Papers found

1. MCTS-Minimax Hybrids
2. Efficient Reinforcement Learning in Adversarial Games
3. A Survey of Monte Carlo Tree Search Methods
4. Real-Time Monte Carlo Tree Search in Ms Pac-Man
5. A comprehensive review on artificial intelligence-based machine learning techniques for designing interactive characters
6. Minimax Guided Reinforcement Learning for Turn-based Strategy Game
7. REINFORCEMENT LEARNING IN A TURN-BASED STRATEGY GAME

(<https://ec.europa.eu/jrc/communities/sites/jrccties/files/andersjonsson-2018.pdf>)

Evaluation Criteria

* Theoretical Analysis
  + Performance Bounds – How far from optimal is an AI algorithm?
  + Time complexity – How fast is it?
  + Memory Complexity – How much memory is consumed?

Empirical Performance measures

* Measure winning?
* Scoring points?

Practices in reinforcement learning

* Run X trials, report average of 3 best runs

<https://ai.stackexchange.com/questions/5570/game-ai-evaluation-function-and-making-progress-towards-winning>

Case 1: Minimax / Alpha-Beta / other similar "exhaustive" searches

When using Minimax / Alpha-Beta / other search algorithms based on those, the easiest solution to the problem you describe is to use iterative deepening. As soon as you prove a win for yourself at a certain depth level d using iterative deepening, you can simply stop the search, don't search if there are any other wins to be proven at depth d + 1, just play along the line you've just proven to be a winning line. This way, you will always go for the win in the lowest number of moves.

## Case 2: Monte-Carlo Tree Search / other searches with randomness

Monte-Carlo Tree Search is a well-known search algorithm that incorporates an element of randomness in its search. With these kinds of algorithms, the problem you describe tends not to be a real issue. Due to the randomness in the search, wins that can be achieved in a small number of moves tend to be evaluated better than longer-distance wins in practice. In long-distance wins, there is a greater chance that the randomness in the search process causes an incorrect move to be played somewhere along the long-distance win, which reduces the evaluation of such a line of play.

## Case 3: (Reinforcement) Learning approaches

These approaches tend to involve some element of randomness due to the need for **exploration** in learning, which leads to similar reasoning as described for MCTS above. Also, in Reinforcement Learning, we typically use a discount factor gamma < 1.0 (e.g. gamma = 0.99) which causes distant rewards to be viewed as less important than close rewards, **even if we don't do such discounting for the final evaluation of the performance of an algorithm**. See, for example, a lot of the work on Atari games (DeepMind's DQN, etc.). Algorithms are evaluated according to their undiscounted scores, but learning still uses a bit of discounting because, in practice, this is found to be beneficial for learning.

<https://ai.stackexchange.com/questions/13886/how-to-evaluate-an-rl-algorithm-when-used-in-a-game?rq=1>

When you want to compare Reinforcement Learning algorithms, you might want to compare the average rewards they generate and how fast and close they get to the optimal policy. Often Reinforcement Learning algorithms are compared by using the rewards (either direct, maximum or average in time/iteration).

27/03/2020 Friday

Papers found

Journal of Artificial Intelligence Research - <https://www.jair.org/index.php/jair/search/search>

1. Heuristic Search When Time Matters
2. Large-Scale Optimization for Evaluation Functions with Minimax Search
3. ON MONTE CARLO TREE SEARCH AND REINFORCEMENT LEARNING

Current Scenario

* Bots available
  + Human vs human
  + Human vs bot
  + Bot vs bot - 1st bot is random. 2nd is next step checking bot (it checks one step ahead after the 1st bots attempt). 3rd follows minimax algorithm and 4th follows expecti-max algorithm. Minimax and Expecti-max have alpha-beta pruning and depth-limited search incorporated to improve their performance.
* Third algorithm could be MCTS or RL (Q-learning).
* Keywords to search papers with
  + Adverserial search
  + Game theory
  + Zero-Sum game
  + Stochastic game
  + And 3 algorithms names

***02-04-2020 THURSDAY***

**Artificial Intelligence A Modern Approach**

**INTRODUCTION**

* The field of AI attempts to understand intelligent entities.
* AI strives to build intelligent entities as well as understand them.
* AI currently encompasses a huge variety of sub-fields, from general purpose areas such as perception and logical reasoning, to specific tasks such as playing chess, proving mathematical theorems, writing poetry, and diagnosing diseases.
* Definitions of AI according to eight recent textbooks vary along two dimensions – thought processes and reasoning and behavior, measuring success in terms of human performance against an ideal concept of intelligence.
* The history of AI has had cycles of success, misplaced optimism, and resulting cutbacks in enthusiasm and funding.
* The term performance measure is used for a criterion that determines how successful an agent is. Obviously, there is not one fixed measure suitable for all agents.

