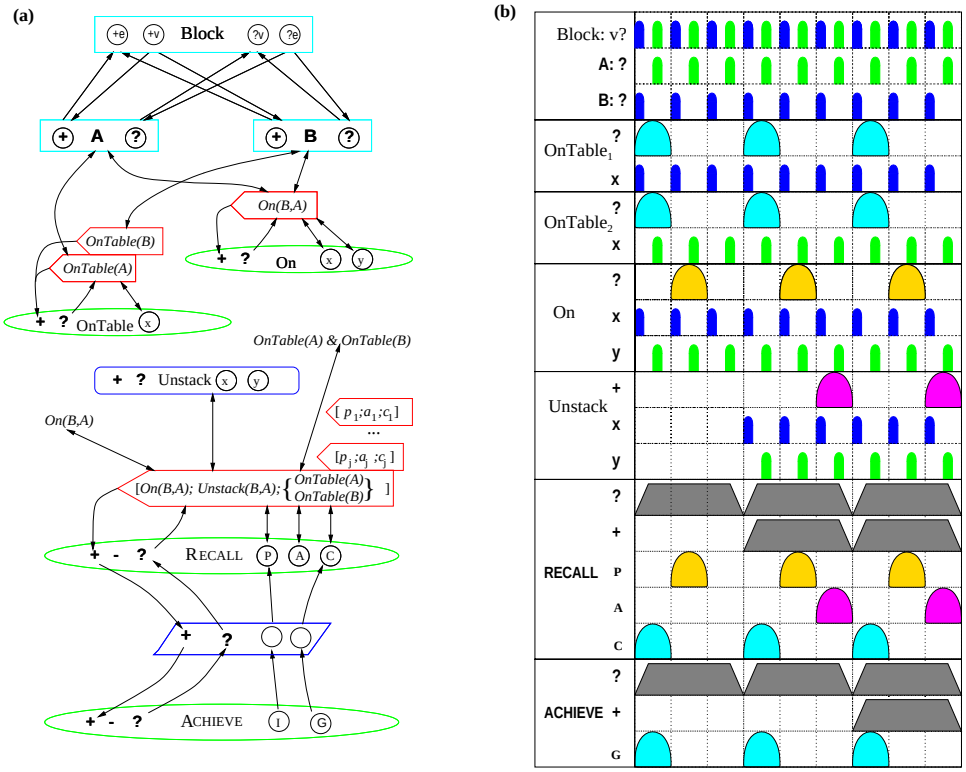

Figure 3: (a) diagram showing network structure for a proto-planning mechanism. (b) brief activation trace, showing activity of select nodes at time when `Recall` fact is matched.

Role nodes within the focal-cluster of a predicate or schema provide a mechanism for expressing role (or parameter) bindings. In particular, a dynamic binding between a role and its filler is expressed by the synchronous firing of the role node and the focal-cluster of the object filling the role. A relational focal-cluster with bound role nodes designates a particular *relational instance*. For example, the synchronous activation of role node `On.x` with the + node in the focal-cluster of `B`, and the synchronous firing of the role node `On.y` with the + node in the focal-cluster for `A` specifies the relational instance `On(B,A)`.

Role nodes have the interesting property that if there exists a link from a role node R_1 to another role node R_2, then the firing of R_1 induces synchronous firing in R_2. We refer to such nodes as *phasic* (or ρ) nodes.

The *enabler* (?) node associated with a focal-cluster may be viewed as an "initiate query" or "initiate activity" node. In contrast, *collector* nodes (+ and –) associated with a focal-cluster indicate the outcome of a query or of other activity pertaining to the focal-cluster. In particular, the activation of the + collector indicates a positive response to a query or signals a successful completion of some activity, and the activation of the – collector indicates a negative response to a query or signals an unsuccessful completion of some activity.

