CSE 411: Artificial Intelligence (Elective Course #6)

400 Level, Mechatronics Engineering
2nd Term 2016/2017, Lecture #7

Hazem Shehata

Dept. of Computer & Systems Engineering
Zagazig University

April 19th, 2017

Credits to Dr. Mohamed El Abd for the slides

Adminstrivia

Notes

- Assignment #3:
  - Due: today.

Course Info:

- Website: http://hshehata.github.io/courses/zu/cse411/
- Office hours: Sunday 11:30am - 12:30pm

Adminstrivia

Notes

- Assignment #3:
  - Due: today.
- Midterm:
  - Date: Wednesday, Apr. 26, 2017.
  - Time: 1:00pm - 2:30pm.
  - Scope: lectures $1 \rightarrow 6$.

Course Info:

- Website: http://hshehata.github.io/courses/zu/cse411/
- Office hours: Sunday 11:30am - 12:30pm

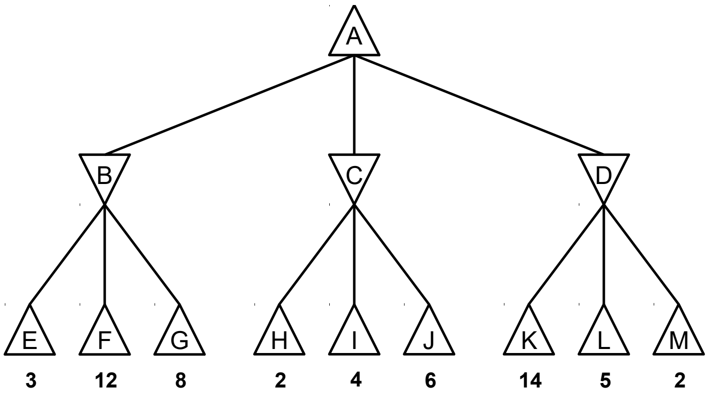
Outline

Outline

Adversarial search

Example: Effect of move ordering on α-β pruning

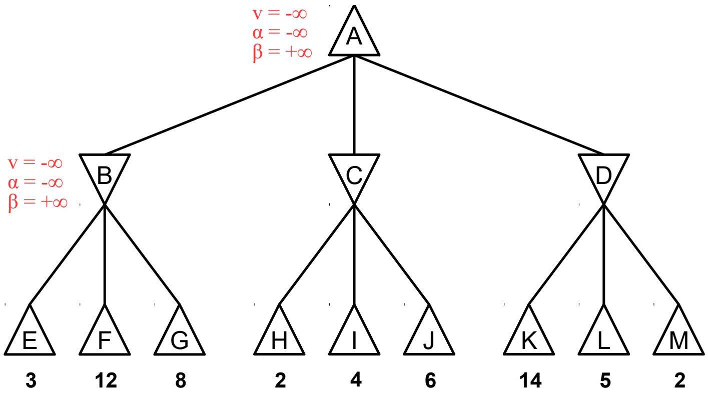
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

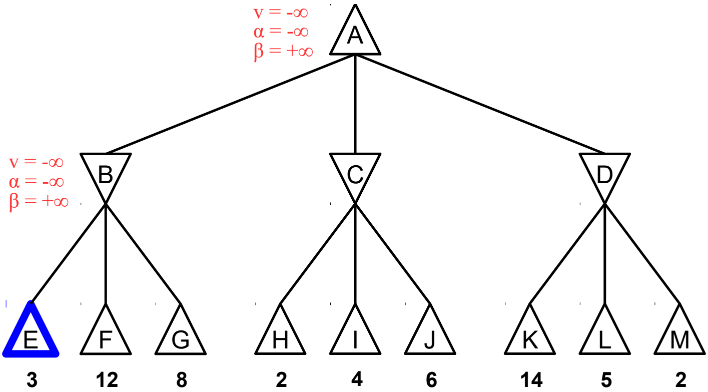
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

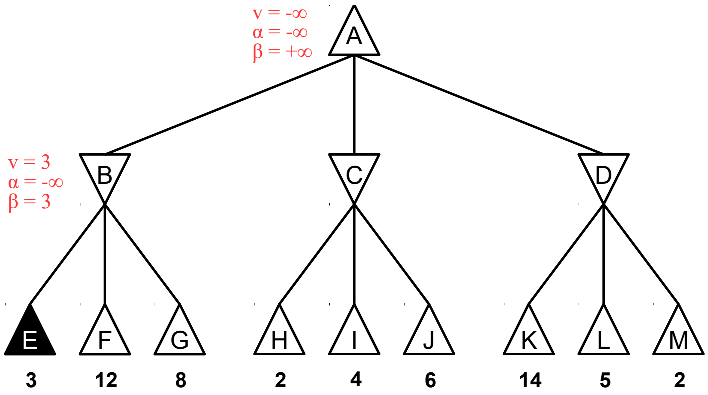
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

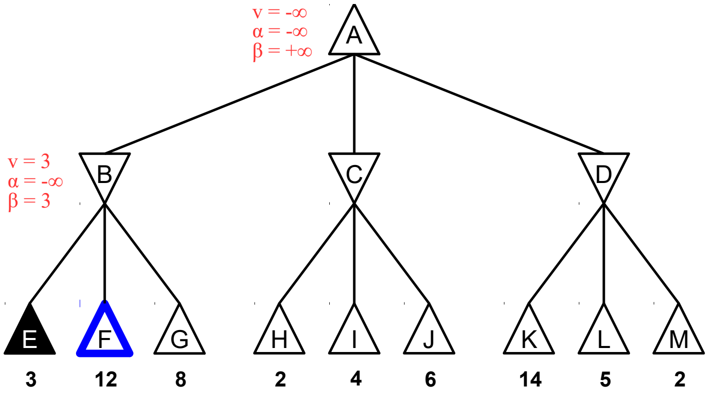
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

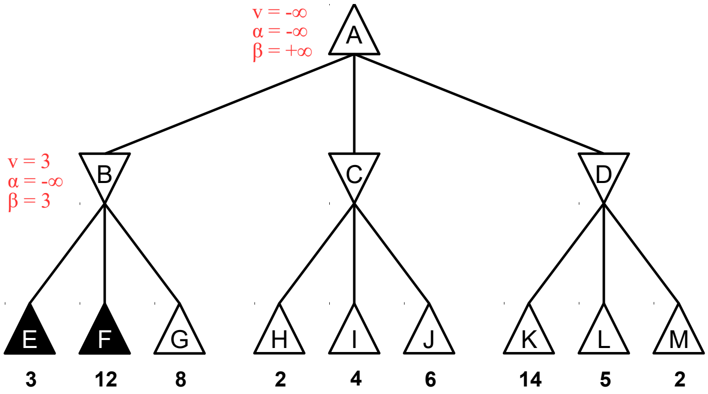
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

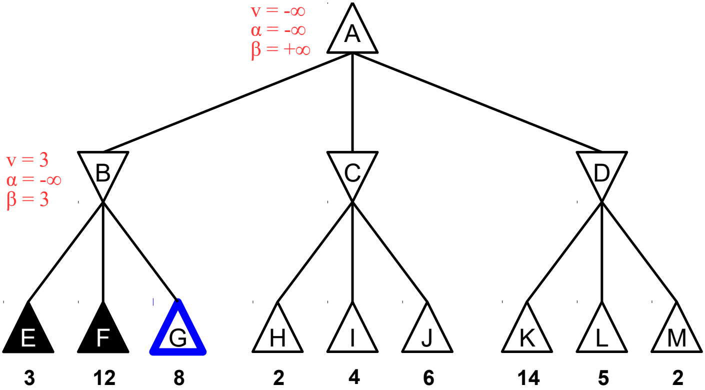
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

