In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pages 823-830, Stanford, California, July 2000

Achieving Efficient and Cognitively Plausible Learning in Backgammon

Scott Sanner SANNER@ANDREW.CMU.EDU
John R. Anderson JA0S@ANDREW.CMU.EDU
Christian Lebiere CL@ANDREW.CMU.EDU
Marsha Lovett LOVETT@ANDREW.CMU.EDU
Psychology Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213 USA

Abstract

Traditionally, computer applications to game domains have taken a brute-force approach, relying on sheer computational power to overcome the complexity of the domain. Although many of these programs have been quite successful, it is interesting to note that humans can still perform extremely well against them. Thus we are compelled to ask, if no human could match the computational power of most of these programs, are there methods for learning and performance in game domains that more closely reflect human cognition? In response to this question, this paper attempts to model how humans learn and play games by developing a Backgammon-playing algorithm based on cognition. Analysis of this algorithm shows that it is efficient and commensurate with human abilities suggesting that it provides a cognitively plausible theory of learning in Backgammon.

1. Introduction

Since the advent of scientific inquiry into artificial intelligence, programming a computer to play board games has been one of the most frequent applications of AI research. Board game domains such as Chess, Checkers, and Backgammon have been popular since they have closed state spaces with well-defined rules. Yet, despite the relative simplicity of these games, they are not trivial since exhaustive search of the state space is difficult, if not impossible. Thus, artificial intelligence research in game domains has primarily concerned itself with explaining how a computer can play a game equivalently to or better than a human given the intractability of an optimal solution.

1.1 Background

In an effort to develop computer-based game playing opponents, many programs have achieved amazing feats – in many cases matching or surpassing the level of play demonstrated by a human grandmaster. Hi-Tech (Berliner, 1986) and Deep Blue (Hsu, 1990) have played at the master and grandmaster level respectively in the game of Chess. Chinook (Russell & Norvig, 1995) has achieved world-champion status in Checkers. And various Backgammon-playing programs have performed well at the world-class level, most notably TD-Gammon (Tesauro, 1992) which has played on par with the world's highest-ranking players.

All of these programs provide good theories of learning or search in complex game domains but they require massive amounts of computation to perform at their respective levels of play. For example, Deep Blue can search to a depth of 14 levels, examining millions of board states by virtue of its parallel custom-designed hardware. And the best version of TD-Gammon requires 1,500,000 training games and a three-ply search to achieve its optimal level of performance (Sutton & Barto, 1998). Although these programs have achieved amazing performance levels, Noam Chomsky (1992) has criticized this aspect of game-playing research as being "about as interesting as the fact that a bulldozer can lift more than some weight lifter."

1.2 Research Goals

If humans cannot reasonably approach the search or training requirements of these programs[1] yet still manage to perform comparably, we are compelled to conclude that humans must use more efficient methods for learning and performance in these domains. This is not to say that current programs do not map onto human approaches to game playing but rather that they make demands as a whole that are beyond reasonable cognitive limitations.

Consequently, in modeling human game playing, it is our goal to find a game domain in which we can address these

[1] It should be obvious that it is impossible for any human to search a million chessboard states or even a fraction of that in a turn lasting only a few minutes. With respect to training in Backgammon, a beginner playing a brisk game every 15 minutes for 8 hours a day could only play 1,000 games in a month.

cognitive limitations and determine how they influence algorithm development. With this purpose in mind, the game domain of Backgammon was chosen for a number of reasons: First, there are many possible moves for each turn making the game tree extremely large and implausible for exhaustive search. This is only compounded by the fact that the Backgammon domain is stochastic, relying on dice rolls to decide the set of possible moves. Thus, Backgammon is a game where recognition of board states and their relative utilities is a useful approach to move selection making it a good candidate for inductive learning methods that have been modeled quite extensively in human cognition.

With the domain defined, we must now ask what algorithmic constraints are imposed by cognition. In reference to search, if we accept that the minimum time required for deliberate human thought is 100 ms (Newell, 1990), the maximum number of states that one could search in a turn lasting one minute is on the order of a few hundred. And if we accept our overestimate that at most a human could play 1,000 games of Backgammon in a month, we would expect a human to achieve a reasonable performance level within a few months since this training is on par with only the most serious players. Thus, the goals of our research are to minimize search to a few hundred states on any turn and yet achieve a reasonable performance level in only a few thousand games.