Inference occurs via the propagation of activity between focal-clusters. The links between the enabler nodes and role nodes of interconnected focal-clusters allow queries posed in one focal-cluster to propagate to other focal-clusters. Since role nodes are *phasic* nodes, the firing of linked role nodes in interconnected focal-clusters get synchronized.

This allows the propagation of bindings from one focal-cluster to another.

A query is communicated to a focal-cluster by activating its enabler node and binding its role nodes to appropriate role fillers. With reference to Figure 3(a), the query "Can block B be placed on the table, given that currently B is stacked on top of A?" is communicated by activating `?:ACHIEVE`, and synchronizing the firing of `ACHIEVE.I` and `+:On`; the firing of `ACHIEVE.G` and `+:OnTable`; the firing of `On.x`, `OnTable.x` and `?:B`; and that of `On.y` and `?:A`. The activity in the `ACHIEVE` cluster propagates to the `RECALL` cluster, resulting in the query "Is there some action which led to `OnTable(B)` from the precondition `On(B,A)`."

Fact structures attached to a relational focal-cluster encode specific instances of that relation. If the query active at a focal-cluster matches an attached fact, the fact becomes active and, in turn, activates the positive collector of the relation's focal-cluster, and binds each of relation's role nodes to the entity filling these roles in the fact (via synchronous firing). A neurally plausible model of how this might happen in the brain is described in [14].

In the case of `RECALL`, the facts structures attached to this focal-cluster represent episodic memories of specific PAC events. With reference to Figure 3(a) the activation of the query "Is there some action which led to `OnTable(B)` from the precondition `On(B,A)`" matches the memorized PAC event depicted in the figure, and leads to this PAC event becoming active (i.e., recalled). This in turn results in the activation of `+:RECALL`, and the synchronous firing of the unbound role `RECALL.A` with the representation of `Unstack(B,A)`. In effect, this leads to the system recalling that the action `Unstack(B,A)` performed in a state where (the precondition) `On(B,A)` was true, lead to a state where `OnTable(B)` (the consequence) was true.

Notice that an important feature of the system consists of its ability to treat a relational instance as a role-filler (e.g `ACHIEVE.I` $\equiv$ `On(B,A)`). In order to support this requirement, SHRUTI allows for two levels of temporal synchrony. Bindings between standard role nodes and entity/type nodes are represented within a rapid *minor* oscillatory cycle, while bindings between specialized role nodes and relational instances are encoded within a slower *major* oscillatory cycle. In Figure 3(b) the binding between `On.x` and `?:A` is realized by the synchronous firing of these nodes within each minor cycle, while the binding between `?:On` and `RECALL.P` is realized by the the synchronous firing of these nodes within a major cycle.

The simple schema described above consisting of the `ACHIEVE` and `RECALL` focal-clusters acting in concert with an episodic memory system can retrieve previously memorized precondition-action-consequent (PAC) tuples. Thus this schema can be construed as a *proto-planner* capable of constructing one-step "plans." In the next section, we explain how the schema presented in Section 2 is capable of searching for *sequences* of actions, therefore constituting the next 'stage of development' of this proto-planner.

3.2 The planning schema in SHRUTI

Figure 4 shows how the planning schema described in Section 2 has been implemented using SHRUTI's representational machinery. It is easy to see how the focal clusters of this schema can be mapped to the elements of figure 1 (the CURRENT GOAL block has been realized with two clusters, `PLAN` and `SUBGOAL`).

The functioning of this schema is very similar to the more abstract description given in Section 2. Hence, we shall use the same example adopted earlier to illustrate how the schema can perform a basic form of *planning as search*.

Let us assume that the memory of the agent (represented only abstractly in the figure) contains the two PAC facts

(P_1, A_1, C_1) = ({ `On(B,A)` }, `Unstack(B,A)`, { `OnTable(A,B)` })
(P_2, A_2, C_2) = ({ `OnTable(A,B)` }, `Stack(A,B)`, { `On(A,B)` })

(where { `OnTable(A,B)` } is equivalent to {`OnTable(A)`,`OnTable(B)`}) and that the PERCEPTION block, when queried with input P activates the + or the – collector depending on whether the event currently bound to P is true or false in the observed state of the world.