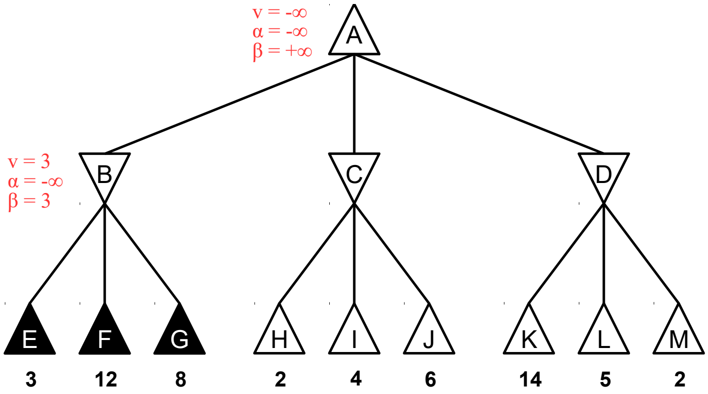
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

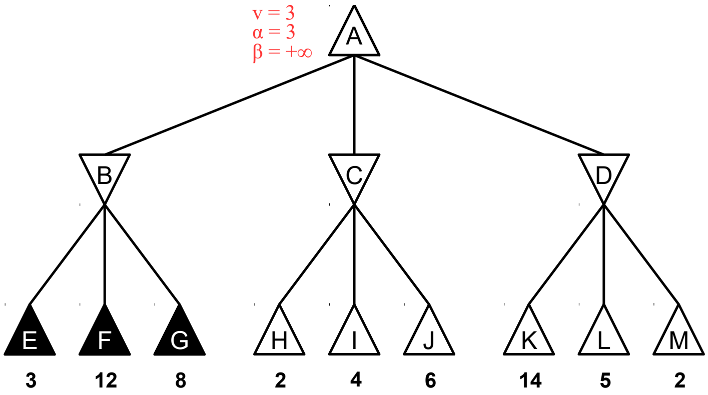
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

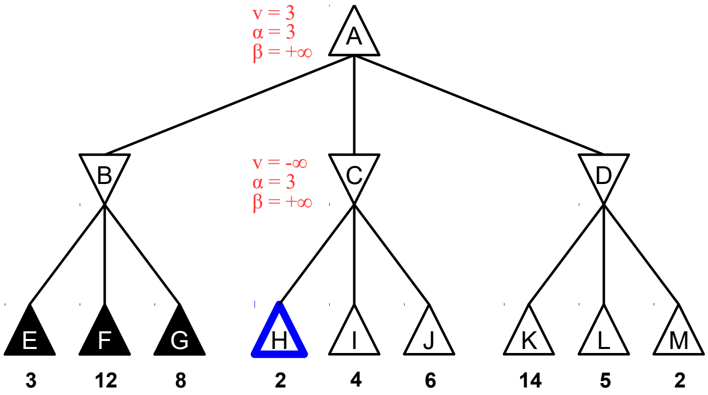
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

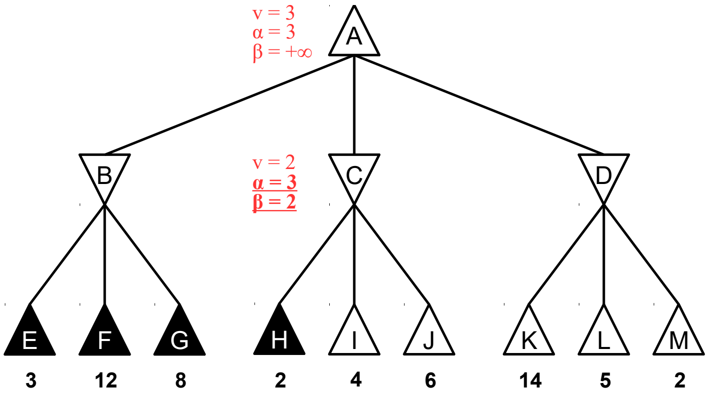
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

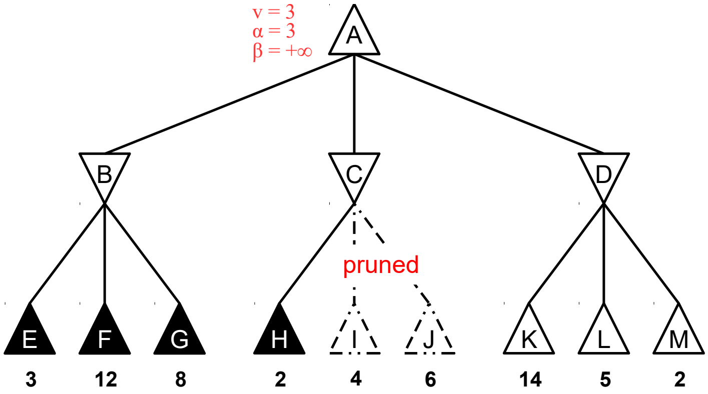
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

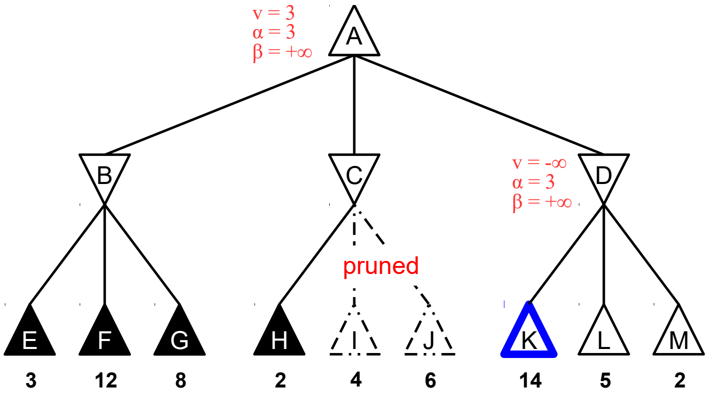
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

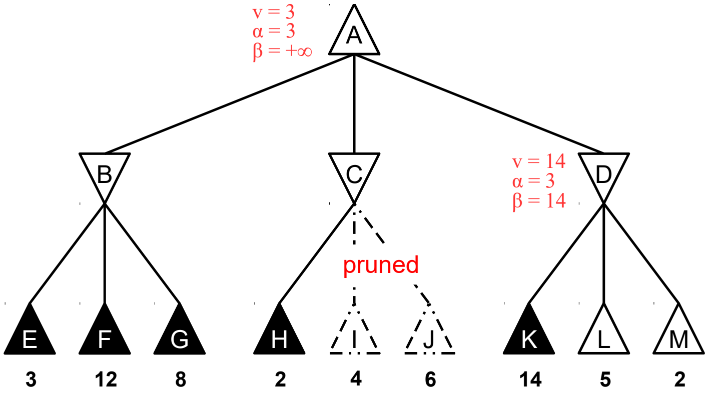
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

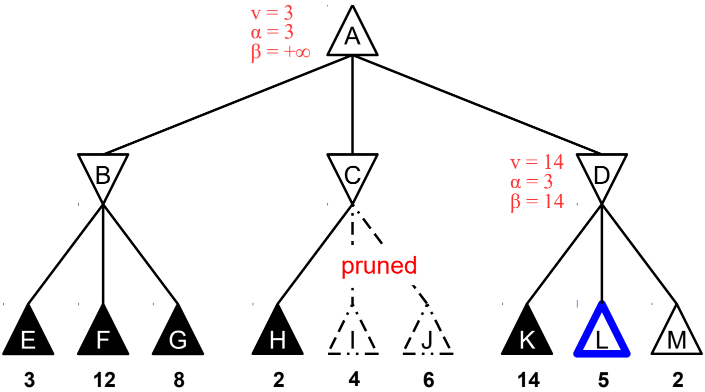
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

