Using Probabilistic Programming for Game AI

Bogac Kerem Goksel (bkgoksel@stanford.edu)
Symbolic Systems Program,
Stanford University, Stanford CA-94305, USA

Abstract

Probabilistic programming is a powerful tool that can be used to model various cognitive phenomena. This paper explores the feasibility of writing a game playing AI using the probabilistic programming language WebPPL and explores the benefits and challenges of a probabilistic game AI by implementing an AI for a simplified version of the tile game Rummy.

Keywords: probabilistic programming, game playing, artificial intelligence

Research Question

Most contemporary game AI techniques involve searching over the possible game states from AI agents' perspectives, and choosing locally optimal actions for the agents themselves. However, the addition of inference on the goals' of other agents in this search process may lead to a more dynamic, believable, or overall fun AI system. The difference may be even more noticeable for games that require higher levels of player-AI interaction, such as cooperative games where knowing other agents' intentions makes a difference for AI agents, or games that require deeper thinking about the opponent's goals, such as Rummikub, the game chosen for this paper.

Background

This question is a recently popularized question in the video game AI research. There have been several attempts in the last 5 years to come up with different efficient ways of human goal inferring for video game AI. CAPIR by Singapore-MIT GAMBIT Game Lab approaches the problem as a Markov decision-making process, and decomposes games to subtasks. It then uses a probabilistic state machine to model the human intentions. Synnave and Bessiere use unsupervised machine learning on past game logs to build models for human plans for a specific game. Macindoe proposes another Markov-based system that also takes actions to learn the human goals better on top of inference. While these papers do not implement probabilistic programming techniques, the final implementations explore the same ideas of goal inference and information levels as this paper does.

Methods

The Game

The game that will be used in this paper is the tile/card game Rummy. The game runs on a deck of 106 tiles: eight tiles of number from 1 to 13, two copies in four different colors, and two jokers. All players are dealt hands of 14 tiles in the beginning of the game. The goal is to group the tiles in groups of same-valued but differently colored tiles or same colored tiles of consecutive tiles. The first player to be able to group all the tiles in their hand validly in groups of at least 3 tiles wins the game. One player starts with 15 tiles, and immediately discards a tile in the beginning of the game. From then on, each player has to either draw the tile discarded by the previous player or draw a random unused tile from the deck. After drawing one of these tiles, the player has to discard a tile from their hand.

For the sake of runtime, the game was simplified for this paper, with the tile number reduced to 16 by including only the tiles with values between 1 and 4, and removing the jokers. The player hands were also shrank to include only 4 tiles. The game was further simplified by reducing the number of players from 4 to 2.

After this, the game was fully implemented in naïve javascript with no AI, letting human players play. However, the implementation would be able to take in the AI actions or provide the AI agent with a snapshot of the game state for it to be able to decide.

The AI

The AI engine was written separately in a WebPPL file, with a smaller external JavaScript module to provide deterministic computation functions. It was designed so that it could accept the part of the game state that contains one player's information regarding the current game state, and it would return a probability distribution over the actions that player could take. It could be changed to return a specific action by sampling over this distribution.

The AI engine, upon receiving the game state, generates all the possible actions the player can take in that state, and generates a probability distribution over all possible outcomes that could happen after taking that action. It then samples an outcome from this distribution and if this ends up being a winning outcome for the player, it weights the original action more in the final distribution of the actions.

When generating a probability distribution over all the possible outcomes given a certain action, the AI engine infers the game state of the other player, and chooses an action as the other player. It then applies this action to the game state and chooses a new action with the updated game

state. This is done until the game is won by an either player. Since finding the outcome involves sampling many random parameters, many possible outcomes emerge and the AI agent factors in these possible outcomes using the Enumeration feature of WebPPL.

This approach works recursively, as guessing the action the other player will take involves calling the AI agent's decision on the inferred state of the other player. However, choosing an action for the other player involves figuring out the guess of the other player for this player's next action, which causes numerous calls to be made even for a single action to be decided. The AI engine prevents this by limiting the number of recursive calls made for efficiency purposes.

