UIT2504 Artificial Intelligence

Playing Games

C. Aravindan
<AravindanC@ssn.edu.in>

Professor of Information Technology
SSN College of Engineering

September 04, 2024

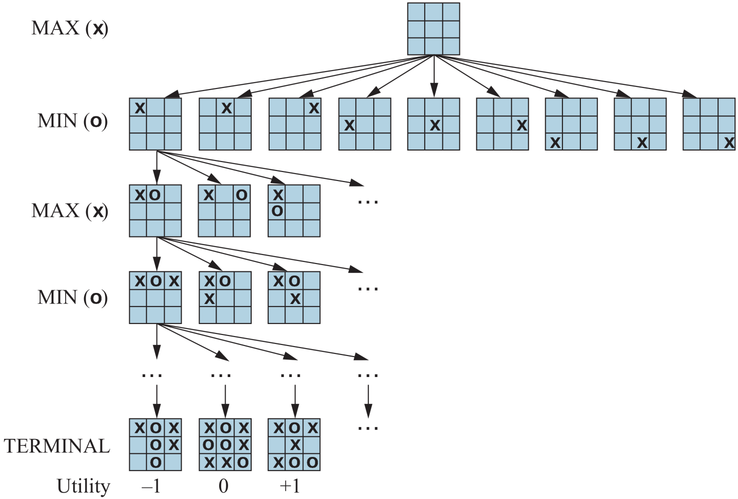
Problem Solving

- Problem formulation in state space
- Basic Search Strategies
- Heuristics — best-first search
- A^* Search
- Local Search Strategies — Hill climbing and its variants, simulated annealing, genetic algorithms
- Searching in continuous domains

Problem Solving

- Problem formulation in state space
- Basic Search Strategies
- Heuristics — best-first search
- A^* Search
- Local Search Strategies — Hill climbing and its variants, simulated annealing, genetic algorithms
- Searching in continuous domains
- Will these ideas be useful for taking decisions in multi-player games?

Search Strategies for Playing Games



Game Theory

- Search tree is huge! — in chess, average branching factor is 35 and a game may go to a depth of 100 (50 moves by each player) — 35^{100}

- Search tree is huge! — in chess, average branching factor is 35 and a game may go to a depth of 100 (50 moves by each player) — 35^{100}
- There may be three approaches to handle multi-agent environments:

Game Theory

- Search tree is huge! — in chess, average branching factor is 35 and a game may go to a depth of 100 (50 moves by each player) — 35^{100}
- There may be three approaches to handle multi-agent environments:
  - Consider all agents in aggregate as an economy

Game Theory

- Search tree is huge! — in chess, average branching factor is 35 and a game may go to a depth of 100 (50 moves by each player) — 35^{100}
- There may be three approaches to handle multi-agent environments:
  - Consider all agents in aggregate as an economy
  - Consider all the adversarial agents as parts of the environment — that makes the environment nondeterministic

Game Theory

- Search tree is huge! — in chess, average branching factor is 35 and a game may go to a depth of 100 (50 moves by each player) — 35^{100}
- There may be three approaches to handle multi-agent environments:
  - Consider all agents in aggregate as an economy
  - Consider all the adversarial agents as parts of the environment — that makes the environment nondeterministic
  - Explicitly model the adversarial agents and develop special techniques for game-tree search

Game Theory

- Search tree is huge! — in chess, average branching factor is 35 and a game may go to a depth of 100 (50 moves by each player) — 35^{100}
- There may be three approaches to handle multi-agent environments:
  - Consider all agents in aggregate as an economy
  - Consider all the adversarial agents as parts of the environment — that makes the environment nondeterministic
  - Explicitly model the adversarial agents and develop special techniques for game-tree search
- We will consider two-player Zero-sum games of perfect information

Game Theory

- Search tree is huge! — in chess, average branching factor is 35 and a game may go to a depth of 100 (50 moves by each player) — 35^{100}
- There may be three approaches to handle multi-agent environments:
  - Consider all agents in aggregate as an economy
  - Consider all the adversarial agents as parts of the environment — that makes the environment nondeterministic
  - Explicitly model the adversarial agents and develop special techniques for game-tree search
- We will consider two-player Zero-sum games of perfect information
- We will call our two players *MAX* and *MIN*

Game Formulation

- States and the initial state S_0

Game Formulation

- States and the initial state S_0
- TO_MOVE(s): The player whose turn it is to move in state s

Game Formulation

- States and the initial state S_0
- TO_MOVE(s): The player whose turn it is to move in state s
- ACTIONS(s): set of legal moves in a state s

