Advances in Cognitive Systems: Papers from the 2011 AAAI Fall Symposium (FS-11-01)

A Case Study in Integrating Probabilistic Decision Making and Learning in a Symbolic Cognitive Architecture: Soar Plays Dice

John E. Laird, Nate Derbinsky, and Miller Tinkerhess

University of Michigan

2260 Hayward St.

Ann Arbor, MI 48109-2121

{laird, nlderbin, mmtinker}@umich.edu

Abstract

One challenge for cognitive architectures is to effectively use different forms of knowledge and learning. We present a case study of Soar agents that play a multiplayer dice game, in which probabilistic reasoning and heuristic symbolic knowledge appear to play a central role. We develop and evaluate a collection of agents that use different combinations of probabilistic decision making, heuristic symbolic reasoning, opponent modeling, and learning. We demonstrate agents that use Soar's rule learning mechanism (chunking) to convert deliberate reasoning with probabilities into implicit reasoning, and then use reinforcement learning to further tune performance.

Introduction

To date, few if any of the applications developed within the Soar cognitive architecture (Laird 2008) have involved explicit reasoning about probabilities. Soar's primary memory systems encode knowledge in symbolic representations. Soar uses non-symbolic processing to bias retrievals from semantic (Derbinsky and Laird 2011; Derbinsky, Laird, and Smith 2010) and episodic memory (Derbinsky and Laird 2009), represent spatial information (Wintermute 2010), and control the selection of operators in Soar. These uses of non-symbolic reasoning are similar to those found in other cognitive architectures such as ACT-R (Anderson et al. 2004) and ICARUS (Langley, Cummings, and Shapiro 2004), where non-symbolic processing supports the reasoning over symbolic structures. In all these systems, knowledge of the task is represented and processed symbolically. Probabilistic knowledge may play a role in controlling reasoning, but it is not the primary representation of task knowledge - instead it plays a supporting role.

To explore the role of probabilistic reasoning in a symbolic cognitive architecture, we developed agents in Soar that play a multiplayer dice game in which the current situation is highly uncertain and probabilities appear to play a central role in decision making. This paper is a preliminary report on these agents. We begin with a description of the game, followed by an analysis in which we make observations about the types of knowledge and mechanisms that might be useful for an agent that plays the game. We describe the structure of our agent, and focus on decision making, where we describe how we developed agents that use the types of knowledge described earlier. We then provide empirical evidence of their usefulness in agents that play the dice game

One claim of this paper is that Soar's decision-making mechanisms (preference-based operator selection and impasse-driven deliberation) provide the necessary architectural support for incorporating symbolic and probabilistic information for effective decision making. A second claim is that Soar's procedural-learning mechanisms (chunking and reinforcement learning) provide the necessary architectural support for compiling and then tuning probabilistic decision making using in-game experience. This unique combination of chunking and reinforcement learning leads to high initial performance that improves with experience, exceeding the performance of our best hand-coded agents.

The Dice Game

Our agents play a dice game that goes by many names, including Perudo, Dudo, and Liar's Dice. The rules of our version are available from the Soar website, as is a playable version of the game, where humans can play against each other. The game begins with the players positioned in a random cyclic ordering (such as sitting around a table), with each player initially having five dice and a cup in which to roll and hide their dice. Play consists of multiple rounds, and at the beginning of each round, all players roll their dice, keeping them hidden under their cup. Players can view their own dice, but not the dice of others. The first player of a round is chosen at random, and following a player's turn, play continues to the next player.

During a player's turn, an action must be taken, with the two most important types of action being bids and challenges. A bid is a claim that there is at least the specified number of dice of a specific face in play, such as six 4's. Following the first bid, a player's bid must increase the previous bid, either by increasing the dice face or by increasing the number of dice. If the dice face does not increase, the number of dice must increase, in which case the dice face can be the same or lower. Thus, legal bids following six 4's include six 5's, six 6's, seven 2's, seven 3's, and so on. Following a bid, it is the next player's turn.

A second type of action a player can take is to challenge the most recent bid. If a player challenges, all dice are revealed, and counted. If there are at least as many dice of the face that was bid, the challenge fails, and the challenger loses a die. Otherwise, the person who made the bid loses a die. A player who loses all dice is out of the game. The last remaining player is the winner.