The schema is invoked by activating the `PLAN` cluster's enabler ('?') node and by binding its role node G to the relational instance expressing the current goal (`On(A,B)` in this example). After initialization, activity propagates upwards along links to clusters `SUBGOAL` and `RECALL`. After few major cycles, `?:RECALL` is activated, with role C firing in synchrony with the current goal {`On(A,B)`}. The second PAC event (P_2, A_2, C_2) present in episodic memory matches the activity in the cluster and is retrieved. Consequently, `+:RECALL` becomes active, and the roles A and C are instantiated with actions A_2 = `Stack(A,B)` and relational instance P_2 = {`OnTable(A,B)`}, respectively (the

focal clusters corresponding to predicate 'OnTable' and action 'Stack' are not shown in the figure). The activity of the role P reaches the COMPARE cluster, and the perceptual system is asked to verify whether `OnTable(A,B)` is currently true in the world state. The answer is 'No', and hence, `-:Compare` becomes active.

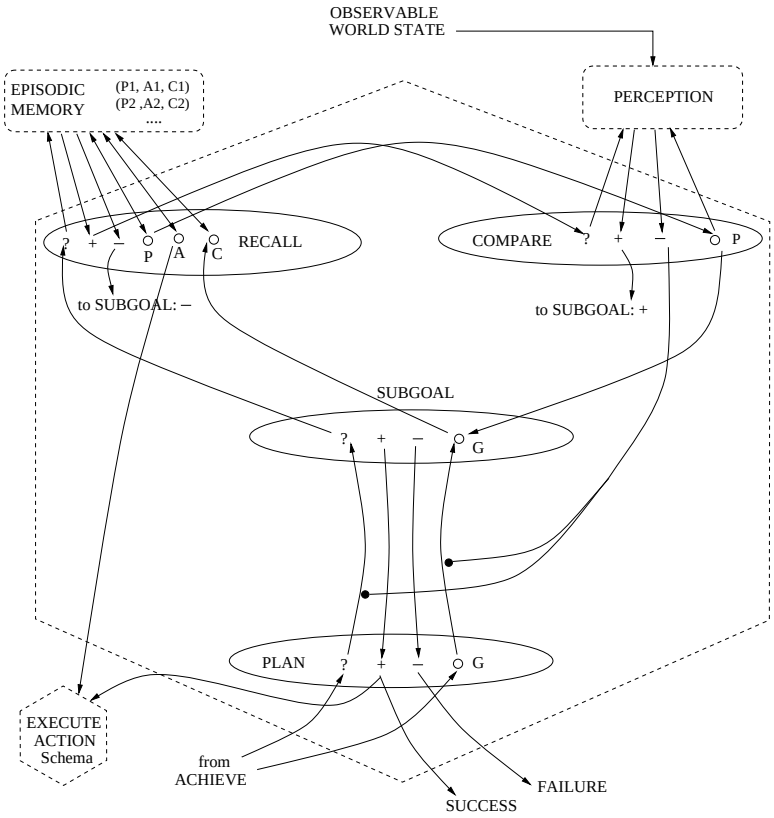

Figure 4: A diagram showing the connectionist structure of the planning schema.

The above immediately leads to the inhibition of the links from PLAN to SUBGOAL, and this blocks the propagation of the query PLAN(`On(A,B)`) through the schema. Simultaneously, activity from `Compare` reaches SUBGOAL: the role node G starts firing in synchrony with P, which was temporally bound to the relational instance {`OnTable(A,B)`}: hence, this becomes the new (sub)goal of the schema, and its focus of attention. Activity from `?:SUBGOAL` reaches `?:RECALL` again, while role node C starts firing in synchrony with G. This leads to the retrieval of the first PAC event, and hence, the precondition P_1 = {`On(A,B)`} gets bound to role P and role A is bound to the action instance A_1 = `Unstack(B,A)`. These bindings are in turn propagated to `Compare`.