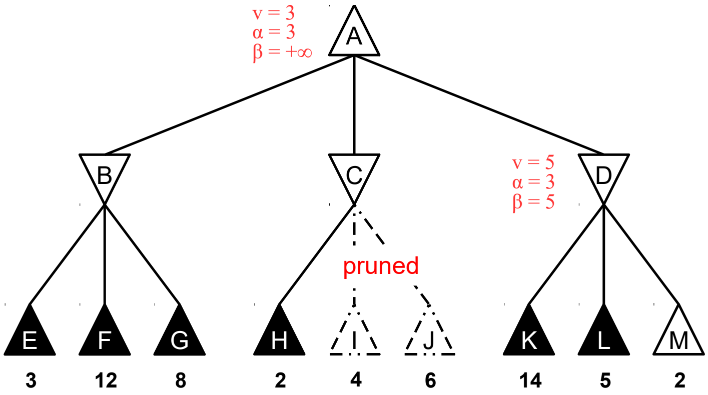
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

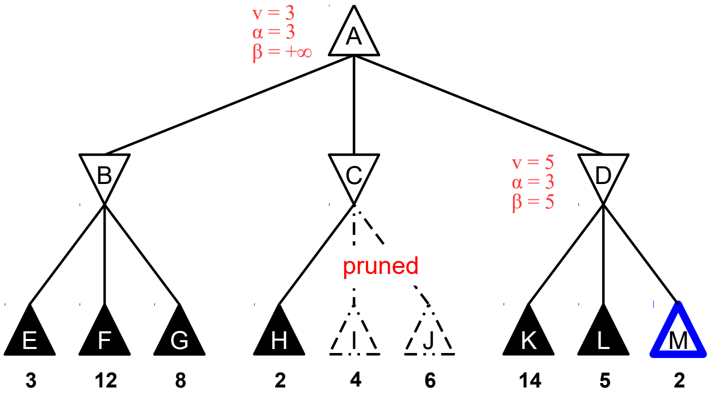
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

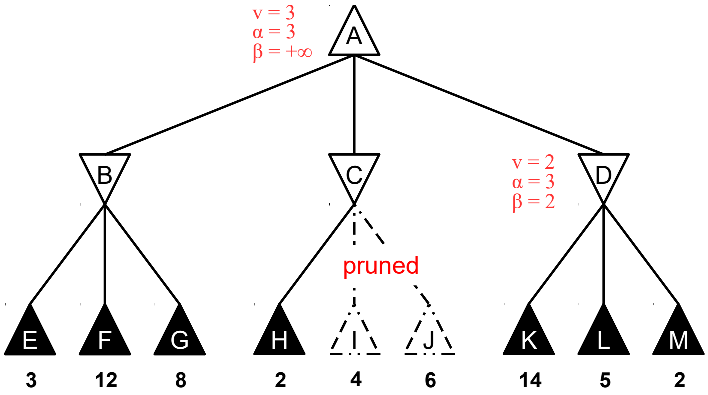
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

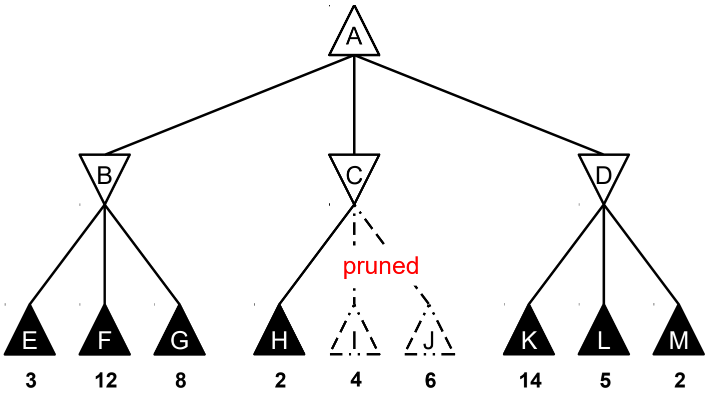
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

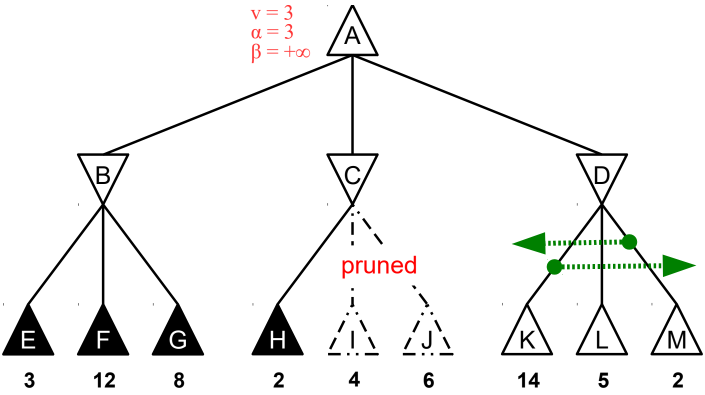
Apply α-β pruning to calculate minimax value for node A.



Adversarial search

Example: Effect of move ordering on α-β pruning

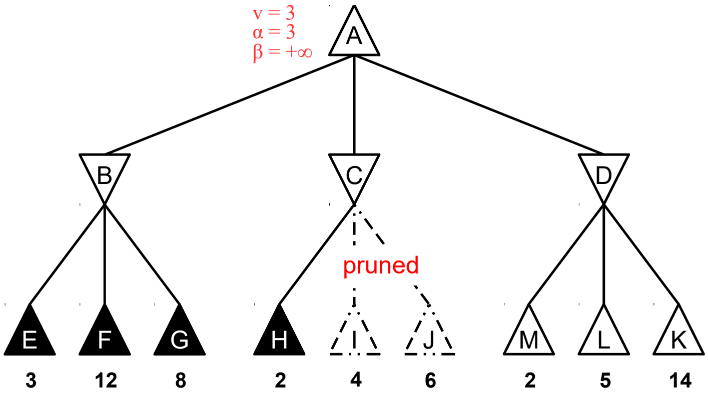
Reverse the order of the utility values for K, L, and M.



Adversarial search

Example: Effect of move ordering on α-β pruning

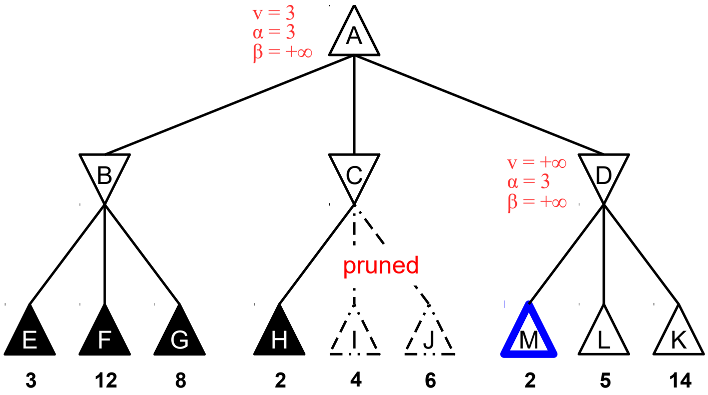
Reverse the order of the utility values for K, L, and M.



