Monte Carlo Tree Search, part 2

CMPUT 355: Games, Puzzles, and Algorithms

Lecture Outline

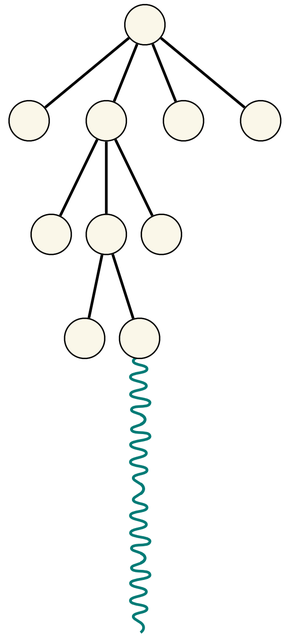
1. Logistics & Recap
2. Strong Solving with Monte Carlo Tree Search
3. Example & Demo

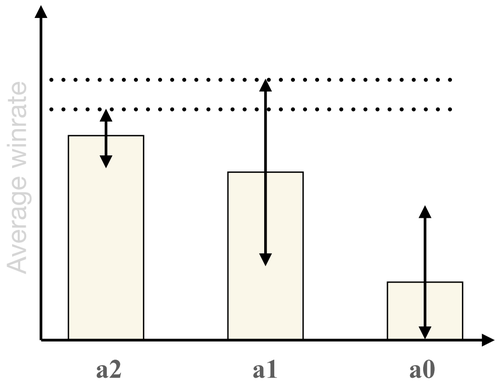
Logistics

- **Practice questions #4** are available
  - Solutions were released yesterday
- **Quiz 4** is **Friday** (Mar 13)
  - *Coverage:* up to and including **Mar 6** (last Friday's lecture)
  - Bring your student ID!
  - No calculators or other devices
  - The quiz will be run by 3 TAs

Recap: Monte Carlo Tree Search

- MCTS maintains a **search tree** which grows as the search progresses
- Leaves are evaluated by rapid, random **simulations**
- Search tree is traversed using **UCT** to balance **exploration** and **exploitation**
- Basic algorithm:
    1. **Traverse:** Starting from the root, traverse the search tree by iteratively choosing children using UCT
    2. If the resulting leaf node has at **least one simulation**:
        1. **Expand:** add all of the leaf's child states to the search tree
        2. **Rollout:** simulate play from one of the child states until the end of the game
    3. Otherwise:
        1. **Rollout:** simulate play from the leaf until the end of the game
    4. **Backpropagate:** Add results of the simulation to performance statistics on every node on the path back to the root
        - E.g., average **score** and number of **visits**





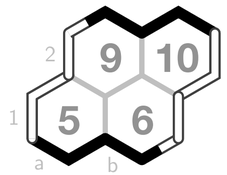
Solving States

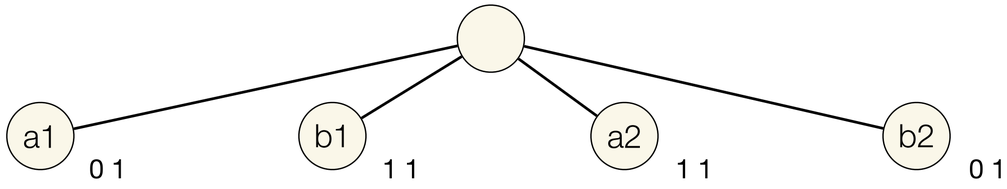
- Recall that a **state** is everything you need to know about a game situation:
  - **Position** (e.g., where the stones are on the board)
  - **Player to move**; sometimes implied by the position (when?)
  - **Question:** What **else** might we need?
- **Question:** What does it mean for a state to be **solved**?
- We distinguish between strongly and weakly solved states:
  - **Strongly solved:** We know the best outcome a player can guarantee, and we know a strategy they can play to guarantee that outcome
  - **Weakly solved:** We know the best outcome a player can guarantee, but not **how**
- **Question:** What games in this course are weakly solved? Which are strongly solved?