The main issues the AI engine needs to handle in this model are the representation of the game state, choosing actions, applying the actions to the game state and switching between the information scopes of different players.

Game State Representation Since the AI makes a decision just like any other player does, it has limited information about the game state. That is, it does not know the locations of all the tiles and it does not know what the opponent's hand looks like. For this reason, the entire AI engine runs on a limited portion of the game state that consists of the deciding player's information. The game state is taken as a JavaScript object from the game and is processed as such. A sample game state would look like this:

```
{    hand: [{value: 0, color: 0},
            {value: 1, color: 0},
            {value: 2, color: 0},
            {value: 3, color: 1}],
      myDrawPile:    [{value: 3, color: 0}],
      myDiscardPile: [{value: 0, color: 2}],
      inOthersHand:  [{value: 1, color: 1},
                      {value: 2, color: 1},
                      {value: 3, color: 2}],
      knownInMyHand: [{value: 0, color: 0},
                      {value: 1, color: 0}],
      unknownLocation: //Array of all the
remaining tiles
    }
```

Here, the hand is the array of the tiles the player has in their hand at this moment, myDrawPile is the array of all the tiles the other player discarded and the current player has not picked up so far, myDiscardPile is the array of all the tiles the player has discarded and the other player has not picked up, inOthersHand is the array of all the tiles that the current player knows the other player has in their hand at this moment, knownInMyHand is the array of all tiles that the current player knows that the other player knows and unknownLocation is the array of all the remaining tiles whose locations the player does not know. This is all the information a player has at any point in the game and it is enough for the AI engine to make a decision.

Choosing and Applying the Actions At each round, a player has a rather limited set of actions they can take. They can either draw the last tile discarded by the other player or draw a random tile from the deck and then they can discard any one of the five tiles they end up with afterwards. The discarding portion of the action is deterministic as the discarded tile is moved from the hand to the discarded pile. The drawing part is also deterministic if the player draws the tile discarded by the other player. However, if the player decides to draw from the deck, a problem arises since the player does not know what tile they will draw. For this reason, if the player chooses to draw from the deck, a random tile from the unknown location tiles is guessed to be drawn. This increases the possible game states by a large margin as every time the player decides to draw from the deck, there are many different possible immediate effects of the action.

Switching Between the Information Scopes of Players While the AI only needs to make one decision per call, it still needs to be able to "guess" the other player's responses to its potential actions to be able to assess their effectiveness. This brings the question of making a decision for the other player. It is possible to handle this easily by running the AI engine with the other player's game state, but this brings up the question of generating the other player's game state based on the current player's game state. This can be done probabilistically by randomly deciding the locations of the tiles unknown by the player. This is a meaningful way to infer the other player's action, as the player needs to consider all possible configurations and guess the other player's actions accordingly. However, once the other player's hand is inferred, the rest of the transformation is deterministic. Consequently, the AI engine probabilistically generates the other player's hand, and then constructs the other player's game state deterministically using the external JS module. These three functions accomplish this:

```
var inferredStateERP = function(state) {
    Enumerate( function() {
        var knownOthers =
state.inOthersHand.slice()
        var possibles =
state.unknownLocation.slice()
        var newHand = sample(handERP(knownOthers,
possibles, 4 - knownOthers.length))
        var otherPlayerState =
externOkey.buildOthersState(state, newHand);
        return otherPlayerState
    });
};
```

```
var handERP = function(curHand, possibleTiles,
numNeededTiles) {
    Enumerate(function() {
        if(numNeededTiles === 0) {
            return curHand
        } else {
            var nextTileIndex =
randomInteger(possibleTiles.length)
            var nextTile =
possibleTiles[nextTileIndex]
            var nextPossibleTiles =
possibleTiles.slice()
nextPossibleTiles.splice(nextTileIndex,1)
            var nextHand = curHand.slice()
            nextHand.push(nextTile)
            return sample(handERP(nextHand,
nextPossibleTiles, 4 - nextHand.length))
        }
    });
};
//Beyond here is part of the external library

 function buildOthersState(thisState,
 othersHand) {
     var otherPlayerState = {};
     otherPlayerState.hand = othersHand;
     otherPlayerState.myDrawPile =
 thisState.myDiscardPile;
     otherPlayerState.myDiscardPile =
 thisState.myDrawPile;
     otherPlayerState.inOthersHand =
 thisState.knownInMyHand;
     otherPlayerState.knownInMyHand =
 thisState.inOthersHand;
     otherPlayerState.unknownLocation =
 buildUnknowns(otherPlayerState);
     return otherPlayerState;
 }
```

