भारतीय प्रौद्योगिकी संस्थान मुंबई

Indian Institute of Technology Bombay

Strategic Multi-agent Artificial Intelligence

Day 1: AI and competitive games

Swaprava Nath

Slide preparation acknowledgments: Drashthi Doshi

ज्ञानम् परमम् ध्येयम्

Knowledge is the supreme goal

Contents

▶ Part 1

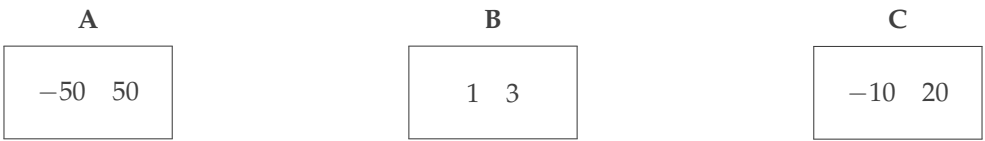
- Sequential move games
- Partial information games
- Subgame perfection

▶ Part 2

- Solutions for 2 player zero sum games
- Simultaneous move games

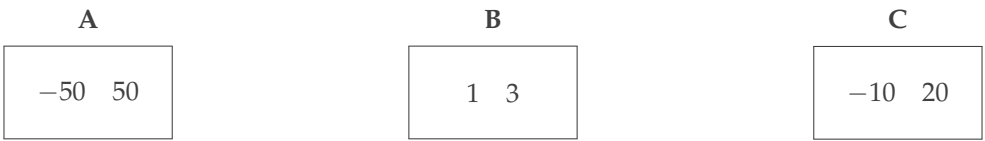
A 2-player game

Consider a two-player game in which there are 3 buckets (A, B, C) and each bucket contains two numbers as shown below.



A 2-player game

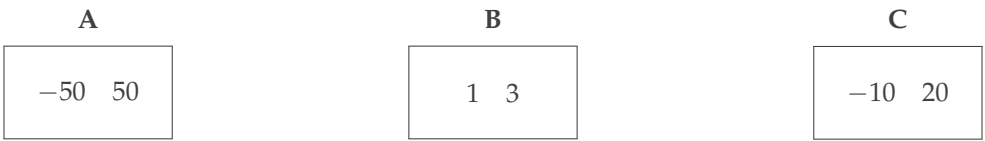
Consider a two-player game in which there are 3 buckets (A, B, C) and each bucket contains two numbers as shown below.



Rules of the game are:

A 2-player game

Consider a two-player game in which there are 3 buckets (A, B, C) and each bucket contains two numbers as shown below.

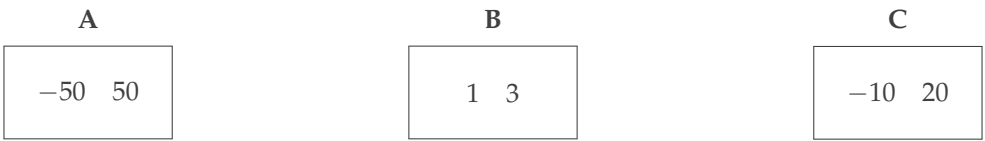


Rules of the game are:

1. Player 1 (agent) picks the bucket

A 2-player game

Consider a two-player game in which there are 3 buckets (A, B, C) and each bucket contains two numbers as shown below.

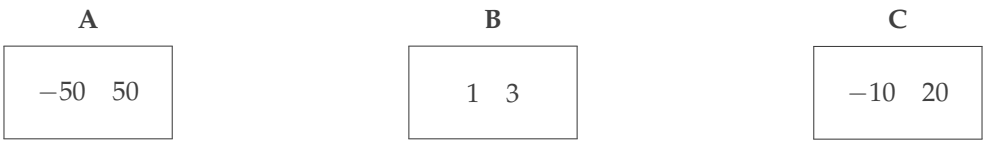


Rules of the game are:

1. Player 1 (agent) picks the bucket
2. Player 2 (opponent) picks a number from the "Selected" Bucket

A 2-player game

Consider a two-player game in which there are 3 buckets (A, B, C) and each bucket contains two numbers as shown below.



Rules of the game are:

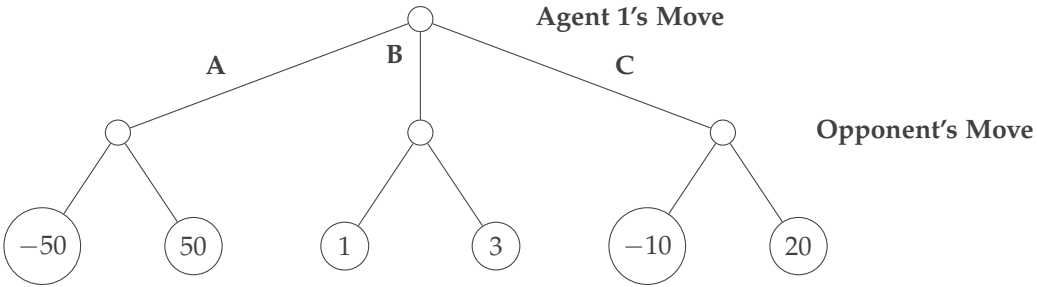
1. Player 1 (agent) picks the bucket
2. Player 2 (opponent) picks a number from the "Selected" Bucket
3. Player 1 (agent) utility will be the number picked by player 2

A 2-player game

Now, let's construct the **game tree**

A 2-player game

Now, let's construct the **game tree**



A 2-player game

Consider the following strategy of the opponent

The opponent is **stochastic**, i.e., the probability of choosing both left and right numbers in a bucket are the same (i.e., $\frac{1}{2}$).

A 2-player game

Consider the following strategy of the opponent

The opponent is **stochastic**, i.e., the probability of choosing both left and right numbers in a bucket are the same (i.e., $\frac{1}{2}$).

- If A is chosen, expected utility $= \frac{-50+50}{2} = 0$.

A 2-player game

Consider the following strategy of the opponent

The opponent is **stochastic**, i.e., the probability of choosing both left and right numbers in a bucket are the same (i.e., $\frac{1}{2}$).

- If A is chosen, expected utility $= \frac{-50+50}{2} = 0$.
- If B is chosen, expected utility $= \frac{1+3}{2} = 2$.

A 2-player game

Consider the following strategy of the opponent

The opponent is **stochastic**, i.e., the probability of choosing both left and right numbers in a bucket are the same (i.e., $\frac{1}{2}$).

- If A is chosen, expected utility $= \frac{-50+50}{2} = 0$.
- If B is chosen, expected utility $= \frac{1+3}{2} = 2$.
- If C is chosen, expected utility $= \frac{-10+20}{2} = 5$.

