# Computing Science (CMPUT) 455

## Search, Knowledge, and Simulations

Ting-Han Wei

Department of Computing Science
University of Alberta
`tinghan@ualberta.ca`

Fall 2020

# Lecture 4: Formalizing Decision-Making

# Lecture Topics



- Intro to computer decision-making
- Search, state spaces, state space of a game
- Terminology for state space search
- Types and models of state spaces
- Game trees, $b^d$ model

# Decision-making Exercise - Use Computers?



- Could a computer do the same decision-making processes as you?
- What would a computer need to make similar decisions?
  - Knowledge about your problem?
  - Knowledge about the world, “common sense”?
  - Logical reasoning?
  - Optimization?
  - Input? Sensors? Memory? Processing power?

# Do you Want Computers to Make Decisions?



- Would you want a computer to make decisions **for you**?
- When would you trust it?
- How about computers supporting human decision-making?
- Who is in control?
  - Machine?
  - Programmer?
  - Employer?
  - Government?
  - Nobody?

# Reasoning - Now vs Future



- Some decision-making considers only the present
- Reflexes
- Intuitive Decisions
- Tasks where looking ahead is not needed
- Example: image recognition
  - All relevant knowledge is available now
  - Neural net has learned knowledge from many (past) examples

# Reasoning - Now vs Future

- Much reasoning involves thinking about the future, in order to make decisions now
- Example: should I take an umbrella to work?
  - It does not rain now
  - The forecast predicts rain later today
- Chess example: should I capture this queen?
  - It looks good now
  - It gets me checkmated 8 moves later

# Decision Making as Search Problem



- Decision making in practice has many complications
- Often, we do not know:
  - The possible actions
  - The goals of others
  - How to evaluate different outcomes
  - How to compute with limited resources
  - ...

*It is difficult to make predictions,*
*especially about the future.*

*https://quoteinvestigator.com/2013/10/20/no-predict/*

- Where do we start? Simplify. A lot.

# A Simple Setting for Decision-Making



Long list of assumptions:

- The *state* of the world is completely known at each time
- There are *terminal states* where we can evaluate the result precisely, e.g.
  - A number: value, score, reward, utility,...
  - One of a set of possible outcomes, e.g. {win, loss, draw}
- The possible *actions* (or moves) in each state are known
- An action changes the state to a new state in a known, deterministic way
- No chance element - no dice rolls, cards drawn, other random events

Do games fit this simple setting?

# Do Games fit Our Simple Setting?



- Single-player (puzzles): often, yes
  - Rubik's cube, solitaire, rushhour, sudoku, crossword puzzles, 15-puzzle
- **Two-player games**: our main topic - see separate slide
- Multi-player games: mixed
  - Yes: Chinese checkers, ...
  - No: most card games,
    all dice games,
    most family board games, ...

# Do Games fit Our Simple Setting?



- Online and computer games: mostly, no
  - Simultaneous moves, often in real time
  - Map only partially known
  - Complex physics simulations
  - Many other complications...

# “Classical” Two Player Games



- Our simple setting, plus:
- Two players, often called Black and White
- Players move alternately: I play, you play, I play,...
- A move instantly changes the state (no duration, no slow transitions)
- Simplest, most frequent case is *zero-sum*: my win is opponent’s loss
  - Opposite: cooperative games
- Examples: chess, checkers, Go, Tic-Tac-Toe, ...

# Why Study Decision-Making using “Classical” Games?



- Simple, controlled environment
- Still hard to solve or play well
- Interesting for many people
- Games and results are easy to understand
- Playing games well requires good decision-making skills
- We can study the *core problems* of decision-making without being distracted by too many complications

# Formalizing State Space Search

# Terminology



The next few slides introduce the terms
we use to talk about state space search in general,
and specifically, games.

- game state, state
- state space, game graph, game tree
- board state, position
- move, action
- move sequence, history, game record
- score, value, evaluation, result

# State, Game State



- Complete description of the current situation
- In games: board position or cards etc, toPlay (whose turn it is)
- State, plus rules of game, allow us to determine actions (moves)
- Often includes (parts of) history:
  - Sequence of moves from start of game to current position

# Example in `Go1`



- How is game state represented in `Go1`?
- In `simple_board.py`
- Class `SimpleGoBoard`
- Contains 1-d array `board`
    - Each array entry contains the color of one point
- Contains field `ko_recapture` to implement simple ko rule
- `current_player` (`BLACK` or `WHITE`)
- Other fields: `name`, `version`,...

