Lecture5

IF we want to solve game How long does it take?(Estimating Search Effort)--mainFactor: speed of program--size of state space

Fighting Exponential Growth--7\*7 board can be 30 moves deep!!--Time Limit--5 sec - 5 min per move --increase speed of program(**profile**), decrease b/d

Branching factor--growth of number of states per level--Decrease it--Use DAG instead of tree

**DAG**--Different paths to same node--Can lead to huge reduction in state space--DAG can be much more effective--ADV--Avoid redundant computations -- No copied subtrees or sub-DAGs-- compute once reuse often--Limitation--Need memory to store--some algorithm only for tree--state with different history(Ko rule?)--One of my students wrote a whole PhD thesis on such questions (Kishimoto 2005)

bd Model and DAG--bd = #nodes-at-depth d + 1 / #nodes-at-depth d--Depth n: b0 × b1 × ... × bn−1 nodes--Total nodes 1 + b0 + b0 × b1 + ... + b0 × b1 × ... × bd−1--Use search or sampling Difficult or impossible to compute them exactly--Use search or sampling Difficult or impossible to compute them exactly--Solving 1 × n Go: Exploring Positional Linear Go recent paper by Noah Weninger (UofA undergrad!) and Ryan Hayward (UofA prof)

Complexity--depends strongly on size of board

Sequential Decision-Making--find good actions: look ahead to future states--Organizing Sequences in Trees/DAG --we get exactly the tree/DAG representation--Tree sequen. Must share nodes!!

Lecture6

Full Board Repetition--superko Rule--Repetition Rules--The position would repeat, inﬁnite loop This is called a (basic) ko.

PositionalSuperKo--Ignore whose turn it is, only compare board---SituationalSuperko--Compare whose turn it is as well as board--DectingSPKo--use hashing to detect potential repetition

Fred Brooks, The Mythical Man-Month--The management question, therefore, is not whether to build a pilot system and throw it away. You will do that.

Don Knuth, Structured Programming with go to Statements--We should forget about small efﬁciencies, say about 97% of the time: premature optimization is the root of all evil.

OptimizationLimit--There is often an (approximate) 80-20 rule: 80% of the improvement can come from 20% of the code--With search, it can be even higher

Amdahl’s Law--p = percentage of program that is speeded up/ s = speedup for that part/ Runtime before optimization: 1/ Runtime after optimization: (1−p) +p/s /Speedup limit for the whole program: limit = 1 / [(1−p)+p/s]/ Simpliﬁed version: limit ≈ 1 / 1−p

Proﬁling--Proﬁler tells you details of the program execution--Profilers can be on the function level or instruction level --WaysOfProﬁlingInPython--cProfile is a built-in module--overhead of proﬁling is also measured(disadv)

For search and simulation, speed is very important

IdealOptimizationProcedure--(loop)Identify the most expensive functions--Try to improve them by optimization or better algorithms

StrategiesForOptimization--Best: avoid calling a function--Second best: speed up a function--detecting captures is most expensive//We do not need to compute the whole list--Stop if we ﬁnd one liberty

Remember Knuth:--premature optimization is the root of all evil

Lecture7

Search--Use search to lookup existing information or discover new information--Information may change during search (think internet) Data may be stored in a structured or unstructured way

StateSpaceSearch--enumerate列举 smaller state spaces, e.g. TicTacTo--state space is given but only implicitly

Blind search--no extra information that helps us search(DFS / BFS/ )--uses no heuristics--State space too large: cannot complete search--Blind Treasure Search--Expected number of visits to reach goal: ---> approx. n/2--(shape does not matter)

heuristicSearch--we have some information about where the goal is--use that information in a heuristic-- use heuristic to guide a search process--Following heuristic is often (much) better than blind search --Perfect heuristic--Always picks a correct action-- Goes directly to goal --Applications of Heuristic Search-- Find a good path for a character in a video game--Play chess, backgammon, Go better than any human…

modern heuristic search methods--Search--Heuristic Function--Simulation

Heuristic Function--h(s) estimates the “goodness” of s--Heuristic is usually not perfect, but “useful”

BlindSearchHopeless--in large space

Exploration--Focusing only on the heuristic is too risky--Need to ﬁnd a balance--Do not trust heuristic blindly--exploration to guard against biases in heuristic

random sampling--go down random path in tree-- Check at each step if treasure is found--may be unlucky and miss the treasure every single time

BFS/DFS--visit each node exactly once--complete search methods--Random sampling--Very strong bias--Some nodes may never be visited in any ﬁnite run

Lecture8

Winning Strategy--Consequence: the winning strategy is a tree (or DAG)--One move when it’s our turn & All moves when it’s the opponent’s turn--is much smaller than the whole state space--can still be very large--branches at each level

Proving a Win--To prove a win we need to find a winning strategy--use search to prove a win

Evaluation of Terminal Positions--Simplest case: binary (or boolean)[Go with non-integer komi]--Popular case: win-draw-loss[TicTacToe,GO with integer komi]

LeafNode--are terminal states of the game

Winning in an OR Node--Finding one winning move is enough(by playing that move)--Shortcut evaluation: can stop at the first child that is a win--Winning in an AND Node--We win only if we win after all opponent moves--Shortcut evaluation: can stop at the first child that is a loss

