Game Playing
Fundamentals of AI (AE1FAI)

Slides created by Dr Rong Qu & Dr Qian zhang
GAN part slides is adopted from cs231n
Instructor: Dr Qian Zhang

March 21 2023, spring

SOLVE PROBLEMS BY SEARCHING

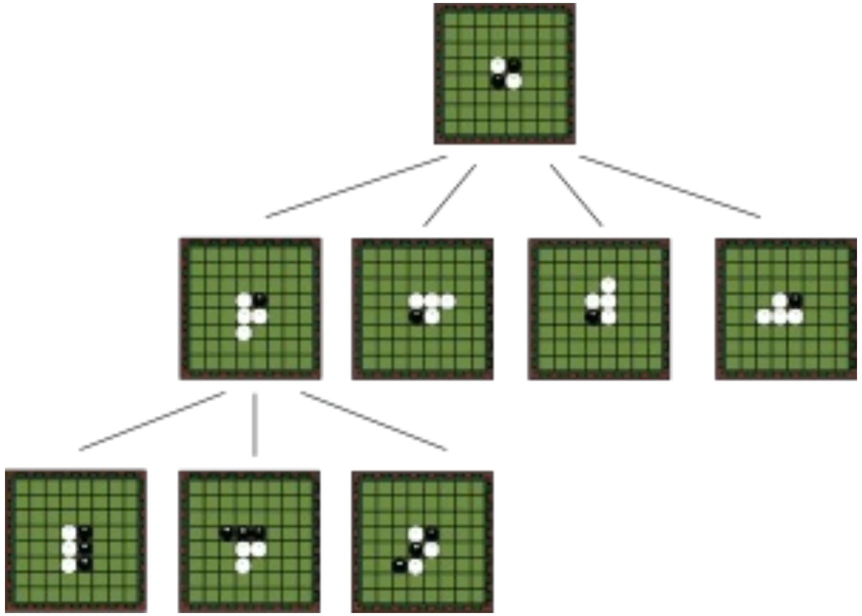
- Problem formulation
  - Initial State, Operators, Goal Test, Path Cost
- Problem Representation
  - Tree structure representation
- Solving problems by searching
  - Blind Search (BFS, DFS and UCS)
  - Heuristic Search (Greedy an A* Search)
- =>Adversarial Search
  - Game Play

LECTURE OUTLINE

- Definition of Games and adversarial search
- Minimax algorithm
- Alpha-beta pruning of search trees
- Usage of Minimax algorithm (*)
  - GAN: State-of-the-art deep learning algorithm

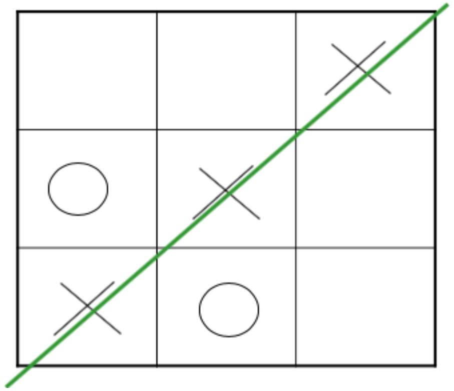
GAME PLAYING

- Why study game playing in AI?
  - Games are intelligent activities
  - It is very easy to measure success or failure
  - Does not require large amounts of knowledge
  - They were thought to be solvable by straightforward search from the starting state to a winning position



GAME PLAYING (ADVERSARIAL SEARCH)

- Until now we have often assumed the situation is not going to change whilst we search
  - Shortest route between two towns
  - The same goal board of 8-puzzle
- Game playing is not like this
  - Not sure of the state after your opponents move
  - Goal of your opponent is to prevent your goal, and vice versa
  - Agent's goals are conflict, blind search won't be helpful
  - giving rise to adversarial search



