Advances in Cognitive Systems (2017) Submitted 3/2017; published 5/2017

Improving Sequential Decision Making with Cognitive Priming

Mark Roberts[1] MARK.ROBERTS@NRL.NAVY.MIL
Laura M. Hiatt[2] LAURA.HIATT@NRL.NAVY.MIL
[1]Naval Research Laboratory (Code 5514); Washington, DC
[2]Naval Research Laboratory (Code 5515); Washington, DC

Abstract

Cognitive priming occurs when items or objects that are the focus of attention prime associated items in memory; these associations are learned over time between related items as an agent interacts with an environment. This mechanism guides attention toward associations relevant to the current situation and facilitates learning by retrieving those associated concepts from long-term memory. We apply priming to two sequential decision problems for the game of Minecraft: selecting the best tool to mine blocks and choosing the next subgoal in a maze. For learning tool selection, we leverage prior work that learned subgoal dependencies (e.g., wheat requires seeds) from the Minecraft community wiki. We apply these dependencies to train an associative memory that primes the selection of a tool for breaking a block and show that priming reduces the regret of choosing the incorrect tool. For subgoal selection, our prior work showed that decision trees can be learned to select the next subgoal for a simple variant of Minecraft. In two studies, we assess how well priming can select subgoals and show that priming performs comparably to decision trees. The first study shows that priming outperforms an incremental decision tree algorithm, while the second study examines the effect of noise in the training and testing data to show that priming performs comparably to a batch decision tree algorithm. Overall, these findings suggest that priming is an effective mechanism for either learning directly or for improving learning.

1. Introduction

Imbuing an agent with the ability to decide what it will do next is a fundamental challenge of intelligent systems design. In some cases, an agent may leverage its experience or imitate an expert to guide its decisions. In others, an agent may apply general knowledge to better inform its exploration of choices. One of the challenges is retrieving relevant experience. Cognitive priming, or associative learning, is a mechanism by which associations of past experience can inform the current context. Prior experience is captured in an associative memory that relates similar things to each other. New context *primes* relevant items in this memory, which can then be examined to determine the closest concepts. For example, an image of a toaster may prime similar kitchen items as well as toasted food items. In this paper, we apply cognitive priming to two sequential decision problems. In an *online* sequential decision problem of choosing a correct tool for resource collection, we show that an associative memory trained from a community wiki can improve the learning rate over a method that lacks priming. In an *offline* decision problem of choosing the next action in maze navigation, we

show that an associative memory trained from expert traces outperforms incremental decision tree induction and performs comparably to batch induction approaches previously used for this task.

Our work is situated in the game of Minecraft, an open-world sandbox game where players choose their own objectives such as building structures, collecting resources, crafting, fighting enemies, or exploring. The world consists of 1 meter voxels (i.e., cube blocks) that can be broken to harvest resources, combined to build more sophisticated tools, or stacked to build structures. In the "vanilla" version of Minecraft, players rarely follow an externally provided story line. Learning to play the game well involves trial-and-error as well as consulting other players or the Minecraft community wiki, which contains a plethora of tips and information on how to play successfully. Two central skills are navigating the world and using various tools to "break" blocks in the world in order to gather resources. Both of these skills require knowledge. Navigation requires knowledge about the immediate state around a player and what not to do (to avoid falling, for example). Breaking blocks efficiently requires knowledge about which tools are best for which blocks.

The open-ended nature of this world means that such details are learned over long periods of time, have a sparse reward signal, require learning by example, or require consulting other players or a wiki. While people excel at learning in these types of situations, these features can pose a significant challenge for off-the-shelf planning and learning algorithms. In this work, we examine how cognitive priming can improve existing learning algorithms even in this challenging situation by borrowing principles from human cognition. Cognitive priming is one of the driving forces behind human cognition, where associations are learned between related items in memory, such that items which are frequently thought about together are retrieved together. Then, items that are currently the focus of attention *prime* the items that are associated with them, making highly-related items more available and active in memory (Hiatt & Trafton, 2016). Priming enables flexible, rapid learning of degrees of correspondences between these items.

We investigate the benefits of priming in two sequential decision problems using datasets from the game of Minecraft and two existing studies. The contributions of this paper include (1) demonstrating priming for tool selection using data from a study by Branavan et al. (2012a) and showing that priming can learn associations from a wiki that improve learning, and (2) demonstrating priming for subgoal selection as previously studied by Roberts et al. (2016) and showing that it performs comparably to decision tree induction. We describe background on priming before detailing our two studies and discussing related work. We close the paper with some thoughts on future directions.

2. Cognitive Priming

Our priming mechanism is based on our theory of priming in human cognition (Harrison & Trafton, 2010; Hiatt & Trafton, 2016; Thomson, Pyke, Trafton, & Hiatt, 2015). Cognitive priming, also called associative learning, is one of the driving forces behind human cognition. In theories of associative learning, associations tie together related items in memory, allowing items that are currently the focus of attention to *prime* these related items, or make them more available and active in memory. Cognitive priming is traditionally studied for its principal role in memory-based functions such as list learning, memory recall, and classical conditioning (Rescorla & Wagner, 1972; Klein, Addis, & Kahana, 2005).

Our theory of priming is situated in an integrated theory of human cognition that is implemented in a computational framework (Trafton et al., 2013; Anderson, 2007). The overarching framework models, in part, human working memory, which includes what a person (or a computational model) is thinking of, looking at, or has as their goal at any give time. As part of our theory on priming, associations are formed between items that are thought about in working memory at the same time (Thomson, Harrison, Trafton, & Hiatt, 2017); then, the more often items are thought about together, the stronger their associations become. Directional associations can also be formed between an item in working memory and its representational features (detailed when we later discuss Figure 3). The associative strength, intuitively, reflects how strongly an item, when currently in working memory, predicts that the item(s) it primes is also relevant to the current situation.

