Piagetian Adaptation Meets Image Schemas: The Jean System

Yu-Han Chang[1], Paul Cohen[1], Clayton Morrison[1], and Robert St. Amant[2]

[1] USC ISI
4676 Admiralty Way
Marina del Rey, CA 90292
{cohen, ychang, clayton}@isi.edu
[2] North Carolina State University
stamant@ncsu.edu

Abstract. Jean is a model of early cognitive development based loosely on Piaget's theory of sensori-motor and pre-operational thought [1]. Like an infant, Jean repeatedly executes schemas, gradually extending its schemas to accommodate new experiences. Jean's environment is a simulated "playpen" in which Jean and other objects move about and interact. Jean's cognitive development depends on several integrated functions: a simple perceptual system, an action-selection system, a motivational system, a long-term memory, and learning methods. This paper provides an overview of Jean's architecture and schemas, and it focuses on how Jean learns schemas and transfers them to new situations.

1 Introduction

Jean is both a synthesis of ideas about cognitive development and the foundations of concepts, and an integrated software system that implements perception, action, learning and memory. Most of this paper is devoted to the Jean system, so let us begin with the underlying ideas. From Piaget we borrow the ideas that children learn some of what they know by repeatedly executing schemas, and executing schemas is in a sense rewarding, and some new schemas are modifications or amalgamations of old ones [1]. The Image Schema theorists [2–6] promote the ideas that primitive schemas are encodings or redescriptions of sensorimotor information; and these schemas are semantically rich, general, and extend or transfer to new situations, some of which have no salient sensorimotor aspects. Another idea, represented by various authors, is that semantic distinctions sometimes depend on dynamics — how things change over time — and so schemas should have a dynamical aspect [7–10].

Jean will test several conjectures about developmental AI: First, it will be possible to provide a relatively small, core set of schemas and a general algorithm to learn others as they are needed or indicated by experience. We are betting on a compositional account of knowledge, in which newly learned schemas are assembled from previously learned and appropriately modified components. Second, schemas will have to be more than the declarative, logical structures proposed by AI researchers over the decades; they will have to include behavior-generating controllers, dynamic maps, deictic variable bindings, and causal theories; these components will not all develop simultaneously.

Third, the generality of these schemas provides a basis for the effective *transfer* of knowledge learned in one task to a new, related task.

2 The Image Schema Language

Image schemas are Jean's elementary and innate representations, and much of Jean's design follows from these representation commitments, so we begin our description of Jean with them. Image schemas are representations that are "close" to perceptual experience. Some authors present them as re-descriptions of experience [5]. Their popularity is due to their supposed generality and naturalness: Many situations are naturally described in terms of paths, up-down relations, part-whole relationships, bounded spaces, and so on. Even non-physical ideas, such as following an argument, containing political fallout, and feeling "up" or "down," seem only a short step from image-schematic foundations [2, 3].

These ideas are attractive but vague, and there have been attempts to formalize them, such as the Image Schema Language (ISL) [11]. The authors found it necessary to distinguish three kinds of image schema. *Static* schemas describe unchanging arrangements of physical things; *dynamic* schemas describe how the environment changes; and *action* schemas describe intentional aspects of static and dynamic schemas. Thus, the action schema *approach-object* includes a path (a static schema) but gives it the intentional gloss that it is the path one intends to follow (or is following). *Moving* might be intentional or it might simply be the result of force acting on an object. Both cases involve a path, but the latter is described by a dynamic schema, not an action schema.

2.1 Static Schemas

As represented in ISL, static image schemas are objects, in the sense of the object-oriented data model. Each schema has a set of operations that determine its capabilities. For example, operations for a basic container schema include putting material into a container and taking material out. Each schema also has a set of internal slots that function as roles in a case grammar sense [12]. Slots permit image schemas to be related to each other through their slot values. For example, the contents of a container can be other image schemas.

To illustrate static schemas in ISL, it will be helpful to walk through an example, which we take from our work on representing chess patterns. Consider a chess board in which the Black queen has the White king in check. In image schema terms, we say that there exists a path from the queen to the king. In ISL, we generate a path schema, which contains a set of locations, as shown in Through a mechanism called *interpretation*, we can substitute one kind of schema for (part of) another in ISL, which allows us to make the locations along the paths be *container* schemas. Then, by setting the capacity of these containers to one piece, we capture the idea that the location (interpreted as a container) is "full" if it is occupied by a piece. A blocked path is then a path with at least one full container/location.