Different paths to reach the same node--Only prove a win (or loss) for a node once, then remember

transpositon table--hash Table

Boolean solver--only deals with two outcomes

Negamax--evaluate from the point of view of the current player--negate result of recursive call

Boolean Minimax--Runtime depends on:--d/ b/ move oding--Efficient pruning -- stops as soon as win is found--Boolean case is simpler special case of minimax search

Boolean Minimax - Efficiency--Best case: about bd/2--Worst case: about b d

Proof Tree--A winning strategy for a player--Dual concept: disproof tree - proves that we cannot win--A subset of a game tree--Gives us a winning move in each position we may encounter--Covers all possible opponent replies at each point when it’s their turn

Definition of Proof Tree--P contains the root of G--All leaf nodes of P are wins--If interior AND node is in P, then all its children are in P If interior OR node is on P, then at least one child is in P--Size of Proof Tree-- General pattern for best case: 1, 1, b, b, b 2 , b 2 , b 3 , b 3 , …--Best Case For Boolean Minimax Search--Search is most efficient if it looks only at the proof tree This means, at OR nodes we only look at a winning move--Good move ordering is crucial for efficient search

Efficient pruning of tree -- stop at first winning move

Lecture9

Minimax search:We maximize score

--Opponent minimizes our score

**MAX Node with Numeric Scores-**-With scores, usually yes We can stop early in **two scenarios**: --We know the highest possible score, and one child achieves it--We have a bound, and only want to know if we can reach at least that bound. Can stop as soon as one child achieves bound

Compute max over all children in OR node

Compute min over all children in AND node

Stop in terminal state, evaluate statically

Inefﬁciency of Plain Minimax/Negamax--No pruning

--We can do two boolean searches to verify if m is the minimax result

Reduce Minimax Search to the Boolean Case--win: score above a threshold m/ loss: score below a threshold m--First search if we can get at least m--Second search: can we get more than m:Assume: Search 1 returns a win //Search 2 returns a loss// Then m must be the minimax value--Search 1 returns win, Search 2 returns loss: Minimax value equal to m

Boolean Searches and Proof Trees--Proof tree of the ﬁrst search: Our winning strategy: achieve at least--Disproof tree of the second search: Opponent’s winning strategy: prevent us from getting more than m

In TicTacToe--Together, both searches prove: TicTacToe is a draw…

Alpha-beta Search--Use if we have more than two outcomes, e.g. numeric score--Idea: keep lower and upper bounds (α,β) on the true minimax value--prune a position if its value v falls outside the (α,β) window--v < α we will avoid this position, we have a better alternative v > β opponent will avoid this position, has a better alternative--Negate scores when changing from player to opponent on each level-

Other moves of value≤5 do not help u--They can be pruned--From Exact Search to Heuristic Search--All our algorithms so far search each move sequence until the end of game

Heuristic play--Stop search earlier (e.g. at depth of d moves) Evaluate leaf node by a heuristic-- Depth-limited searches can be good for move ordering

Lecture10

QUIZ:

(2)When the article was written, AlphaGo had won every single game against human Go champions. --F--The credit assignment problem makes machine learning easier.--F--AlphaGo learns by improving the information in its lookup tables, which store individual game positions and their values.--F--In games like chess and Go, A learning agent can store the best action for every possible situation in a lookup table.--F--In professor Schmidhuber's opinion cited in the article, humans were still much better than computers in playing Go.--F--(3)The Interpolis legal aid department has taken up Chris Snijders' challenge.--F--math model predict future better than human--face to face interviews degrade decision quality--human can see beyond model--According to the article, one disadvantage of computer models is their lack of emotion.--F--The computer-assisted decision-making for Long Term Capital Management ran into some initial problems, but it was ultimately very successful.--F--(5)To use depth-limited alphabeta search, we need to have an evaluation function.--T--Alphabeta search can stop as soon as it finds a move that is better than the first move tried.--F--Negamax Alphabeta can prune the search as soon as a child has a value of beta or more--T--In Negamax Alphabeta, the alpha value is the minimum that the current player is guaranteed to achieve so far.--T--If we guess a minimax value m, then we can verify that it is correct by using one boolean minimax search.--F

Reinforcement Renaissance 5. credit assignment problem—remains a major challenge in reinforcement learning. Maybe We Should Leave...1.Models can be faster, more accurate, less inconsistence, lack of emotion, no degradation of decision quality over time, and more experience 3. face-to-face interviews degrade decision quality

How algorithms rule our working lives 3. Its goal was to replace subjective judgments with objective measurements 4. Few of the algorithms and scoring systems have been vetted with scientific rigour. (inaccuracy) 5. Orchestras started in the 1970s to hold auditions with the musician hidden behind a sheet. Result in increasing the percentage of women playing in major orchestras. 20 years after Deep Blue 1.Deep Blue computer was the first machine to beat a reigning world chess champion in a six-game match. 2. Feng-hsiung Hsu, Thomas Anantharaman and Murray Campbell developed Deep Thought, which became the first program to defeat a grandma

Reinforcement Renaissance

1. DeepMind’s methods: a blend of deep neural networks and reinforcement learning called “deep reinforcement learning.”

2. The clearest contrast is with supervised learning, the kind used to train image recognition software, in which the supervision comes in the form of labeled examples (and requires people to label them)

3. understanding similarity—being able to extract general features from many specific examples—is the great strength of deep neural networks.

