CSCE 580: Introduction to AI

Lecture 10: Game Search

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

19TH SEP 2024

Carolinian Creed: "I will practice personal and academic integrity."

Credits: **Copyrights of all material reused acknowledged**

Organization of Lecture 10

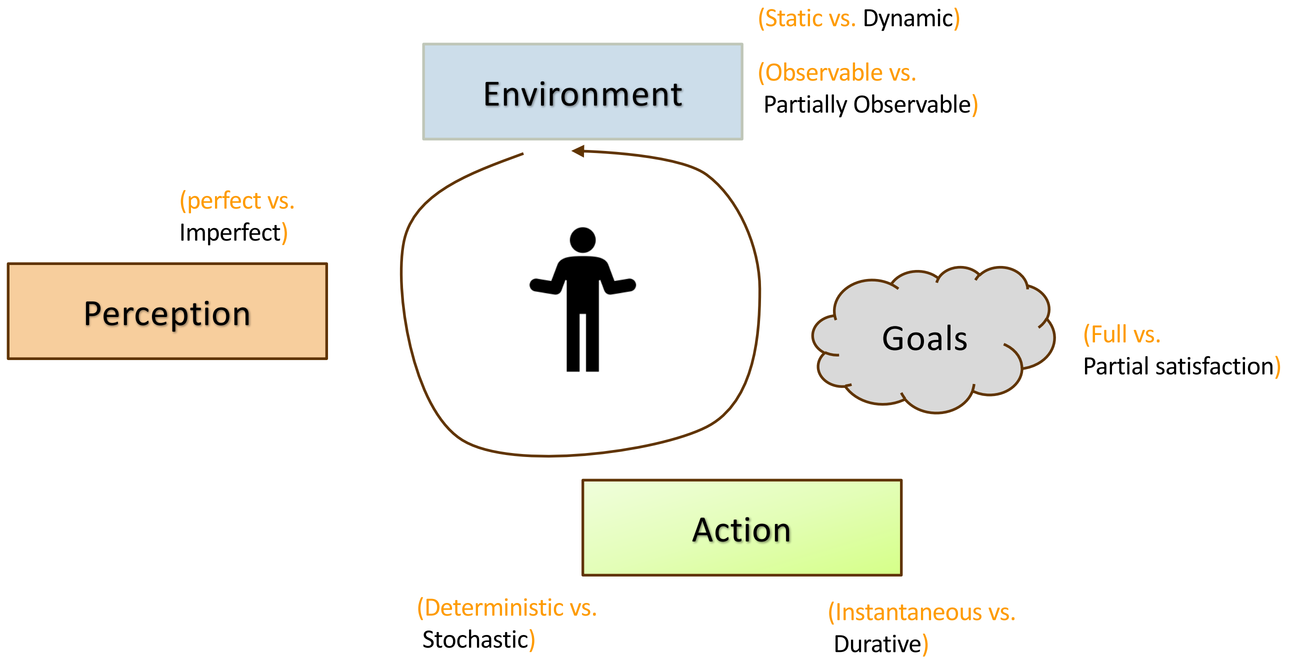
- Introduction Segment
  - Recap of Lecture 9
- Main Segment
  - Games and Search
  - Minimax
  - Alphabeta
  - Monte Carlo Tree Search
- Concluding Segment
  - Course Project Discussion
  - About Next Lecture – Lecture 11
  - Ask me anything