Game Formulation

- States and the initial state S_0
- TO_MOVE(s): The player whose turn it is to move in state s
- ACTIONS(s): set of legal moves in a state s
- RESULT(s,a): the transition model that defines the result of a move a in a state s

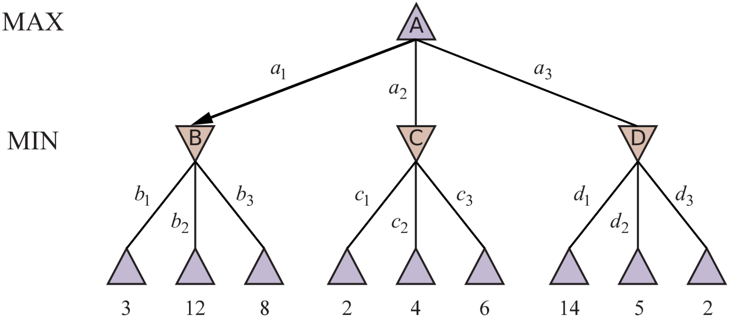
Game Formulation

- States and the initial state S_0
- TO_MOVE(s): The player whose turn it is to move in state s
- ACTIONS(s): set of legal moves in a state s
- RESULT(s,a): the transition model that defines the result of a move a in a state s
- IS_TERMINAL(s): Terminal test — 'true' when the game is over — terminal states

Game Formulation

- States and the initial state S_0
- TO_MOVE(s): The player whose turn it is to move in state s
- ACTIONS(s): set of legal moves in a state s
- RESULT(s,a): the transition model that defines the result of a move a in a state s
- IS_TERMINAL(s): Terminal test — 'true' when the game is over — terminal states
- UTILITY(s,p): Utility function — an objective function that defines the final numeric value to a player p when the game ends in a terminal state s — in chess, outcome is a win, loss, or draw, with values $+1, 0, \frac{1}{2}$

Optimal Decisions



Minimax Decision

- Optimal strategy for deterministic games
- Utility values percolate up from terminal states
- At MAX level, choose successor with highest utility
- At MIN level, choose successor with lowest utility (note that utility is from MAX's point of view)

Minimax Decision

- Optimal strategy for deterministic games
- Utility values percolate up from terminal states
- At MAX level, choose successor with highest utility
- At MIN level, choose successor with lowest utility (note that utility is from MAX's point of view)

MINIMAX(s) $=$

$$\begin{cases} Utility(s, MAX) & \text{if IS\_TERMINAL}(s) \\ max_{a \in ACTIONS(s)} MINIMAX(RESULT(s, a)) & \text{if TO\_MOVE}(s) = \text{MAX} \\ min_{a \in ACTIONS(s)} MINIMAX(RESULT(s, a)) & \text{if TO\_MOVE}(s) = \text{MIN} \end{cases}$$

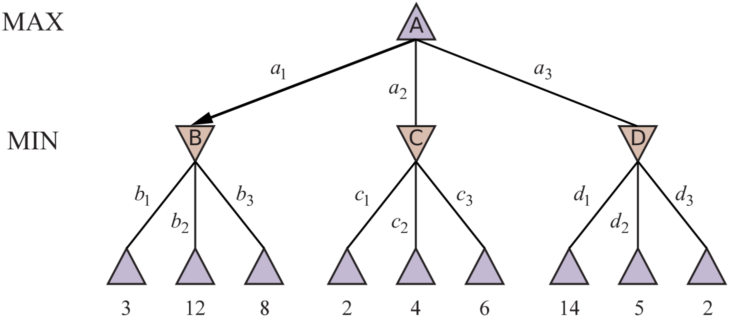
Minimax Algorithm

```
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 ← −∞
  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 ← +∞
  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
```

Optimal Decisions



Complexities

- Complete?

Complexities

- Complete? — Yes, if the tree is finite

Complexities

- Complete? — Yes, if the tree is finite — complete depth-first search

Complexities

- Complete? — Yes, if the tree is finite — complete depth-first search
- Optimal?

Complexities

- Complete? — Yes, if the tree is finite — complete depth-first search
- Optimal? — Yes, if the opponent is optimal

Complexities

- Complete? — Yes, if the tree is finite — complete depth-first search
- Optimal? — Yes, if the opponent is optimal
- Space Complexity?