There are additional rules that enrich the game. A die with a face of 1 is wild, and it contributes to making any bid. Given the special status of 1's, all 1 bids are higher than twice the number of other bids. For example, three 1's is higher than six 6's and the next bid after three 1's is seven 2's. When a player makes a bid, they can "push" out a subset of their dice (usually 1's and those with the same face as the bid), exposing them to all players, and reroll the remaining dice. A push and reroll can be used to increase the likelihood of a bid being successful, and provide additional information to other players that might dissuade them from challenging a bid. In addition, a player can bid "exact" once per game. An exact bid succeeds if the number of dice claimed by the previous bid is accurate, otherwise it fails. If the exact bid succeeds, the player gets back a lost die, otherwise the player loses a die. Finally, a player with more than one die can "pass," which is a claim that all of the player's dice have the same face. A pass can be challenged, as can the bid before a pass. A player can pass only once with a given roll of dice.

Dice Game Analysis

In the dice game, the success of a player's bid or challenge depends not only on the player's dice, but also on the dice unknown to the player. Because of this uncertainty, it would appear that reasoning with probabilities would be useful if not necessary. For example, in making a challenge, it would appear to be useful to know what the probability is that the challenge will succeed. That probability could be compared to probabilities for other actions the agent can take, such as making a specific bid. However, through discussions with human players, we found that people do not compute the actual probabilities, but instead compute the expected number for a specific face. For non-one bids, they first sum the number of known 1 dice and the known dice of the face in question (exposed and under their cup). They then add the number of unknown dice (under other players' cups) divided by 3 (dice that are 1's and dice with the face being bid make up approximately 1/3 of the available dice). If they are considering a bid of 1's, they divide the number of unknown dice by 6. They then use the difference between this expected number of dice and the bid under question as the basis for comparison to other bids. Thus, our first observation is that there are at least two basic ways to evaluate bids, and an interesting question is whether the probability calculation is significantly better than the expected number approach.

A second observation is that when a player makes a bid of a specific face, it is often because the player has a sufficient number of dice of that face to justify making that bid (via whatever reasoning mechanism the player is using). Of course the player could be bluffing, but in the dice game, you don't lose a die if you don't challenge and if no player challenges you. The best result is for another player to challenge a third player. Therefore, experienced players use bluffing sparingly and there is usually a correlation between a player's bid and the dice under their cup, making it useful to analyze the bid of the previous player from their perspective – what dice would they have to have in order to make their bid? We refer to using this type of knowledge as using a *model* of an opponent.

A third observation is that there is additional structure to the game that is not easily captured in the pure probabilities or the expectations. For example, if a player has a valid pass, it is wise to save the pass until the player has no other safe or certain bids. Along similar lines, it is better not to push with a bid if that bid without a push is unlikely to be challenged by the next player. A push reveals information to the other players and decreases the agent's options in the future. These are examples of heuristic knowledge that is easily encoded as symbolic rules that depend on qualitative evaluations of certainty of bids, but do not require reasoning about probabilities.

There is additional knowledge not included in the above discussions. For example, there is a the difference between making a bid of 6's, which forces the next player to increase the count of dice bid, versus a bid of 2's, which does not. There also are most likely regularities in the actions of other players that can be exploited. It is difficult to conceive of how these can be pre-encoded in an agent, and one alternative is that they must be learned.

The result of this analysis is that there are four classes of knowledge that could prove useful to an agent:

1. Probability or expected-number calculations
2. Opponent models
3. Expert heuristics
4. Knowledge learned by experience

Our challenge is to design agents that incorporate these forms of knowledge.

Dice-Game Agents: Overall Structure

Figure 1 shows our dice game system. There is a game server that enforces the rules of the game, advances play to the next player, and provides information on the current state of the game. There are other players, human or Soar

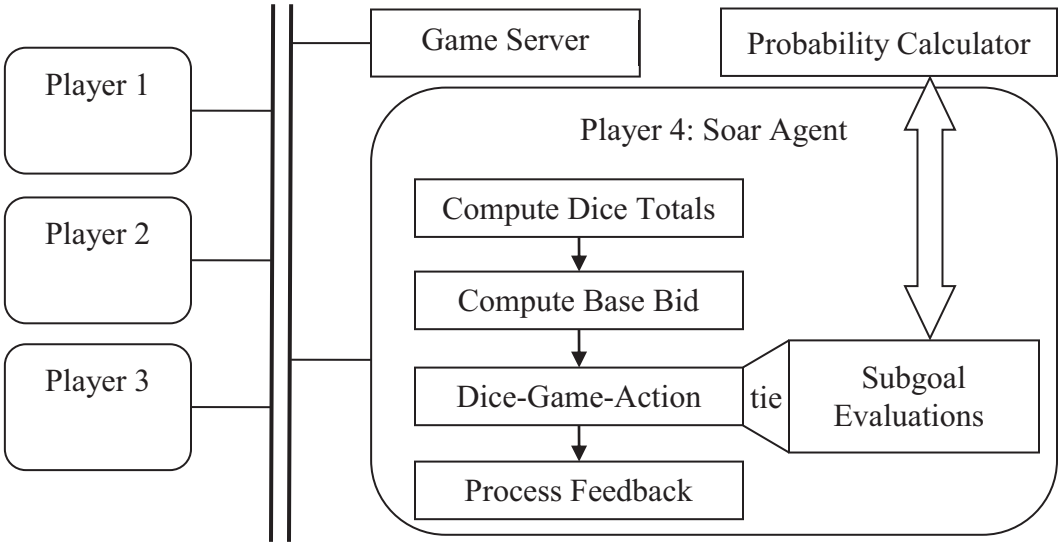

Figure 1. Dice game system.