Priming is then a short-term activation value that stems, again, from working memory: at any given time, items are primed according to the strength of their associations with the current contents of working memory. Importantly, using working memory as the basis for creating and strengthening associations, as well as for spreading priming, creates an integrated view of priming that is based on the model's entire state and all of its modalities (vision, aural, its goals and reasoning, etc.), and can incorporate both semantic and statistical correlation information. Recently, we have also shown cognitive priming to be a fundamental component in higher-level cognitive processes, including similarity (Hiatt & Trafton, 2016), feature inference (Hiatt, 2017), and performance on routine procedural tasks (Hiatt & Trafton, 2015).

The exact association strengths are calculated in a Bayesian-like way. The strength S_{ji} from item j to item i is:

$$S_{ji} = mas \cdot e^{\left(\frac{-1}{al \cdot R_{ji}}\right)} \tag{1}$$

$$R_{ji} = \frac{f(N_i C_j)}{f(\hat{N}_i C_j) + 1} \tag{2}$$

where mas is the maximum associative strength, al is the associative learning rate and $f(N_i C_j)$ tallies the number of times that memory j has been in working memory, either independently or as a feature of another item in working memory, while memory i was directly in working memory. The function $f(\hat{N}_i C_j)$ tallies how memory j is in working memory, again independently or as a feature of an item in working memory, while memory i was *not* in working memory. Here, the term R_{ji} is a modification of Bayes' theorem, and the exponential term S_{ji} modifies the conditional probability to fit human behavior. Further details are discussed by Hiatt & Trafton (2016).

To summarize, priming can learn a rich network of associations that can capture correspondences between features/goals that are frequently relevant at roughly the same time. We rely on this aspect of priming as we next discuss the two sequential decision-making problems in which we investigate cognitive priming: using priming to assist with subgoal selection, and using priming to select the appropriate tools to mine different resources.

3. Priming for Tool Selection

We begin with our study of using priming to bias selection of the best tool to break a block. In Minecraft, selecting the incorrect tool for a block (e.g., using a sword on cobblestone) slows the

breaking speed and damages the tool more, decreasing its durability. A player learns to select the correct tool through a combination of repeated interaction with the game, reading the wiki, or watching other players. Over time, the player learns to choose the best tool to minimize tool wear and speed up resource collection. We simulate this iterative process by formulating the tool choice as a multi-armed bandit problem: for each block to break (i.e., the reward), our simulated agent decides which tool (i.e., which arm) to try. We train an associative memory with data from a previous study to examine whether using these primed values to bias action selection (i.e., as a kind of *Bayesian prior*) improves the learning rate of standard algorithms for the multi-armed bandit problem. Our research hypothesis is that learning and priming, together, will reduce regret over learning alone.

3.1 Minecraft Data

Branavan et al. (2012a) learned to plan subgoal sequences in the game of Minecraft by parsing the community wiki. Although they did not specifically examine the tool selection problem, we describe their study so it is clear how we apply their results. Consider a player that wants to produce wheat and must use a hoe to till the soil before planting seeds. To do so, the player should formulate subgoals to craft a hoe, till some dirt, collect seeds, plant them, and harvest the wheat once the wheat matures. Actions have preconditions and produce effects that determine the order in which an agent must perform actions. So each subgoal may depend on previous subgoals (e.g., craft(hoe)→have(hoe)→till(dirt) or collect(seeds)→have(seeds)→plant(seeds)). The authors showed that these dependencies could be learned by parsing wiki pages. From the full game, they considered 74 Minecraft "concepts" (e.g., seeds, hoe, dirt) and actions that acted on these concepts (e.g., craft(hoe), plant(seeds), till(dirt)).

The authors created a baseline, hand-coded model by creating an action model in a metric variant of PDDL (Fox & Long, 2003). This PDDL model served as both a baseline as well as a validator. A learner could test a hypothesis of two subgoals being dependent by "executing" a sequence of subgoals with an automated planner. A valid plan returned by the planner validated the learner's hypothesis as correct.

To learn the pair-wise subgoal dependencies from the wiki, the authors identified 242 wiki sentences related to the 74 concepts, resulting in a corpus with 979 word types. These sentences were parsed using the Stanford parser (de Marneffe, MacCartney, & Manning, 2006) into 694 potential dependencies; full details for how the sentences were parsed are found in (Branavan, Silver, & Barzilay, 2012b). The authors then used reinforcement learning to learn the subgoal dependencies. They showed that the learner using the parsed subgoal dependencies was more effective than the hand-coded model and only slightly less effective than a perfect model.

3.2 The Priming Model

We use these parsed dependencies to build an associative memory of the objects that can be mined with tools. While leveraging the data from Branavan et al. (2012a) to train a priming model has the simplicity of building on prior work in the area, it presents two limitations. First, the aims of the study were slightly different and it is not clear that constructing a model out of dependencies used for a different purpose will be useful. In our study, we stem plural words and remove unique

identifiers for the final dependencies. Second, Branavan et al. (2012a) focused on a relatively small set of 74 concepts but excluded many common game concepts, leaving gaps in the dependencies with respect to tool usage. For example, they excluded the concepts of dirt or gravel, which are both abundant in the game and require a shovel to break. Consider the following sample sentences from Branavan et al. (2012a) with some of the pair-wise dependencies listed below each sentence.