2. Implementation

Before we build a Backgammon player it would probably help to have a brief overview of the domain. Then we can proceed onto the influence of cognitive models on the player design, the representational abstraction used by the algorithm, the equations used for generalization and learning, and finally the overall architecture.

2.1 Backgammon Overview

Since the reader may be a little unclear on the terminology and setup of the game of Backgammon, an example board is displayed in figure 1. In this board there are 24 labeled points consisting of an initial configuration of 15 checkers for each player. The players must move in opposite directions (in this case the black player moving by increasing point number and the white player moving by decreasing point number). For each turn, the player rolls two dice and is allowed to move a checker the distance of each die roll, or in the case that doubles are rolled, two checkers the distance of each die roll. If a player exposes one checker alone on a point, it may be attacked by an opponent and moved to the *bar* awaiting reentry into the board. A group of two or more checkers of the same color on a point constitutes a block and the opponent cannot land on this point. Thus, building adjacent points with two or more checkers can be a good strategy, blocking the opponent's forward progress. Once all of a player's checkers have been moved to any of the six points in their respective home board (i.e. the final six points in the direction the player is moving), the player can begin to bear checkers off of the board based on the die rolls. The first player to successfully bear all checkers off of the board wins the game.

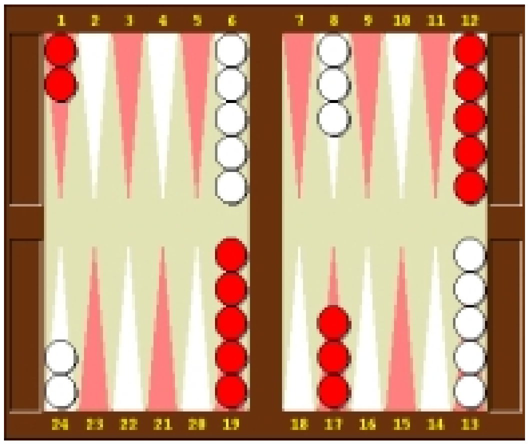

Figure 1. Backgammon initial board configuration

2.2 Player Design

Since our goal is to model human learning and performance, we will use the ACT-R theory of cognition (Anderson & Lebiere, 1998) to guide our algorithm development. And thus, in the spirit of TD-Gammon, from here out we will refer to our Backgammon-playing algorithm as ACT-R-Gammon.

2.2.1 Hints from Human Cognition

ACT-R is an empirically derived cognitive architecture intended to model the data from a wide range of cognitive science experiments. Although we will not be building our Backgammon player directly in the ACT-R architecture for reasons of programming efficiency, ACT-R is based on many empirically derived assumptions and mechanisms that should be useful in modeling our player on human cognition.[2]

One of the main tenets of ACT-R is that human cognition reflects the probabilistic nature of the environment and furthermore that this probabilistic behavior is Bayesian in nature. Consequently, this tenet suggests that we should focus on Bayesian inference methods as a means for inductive learning from experience.

Another tenet of ACT-R is that declarative memory retrieval is only approximate and places a preference on retrieving the item that most closely matches what is desired. This characteristic of cognition is useful for instance when trying to match a face with a name; the face may change over time but we are still able to make the correct association. This ability to perform

[2] Since we will only be using a subset of the ACT-R theory the complete system is fully contained within this paper.

approximate matching in ACT-R is known as partial matching.

One of the more recent research inquiries in ACT-R along the theme of partial matching involves the notion of generalization. What happens when we want to determine the utility of a certain move in Backgammon but do not have any direct experience with that move? How do we generalize based on one or more similar example moves that do not exactly match? One logical answer to this question involves the concept of blending. In this concept, example instances are combined based on their relative similarity to the target. How exactly relative similarity is measured is somewhat ambiguous, but in ACT-R one could easily use the partial match value to determine similarity since this is already an inherent feature of the architecture.

Consequently, it seems the ACT-R theory of cognition would suggest that a cognitively plausible model of learning in Backgammon should include some form of Bayesian inference and a generalization mechanism based on partial matching.