Adversarial search

Example: Effect of move ordering on α-β pruning

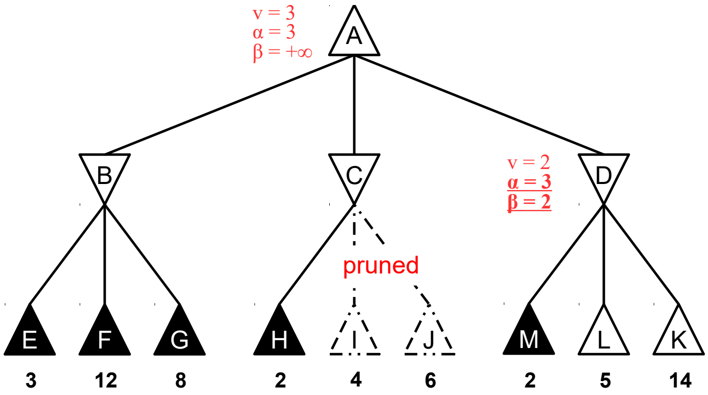
Reverse the order of the utility values for K, L, and M.



Adversarial search

Example: Effect of move ordering on α-β pruning

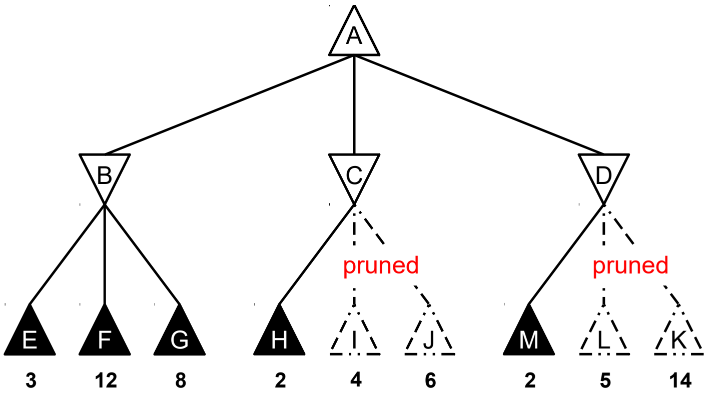
Reverse the order of the utility values for K, L, and M.



Adversarial search

Example: Effect of move ordering on α-β pruning

Reverse the order of the utility values for K, L, and M.



Adversarial search

Effect of move ordering on α-β pruning

- In the previous example, changing the order of nodes increases the efficiency of α-β pruning.

Adversarial search

Effect of move ordering on α-β pruning

- In the previous example, changing the order of nodes increases the efficiency of α-β pruning.
- Efficiency dependant on the ordering of children:
  - Checks each of **MAX's** children until finding one with a value higher than β.
  - Checks each of **MIN's** children until finding one with a value lower than α.

Adversarial search

Effect of move ordering on α-β pruning

- In the previous example, changing the order of nodes increases the efficiency of α-β pruning.
- Efficiency dependant on the ordering of children:
  - Checks each of **MAX's** children until finding one with a value higher than β.
  - Checks each of **MIN's** children until finding one with a value lower than α.
- Can use heuristics to order the nodes to check:
  - Check the highest-value children first for **MAX**.
  - Check the lowest-value children first for **MIN**.

Adversarial search

Effect of move ordering on α-β pruning

- In the previous example, changing the order of nodes increases the efficiency of α-β pruning.
- Efficiency dependant on the ordering of children:
  - Checks each of **MAX's** children until finding one with a value higher than β.
  - Checks each of **MIN's** children until finding one with a value lower than α.
- Can use heuristics to order the nodes to check:
  - Check the highest-value children first for **MAX**.
  - Check the lowest-value children first for **MIN**.
- Good ordering can reduce time complexity to $O(b^{m/2})$, random ordering gives roughly $O(b^{3m/4})$, remember that Minimax is $O(b^m)$.

Adversarial search

Optimal decisions in multiplayer games

- Minimax can be extended to mutiplayer games:

Adversarial search

Optimal decisions in multiplayer games

- Minimax can be extended to mutiplayer games:
  - The single value for each node is replaced with a vector.

Adversarial search

Optimal decisions in multiplayer games

- Minimax can be extended to mutiplayer games:
  - The single value for each node is replaced with a vector.
  - For instance, a vector $\langle v_a, a_b, v_c \rangle$ is associated with each node in a 3-palyer game with players A, B, and C.

Adversarial search

Optimal decisions in multiplayer games

- Minimax can be extended to mutiplayer games:
  - The single value for each node is replaced with a vector.
  - For instance, a vector $\langle v_a, a_b, v_c \rangle$ is associated with each node in a 3-palyer game with players A, B, and C.
  - For a terminal state, UTILITY function returns a vector representing utility of that state from each player's viewpoint.

Adversarial search

Optimal decisions in multiplayer games

- Minimax can be extended to mutiplayer games:
  - The single value for each node is replaced with a vector.
  - For instance, a vector $\langle v_a, a_b, v_c \rangle$ is associated with each node in a 3-palyer game with players A, B, and C.
  - For a terminal state, UTILITY function returns a vector representing utility of that state from each player's viewpoint.
- The multiplayer version of minimax (M-MAX) can be defined as follows:

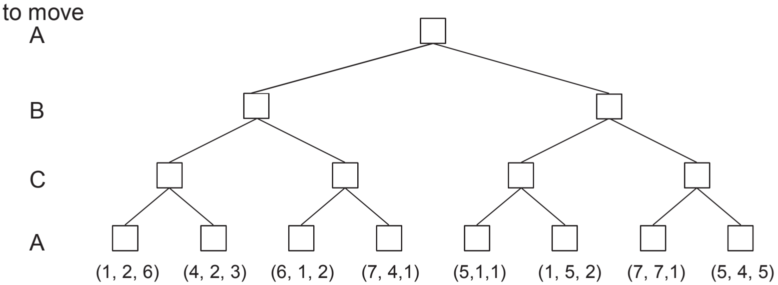
$$\text{M-MAX}(s) = \begin{cases} \text{UTILITY}(s) & \textbf{if } \text{TERMINAL-TEST}(s) \\ max^p_{a \in \text{ACTIONS}(s)} \text{M-MAX}(\text{RESULT}(s,a)) & \textbf{if } \text{PLAYER}(s) = p \end{cases}$$

where max^p is a function that chooses from among a group of vectors the one whose p^{th} element is the maximum.

Adversarial search

Example: multiplayer game tree

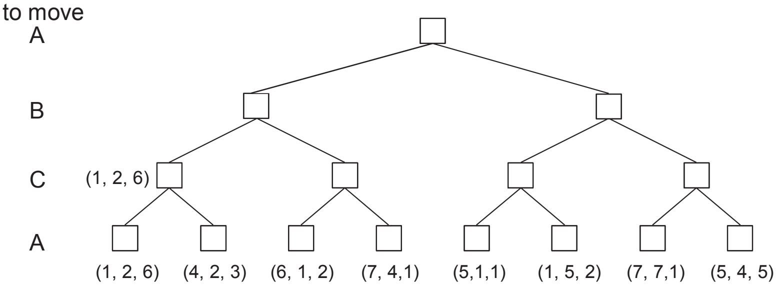
A partial game tree with 3 players (A, B, C).



Adversarial search

Example: multiplayer game tree

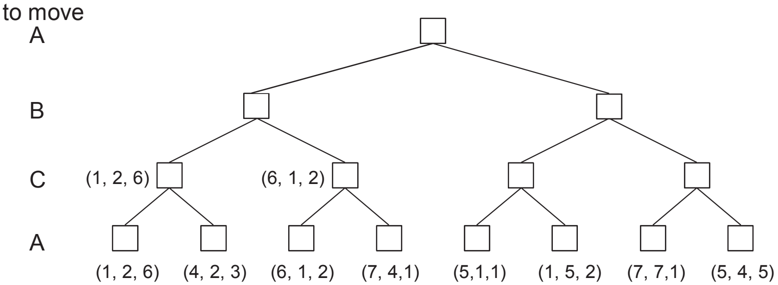
A partial game tree with 3 players (A, B, C).