2.2 Dynamic and Action Schemas

If Jean lived in a chess game, we might choose to represent the dynamics of the environment as a sequence of static schemas, like fluents in the situation calculus: Some schemas describe the environment until an instantaneous chess move happens, then others do. Jean's environment changes continuously, and there is much structure in *how* it changes, so we have adopted the following representation for dynamic schemas and action schemas: Both are finite state machines, the states of which are themselves composed of image schemas. For example, to catch a simulated cat, Jean must sneak up on the cat slowly and then, when it gets near the cat, Jean must pounce. As illustrated in Figure 1, the action schema for catching a cat must have at least these two states (and actually, a couple more). Each state contains several static and dynamic schemas. For instance, state s_3 of Figure 1 comprises an object schema that binds its deictic variable to the cat, and a near-far schema that binds its two deictic variables to the robot (i.e., Jean) and the cat, respectively. The near-far schema also asserts that the cat is more than six units away from the robot. s_3 is associated with two action schemas, fast-approach-object (F) and slow-approach-object (S), represented as arcs leaving s_3.

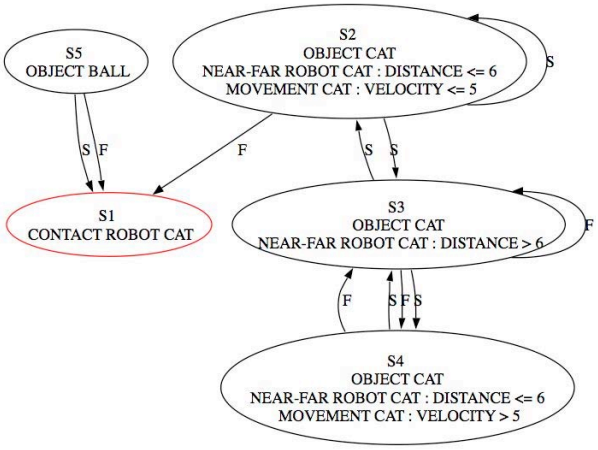

Fig. 1. A learned composite action schema for catching a cat

Composite action schemas like catching a cat are called *gists* because they involve story-like sequences of actions and states and they abstract away many of the details of particular instances.

In addition to the familiar, declarative components of schemas, dynamic and action schemas contain dynamic *maps*, which describe continuous, smooth changes in state variables. Dynamic maps have three functions: They let Jean estimate when changes will occur, they tell Jean when its actions are, or are not, likely to achieve its goals, and they help Jean find the boundaries between states. These functions are elaborated in Section 4.

Action schemas also contain *controllers* that control behavior. For instance, Jean's schema for *approaching* binds its variables to the approaching object (typically Jean) and the approached object, a map that shows how distance between the objects changes over time, and a controller that makes Jean move toward the location of the object it is approaching. Eventually, schemas will also be augmented with causal relations (see Sec. 7).

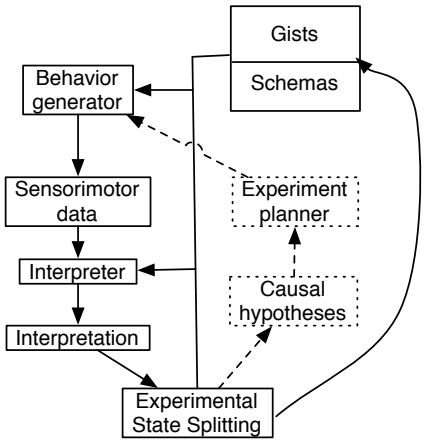

Fig. 2. The main functional components of Jean

3 The Architecture of Jean

The main functional components of Jean are illustrated in Figure 2. Over time, Jean learns new schemas and gists, all represented in the Image Schema Language (ISL). When Jean has a goal, such as catching a simulated cat, it retrieves an appropriate gist and runs its controller, which means taking the actions that produce transitions between states. The component of Jean that runs schemas is called the *behavior generator*. Exercising schemas produces sensory data, and lots of it. The job of the *interpreter* is to retrieve and instantiate (i.e., bind the deictic variables of) schemas from the repository. Interpretation also updates conditional probabilities associated with state transitions within gists. Jean will try to interpret the sensory data in terms of the schemas in the gist it is running; for example, if it is trying to catch a cat, then it will prefer to interpret the sensory data as matching the states in Figure 1. Sometimes, though, the fit is poor, and another schema in the repository does a better job of explaining the sensory data. And sometimes, it is necessary to construct a novel static, dynamic or action schema, as described in the next section.