agents, that connect to the game server on the same computer. Usually there are between two and six total players, but there is no inherent upper bound, and Soar can be used for any number of players. In the figure, there are four players, with the Soar agent playing as player 4.

When it is the agent's turn, it receives a description of the current state of the game that is equivalent to the information available to human players. This includes the number of dice under each player's cup, players' exposed dice, the dice under the agent's cup, and the history of bids.

The basic structure of the agent is straightforward, with the following processing performed by Soar operators, implemented with 370 rules. First, the agent selects and applies an operator that computes the total number of dice of each face that it knows (those exposed from pushes plus those under its cup), and the number of unknown dice.

The next operator determines a base bid to be used for computing the set of possible bids the agent will propose. An agent can make any bid that is higher than the previous bid; however, the higher bids risk being challenged, so usually a player will make a bid that is close to the previous bid. The one exception is that sometimes human players make very low initial bids (possibly in frustration after losing a die), and basing a following bid on such a low bid wastes the opportunity to make a bid that forces more risky bids by following players, while still being safe. In the agents, we define a safe bid to be one that is at least one less than the expected number for a face given the current number of dice in play. For example, if there are fifteen dice in play, and 1's are wild, four 5's is a safe bid. If there is no previous bid, or if the safe bid is higher than the previous bid, the safe bid is used as a base bid to compute possible bids; otherwise the agent uses the previous bid as the base bid.

Once the base bid is determined, the agent proposes operators for all bids that are higher than the base bid, up to and including one full number higher than the base bid. Thus, if the base bid is six 4's, the agent proposes six 5's, six 6's, three 1's, seven 2's, seven 3's, and seven 4's. If there are relevant dice under its cup (dice with the same face as the bid or 1's), the agent also proposes bids with pushes for those faces. The agent also proposes available challenge, pass, and exact actions. We refer to the operators described above as *dice-game-action* operators.

The agent then selects a dice-game-action operator (based on deliberations described below), and submits the action to the game server. If the action is a challenge or exact, the game server determines whether the challenge or exact is successful, updates the number of dice for the players as appropriate, and provides feedback as to whether the action was successful. The game also provides feedback when the agent is challenged.

Dice-Game Agents: Selecting an Action

The complexity in the dice-game agents is in choosing between the competing dice-game-action operators. This is where different types of knowledge are used to control behavior.

In Soar, *preferences* are used to select between competing operators. Preferences can be either symbolic or numeric. One class of symbolic preferences creates a partial ordering between two operators, stating that one operator is *better* than another. There are also symbolic preferences that state that two operators are equivalent (an indifferent preference), that an operator should be selected only if there are no other alternatives (a *worst* preference), or that an operator should be preferred to all other operators (a *best* preference). Numeric preferences specify the expected value of an operator.

A decision procedure processes the preferences, using the symbolic preferences to filter the candidate operators. If all the remaining operators have numeric preferences, an operator is selected using a Boltzmann distribution based on the values of the preferences. If there are insufficient preferences to make a selection, an impasse arises. The

decision procedure and these preference types provide an architectural mechanism for integrating different types of knowledge.

In the Dice agents, when the dice-game-action operators are proposed, there are no preferences to prefer one operator to another, so an impasse arises. In response to the impasse, Soar automatically generates a subgoal in which other operators can be selected and applied to resolve the impasse. Thus, a subgoal allows for deliberate reasoning about which operator in the superstate should be selected, and that reasoning can incorporate the different types of knowledge described earlier. As in the original dice game task, the reasoning in the subgoal consists of the selection and application of operators, but in the subgoal the operators can analyze, evaluate, and compare the dice-game-action operators, with the goal being to generate sufficient preferences to resolve the impasse.

Probability or Expected-Number Calculations

In most previous Soar systems that reasoned in subgoals about which task operator to select, the reasoning involved internal searches using models of the tied task operators. However, given the uncertainty inherent to the dice game, that type of look-ahead search is not productive. The approach employed here is to select operators that evaluate the likelihood of success of the proposed dice-game-action operators, where success is defined as a successful challenge or exact, or a bid or pass that is not successfully challenged. We have implemented the two different methods for evaluating the likelihood of success of a bid described earlier: one based on probability calculations, and the second based on our anecdotal model of how humans evaluate their bids using deviations from the expected number of dice. During a game, the agent uses only one method.