Adversarial search

Example: multiplayer game tree

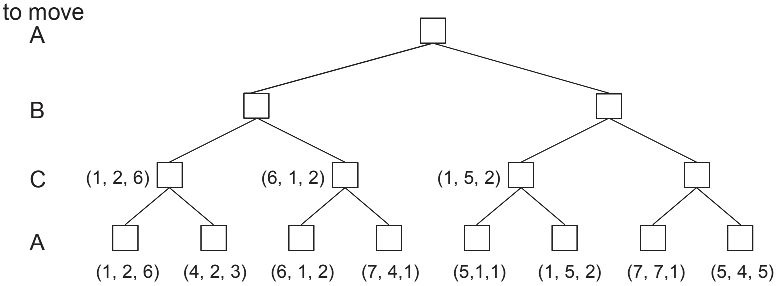
A partial game tree with 3 players (A, B, C).



Adversarial search

Example: multiplayer game tree

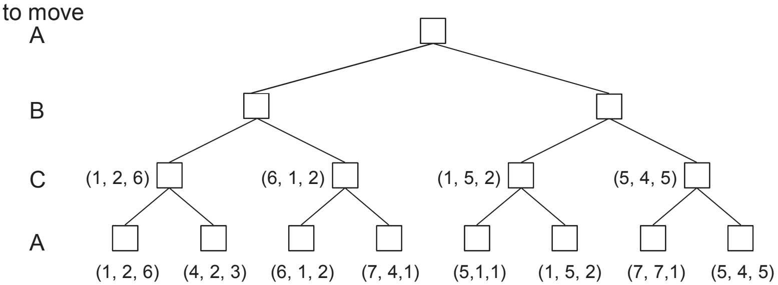
A partial game tree with 3 players (A, B, C).



Adversarial search

Example: multiplayer game tree

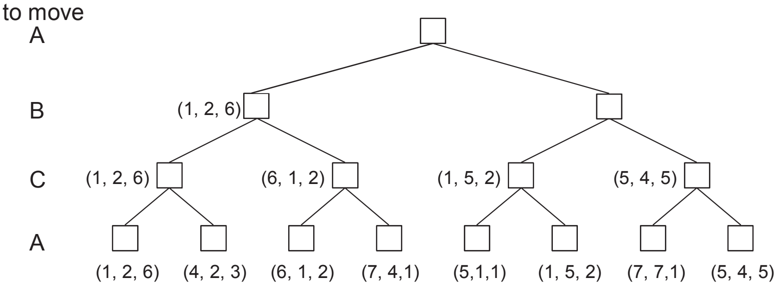
A partial game tree with 3 players (A, B, C).



Adversarial search

Example: multiplayer game tree

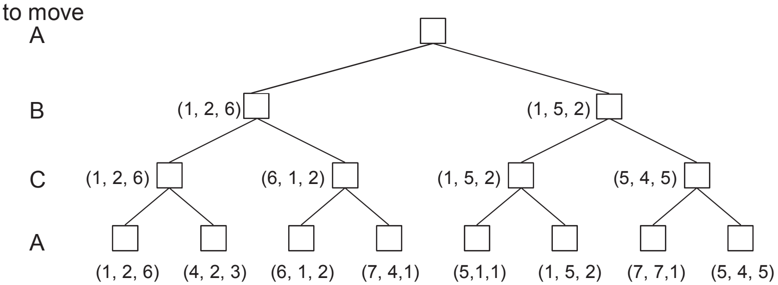
A partial game tree with 3 players (A, B, C).



Adversarial search

Example: multiplayer game tree

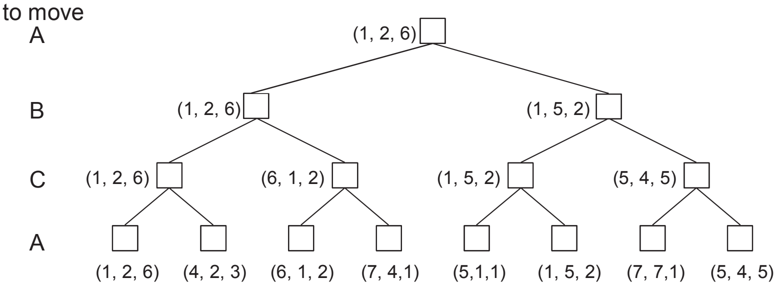
A partial game tree with 3 players (A, B, C).



Adversarial search

Example: multiplayer game tree

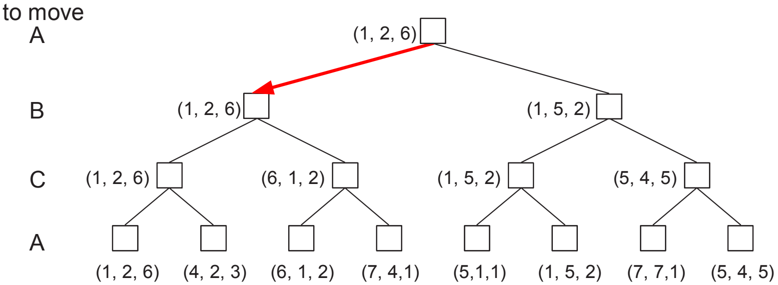
A partial game tree with 3 players (A, B, C).



Adversarial search

Example: multiplayer game tree

A partial game tree with 3 players (A, B, C).



Adversarial search

Imperfect real-time decisions

- How to reach a good decision when time is limited?

Adversarial search

Imperfect real-time decisions

- How to reach a good decision when time is limited?
  - **Cut off** search earlier (before reaching terminal states).

Adversarial search

Imperfect real-time decisions

- How to reach a good decision when time is limited?
  - **Cut off** search earlier (before reaching terminal states).
  - Apply a **heuristic evaluation function** to evaluate the desirability of states (instead of their utility values).

Adversarial search

Imperfect real-time decisions

- How to reach a good decision when time is limited?
  - **Cut off** search earlier (before reaching terminal states).
  - Apply a **heuristic evaluation function** to evaluate the desirability of states (instead of their utility values).
- For inst., suppose we have 100 sec's to make a move in a chess game, and we explore 10^4 nodes/sec's:

Adversarial search

Imperfect real-time decisions

- How to reach a good decision when time is limited?
  - **Cut off** search earlier (before reaching terminal states).
  - Apply a **heuristic evaluation function** to evaluate the desirability of states (instead of their utility values).
- For inst., suppose we have 100 sec's to make a move in a chess game, and we explore 10^4 nodes/sec's:
  - We can explore 10^6 nodes per move $\approx 35^{8/2}$.

Adversarial search

Imperfect real-time decisions

- How to reach a good decision when time is limited?
  - **Cut off** search earlier (before reaching terminal states).
  - Apply a **heuristic evaluation function** to evaluate the desirability of states (instead of their utility values).
- For inst., suppose we have 100 sec's to make a move in a chess game, and we explore 10^4 nodes/sec's:
  - We can explore 10^6 nodes per move $\approx 35^{8/2}$.
  - α-β can reach depth of $8 \Rightarrow$ pretty good chess program.

Adversarial search

Heuristic minimax

- Need to alter minimax (or α-β) in two ways:

Adversarial search

Heuristic minimax

- Need to alter minimax (or α-β) in two ways:
  1. Replace utility function (UTILITY) by heuristic evaluation function (EVAL) $\Rightarrow$ estimates position's utility.

Adversarial search

Heuristic minimax

- Need to alter minimax (or α-β) in two ways:
  1. Replace utility function (UTILITY) by heuristic evaluation function (EVAL) $\Rightarrow$ estimates position's utility.
  2. Replace terminal test (TERMINAL-TEST) by cutoff test (CUTOFF-TEST) $\Rightarrow$ decides when to apply EVAL.