The positive outcome of the comparison leads to the activation of the `+:COMPARE` collector, which communicates to PLAN that the search has terminated with success. After action A_1 is executed, the original query (still present in the system) causes the schema to restart with goal G={`On(A,B)`} since this goal has not been achieved.

The subsequent flow of activation is identical to the initial part of the sequence described above, except that when P_2 = {`OnTable(A,B)`} is compared with the current state, the outcome is positive, and the currently active action (`Stack(A,B)`) is executed. This achieves the initial goal G, and terminates the activity.

4 Simulation results

The planning schema described in Section 3 has been realised and tested using the "SHRUTI Agent Simulator" software written in Java. The example described in the previous section has been used to test the functioning of the schema. Figure 5 contains two 'screen captures' of an actual simulation. The top half of the screen contains the graphical

representation of the schema control structure with the related PAC events, while the lower part shows the activity in the predicates '`On(x,y)`' and `OnTable(x)`' with associated facts, and the type hierarchy for the blocks (entities) `A` and `B`.

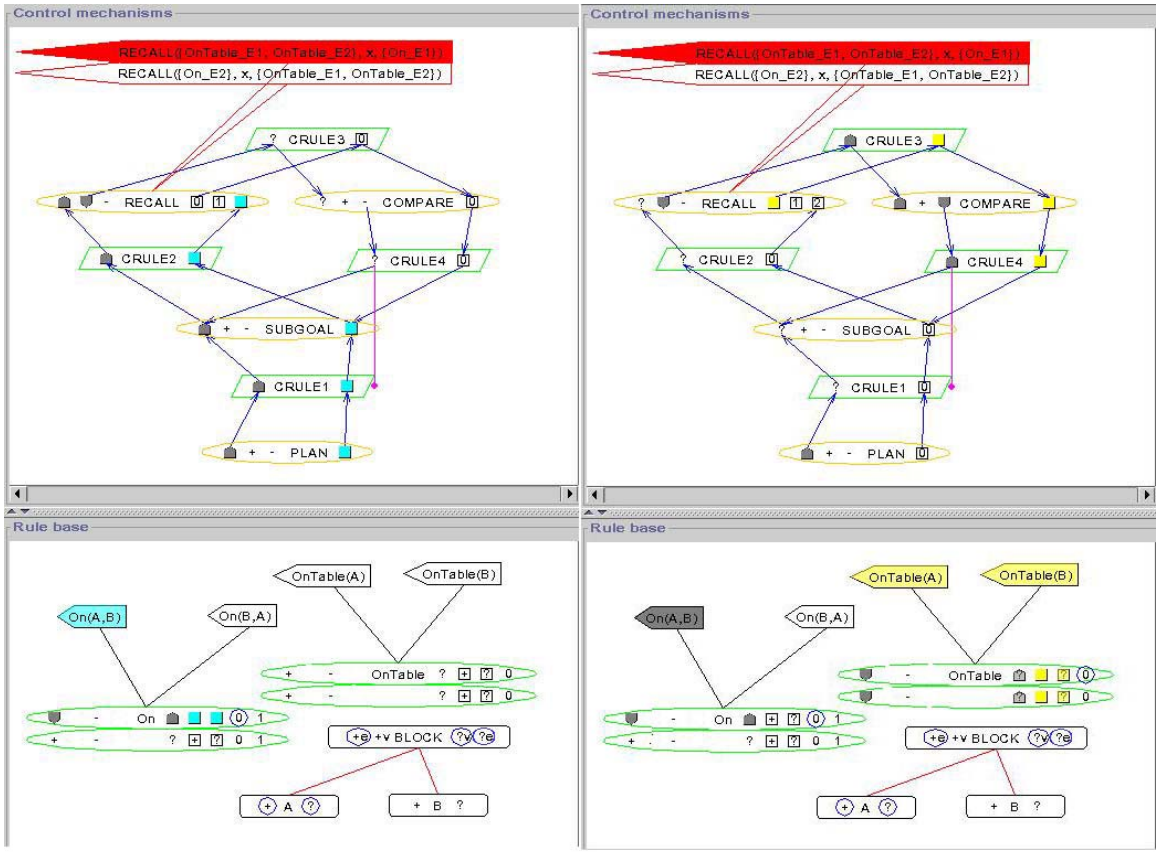

Figure 5: Two screen captures taken during an actual simulation.