A 2-player game

Consider the following strategy of the opponent

The opponent is **stochastic**, i.e., the probability of choosing both left and right numbers in a bucket are the same (i.e., $\frac{1}{2}$).

- If A is chosen, expected utility $= \frac{-50+50}{2} = 0$.
- If B is chosen, expected utility $= \frac{1+3}{2} = 2$.
- If C is chosen, expected utility $= \frac{-10+20}{2} = 5$.

So in this case agent picks **bucket C**, to maximize it's utility.

A 2-player game

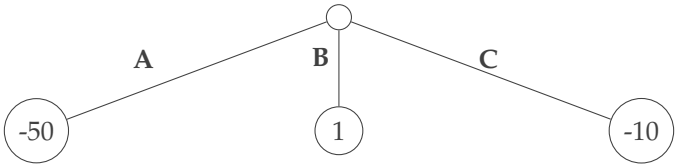
Consider another strategy that the opponent can adopt.

The opponent is a **min player**, i.e., the opponent always chooses the **minimum number** from the bucket.

A 2-player game

Consider another strategy that the opponent can adopt.

The opponent is a **min player**, i.e., the opponent always chooses the **minimum number** from the bucket.

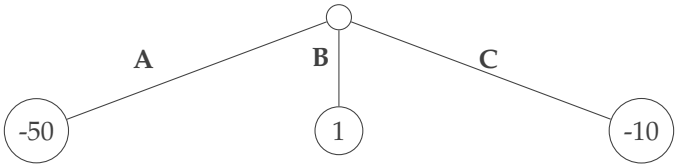


A 2-player game

Consider another strategy that the opponent can adopt.

The opponent is a **min player**, i.e., the opponent always chooses the **minimum number** from the bucket.

In this case,the agent picks **Bucket B**, as $\max\{-50,\ 1, -10\} = 1$.



2-player zero sum games

Lets start with some of the **basic terminologies** for 2-player games:

2-player zero sum games

Lets start with some of the **basic terminologies** for 2-player games:

- **Players:** $\{\text{Agent}, \text{Opponent}\}$

2-player zero sum games

Lets start with some of the **basic terminologies** for 2-player games:

- **Players:** $\{\text{Agent}, \text{Opponent}\}$
- **Starting State:** S_0

2-player zero sum games

Lets start with some of the **basic terminologies** for 2-player games:

- **Players:** $\{\text{Agent}, \text{Opponent}\}$
- **Starting State:** S_0
- **Actions(s):** Possible actions at state 's'

2-player zero sum games

Lets start with some of the **basic terminologies** for 2-player games:

- **Players:** $\{\text{Agent}, \text{Opponent}\}$
- **Starting State:** S_0
- **Actions(s):** Possible actions at state 's'
- **Player(s):** Player who makes the move at state 's'

2-player zero sum games

Lets start with some of the **basic terminologies** for 2-player games:

- **Players:** $\{\text{Agent}, \text{Opponent}\}$
- **Starting State:** S_0
- **Actions(s):** Possible actions at state 's'
- **Player(s):** Player who makes the move at state 's'
- **Successor(s, a):** Resulting state if action a is taken at state 's'

2-player zero sum games

Lets start with some of the **basic terminologies** for 2-player games:

- **Players:** $\{\text{Agent}, \text{Opponent}\}$
- **Starting State:** S_0
- **Actions(s):** Possible actions at state 's'
- **Player(s):** Player who makes the move at state 's'
- **Successor(s, a):** Resulting state if action a is taken at state 's'
- **isEnd(s):** Is state an end state/ Flag to identify terminal state

2-player zero sum games

Lets start with some of the **basic terminologies** for 2-player games:

- **Players:** $\{\text{Agent}, \text{Opponent}\}$
- **Starting State:** S_0
- **Actions(s):** Possible actions at state 's'
- **Player(s):** Player who makes the move at state 's'
- **Successor(s, a):** Resulting state if action a is taken at state 's'
- **isEnd(s):** Is state an end state/ Flag to identify terminal state
- **Utility(s):** Agent's utility at the "end state"

2-player zero sum games

Lets start with some of the **basic terminologies** for 2-player games:

- **Players:** $\{\text{Agent}, \text{Opponent}\}$
- **Starting State:** S_0
- **Actions(s):** Possible actions at state 's'
- **Player(s):** Player who makes the move at state 's'
- **Successor(s, a):** Resulting state if action a is taken at state 's'
- **isEnd(s):** Is state an end state/ Flag to identify terminal state
- **Utility(s):** Agent's utility at the "end state"

2-player zero sum games

Lets start with some of the **basic terminologies** for 2-player games:

- **Players:** $\{\text{Agent}, \text{Opponent}\}$
- **Starting State:** S_0
- **Actions(s):** Possible actions at state 's'
- **Player(s):** Player who makes the move at state 's'
- **Successor(s, a):** Resulting state if action a is taken at state 's'
- **isEnd(s):** Is state an end state/ Flag to identify terminal state
- **Utility(s):** Agent's utility at the "end state"

Note:

i) In Actions(s), Player(s), Successor(s, a), 's' is an intermediate state.

2-player zero sum games

Lets start with some of the **basic terminologies** for 2-player games:

- **Players:** $\{\text{Agent}, \text{Opponent}\}$
- **Starting State:** S_0
- **Actions(s):** Possible actions at state 's'
- **Player(s):** Player who makes the move at state 's'
- **Successor(s, a):** Resulting state if action a is taken at state 's'
- **isEnd(s):** Is state an end state/ Flag to identify terminal state
- **Utility(s):** Agent's utility at the "end state"

Note:

i) In Actions(s), Player(s), Successor(s, a), 's' is an intermediate state.

ii) Utility(s) is not defined in other (intermediate) states.

2-player zero sum games

For example in the game of chess

Example: Chess

2-player zero sum games

For example in the game of chess

Example: Chess

- **Players:** {White, Black}

2-player zero sum games

For example in the game of chess

Example: Chess

- **Players:** $\{\text{White}, \text{Black}\}$
- **State:** A board position (positions of the chess pieces at a given time), denoted s

2-player zero sum games

For example in the game of chess

Example: Chess

- **Players:** {White, Black}
- **State:** A board position (positions of the chess pieces at a given time), denoted s
- **Actions(s):** All legal moves possible by Player(s)

2-player zero sum games

For example in the game of chess

Example: Chess

- **Players:** {White, Black}
- **State:** A board position (positions of the chess pieces at a given time), denoted s
- **Actions(s):** All legal moves possible by Player(s)
- **Player(s):** Player who makes the move at state 's'