# Action, Move



- Leads from one state to another
- Move in games may include `toPlay`:
  color of the player who plays the move
- *Alternating* play:
  - `current_player` changes after each move
  - We do not need to specify color with the move (e.g. can just store it in the state)
- Move sequence, history, game record:
  all moves played in a game
- In `Go1`: move represented by index of point in array, and color
  - Used e.g. in `play_move(point, color)`

# State vs History



- When is history needed?
  - Depends on rules, structure of search space
- Example: Ko (repetition), legal moves in Go depend on previous move history
- Example: TicTacToe does *not* need history
- Compare with *Markov Decision Problems* (MDP) in single agent search, *Markov property*:
  - History is irrelevant in MDP
  - Current state contains all relevant information

# History-only State?



- How about having *only* history, no other information in state?
- Yes, that works in principle
  - Rules plus complete history determine the state exactly
  - Examples:
    - Game records with list of moves
    - Sequence of GTP play commands
- It may be inefficient if we always have to replay all actions from the beginning in order to find the current state
- Some forms of machine learning work with this representation

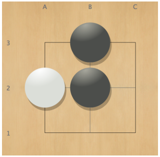
# State Space



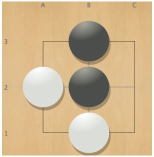
- A *state space* is:
  - A graph with all the possible states of a problem
  - Edges in graph show how states are connected by actions
- State space represented as directed graph $G = (V, E)$:
- Nodes in $V$: game states
- Directed edges in $E$: moves
  - Edge $e = (s_1, s_2)$ contains:
  - State $s_1$ before move
  - State $s_2$ after move

# State Space Representation - `Go1` Example






`board1`




`board2`

- State before move:
  `(board1, to_play = WHITE, ko_recapture = None, ...)`
- Play move: White B1
- State after move:
  `(board2, to_play = BLACK, ko_recapture = None, ...)`

# Terminal State



- A *terminal state* has no possible moves (actions)
- No outgoing edges in graph
- The rules of a game decide:
- When is the game over?
  (did we reach a terminal state?)
- What is the outcome in a terminal state?

# Terminal State in Go



- Game can end in one of two ways
  - A player resigns
  - Both players pass in turn
- Most Go players do not keep playing until there are no legal moves
  - Some moves are bad anyway and should not be played (see `Go0` vs `Go1`)
- Stop playing when the ownership of each point is clear to both players, then count the score
- Example: `Go1.py` stops playing when there are only single point eyes left

# Terminal States and Rewards



- Later, in reinforcement learning, we will talk a lot about *rewards* (or costs: negative rewards)
- In many games, the only reward is at the end, in the terminal state
- Example: +1 if you win, -1 if you lose
- A few games have other, earlier rewards/costs
  - Example: blinds and bidding in poker

# Types of State Spaces



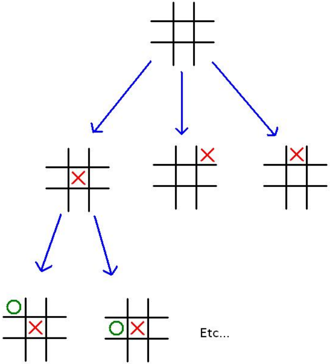
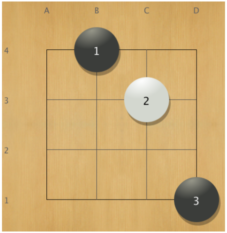


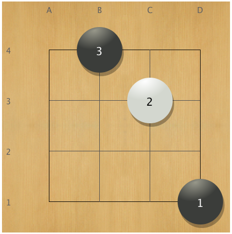

Image source:

`sciencefair.math.iit.edu`

- Assume *root* at the top is current state
  1. Tree
  2. DAG (directed acyclic graph)
  3. DCG (directed cyclic graph)
  - Tree is easiest for search, DCG hardest
- *Game graph, game tree* are other terms for state space of games

# Examples of Types of State Spaces




Same Go state reached with different *move order*

- Tic-Tac-Toe: DAG
  - Different move order can lead to same result
- Go without repetition rules: DCG (cycles exist, e.g. simple ko)
- Go with simple ko rules: still DCG (longer cycles still exist)
- Go with full repetition rules: DAG (details later)
- Chess, checkers, NoGo: discuss.

# Complexity of a State Space



- Some measures of game complexity:
- Size of state space
- Branching factor (number of actions in state)
- More on these later
- Difficulty of game can depend on many other things
  - Is there a simple strategy?
  - A mathematical theory?
  - Many master games to learn from?
  - Good heuristics?
- Difficulty for humans $\neq$ difficulty for computers

# Game Result



- What is the result when a game is over?
- Simplest: win/loss
  - Go with non-integer komi, NoGo, Nim, . . .
- Many games: win/loss/draw
  - Chess, checkers, tic-tac-toe, Go with integer komi, five-in-a-row (gomoku), . . .
- Point-scoring games: size of win matters
  - Score, value, evaluation, result: terms with similar meaning

# Types of Game Representation



- Board representation
- Move history - move list
- Formats:
  - Internal representation
  - File storage: move list + annotations, e.g. sgf file format
    - `www.red-bean.com/sgf/`
    - `en.wikipedia.org/wiki/Smart_Game_Format`
  - Inter-program communication: GTP - Go Text Protocol
    - `www.lysator.liu.se/~gunnar/gtp/`
    - Used to connect `Go0` etc to `GoGui` and other tools

# Review - Board Representation



- Many games are grid-based
- 2-d array
- 1-d array (often faster, standard)
- Bitmaps (sometimes fastest, depends on use case)
- Specialized, e.g. piece list if large board, few pieces.