MCTS vs Solving

- MCTS looks for **good** moves, but it does not in general solve its root state
  - Minimax, negamax, alpha-beta search all operate by **solving** their root state
  - **Question:** Does **alpha-beta** search **strongly** or **weakly** solve its root state?
- Although MCTS is not guaranteed to solve its root state, sometimes it does (**when?**)
  - Every **simulation** gets to a terminal state, but that **doesn't prove** a win or loss for the starting search tree state (**why?**)
  - But sometimes the search tree makes it all the way to a terminal node! (**how?**)
- In these cases, a proven win/loss can be back-propagated as a score of +/-∞
  - Whenever we add a child that is a proven win for the parent, we don't need to expand any further children (**why?**)

Example: Strong Solving with MCTS





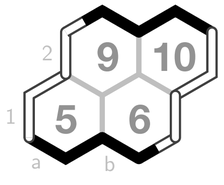
* > 5, 6, 9, 10,
trv_xpnd bu * 5 no-sims child
sim 1. * 5 roll 9 6 10 parent loss

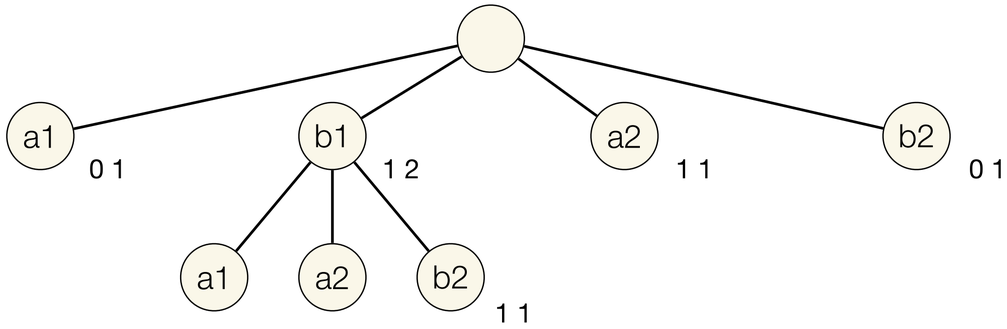
trv_xpnd bu * .0 6 no-sims child
sim 2. * 6 roll 9 10 parent win

trv_xpnd bu * .2 1.2 9 no-sims child
sim 3. * 9 roll 10 6 parent win

trv_xpnd bu * .3 1.3 1.3 10 no-sims child
sim 4. * 10 roll 9 5 6 parent loss

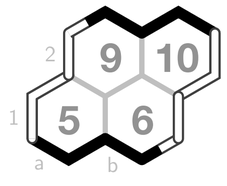
Example: Strong Solving with MCTS

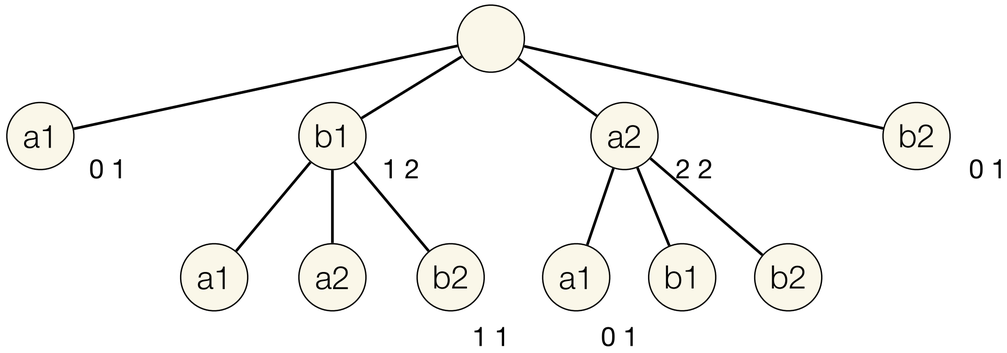




trv_xpnd bu * .4 1.4 1.4 .4 6
  xpnd_nd * 6 > 5
  xpnd_nd * 6 > 9
  xpnd_nd * 6 > 10
sim 5. * 6 10 roll 5 9 parent win

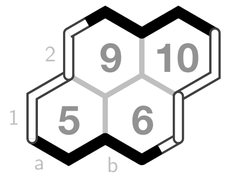
Example: Strong Solving with MCTS

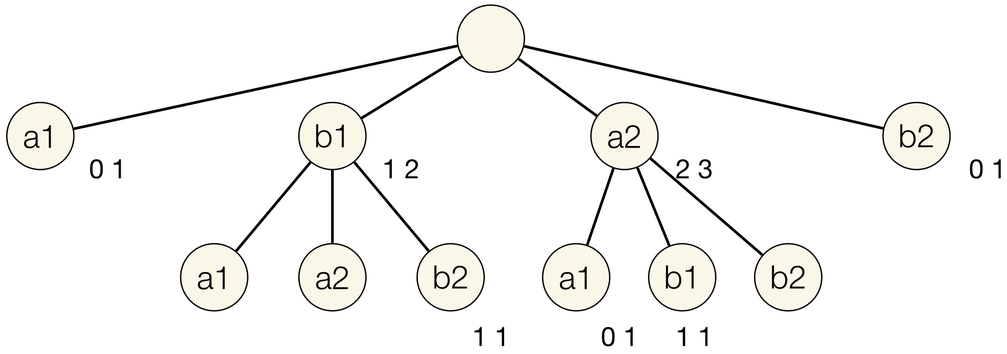




trv_xpnd bu * .4 .8 1.4 .4 9
  xpnd_nd * 9 > 5
  xpnd_nd * 9 > 6
  xpnd_nd * 9 > 10
sim 6. * 9 5 roll 6 parent loss