Complexities

- Complete? — Yes, if the tree is finite — complete depth-first search
- Optimal? — Yes, if the opponent is optimal
- Space Complexity? — $O(bm)$

Complexities

- Complete? — Yes, if the tree is finite — complete depth-first search
- Optimal? — Yes, if the opponent is optimal
- Space Complexity? — $O(bm)$ — may be reduced to $O(m)$ if successors are generated one at a time

Complexities

- Complete? — Yes, if the tree is finite — complete depth-first search
- Optimal? — Yes, if the opponent is optimal
- Space Complexity? — $O(bm)$ — may be reduced to $O(m)$ if successors are generated one at a time
- Time Complexity?

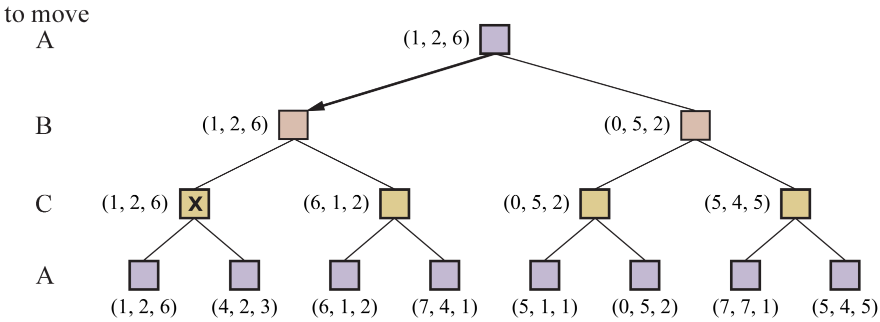
Complexities

- Complete? — Yes, if the tree is finite — complete depth-first search
- Optimal? — Yes, if the opponent is optimal
- Space Complexity? — $O(bm)$ — may be reduced to $O(m)$ if successors are generated one at a time
- Time Complexity? — $O(b^m)$

Complexities

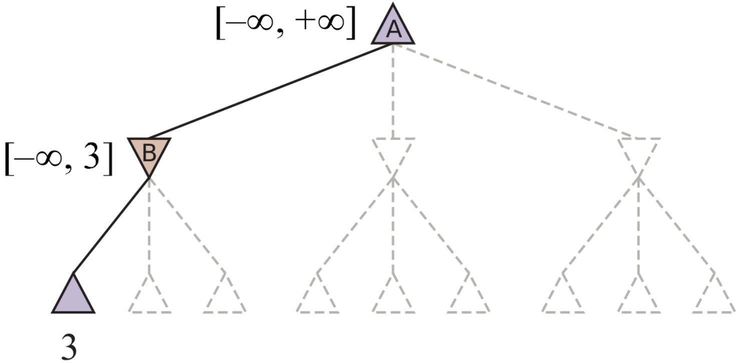
- Complete? — Yes, if the tree is finite — complete depth-first search
- Optimal? — Yes, if the opponent is optimal
- Space Complexity? — $O(bm)$ — may be reduced to $O(m)$ if successors are generated one at a time
- Time Complexity? — $O(b^m)$
- Impractical for non-trivial games such as chess — 35^{100}

Multiplayer Games

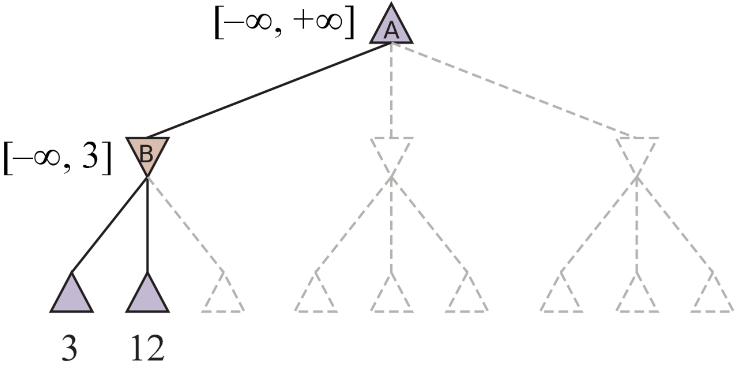


Questions?