> A chest is crafted from eight wooden plank, as shown below.
> (chest, plank) (plank, chest)
>
> Axe are tools used to ease the process of collecting wood, plank, chest and bookcases, but are not required to gather them.
> (axe, wood) (axe, plank) (axe, chest) (wood, axe) ... (chest, wood) (chest, plank)
>
> Sand can be mined easily by hand, although using a shovel is faster, and gives resources regardless of the tool used.
> (sand, shovel) (shovel, sand)
>
> Shovels are auxiliary tools used to ease the process of collecting dirt, sand, gravel, clay and snow.
> (shovel, sand) (shovel, clay) (sand, shovel) (sand, clay) (clay, shovel) (clay, sand)
>
> Shovels are not effective at destroying soul sand and mycelium.
> (shovel, sand) (sand, shovel)

Dependencies are the cross-product of the mentioned concepts without regard to a direction. Some items in sentences (e.g., dirt, gravel, bookcases) lack dependencies even though the sentences mention them because these concepts were not considered in the original study. Also, the last sentence produced an incorrect dependency for sand when the correct block was actually "soul sand".

When presented to the cognitive priming model, the first item in the pair was added to working memory, and the second item was added as a feature of the first item in working memory. Dependency pairs are also always present in both directions; this results in tools and materials symmetrically priming each other. Overall, from these dependencies, the model learned stronger associations between items that are currently mentioned together in the wiki and weaker associations between items that are not mentioned together as often.

To extract the values to seed the table, we prime the model with the block to be broken (e.g., sand) and collect all the tool associations for that block, which produces a weight with the range of $(0, 1)$ for each associated item. Table 1 (top) shows the associations produced from the model for each tool when primed with the block in the left column. Other items not in this table may be primed but are not considered for this study. For example, the original concepts from Branavan et al. (2012a) did not contain dirt or gravel (i.e., the bottom rows of each table), so the bias value is uniformly zero.

To determine the best tool, we use the Minecraft community wiki, which contains a page[1] with details on the block breaking mechanics. We parsed a table from this page to determine the best tools for the blocks we study. The best tool does not yet account for tool speed or material durability. We exclude blocks from Branavan et al. (2012a) that can be mined with any tool. For 15 blocks,

1. `http://minecraft.gamepedia.com/Tools`

Values Provided by Priming

	hoe		axe			shears			pickaxe			shovel		
bars	0.36	0	0.40		0	0.00		0	**0.58**	*	0	0.36		0
brick	0.14	0	0.17		0	0.00		0	0.25	*	0	**0.46**		0
chest	0.50	0	0.51	*	0	0.05		0	**0.53**		0	0.47		0
clay	0.15	0	0.17		0	0.00		0	0.33		0	**0.65**	*	0
coal	0.00	0	0.14		0	0.00		0	**0.29**	*	0	0.00		0
cobblestone	0.28	0	0.35		0	0.00		0	**0.42**	*	0	0.41		0
door	0.50	0	0.57	*	0	0.06		0	**0.60**		0	0.52		0
fence	0.48	0	**0.54**	*	0	0.05		0	0.51		0	0.49		0
furnace	0.19	0	0.29		0	0.00		0	**0.32**	*	0	0.25		0
iron	0.19	0	0.24		0	0.10		0	**0.54**	*	0	0.31		0
sand	0.25	0	0.20		0	0.08		0	0.26		0	**0.40**	*	0
sandstone	0.00	0	0.00		0	0.00		0	0.00	*	0	**0.50**		0
stair	0.57	0	**0.64**		0	0.00		0	0.61	*	0	0.59		0
wood	0.40	0	**0.58**	*	0	0.00		0	0.41		0	0.41		0
wool	0.15	0	0.20		0	**0.32**	*	0	0.15		0	0.24		0
dirt	0.00	0	0.00		0	0.00		0	0.00		0	0.00	*	0
gravel	0.00	0	0.00		0	0.00		0	0.00		0	0.00	*	0

Learned Action Model After Learning for SM+Priming

bars	0.36	0	0.40		0	0.00		0	**1.00**	*	50	0.36		0
brick	0.14	0	0.17		0	0.00		0	**1.00**	*	49	0.00		1
chest	0.50	0	**1.00**	*	49	0.05		0	0.53		0	0.00		1
clay	0.15	0	0.17		0	0.00		0	0.33		0	**1.00**	*	50
coal	0.00	0	0.14		0	0.00		0	**1.00**	*	50	0.00		0
cobblestone	0.28	0	0.00		1	0.00		0	**1.00**	*	48	0.00		1
door	0.50	0	**1.00**	*	49	0.06		0	0.00		1	0.52		0
fence	0.48	0	**1.00**	*	49	0.05		0	0.00		1	0.49		0
furnace	0.19	0	0.29		0	0.00		0	**1.00**	*	50	0.25		0
iron	0.19	0	0.24		0	0.10		0	**1.00**	*	50	0.31		0
sand	0.00	1	0.20		0	0.08		0	0.26		0	**1.00**	*	49
sandstone	0.00	0	0.00		0	0.00		0	**1.00**	*	49	0.00		1
stair	0.57	0	0.00		1	0.00		0	**1.00**	*	49	0.59		0
wood	0.40	0	**1.00**	*	50	0.00		0	0.41		0	0.41		0
wool	0.15	0	0.20		0	**1.00**	*	50	0.15		0	0.24		0
dirt	0.00	0	0.00		0	0.00		0	0.00		1	**1.00**	*	49
gravel	0.00	4	0.00		3	0.00		3	0.00		2	**1.00**	*	38

Table 1. Priming values used to bias tool selection (top) and final reward values for SM+Priming after learning (bottom). Asterisks indicate the best tool for breaking a block, while bold indicates the highest weight for a block. The count in the far right of each column indicates how many times that tool was tried out of 50 attempts.

excluding dirt and gravel, priming correctly associates 10 tools and incorrectly associates the best tools for brick, chest, door, sandstone, and stair.

