A Case Study in Integrating Probabilistic Reasoning 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

In this paper we explore the role of probabilistic reasoning in a symbolic cognitive architecture. We present a case study of Soar agents that play a multiplayer dice game, in which uncertainty and probabilistic reasoning play a central role, and discuss and evaluate the efficacy of representing, reasoning with, and learning combinations of symbolic and probabilistic knowledge that we derive from a task analysis. We demonstrate that through chunking, an agent can convert deliberate reasoning about 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 symbol structures. In all of 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 plays a multiplayer dice game where uncertainty dominates 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 of the game in which we make four observations about the type of knowledge and mechanisms that might be useful in an agent that plays the game. We then describe the overall structure of our agent, and then focus in on decision making, where we incrementally describe how we developed agents that can use the four types of knowledge described earlier, providing empirical evidence of their usefulness. We develop two alternative approaches to decision making, that use of Soar's preference and impasse mechanisms.

One claim is that Soar's decision-making mechanisms (preference-based operator selection and impasse-driven deliberation), and its procedural-learning mechanisms (chunking and reinforcement learning) provide the necessary architecture mechanisms to not only support probabilistic and symbolic reasoning and learning independently, but to also support combinations of probabilistic and symbolic reasoning and learning.

One contribution of this paper is a demonstration of using Soar's chunking mechanism to compile deliberate probabilistic and symbolic reasoning into rules that are then tuned by RL using in-game experience. The conditions of the learned rules are determined by the reasoning and they partition the state space for learning. The actions of the learned rule provide initial values that are then tuned by reinforcement learning. This unique combination of chunking and RL leads to high initial performance that improves with experience, avoiding the long training times commonly associated with RL.

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 or our agent. The game begins with the

players positioned in a random circular 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. At the beginning of a game, the first player of a round is chosen at random, and following a player's turn, play continues to the next player based on the assigned circular ordering.

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, or both. 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.

The 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. When a player loses all dice, he/she 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) 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 is a claim that there is exactly the number of dice of the face claimed in the previous bid. An exact bid fails if there are either more or fewer dice of the face than bid, and the player loses a die. If the exact bid is correct, the player gets back a lost die. Finally, a player with more than one die can "pass," which is a claim that all of the player's dice are the same. 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 is dependent not only on the player's dice, but also on the dice unknown to the player. Because of this wealth of uncertainty, it would appear that reasoning directly with probabilities would be useful if not necessary. For example, in making a challenge, it would be useful to know what the probability is that the challenge will succeed. That probability can be compared to probabilities for other actions that agent can take, such as making a specific bid. However, through discussions with human players, we have found people do not compute the actual probabilities, but instead first compute the expected number for a specific face. For non-one bids, they sum the number of known dice (exposed and under their cup) of the face under question with 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 pure probability calculation is significantly better than the expected value 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, one important observation is that you don't lose a die if you don't challenge and if no player challenges you. The best possible result is for another player to challenge a third player, leaving you at no risk. Thus, although bluffing is part of the game, experienced players use it sparingly. Because of this, there is usually a correlation between a player's bid and the dice under its cup. Therefore, it can be 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 will refer to using this type of knowledge as having some type of *model* of another player.

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 usually wise to wait to use it 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 is unlikely to be challenged by the next player. A push reveals information to the other players and decreases the agent's future possible bids. Both of these examples are easily captured as symbolic rules, rules that depend on some evaluation of certainty of bids, but at a qualitative level and do not require reasoning about multiple probabilities. This is similar to "expert" knowledge as encoded in expert systems.

A final observation is that 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 probably 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 the only alternative is that they must be learned. However, a related observation is that without some significant abstraction, for a just two-player game, there are at least ten million states. Thus,