In our expected-number model, the agent evaluates each bid by computing the expected number of dice for the face of the die of that bid. This involves adding the number of the known dice of the bid face (as well as 1's for non-1 bids), with the expected number of that face given the number of unknown dice. For example, if the agent is computing the likelihood of six 4's and there is one 1 and one 4 showing (or under the agent's cup), and there are nine unknown dice, then the expected number is 2 + 9/3 = 5. The divisor is 3 because both 1's (which are wild) and 4's contribute to achieving the bid. The agent takes the difference between the bid and expected number, which in this case is -1, and classifies the bid, which in this case is "risky." Conversely, if there are 15 unknown dice, the total expected number is 7, and the bid is classified as "safe." If the agent knows for certain that there are six 4's, because of what is exposed and under its cup, it is a "certain" bid. The agent similarly categorizes a challenge bid based on how much the previous bid deviated from the expected number. In this model, all calculations involve additions, subtractions, and divisions, and they appear to correspond to the calculations performed by human players.

As mentioned above, when using the expected-number model, the agent uses a simple classification system for bids based on deviations from the expected number and known values. The agent uses symbolic preferences to create equivalence classes for similarly classified actions (such as all risky bids) via indifferent preferences. Better preferences are also used to create orderings between equivalence classes (safe bids are better than risky bids). A random decision is made from among the remaining actions in the top equivalence class. Randomness is important because it makes it more difficult for other players to induce exactly which dice the player has under its cup. Using this approach, the agent does not explicitly decide to bluff (make a risky bid when safer bids are available); however, the agent will make risky bids when it has no better alternatives.

The probability-based model uses an external probability calculator (Figure 1), which the agent interfaces to through Soar's input/output system. An agent uses the calculator by creating a query, such as: determine the probability that given thirteen dice, there are at least five of them with a given face. The queries are parameterized by the number of distinct possible values: either three for non-1 bids, or six for 1 bids. The computed probability is assigned to a numeric preference for the appropriate dice-game-action operator. The final selection is made based on a Boltzmann distribution, giving some randomness to the agent's play.

Opponent Model

In playing against these agents, one apparent weakness is that they are quick to challenge whenever the expected number or probability of the previous bid is low. Thus, if a player has a high number of some dice face and bids accordingly, the Soar agent will often challenge (and lose a die). Although any one specific set of dice is unlikely, it is not unlikely that a player with five dice will have two to three of some dice face (especially when 1's are included) which can form the basis of their bid.

To remedy this problem (and experiment with additional forms of knowledge and reasoning), an abstract operator was added that embodies a model of the previous player's bidding, and attempts to induce the dice under the previous player's cup given the previous bid. This operator is proposed only if the prior player did not push and reroll, because following a reroll, there is nothing to be gained from the model. The agent always does an initial evaluation before using the model to determine if there are actions that are certain, independent of the information obtained from the model.

To use the model, the agent recreates the situation that existed when the opponent had to make its bid, with the dice under the Soar agent's cup being unknown to the other player. For example, if the opponent has four dice under its cup and has just bid four 2's, and the Soar agent has one 2 showing from a previous push, and has three additional dice under its cup, the Soar agent creates the situation where there is one 2 known, and seven dice unknown.

The agent then incrementally hypothesizes different numbers of dice with the face that was bid for the opponent (in this case 2), evaluating the likelihood of the player having that many dice and determining whether that number of dice would support the given bid. In this case, it starts with assuming that the opponent has one 2. If the number of dice gets high enough so that the likelihood of actually getting that number of dice is very low (indicating that the player might be bluffing), or if the number of dice gets high enough to support the original bid, the modeling stops. In this case, having one 2 is not unlikely and together with the exposed 2 it does not make a four 2 bid likely to succeed. Thus, the agent considers what would happen if the opponent had two 2's, and in this case, that number supports the opponent's bid. The agent uses the result (in this case that the opponent has two 2's and two additional dice that are neither 1's nor 2's) in recalculating probabilities (or the expected number) for its bids to select a dice-game-action operator. The rules to evaluate the likelihood of hypothetical opponent situations are distinct from dice-game-action task knowledge, and so the outcome of this process is independent of using the probability or expected-number calculations.

Expert Heuristics

Once the probability or expected numbers are computed, additional knowledge can be included that affects selection from among the dice-game-action operators in the top rated equivalence class. These heuristics test symbolic state information, such as relative action type (ex. bid versus bid and push; bid versus pass) or round history (ex. face of the previous bid, face of next player's previous bid), and produce additional preference selection knowledge. For our agents, we developed five heuristics and all of them condition upon symbolic features of state and result in symbolic preference knowledge. For instance, one heuristic states that within an equivalence class, a bid *without* a push is preferred to that same bid with a push. The result of these heuristics is that the top-rated dice-game-action operators are pruned, such that final probabilistic selection is limited to only the best-of-the-best action(s).

