CS 232: Artificial Intelligence

Fall 2023

Prof. Carolyn Anderson
Wellesley College
Recap
Search

We’ve seen two kinds of search strategies so far:

- Uninformed search
  - Breadth-first search
  - Depth-first search
- Informed search
  - Uniform cost search
  - A* search

Annotate nodes w/ cost from
Priority Queue

\[ F(Node) = G(Node) + H(Node) \]

Start \(\rightarrow\) Node

Node \(\rightarrow\) Goal
**A* search example**

We add the three nodes we found to the Frontier queue. We sort them according to the $g() + h()$ calculation.
We add the three nodes we found to the Frontier queue.
We sort them according to the $g() + h()$ calculation.
A* search example

Ordered by cost

Frontier queue: Fagaras 415 Pitesti 417

Start

Craiovea 526
Sibiu 553
Arad 671
Oradea 671

Timisoara 447
Zerind 449

We add the three nodes we found to the Frontier queue.
We sort them according to the $g()+h()$ calculation.
A* search example

Ordered by cost:

Frontier queue:

- Sibiu 393
- Timisoara 447
- Zerind 449
- Bucharest 450
- Craiovea 526
- Sibiu 553
- Sibiu 591
- Arad 696
- Oradea 671
- Sibiu 591

We add the three nodes we found to the Frontier queue.
We sort them according to the $g() + h()$ calculation.
A* search example

Ordered by cost
Frontier queue:
- Sibiu 393
- Timisoara 447
- Zerind 449
  - Bucharest 450
  - Craiova 526
  - Sibiu 553
  - Sibiu 591
  - Rimnicu Vilcea
  - Crasovia 615
  - Arad 691
- Oradea 671

We add the three nodes we found to the Frontier queue.
We sort them according to the $g() + h()$ calculation.
We add the three nodes we found to the Frontier queue.

We sort them according to the $g() + h()$ calculation.
A* search example

Frontier queue:

Pitesti 417
Timisoara 447
Zerind 449
Bucharest 450
Craiava 526
Sibiu 553
Sibiu 591
Arad 646
Oradea 671

When we expand Fagaras, we find Bucharest, but we’re not done. The algorithm doesn’t end until we “expand” the goal node – it has to be at the top of the Frontier queue.
**A* search example**

Frontier queue:
- Bucharest 418
- Timisoara 447
- Zerind 449
- **Bucharest 450**
- Craiova 526
- Sibiu 553
- Sibiu 591
- Rimnicu Vïcea 607
- Craiova 615
- Arad 646
- Oradea 671

Note that we just found a better value for Bucharest!
Now we expand this better value for Bucharest since it’s at the top of the queue.
We’re done and we know the value found is optimal!
Adversarial Search
So far, we have only considered one-player games.

What happens when we add another player?
Multiplayer Games

In competitive multiplayer games, we have to consider our opponent’s possible actions, as well as our own.

We call this adversarial search.
Game Playing State-of-the-Art

- **Checkers:** 1950: First computer player. 1994: First computer champion: Chinook ended 40-year-reign of human champion Marion Tinsley using complete 8-piece endgame. 2007: Checkers solved!

- **Chess:** 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200M positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

- **Go:** 2016: Alpha GO defeats human champion. Uses Monte Carlo Tree Search + neural network to learn evaluation function.

- **Go + Chess + Shogi:** 2017: Alpha Zero learns all 3 games using reinforcement learning to play against itself.

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Types of Games

- Many different kinds of games!

- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?

- Want algorithms for calculating a strategy (policy) which recommends a move from each state

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Deterministic Games

- Many possible formalizations, one is:
  - States: $S$ (start at $s_0$)
  - Players: $P=\{1...N\}$ (usually take turns)
  - Actions: $A$ (may depend on player / state)
  - Transition Function: $S \times A \rightarrow S$
  - Terminal Test: $S \rightarrow \{t,f\}$
  - Terminal Utilities: $S \times P \rightarrow R$

- Solution for a player is a policy: $S \rightarrow A$
Zero-Sum Games

- Agents have opposite utilities (values on outcomes)
- Lets us think of a single value that one maximizes and the other minimizes
- Adversarial, pure competition

General Games

- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, and more are all possible
Adversarial Search
Single-Agent Trees
Adversarial Game Trees
Minimax Values