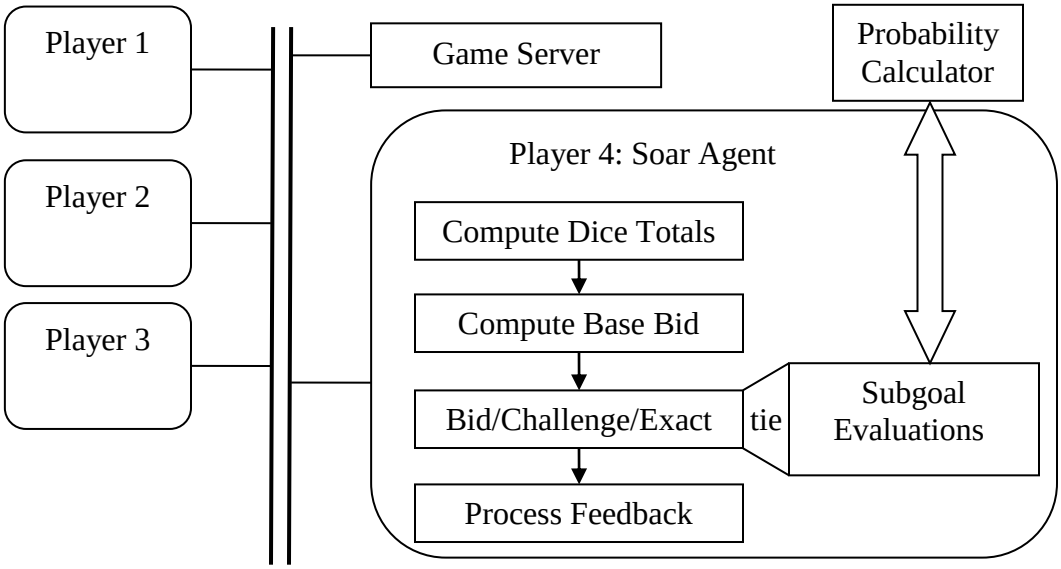

Figure 12.1 Structure of the Dice game system.

learning alone might require significant training, so starting with some initial knowledge could be extremely useful.

The result of this analysis is that there are four classes of knowledge to include in 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 (and learning). An additional challenge is to evaluate our agents. Because players reroll dice when they push after a bid, it is not possible to control a sequence of games across agent variants. The number of initial players and the ordering of those players can significantly affect individual player outcomes. Using two-player games is undesirable because they have unusual features that change game-play strategy: during every round an agent must challenge or be challenged.

Consequently, the data we collected and analyzed for this paper were the results of 1000-game tournaments. We always pair two agent variants but, to control for ordering effects, we employ four-player games, testing all four initial player orderings that are relevant to interactions amongst neighboring players (XXXY, YYYX, XXYY, XYXY). Our metric of interest is number of times an agent wins a game and we calculate statistical significance using a one-tailed binomial test, comparing the improvement the agent makes over the expected number of player victories (number of games / number of players = 1000 / 4 = 250). Our threshold for significance is 282 victories (p=0.01).

Dice Agent: Overall Structure

Figure 12.1 shows the overall structure of 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 agents, that connect to the game server either on the same computer or over a network. Usually there are between two to 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 players' cup, the exposed dice for all players, the dice under the agent's cup, as well as the history of bids. As currently implemented, the agent is active only during its turn, although in the future one could imagine the agent deliberating over bids of other players when it is not its turn.

The basic structure of the agent is straight forward, 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 general, a safe bid is one that is one below 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 incrementally 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 proposes bids with those faces along with an associated push. It also proposes challenge, pass, and exact action, if they are legal at that point in the game. We refer to the operators described above as *dice-game-action* operators.

The agent then selects an action (based on deliberations described below), and submits it to the game server. If the action is a challenge or exact, the game server determines whether the challenge or exact is valid, 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 Agent: Selecting an Action

The complexity of the Dice agent 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. Symbolic preferences state can create a partial ordering between two operators, stating that one operator is *better* than another. There are also symbolic preferences that state 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, a Boltzmann selection is made based on the values of the preference. If there are insufficient preferences to make a selection, an impasse arises. The decision procedure and these two types of preferences provide a primitive mechanism for integrating different types of knowledge.

In the Dice agent, 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 analyze, evaluate, or compare the dice-game-action operators, with the ultimate goal of generating symbolic or numeric preferences to resolve the impasse.

Probability or Expected-Number Calculations

In most previous Soar systems that reasoned in subgoals about which operator to select in a superstate, the reasoning involved internal searches using models of the tied 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 that was 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 5's and there is one 1 and one 5 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 5'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 5'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 simple additions, subtractions, and divisions, and they appear to correspond to the calculations human players often perform.

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 include any explicit concept of bluffing (such as making a very risky bid to force higher bids by opponents); however, the agent

Ordering	Best/Position
PPPE	Expectation/4 (wins=379, p<0.01)
PEPE	Expectation/4 (wins=289, p<0.01)
PPEE	Expectation/3 (wins=423, p<0.01)
PEEE	Expectation/2 (wins=365, p<0.01)

Table 1. Comparison of (P)robability and (E)xpectation models. Position is relative to ordering, where the first position is 1. Showing only the highest scoring agent.

will make bids that are essentially bluffs when it has no better bids to make.