3.3 Improving Tool Selection

Tool selection can be formulated as an iterated multi-armed bandit problem. For the ith block, the tools represent a choice from K probability distributions $(D_{i,1}, \ldots, D_{i,K})$ with expected means $(\mu_{i,1}, \ldots, \mu_{i,K})$ and variances $(\sigma^2_{i,1}, \ldots, \sigma^2_{i,K})$. The correct tool is exactly known, so the mean of that correct tool is 1 while the mean of the incorrect tools is zero; all the variances are also zero. At each

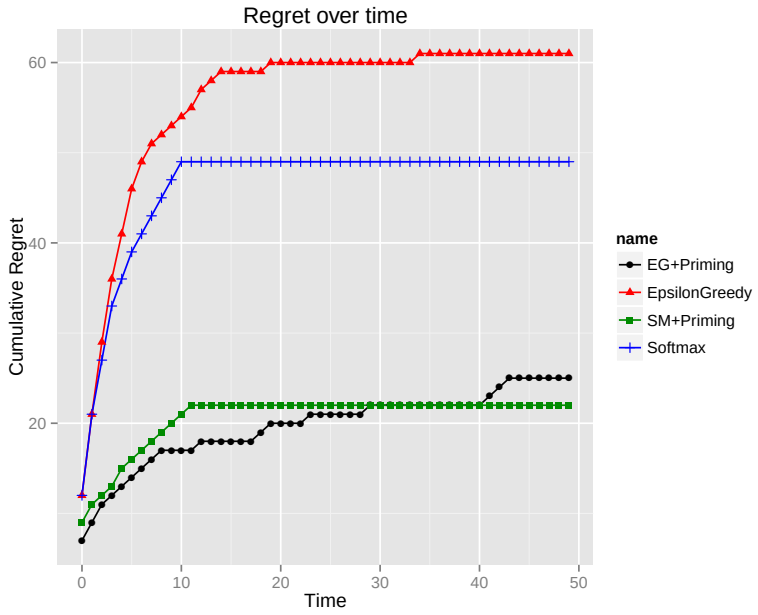

Figure 1. Cumulative reward (left) and cumulative regret for EpsilonGreedy and Softmax without priming and with priming.

time t and for the ith block, an algorithm selects a tool with index $i, j(t)$ and reward $(r_i(t) = D_{i,j(t)})$, which is 1 for choosing correctly and 0 otherwise. A common way to evaluate an algorithm is regret. For I blocks the total regret of an algorithm is calculated: $R_T = \Sigma_{i=1}^{I}(T\mu_i^* - \Sigma_{t=1}^{T}\mu_{i,j(t)})$, where $\mu_i^* = max_{j=1,...,k}\mu_{i,j}$.

We compare a baseline approach with and without priming used to bias selection. Our baseline approaches are EpsilonGreedy with an $\epsilon = 0.10$ and Softmax with a temperature of 0.10; these are standard approaches. When more than one tool equals the maximum value, as can happen in the case of the priming values, EpsilonGreedy selects a random choice among the possible maximums while Softmax uses a weighted roulette wheel across the ranked choices. We run these algorithms with and without priming with a learning rates of 0.30. We tried variations of the parameters for priming, Softmax, and EpsilonGreedy with similar results.

The evidence suggests that seeding the algorithm priming values reduces regret not only before learning begins but also over time. Figure 1 shows *cumulative* regret over time for learning and learning+priming. It is clear that priming improves the baseline approaches. Both learning algorithms, when using priming, start with lower regret, and regret remains lower than their respective versions without priming. The Softmax+priming (SM+Priming) explores more at the start and shows higher

regret at the start but seems to quickly learn a good policy compared to EpsilonGreedy+priming (EG+Priming), which explores less at the beginning but continues to accumulate regret. Figure 1 (bottom) shows the final final expected reward values for SM+Priming. When compared with the top table, it is clear that relatively little exploration resulted in a good policy (note the number of times the incorrect tool was chosen). Even for blocks for which priming had an incorrect association (e.g., chest or door), priming eliminates several poor choices (e.g., shears) and still results in lower regret for those items than a uniform selection.

4. Priming For Subgoal Selection

We now examine the use of priming to choose the best subgoal in the maze problem. Our prior study showed that this problem could be learned from expert traces using decision tree induction (Roberts, Shivashankar, Alford, Leece, Gupta, & Aha, 2016b) or deep learning (Bonanno, Roberts, Smith, & Aha, 2016). Thus, it provides a suitable baseline against which to assess priming. We begin by comparing incremental decision tree induction with Hoeffding trees against priming to answer the question: how well does priming compare to incremental decision tree induction? Building on prior results for decision tree induction with J48, we then construct "noisy" data and examine two questions which assess how well a trained learner generalizes to unseen examples or to "sensor" error: (1) When both a decision tree learner and priming are trained with non-mutated training data but tested with increasingly mutated testing data, how does the performance of the two approaches differ? (2) When trained with increasingly mutated training data and tested with non-mutated data, how do they differ? We first describe the data we used.