Results

To evaluate the contribution of these different types of knowledge, we created agents with combinations of modeling (M) and expert heuristics (H) on top of the expectation-based (E) and the probability-based (P) decision making agents. We played four-player matches with pairs of agent types. To eliminate ordering effects, each match consisted of 1000 games for each of the four unique orderings of two distinct players (X and Y) in a four-player match (XXXY, YYYX, XXYY, XYXY). We summed the results of the four orderings, giving a possible win total of 4000.

Our metric of interest is number of times an agent (say X) wins and we calculate statistical significance using a two-sided binomial test. Our threshold for significance is 2082 victories out of 4000 (p=0.01), which is 52.02%. The

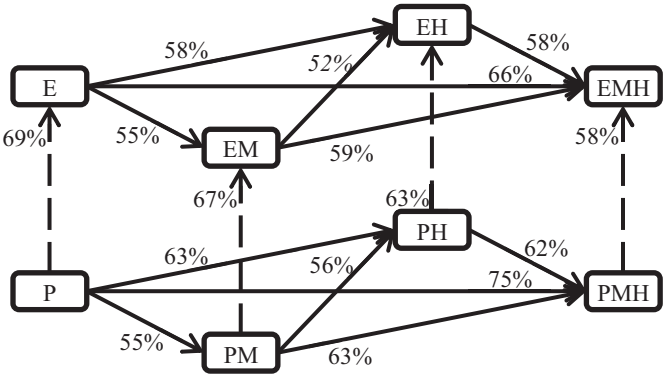

Figure 2. Pair-wise results of 4000 game competitions between agents with different types of knowledge.

results are summarized in Figure 2, where we report these as a percentage of possible victories for the dominant agent which has greater than 50% wins, rounded to the nearest percent. In the figure, each node is an agent, and the letters define its composition. For example, the EMH agent includes the expected-number decision making (E), modeling (M), and expert heuristics (H). The lines between nodes point to the dominant agent. In this case, all lines point right or up, indicating that the agents to the right and at the top (the E agents) won more often.

All of the winning percentages are significant except for EM to EH, where there were 2080 victories, as opposed to the 2082 required for significance. In all but that case, all the dominant agents had higher win totals than all the losing agents in all orderings. For example, in every ordering of the P vs. PH, the individual PH agents always won more games than any individual P agents did.

The most important results are clear. The model and the heuristics knowledge improve performance for both the probability-based and expected-number agents. The combination of those two types of knowledge also dominates each one individually.

Somewhat surprisingly, the expected-number agents dominate the probability agents, for all orderings. This result suggests that the expected-number calculations and symbolic categorizations capture enough of the underlying probabilistic structure that is relevant in the game, as well as include some additional structure not present in the pure probability calculations. However, as we describe below, the probabilistic agent opens up the unique ability to tune action-selection knowledge using reinforcement learning.

Informally, the EMH agents are competitive with human players. The EMH agents do not make obvious gaffs and they regularly beat experienced human players.

Learning

The Soar agents use the knowledge sources described above to deliberately reason about alternative dice-game-action operators in a subgoal. The results of those calculations are converted to preferences (by operators in the subgoal) for the dice-game-action operators. Soar's chunking mechanism compiles the deliberate processing in

```
Dice Rule 1:
If the operator is to bid five 2's with no push and
   there are zero 1's and one 2 dice, and four unknown dice then
create a numeric preference of -0.8754 for that operator

Dice Rule 2:
If the operator is to bid five 2's pushing one 1 and two 2's and
   the previous bid was two 1's and
   there are five dice under my cup, with one 1 and two 2's and
   the other player has only a single die, which is unknown, then
create a numeric preference of 0.12 for that operator

Dice Rule 3:
If the operator is to challenge and
   the bid by the previous player is two 1's and
   the player pushed and rerolled and
   there are two unknown dice and one 1 known, then
create a numeric preference of 0.38 for that operator
```

Figure 3. Example RL rules learned by chunking in the dice game.

subgoals into rules that create those preferences, without deliberation in a subgoal. Chunking leads to faster decision making; however, in this task the Soar agents are much faster than humans are, and the additional speed has little functional value. However, the rules that are created by chunking over probability calculations have numeric preferences as actions. Essentially, these rules are assigning expected values to subsets of operators.