4. When the article was written, AlphaGo had won every single game against human Go champions.

5. credit assignment problem—remains a major challenge in reinforcement learning.

Maybe We Should Leave That Up to the Computer

1.Models can be faster, more accurate, less inconsistence, lack of emotion, no degradation of decision quality over time, and more experience

2. Research has shown that mathematical models predict future performance in graduate school better than humans.

3. face-to-face interviews degrade decision quality

How algorithms rule our working lives

1.Personality test grades people for extraversion, agreeableness, conscientiousness, neuroticism, and openness to ideas.

2. computer program could save time but also was marketed as fair and objective, save money.

3. Its goal was to replace subjective judgments with objective measurements

4. Few of the algorithms and scoring systems have been vetted with scientific rigour. (inaccuracy)

5. Orchestras started in the 1970s to hold auditions with the musician hidden behind a sheet. Result in increasing the percentage of women playing in major orchestras.

6. Most problematic correlation had to do with geography

7. Gild’s category of predictive model has more to do with rewarding people than punishing them.

20 years after Deep Blue

1.Deep Blue computer was the first machine to beat a reigning world chess champion in a six-game match.

2. Feng-hsiung Hsu, Thomas Anantharaman and Murray Campbell developed Deep Thought, which became the first program to defeat a grand master, a professional level player in a tournament.

3. Back in days, the AI is limited by the hardware.

4. Grand masters were sparring partners for Deep Blue, help with opening library, which every chess program uses in order to save time and make sure it gets into reasonable positions.

5. It decides where to move by running on the supercomputer to carry out part of a chess computation and then hand off the more complex parts of a move to the accelerator, which would then calculate [possible moves and outcomes].

6. The deep blue upgraded by speed up the system, increased the chess knowledge.

Checker Is Solved

1.Claude Shannon’s seminal paper on the structure of a chess-playing program in 1950, artificial intelligence researchers have developed programs capable of challenging and defeating the strongest human players in the world.

2. However strong these programs are, they are not perfect.

3. Perfection implies solving a game—determining the final result (game-theoretic value) when neither player makes a mistake.

4. three levels of solving a game: 1) ultraweakly solved, the perfect-play result, but not a strategy for achieving that value. 2) both the result and a strategy for achieving it from the start of the game are known. 3) have the result computed for all possible positions that can arise in the game.

5. The effort to solve checkers began in 1989.

6. we have a computational proof that checkers is a draw. the program can achieve at least a draw against any opponent, playing either the black or white pieces.

7. The easiest path is to emulate the human way to solve it. But Human-like strategies are not necessarily the best computational strategies.

8. four-color theorem: given an arbitrary map with countries, you need at most four different colors to guarantee that no two adjoining countries have the same color. In 1976, a computational proof was demonstrated.

9. Developing the checker solve program started in 1950. In 1963, Arthur Samuel’s program beat a human, but didn’t really solve the game.

10. The Chinook project began in 1989 with the goal of building a program capable of challenging the world checkers champion. It earned the right to play in 1990. By 1996 Chinook was much stronger than all human players, and with faster processors this gap has only grown

11. Tinsley had only three losses in the period from 1950 to 1991.

12. How difficult is it to solve a game? (i) decision complexity, the difficulty of making correct move decisions, and (ii) space complexity, the size of the search space.

13. The proof procedure has three algorithm/ data components: (i) Endgame databases (backward search). (ii) Proof-tree manager (forward search). (iii) Proof solver (forward search).

14. Backward search. Positions at the end of the game can be searched and their win/loss/draw value determined. Retrograde analysis has been successfully used for many games.

15. The databases contain the win/loss/draw result for a position, not the number of moves to a win/loss.

16. The first databases, constructed in 1989, were for less than or equal to four pieces. In 1994, Chinook used a subset of the eight-piece database for the Tinsley match. In 2005, the 10-piece database computation finished.

17. The forward search consists of two parts: the proof- tree manager, which builds the proof by identifying positions that need to be assessed, and the proof solvers, which search individual positions.

18. The alpha-beta search algorithm is the main- stay of game-playing programs. In the best case, the alpha-beta algorithm only needs to examine roughly b^(d/2) positions.

19. If Chinook does not find a proven result, then a second program is invoked (100 s). It uses the Df-pn algorithm, a space-efficient variant of Proof Number search. The search returns a proven, partially proven, or unknown result.

20. graph-history inter- action (GHI) problem. It is possible to reach the same position through two different sequences of moves.

21. there might be many potential sources of errors, including algorithm bugs and data transmission errors.

22. checkers is a draw is not a surprise; grandmaster players have conjectured this for decades.

23. Numerous nontrivial games have been solved, including Connect Four, Qubic, Go-Moku, Nine Men’s Morris, and Awari.