**Artificial Intelligence in Games**

**Session 4**

1. **Statistical Forward Planning in Complex Games**:
   1. Pruning in Tree Search:
      1. Pruning suboptimal moves is highly beneficial for tree search. This was the case in Minimax and its enhanced variant a-b search.
      2. The same is true for Monte Carlo Tree Search, if we are able to eliminate poor actions from the possible selections that the tree can make to focus more iterations on the most promising options.
      3. While a-b search used a general algorithmic approach to prune certain branches, we can also incorporate domain knowledge that permits determining when and how to perform pruning.
      4. Attending to the type of pruning, we can classify this enhancement into two types:
      5. Soft pruning: some moves are ignored for some time but can be selected later.
      6. Hard pruning: certain moves can never be selected.
      7. Although soft pruning prevents making final mistakes while discarding some actions that may reveal good after several iterations, it is also computationally less effective than hard pruning.
   2. First Play Urgency:
      1. The selection step in MCTS stops when finding a not fully expanded node: it then moves to the Expansion phase by picking an action at random. This can be a problem:
         1. The Tree Selection policy behaves well when a node is visited often (high N(s)).
         2. In nodes far from the root, with low N(s) values, this is not the case
         3. In high branching factor games, this is extreme. Values associated to moves far from the tree are not meaningful.
         4. First Play Urgency (FPU) is a modification for the Selection step of MCTS designed to tackle this issue. In FPU, all nodes have a default FPU value (theta) assigned to them. When, during the Selection step, we find a not fully expanded node, we do the following:  
              
            int FPU\_Selection() {   
             for(Action a in available\_actions(state)) {   
             if (children[a].isInTree())   
             urgency[a] = FPU\_Value 5   
             else   
             urgency[a] = UCB1-Tuned()   
             }   
             return arg\_max(urgency);   
             }
         5. FPU value (theta) is now another parameter of MCTS.
            1. If (theta) is very high, FPU makes the Selection behave like in normal MCTS
            2. If (theta) is very low, FPU will make MCTS to greedily choose actions that have been explored and ignore other possible actions to expand.
            3. The optimal value of (theta) strongly depends on the game that is being played.
   3. Progressive Unpruning:
      1. Two main ideas:
         1. Reduce the branching factor artificially for Selection at first.
         2. Progressively increase it with iterations.
      2. Expands a previously seen example: progressive bias:  
           
         a = maximise argument of { Q(*s*, *a*) + C times sqrt(natural logrithm N(*s*) / N(*s*, *a*)) + (h(s, a) / (1 + N(s, a)) }
      3. Steps:
         1. Start MCTS as normal, using Progressive Bias as normal.
         2. When N(S) more than T, prune all nodes except the kinit nodes with the highest heuristic h(s, a) value. MCTS can’t choose among the pruned nodes in the Selection step.
         3. Progressively unprune the pruned nodes when the number of simulations in S surpasses A \* B \_{k-k\_init}. At each time, the node with the highest h(s,a) becomes available.

|  |  |  |
| --- | --- | --- |
| k | A \* B \_{k-k\_init} (A=50, B=1.3) | Progressive unpruning |
| K = k\_{ init } | 50 \* 1.3^0 = 50 | - |
| K = k\_{ init } + 1 | 50 \* 1.3^1 = 65 | When N(s) more than 65, a new node becomes available |
| K = k\_{ init } + 2 | 50 \* 1.3^2 = 84.5 | When N(s) more than 84, a new node becomes available |
| K = k\_{ init } + 3 | 50 \* 1.3^3 = 109.85 | When N(s) more than 109, a new node becomes available |

* 1. Progressive Widening:
     1. A very similar approach is Progressive Widening.
     2. It is the same algorithm, with the difference that the threshold T for each node is variable, depending on the state of the board: when N(s) is greater than the number of points in the board (Go), pruning starts.