4 Learning: Experimental State Splitting

Jean learns new schemas in two ways, by *composing* schemas into gists and by *differentiating* states in schemas. Both are accomplished by the Experimental State Splitting (ESS) algorithm. The basic idea behind Experimental State Splitting (ESS) is simple. The algorithm starts with a minimal state model of the world, in which it has only one all-encompassing state. This model is modified as the agent explores its world, so that it becomes more predictive of some measure of observed action outcomes.

For a general developmental account we want a general measure, not a task-specific one. To accord with the idea that learning is itself rewarding, this measure might have something to do with the informativeness or novelty or predictability of states. In Jean, the ESS algorithm uses a measure we call *boundary entropy*, which is the entropy of the next state given the current state and an intended action. Initially, when there is only one state in the model, the entropy will always be zero. One way to drive Jean toward more states, and, thus toward states that have boundary entropy, is to have a goal state in addition to the initial (non-goal) state. At any moment in time, the agent is in one of these two states, and each state-action pair generates some probability distribution over the set of possible next states. ESS calculates the entropy of this distribution and uses it as a state splitting criterion.

In general, Jean is driven by ESS to modify its world model by augmenting existing states with new states that reduce the boundary entropies of state-action pairs. This augmentation is achieved by splitting an old state into two (or more) new states based on distinguishing characteristics. For example, if an agent is navigating a city intersection, it might decide that "green light" is an important characteristic because, where it observes a green light, it is much less likely to be involved in crashes. Thus, its non-goal state would be split into two states: (1) a green light is observed and its goal has not yet been attained, and (2) neither a green light is observed nor its goal has been obtained. The state machine thus grows over time as the agent adds more attributes to its state descriptions.

As Jean interacts with the world, it counts the transitions between the states via particular actions. If the developing state machine model is Markov, then the model is a Markov Decision Process (MDP), which Jean can solve for the optimal policy to reach its goal state. In this way, the developing model can be used for planning. However, it is worth pointing out that since Jean operates in a continuous environment using fairly general action schemas, its world model is more like a semi-Markov Decision Process (SMDP), where each action results in a transition between states according to some probability distribution.

If Jean lives always in one environment with one set of goals, then ESS will eventually produce optimal policies for the environment. However, the purpose of the Jean project is not to produce optimal policies for each task and variant of Jean's environment, but, rather, to explain how a relatively small set of policies may quickly *accommodate* (as Piaget called it) or *transfer* to new tasks and environments. Our approach to this problem is to extract *gists* from policies. Gists are like policies, in that they tell Jean what to do in different situations, but they are more general because they extract the "usual storyline" or essential aspects of policies. We claim that these essential aspects are typically the *causal relationships* that govern actions and effects in the environment.

If an agent can identify and learn these causal relationships, then it should have a very good idea of how its actions affect the world and how act to achieve its goals, *even in novel situations*. We will discuss gists and transfer further in Section 5.

4.1 An Example

Figure 1 illustrates a gist for approaching and contacting a simulated cat. In the scenario in which this gist was learned, the cat is animate, capable of sitting still, walking or running away. The cat responds to Jean. In particular, if Jean moves toward the cat rapidly, the cat will run away; if Jean approaches slowly, the cat will tend to keep doing what it is doing. Because of these programmed behaviors, there is uncertainty in Jean's representation of what the cat will do, but there is a general rule about how to catch the cat, and it can be represented in a gist: The only way to catch the cat is to first get into state s_2 (Fig. 1), where the cat is nearby and not moving quickly, and then to move fast toward the cat, reaching state s_1. All other patterns of movement leave Jean in states s_3 or s_4. This corresponds to the strategy of slowly sneaking up to the cat and then quickly pouncing on it to catch it.

To learn a gist like the one in Figure 1, Jean repeatedly retrieves action schemas from its memory, runs the associated controllers, producing actions, specifically slow and fast movement to a location; assesses the resulting states, and, if the transitions between states are highly unpredictable, Jean splits states to make the resulting states more predictable.