In the capture on the left, the top PAC fact has become active because of the initial query '`PLAN(On(A,B))`', which has been propagated upwards and has led to the query '`RECALL(`x`,`y`,On(A,B))`'. As a consequence, the two preconditions {`OnTable(A)`,`OnTable(B)`} are about to become active, and will be propagated to the `COMPARE` cluster, where they will be matched against the current state[2].

The capture on the right is a snapshot of the situation immediately following the negative outcome of the (simulated) comparison. Notice how the flow of activity going from the `PLAN` cluster to the `RECALL` cluster has been interrupted by cluster `CRULE4` to allow the new goal {`OnTable(A)`,`OnTable(B)`} to make its way through to cluster `RECALL`.

The two situations depicted can be mapped to two specific 'time-points' of Figure 6, which shows a detailed trace of activation for this example of simulation. The two 'snapshots' of Figure 5 correspond, respectively, to time-points α and β in the diagram.

[2]The perceptual task of comparing world states was simulated by manually activating the $+$ or $-$ collector of the cluster 'Compare' as appropriate.

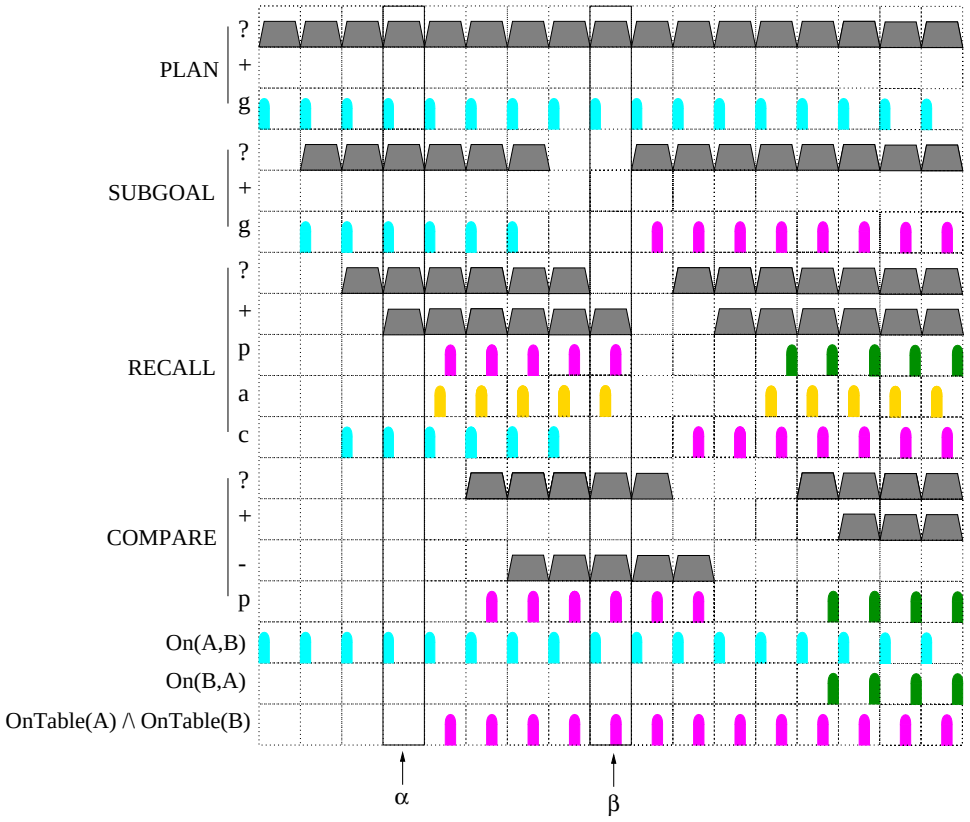

Figure 6: Node activation trace of the simulation.