Introduction Section

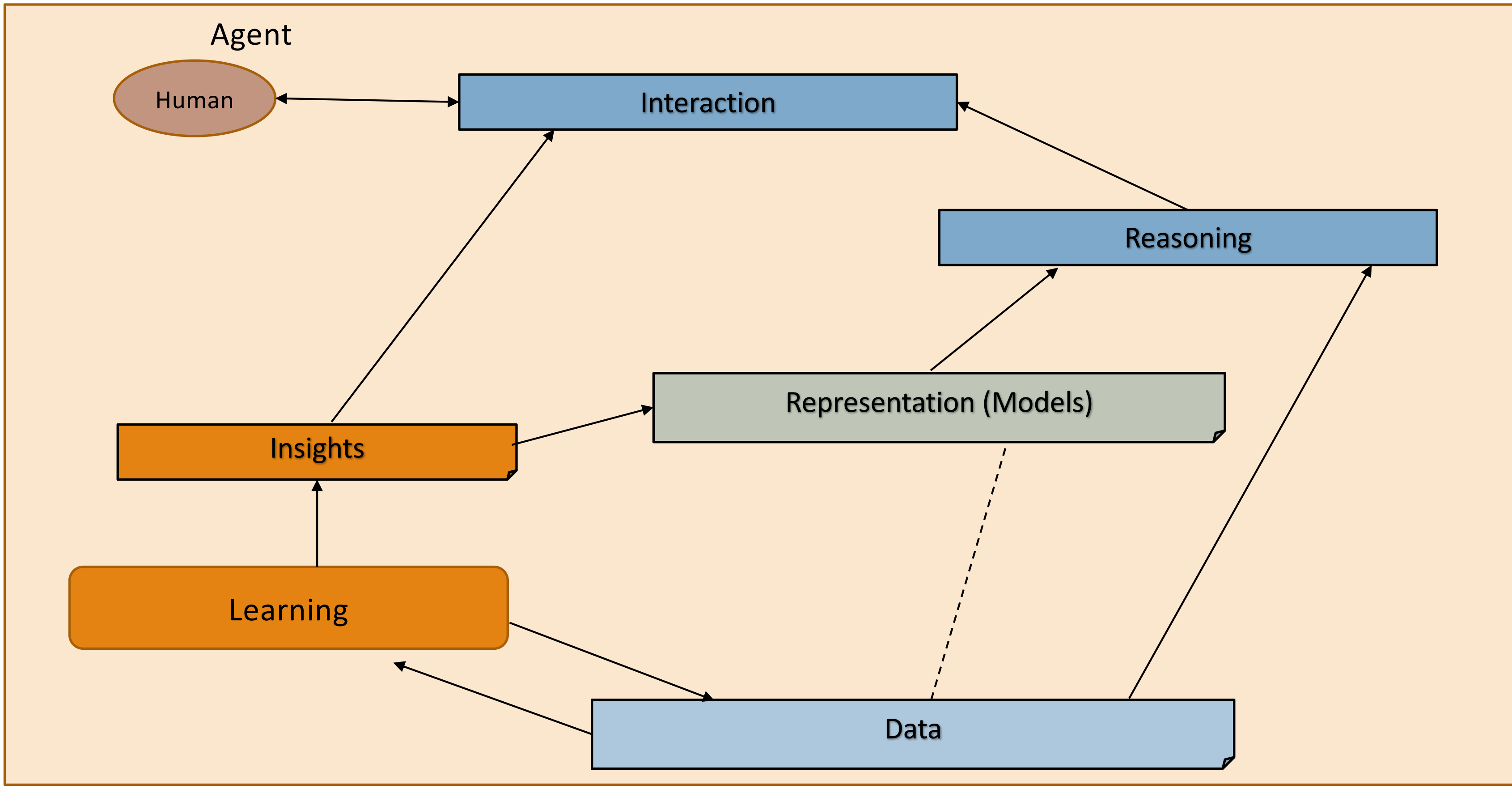
Recap of Lecture 10

- Hill climbing
- Simulated Annealing
- Genetic programming
- Search in complex environments

Intelligent Agent Model



Relationship Between Main AI Topics



Where We Are in the Course

CSCE 580/ 581 – In This Course

- Week 1: Introduction, Aim: Chatbot / Intelligence Agent
- Weeks 2-3: Data: Formats, Representation and the Trust Problem
- Week 4-5: Search, Heuristics - Decision Making
- Week 6: Constraints, Optimization – Decision Making
- Week 7: Classical Machine Learning – Decision Making, Explanation
- Week 8: Machine Learning - Classification
- Week 9: Machine Learning - Classification – Trust Issues and Mitigation Methods
- Topic 10: Learning neural network, deep learning, Adversarial attacks
- Week 11: Large Language Models – Representation, Issues
- Topic 12: Markov Decision Processes, Hidden Markov models - Decision making
- Topic 13: Planning, Reinforcement Learning – Sequential decision making
- Week 14: AI for Real World: Tools, Emerging Standards and Laws; Safe AI/ Chatbots

Main Section

Offline and Online Search

- All search studied until now were offline search
  - Solution found and then agent executed
- Online search
  - Interleave solution finding and execution
  - Cannot guarantee solution, <u>unless</u> actions have inverse-actions to undo state change