GAME PLAYING - MINIMAX

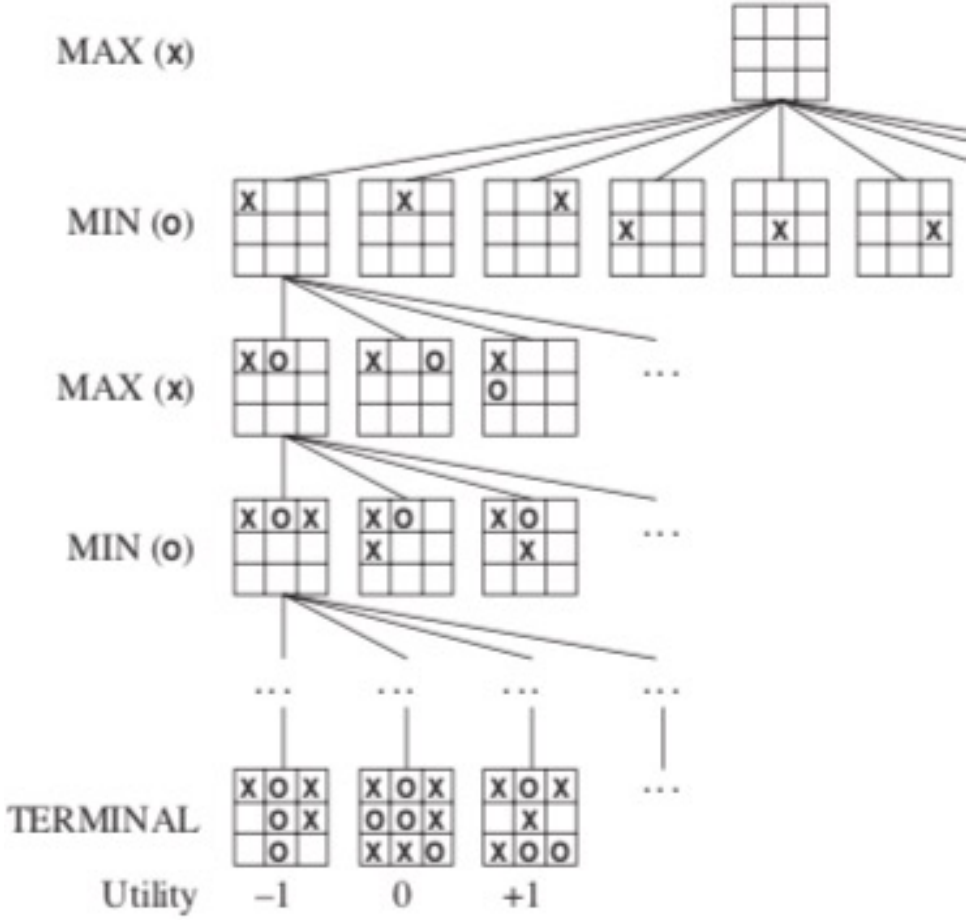
- An opponent tries to prevent your win at every move
  - 1944 - John von Neumann
  - A search method (Minimax)
  - maximise your position whilst minimising your opponent's
- Utility is an abstract measuring the amount of satisfaction you receive from something
  - We need a method of measuring how good a position is
  - Often called a utility function
  - Initially this will be a value that describes our position exactly

ZERO-SUM GAMES

- Fully observable environments (perfect information) in which **two agents** act **attend**
- Utility values at the end of the game are always **equal** or **opposite**. (0+1, 1+0, or ½+½, total payoff is the same)
  - For example, if one player wins a game, the other player necessarily loses.
- This **opposition** between the agents' utility functions makes the situation **adversarial**.

COMPONENTS OF GAME SEARCH

- A game can be defined as a kind of search problem with the following components :
  - The initial state: board position, indication of whose move it is
  - A set of operators: define the legal moves that a player can make
  - A terminal test: determines when the game is over (terminal states)
  - A utility (payoff) function: gives a numeric value for the outcome (terminal state) of a game (chess: +1,0, 1/2) ?