In fact, the three states, s_2, s_3 and s_4 were all originally one undifferentiated state in which Jean moved either fast or slowly toward the cat. Jean's learning history — the distinctions it makes when it splits states — begins with a single, undifferentiated non-goal state. Then, Jean learns that the type of object is an important predictor of whether or not it can catch the object. Balls are easy to catch, whereas cats are hard to catch. From here, ESS recognizes that distance also influences whether or not it can catch a cat. Starting near the cat, a fast-approach-object (F) action will often catch the cat, whereas this action will not usually catch the cat from further away. Thus, ESS splits on distance with a threshold of 6, where $<= 6$ is considered near, and > 6 is far. Finally, ESS may notice that even when Jean is near the cat, sometimes it does not succeed in catching the cat. This might be because the cat is already moving away from the agent with some speed. Thus, ESS may do a final split based on the velocity of the cat. This process leads to the states s_2, s_3, s_4 and s_5 that we see in Figure 1.

4.2 The Algorithm

We give a formal outline of the ESS algorithm in this section. We assume that agent receives a set of features $F^t = f_1, \ldots, f_n$ from the environment at every time tick t; these features could be sensor readings, for example. We also assume that Jean is initialized with a goal state s_g and a non-goal state s_0. S_t is the entire state space at time t. A is the set of all actions, and $A(s) \subset A$ are the actions that are valid for state $s \in S$. Typically $A(s)$ should be much smaller than A. $H(s_i, a_j)$ is the boundary entropy of a state-action pair (s_i, a_j), where the next observation is one of states in S_t. A small boundary entropy corresponds to a situation where executing action a_j from state s_i is

highly predictive of the next observed state. Finally, $p(s_i, a_j, s_k)$ is the probability that taking action a_j from state s_i will lead to state s_k.

For simplicity, we will focus on the version of ESS that only splits states; an alternative version of ESS is also capable of splitting actions and learning specializations of parametrized actions. The ESS algorithm follows:

- Initialize state space with two states, $S_0 = \{s_0, s_g\}$.
- While ϵ-optimal policy not found:
  - Gather experience for some time interval τ to estimate the transition probabilities $p(s_i, a_j, s_k)$.
  - Find a state s_i to split that maximizes the boundary entropy score reduction of the split: $\max_{S,A,F,\Theta} H(s_i, a_i) - \min(H(s_{k_1}, a_i), H(s_{k_2}, a_i))$, where s_{k_1} and s_{k_2} result from splitting s_i using feature $f \in F$ and threshold $\theta \in \Theta$.
  - Split $s_i \in S_t$ into s_{k_1} and s_{k_2}, and replace s_i with new states in S_{t+1}.
  - Re-solve for optimal plan according to p and S_{t+1}

The splitting procedure iterates through all state-action pairs, all of the features F, and all possible thresholds in Θ and tests each such potential split by calculating the reduction in boundary entropy that results from that split. This is clearly an expensive procedure. In future we will speed it up with heuristics that limit the Jean's attention to relevant features and state-action pairs. Heuristics to find potential thresholds are discussed next.

4.3 Finding splitting thresholds, learning new dynamic schemas, and other criteria for splitting

Given a candidate schema f for splitting, ESS must find a threshold on which to split the state using that schema. To do this, Jean uses a simple heuristic: States change when several state variables change more or less simultaneously. This heuristic is illustrated in Figure 3. The upper two graphs show time series of five state variables: headings for the robot and the cat (in radians), distance between the robot and the cat, and their respective velocities. The bottom graph shows the number of state variables that change value (by a set amount) at each tick. When more than a set number of state variables change, Jean concludes that the state has changed. The value of the schema f at the moment of the state change is likely to be a good threshold for splitting f. For example, between time period 6.2 and 8, Jean is approaching the cat, and the heuristic identifies this period as one state. Then, at time period 8, several indicators change at once, and the heuristic indicates Jean is in a new state, one that corresponds to the cat moving away from Jean. The regions between these changes become the dynamic maps associated with dynamic and action schemas, and the active schemas in these regions are bundled together into composite dynamic and action schemas such as "s_2: Object : Cat ; Near-Far : Robot, Cat :Distance ≤ 6 ; Movement Cat : Velocity ≤ 5."