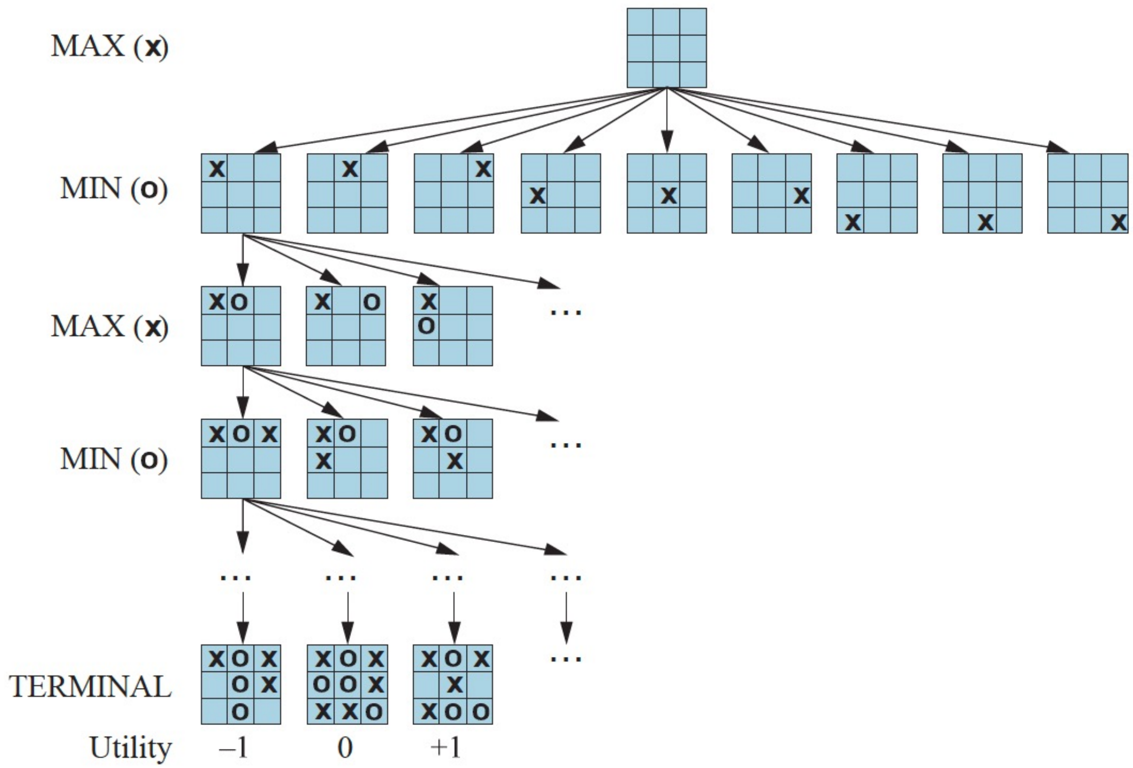
Games and Search

- Common setting
  - Two players
    - One human, one automated [**Common case**]
    - Both automated
  - Perfect information (fully observable)
  - Zero-sum – if one wins, another looses
- Captures many types of games
  - Tic-tac-toe, chess, Go, backgammon, …

Games and Search

- Setup
  - **So**: the initial state
  - **TO-MOVE(s)**: the player to play in state s
  - **ACTIONS(s)**: the set of legal actions at state s
  - **RESULT (s, a):** transition model
  - **IS-TERMINAL (s)**: terminal test to determine if game is over
  - **UTILITY (s, p):** utility or objective function. Numeric value capturing value of state s to player p