2-player zero sum games

For example in the game of chess

Example: Chess

- **Players:** $\{\text{White}, \text{Black}\}$
- **State:** A board position (positions of the chess pieces at a given time), denoted s
- **Actions(s):** All legal moves possible by Player(s)
- **Player(s):** Player who makes the move at state 's'
- **Successor(s, a):** Resulting state if action a is taken at state 's'

2-player zero sum games

For example in the game of chess

Example: Chess

- **Players:** {White, Black}
- **State:** A board position (positions of the chess pieces at a given time), denoted s
- **Actions(s):** All legal moves possible by Player(s)
- **Player(s):** Player who makes the move at state 's'
- **Successor(s, a):** Resulting state if action a is taken at state 's'
- **isEnd(s):** Whether 's' is a checkmate or a draw

2-player zero sum games

For example in the game of chess

Example: Chess

- **Players:** $\{\text{White}, \text{Black}\}$
- **State:** A board position (positions of the chess pieces at a given time), denoted s
- **Actions(s):** All legal moves possible by Player(s)
- **Player(s):** Player who makes the move at state 's'
- **Successor(s, a):** Resulting state if action a is taken at state 's'
- **isEnd(s):** Whether 's' is a checkmate or a draw
- Utility function is defined as

$$\textbf{Utility}(s) = \begin{cases} +M & \text{if white wins} \\ -M & \text{if black wins/white loses} \\ 0 & \text{if it is a draw} \end{cases}$$

Contents

▶ Part 1

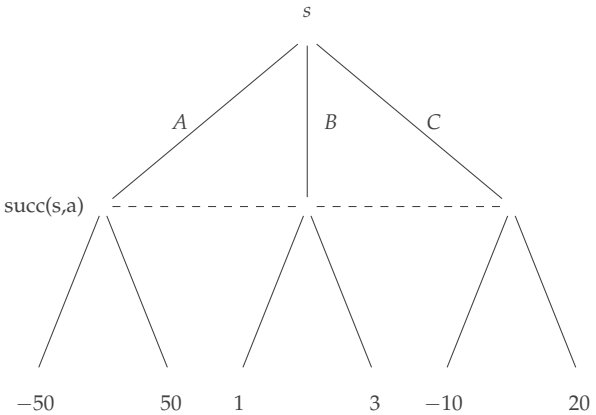
• Sequential move games • Partial information games • Subgame perfection

▶ Part 2

• Solutions for 2 player zero sum games • Simultaneous move games

Games with partial information

Consider the game tree below:



$a \in \{A, B, C\}$

Games with partial information

$$u_{agent}(s) = \begin{cases} \text{utility(s)} & \textit{if } \text{isEnd(s)} \\ \sum_{a \in actions(s)} \Pi_{agent}(s)[a] \ \text{u}_{agent}(\text{successor}(s,a)) & \textit{if } \text{player(s) = agent} \\ \sum_{a \in actions(s)} \Pi_{opponent}(s)[a] \ \text{u}_{agent}(\text{successor}(s,a)) & \textit{if } \text{player(s) = opponent} \end{cases}$$

`Note:` $\Pi_{agent}(s)[a]$ and $\Pi_{opponent}(s)[a]$ are probabilities that the agent and the opponent pick action 'a' respectively.

Contents

▶ Part 1

- Sequential move games
- Partial information games
- Subgame perfection

▶ Part 2

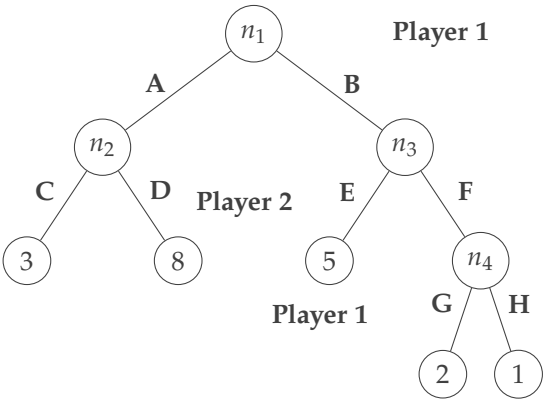
- Solutions for 2 player zero sum games
- Simultaneous move games

Subgame

A **subgame** at ‘S’ is a restriction of the game at the subtree rooted at ’S’ where isEnd(s) is false

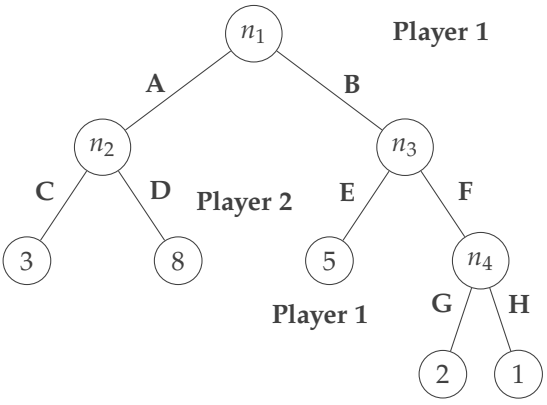
Subgame

A **subgame** at 'S' is a restriction of the game at the subtree rooted at 'S' where isEnd(s) is false



Subgame

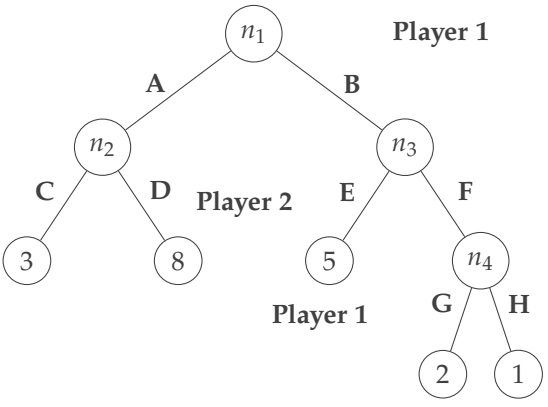
A **subgame** at 'S' is a restriction of the game at the subtree rooted at 'S' where isEnd(s) is false



- **Player 1** is a utility maximizer (**Max Player**) i.e. plays to maximize his utility.
  Hence if player(s) = player 1, the utility of Player 1 changes to,
  $u_1(s) =$
  $\max_{a \in actions(s)} u_1(\text{successor}(s, a))$

Subgame

A **subgame** at 'S' is a restriction of the game at the subtree rooted at 'S' where isEnd(s) is false



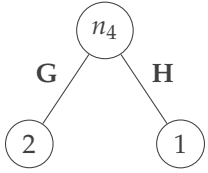
- **Player 1** is a utility maximizer (**Max Player**) i.e. plays to maximize his utility.
  Hence if player(s) = player 1, the utility of Player 1 changes to,
  $u_1(s) = \max_{a \in actions(s)} u_1(\text{successor}(s, a))$
- **Player 2** is the utility minimiser(**Min Player**) i.e. plays to minimize his utility.
  Hence if player(s) = player 2, the utility of Player 1 changes to,
  $u_1(s) = \min_{a \in actions(s)} u_1(\text{successor}(s, a))$

Subgame

Let us look at the **subgame at node** n_4 :

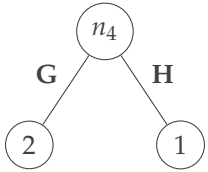
Subgame

Let us look at the **subgame at node** n_4 :



Subgame

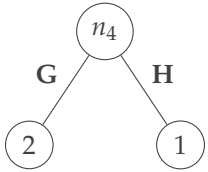
Let us look at the **subgame at node** n_4 :



In this subgame, it is Player 1's turn and he is a **max** player.