4.1 Minecraft Data

Our work replicates and extends a prior study. To this end, we regenerated portions of the data from Roberts et al. (2016) using the ACTORSIM-Minecraft connector, which provides abstract methods for axis-aligned control for looking at, moving, jumping, and destroying blocks. The connector can observe blocks directly around Alex and produces an alpha-numeric code to indicate the local position relative to Alex's feet, "[lN | rN][fN | bN][uN | dN]", where N is a non-negative integer and each letter designates **l**eft,**r**ight, **f**ront, **b**ack, **u**p, and **d**own. Thus, "f1d1" indicates the block directly in front and down one (i.e., the block that Alex would stand on after stepping forward one block), while "l1" indicates the block directly to the left at the same level as Alex's feet. Each relative position is denoted with a single letter: air (A), safe or solid (S), water (W), or lava (L). Alex moves using one of five subgoals: stepTo, stepAround, build a bridge, build a stair one block high, and mine.

Figure 2 (left) shows four of the ten section types we use (arch, comb, pillar, and hill), while (right) shows a portion of a course with upcoming sections of a comb, swamp, lava and short wall. Not shown are tall walls (3 blocks high), deep ponds (water 3 deep), and ponds (water 2 deep). Each obstacle has an appropriate subgoal choice. For lava or ponds, the best choice is to create a bridge if approaching the center or to go around if approaching the edge. For the short walls, the best subgoal is to create a single stair and step up. For the tall walls, combs, and pillars, the best subgoal is to mine through if approaching the center or go around if approaching the edge.

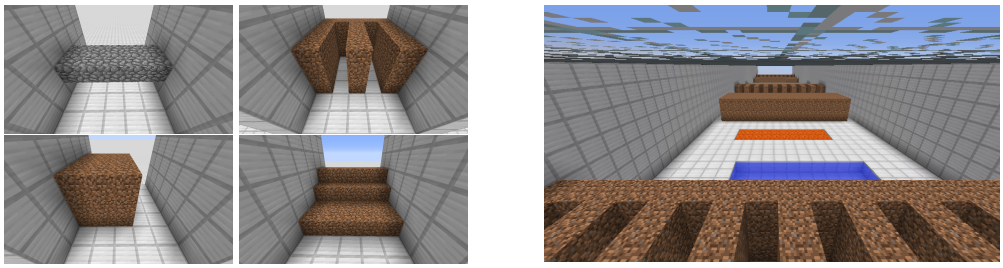

Figure 2. Left: four example section types (top left to bottom right) include arch, comb, pillar, and steps. Right: portion of a course where the GRPROCESS must traverse from the emerald block behind it (not shown) to a gold block in front of it (not shown). Glass blocks on the top prevent Alex from walking along the wall.

We replicated the prior study by using the same 30 courses with 20 sections each to generate a set of Expert traces. In this case, the "Expert" is a hand-coded selection policy. We also improved the Expert decision maker so that it makes errors even less frequently than before; the prior study had some corner cases where the Expert would get stuck and the revised Expert procedure never fails to complete a course. A run terminates when Alex reaches the goal. For each decision, we capture the state, distance to the goal, subgoal chosen, whether the chosen subgoal succeeded, and how many blocks were placed or mined. Together, these runs produced 5931 decisions.

4.2 The Priming Model

The priming model learns about what subgoals correspond to which features via an iterative learning process. During training, the model considers each decision. It adds the selected subgoal to working memory and uses the state (i.e., the perceived blocks around Alex) as the subgoal's features. Figure 3 (left) illustrates the model's working memory during training.

As more decisions are considered, the model learns differing strengths for the blocks around Alex and the selected subgoal (e.g., Figure 3, right). Three important patterns emerge from the associative strengths between the coded features j and subgoals i (cf. Equations 1-2). First, coded features that were strongly indicative of certain subgoals were typically strongly associated with them, because they occurred frequently with that subgoal but occurred infrequently with other subgoals, leading to a low $f(\hat{N}_i C_j)$ and thus a stronger associative weight. Second, features that were weakly indicative of certain subgoals were typically weakly associated with them, because they had a lower $f(\hat{N}_i C_j)$ value and thus a lower associative weight. Finally, features that were associated with many different subgoals spread a low association to all of them, because they would have a high $f(\hat{N}_i C_j)$ for each of their associations.

For the priming model, we prime the memory with the current state – the (location, block) values around Alex. These features then spread activation, according to their associative weights, to all connected subgoals. Priming then selects the subgoal with the most highest association.

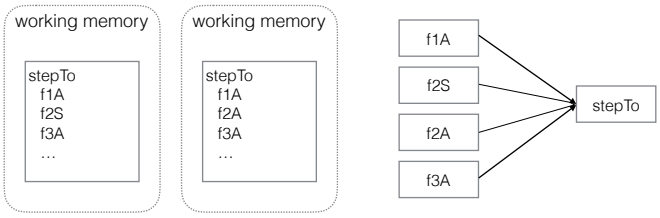

Figure 3. Associations after two snapshots of items in working memory. The association between the environment's features (f1A, etc.) and the items in working memory are directional. f1A and f3A more strongly prime stepTo (thus a darker line) because they have occurred together more often than f2S, f2A and stepTo.

4.3 Comparing the Performance of Priming and Decision Tree Induction

The 5931 decisions from the Expert procedure are heavily biased because stepping forward (StepTo) is the most common action. Therefore, we balance the data down to 1523 decisions which include 437 StepTo, 359 StepAround, 347 Stairs, 204 Bridge, and 175 Mine subgoals. We examine two kinds of decision tree induction: incremental and batch.

To approximate incremental learning, we create 10-folds to perform leave-one-out cross-validation. For each fold, we train each algorithm on increments of 10%, 20%, ..., 100% of the data. At each increment, we observe the accuracy against the held-out test data. We compare priming to the Hoeffding tree as implemented in WEKA[2]. Figure 4 shows the average accuracy of decision trees learned by Hoeffding induction versus priming trained with the same data. While priming gracefully improves with more training data, it is clear that Hoeffding trees are more susceptible to fewer training samples.