Tic-Tac-Toe – Game Tree



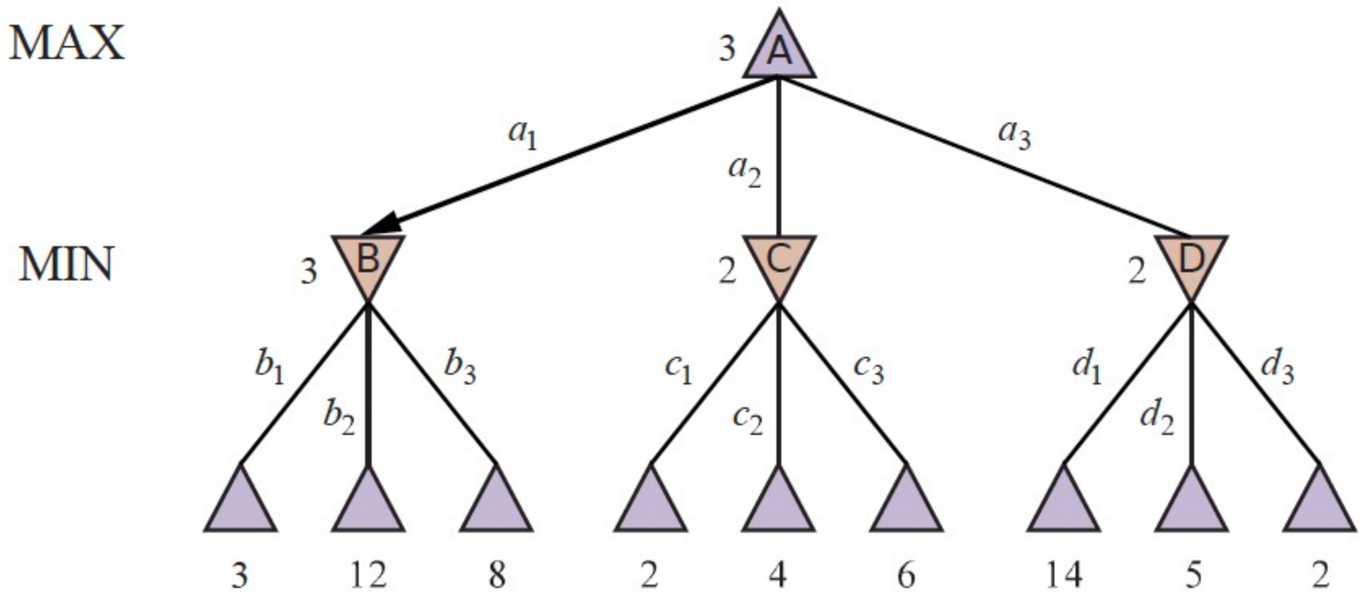
Alternate player turns

Adapted from:
Russell & Norvig, AI: A Modern Approach

MiniMax Utility

MINMAX(s) =

- UTILITY(s, MAX) if IS-TERMINAL(s)
- Max a in ACTIONS(S) MINIMAX(RESULT(s, a)) if TO-MOVE(s) = MAX
- Min a in ACTIONS(S) MINIMAX(RESULT(s, a)) if TO-MOVE(s) = MIN



Adapted from:
Russell & Norvig, AI: A Modern Approach

Minimax Search Algorithm

function MINIMAX-SEARCH(*game*, *state*) **returns** *an action*
  player ← *game*.TO-MOVE(*state*)
  value, *move* ← MAX-VALUE(*game*, *state*)
  return *move*

function MAX-VALUE(*game*, *state*) **returns** a (*utility*, *move*) pair
  if *game*.IS-TERMINAL(*state*) **then return** *game*.UTILITY(*state*, *player*), *null*
  $v \leftarrow -\infty$
  for each *a* **in** *game*.ACTIONS(*state*) **do**
    v2, *a2* ← MIN-VALUE(*game*, *game*.RESULT(*state*, *a*))
    if $v2 > v$ **then**
      v, *move* ← *v2*, *a*
  return *v*, *move*

function MIN-VALUE(*game*, *state*) **returns** a (*utility*, *move*) pair
  if *game*.IS-TERMINAL(*state*) **then return** *game*.UTILITY(*state*, *player*), *null*
  $v \leftarrow +\infty$
  for each *a* **in** *game*.ACTIONS(*state*) **do**
    v2, *a2* ← MAX-VALUE(*game*, *game*.RESULT(*state*, *a*))
    if $v2 < v$ **then**
      v, *move* ← *v2*, *a*
  return *v*, *move*