Adversarial search

Heuristic minimax

- Need to alter minimax (or α-β) in two ways:
  1. Replace utility function (UTILITY) by heuristic evaluation function (EVAL) $\Rightarrow$ estimates position's utility.
  2. Replace terminal test (TERMINAL-TEST) by cutoff test (CUTOFF-TEST) $\Rightarrow$ decides when to apply EVAL.
- Modified version of minimax is called **heuristic minimax**:

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \textbf{if } \text{CUTOFF-TEST}(s,d) \\ max_{a \in \text{ACTIONS}(s)}\ \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \textbf{if } \text{PLAYER}(s) = \text{MAX} \\ min_{a \in \text{ACTIONS}(s)}\ \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \textbf{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

Adversarial search

Heuristic minimax

- To implement this modification in the minimax and α-β pruning algorithms, all that has to be done to is to replace the line:

Adversarial search

Heuristic minimax

- To implement this modification in the minimax and α-β pruning algorithms, all that has to be done to is to replace the line:

  if TERMINAL-TEST(*s*) **then return** UTILITY(*s*)

Adversarial search

Heuristic minimax

- To implement this modification in the minimax and α-β pruning algorithms, all that has to be done to is to replace the line:

  if TERMINAL-TEST(*s*) **then return** UTILITY(*s*)

  with the line:

Adversarial search

Heuristic minimax

- To implement this modification in the minimax and α-β pruning algorithms, all that has to be done to is to replace the line:

  if TERMINAL-TEST(*s*) **then return** UTILITY(*s*)

  with the line:

  if CUTOFF-TEST(*s*,*d*) **then return** EVAL(*s*)

Adversarial search

Heuristic minimax

- To implement this modification in the minimax and α-β pruning algorithms, all that has to be done to is to replace the line:

  if TERMINAL-TEST(*s*) **then return** UTILITY(*s*)

  with the line:

  if CUTOFF-TEST(*s*,*d*) **then return** EVAL(*s*)

- Also, some extra bookkeeping is needed to increment the depth on each recursive call of the algorithm.

Adversarial search

Evaluation functions

- Heuristic evaluation functions are functions to estimate the expected utility of a game from a non-leaf node.

Adversarial search

Evaluation functions

- Heuristic evaluation functions are functions to estimate the expected utility of a game from a non-leaf node.
- A good function:

Adversarial search

Evaluation functions

- Heuristic evaluation functions are functions to estimate the expected utility of a game from a non-leaf node.
- A good function:
  - Should order terminal states as true utility values do.

Adversarial search

Evaluation functions

- Heuristic evaluation functions are functions to estimate the expected utility of a game from a non-leaf node.
- A good function:
  - Should order terminal states as true utility values do.
  - Should not take too much time to compute.

Adversarial search

Evaluation functions

- Heuristic evaluation functions are functions to estimate the expected utility of a game from a non-leaf node.
- A good function:
  - Should order terminal states as true utility values do.
  - Should not take too much time to compute.
  - For non-leaf nodes, it should be strongly correlated with the actual chances of winning.

Adversarial search

Evaluation functions

- Heuristic evaluation functions are functions to estimate the expected utility of a game from a non-leaf node.
- A good function:
  - Should order terminal states as true utility values do.
  - Should not take too much time to compute.
  - For non-leaf nodes, it should be strongly correlated with the actual chances of winning.
- Mostly work by calculating various ***features*** of state.

Adversarial search

Evaluation functions

- Heuristic evaluation functions are functions to estimate the expected utility of a game from a non-leaf node.
- A good function:
  - Should order terminal states as true utility values do.
  - Should not take too much time to compute.
  - For non-leaf nodes, it should be strongly correlated with the actual chances of winning.
- Mostly work by calculating various ***features*** of state.
  - Chess example: features could be number of white queens, black queens, white pawns, black pawns, *etc*.

Adversarial search

Evaluation functions

- A simple version is to set it up by calculating a weighted average of the numerical values of all the features.

Adversarial search

Evaluation functions

- A simple version is to set it up by calculating a weighted average of the numerical values of all the features.
- In this case, the evaluation function takes the form of a **weighted linear function** expressed as follows:

Adversarial search

Evaluation functions

- A simple version is to set it up by calculating a weighted average of the numerical values of all the features.
- In this case, the evaluation function takes the form of a **weighted linear function** expressed as follows:

  $\textsc{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

Adversarial search

Evaluation functions

- A simple version is to set it up by calculating a weighted average of the numerical values of all the features.
- In this case, the evaluation function takes the form of a **weighted linear function** expressed as follows:

  $\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

  - Chess example: f_i could be number of each kind of piece on board, w_i could represent strength of that kind.

Adversarial search

Evaluation functions

- A simple version is to set it up by calculating a weighted average of the numerical values of all the features.
- In this case, the evaluation function takes the form of a **weighted linear function** expressed as follows:

  $$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

  - Chess example: f_i could be number of each kind of piece on board, w_i could represent strength of that kind.
- Evaluation function need not return actual expected values as long as the ordering of states is the same.

Adversarial search

Stochastic games

- In some games, there are many unpredictable external events that could affect the course of the game.

Adversarial search

Stochastic games

- In some games, there are many unpredictable external events that could affect the course of the game.
- These games have stochastic environments, and hence, called **stochastic games**.

Adversarial search

Stochastic games

- In some games, there are many unpredictable external events that could affect the course of the game.
- These games have stochastic environments, and hence, called **stochastic games**.
- Typical sources of unpredictability in stochastic games include events such as: roll of a dice, toss of a coin, withdrawal of a card, *etc*.

Adversarial search

Stochastic games

- In some games, there are many unpredictable external events that could affect the course of the game.
- These games have stochastic environments, and hence, called **stochastic games**.
- Typical sources of unpredictability in stochastic games include events such as: roll of a dice, toss of a coin, withdrawal of a card, *etc*.
- Due to such uncertainty, we cannot construct a **complete/exact** game tree for a stochastic game.

Adversarial search

Stochastic games

- In some games, there are many unpredictable external events that could affect the course of the game.
- These games have stochastic environments, and hence, called **stochastic games**.
- Typical sources of unpredictability in stochastic games include events such as: roll of a dice, toss of a coin, withdrawal of a card, *etc*.
- Due to such uncertainty, we cannot construct a **complete/exact** game tree for a stochastic game.
- Minimax strategy becomes inapplicable to such games!

Adversarial search

Stochastic games

- To model such uncertainty in stochastic games, a game tree would include **chance nodes** in addition to MAX and MIN nodes.

Adversarial search

Stochastic games

- To model such uncertainty in stochastic games, a game tree would include **chance nodes** in addition to MAX and MIN nodes.
  - Branches leading from each chance node denote the possible outcomes of an unpredictable event (*e.g.*, dice roll).

Adversarial search

Stochastic games

- To model such uncertainty in stochastic games, a game tree would include **chance nodes** in addition to MAX and MIN nodes.
  - Branches leading from each chance node denote the possible outcomes of an unpredictable event (*e.g.*, dice roll).
  - Each branch is labeled with an **outcome** r and its **probability** $P(r)$.