2.2.2 Representation and Abstraction

Before we define our learning and generalization equations, it is important to define a representational abstraction that reduces the hypothesis space while introducing as little inductive bias as possible. The fact that humans use abstractions in their representation of the environment should come as no surprise to a cognitive scientist or artificial intelligence researcher but the question of how this representation is formed is a little more enigmatic. However, automated feature learning is an entire field to itself and for the purposes of this research we will assume that ACT-R-Gammon has formed its representational abstraction prior to learning.

In defining the representation for Backgammon, it is important to include only features that are relevant to the game. For example in blocking an opponent, it is important to know how many of the opponent's checkers lie ahead of the block but when attacking an opponent, knowing how many checkers lie ahead of the attack is not necessarily relevant. Consequently we have defined three important features in the game of Backgammon and for each feature a number of relevant subfeatures:

1) Attack <Point>
2) Expose <Point, Opponents Ahead>
3) *Block <Point, Opponents Ahead, Size of Block>*

The angle-bracketed subfeatures of each feature represent relevant elements of that feature. Thus, an *attack* feature encodes the point of the Backgammon board on which the attack occurs. An *expose* feature encodes the point that a checker is exposed as well as the number of opponents lying ahead of the point that could potentially attack the exposed checker. And a *block* encodes the final point of a set of adjacent points with two or more checkers, the number of opponents lying ahead of the block, and the size of the block in terms of the number of adjacent points that the block occupies. Thus one could imagine the following two feature sets resulting from competing moves on a given turn:

Move Option 1:	Move Option 2:
1. Block<7,7,1>	*1. Block<7,7,1>*
2. *Block<12,7,4>*	*2. Block<11,7,3>*
3. *Attack<12>*	3. *Attack<12>*
4. *Attack<14>*	4. *Expose<12,6>*
5. Expose<14,5>	*5. Block<13*,6,2>

Consequently, ACT-R-Gammon's Bayesian learning task could be to infer the likelihood that each of the features resulting from a move would be present in a winning game. Then ACT-R-Gammon could simply combine these likelihoods to rate each move and choose the one of greatest advantage.

2.2.3 Matching and Generalization Equations

Given these assumptions, the main inductive component of ACT-R-Gammon should learn a set of features and their likelihood of occurring given a win or loss. To do this we can endow ACT-R-Gammon with a declarative memory of previously experienced features including the number of wins and losses in which each feature was involved. Then, if we endow ACT-R-Gammon with a method for generalization, we can derive the estimated likelihood for any feature and combine these estimates to derive the overall likelihood of a feature set resulting from a move.

However, to store every encountered feature would place too much of a computational demand on the system and likely defeat any psychological plausibility. Thus, we will selectively store features only if there are no similar features currently in memory. If we successfully query the declarative memory store for a feature within some similarity threshold, we will use that feature for the likelihood estimate and update it with experience. If no feature is found within the similarity threshold, we will store that feature (i.e. chunk the feature) and use that in the future. Thus, the feature space for ACT-R-Gammon becomes a fairly uniformly distributed set of instances with its boundaries defined by experience.

Given a feature representation and a method for selectively storing features, we now need to define the equations for generalization from this feature store. The generalization or partial matching (PM) equation as it is known in ACT-R is quite simple and is based on empirical research that suggests that humans perform matching in much the same way. Note that the following match function is defined in terms of the mismatch between features, consequently, as the mismatch between two features increases, the partial match penalty will likewise increase.

$$PMPenalty(A,B) = \frac{1}{|A|} \sum_{1 \le i \le |A|} Mismatch(A_i, B_i) \qquad \text{(Eq. 1)}$$

A and *B* in Equation 1 are features (e.g. *attack*, *expose*, or *block*) and A_i and B_i are subfeatures of their respective features (e.g. *point*, *opponents*, *size*). The *Mismatch* equation mentioned here is quite simple and varies linearly from 0.0 for a perfect match to 1.0 for the maximum mismatch between feature slots A_i and B_i. One addition here to the general ACT-R equation is the presence of the $1/|A|$ factor in front of the mismatch summation. This is included to normalize the minimum and maximum penalties to values between zero and one thus allowing the *PMPenalty* to be interpreted as a percentage mismatch. This seems to make sense in general since we would rate a match between two features each having a single similar and single differing component as poor, while we would rate a match between two features each having 99 similar components and a single differing component as much better.