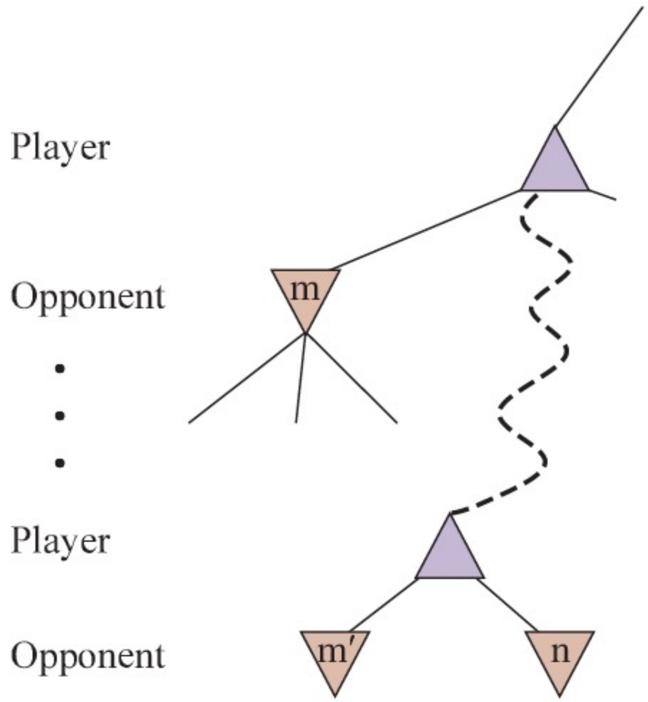
Starts from a Max node
Recursively does min on children,
 who in-turn does max on children
 to get value and corresp. action

Adapted from:
Russell & Norvig, AI: A Modern Approach

Multi-player (>2) Games

- Possible to extend MINMAX, algorithm is more complex
  - Extend values with vectors, corresponding to per player
  - Levels increase (per turn)
- In real games, players can often form alliances
  - Hard to model

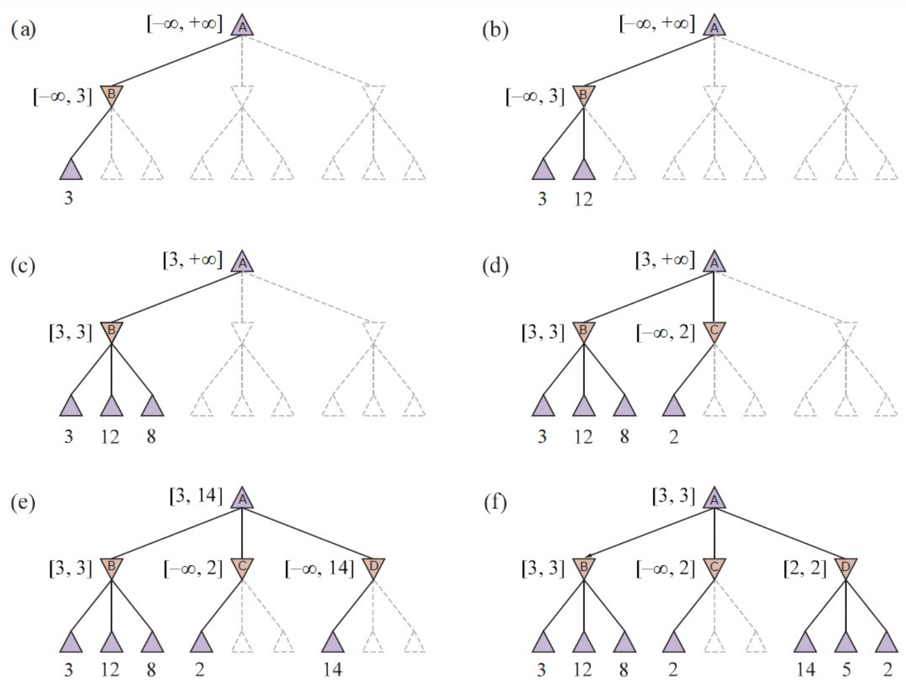
Alpha-Beta Pruning



If m or m′ is better than n for Player, search will never get to n

Adapted from:
Russell & Norvig, AI: A Modern Approach

Alpha-Beta Pruning



The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of at most 2. But we know that B is worth 3, so MAX would never choose C.