5 Discussion

The work described in this paper is part of a larger effort whose goal is to develop a neurally plausible architecture for reasoning, remembering, planning, and decision making. This paper presents progress along an important dimension of this ongoing effort. Perhaps the most interesting aspect of this work is the demonstration that general purpose cognitive faculties such as episodic memory, semantic memory and perception can be harnessed to solve a restricted – but interesting – subclass of the planning problem in a transparent manner. The proposed planning schema uses a simple control structure that works in conjunction with episodic memory, semantic memory, and perception to produce the requisite planning behavior.

The planning schema discussed in this paper is limited in a number of ways, but as discussed below, this schema can be extended into a much more powerful and expressive planning system by incorporating additional control and memory structures, and by leveraging the full representational and expressive power of SHRUTI.

The proposed planning schema is susceptible to getting trapped in deadends. As the system searches for a path from the goal to the initial state, it can get caught in a state that subsumes a set of conditions D which do not match the consequent of any PAC event in memory. There is, however, a simple three part solution to this problem. First, the agent detects that it has reached a deadend state. Doing so is straightforward since such a state is signaled by the activation of `-:RECALL`. Second, the agent memorizes that this path leads to a deadend in the context of the current problem. It can do so by memorizing the following episodic memory trace: "when trying to achieve the goal G, instantiating a subgoal D leads to a deadend." The representational machinery required to encode such DEADEND "events" is similar to that required to encode PAC events (like PAC events, the episodic memory trace of DEADEND events also involves role-fillers that are partial state-descriptions, specified by sets of conditions). Third, the agent restarts the search and at each step in the search process retrieves both PAC and DEADEND events that match the current subgoal. Any retrieved PAC event that are counterindicated by retrieved DEADEND event are ignored. Notice

that the memorization of deadends prunes the potential search space. With sufficient practice, the agent may memorize a large number of deadend events, and this may enable it to carry out a highly efficient search.

Another limitation of the proposed planning schema is that it needs to traverse the same paths through the state space several times during the course of finding a plan (refer to Figure 2). If however, the agent could remember the path traversed from the current goal to the initial state, it would not have to rediscover the same plan subsequences many times over. Note that remembering such a path can be viewed as memorizing a sequence of PAC events. Learning of event sequences is a well-known property of episodic memory, but it remains to be seen how the process of such on-line memorization of event sequences can be fully integrated with the on-line retrieval of previous episodic memories. Working memory mechanisms can also play a complementary role in such on-line memorization. Our current research addresses the functioning of episodic memory [14, 15] as well as that of working memory [23], and we hope that the development of powerful episodic and working memory models will directly benefit future work in the development of planning schemas.

With the incorporation of path learning mechanisms, an agent would have to find a path from the goal state to the initial state only once, and plan execution would involve traversing the memorized sequence of PAC events in the reverse order and executing the actions associated with each PAC event in the sequence.

Since the proposed planning schema and its extensions are fully compatible with other representational components of SHRUTI, the full range of knowledge representation and reasoning capabilities of SHRUTI can be leveraged during planning. This includes the representation of, and reasoning with, commonsense (semantic) knowledge, causal models, type hierarchies, context-sensitive prior probabilities of events conditioned by the type of role-fillers, and the utility (or value) associated with partial world-states. Thus general purpose domain knowledge as well as planning specific knowledge about operators can be seamlessly combined to support complex planning involving not just memory retrieval, but also inference.