Soar's reinforcement learning (RL) mechanism uses Q-learning to adjust the numeric preferences of such rules when there is a reward signal. We use a straightforward reward function: +1 for winning a challenge/exact bid and -1 for losing a challenge/exact bid. To align the probabilities with the reward function, the probabilities are linearly rescaled from (0, +1) to (-1, +1).

By having subgoals, chunking, and reinforcement learning in the same architecture, the agents have the following capabilities:

1. The agent initially uses a combination of probabilistic knowledge, symbolic heuristic knowledge and an opponent model to evaluate alternative actions.
2. Chunking automatically combines those forms of knowledge into new rules that compute parts of the value function, which is a mapping from states and operators to expected values. In Soar, a learned rule maps states and operators that match the rule conditions to the expected value in the rule action. If the subgoal creates symbolic preferences, rules to generate those preferences are also created.
3. With experience, chunking learns rules that incrementally fill out the value function, ultimately eliminating the need for deliberate reasoning.
4. When the agent uses learned rules to select a dice-game-action operator, reinforcement learning tunes those rules based on the reward received for performing the associated action. Over time, the rules approximate the actual expected reward as opposed to the original probability calculations.

Figure 3 shows three RL rules learned by chunking whose actions have been tuned a few times. Dice Rule 1 captures the probability calculation for an unlikely bid and is typical of many chunks which test the bid, the relevant known dice, and the number of unknown dice. The bid is five 2's, with one 2 known and four dice unknown, which all must be 1's or 2's for the bid to succeed.

Dice Rule 2 is also for the selection of an operator that bids five 2's, but this operator includes pushing three dice and rerolling two and is proposed when the opponent has only one die, which is unknown. Thus, the agent knows there are at least three 2's and must pick up two more out of the two dice it rerolls plus the one die under the opponent's cup. This outcome is unlikely if only the probability is considered; however, in this case the rule's value includes the result of modeling the opponent. Using the model leads the agent to conclude that in order to make the bid of two 1's, the opponent likely had one 1 (otherwise it would have challenged). Thus, the agent believes it needs to get only one 2 or one 1 out of the two dice it rerolls, giving a normalized value of 0.12.

Dice Rule 3 evaluates a challenge bid and gives it a high probability of success. In this case, the previous player bid two 1's and rerolled. There are two unknown dice and one 1 is known. Because of the reroll, the agent model was not used. Notice that this rule only tests the number of known dice and that one 1 is known, and that 1 could have been pushed by another player or it could be under the agent's cup. This situation probably arose when the current player had many hidden dice, and the previous player calculated that with the combination of a reroll and the current player's hidden dice, it was likely that there was a 1.

One characteristic of the chunks is that they are specific to the dice face being bid, and all are specific to the existence of 1's. Given the special nature of 1's as wild cards, they must be accounted for. In addition, the 2 bids cannot be generalized because they have different utility than bids of other dice faces. For example, in comparison to a 2 bid, a 6 bid forces the opponent to increase the number of dice bid, making such a bid less likely to be achieved. This could increase the chance that the 6 bid will be challenged (because other bids are less likely to succeed). However, a 6 bid makes it less likely that the player must bid again, as the higher bid may lead to a

challenge among other players, which is the most desirable result as then the player is guaranteed to not lose a die.

Beyond the specificity in testing the dice faces, the rules generalize over other aspects of the situation when they are learned. For example, rule 2 does not test which other two dice are under its cup, except that they are not 1's or 2's. Similarly, rule 1 does not test the previous bid, nor does it test what other non-1 or non-2 dice are known to the agent, as they are irrelevant to computing the bid's probability.

Because the determination of the conditions of the new rules in dependent on the knowledge used in the subgoal, we expect that the value functions learned from different initial knowledge to be different. If the model is used, the learned rules will test features of the previous bid, whereas if neither the model nor the heuristics are used, we expect the learned rules to test only the player's dice, the number of dice under cups, and the exposed dice. When more features are tested, those rules will be more specific, and more rules will be learned. We expect both of these factors to influence the speed and quality of learning.

To evaluate the performance of agents using learning, we used two-player games. Two-player games are easier and faster to evaluate, and require smaller value functions than four-player games – two-player games start with 10 dice, whereas four-player games start with 20. The learning agents all played against the best hand-coded agent for two-player games, which is the PMH agent. One question is why the EMH agent is best for four-player games, but the PHM agent is best for two-player games.

We evaluated two variants of the probability-based agents. In the first variant (labeled with -0), the probability calculation is disabled and the initial expected value of the RL rules is initialized to 0.0 to eliminate the contribution of the initial probability calculation. Differences in learning can be ascribed to differences in the value function defined by the learned rules, and to differences that arise from the inclusion of heuristics in the H and PMH agents. In the second variant (labeled with -#), the agents initialize the expected value of the RL rules using the scaled probabilities. In this variant, the agents' initial performance is determined by the probability calculations, which are then tuned by RL.