This segmentation of the time series helps Jean learn new dynamic schemas. Segments correspond to dynamic *maps* in dynamic and action schemas. As long as Jean is safely within a schema, state variables will change as the schema's maps prescribe. Over time, Jean builds up a statistical model of how state variables are expected to

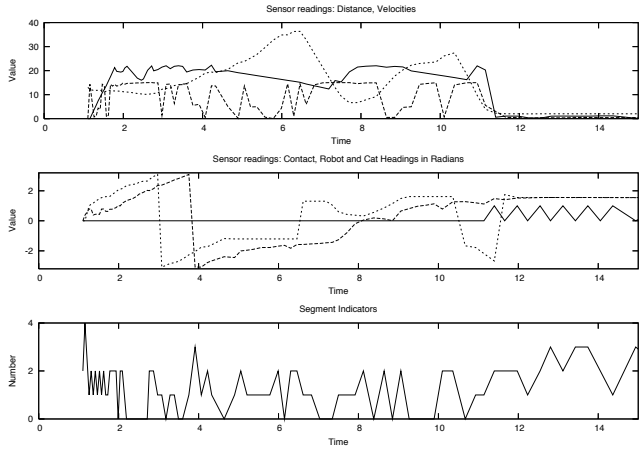

Fig. 3. New states are extracted by cutting multiple time series at places where multiple state variables change simultaneously.

change in a state. If the variance of this model is high, then it suggests the state is a candidate for splitting.

Other useful indicators that states need to be split include rewards (and costs), and repeatability or closure. Reward is fairly straightforward: we wish to split states so that new states will better predict future rewards (rather than simply predicting future states). Repeatability or closure refers to whether actions are easily repeated once executed. For example, picking up and dropping a block onto a table is easily repeated because the action leads to a state where it can immediately be repeated. Such "closed" actions are more easily learned by children, thus we may want closure to be an indicator for splitting actions.

5 Transferring Learning

Although ESS can learn policies for new situations from scratch, we are much more interested in how previously learned policies can accommodate or transfer to new situations. We call transferrable policies *gists*, or generalized summaries of policies. Gists capture the most relevant states and actions for accomplishing past goals. It follows that gists may be transferred to situations where Jean has similar goals. Jean could, of course, try to execute gists without modification in these situations. However, in situations that are not very similar, it may be more effective to execute a gist from which state transition probabilities have been excised. The extra cost of relearning the probabilities may be offset by the cost of *negative transfer*. Sometimes a learned gist can actually inhibit learning of a new gist. For instance, having learned to drive in the U.S. can interfere with learning to drive in the U.K. In cases like this, one wants to keep the general structure of a gist because it describes the states and transitions that occur in both situations, but learn new transition probabilities. By dropping transition probabil-

ities, we may have an easier time finding a policy in the new situation without being burdened by the weight of past experiences.

6 Experiments

We tested Jean's transfer of schemas between situations in a simulated 3-D physical environment. Jean's task is to catch a given target object as quickly as possible. The targets, which we refer to as a "ball" and a "cat," have different dynamics. The ball moves ballistically when Jean makes contact with it (and may be moving at the start of a learning episode), whereas the cat is self-moving and has a preference to not be caught by Jean. In these experiments, the cat runs away if Jean is far away but approaching quickly, or when Jean is very near. Thus, the best way to catch the cat is to sneak up slowly and then quickly pounce upon it. In these experiments, Jean has four innate (not learned) action schemas: fast-approach, slow-approach, stop, and wander. In some experiments there are obstacles, such as walls.

In each trial, Jean has a chance to complete its given task — catch a ball or catch a cat — within a time limit. The time Jean requires to perform its task is the dependent variable, and is expected to decrease as Jean learns to catch its targets. An experimental condition includes 100 trials. The value of the dependent variable, time to catch the target, is smoothed over these trials using a smoothing window of 15 trials. Good learning performance should corresponds to a line that slopes down from left (early trials) to right (later trials).

To evaluate the effect of transfer, we follow an "B vs. A+B" protocol. A and B refer to tasks across which we might observe transfer; for instance, A might be catching a ball, and B, catching a cat. The protocol involves two conditions: The "B" condition involves learning to perform a task, B, whereas the "A+B" condition involves learning task B after learning to perform task A. If the gist for task A transfers, then it will improve some aspect of performance on B, either the initial level of competence on B (before learning) or the rate of learning to perform B.