To incorporate probabilistic calculations, we added an external probability calculator to Soar, which the agent invokes through the input/output system. The calculator provides an example of how complex numeric calculations can be performed by a Soar agent using external software modules. To use the calculator, the agent makes a query, such as: determine the probability that out of thirteen dice, there are at least five of them with a given face. The calculations are parameterized by the number of face on the dice – either three for non-1 bids, or six for 1 bids. The computed probability is assigned to numeric preferences for the appropriate dice-game-action operator. The final selection is made based on a Boltzmann distribution so that there is some randomness in the agent's play.

We played the probability agent against the expected-number agent and, somewhat surprisingly, found that for all orderings, when there was a statistically significant best player, it was the expected-number agent (see Table 1). We verified that this outcome also occurs, although to a lesser extent, when we add the two other forms of knowledge (opponent modeling and expert heuristics), but we do not present those data. This result suggests that the expectation-based calculations and symbolic categorizations capture enough of the underlying probabilistic structure that is relevant in the game, as well as some additional structure not present in the pure probability calculations. However, as we shall describe shortly, the probabilistic agent does open up some unique possibilities for tuning action selection knowledge using reinforcement learning.

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 an usually 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 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 is 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, in a subgoal, 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 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, and then selects an action. 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 not affected by whether the agent is using the probability or expected-number calculations.

To evaluate the efficacy of using the model, we played both of our calculation methods (expected-number in Table

2, probability in Table 3) with and without a model. For all but one configuration we found that the model yielded the best overall player, with the remaining configuration still showing an improvement using the model. Additionally, in all but one configuration the results were statistically significant. Finally, we notice that the relative advantage afforded by the model is greater for the probability agent as compared to the expectation agent.

Ordering	Best/Position
EMMM	Model/3 (wins=265, p>0.01)
EEEM	Expectation/3 (wins=286, p<0.01)
EEMM	Model/3 (wins=306, p<0.01)
EMEM	Model/4 (wins=329, p<0.01)

Table 2. Comparison of (E)xpectation against Expectation using opponent (M)odeling knowledge. Position is relative to ordering, where the first position is 1. Showing only the highest scoring agent.

Ordering	Best/Position
PPPM	Model/4 (wins=373, p<0.01)
PMPM	Model/2 (wins=340, p<0.01)
PPMM	Model/3 (wins=358, p<0.01)
PMMM	Model/2 (wins=325, p<0.01)

Table 3. Comparison of (P)robability against Probability using opponent (M)odeling knowledge.

Expert Heuristics

Once the probability or expected numbers are computed, additional knowledge can be included that affects selection from among dice-game-action operators within 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 (i.e. all other things being equal), a bid *without* a push is preferred to a 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).

We evaluated efficacy of using the heuristics , we played both of our agents with models against versions with and without heuristics. For all configurations based on probability calculations (Table 4), using the heuristics yielded the best overall player, with the remaining configuration still showing an improvement using the heuristics. Additionally, in all but one configuration the results were statistically significant. For the expectation-based agents, the results (not shown) were mixed with none of the agents performing significantly higher than other agents in any of the configurations, possibly from ceiling effects.

Ordering	Best/Position
PHHH	Heuristic/2 (wins=290, p<0.01)
PPPH	Heuristic/4 (wins=268, p>0.01)
PPHH	Heuristic/3 (wins=300, p<0.01)
PHPH	Heuristic/2 (wins=287, p<0.01)

Table 5. Comparison of (P)robability with model against probability with model and (H)euristic knowledge.

The final agents that use all of these sources of knowledge, play at a level that is competitive with humans. We have not formally compared them to human players; however, in exhibition games they do not make obvious gaffs and they regularly beat experienced 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 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 operators. Soar's reinforcement learning mechanism automatically adjusts the numeric preferences of such rules. The rewards used in this case are straightforward: +1 for winning a challenge/exact bid and -1 for losing a challenge/exact bid. We make no claims as to the quality of this reward function, let alone its optimality.

By having subgoals, chunking, and RL in the same architecture we obtain the following:

1. The agent initially uses a combination of general symbolic background knowledge and an opponent model to deliberately evaluate alternative actions.
2. Chunking automatically combines those forms of knowledge into rules that compute parts of the value function used in reinforcement learning. Chunking includes only those aspects of the situation that are

```
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 0.0123 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 0.56 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 preference of 0.69 for that operator
```

Figure 12.2 Example RL rules learned by chunking in the Dice game.

relevant to producing the result leading to general RL rules. The action of the rule is the initial value. If the reasoning in the subgoal results in symbolic preferences, chunks for those results are also learned.

3. With experience, chunking learns RL rules that fill out the value function, ultimately eliminating the need for deliberate reasoning.
4. Based on reward, the agent tunes the values of the RL rules to be in line with the actual reward as opposed to the likelihoods computed in the subgoal.