Adversarial search

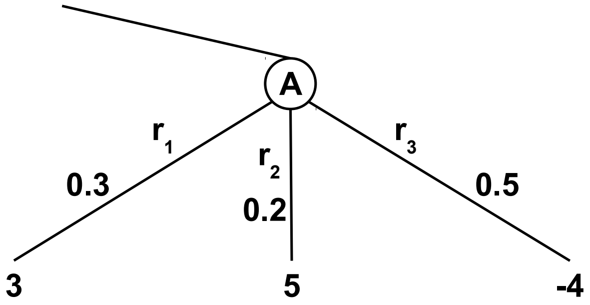
Stochastic games

- To model such uncertainty in stochastic games, a game tree would include **chance nodes** in addition to MAX and MIN nodes.
  - Branches leading from each chance node denote the possible outcomes of an unpredictable event (*e.g.*, dice roll).
  - Each branch is labeled with an **outcome** r and its **probability** $P(r)$.
  - **Average expected utility** of a chance node $= \sum_r P(r) * \textsc{Next}(r)$, where $\textsc{Next}(r)$ represents the average expected utility of the node reached as a consequence of the outcome r.

Adversarial search

Example: chance nodes

The expected utility of the chance node given below
$$= 0.3 * 3 + 0.2 * 5 + 0.5 * (-4) = -0.1.$$



Adversarial search

Stochastic games

- In **single-player stochastic games** (*e.g.*, solitaire and minesweeper), the game tree has only two types of nodes:

Adversarial search

Stochastic games

- In **single-player stochastic games** (*e.g.*, solitaire and minesweeper), the game tree has only two types of nodes:
  1. MAX nodes to model the player's behavior.

Adversarial search

Stochastic games

- In **single-player stochastic games** (*e.g.*, solitaire and minesweeper), the game tree has only two types of nodes:
  1. MAX nodes to model the player's behavior.
  2. Chance nodes to model the stochastic environment.

Adversarial search

Stochastic games

- In **single-player stochastic games** (*e.g.*, solitaire and minesweeper), the game tree has only two types of nodes:
  1. MAX nodes to model the player's behavior.
  2. Chance nodes to model the stochastic environment.
- Average expected utility of a node in such tree is called **expectimax**.

Adversarial search

Stochastic games

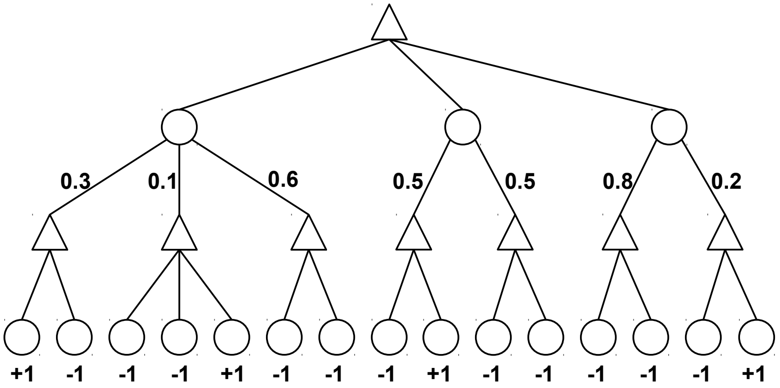
- In **single-player stochastic games** (*e.g.*, solitaire and minesweeper), the game tree has only two types of nodes:
  1. MAX nodes to model the player's behavior.
  2. Chance nodes to model the stochastic environment.
- Average expected utility of a node in such tree is called **expectimax**.
- The expectimax value of a node is defined as follows:

$$\textsc{ExpectiMax}(s) = \begin{cases} \textsc{Utility}(s) & \textbf{if } \textsc{Terminal-Test}(s) \\ max_a\ \textsc{ExpectiMax}(\textsc{Result}(s,a)) & \textbf{if } \textsc{Player}(s) = \textsc{max} \\ \sum_r P(r) * \textsc{ExpectiMax}(\textsc{Result}(s,r)) & \textbf{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$

Adversarial search

Example: expectimax calculation

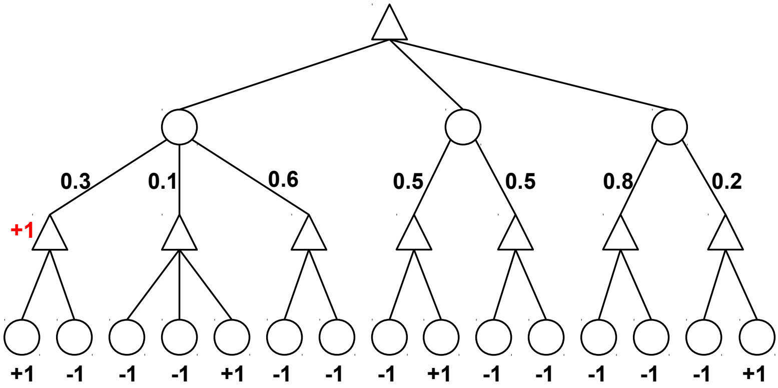
Calculate the expectimax value of the root node for the given game tree.



Adversarial search

Example: expectimax calculation

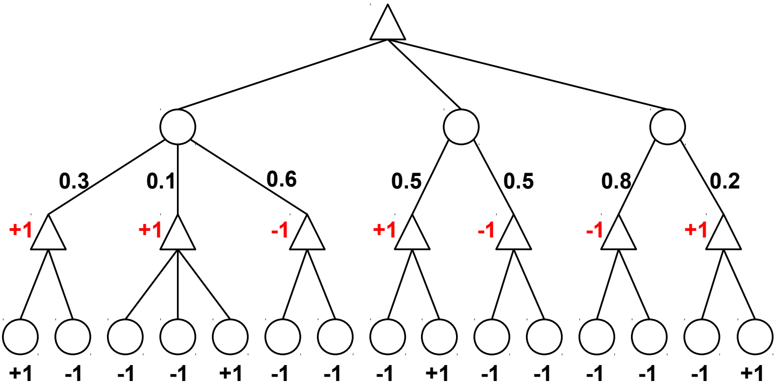
Calculate the expectimax value of the root node for the given game tree.



Adversarial search

Example: expectimax calculation

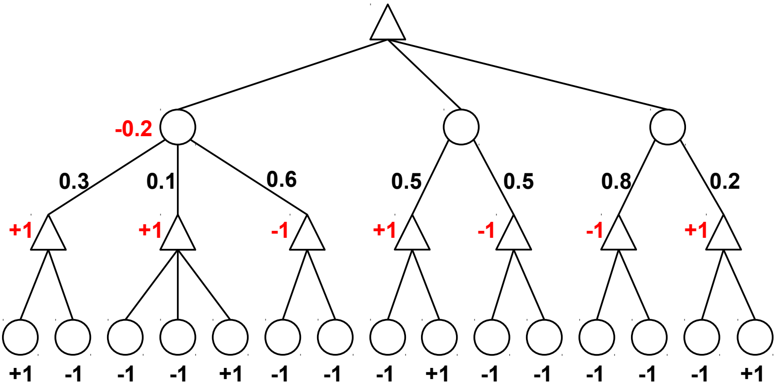
Calculate the expectimax value of the root node for the given game tree.



Adversarial search

Example: expectimax calculation

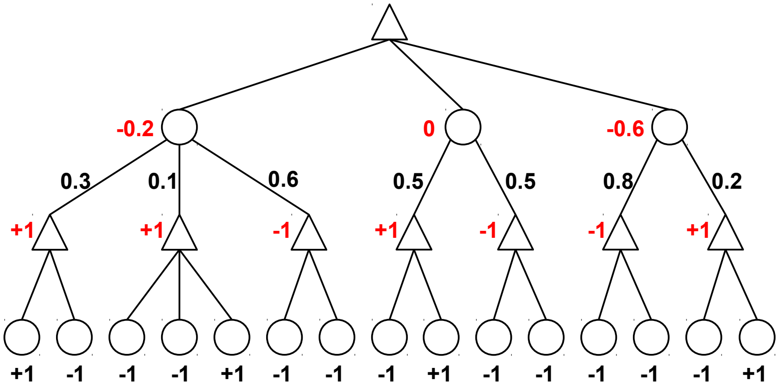
Calculate the expectimax value of the root node for the given game tree.