Thus, given the *PMPenalty* equation, we can simply set as an ACT-R-Gammon parameter a threshold for the maximum penalty to allow when considering a match. The higher this threshold, the more generalization there is and consequently the less specialization.

If each feature has a likelihood or more generally a local evaluation of its utility function *U()*, we can generalize to the utility value of a target feature based on a number of similar instances in a variety of ways. One method which is a recent proposal in ACT-R is that of blending (Lebiere, 1999), also referred to in machine learning terms as the weighted nearest neighbor algorithm (Mitchell, 1997). This equation is shown below:

$$U(A) = \frac{\sum_{i=1}^{k} w_i U(B_i)}{\sum_{i=1}^{k} w_i} \qquad \text{(Eq. 2)}$$

Here, *A* corresponds to the target feature and B_i corresponds to a matching feature in the feature space (not a slot of feature *B* as in Equation 1). w_i corresponds to $(1 - PMPenalty(A,B_i))$ making it essentially equivalent to the distance in terms of match penalty between the target feature *A* and an arbitrary feature B_i chunked during ACT-R-Gammon's experience.

Thus Equations 1 and 2 allow ACT-R-Gammon to match features from previous experience and generalize to the utility of arbitrary features.

2.2.4 Learning Equations

Based on the ACT-R assumptions about human cognition, it seems logical to use Bayesian odds likelihood estimates (i.e. ratio of feature probability given a win to the feature probability given a loss) for feature utility. In this way, the odds likelihood of each feature resulting from a move can be multiplied under a naïve Bayes assumption to yield an overall odds likelihood of the move. This approach has some useful characteristics that have been found to be empirically quite useful: If a feature utility is neutral in predicting the game outcome (i.e. equal likelihood of leading to a win or loss), the utility works out to be 1.0. When multiplying all of the features resulting from a move to yield the utility of the move, such a neutral feature will have no effect on the outcome. Furthermore, if a feature utility is either highly likely to lead to a success or failure then its odds are either <<1 or >>1 resulting in a feature product that is largely skewed by this single extreme feature. This sort of estimate seems to be in line with human reasoning since we are likely to evaluate a possible move as good or bad if it has one outstanding feature that we recognize from previous experience.

Thus, given the need for a Bayesian style of inductive inference, all we need to do is derive the feature odds likelihood equations. For any feature *f*, we can directly measure the frequencies *F(W & f)* and *F(L & f)*, where *W* and *L* respectively denote win and loss. We can also measure the number of total wins and losses and thus determine *F(W)* and *F(L)*. Thus, all we need for each feature is to determine the odds likelihood of its occurrence in a win vs. a loss from the above frequency measurements. The derivation for this is shown below.

From the basic Bayes rule equation we can easily derive Equation 3 which represents the odds of a win vs. a loss given that a feature is present:

$$\frac{P(W \mid f)}{P(L \mid f)} = \frac{P(W)}{P(L)} \bullet \frac{P(f \mid W)}{P(f \mid L)} \qquad \text{(Eq. 3)}$$

Rewriting Equation 3 and substituting frequency counts we get the odds likelihood of a feature:

$$\frac{P(f \mid W)}{P(f \mid L)} = \frac{F(f \,\&\, W)}{F(f \,\&\, L)} \bullet \frac{F(L)}{F(W)} \qquad \text{(Eq. 4)}$$

This is the individual odds likelihood for a given single feature *f*, but we need the overall odds of a move composed of a set of features, *F*. Thus, using a naïve Bayes assumption, we can derive the following calculation for the overall move utility (likelihood odds).

$$U(F) = \frac{P(F \mid W)}{P(F \mid L)} = \prod_{f \in F} \frac{P(f \mid W)}{P(f \mid L)} \qquad \text{(Eq. 5)}$$

Consequently, if we update the feature set that ACT-R has chunked during its experience with the number of wins and losses that each feature accumulates, we can easily determine the odds likelihood of a single feature in Equation 4. From this we can derive the overall odds likelihood of a move from Equation 5.