1. **Imperfect Information Games**:
   1. How to perform simulations in Statistical Forward Planning when we do not know the complete state of the game?
   2. In Phantom Go, from the point of view of each player, the opponent stones can be divided into two categories, depending on the knowledge about them:
      1. Stones found in the board by trying illegal moves.
      2. Stones with no information that they exist
         1. Although how many are there is known: number of moves made – stones found.
   3. This game can be treated as a stochastic game, where the opponent’s stones of category (2) are placed at random in valid positions.
   4. Monte Carlo Phantom Go turns Phantom Go into a game of Perfect Information:
      1. Agent memorizes all known positions (1) of the opponent’s pieces.
      2. On each MC Rollout:
         1. Place the N opponent’s pieces at random in empty intersections.
         2. Play a random rollout until the end of the game.
   5. Determinizations:
      1. Determinization is the process of sampling several instances of the game to turn it into a game with perfect information, analysing them with AI methods to make a final decision for the real game.
      2. An Information Set is a collection of several states that can be formed when one player has information that other does not. For instance, in a card game where each player hides their own cards, the information set contains all possible combinations of cards of the opponent.
         1. A player knows what information set the state is in, but not what particular state within it.
      3. There are certain problems with Determinizations:
         1. “Averaging over clairvoyance”
            1. Determinization will never choose to make an information gathering play.
            2. Also, it will never choose to make an information hiding play.
         2. Strategy Fusion: Given an information set, the recommended action of any agent (MC, MCTS, RHEA) will be the same independently of the actual state the game is in.
         3. Nonlocality: Some determinizations can be extremely unlikely due to the skill of other players to avoid reaching or playing away from such states.
   6. Expectimax:
      1. Minimax assumes there are two players and they are deterministic.
      2. Min player will choose the action that minimizes the reward for the max player.
      3. What if the opponent plays stochastically? The value of each chance node will be the expected utility.
   7. Strategy Fusion Problem:
      1. Assuming states x and y are equally likely, the expectimax value of a1 is 0, while the expectimax value of a2 is 0.5. Then it makes sense to pick a2.
      2. A determinizing player assumes independently that the state can be x or y. The minimax value of a1 is 1 and a2 is 0.5. Thus, it chooses a1.
   8. (Single Observer) Information Set MCTS:
      1. The same game, with Information Sets:
         1. Nodes in the tree are information sets from the root’s player point of view, not states.
         2. Edges correspond to actions.
         3. Actions leaving nodes are the actions available at the information set – i.e. from the states that conform the information set.
         4. After a sufficiently large number of iterations, ISMCTS can assign a1 the expected value of 0 and a2 the expected value of 0.5.
   9. SO-ISMCTS:
      1. How to address this problem?   
           
         At each iteration, perform a determinization.   
           
         Only states and actions that are consistent with that determinization are available in the information set.
         1. For instance, determinizing in the true real state being x:
            1. The probability of an action being available for selection on a given iteration is precisely the probability of sampling a determinization in which that action is available.
   10. Partially Observable Actions:
       1. Problem not addressed in SO-ISMCTS:
          1. What if some actions are Partially Observable: the opponent may perform a move but some detail of the action is hidden for an opponent.
   11. Multiple-Observer Information Set MCTS:
       1. But we have already seen that a random opponent model is a very poor choice.
       2. Multiple-Observer Information Set MCTS
          1. Solution: give each player a tree.
             1. Each tree maintains coherence on their own information sets.
             2. At each iteration, all trees are descended simultaneously.
             3. Each player makes decisions according to their own tree.

1. **Portfolio Search**:
   1. Real-Time Strategy (RTS) games are among the most complex games for Game AI research.
   2. These environments:
      1. They are multi-unit: players need to control large number of units on large maps.
      2. Players must manage resource gathering, economy, technology trees…
      3. There is imperfect information.
      4. There may be cooperation with allies or team members.
   3. There’s multiple research on RTS games and many different approaches have been tried.
      1. Statistical Forward Planning, Reinforcement Learning, Evolutionary approaches.
      2. Script based players – expert rule systems.
   4. Alpha Beta, MCTS… struggle considerably with huge branching factors and the multiple levels of management required.
      1. In many RTS work, there is an attempt for dividing action selection in two parts:
         1. Macro / Long-term planning.
         2. Micro / Short term planning.
   5. Portfolio Greedy Search:
      1. We define a script as an expert rule system that, given a state s and a unit u, provides a move ’a’ to be performed in the game by unit ‘u’:  
           
         a = c(s, u)
      2. A scripted player can be seen as a collection of scripts <c1, c2, …, c\_n>; where each script ci determines the action of the unit u\_i.
      3. The collection of scripts that an agent has at their disposal is known as its portfolio.
      4. The problem reduces to assigning scripts to actions depending on the current game state. Search methods don’t need to search in the space of actions, but in the space of scripts.
   6. Portfolio Greedy Search uses Local Search to assign scripts to units:  
        
      Move[] portfolio\_greedy\_search(State s){   
       Script enemy[enemy.nUnits] = Init(s)   
       Script mine[self.nUnits] = Init(s)   
       Improve(mine)   
       while(r < R)   
       Improve(enemy)   
       Improve(mine)  
      return generate\_moves(mine)   
      }  
        
      Script[] Init(State s){   
       best\_value = -infinity; best = phi   
       for (script c in P){   
       value = Rollout(c, s)   
       if(value > best\_value){   
       best\_value = value; best = c   
      return best   
      }  
        
      Portfolio Greedy Search uses Local Search to assign scripts to units:  
        
      Move[] Improve(State s, Script[] scripts){   
       for(i=1 to I)   
       for(u=1 to scripts.size())  
       best\_value = -infinity; best = theta   
       for (script c in P){   
       value = Rollout(c, s)   
       if(value > best\_value)   
       best\_value = value; best = c   
      scripts[u] = best 10   
      }
   7. Portfolio Search Variants:
      1. Hierarchical Portfolio Search, a two-level hierarchical system with:
      2. Bottom: Portfolio of algorithms that generate suggestions for different tactical areas
         1. Economy, defence, offense, etc.
      3. Top: High level search technique that iterates through these suggestions
         1. Minimax, MCTS, etc.
      4. Monte Carlo Planning over scripts in a portfolio
      5. MCTS Planning over scripts in a portfolio