Subgame

Let us look at the **subgame at node** n_4 :

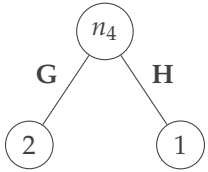


In this subgame, it is Player 1's turn and he is a **max** player.

Player 1 (max player) should pick **G** here so as to get **2** (> 1, that he would have got by choosing **H**).

Subgame

Let us look at the **subgame at node** n_4 :



In this subgame, it is Player 1's turn and he is a **max** player.

Player 1 (max player) should pick **G** here so as to get **2** (> 1, that he would have got by choosing **H**).

We are done solving this subgame.

Subgame

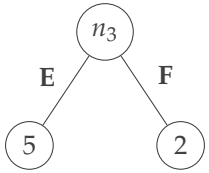
Similarly using the result of the subgame at n_4, we can solve the subgame at n_3, and this way we move to the upper levels.

Subgame

Similarly `using the result of the subgame at` n_4, we can solve the subgame at n_3, and this way we move to the upper levels.
Let us take a look at the **subgame at** n_3 :

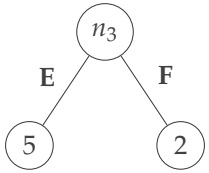
Subgame

Similarly using the result of the subgame at n_4, we can solve the subgame at n_3, and this way we move to the upper levels.
Let us take a look at the **subgame at** n_3 :



Subgame

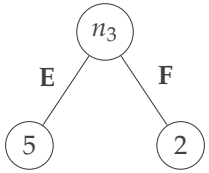
Similarly `using the result of the subgame at` n_4, we can solve the subgame at n_3, and this way we move to the upper levels.
Let us take a look at the **subgame at** n_3 :



In this subgame, it's Player 2's turn and he is a **min** player.

Subgame

Similarly `using the result of the subgame at` n_4, we can solve the subgame at n_3, and this way we move to the upper levels.
Let us take a look at the **subgame at** n_3 :



In this subgame, it's Player 2's turn and he is a **min** player.

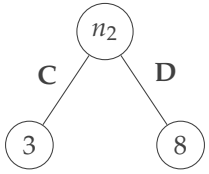
Therefore, he should pick **F** so that player 1 gets **2** (from n_4, instead of getting **5** by picking **E**).

Subgame

Let us look at the **subgame at** n_2 :

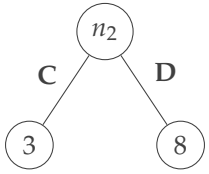
Subgame

Let us look at the **subgame at** n_2 :



Subgame

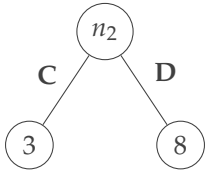
Let us look at the **subgame at** n_2 :



In this subgame, it's Player 2's turn and he is a **min** player.

Subgame

Let us look at the **subgame at** n_2 :

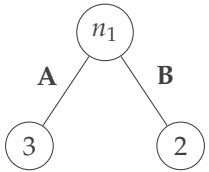


In this subgame, it's Player 2's turn and he is a **min** player.

Therefore, he should pick **C** so that player 1 gets **3** (instead of getting **8** by picking **D**).

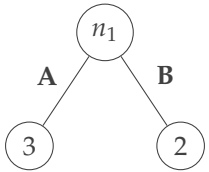
Subgame

Now, let us take a look at the **subgame at** n_1 :



Subgame

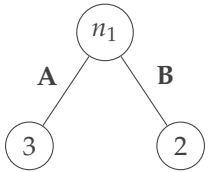
Now, let us take a look at the **subgame at** n_1 :



In this subgame, it's Player 1's turn.

Subgame

Now, let us take a look at the **subgame at** n_1 :

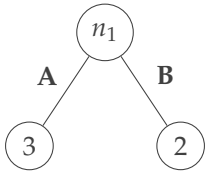


In this subgame, it's Player 1's turn.

Player 1 (max player) should pick A here to get **3** (> 2, that he would have got by choosing **B**).

Subgame

Now, let us take a look at the **subgame at** n_1 :



In this subgame, it's Player 1's turn.

Player 1 (max player) should pick A here to get **3** (> 2, that he would have got by choosing **B**).

Therefore, the **final utility of the Player 1 = 3**

Subgame perfection

Subgame Perfect Equilibrium is an equilibrium at every subgame/subtree.

Subgame perfection

Subgame Perfect Equilibrium is an equilibrium at every subgame/subtree.

The summary of solving the above game is:

Subgame perfection

Subgame Perfect Equilibrium is an equilibrium at every subgame/subtree.

The summary of solving the above game is:

- Player 1 will choose **H** in the subgame at n_4

Subgame perfection

Subgame Perfect Equilibrium is an equilibrium at every subgame/subtree.

The summary of solving the above game is:

- Player 1 will choose **H** in the subgame at n_4
- Player 2 will choose **F** in the subgame at n_3

Subgame perfection

Subgame Perfect Equilibrium is an equilibrium at every subgame/subtree.

The summary of solving the above game is:

- Player 1 will choose **H** in the subgame at n_4
- Player 2 will choose **F** in the subgame at n_3
- player 2 will choose **C** in the subgame at n_2

Subgame perfection

Subgame Perfect Equilibrium is an equilibrium at every subgame/subtree.

The summary of solving the above game is:

- Player 1 will choose **H** in the subgame at n_4
- Player 2 will choose **F** in the subgame at n_3
- player 2 will choose **C** in the subgame at n_2
- player 1 will choose **A** in the subgame at n_1

Subgame perfection

Subgame Perfect Equilibrium is an equilibrium at every subgame/subtree.

The summary of solving the above game is:

- Player 1 will choose **H** in the subgame at n_4
- Player 2 will choose **F** in the subgame at n_3
- player 2 will choose **C** in the subgame at n_2
- player 1 will choose **A** in the subgame at n_1

Subgame perfection

Subgame Perfect Equilibrium is an equilibrium at every subgame/subtree.