Given the odds likelihood measure of move utility, all that ACT-R-Gammon has to do when faced with a move is to determine the utility of each feature set resulting from a move and select the best one. Since ACT-R makes the assumption that human cognition is non-deterministic and since this is in fact advantageous for promoting exploration of the state space, ACT-R-Gammon probabilistically selects its moves on each turn according to the Boltzmann soft-max rule (using temperature *t* as a noise parameter). In general, it is a good idea to use logarithmic odds since this will prevent gross overestimates of feature utility. However, the soft-max rule uses exponential weighting; Thus, the exponential and logarithm cancel each other (i.e. $e^{log(U())} = U()$) resulting in the following equation that denotes the probability that the set of features F_i resulting from a move will be selected:

$$P(F_i) = \frac{U(F_i)^{1/t}}{\sum_i U(F_i)^{1/t}} \qquad \text{(Eq. 6)}$$

2.2.5 Final Architecture

Therefore, given the algorithm described and the equations derived above, we arrive at the following basic decision cycle for the ACT-R player:

```
For each turn
{
   For each move
   {
      *Match each feature for the move and get
       its odds estimate (Using partial match
       values for weighted nearest neighbor
       weighting and Equations 1,2,4)
      *If there is no match for a feature then
       chunk a new feature with neutral odds
      *Accumulate product of feature odds (Using
       Equation 5) to determine move utility
   }
   *Stochastically select best move (Using
    Equation 6)
}

When player attacked
{
   *Penalize the expose feature responsible for
    making a checker vulnerable to attack by
    incrementing it with a partial loss
}

At end of game
{
   *Update all features with win or loss based
    on outcome of game (When updating, update
    the feature store by the degree to which it
    partial matched the actual move)
}
```

One must note here that learning occurs on two distinct levels. Most obviously, when a game is won or lost, all features involved in that game are likewise updated with the outcome. But on a more incremental level, *expose* features are also updated with a partial loss if the checker made vulnerable by the expose is attacked. This may seem contrived and intended to artificially force the player to avoid exposing its checkers, but it is interesting to note that if a certain expose consistently leads to a win (e.g. in a bait tactic), the long-term win accumulation will overshadow the short-term partial loss accumulation.

It is also important to point out that this decision cycle is within the cognitive constraints outlined earlier. On any turn, the number of possible moves usually ranges from 20 moves for a typical throw of the dice to a maximum of a few hundred moves for a doubles roll. Thus, since ACT-R-Gammon only examines current moves and uses no look ahead, it never has to evaluate over a few hundred moves in a turn when making its decision – well within the first cognitive search constraint outlined earlier.

Overall, given its basis on a cognitive architecture and its operation within cognitive constraints, the ACT-R-Gammon architecture should approximate the decision process which humans use when playing Backgammon. It uses an abstraction that makes the search space more tractable; it relies on probabilistic estimates reflective of human reasoning to select future moves; and it has as its sole trainer the results of a game and knowledge of its checkers being attacked. Most of all however, the described algorithm does not place much demand on the learning system; Overall, a trained system requires fewer than 50 chunked features from which to generalize all of its feature likelihood estimates.

3. Results

Thus, with the architecture for the system defined, we now move onto the training and testing of ACT-R-Gammon. A number of variants of ACT-R-Gammon were tried, but the previously described algorithm using Bayesian odds likelihood feature utility estimates and weighted nearest neighbor blending for generalization performed best overall. Training results against Gerald Tesauro's single layer neural network opponent *pubeval* are described below.

3.1 Training Method

ACT-R-Gammon was trained for 1000 games against the *pubeval* opponent starting with no initial experience.[3] The average winning percentage vs. number of games played is displayed in figure 2. The best training run was saved and played for 5,000 games against *pubeval* with learning turned off; the result of this evaluation run along with two other machine learning approaches is displayed in table 1.

3.2 Training Results

We see in figure 2 that the average learning curve for the ACT-R player shows quick initial learning after which it

[3] Since training is stochastic, the training runs described below were based on the average of 20 separate training trials except where otherwise noted.