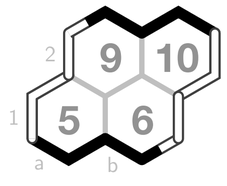
Example: Strong Solving with MCTS

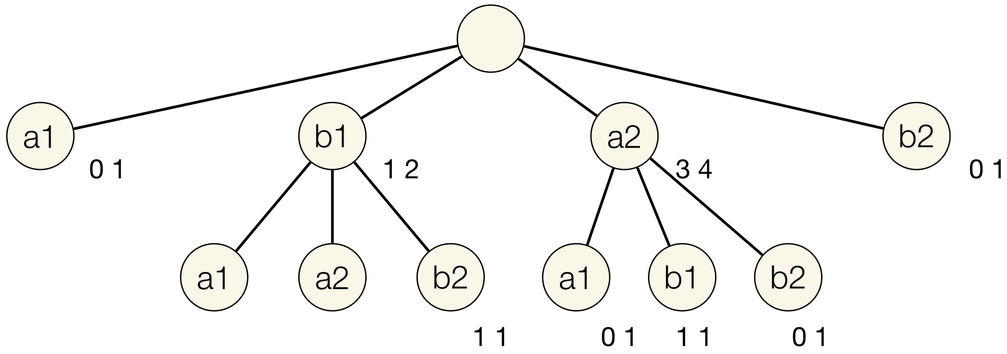




trv_xpnd bu * .4 .8 1.3 .4 9 bu 9 .4 6 no-sims child
sim 7. * 9 6 roll 10 5 parent win

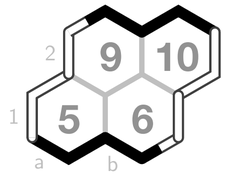
Example: Strong Solving with MCTS

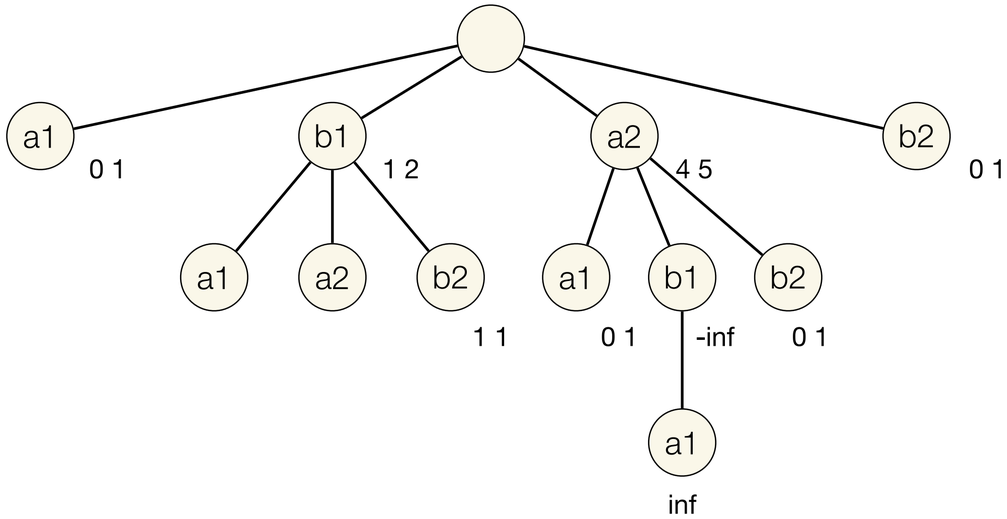




trv_xpnd bu * .4 .8 .9 .4 9 bu 9 .4 1.4 10 no-sims child
sim 8. * 9 10 roll 6 parent loss