Jean was tested on three tasks: **A:** Catch a ball in an unobstructed room; **B:** Catch a cat in an unobstructed room; **C:** Catch a cat in a room with obstacles such as additional walls. Here Jean must learn to avoid bumping into the walls while chasing the cat. We constructed five experimental conditions:

B. Catch the cat without any prior training.
A+B. First learn to catch a ball, then learn to catch a cat.
C. Catch the cat in the obstructed room without any prior training.
B+C. First learn to catch a cat in an unobstructed room, then learn to do this in a room with additional interior walls.
B+A. First learn to catch a cat in an unobstructed room, then learn to catch a ball in the same room.

The data presented here are a representative sample of system performance in the conditions in which we expect positive or negative transfer.[3]

[3] We are currently conducting tests for statistical significance, based on the methods of Piater et al. for comparing learning curves [13]; these will be ready in time for the camera-ready

Guys, calling the horizontal axis Time is misleading. Can we call it Trials, instead?

Figure 4 shows Jean learning to catch a cat in an unobstructed room. The dotted line represents Jean learning to catch a cat without any prior knowledge (i.e., condition **B**). We see little improvement in the performance measure as Jean acquires more experience. The bold line represents Jean's performance on the catch-a-cat task after learning to catch a ball. Clearly Jean is able to transfer some of its knowledge from catching a ball to the task of catching a cat. In particular, it has already established a preference ordering on its action schemas, whereby it prefers fast-approach and slow-approach over stop and wander, since these result in shorter ball-catching times. Thus, in the **A+B**, condition, Jean does not explore the stop and wander actions, but directly accommodates its ball-catching gist to cat-catching. It is quickly able to identify the proper state to split (i.e., split based on the NEAR-FAR schema), and learns the optimal policy for catching a cat. This is clearly represented by the large drop in catching time seen around time 20.

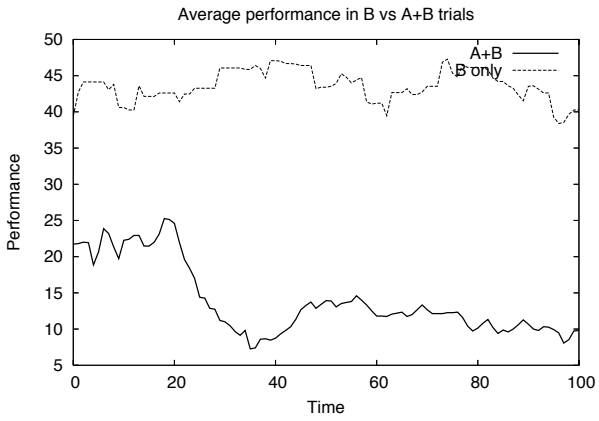

Fig. 4. Graph showing average performance over time in the B versus A+B regimes. Performance is measured as the time taken for the robot to catch the target object. If in some trial the object is not caught within the time limit, then the time limit value is used as that trial's performance.

A similar benefit of prior learning is seen in Figure 5. Here, the dotted line shows Jean's performance while attempting to learn from scratch how to catch a cat in an obstructed room (condition **C**). The bold-face line shows the performance given that Jean has already learned how to catch a cat in an open room (condition **B+C**). The

submission. These are particularly important since there is a high degree of variability inherent in the simulation domain.

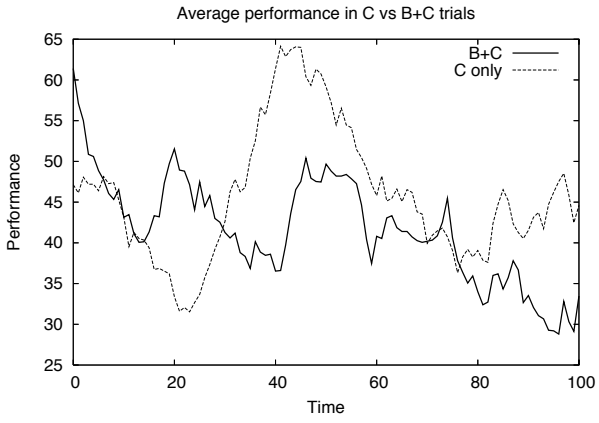

Fig. 5. Graph showing average performance over time in the C versus B+C regimes.

transfer premium is due to Jean using its learned Task B gist to accomplish Task C when the cat is not hiding behind a wall. Jean then only needs to learn to amend this gist to deal with the situation in which the cat is hiding. There is more variability in the **C** condition because random exploration, which is more prevalent in the **C** condition, may cause the cat to run far away.