In our second study, we examine how well priming compares to prior results using decision tree classifiers learned by the J48 algorithm implemented in WEKA. Further, we wanted to understand how both priming and decision tree induction was sensitive to errors in the training or testing. Based on the incremental results, we use smaller sample sizes of 20 folds with 5% each, and we also mutate the original data by randomly changing the values in the state. Figure 5 shows a performance comparison of the J48 decision trees and priming as the mutation rate increases. The priming examples use the best of 5 runs varying the learning weight parameter and we use J48's default parameter settings as in the previous study. It is clear that priming and decision trees are functionally equivalent because there is little difference between the plots. In fact, the right plot shows nearly identical results. Still, this result is admittedly not as strong as we expected – we expected priming would be significantly more robust than decision tree induction because trained decision trees can be brittle to unseen examples that differ from the original problem distribution due to mutation, while priming has a more probabilistic interpretation of feature-class similarity.

Priming dominates Hoeffding tree induction during incremental learning and performs less well against the batch learning of J48 tree induction under noisy observations. We offer some possible explanations that we will examine in future work. First, this particular problem is amenable to batch tree induction because it has a high feature-to-class ratio. For example, there are 24 features (i.e.,

2. `http://www.cs.waikato.ac.nz/~ml/weka/`

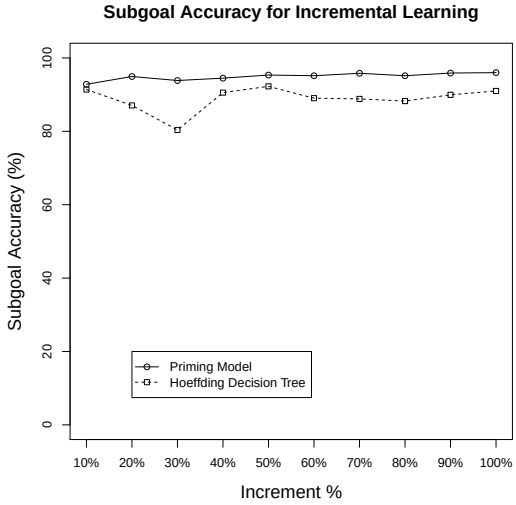

Figure 4. Subgoal accuracy for 10-fold cross validation training for 10%, 20%, ..., 100% of the data to simulate incremental learning.

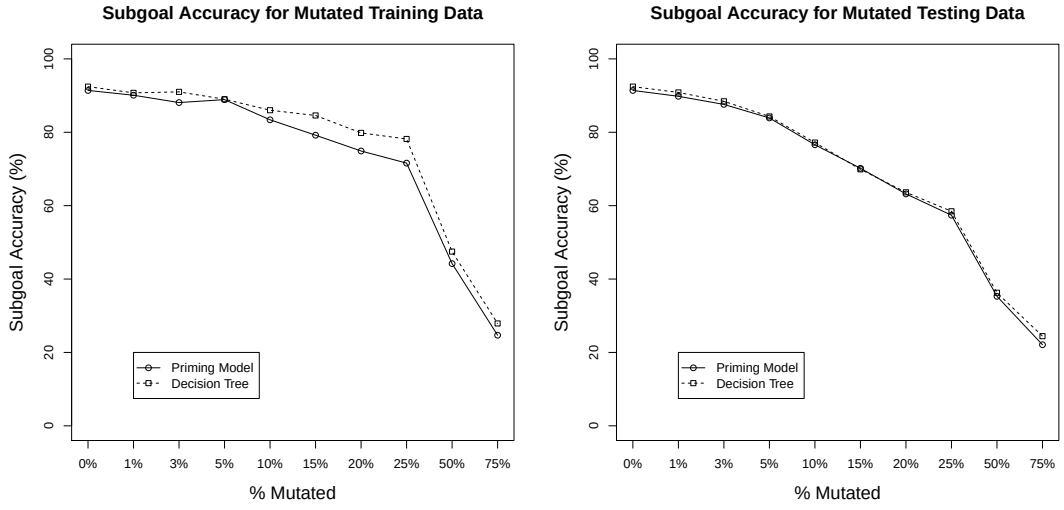

Figure 5. Subgoal accuracy for 20-fold cross validation training on a single fold (5% of the data) and testing on the other 19 folds (95%). The left plot shows mutated training data and the right shows mutated testing data.

(location,block) values) that take on 4 possible values and only 5 classes. From that standpoint, it is impressive that priming was able to compete so strongly with J48. Second, the noise we introduce may not mutate critical features. For example, building a bridge is relatively rare and only relevant

when water and lava are present directly in front of Alex. By randomly mutating across all 24 features instead of focusing the mutations on the handful of features used by the decision tree, we may have failed to change fundamentally the underlying structure of the learning problem. Third, there is latent information in the kinds of errors each algorithm makes and what this means for improving them in an *online* setting – some homologous subgoals have negligible impact if chosen instead of the expert choice while other subgoals would result in great harm to Alex if executed. For example, in most situations stepTo and stepAroundTo are applicable and have a nearly identical effect of moving the player closer to the target, but buildBridgeTo when lava is in front of Alex is really the only appropriate decision. A quick glance at the confusion matrices as mutation increases reveals several trends: (1) both algorithms identify the stepTo/stepAroundTo similarity, (2) priming maintains a clear identification of buildStairsTo, and (3) J48 maintains clear classification of buildBridgeTo. Together, these results warrant continuing this line of work while moving toward more challenging *online* scenarios, better characterizing where each approach excels, and including a negative reward signal for damaging choices.