**INTELLIGENT AGENTS**

* Goal Based Agents – knowing about the current state of the environment is not always enough to decide what to do, the agent also needs some sort of goal information which describes situations that are desirable. Search and planning are subfields for AI devoted to finding action sequences that do achieve the agent’s goals.
* Utility Based Agents – Goals alone are not enough to generate high-quality behavior. Utility is a function that maps a state onto a real number, which describes the associated degree of happiness.
* Properties of environments (<https://0xadada.pub/2003/12/15/connect-four-playing-ai-agent/>)
  + Accessible vs Inaccessible - If an agent's sensory apparatus gives it access to the complete state of the environment, then we say that the environment is accessible to that agent. An environment is effectively accessible if the sensors detect all aspects that are relevant to the choice of action. An accessible environment is convenient because the agent need not maintain any internal state to keep track of the world.
    - The connect four env is full observable. The env consists of the board, which has constant dimensions, and the pieces which belong to either the player or the opponent. The agent has access to all of this information.
  + Deterministic vs Non-Deterministic - If the next state of the environment is completely determined by the current state and the actions selected by the agents, then we say the environment is deterministic. If the environment is inaccessible, however, then it may appear to be nondeterministic.
    - The connect four env could be considered deterministic, as there are no random elements at work here.
  + Episodic vs Non-Episodic - In an episodic environment, the agent's experience is divided into "episodes." Each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself.
    - The environment could be either depending on the algorithm the agent uses. If the algorithm calls for random placement of a piece, then the environment is episodic. However, if the algorithm is more sophisticated, calling for prediction of the opponent’s move then the environment is sequential.
  + Static vs Dynamic - If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static.
    - The environment is full static. Time is not a factor in making the decision as to where to place the pieces. Once it is the agent’s turn, the state cannot be changed until it makes its move.
  + Discrete vs Continuous - If there are a limited number of distinct, clearly defined percepts and actions we say that the environment is discrete.
    - Connect four is a fairly simple game with finite, albeit large number of different states. Therefore, the environment is decidedly discrete.

**SOLVING PROBLEMS BY SEARCHING**

* Measuring problem-solving performance - The effectiveness of a search can be measured in at least three ways.
  + Does it find a solution at all?
  + Second, is it a good solution (one with a low path cost)?
  + Third, what is the search cost associated with the time and memory required to find a solution? The total cost of the search is the sum of the path cost and the search cost.
* TOY PROBLEMS
  + CONNECT FOUR – need to write the states, operators, goal test and path cost variables.
* SEARCH STRATEGIES are evaluated in terms of four criterion
  + Completeness: is the strategy guaranteed to find a solution when there is one?
  + Time complexity: how long does it take to find a solution?
  + Space complexity: how much memory does it need to perform the search?
  + Optimality: does the strategy find the highest-quality solution when there are several different solutions?
* DEPTH-LIMITED SEARCH
  + Depth-limited search avoids the pitfalls of depth-first search by imposing a cutoff on the maximum depth of a path.
  + This cutoff can be implemented with a special depth-limited search algorithm, or by using the general search algorithm with operators that keep track of the depth.
  + depth-limited search is complete but not optimal – ASK MUKESH WHY HE CHOSE THIS

**GAME PLAYING**

* Game playing is one of the oldest areas of endeavor in AI.
* MINIMAX ALGORITHM – designed to determine the optimal strategy and consists of five steps:
  + Generate the whole game tree, all the way down to the terminal states.

Apply the utility function to each terminal state to get its value.

Use the utility of the terminal states to determine the utility of the nodes one level higher up in the search tree.

Continue backing up the values from the leaf nodes toward the root, one layer at a time.

Eventually, the backed-up values reach the top of the tree; at that point, MAX chooses the move that leads to the highest value

* + It maximizes the utility under the assumption that the opponent will play perfectly to minimize it.
  + An algorithm for calculating minimax decisions

**function** MINIMAX-VALUE (state, game) **returns** a utility value

**if** TERMINAL-TEST [game](state) **then**

**return** UTILITY [game](state)

**else if** MAX is to move in state **then**

**return** the highest MINIMAX-VALUE of SUCCESSORS (state)

**else**

**return** the lowest MINIMAX-VALUE of SUCCESSORS (state)

* ALPHA-BETA PRUNING – the process of eliminating a branch of the search tree from consideration without examining it is called pruning the search tree. It is ALPHA-BETA PRUNING when applied to a standard minimax tree and returns the same moves as minimax but prunes away branches that cannot possibly influence the final decision.
* There are games that mirror the unpredictable external events by including a random element.
* EXPECTIMAX
  + Each of the possible positions no longer have a definite minimax value (which in deterministic games was the utility of the leaf reached by best play).
  + Instead we calculate an average or expected value.
  + To calculate the best move, simply replace MINIMAX-VALUE by EXPECTIMINIMAX-VALUE