The summary of solving the above game is:

- Player 1 will choose **H** in the subgame at n_4
- Player 2 will choose **F** in the subgame at n_3
- player 2 will choose **C** in the subgame at n_2
- player 1 will choose **A** in the subgame at n_1

The algorithm described in the above example is **Backward Induction**.

Feasibility of backward induction

We can apply Backward induction on small games like Tic-tac-toe.

Feasibility of backward induction

We can apply Backward induction on small games like Tic-tac-toe.

But can we apply it to Chess, Go, Checkers, etc.?

Feasibility of backward induction

We can apply Backward induction on small games like Tic-tac-toe.

But can we apply it to Chess, Go, Checkers, etc.?

We can, but the game tree is huge.

Feasibility of backward induction

We can apply Backward induction on small games like Tic-tac-toe.

But can we apply it to Chess, Go, Checkers, etc.?

We can, but the game tree is huge.

- Checkers game tree $\sim 10^{20}$ nodes

Feasibility of backward induction

We can apply Backward induction on small games like Tic-tac-toe.

But can we apply it to Chess, Go, Checkers, etc.?

We can, but the game tree is huge.

- Checkers game tree $\sim 10^{20}$ nodes
- Chess game tree $\sim 10^{40}$ nodes

Feasibility of backward induction

We can apply Backward induction on small games like Tic-tac-toe.

But can we apply it to Chess, Go, Checkers, etc.?

We can, but the game tree is huge.

- Checkers game tree $\sim 10^{20}$ nodes
- Chess game tree $\sim 10^{40}$ nodes
- Go game tree $\sim 10^{170}$ nodes

Feasibility of backward induction

We can apply Backward induction on small games like Tic-tac-toe.

But can we apply it to Chess, Go, Checkers, etc.?

We can, but the game tree is huge.

- Checkers game tree $\sim 10^{20}$ nodes
- Chess game tree $\sim 10^{40}$ nodes
- Go game tree $\sim 10^{170}$ nodes

Feasibility of backward induction

We can apply Backward induction on small games like Tic-tac-toe.

But can we apply it to Chess, Go, Checkers, etc.?

We can, but the game tree is huge.

- Checkers game tree $\sim 10^{20}$ nodes
- Chess game tree $\sim 10^{40}$ nodes
- Go game tree $\sim 10^{170}$ nodes

Checkers was solved in 2007 after 18 years of computation and the optimal solution was a **draw**.

Contents

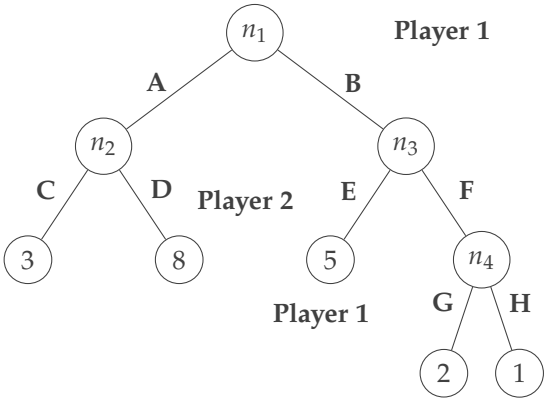
▶ Part 1

• Sequential move games • Partial information games • Subgame perfection

▶ Part 2

• Solutions for 2 player zero sum games • Simultaneous move games

Recall: 2-player zero sum game



Depth Limited Search

- Depth-limited search algorithm explores as far as possible along each branch before backtracking.

Depth Limited Search

- Depth-limited search algorithm explores as far as possible along each branch before backtracking.
- A **depth limit** is imposed and when the algorithm reaches the specified depth limit, it stops exploring that branch further and backtracks to explore other branches

Depth Limited Search

- Depth-limited search algorithm explores as far as possible along each branch before backtracking.
- A **depth limit** is imposed and when the algorithm reaches the specified depth limit, it stops exploring that branch further and backtracks to explore other branches
- This prevents the algorithms from getting stuck in deep paths and allows it to handle very deep trees effectively, reducing the computational time.

Depth Limited Search

- Depth-limited search algorithm explores as far as possible along each branch before backtracking.
- A **depth limit** is imposed and when the algorithm reaches the specified depth limit, it stops exploring that branch further and backtracks to explore other branches
- This prevents the algorithms from getting stuck in deep paths and allows it to handle very deep trees effectively, reducing the computational time.

Depth Limited Search

- Depth-limited search algorithm explores as far as possible along each branch before backtracking.
- A **depth limit** is imposed and when the algorithm reaches the specified depth limit, it stops exploring that branch further and backtracks to explore other branches
- This prevents the algorithms from getting stuck in deep paths and allows it to handle very deep trees effectively, reducing the computational time.

The utility of the agent at node s and depth d can be written as

$$u_{agent}(s,d) = \begin{cases} utility(s) & \text{if } isEnd(s) \\ eval(s) & \text{if } d = 0 \\ \max\limits_{a \in actions(s)} \{u_{agent}(succ(s,a), d-1)\} & \text{if } player(s) = agent \\ \min\limits_{a \in actions(s)} \{u_{agent}(succ(s,a), d-1)\} & \text{if } player(s) = opponent \end{cases}$$

Depth Limited Search

Where **eval**(**s**) is a domain specific function denoting the possible utility to the agent.

Depth Limited Search

Where **eval**(**s**) is a domain specific function denoting the possible utility to the agent.

For example in the game of chess, it can be written as

$$eval(s) = army + mobility + king's\ safety + \ldots \tag{1}$$

Depth Limited Search

Where **eval**(**s**) is a domain specific function denoting the possible utility to the agent.

For example in the game of chess, it can be written as

$$eval(s) = army + mobility + king's\ safety + \ldots \tag{1}$$

The contribution of army towards the utility can be represented by the following equation

$$army = 10^{100}(K - K') + q(Q - Q') + r(R - R') + \ldots \tag{2}$$

Depth Limited Search

Where **eval**(**s**) is a domain specific function denoting the possible utility to the agent.

For example in the game of chess, it can be written as

$$eval(s) = army + mobility + king's\ safety + \ldots \tag{1}$$

The contribution of army towards the utility can be represented by the following equation

$$army = 10^{100}(K - K') + q(Q - Q') + r(R - R') + \ldots \tag{2}$$

Here K, Q, R and K', Q', R' are the number of king, queen and rooks that the agent and the opponent respectively have.

Pruning

Another method is the α, β pruning method. It aims to reduce the number of nodes evaluated by eliminating irrelevant branches.

Pruning

Another method is the α, β pruning method. It aims to reduce the number of nodes evaluated by eliminating irrelevant branches.

- We initialise α with -∞ and β with +∞ in all action nodes.