Adapted from:
Russell & Norvig, AI: A Modern Approach

Alpha-Beta Pruning

```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, −∞, +∞)
  return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

Alpha: the best (highest, at least) value
Beta: the best (lowest, at most) value

Adapted from:
Russell & Norvig, AI: A Modern Approach

Impact of Alpha Beta

- Cuts down on size of game tree searched
  - Best case: O (b ^ (m/2)), where b is branching factor and m is depth
  - MINIMAX(O (b ^ m)
- Reduction depends on order of nodes traversed

Game Setting

- Gaming strategies
  - Random
  - Minimax
  - alphabeta
- two automated players
  - First: random
  - Second: alphabeta
  - Code Example: https://github.com/aimacode/aima-python/blob/master/games4e.ipynb
- one human, one automated
  - First: human
  - Second: minimax

Heuristic Alpha Beta Tree Search

H-MINIMAX(s) =

• Eval(s, MAX)	if IS-CUTOFF(s, d)
• Max a in ACTIONS(S) H-MINIMAX(RESULT(s, a), d+1)	if TO-MOVE(s) = MAX
• Min a in ACTIONS(S) H-MINIMAX(RESULT(s, a), d+1)	if TO-MOVE(s) = MIN

* Cut-off search at depth d
* Estimate utility of a state to player just like a heuristic function

UTILITY(loss, p) <= EVAL (s, p) <= UTILITY(win, p)

Useful when

- Depth is very high, example Chess
- Good (informative) eval functions exists

Adapted from:
Russell & Norvig, AI: A Modern Approach

Coding Example

- 2-party games code notebook
  - https://github.com/aimacode/aima-python/blob/master/games4e.ipynb
  - Has
    - Minimax
    - Alphabeta
    - Heuristic alpha beta

Searching in Larger Games

Game characteristics

- large branching factor – Go (361)
- difficult to get a meaningful evaluation function (heuristics)

Key Idea

- Value of a state is estimated as the average utility over a number of simulations of complete games from that state
- Get information of good plays by self-play and learning using neural networks

Monte Carlo Tree Search (MTCS)

function MONTE-CARLO-TREE-SEARCH(*state*) **returns** *an action*
  tree ← NODE(*state*)
  while IS-TIME-REMAINING() **do**
    leaf ← SELECT(*tree*)
    child ← EXPAND(*leaf*)
    result ← SIMULATE(*child*)
    BACK-PROPAGATE(*result*, *child*)
  return the move in ACTIONS(*state*) whose node has highest number of playouts

Adapted from:
Russell & Norvig, AI: A Modern Approach

Monte Carlo Tree Search (*MCTS*)

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree ← NODE(state)
  while IS-TIME-REMAINING() do
    leaf ← SELECT(tree)
    child ← EXPAND(leaf)
    result ← SIMULATE(child)
    BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

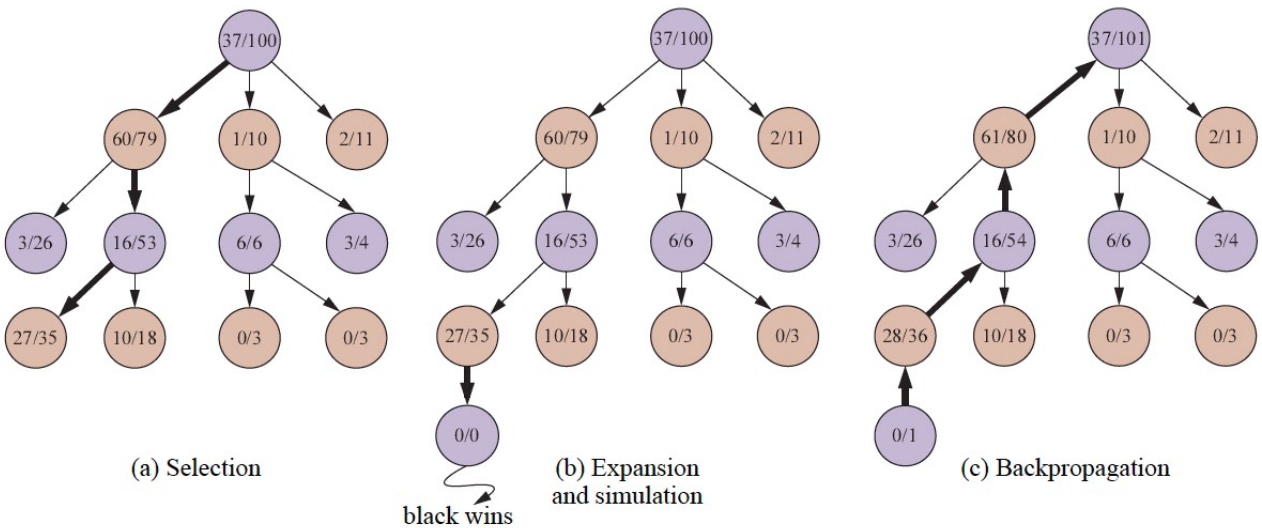


Figure 5.10 One iteration of the process of choosing a move with Monte Carlo tree search (MCTS) using the upper confidence bounds applied to trees (UCT) selection metric, shown after 100 iterations have already been done. In (a) we select moves, all the way down the tree, ending at the leaf node marked 27/35 (for 27 wins for black out of 35 playouts). In (b) we expand the selected node and do a simulation (playout), which ends in a win for black. In (c), the results of the simulation are back-propagated up the tree.

Adapted from:
Russell & Norvig, AI: A Modern Approach

Coding Example

- MCTS code notebook
  - https://pypi.org/project/mcts-simple/
  - https://colab.research.google.com/drive/1uYCDn_IymEhexepKfBXcMqiHquyhZpZ5?usp=sharing

References:
- https://deepmind.google/technologies/alphago/
- https://deepmind.google/technologies/alphazero-and-muzero/
- https://en.wikipedia.org/wiki/AlphaGo
- https://deepmind.google/discover/blog/alphastar-grandmaster-level-in-starcraft-ii-using-multi-agent-reinforcement-learning/

Alpha Solvers

- AlphaGo
  - Search with neural network based estimation
  - Two NNs: first - “policy network” — selects the next move to play, second — the “value network” — predicts the winner of the game.
  - Developed for Go; learned from human players and self-play
- AlphaZero
  - Given rules of the game (in mathematical form)
  - Uses reinforcement learning
  - Plays multiple games (Go, Chess, Shogi), learned from self-play
  - Developed newer methods for faster sorting, hashing, and matrix multiplication algorithms
- MuZero
  - Models three aspects of its environment - how good is the current position? Which action is the best to take? And how good was the last action?
  - Plays multiple games (Go, Chess, Shogi, Atari family), learned from self-play
- AlphaDev
  - self-play via reinforcement learning, multi-agent learning, and imitation learning
  - Example: self-play with a group of players (league)

Lecture 10: Summary

- We talked about
  - Games and Search
  - Minimax
  - Alphabeta
  - Monte Carlo Tree Search

Concluding Section

Course Project

Discussion: Projects

- New: two projects
  - Project 1: model assignment
  - Project 2: single problem/ llm based solving / fine-tuning/ presenting result

Project Discussion

1. Go to Google spreadsheet against your name
2. Enter model assignment name and link from (http://modelai.gettysburg.edu/)

1. Create a private Github repository called "CSCE58x-Fall2024-<studentname>-Repo". Share with Instructor (biplav-s) and TA (vishalpallagani)
2. Create Google folder called "CSCE58x-Fall2024-<studentname>-SharedInfo". Share with Instructor (prof.biplav@gmail.com) and TA (vishal.pallagani@gmail.com)
3. Create a Google doc in your Google repo called "Project Plan" and have the following by next class (Sep 5, 2024)

Timeline

1. Title:
2. Key idea: (2-3 lines)
3. Data need:
4. Methods:
5. Evaluation:
6. Milestones
    1. // Create your own
7. Oct 3, 2024

Reference: Project 1 Rubric (30% of Course)

Assume total for Project-1 as 100

- **Project results** – 60%
  - Working system ? – 30%
  - Evaluation with results superior to baseline? – 20%
  - Went through project tasks completely ? – 10%
- **Project efforts** – 40%
  - Project report – 20%
  - Project presentation (updates, final) – 20%
- **Bonus**
  - Challenge level of problem – 10%
  - Instructor discretion – 10%
- **Penalty**
  - Lack of timeliness as per your milestones policy (right) - up to 30%

Milestones and **Penalties**

- Project plan due by Sep 5, 2024 **[-10%]**
- Project deliverables due by Oct 3, 2024 **[-10%]**
- Project presentation on Oct 8, 2024 **[-10%]**

About Next Lecture – Lecture 11

Lecture 11: Constraints

- Constraints Satisfaction Problems
- Optimization

7	Sep 10 (Tu)	Search - Uninformed
8	Sep 12 (Th)	Search - Informed; Heuristics
9	Sep 17 (Tu)	Local search
10	Sep 19 (Th)	Adversarial games and search
11	Sep 24 (Tu)	Constraints & optimization
12	Sep 26 (Th)	Machine Learning - Basics
13	Oct 1 (Tu)	Machine Learning – Classification – Decision Trees, Random Forest
14	Oct 3 (Th)	Machine Learning – Classification – NBC, Gradient Boosting, ML-Text