As can be seen, the player starts with the tiles they know are in the other player's hand and randomly choose tiles until they have a full hand. Once this is done, the new state can be built. Since the discard piles are separate, this player's discard pile becomes the other player's draw pile and vice versa. Because of the nature of the game, both players know what the other players know about their hands, so this information can be deterministically transferred. Finally, any tile that is not included in the other player's hand, piles, or knowledge of the other hand are not located by the other player, so these are added to the unknown array.

The important point with changing the information scopes is doing it at the right time. The information scope change is only done when the other player's action is sampled, but once the other player chooses an action, that action is applied to the game state of the current player (not the other player) so the current player can keep guessing further into the game. However, since the other player's action depends on what we inferred for the other player's hand, the tile the other player discards is inferred based on the inferred hand of the other player.

Current State

The AI agent is currently fully implemented, however is not functional due to having too many recursive calls both horizontally (inferring the other player's action) and vertically (foreseeing the future). The biggest issue is the sheer size of the state space. Since the engine makes decisions with a limited set of information, it needs to "guess" or infer a lot of aspects of the game every time it considers an action. This, when combined with the high number of possible configurations of tiles causes the program to have just too much to enumerate efficiently. As a result, when the AI engine is run, the node process runs out of memory before the AI agent can conclude.

Evaluation

Feasibility as Real Time Game AI

While the use of probabilistic programming lets us create a potentially very powerful and realistic AI agent, the current state of probabilistic programming tools like WebPPL seems to be inadequate to handle the massive state spaces of real games. However, it must be noted that any such "fully probabilistic" approach seems unreasonably expensive in terms of computation. There are simply too many dependent and unknown variables that need to be inferred and since each random choice means a whole new set of possible outcomes, the computations become unreasonably complex very soon, even with a rather simplified version of a rather simple game such as Rummy. This shows that direct probabilistic programming is not a great tool to write a real time game AI, at least not at the idealistic scale the AI engine in this paper attempted where every single possible configuration was considered by the player. In this light, systems like CAPIR or POMCoP which are heavily optimized and either simplify the way they guess the other players' actions manage to add goal inference to real time game AI. Probabilistic programming seems like an ill suited tool for this purpose.

Usage in Studying Human Game Playing Behavior

However, an interesting possible application of modeling game AI in probabilistic programming shows itself when the working principles of the probabilistic AI are analyzed as a model of the human thought process that goes into the playing of the game.
Rummy is an interesting case in this sense, because while the game is fairly simple and there is a limited number of actions, considering other players' goals while planning has significant effect on the game. To demonstrate this, we will compare the hypothetical approaches of different AI models when deciding what to do next:

A traditional AI may keep track of which tiles are used throughout the game and thus will be unavailable in the future and which tiles may still be drawn. Using this, it may evaluate the chance of completing a certain grouping, and prefer groupings that will more likely be completed sooner. When having the option of drawing a specific tile discarded by the previous player and drawing a random tile, it can estimate the value of the known tile and the groupings it can be used for, and the expected value of drawing an unused tile from the pile of unused tiles, and choose. When discarding however, it will be pretty limited to choosing the tile that is the least likely to be grouped in the future.