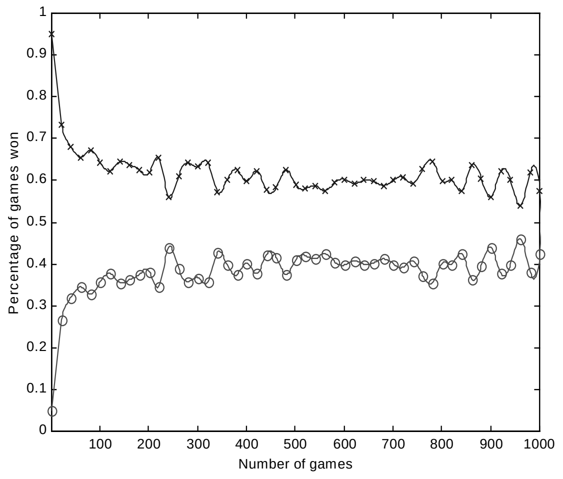

Figure 2. Training run of ACT-R-Gammon(o) vs. pubeval(x)

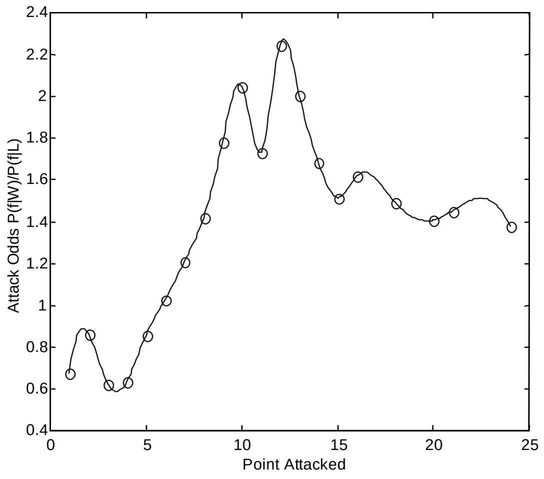

Figure 3. Attack odds learned by ACT-R-Gammon (10 run avg)

asymptotes to a fairly stable level. This result is well within our research goal of achieving significant learning in the first few thousand games and it is fascinating to see just how quickly ACT-R-Gammon actually learns. In 20 games it learns to over 50% of its asymptotic level and in 100 games it is within 15% of its asymptotic level.

3.3 Asymptotic Performance Comparison

One obvious question in response to figure 2 is how well ACT-R-Gammon is performing in comparison to other approaches. It has achieved quick initial learning, but is its performance against *pubeval* notable?

In comparison, Galperin and Viola (1998) give the asymptotic performance of playing a reinforcement learning trained neural net vs. *pubeval*. This is a quite computationally demanding training method based on the TD(λ) algorithm used by TD-Gammon and requires the simulation of over 1,000,000 games. In another approach, Pollack, Blair, and Land (1996) provide the asymptotic results of their genetic algorithm-trained neural net (named HC-Gammon) vs. *pubeval*. This method is also computationally demanding as well, requiring the coevolution of two neural net driven players over 100,000 games. Both groups claim their methods have produced respectable players and Tesauro is quoted in the latter paper as having stated that *pubeval* is "quite a strong machine player". A comparison of asymptotic performance rates vs. *pubeval* is shown in table 1.

Table 1. Performance results for three Backgammon players

Backgammon Player	Winning pct. vs. *pubeval*
TD-Trained Neural Net	59.25% ± 0.15%
ACT-R-Gammon	45.23% ± 0.71%
HC-Gammon	40.00% ± 3.46%

In light of this information, ACT-R-Gammon's performance seems impressive since it only had 1,000 training games as opposed to the TD-trained neural network with over 1,000,000 training games and HC-Gammon with 100,000 training games. Consequently, we have achieved our goal of obtaining reasonable performance within a few thousand training games thus satisfying our second cognitive constraint.

3.4 Learning Analysis

Now that we know that ACT-R-Gammon has achieved our original goals within the specified cognitive constraints, it is interesting to see what it has learned.

3.4.1 Attacking

Figure 3 shows the learned odds likelihood that an *attack* feature will be present in a winning game. The initial striking characteristic of this curve is its non-uniformity, but this turns out to likely represent common sense when it comes to attacking. That is, for the lower point positions of the attack odds curve, attacking an opponent early on in the board can be dangerous since it often involves exposing a checker. Since this can allow the opponent to build a blockade while the checker is stuck on the bar, it seems that the low odds of an early attack are probably related to the overall game state that leads to such a feature. However, after point 6, we see a steady rise in the odds of an attack until it peaks at point 13. This likely represents the fact that attacks can be carried out more cautiously as the position increases since enough men should be available at these points to avoid a potentially costly expose. The overall attack odds seem to slightly decrease from the middle of the board through the player's home board. This could be the result of one additional factor – as the point of attack increases, the opponent's cost of being attacked decreases since fewer dice rolls have been invested in getting the opponent to that point.