Finally, in the **B+A** condition, we found an example of negative transfer (graph not shown). Recall that transfer is not always beneficial when applied blindly. Here, Jean first learns task **B**. It learns to sneak up to, and then pounce on, the cat. When Jean directly applies this gist to task A, it results in a sub-optimal behavior, since one does not need to sneak up to a ball. Depending on its exploration behavior, Jean might take a long time to realize that its behavior is suboptimal. However, we can ameliorate this negative transfer by modifying the knowledge being transferred. Instead of transferring a gist that includes all states, actions, and transition probabilities, we can transfer only the state descriptions, leaving out the transition probabilities that were observed when learning the old task. This removes most of the negative transfer effect. Any remaining decrease in performance simply results from the fact that the agent has a slightly larger state space to explore. This transfer mechanism, where transition probabilities are dropped but state description are retained, is used in the **A+B** and **B+C** conditions as well.

7 Future Work

We are currently extending this work in a variety of ways. We have implemented a different learning domain in a real-time 3-D strategy game, complete with terrain, natural obstacles such as trees and water, and enemy units. We have observed some interesting transfer between the simple robot-and-cat domain and this military domain. Sneaking

up to the cat is analogous to sneaking up on the enemy, for example. Future experiments will explore learning in this new domain, and cross-domain transfer.

Another line of development for Jean is suggested by the word "experimental" in Experimental State Splitting, and by the boxes labeled *causal hypotheses* and *experiment planner* in Figure 2. State splitting finds factors that reduce the entropy of state transitions, or conversely, increase the predictability of these transitions. Not all predictive relations are causal. The *counterfactual theory of causality* says X causes Y iff X precedes Y, and X and Y covary, and X is necessary to affect Y. The necessity condition is framed as a counterfactual: $\neg X \rightarrow \neg Y$. The problem with this theory is that it does not distinguish true causes from mere conditions; for instance, a wire is necessary for electricity to travel from a light switch to a light bulb, but we would not call a wire the cause when we turn on the light. A heuristic to get around this is to assign counterfactually necessary and proximal actions to X's in causal models. Thus flipping a light switch, being the most proximal action to illumination, and counterfactually necessary, is a candidate cause. It is easy to find actions that are proximal to effects, and to formulate counterfactuals relating these actions to effects. These counterfactuals serve as causal hypotheses for Jean to try to refute. Jean will develop causal models of its action schemas, and will learn not only what works, but why it works.

References

1. Piaget, J.: The Construction of Reality in the Child. New York: Basic (1954)
2. Lakoff, G.: Women, Fire and Dangerous Things. University of Chicago Press, Chicago, IL (1987)
3. Johnson, M.: The Body in the Mind: The Bodily Basis of Meaning, Imagination, and Reason. University of Chicago Press, Chicago, IL (1987)
4. Gibbs, R.W., Colston, H.L.: The cognitive psychological reality of image schemas and their transformations. Cognitive Linguistics **6**(4) (1995) 347–378
5. Mandler, J.: The Foundations of Mind: Origins of Conceptual Thought. Oxford University Press (2004)
6. Oakley, T.: Image schema. In Geeraerts, D., Cuyckens, H., eds.: Handbook of Cognitive Linguistics. Osford University Press (2006)
7. Thelen, E., Smith, L.: A Dynamic Systems Approach to the Development of Cognition and Action. The MIT Press, Cambridge, MA (1994)
8. Regier, T.: The Human Semantic Potential: Spatial Language and Constrained Connectionism. The MIT Press (1996)
9. Cohen, P.R.: Maps for verbs. In: Proceedings of the Information and Technology Systems Conference, Fifteenth IFIP World Computer Conference. (1998)
10. Talmy, L.: Toward a Cognitive Semantics. Volume 1: Conceptual Structuring Systems (Language, Speech and Communication). The MIT Press, Cambridge, MA (2003)
11. St. Amant, R., Morrison, C.T., Chang, Y., Cohen, P.R., Beal, C.: An image schema language. To appear in the Proceedings of The 7th International Conference on Cognitive Modelling (ICCM 2006) (2006)
12. Fillmore, C.: The case for case. In Bach, E., Harms, R.T., eds.: Universals in Linguistic Theory. Holt, Rinehart & Winston, London (1968)
13. Piater, J.H., Cohen, P.R., Zhang, X., Atighetchi, M.: A randomized anova proceedure for comparing performance curves. In: Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998). (1998) 430–438