We include a baseline agent (B-0) that was created by generating RL rules using templates. These rules test the dice directly relevant to the bid being evaluated (such as the number of known 6's and 1's for a bid of five 6's), and the number of unknown dice. In these rules, totaling 153,132, the expected value is initialized to 0.0.

In our evaluation, 1000 games of testing (chunking and reinforcement learning disabled) are alternated with 1000 games of training (chunking and RL enabled), for a total of 35 blocks of training and testing. All data are the average of five independent runs. Each game takes between between 5 and 13 rounds of play (players can win back a die with an exact).

Figure 4 shows the testing trials of all the -0 agents. All agents significantly improve performance with experience, with three agents beating the baseline more than 60% of the time. The B-0 and P-0 agents have similar value functions and thus learn at about the same rate. The PH-0 agent learns fastest early on, but appears to level off just above 60%. The PMH-0 agent learns slower than PH-0, but appears to be slowly improving even at 35,000 games. The PM-0 agent starts slower than PH-0 and PMH-0, but ultimately win 70% of its games, and appears to be improving even after 35,000 games.

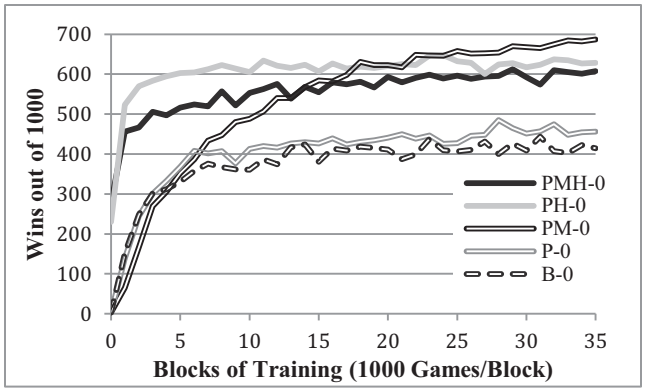

Figure 4. Performance for agents with learned rules with expected value initialized to 0.

The two most important factors for the differences in learning behaviors are the value function of each agent, as defined by the RL rules it learns, and any heuristics (in H agents) that are incorporated in symbolic selection rules. Figure 5 shows the number of rules learned by the different classes of agents. The P and PH agents learn fewer rules, and learn them early. The PM and PMH agents learn an order of magnitude more rules and continue to learn new rules throughout the training, with no end in sight.

Agents with more rules split the value function into finer-grain regions, giving them the potential of ultimately achieving better performance (which is why PM-0, PH-0, and PMH-0 perform better that B-0 or P-0). However, when there are more rules, each rule receives fewer training updates, leading to slower learning, as demonstrated by the PM-0 and PMH-0 agents. One hypothesis is that both PM-0 and PMH-0 will continue to improve with experience as they learn and tune new rules, while B-0, P-0, and PH-0 will not. One surprising result is that the PH-0 agent does so well with so few rules,

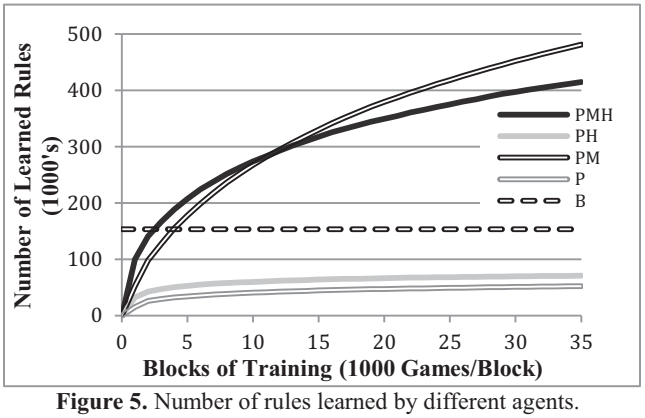

Figure 5. Number of rules learned by different agents.

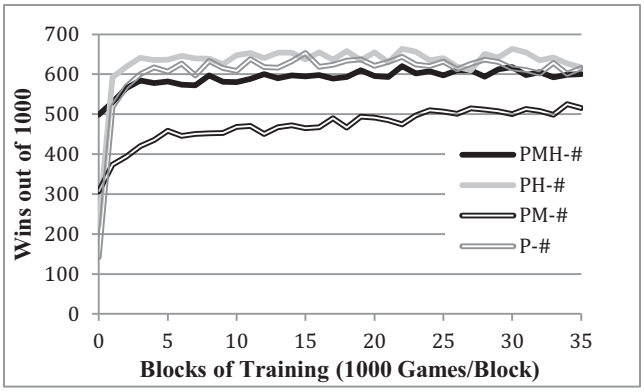

Figure 6. Performance for agents with learned rules initialized by internal deliberation in subgoals.

suggesting that the non-RL rules play an important role even though they are not tuned by RL.