The representation of types offers an elegant means of achieving plan abstraction. For example, episodic memory acting in concert with a type hierarchy can enable a PAC event such as PAC({`OnTable(A,B)`}, `Stack(A,B)`, {`On(A,B)`}) to not only solve the goal `On(A,B)`, but also goals such as `On(C,D)`, where C and D are objects of the same type as A and B, respectively. This will enable a PAC event such as PAC({`OnTable(A,B)`}, `Stack(A,B)`, {`On(A,B)`}) to be generalized to a PAC event such as

`x:Block,y:Block` · PAC({`OnTable(x,y)`}, `Stack(x,y)`, {`On(x,y)`})

In the current proposal there is no explicit representation of "delete lists" in the encoding of PAC events. For now, we are assuming that during search, deletions of facts in the internal representation of the world state occur as a result of inferences drawn by SHRUTI. These inferences occur as a result of interactions between (i) facts added to the world state by retrieved PAC events and (ii) the causal domain knowledge encoded in SHRUTI (note that such "add lists" are currently encoded in PAC events). An example of such an inference-based deletion would be the deletion of the fact `OnTable(a)` after the addition of the fact `On(a,b)`. Of course, deletions of facts resulting from the actual *execution* of an action will not have to be inferred since these will be known to the agent as a result of perception.

The functionality of the current planning schema is also limited by its inability to make use of plan decomposition. Imagine that the agent is trying to find a plan for the goal $G_1 \& G_2$ given the world state I, and the agent has the PAC events PAC(I', a_1, G_1) and PAC(I'', a_2, G_2) in its episodic memory (let us assume that I' and I'' hold in state I, and that I'' also holds in the state resulting from performing action a_1 in state I). The planning schema described in this paper will be unable to solve the composite goal G_1 & G_2, even though it will be able to solve each of the subgoals G_1 and G_2 if presented individually. In order to deal with plan decomposition, the schema must (i) recognize that it can solve one of the subproblems using one of the PAC facts, (ii) pick the subproblem to be solved, (iii) note down the subproblem that it is deferring for now, (iv) find a solution to the selected subproblem, (v) shift attention back to the deferred subproblem, and (vi) solve the deferred subproblem. A connectionist implementation of this algorithm would require a more complex schema (control structure) than the one described in Section 3, together with the ability to remember deferred goals. The memory of deferred goals can take the form of working memory (if deferred goals have to be remembered for a few seconds) or episodic memory (if the goals have to be remembered over longer time periods).

Another area of ongoing research that is of direct relevance to the work described here concerns the representation of complex action schemas and plans using SHRUTI's representational machinery [17]. In past work, we have shown that parameterized schemas capable of dealing with partially ordered actions, conditional actions, concurrent and

iterative actions, as well as compositional and hierarchical actions can be encoded using SHRUTI's representational machinery. This makes us confident that the more complex control structures required for encoding more sophisticated planning schemas would not present an insurmountable problem.

The planning system resulting from this line of work would possess an interesting and powerful set of attributes that typically do not co-occur in a single planning system. In particular, the system could exploit the lower branching-factor of a backward-chaining regression search which is one of the key advantages of causal-link planners [9, 7]. It would use heuristic, domain specific as well as generic knowledge and inferential mechanisms to guide its state space search when two or more possible PAC events are simultaneously active (as in [8, 5, 4]). The extended system would interleave planning and execution with perception, it would handle uncertainty and contradictory information, and would be able to perform probabilistic reasoning using weighted links (e.g. see [22, 2] or [21] for a useful account).

A key issue that remains open is the learning of appropriate control structures. We are investigating this question within the frameworks of spike-timing dependent synaptic plasticity [25, 19] and recruitment learning [3, 10, 20] based on long-term potentiation [6, 13].

Acknowledgments

This work was partially funded by NSF grants 9720398 and 9970890.

References

[1] F. Bacchus and Y. W. Teh. Making forward chaining relevant. In *Proceedings of the Fourth International Conference on AI Planning Systems*, pages 54–61, June 1998.