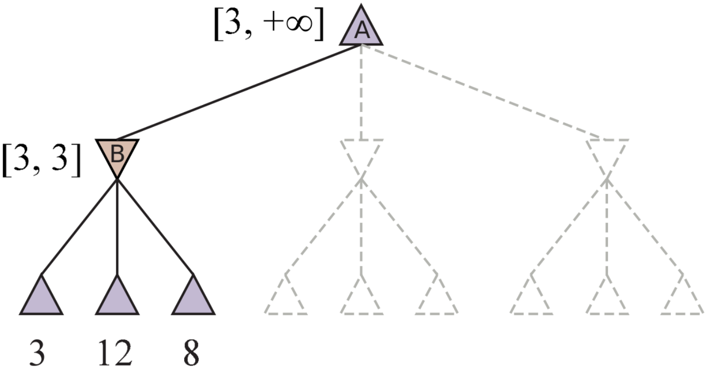
Pruning



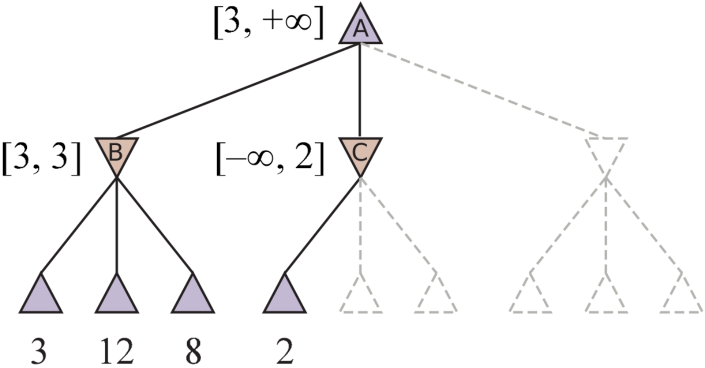
Pruning



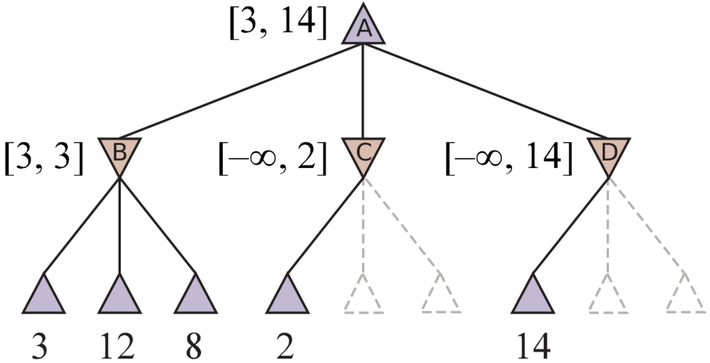
Pruning



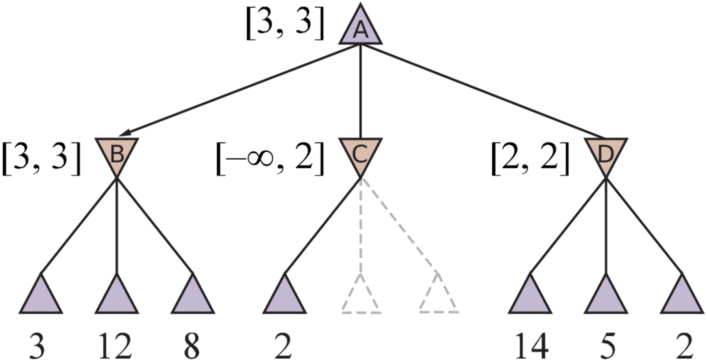
Pruning



Pruning



Pruning

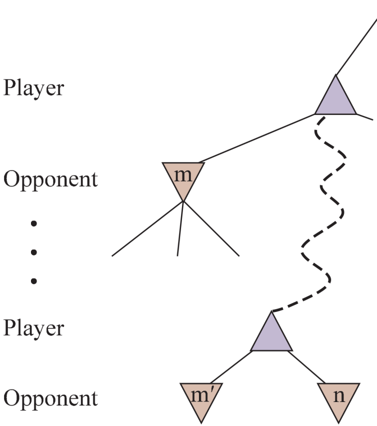


Alpha-Beta Pruning

- α : Value of the best choice we have found so far along a path for MAX — α = "at least" — lower bound for MAX
- β : Value of the best choice we have found so far along a path for MIN — β = "at most" — upper bound for MAX

Alpha-Beta Pruning

- α : Value of the best choice we have found so far along a path for MAX — $\alpha =$ "at least" — lower bound for MAX
- β : Value of the best choice we have found so far along a path for MIN — $\beta =$ "at most" — upper bound for MAX



Alpha-Beta Search Algorithm

```
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
```

Complexities

- α–β pruning does not affect the completeness or optimality of the minimax algorithm

Complexities

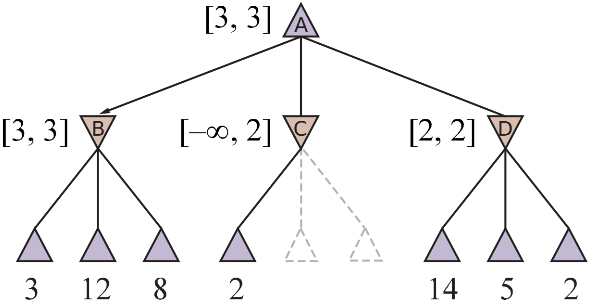
- α–β pruning does not affect the completeness or optimality of the minimax algorithm
- Space complexity is also same as that of the minimax algorithm