Pruning

Another method is the α, β pruning method. It aims to reduce the number of nodes evaluated by eliminating irrelevant branches.

- We initialise α with -∞ and β with +∞ in all action nodes.
- we mark subtrees to be pruned based on the values of α and β. If $\alpha \geqslant \beta$ at a node, we can prune the subtree rooted at that node as it won't affect the final decision.

Pruning

Another method is the α, β pruning method. It aims to reduce the number of nodes evaluated by eliminating irrelevant branches.

- We initialise α with -∞ and β with +∞ in all action nodes.
- we mark subtrees to be pruned based on the values of α and β. If $\alpha \geqslant \beta$ at a node, we can prune the subtree rooted at that node as it won't affect the final decision.

Pruning

Another method is the α, β pruning method. It aims to reduce the number of nodes evaluated by eliminating irrelevant branches.

- We initialise α with -∞ and β with +∞ in all action nodes.
- we mark subtrees to be pruned based on the values of α and β. If $\alpha \geqslant \beta$ at a node, we can prune the subtree rooted at that node as it won't affect the final decision.
- — For 'Max' nodes: $\alpha = \max(\alpha, \text{value})$

Pruning

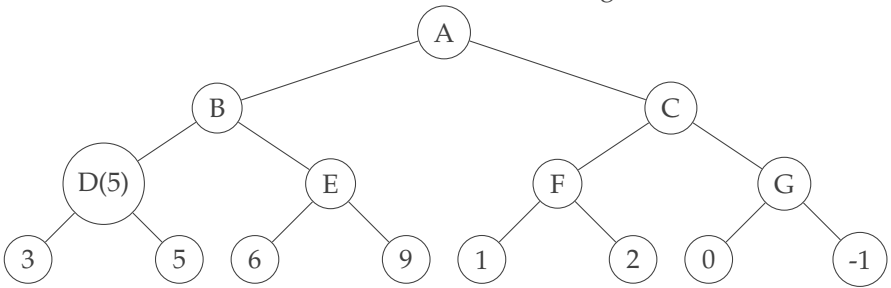
Another method is the α, β pruning method. It aims to reduce the number of nodes evaluated by eliminating irrelevant branches.

- We initialise α with -∞ and β with +∞ in all action nodes.
- we mark subtrees to be pruned based on the values of α and β. If $\alpha \geqslant \beta$ at a node, we can prune the subtree rooted at that node as it won't affect the final decision.
- — For 'Max' nodes: $\alpha = \max(\alpha, \text{value})$
  — For 'Min' nodes: $\beta = \min(\beta, \text{value})$

An example of α, β pruning

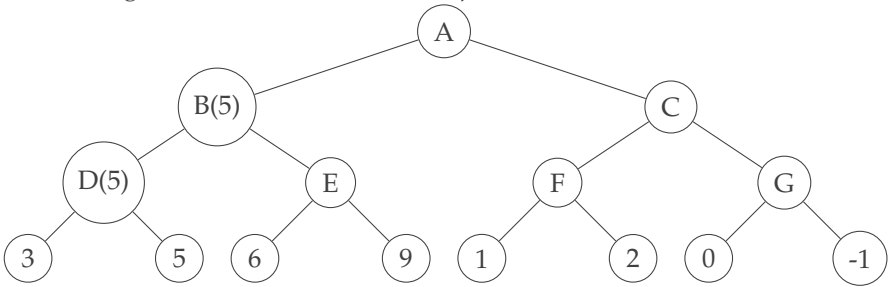
Considering the given binary tree, these are the following steps :-