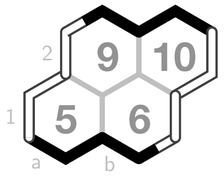
Example: Strong Solving with MCTS

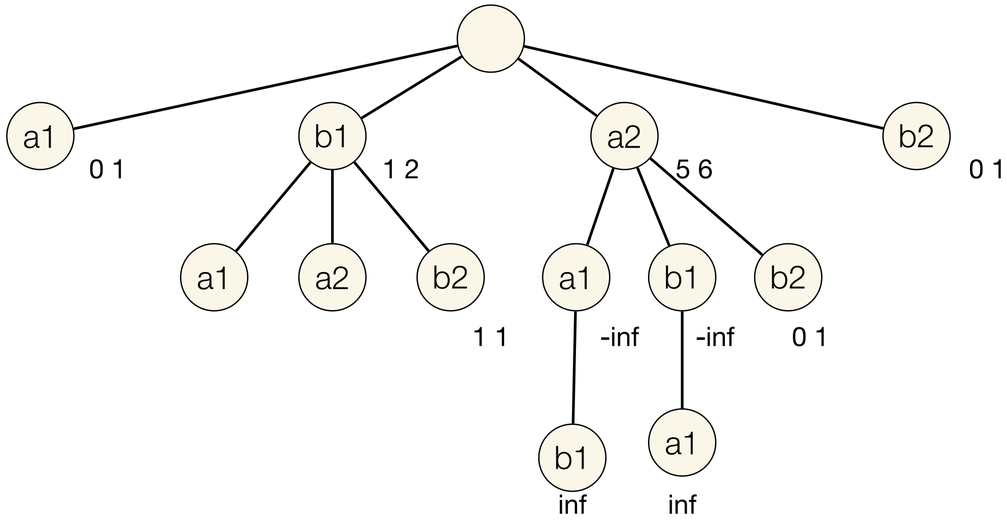




trv_xpnd bu * .4 .8 1.0 .4 9 bu 9 .4 1.4 .4 6
 xpnd_nd * 9 6 > 5
won sim 9. * 9 6 5 win, no more sibs

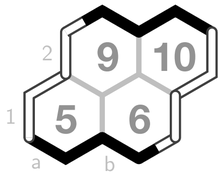
Example: Strong Solving with MCTS

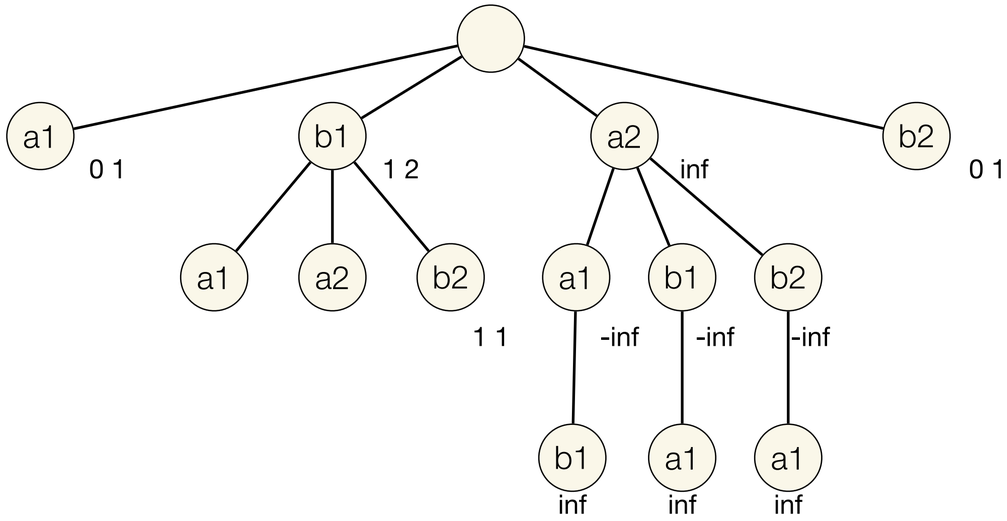




trv_xpnd bu * .4 .8 1.0 .4 9 bu 9 .4 -inf .4 5
 xpnd_nd * 9 5 > 6
won sim 10. * 9 5 6 win, no more sibs

Example: Strong Solving with MCTS






```
trv_xpnd bu * .5 .8 1.0 .5 9 bu 9 -inf -inf .5 10
  xpnd_nd * 9 10 > 5
won sim 11. * 9 10 5 win, no more sibs

done  proven win 0.0 sec
```

Demo: Solving 4×4 Hex

- 4×4 Hex can be solved using **negamax** with early stopping:

cd hex; python3 hex_simple.py; ? x

    - But only if the **cell ordering** is good (**why?**)

- Solved much more rapidly by MCTS!

cd mcts; python3 main.py; g x

    - Even without any Hex-specific optimizations

Summary

- MCTS finds a **good** move, but not necessarily an **optimal** move
- However, if the **search tree** reaches **terminal** nodes it can sometimes **prove** that a move is optimal!
- When a search node is itself a terminal, it has a score of infinity or negative infinity
  - If infinity (i.e., win for parent), there is no need to expand its **siblings**
- When **all** of a search node's children have scores of negative infinity, it is an **optimal** move
- This is an optimization that can be applied to **any game** where we can detect a win
  - I.e., any game that we could apply **boolean negamax**