5. Related Work

Recently, AI research in game-playing concentrated on exploiting Deep Learning techniques, particularly deep Q-networks, which outperform expert human players on a range of ATARI games (Mnih et al., 2015). ATARI games have also been addressed using classical planning (Lipovetzky, Ramirez, & Geffner, 2015). Games like Minecraft require a non-trivial transition to a much more complex environment with sparse (or intrinsic) rewards. Recent efforts at this game show promise at solving simplified tasks from Minecraft (e.g., Bonanno et al., 2016; Tessler, Givony, Zahavy, Mankowitz, & Mannor, 2016; Abel, Hershkowitz, Barth-Maron, Brawner, O'Farrell, MacGlashan, & Tellex, 2015; Usunier, Synnaeve, Lin, & Chintala, 2016; Vinyals & Povey, 2012; Aluru, Tellex, Oberlin, & Macglashan, 2015).

Cognitive priming is traditionally studied for its principal role in memory-based functions such as list learning, memory recall, and classical conditioning (Rescorla and Wagner 1972; Klein, Addis, and Kahana 2005). Priming can also be interpreted as a type of case-based reasoning in that it reasons about previous cases that are relevant to the current situation. Many approaches, additionally, rely on representations that convey explicit, symbolic feature correspondences (Kolodner, 1992), and so fail to capture implicit relationships between concepts. Lenz & Burkhard (1996) describe, for example, an approach to case-based reasoning that utilizes spreading activation. However, it spreads activation between only explicit associations among features and from features to cases, without learning implicit relationships; and, perhaps most critically, it explicitly sets and tunes similarities instead of learning their importance automatically over time, as ours does. It also performs batch learning, which makes it unsuitable for the online, dynamic situations we consider here.

The tool selection work is similar to Goal-Based Action Priors by Abel et al. (2015), where the authors showed that using prior knowledge to bias action selection can significantly improve the learning rate for a reinforcement learning algorithm. Similar to our work, the bias could be learned from either previous experience or an expert "teacher". The tool selection problem relates to simple variants of Reinforcement Learning (Sutton & Barto, 1998; Kaebling, Littman, & Moore, 1996),

which faces a similar tradeoff of balancing exploration to learn new knowledge versus exploitation of knowledge already acquired. Details of this relationship are covered by Sutton & Barto (1998).

A variety of research systems have been developed to study Minecraft. The first, called BurlapCraft, by Abel et al. (2015) integrated the BURLAP machine learning platform[3] and examined how to use knowledge to select actions. More recently, Microsoft released an open-source platform called Malmo[4] that provides extensive support for multiple programming languages, the ability to set up experiments, as well as support for reinforcement learning. Our system, called ActorSim (Roberts et al., 2016), is primarily distinguished from these other systems in its implementation of goal reasoning, existing experiments using deep learning (Bonanno et al., 2016), and integration with additional simulators and robotics platforms.

Goal reasoning is a multi-disciplinary topic that has origins in early AI systems (Vattam et al., 2013) . One early variant of goal reasoning systems included Goal-Driven Autonomy (Molineaux & Aha, 2014), where an agent reasoned about what goals to achieve based on the changing environment. A complementary approach combines meta-cognition and goal reasoning (Cox et al., 2016) , where an agent reasons about its planning model to correct past failures. Goal reasoning has been used in other gaming domains such as Battle of Survival, a real-time strategy game (Klenk et al., 2013).

6. Summary and Future Work

We demonstrated the use of cognitive priming on two sequential decision problems. In a subgoal selection problem, we showed that priming degrades similarly to J48 when the training or testing data is increasingly mutated. Priming is comparable – but not strictly competitive – with J48 as used in a prior study by Roberts et al. (2016). While this result was less in favor of priming than we expected, it opened avenues for improving our comprehension of why J48 does so well and highlighted at least one case where priming performs particularly well – that of selecting when to build a staircase.

In a tool selection problem, we showed that priming can substantially improve learning over common baselines by using data from a study by Branavan et al. (2012a) that collected conceptual dependencies from the Minecraft community wiki. The study shows a case where priming is particularly useful at leveraging existing knowledge to make more informed choices and reduce regret over time. This study justifies further work in extending beyond concepts of the original study and exploring other situations where priming can improve learning in Minecraft scenarios.

We have demonstrated the merit of applying cognitive priming to improve learning, which lays a foundation for a variety of possible future directions once we have extended these approaches to work within the actual game rather than use data from prior studies. First, we plan to use cognitive priming to retrieve goal strategies, which are the operations of a goal reasoning system discussed by Roberts et al. (2016) and formalized as a goal-task network Alford, Shivashankar, Roberts, Frank, & Aha (to appear). Second, we plan to integrate the primed goal-task networks and goal strategies with deep learning approaches. Finally, we plan to combine cognitive priming with perpetual learning Roberts, Hiatt, Coman, Choi, Johnson, & Aha (2016a) to demonstrate an agent that learns appropriate cues from the environment over extended lifetimes while leveraging its existing knowledge.

3. http://burlap.cs.brown.edu/
4. https://github.com/Microsoft/malmo

Acknowledgements

This research was funded by NRL. We thank the David Aha and the anonymous reviewers for their comments that helped improve this paper.

References

Abel, D., Hershkowitz, D. E., Barth-Maron, G., Brawner, S., O'Farrell, K., MacGlashan, J., & Tellex, S. (2015). Goal-based action priors. *Proc. Int'l Conf. on Automated Planning and Scheduling*.

Alford, R., Shivashankar, V., Roberts, M., Frank, J., & Aha, D. W. (to appear). Hierarchical planning: Relating task and goal decomposition with task sharing. *Proc. of the Int'l Joint Conf. on AI (IJCAI)*. AAAI Press.