Figure 6 shows the results when the learned rules are initialized with the calculated probabilities, and Figure 7 shows the differences between the agents' performances in Figures 5 and 6. As expected, initial performance is significantly better for all the agents, suggesting that deliberate reasoning can usefully initialize Q values for reinforcement learning. However, there are some striking results. The P-# and PH-# agents achieve high performance quickly suggesting that although the probabilities are useful, performance can be much better with a little tuning. We hypothesize that tuning of challenge bids may be responsible for these improvements, but more analysis of the learned rules is necessary to verify that hypothesis.

Another notable result is that although the PM-# agent gets an initial boost, it then learns slower than the PM-0 agent does. One distinguishing characteristic of the PM and PMH agents is that they learn new rules throughout the trials. If the initial values of these new rules need to be tuned for good performance (as suggested by the improvements in the P and PH agents), then we expect that the PM-# and PMH-# agents will require many more training trials to level off, not just because they have more rules, but because new rules disrupt the decision making. It is unclear why the PM-0 and PMH-0 agents learn faster, but it might be that once some tuning has occurred, the computed expected values are more disruptive than initial values of 0.0.

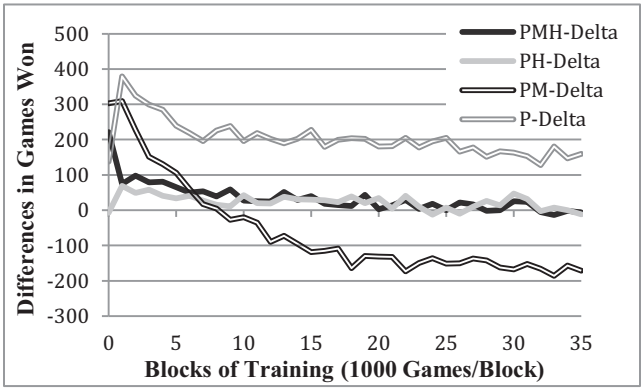

Figure 7. Difference in performance between agents with and without initialized expected values for RL rules.

Discussion

We started this project with the goal of investigating how a task that appeared to require probabilistic knowledge could be attacked with a symbolic cognitive architecture. These agents demonstrate how Soar's architectural components (preferences, impasses, subgoals, chunking, and reinforcement learning) provide the necessary structures for combining probabilistic and symbolic knowledge for decision making and learning in a task with high uncertainty. Our agents initially use deliberate reasoning to compute probabilities or expected values, which can combine heuristic knowledge, and a model of the opponent to make decisions. Through chunking, the agents can combine these types of knowledge and create new rules, and through reinforcement learning tune the rules to achieve significant improvements in performance.

The dice game has turned out to be surprisingly rich. In four-player games, the expectation-based decision making dominated the probability-based decision making. However, our experiments with learning in two-player games based on the probability-based agents, demonstrated the power of learning, with some agents achieving performance, which if duplicated in the four-player games, should dominate the expectation-based agents. The learning results also emphasized some of the tradeoffs in having small vs. large state spaces for value functions.

References

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., Qin, Y. 2004. An Integrated Theory of the Mind. *Psychological Review* 111 (4): 1036-1060.

Derbinsky, N., Laird, J. E. 2009. Efficiently Implementing Episodic Memory. Proc. of the Eighth International Conference on Case-Based Reasoning. 403-417, Seattle, WA.

Derbinsky, N., Laird, J. E. 2011. A Functional Analysis of Historical Memory Retrieval Bias in the Word Sense Disambiguation Task. Proc. of the 25th AAAI Conference on Artificial Intelligence. 663-668, San Francisco, CA.

Derbinsky, N., Laird, J. E., Smith, B. 2010. Towards Efficiently Supporting Large Symbolic Declarative Memories. Proc. of the Tenth International Conference on Cognitive Modeling. 49-54, Phil, PA.

Laird, J. E. 2008. Extending the Soar Cognitive Architecture. Proc. of the First Conference on Artificial General Intelligence, 224-235. Amsterdam, NL.: IOS Press.

Langley, P., Cummings, K., Shapiro, D. 2004. Hierarchical Skills and Cognitive Architectures. Proc. of the 26th Annual Conference of the Cognitive Science Society, 779-784. Chicago, IL.

Wintermute, S. 2010. Abstraction, Imagery, and Control in Cognitive Architecture. PhD diss. Computer Science and Engineering Dept., University of Michigan, Ann Arbor, MI.