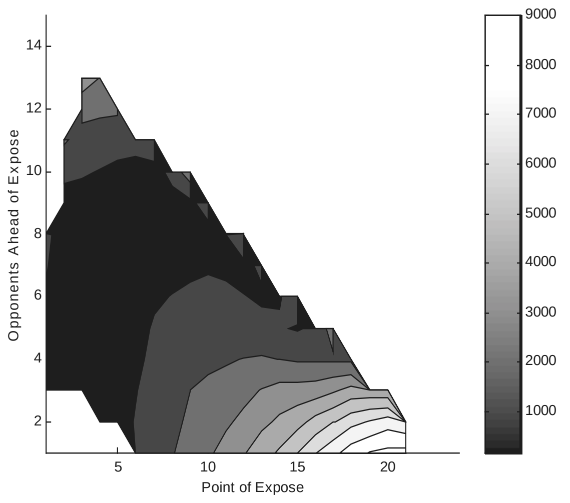

Figure 4. Frequency distribution of Expose Feature (5 run avg)

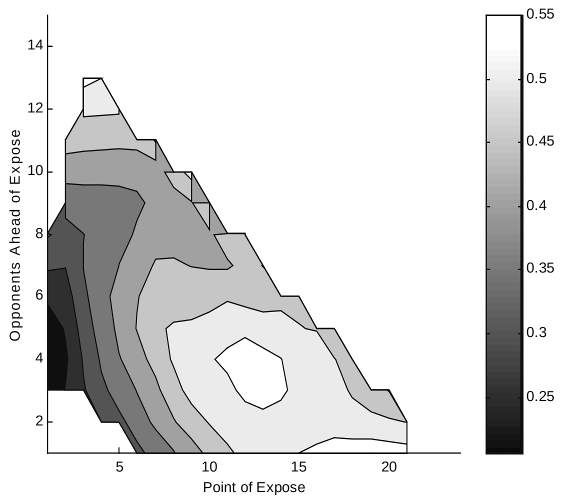

Figure 5. Expose odds learned by ACT-R-Gammon (5 run avg)

3.4.2 EXPOSING

Before we look at the learned odds for the *expose* feature, it is interesting to look at the experience distribution of exposing since this demonstrates a few notable characteristics of ACT-R-Gammon's learning algorithm. Figure 4 shows a contour plot of the experience received for the expose feature vs. the point that the expose occurred on (x-axis) and the number of opponents that lie ahead of the expose (y-axis). It is interesting to note that no features were chunked where no experience was gained (upper right quadrant of figures 4 & 5). As it turns out, only half of the feature space is ever visited making ACT-R-Gammon fairly efficient in its method of feature learning. Since we would expect more exposes to occur during the endgame we obviously see this trend in the number of experiences of an expose feature toward the higher points. However, this experience distribution seems to be only slightly correlated with the average learned odds for exposing. Thus, although features with higher odds are more likely to be selected, the experience distribution does not seem to skew the odds estimates – this is a nice result of using Bayesian odds inference.

In terms of the learned odds value of the expose feature shown in figure 5, ACT-R-Gammon never learns that an expose is advantageous (i.e. odds > 1), but it does have two distinct peaks where an expose seems to be less detrimental than in other positions. The first of these positions is early on in the board (i.e. low point position with many opponents ahead). This makes sense in retrospect since losing a man early in the board does not sacrifice a great amount of work. However, it does take the chance of getting blockaded by the opponent, which would explain why the odds are still in favor of losing. The other place that the odds seem to be least detrimental is around point 13 with 2-4 opponents ahead of the exposure. This may indicate one inadequacy in the expose representation in that it takes into account the number of opponents ahead of the expose but not their distances. It is likely the case that exposing under these conditions is least detrimental since a blockade exists that prevents the opponent from attacking these positions.

3.4.3 BLOCKING

Since the *block* feature has three subfeatures (i.e. point, opponents ahead, size of block), it makes for a difficult feature space to graph. Consequently, a selected listing of learned block features and their odds is shown in Table 2.