[2] A. L. Blum and J. C. Langford. Probabilistic planning in the graphplan framework. In *Proceedings of the AIPS98 Workshop on Planning as Combinatorial Search*, pages 8–12, June 1998.

[3] J. Feldman. Dynamic connections in neural networks. *Bio-Cybernetics*, 46:27–39, 1982.

[4] P. Haslum H. Geffner. Admissible heuristics for optimal planning. In *Proceedings of the 5th Internat. Conf. of AI Planning Systems (AIPS 2000)*, pages 140–149, Breckenridge, Colorado, April 2000. AAAI Press.

[5] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[6] R. C. Malenka and R. A. Nicoll. Long-term potentiation - a decade of progress? *Nature*, 285:1870–1874, 1999.

[7] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Nineth National Conference on AI*, pages 634–639, July 1991.

[8] D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the International Joint Conference on AI*, pages 968–973, August 1999.

[9] J. S. Penberthy and D. Weld. Ucpop: A sound, complete, partial order planner for adl. In *Proceedings of the Third International Conference on Principles of Knowl. Repr. and Reasoning*, pages 103–114, October 1992.

[10] L. Shastri. *Semantic Networks: An evidential formalization and its connectionist realization*. Los Altos/Pitman Publishing Company, London, 1988.

[11] L. Shastri. Advances in SHRUTI - a neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Applied Intelligence*, 11, 1999.

[12] L. Shastri. Hybrid information processing in adaptive autonomous vehicles. In S. Werter and R. Sun, editors, *Types and Quantifiers in Shruti — a connectionist model of rapid reasoning and relational processing*, pages 28–45. Springer-Verlag, Heidelberg, 2000.

[13] L. Shastri. A biological grounding of recruitment learning and vicinal algorithms. In J. Austin, S. Wermter, and D. Wilshaw, editors, *Emergent neural computational architectures based on neuroscience*. Springer-Verlag, 2001.

[14] L. Shastri. A computational model of episodic memory formation in the hippocampal system. *Neurocomputing*, 38-40:889–897, 2001.

[15] L. Shastri. From transient patterns to persistent structure: A model of episodic memory formation via cortico-hippocampal interactions. In *submitted*, 2001. http://www.icsi.berkeley.edu/ shastri/psfiles/shastri_em.pdf.

[16] L. Shastri and V. Ajjanagadde. From simple associations to systematic reasoning. *Behavioral and Brain Sciences*, 16(3):417–494, 1993.

[17] L. Shastri, D. Grannes, S. Narayanan, and J. Feldman. A connectionist encoding of parameterized schemas and reactive plans. In G. Kraetzschmar and G. Palm, editors, *Hybrid Information Processing in Adaptive Autonomous Vehicles*. Springer-Verlag, 1997.

[18] L. Shastri and C. Wendelken. Seeking coherent explanations - a fusion of structured connectionism, temporal synchrony, and evidential reasoning. In *Proceedings of the Twenty-Second Conference of the Cognitive Science Society*, Philadelphia, August 2000.

[19] S. Song, K. Miller, and L. Abbott. Competitive hebbian learning through spike-timing dependent synaptic plasticity. *Nature Neuroscience*, 3:919–926, 2000.

[20] L. Valiant. *Circuits of the mind*. Oxford University Press, 1994.

[21] D. S. Weld. Recent advances in ai planning. *AI Magazine*, 20(2):93–123, 1999.

[22] D. S. Weld, C. R. Anderson, and D. E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the 15th National Conference on AI*, pages 897–904, July 1998.

[23] C. Wendelken. The role of mid-dorsolateral prefrontal cortex in working memory: a connectionist model. In *submitted*, 2001.

[24] C. Wendelken. Shruti-agent: A structured connectionist model of decision-making. In *Proceddings of the Fourth Annual Conference on Cognitive Modeling*, August 2001.

[25] C. Wendelken and L. Shastri. Probabilistic inference and learning in a connectionist causal network. In *Proceedings of the Second International Symposium on Neural Computation*, May 2000.