Adversarial search

Example: expectimax calculation

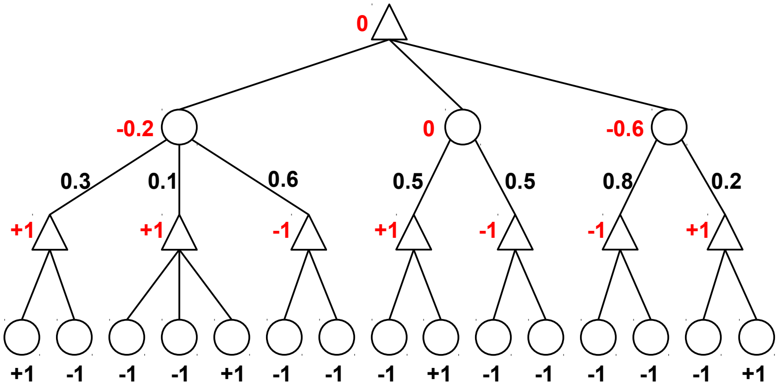
Calculate the expectimax value of the root node for the given game tree.



Adversarial search

Example: expectimax calculation

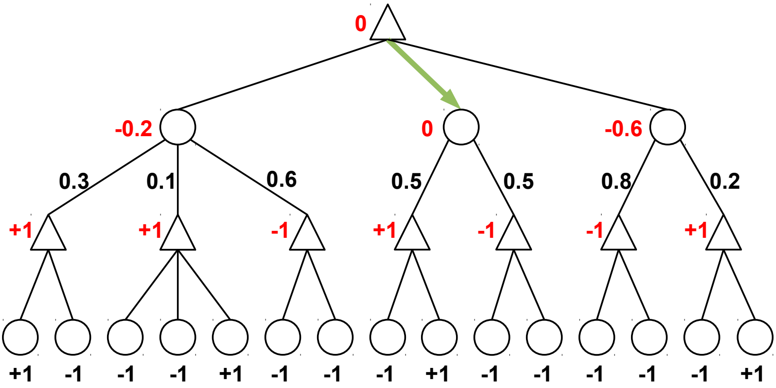
Calculate the expectimax value of the root node for the given game tree.



Adversarial search

Example: expectimax calculation

Calculate the expectimax value of the root node for the given game tree.



Adversarial search

Stochastic games

- Notice that in expectimax, we choose the move that maximizes the **average** outcome of the game.

Adversarial search

Stochastic games

- Notice that in expectimax, we choose the move that maximizes the **average** outcome of the game.
- The actual outcome (*i.e.*, utility value when game ends) may be **more or less** than the expectimax value.

Adversarial search

Stochastic games

- Notice that in expectimax, we choose the move that maximizes the **average** outcome of the game.
- The actual outcome (*i.e.*, utility value when game ends) may be **more or less** than the expectimax value.
- Unlike minimax where the actual outcome is guaranteed to be **no less** than the minimax value.

Adversarial search

Stochastic games

- Notice that in expectimax, we choose the move that maximizes the **average** outcome of the game.
- The actual outcome (*i.e.*, utility value when game ends) may be **more or less** than the expectimax value.
- Unlike minimax where the actual outcome is guaranteed to be **no less** than the minimax value.
- However, expectimax may still be applicable to 2-player deterministic games (*e.g.*, tic-tac-toe and chess) if a probability distribution for opponent's actions is known.

Adversarial search

Stochastic games

- In **two-player stochastic games** (*e.g.*, backgammon), aim is to minimize opponent's chances of wining in presence of a stochastic environment.

Adversarial search

Stochastic games

- In **two-player stochastic games** (*e.g.*, backgammon), aim is to minimize opponent's chances of wining in presence of a stochastic environment.
- The game tree would have three types of nodes:
  1. MAX nodes to model the player's behavior.
  2. MIN nodes to model the opponent's behavior.
  3. Chance nodes to model the stochastic environment.

Adversarial search

Stochastic games

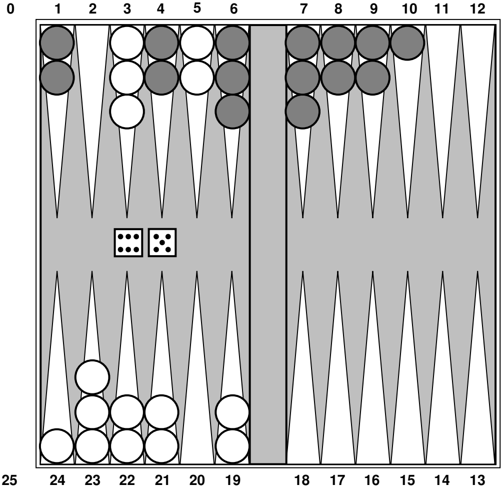
- In **two-player stochastic games** (*e.g.*, backgammon), aim is to minimize opponent's chances of wining in presence of a stochastic environment.
- The game tree would have three types of nodes:
  1. MAX nodes to model the player's behavior.
  2. MIN nodes to model the opponent's behavior.
  3. Chance nodes to model the stochastic environment.
- The average expected value in this case is called **expectiminimax**, and defined as follows:

$$\textsc{ExpectiMiniMax}(s) = \begin{cases} \textsc{Utility}(s) & \textbf{if } \textsc{Terminal-Test}(s) \\ max_a\ \textsc{ExpectiMiniMax}(\textsc{Result}(s,a)) & \textbf{if } \textsc{Player}(s) = \textsc{max} \\ min_a\ \textsc{ExpectiMiniMax}(\textsc{Result}(s,a)) & \textbf{if } \textsc{Player}(s) = \textsc{min} \\ \sum_r P(r) * & \\ \quad \textsc{ExpectiMiniMax}(\textsc{Result}(s,r)) & \textbf{if } \textsc{Player}(s) = \textsc{chance} \end{cases}$$

Adversarial search

Example: Backgammon

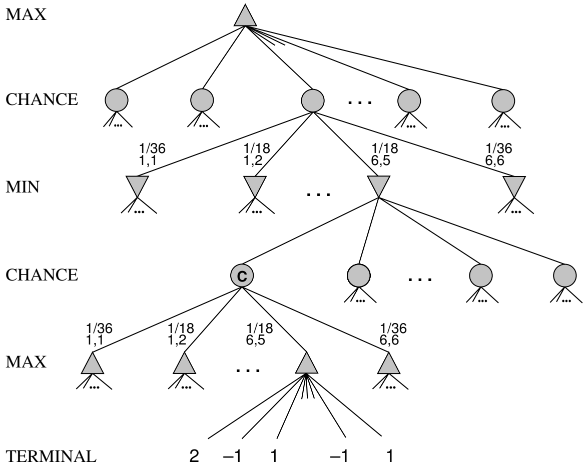
White has rolled 6-5 and has 4 possible moves.



Adversarial search

Example: Backgammon

Schematic game tree for a backgammon position.



Outline

1 Adversarial Search (Continued)

2 Requirements & Reading Material

Requirements

What do I need from you

- When given a certain problem you should be able to:

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Build the game tree up to a given depth.

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Build the game tree up to a given depth.
  - Apply the expectimax and expectiminimax algorithms.

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Build the game tree up to a given depth.
  - Apply the expectimax and expectiminimax algorithms.
  - Propose a good heuristic evaluation function

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - Build the game tree up to a given depth.
  - Apply the expectimax and expectiminimax algorithms.
  - Propose a good heuristic evaluation function
- Answer descriptive questions.

Reading Material

Which parts of the textbook are covered

- Russell-Norvig, Chapters 5:
  - Pages 171 - 173.
  - Pages 177 - 178