States Under Agent’s Control:
-8
-5

States Under Opponent’s Control:
-10
+8

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result

- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node’s minimax value: the best achievable utility against a rational (optimal) adversary
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v

def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v

\[ V(s) = \max_{s' \in \text{successors}(s)} V(s') \]
\[ V(s') = \min_{s \in \text{successors}(s')} V(s) \]
Minimax Implementation (Dispatch)

```python
def value(state):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)

def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Which nodes are under Max's control?

Which nodes are under Min's control?
Minimax Properties

Optimal against a perfect player. Otherwise?
Video of Demo Min vs. Exp (Min)

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Video of Demo Min vs. Exp
(Exp)
Another Demo

Diagram:

- A
  - B
    - E
      - M
      - N
    - F
      - 10
  - C
    - G
      - 3
    - H
      - 6
    - I
      - 9
  - D
    - J
      - 10
    - K
      - 1
    - L
      - 2
Minimax Summary

- Rank final game states by their final scores (for tic-tac-toe or chess: win, draw, loss).

- Rank intermediate game states by whose turn it is and the available moves.
  - If it's X's turn, set the rank to that of the maximum move available. If a move will result in a win, X should take it.
  - If it's O's turn, set the rank to that of the minimum move available. If a move will result in a loss, X should avoid it.
Efficiency
How efficient is minimax?
- Just like (exhaustive) DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

Example: For chess, $b \approx 35$, $m \approx 100$
- Exact solution is completely infeasible
- But, do we need to explore the whole tree?
Game Tree Pruning
Pruning

- **Key idea**: give up on paths when you realize that they are worse than options you’ve already explore.
- Track the maximum score that the minimizing player (beta) can get
- Track the minimum score that the maximizing player (alpha) can get
- Whenever the maximum score that beta can get becomes less than the minimum score that alpha can get, the maximizing player can stop searching down this path, because it will never be reached.
Minimax Example

Alpha-Beta Pruning

- **General configuration (MIN version)**
  - We’re computing the MIN-VALUE at some node $n$
  - We’re looping over $n$’s children
  - $n$’s estimate of the childrens’ min is dropping
  - Who cares about $n$’s value? MAX
  - Let $a$ be the best value that MAX can get at any choice point along the current path from the root
  - If $n$ becomes worse than $a$, MAX will avoid it, so we can stop considering $n$’s other children (it’s already bad enough that it won’t be played)
- **MAX version is symmetric**
def min-value(state, α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
    if v ≤ α return v
    β = min(β, v)
    return v

def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
    if v ≥ β return v
    α = max(α, v)
    return v

α: MAX’s best option on path to root
β: MIN’s best option on path to root
Alpha-Beta Pruning Properties

- This pruning has **no effect** on minimax value computed for the root!

- **Values of intermediate nodes might be wrong**
  - Important: children of the root may have the wrong value
  - So the most naïve version won’t let you do action selection

- **Good child ordering improves effectiveness of pruning**

- **With “perfect ordering”:**
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless...

- **This is a simple example of metareasoning** (computing about what to compute)
Problem: In realistic games, cannot search to leaves!

Solution: Depth-limited search
- Instead, search only to a limited depth in the tree
- Replace terminal utilities with an evaluation function for non-terminal positions

Example:
- Suppose we have 100 seconds, can explore 10K nodes/sec
- So can check 1M nodes per move
- $\alpha-\beta$ reaches about depth 8 - decent chess program

Guarantee of optimal play is gone

More plies makes a BIG difference

Use iterative deepening for an anytime algorithm
Video of Demo Thrashing (d=2)
Why Pacman Starves

A danger of replanning agents!

- He knows his score will go up by eating the dot now (west, east)
- He knows his score will go up just as much by eating the dot later (east, west)
- There are no point-scoring opportunities after eating the dot (within the horizon, two here)
- Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!
Why Pacman Starves

- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!
Evaluation Functions
Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search

  - Ideal function: returns the actual minimax value of the position
  - In practice: typically weighted linear sum of features:

    - e.g. \( f_1(s) = (\text{num white queens} - \text{num black queens}) \), etc.

\[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]
Evaluation for Pacman
Video of Demo Smart Ghosts (Coordination)
Video of Demo Smart Ghosts (Coordination) - Zoomed In

Slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley
Depth Matters

- Evaluation functions are always imperfect
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
- An important example of the tradeoff between complexity of features and complexity of computation
Video of Demo Limited Depth (10)
Synergies between Evaluation Function and Alpha-Beta?

- **Alpha-Beta**: amount of pruning depends on expansion ordering
  - Evaluation function can provide guidance to expand most promising nodes first (which later makes it more likely there is already a good alternative on the path to the root)
    - (somewhat similar to role of A* heuristic)

- **Alpha-Beta**: (similar for roles of min-max swapped)
  - Value at a min-node will only keep going down
  - Once value of min-node lower than better option for max along path to root, can prune
  - Hence: IF evaluation function provides upper-bound on value at min-node, and upper-bound already lower than better option for max along path to root
    THEN can prune