1. Start with the subtree rooted at D. After transfering values from its two leaves, α=5, β=∞.



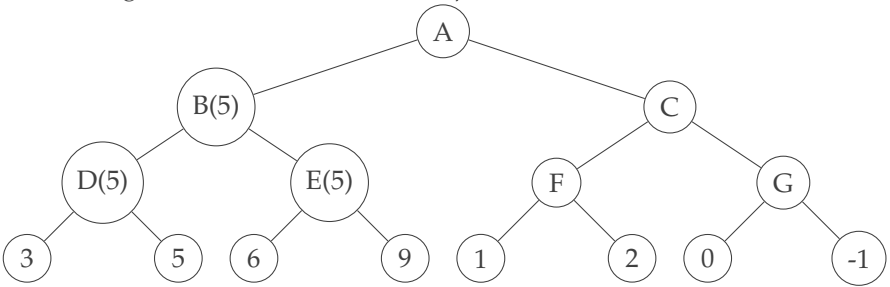
An example of α, β pruning

1. Transfering the value of D to B, α=-∞, β=5.



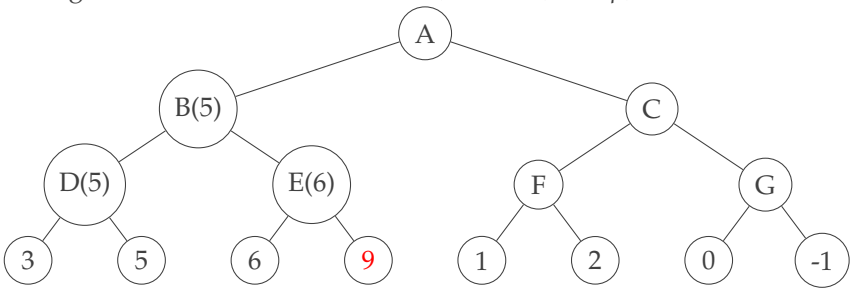
An example of α, β pruning

1. Transfering the value of B to E, α=-∞, β=5.



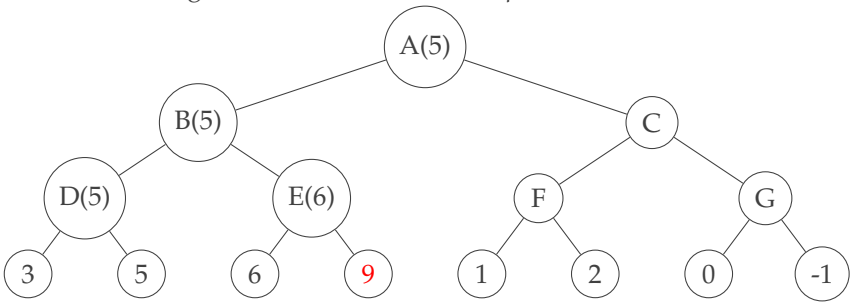
An example of α, β pruning

1. Taking the value from first child of E, α=6. So, $\alpha > \beta$, hence the rest branch of E is ignored.



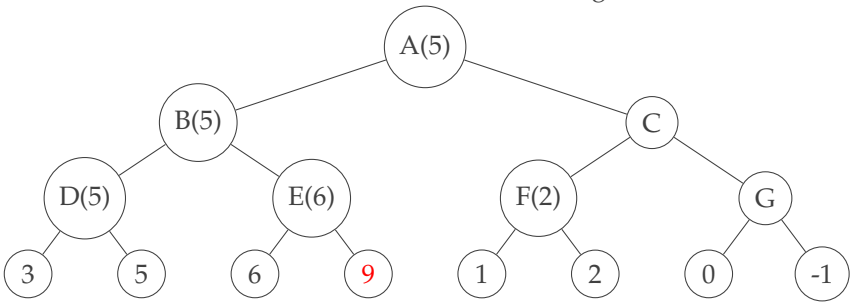
An example of α, β pruning

1. After transfering values from B to A, α=5, β=∞.



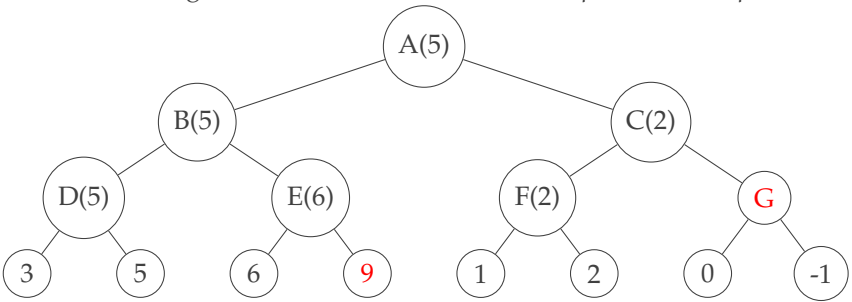
An example of α, β pruning

1. Start with the subtree rooted at F. After transfering values from its two leaves, α=2, β=∞.



An example of α, β pruning

- After transfering values from A and F to C, α=5, β=2. So, $\alpha > \beta$. Hence, G subtree is ignored.



Likewise, the whole game can be solved in a computionally less expensive way by pruning.

Contents

▶ Part 1

• Sequential move games • Partial information games • Subgame perfection

▶ Part 2

• Solutions for 2 player zero sum games • Simultaneous move games

Simultaneous move games

Consider football where the shooter makes a decision on whether to aim towards the left, right, or center of the goal. At the same time the goalkeeper must decide whether to block the left, right or center of the goal

Simultaneous move games

Consider football where the shooter makes a decision on whether to aim towards the left, right, or center of the goal. At the same time the goalkeeper must decide whether to block the left, right or center of the goal

Shooter \ Football	L	C	R
L	−1	1	1
C	1	−1	1
R	1	1	−1

Simultaneous move games

Consider football where the shooter makes a decision on whether to aim towards the left, right, or center of the goal. At the same time the goalkeeper must decide whether to block the left, right or center of the goal

Shooter \ Football	L	C	R
L	−1	1	1
C	1	−1	1
R	1	1	−1

This is an example of a **2-player zero sum simultaneous move game** and is called a **Matrix Game**.

Equilibrium

Definition

Equilibrium : A tuple of actions from which no player **gains** (a strict gain) by a **unilateral deviation**. A unilateral deviation from a tuple of actions $(a_1, a_2, \ldots, a_n)$ is a tuple of actions of the form $(b_1, b_2, \ldots, b_n)$ where $a_i = b_i$, $\forall i \neq j$ for some j ie, the actions of exactly one player j differs from the previous set.

Equilibrium

Definition

Equilibrium : A tuple of actions from which no player **gains** (a strict gain) by a **unilateral deviation**. A unilateral deviation from a tuple of actions $(a_1, a_2, \ldots, a_n)$ is a tuple of actions of the form $(b_1, b_2, \ldots, b_n)$ where $a_i = b_i$, $\forall i \neq j$ for some j ie, the actions of exactly one player j differs from the previous set.

Observe that for the game described above, no such equilibrium exists. This can be shown by analysing all possible pairs of actions.

Equilibrium

Definition

Equilibrium : A tuple of actions from which no player **gains** (a strict gain) by a **unilateral deviation**. A unilateral deviation from a tuple of actions $(a_1, a_2, \ldots, a_n)$ is a tuple of actions of the form $(b_1, b_2, \ldots, b_n)$ where $a_i = b_i$, $\forall i \neq j$ for some j ie, the actions of exactly one player j differs from the previous set.

Observe that for the game described above, no such equilibrium exists. This can be shown by analysing all possible pairs of actions.

1. (L,L) : Player 1 gains additonal utility by (C, L) or (R, L). Similarly for (C, C) and (R, R), player 1 stands to gain by unilaterally deviating.

Equilibrium

Definition

Equilibrium : A tuple of actions from which no player **gains** (a strict gain) by a **unilateral deviation**. A unilateral deviation from a tuple of actions $(a_1, a_2, \ldots, a_n)$ is a tuple of actions of the form $(b_1, b_2, \ldots, b_n)$ where $a_i = b_i$, $\forall i \neq j$ for some j ie, the actions of exactly one player j differs from the previous set.

Observe that for the game described above, no such equilibrium exists. This can be shown by analysing all possible pairs of actions.

1. (L,L) : Player 1 gains additonal utility by (C, L) or (R, L). Similarly for (C, C) and (R, R), player 1 stands to gain by unilaterally deviating.
2. (L,R) : Player 2 gains additonal utility by (L, L). Similarly for any tuple (A, B), where $A \neq B$, if player 2 deviates and chooses the action A, there is a utility gain.

Equilibrium

Definition

Equilibrium : A tuple of actions from which no player **gains** (a strict gain) by a **unilateral deviation**. A unilateral deviation from a tuple of actions $(a_1, a_2, \ldots, a_n)$ is a tuple of actions of the form $(b_1, b_2, \ldots, b_n)$ where $a_i = b_i$, $\forall i \neq j$ for some j ie, the actions of exactly one player j differs from the previous set.

Observe that for the game described above, no such equilibrium exists. This can be shown by analysing all possible pairs of actions.

1. (L,L) : Player 1 gains additonal utility by (C, L) or (R, L). Similarly for (C, C) and (R, R), player 1 stands to gain by unilaterally deviating.
2. (L,R) : Player 2 gains additonal utility by (L, L). Similarly for any tuple (A, B), where $A \neq B$, if player 2 deviates and chooses the action A, there is a utility gain.

Equilibrium

Definition

Equilibrium : A tuple of actions from which no player **gains** (a strict gain) by a **unilateral deviation**. A unilateral deviation from a tuple of actions $(a_1, a_2, \ldots, a_n)$ is a tuple of actions of the form $(b_1, b_2, \ldots, b_n)$ where $a_i = b_i$, $\forall i \neq j$ for some j ie, the actions of exactly one player j differs from the previous set.