Figure 12.2 shows three RL rules learned by chunking, whose actions have been tuned only a few times by RL. 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. Here the bid is for five 2's and there is one 2 known and four unknown dice, which must all 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 pure probability is considered; however, in this case the agent'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 value of 0.56.

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 is 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 lack of generality 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 probability of that bid.

In our preliminary evaluations, we played two learning agents against two common baselines. The first baseline is the agent that uses probabilities, but does not include any model or heuristic knowledge (P). The second baseline is the agent that uses probabilities and includes the model by no heuristic knowledge (PM). The second baseline provides a higher bar for the learning agents to achieve.

The first learning agent (L0) is the agent that uses probabilities, but during deliberate evaluation it sets the probability for every bid to 0. Attempting to learn with this agent tests the basics of reinforcement learning in our agents. Does the reward function help and do the RL rules effectively partition the value function? The second learning agent (L#) is the agent that uses probabilities without modification. This agent tests whether the initial

values for the RL rules are consistent with the reward function, and whether reinforcement learning can improve on the agents. In the future, we will test learning agents that include models and heuristics.

In our evaluation, each agent is run first without learning for 1000 games against the baselines. Then we alternative 1000 game batches of training, where chunking and reinforcement learning are enabled, with 1000 game batches of testing, where we run with chunking and reinforcement learning disabled. The testing results are illustrated in Figure 12.3. These agents learn between 15,000-30,000 RL-rules.

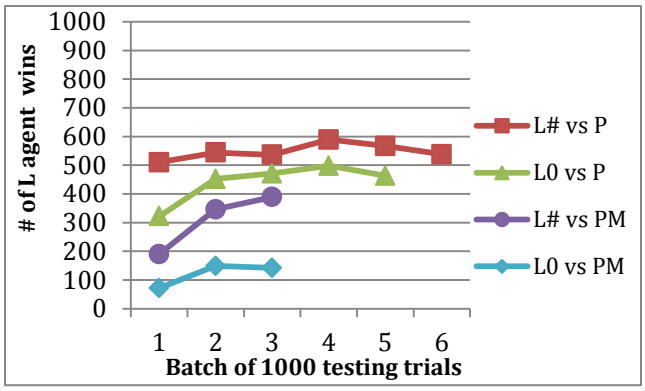

Figure 12.3. Performance for agents with learned reinforcement-learning rules.

Even with limited data, there are many important qualitative results. As expected, the agents playing against the P (no model) perform better than the agents playing against PM (model). Also as expected, the learning agents with RL-rules initialized with the calculated probabilities (L#) perform better than the agents with RL-rules initialized to 0 (L0). This demonstrates that agents can use deliberate reasoning to *usefully* initialize Q values for reinforcement learning. Finally, learning improved performance in all cases, even for the L# vs. P case, where the agent plays against itself, and ultimately performs better than the original agent. The L# agent wins 59% of time in the fourth testing trial before regressing in the fifth and sixth training trial.

Discussion

These agents demonstrates how many different architectural components in Soar work together, without any deliberate additional design or modifications to the architecture to combine probabilistic and multiple forms of symbolic knowledge. The agent developer does not need to modify or extend the architecture to get the benefits they provided by preferences, impasses, subgoals, chunking, and reinforcement learning. To compute probabilities, we added the probability calculator as an external device, but did not need to modify the architecture.

A second observation is that all these components are necessary to get the desired behavior of this agent.

- Without the background knowledge, probability calculations, and the opponent model, the agent would not have the knowledge to calculate the initial evaluations and in turn determine the conditions of the RL rules. Moreover, the background knowledge and opponent model enhanced the agent's performance beyond pure probability calculations. For example, if the opponent model is not used in the reasoning that led to Rule 2, the initial value of the action would be much lower.
- Without impasses and subgoals, chunking could not integrate these disparate forms of reasoning.
- Without chunking, it would not be possible to dynamically create RL rules, with their selective conditions and their initial values that are subsequently tuned by reinforcement learning. Until RL rules are created, the agent cannot learn from the feedback it receives from the environment.
- Without RL, there would be no tuning of the RL rules based on experience-based reward.

Clearly, more work needs to be done. We need to run the agents for more trials, test learning agents with model and heuristic knowledge, and train them against our best agents. We also need to compare our agents to humans.

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. It led us to two different approaches to deliberately reason about uncertainty and also led to unexpected and novel interactions between chunking and reinforcement learning.

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. In Proc. of the Eighth International Conference on Case-Based Reasoning. Seattle, WA.

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

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

Laird, J. E. 2008. Extending the Soar Cognitive Architecture. In 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. In Proc. of the Twenty-Sixth 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.