# Summary - Concepts so Far



- State space search - formal model for decision-making
- Basic general concepts:
  states, actions, state space
- In games:
  game position, game state, move, game tree/game graph
- Terminal states, result, reward
- Representation of board and moves, internal and external (file storage, text communication)

# Models of State Spaces

# Models of State Spaces



- How to choose a state space for a decision-making problem?
- More on Tree, DAG, DCG
- Estimating the size of state space
- Reachable vs unreachable
- Symmetries, equivalent states

# How to Choose States and State Spaces



- State needs all the *relevant information*
- Make it as simple as possible ...
- ... but not simpler

# Example 1: Simplifying Rewards in States



- Example: a zero-sum game, both players collect rewards during the game
- Most direct representation:
  list of rewards at each move for each player
- Example of state:
  `([0, 10, 0, 0, 10], [20, 0, -5, 0, 0], rest-of-state)`
- Better: just keep sum of awards so far:
  `(20, 15, rest-of-state)`
- Even better: only keep *difference* of rewards
  `(5, rest-of-state)`

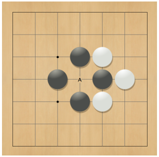
# Example 1 Discussion

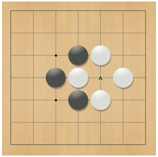


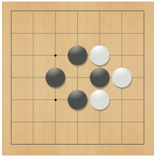
- Why is this representation better?
- All states with same reward difference are *equivalent*
  - The players do not care which of them they are in
- Great reduction in number of states
  - If one of the equivalent states is solved, then *all* are solved
  - Idea here: compress history into a single number, add that to state

# Example 2: Simplifying State in Go









- Consider the *simple ko* rule in Go
- Need to check if the position is the same as two moves ago
- Simple way: store the positions, compare each point on board
- Better way: work out the conditions when the simple ko rule applies
- We need to only store a single point.
  - See `ko_recapture` in `Go1`
- Idea here: compress history into information of a single relevant point, add that to state

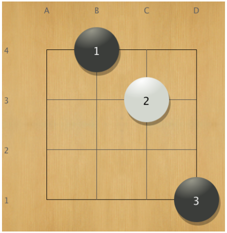
# More on State Spaces - Tree, DAG, DCG

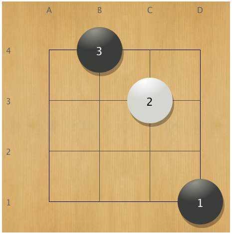


- What are the differences between tree, DAG, DCG?
- If we have a choice, which one to choose?
- We often have a choice!

# State Space - Tree






- Simplest state space model is tree
- Every action leads to new state
- Only one path from root to a node
- We simply ignore it, if different paths lead to the same situation
- Each copy is a separate state
- Example: Two sequences of Go moves
- 1. Black B4, 2. White C3, 3. Black D1
- 1. Black D1, 2. White C3, 3. Black B4
- Different sequences, different states in tree model. Duplication!

# Pros and Cons of Tree



Advantages

- Simplest model
- Single path to each node
- No dependencies

Disadvantages

- Duplication, no re-use of information
- State space can be much larger than needed
- Search can become very inefficient, searches many copies of equivalent sub-trees

# Size of State Space - Simple Model



- How many states?
- Model:
- Assume a *constant branching factor b*
- Each *interior* node in tree has *b* children
- Assume a *uniform depth d*
- Each path from root is *d* actions long

# State Space - Simple Model



- How many nodes?
- 1 root node, 1 = $b^0$ total nodes at depth 0
- $b$ children of root, $b^1$ total nodes at depth 1
- Each child has $b$ new children, total $b^2$ at depth 2
- ...
- Last level $b^n$ nodes at depth $n$
- Total nodes $1 + b + ...b^n = (b^{n+1} - 1)/(b - 1)$
- For large $b$, this is close to $b^n$ - last level dominates

# (Bad) Example - Tic Tac Toe



- 9 possible moves, $b = 9$
- depth at most 9, $d = 9$
- So about $9^9 \approx 387$ million nodes?
- This is a huge over-estimate, why?
- (Discuss)

# A Better Model?



- Games such as TicTacToe only add stones, never remove
- $n$ choices for first move
- $n - 1$ for second move, ...
- Total $n \times (n - 1) \times ...1 = n!$ possible games
- TicTacToe: $9! = 362,880$
- Still too much. Discuss why.

# (Very rough) Example - $7 \times 7$ Go



- 49 moves at start, maybe 25 on average during a game
- Length of game - about 30 moves?
- Rough estimate $25^{30} \approx 10^{42}$
- Compare with $49! \approx 10^{63}$

# Summary - Models of State Spaces



- State space search - formal model for decision-making
- Basic general concepts: states, actions, state space
- In games: game position, game state, move, game tree/game graph
- Terminal states, result, reward
- Representation of board and moves, internal and external