Complexities

- α–β pruning does not affect the completeness or optimality of the minimax algorithm
- Space complexity is also same as that of the minimax algorithm
- If the successors can be generated in best-first order (can it be done?), number of nodes to be examined may be halved

Complexities

- α–β pruning does not affect the completeness or optimality of the minimax algorithm
- Space complexity is also same as that of the minimax algorithm
- If the successors can be generated in best-first order (can it be done?), number of nodes to be examined may be halved



Move Ordering

- If best move ordering can be achieved, time taken may be halved

Move Ordering

- If best move ordering can be achieved, time taken may be halved — $O(b^{m/2})$

Move Ordering

- If best move ordering can be achieved, time taken may be halved — $O(b^{m/2})$ — which is same as $O\left(\left(\sqrt{b}\right)^m\right)$ —

Move Ordering

- If best move ordering can be achieved, time taken may be halved — $O(b^{m/2})$ — which is same as $O\left(\left(\sqrt{b}\right)^m\right)$ — effective branching factor is reduced to $\sqrt{b}$

Move Ordering

- If best move ordering can be achieved, time taken may be halved — $O(b^{m/2})$ — which is same as $O\left(\left(\sqrt{b}\right)^m\right)$ — effective branching factor is reduced to $\sqrt{b}$
- If the successors are examined in a random order, time complexity will be roughly $O(b^{3m/4})$

Move Ordering

- If best move ordering can be achieved, time taken may be halved — $O(b^{m/2})$ — which is same as $O\left(\left(\sqrt{b}\right)^m\right)$ — effective branching factor is reduced to $\sqrt{b}$
- If the successors are examined in a random order, time complexity will be roughly $O(b^{3m/4})$
- Heuristics may be used to order the moves

Move Ordering

- If best move ordering can be achieved, time taken may be halved — $O(b^{m/2})$ — which is same as $O\left(\left(\sqrt{b}\right)^m\right)$ — effective branching factor is reduced to $\sqrt{b}$
- If the successors are examined in a random order, time complexity will be roughly $O(b^{3m/4})$
- Heuristics may be used to order the moves — in chess, captures first, then threats, then forward moves, and then backward moves

Move Ordering

- If best move ordering can be achieved, time taken may be halved — $O(b^{m/2})$ — which is same as $O\left(\left(\sqrt{b}\right)^m\right)$ — effective branching factor is reduced to $\sqrt{b}$
- If the successors are examined in a random order, time complexity will be roughly $O(b^{3m/4})$
- Heuristics may be used to order the moves — in chess, captures first, then threats, then forward moves, and then backward moves
- Dynamic move ordering — moves found to be best in the past (learning) — or, iterative-deepening

Move Ordering

- If best move ordering can be achieved, time taken may be halved — $O(b^{m/2})$ — which is same as $O\left(\left(\sqrt{b}\right)^m\right)$ — effective branching factor is reduced to $\sqrt{b}$
- If the successors are examined in a random order, time complexity will be roughly $O(b^{3m/4})$
- Heuristics may be used to order the moves — in chess, captures first, then threats, then forward moves, and then backward moves
- Dynamic move ordering — moves found to be best in the past (learning) — or, iterative-deepening
- Transposition table — hash the α–β values for future use

Type A and Type B Strategies

- Even with alpha-beta pruning and other techniques such as move ordering, minimax algorithm may not work for games such as Chess and Go

Type A and Type B Strategies

- Even with alpha-beta pruning and other techniques such as move ordering, minimax algorithm may not work for games such as Chess and Go
- Claude Shannon proposed two additional strategies:
- Type A strategy: consider all possible moves to certain depth and then use a heuristic evaluation function to estimate the utilities of the states at that depth (wide but shallow strategy)

Type A and Type B Strategies

- Even with alpha-beta pruning and other techniques such as move ordering, minimax algorithm may not work for games such as Chess and Go
- Claude Shannon proposed two additional strategies:
- Type A strategy: consider all possible moves to certain depth and then use a heuristic evaluation function to estimate the utilities of the states at that depth (wide but shallow strategy)
- Type B Strategy: ignore moves that look bad, and follow promising paths as far as possible (deep but narrow strategy)

Questions?