Table 2. Block odds learned by ACT-R-Gammon (1 run)

Point	Opponents	Size	Odds
1	10	1	0.17
4	5	1	0.13
7	9	1	0.63
12	0	1	0.63
12	7	1	1.79
16	5	2	2.48
20	3	2	2.54
21	5	1	6.94
22	4	3	3.86
24	1	5	1.22
24	3	4	3.79

Although one cannot easily get a good idea of the overall topology of the feature space from this table, there are a few important trends that represent ACT-R-Gammon's strategy. If one looks at any odds over a threshold of 1.5, it is obvious that these usually correspond to large adjacent blocks on the high points. As discussed earlier, this is an excellent strategy since it will generally cause ACT-R-Gammon to seek out positions in which it can blockade the opponent near the beginning of its board thus preventing it from entering into the endgame condition. Also generally, the only block features to show less than neutral odds are blocks that occur early on in the board or with few opponents ahead. In this case the block can either lead to potentially harmful board positions or it serves no purpose since the player cannot be attacked.

4. Conclusion

With the results now collected and analyzed, it is important to reflect on our work to determine which issues to further explore in the future.

4.1 Achievements

We have achieved our initial goal of cognitively plausible learning. That is, we have based our learning algorithm on the principles of a cognitive architecture used to model hundreds of psychological experiments. And we have analyzed the time scale of human cognition to constrain the amount of search and training to that which is psychologically plausible. Within these constraints we have achieved a reasonable level of performance, something we would rationally expect a human to accomplish in only a few thousand games. This shows that cognitively principled architectures are not limited to modeling psychology experiments – they can adequately scale to large problems in the domain of cognition.

Although the main goal of this project was to achieve a Backgammon player that was cognitively plausible, we can also reflect on how it can inform more performance-minded approaches. Probably the most important element of ACT-R-Gammon's efficiency was its use of feature abstractions. These abstractions greatly simplified the task of Bayesian inference and are one reason why it was able to beat HC-Gammon's performance level in 1/100th of the training time. The other reason was that ACT-R-Gammon could learn within each game from attack penalties as well as from each win or loss. These results suggest that feature abstraction and intermediate rewards could be integrated into current approaches to help them more quickly acquire their asymptotic performance level.

4.2 Further Exploration

Consequently, ACT-R-Gammon has demonstrated a lot of interesting results but there are many improvements that could be made to increase its psychological plausibility and performance. A rule inference system could be added to deal with move selection pruning and the nuances that the feature generalization does not catch. The feature chunking mechanism could chunk probabilistically to get better estimates of odds in heavily explored areas of the feature space. And we have not even attempted to explain how the feature representation is formed – this alone would be an interesting project.

But even with the current, in effect simple, ACT-R-Gammon player we have achieved our main initial goal. Using human cognition as our constraint and ACT-R as our theory, we have built a psychologically plausible Backgammon player that sharply contrasts the traditional brute force approach.

Acknowledgements

We would like to thank Richard Stern for his comments and suggestions during the completion of this research.

References

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Lawrence Erlbaum Associates.

Berliner, H. (1986). Hitech wins north american computer chess championship. *AI Magazine*, Winter, 30.

Chomsky, N. (1992). *Language and thought*. Wakefield, RI: Moyer Bell.

Galperin, G., & Viola, P. (1998). Rollout-based policy retraining. Artificial Intelligence Laboratory, MIT, http://www.ai.mit.edu/lab/abstracts/1999/ps/z-grg2.ps, (pp. 1-2).

Hsu, F., Anantharaman, T., Campbell, M., & Newatzyk, A. (1990). A grandmaster chess machine. *Scientific American*, 263:4, 44-50.

Lebiere, C. (1999). Blending: An ACT-R mechanism for aggregate retrievals. *Presented at the Sixth Annual ACT-R Workshop at George Mason University*. Fairfax, VA. http://hfac.gmu.edu/actr99.

Mitchell, T. M. (1997). *Machine learning*. Boston: McGraw-Hill.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Pollack, J. B., Blair, A.D., & Land, M. (1996). Coevolution of a backgammon player. *Proceedings of the Fifth Artificial Life Conference*. Nara, Japan. http://www.demo.cs.brandeis.edu/papers/long.html.

Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach.* Upper Saddle River, NJ: Prentice Hall.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Tesauro, G. (1992). Temporal difference learning of backgammon strategy. In Sleeman, D., & Edwards, P. (Eds.), *Machine Learning*, (pp. 451-57). San Mateo, CA: Morgan Kaufmann.