Observe that for the game described above, no such equilibrium exists. This can be shown by analysing all possible pairs of actions.

1. (L,L) : Player 1 gains additonal utility by (C, L) or (R, L). Similarly for (C, C) and (R, R), player 1 stands to gain by unilaterally deviating.
2. (L,R) : Player 2 gains additonal utility by (L, L). Similarly for any tuple (A, B), where $A \neq B$, if player 2 deviates and chooses the action A, there is a utility gain.

Therefore there exists no such equilibrium for the game.

Equilibrium

Consider a modification of the above game where the shooter can score by aiming to the left regardless of whether the goal keeper defends the left side or not.

Equilibrium

Consider a modification of the above game where the shooter can score by aiming to the left regardless of whether the goal keeper defends the left side or not.

Shooter \ Football	L	C	R
L	1	1	1
C	1	−1	1
R	1	1	−1

Equilibrium

Consider a modification of the above game where the shooter can score by aiming to the left regardless of whether the goal keeper defends the left side or not.

Shooter \ Football	L	C	R
L	1	1	1
C	1	−1	1
R	1	1	−1

In this case , (L, L) is a simultaneous move equilibrium for game 2 because, the agent (player 1) will not gain any extra utility from (C, L) or (R, L) and similarly the opponent (player 2) will not gain any extra utility from (L, C) or (L, R).

Max-min criteria for equilibrium

The agent is a max player and the opponent is a min player.

Max-min criteria for equilibrium

The agent is a max player and the opponent is a min player.

The agent tries to maximise his utility at each step and the opponent tries to minimise the agent's utility thereby maximising his own utility. :

Max-min criteria for equilibrium

The agent is a max player and the opponent is a min player.

The agent tries to maximise his utility at each step and the opponent tries to minimise the agent's utility thereby maximising his own utility. :

1. For every row of the matrix calculate the minimum utility possible. The value corresponding to row i denotes the least utility for the agent upon performing move i. Maximize over these values. This quantity is denoted as $\max_{s1} \min_{s2} u(s1, s2)$.

Max-min criteria for equilibrium

The agent is a max player and the opponent is a min player.

The agent tries to maximise his utility at each step and the opponent tries to minimise the agent's utility thereby maximising his own utility. :

1. For every row of the matrix calculate the minimum utility possible. The value corresponding to row i denotes the least utility for the agent upon performing move i. Maximize over these values. This quantity is denoted as $\max_{s1} \min_{s2} u(s1, s2)$.
2. Similarly, for every column of the matrix calculate the maximum utility possible. The values corresponding to column j denotes the most utility the agent can obtain if the opponent performs move j. Minimize over these values. This quantity is denoted as $\min_{s2} \max_{s1} u(s1, s2)$.

Max-min criteria for equilibrium

Lemma

$$\max_{s_1} \min_{s_2} u(s_1, s_2) \leqslant \min_{s_2} \max_{s_1} u(s_1, s_2).$$

Max-min criteria for equilibrium

Lemma

$$\max_{s_1} \min_{s_2} u(s_1, s_2) \leqslant \min_{s_2} \max_{s_1} u(s_1, s_2).$$

If for some game, these two quantities equal each other, we have an equilibrium and the equilibrium point is called the saddle point.

Max-min criteria for equilibrium

Proof.

For any $(s1, s2)$,

Max-min criteria for equilibrium

Proof.

For any $(s1, s2)$,

$$u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2) \text{ and}$$
$$\min_{s_2} u(s_1, s_2) \leqslant u(s_1, s_2)$$
$$\Rightarrow \min_{s_2} u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$

Since the previous inequality is true $\forall s_1$, it is also true for s_1^* which maximises $\min_{s_2} u(s_1, s_2)$.

Therefore we have,

$$\min_{s_2} u(s_1^*, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$
$$\Rightarrow \max_{s_1} \min_{s_2} u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$

Max-min criteria for equilibrium

Proof.

For any $(s1, s2)$,

$$u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2) \text{ and}$$

$$\Rightarrow \min_{s_2} u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$

Since the previous inequality is true $\forall s_1$, it is also true for s_1^* which maximises $\min_{s_2} u(s_1, s_2)$.
Therefore we have,

$$\min_{s_2} u(s_1^*, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$
$$\Rightarrow \max_{s_1} \min_{s_2} u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$

Max-min criteria for equilibrium

Proof.

For any $(s1, s2)$,

$$u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2) \text{ and}$$
$$\min_{s_2} u(s_1, s_2) \leqslant u(s_1, s_2)$$
$$\Rightarrow \min_{s_2} u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$

Max-min criteria for equilibrium

Proof.

For any $(s1, s2)$,

$$u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2) \text{ and}$$
$$\min_{s_2} u(s_1, s_2) \leqslant u(s_1, s_2)$$
$$\Rightarrow \min_{s_2} u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$

Since the previous inequality is true $\forall s_1$, it is also true for s_1^* which maximises $\min_{s_2} u(s_1, s_2)$.

Max-min criteria for equilibrium

Proof.

For any $(s1, s2)$,

$$u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2) \text{ and}$$
$$\min_{s_2} u(s_1, s_2) \leqslant u(s_1, s_2)$$
$$\Rightarrow \min_{s_2} u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$

Since the previous inequality is true $\forall s_1$, it is also true for s_1^* which maximises $\min_{s_2} u(s_1, s_2)$.

Therefore we have,

$$\min_{s_2} u(s_1^*, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$
$$\Rightarrow \max_{s_1} \min_{s_2} u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$

Max-min criteria for equilibrium

Proof.

For any $(s1, s2)$,

$$u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2) \text{ and}$$
$$\min_{s_2} u(s_1, s_2) \leqslant u(s_1, s_2)$$
$$\Rightarrow \min_{s_2} u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$

Since the previous inequality is true $\forall s_1$, it is also true for s_1^* which maximises $\min_{s_2} u(s_1, s_2)$.

Therefore we have,

$$\min_{s_2} u(s_1^*, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$
$$\Rightarrow \max_{s_1} \min_{s_2} u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2)$$

Max-min criteria for equilibrium

Proof.

Similarly, since the previous inequality is true $\forall s_2$, it is also true for s_2^* which minimises $\max_{s_1} u(s_1, s_2)$. Therefore we have,

$$\max_{s_1} \min_{s_2} u(s_1, s_2) \leqslant \max_{s_1} u(s_1, s_2^*)$$
$$\Rightarrow \max_{s_1} \min_{s_2} u(s_1, s_2) \leqslant \min_{s_2} \max_{s_1} u(s_1, s_2)$$

And we are done with the proof. □

भारतीय प्रौद्योगिकी संस्थान मुंबई

Indian Institute of Technology Bombay