However, a human player likely considers the goals and actions of other players, playing possibly closely to the following manner:

First, a human player keeps track of the probability of each other player having a tile, and needing a certain tile for a grouping. For example, if another player draws a yellow 2, and later draws a yellow 3, a human player will note that it is likely that they have the yellow 1, yellow 4 or even yellow 5, or will likely keep if they draw it, and hence will reduce its expectation of drawing or receiving those tiles in the future. Such a belief will be further strengthened if the same player discards a black 2 or red 3, as the belief that they are building a run of green tiles of ... 1,2,3,4... will be even stronger. As a result of this, the human player will have a better estimate of the likelihood of drawing a certain tile in the future, and will thus choose better groupings ahead of the time.

The other difference comes into practice when the AI decides to discard a tile. While a standard AI discards the tile with the lowest grouping value from the hand, the human will consider the next player's goals, and will try to optimize between the grouping values of tiles for itself, and the grouping values for the next player. In this situation, if the player discards a yellow 2, and the next player draws it, the human player will be less likely to discard a yellow 3, a yellow 1 or a 2 of a different color in the next round, as these will probably have a higher value for the next player whereas the traditional AI cannot plan for this ahead, and may easily feed the next player with good tiles.

All these aspects of a possible human player can be modeled by the goal inference of the probabilistic AI. In fact, the way every inference is an explicit probability distribution in WebPPL allows us to dissect the decision making process of the AI agent and look at the various inferences made and various beliefs it has about the game state. For example, it is possible to look at the distribution over the other player's responses to a given action for different actions and analyze how that affects human decision making in the game.

Furthermore, the so far perfectionist inferring AI can be made even more similar to a human player. For example, it is possible to add an element of memory to the AI. In such a model, as the game goes, the AI will be less and less certain of its observations and beliefs of the game state, erring just like a human player could do.

Conclusion

Probabilistic programming is a very strong tool to model human cognition as shown by various papers examining different aspects of human cognition. It has been shown that probabilistic programming can be used for modeling goal inference in rather simple situations, however, it has the expressive capacity to represent and run on fairly more complex situations such as decision making in a game while inferring the other players' goals. As shown by Stuhlmueller and Goodman, probabilistic programming can play very simple games like Tic-Tac-Toe in a reasonable amount of time, and catch nuances like the importance of starting from the middle position or covering the corners first without any direct training. The same can be theoretically applied to a slightly larger game like Rummy, where a probabilistic AI discovers various quirks of the game by simply randomly sampling actions and their outcomes. However, as seen by the author of this paper, this results in too much computational complexity to the point where it is no longer possible to use it as a real time AI agent.

However, a probabilistic AI is still a promising subject as a tool for studying how humans make decisions when playing games, how much recursive depth do they go to while inferring other players actions and how do they optimize this very complex process.

References

Dinh, T.-H. N., Hsu, D., Lee, W.-S., Leong, T.-Y., Kaelbling, L. P., Lozano-Perez, T., et al. *CAPIR: Collaborative Action Planning with Intention Recognition.* MIT, CSAIL. Cambridge, MA: CSAIL.

Macindoe, O. (2011). *Assistant Agents For Sequential Planning Problems.* MIT, CSAIL. Cambridge, MA: MIT CSAIL.

Macindoe, O., Kaelbling, L., & Lozano-Perez, T. (2012). *POMCoP: Belief Space Planning for Sidekicks in Cooperative Games.* MIT, CSAIL. Cambridge, MA: MIT CSAIL.

Stuhlmueller, A., & Goodman, N. D. (2013). *Reasoning about Reasoning by Nested Conditioning: Modeling Theory of Mind with Probabilistic Programs.* Stanford, CA: MIT, Stanford University.

Synnaeve, G., & Bessiere, P. (2011). *A Bayesian Model for Plan Recognition in RTS Games applied to StarCraft.* Palo Alto, CA: AIIDE.

N. D. Goodman and A. Stuhlmüller (electronic). *The Design and Implementation of Probabilistic Programming Languages*. Retrieved 2015-6-5 from http://dippl.org.