Aluru, K., Tellex, S., Oberlin, J., & Macglashan, J. (2015). Minecraft as an experimental world for AI in robotics. *AAAI Fall Symposium*.

Anderson, J. R. (2007). *How Can the Human Mind Occur in the Physical Universe?*. Oxford University Press.

Bonanno, D., Roberts, M., Smith, L., & Aha, D. W. (2016). Selecting Subgoals using Deep Learning in Minecraft: A Preliminary Report. *IJCAI Workshop on Deep Learning for Artificial Intelligence*.

Branavan, S., Kushman, N., Lei, T., & Barzilay, R. (2012a). Learning high-level planning from text. *Proc. of the 50th Annual Meeting of the Association for Computational Linguistics* (pp. 126–135). Jeju, Republic of Korea.

Branavan, S., Silver, D., & Barzilay, R. (2012b). Learning to win by reading manuals in a monte-carlo framework. *J. of Art. Intell. Res.*, *43*, 661–704.

Cox, M. T., Alavi, Z., Dannenhauer, D., Eyorokon, V., Munoz-Avila, H., & Perlis, D. (2016). MIDCA: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy. *Proc. of the 30th AAAI Conf. on AI* (pp. 3712–3718).

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. of Art. Intell. Res.*, *20*, 61–124.

Harrison, A. M., & Trafton, J. G. (2010). Cognition for action: an architectural account for "grounded interaction". *Proceedings of the Annual Meeting of the Cognitive Science Society*.

Hiatt, L. M. (2017). A priming model of category-based feature inference. *Proceedings of the Annual Meeting of the Cognitive Science Society*.

Hiatt, L. M., & Trafton, J. G. (2015). An activation-based model of routine sequence errors. *Proceedings of the International Conference on Cognitive Modeling*.

Hiatt, L. M., & Trafton, J. G. (2016). Familiarity, priming and perception in similarity judgments. *Cognitive Science*. Doi: 10.1111/cogs.12418.

Kaebling, L., Littman, M., & Moore, A. (1996). Reinforcement learning: A survey. *J. of Art. Intell. Res.*, *4*, 237–285.

Klein, K. A., Addis, K. M., & Kahana, M. J. (2005). A comparative analysis of serial and free recall. *Memory & Cognition*, *33*, 833–839.

Klenk, M., Molineaux, M., & Aha, D. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, *29(2)*, 187–206.

Kolodner, J. L. (1992). An introduction to case-based reasoning. *Artificial Intelligence Review*, *6*, 3–34.

Lenz, M., & Burkhard, H.-D. (1996). Case retrieval nets: Basic ideas and extensions. *KI-96: Advances in Artificial Intelligence*.

Lipovetzky, N., Ramirez, M., & Geffner, H. (2015). Classical planning with simulators: results on the atari video games. *International Joint Conference on Artificial Intelligence*.

de Marneffe, M.-C., MacCartney, B., & Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. *Proceedings of the 5th International Conference on Language Resources and Evaluation*.

Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–533.

Molineaux, M., & Aha, D. W. (2014). Learning unknown event models. *Proc. of the 28th AAAI Conf. on AI* (pp. 394–401).

Rescorla, R. A., & Wagner, A. R. (1972). A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Classical Conditioning II: Current Research and Theory*, *2*, 64–99.

Roberts, M., Hiatt, L. M., Coman, A., Choi, D., Johnson, B., & Aha, D. (2016a). Actorsim, a toolkit for studying cross-disciplinary challenges in autonomy. *Working Notes of the Symposium on Cross-Disciplinary Challenges in Autonomy, Fall Symposium Series*. URL `publications/robertsEtAl16.aaaifss.actorsimToolkit.pdf`.

Roberts, M., Shivashankar, V., Alford, R., Leece, M., Gupta, S., & Aha, D. (2016b). Goal reasoning, planning, and acting with actorsim, the actor simulator. *Poster Proceedings of the Fourth Annual Conference on Advances in Cognitive Systems*. Evanston, IL, USA.

Sutton, R., & Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., & Mannor, S. (2016). A Deep Hierarchical Approach to Lifelong Learning in Minecraft. *Proc. of the 31st AAAI Conf. on AI* (pp. 1553–1561).

Thomson, R., Harrison, A. M., Trafton, J. G., & Hiatt, L. M. (2017). An account of interference in associative memory: Learning the fan effect. *Topics in Cognitive Science*. Doi: 10.1111/tops.12244.

Thomson, R., Pyke, A. A., Trafton, J. G., & Hiatt, L. M. (2015). An account of associative learning in memory recall. *Proceedings of the Annual Meeting of the Cognitive Science Society*.

Trafton, J. G., Hiatt, L. M., Harrison, A. M., Tamborello, II, F. P., Khemlani, S. S., & Schultz, A. C. (2013). ACT-R/E: An embodied cognitive architecture for human-robot interaction. *Journal of Human-Robot Interaction*, *2*, 30–55.

Usunier, N., Synnaeve, G., Lin, Z., & Chintala, S. (2016). Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks. *arXiv preprint arXiv:1609.02993*.

Vattam, S., Klenk, M., Molineaux, M., & Aha, D. (2013). Breadth of approaches to goal reasoning: A research survey. *Goal Reasoning: Papers from the ACS Workshop (Technical Report CS-TR-5029). College Park, MD: University of Maryland, Department of Computer Science* (pp. 222–231).

Vinyals, O., & Povey, D. (2012). Krylov Subspace Descent for Deep Learning. *AISTATS* (pp. 1261–1268).