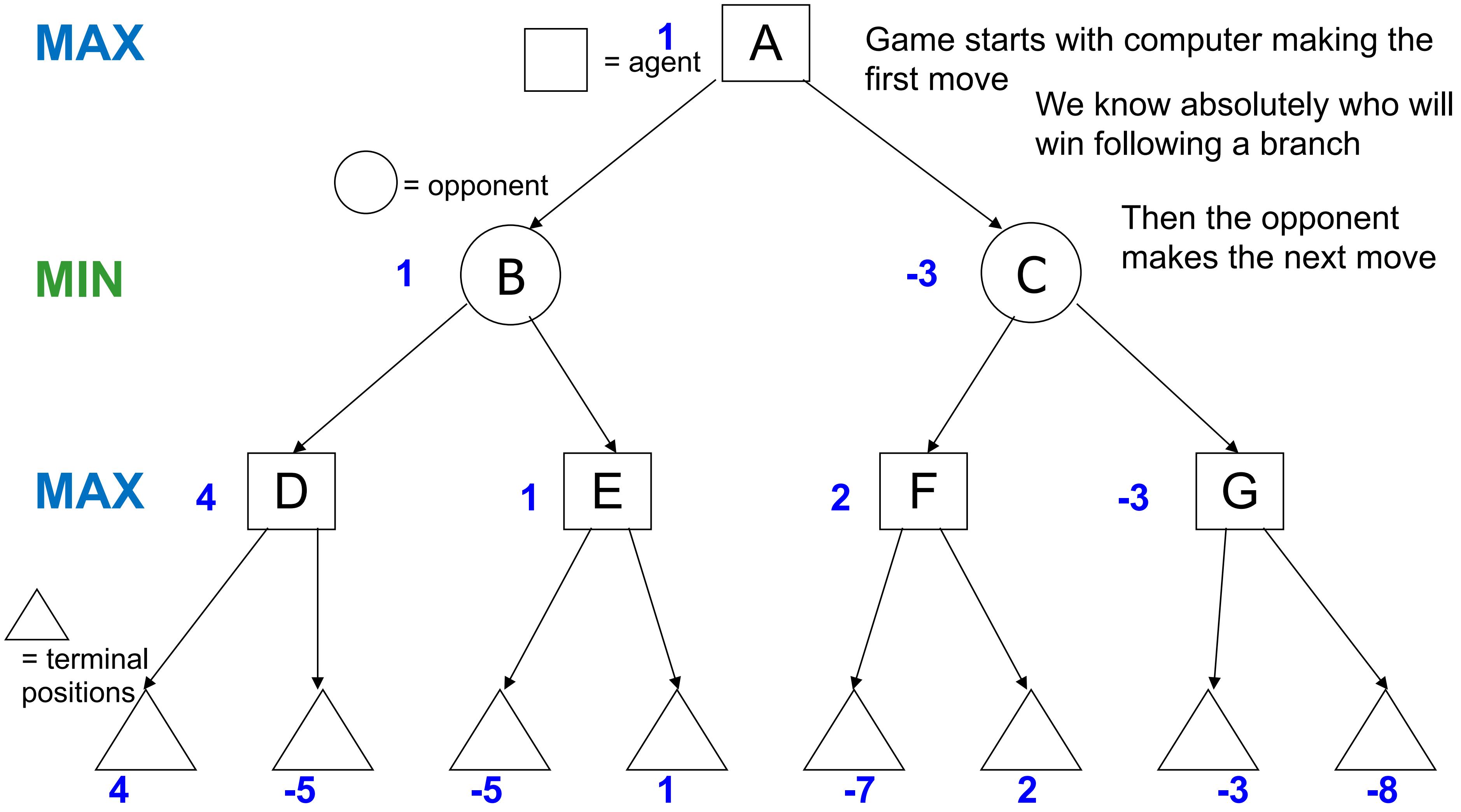
GAME PLAYING - MINIMAX

- In discussion of minimax
  - two players “MAX” and “MIN”
  - utility function (minimax value) of a node: the utility of (for MAX) being in the corresponding state (larger values are better for “MAX”, vice versa)
- MAX: take the best move for MAX
  - Next state: the one with the highest utility, i.e. the maximum of its children in the search tree
- MIN: take the best move for MIN (the worst for MAX)
  - Next state: the one with the lowest utility i.e. the minimum of its children in the search tree

Assume we can generate the full search tree

Of course for larger problem it's not possible to draw the entire tree

The idea is computer wants to force the opponent to lose, and maximise its own chance of winning



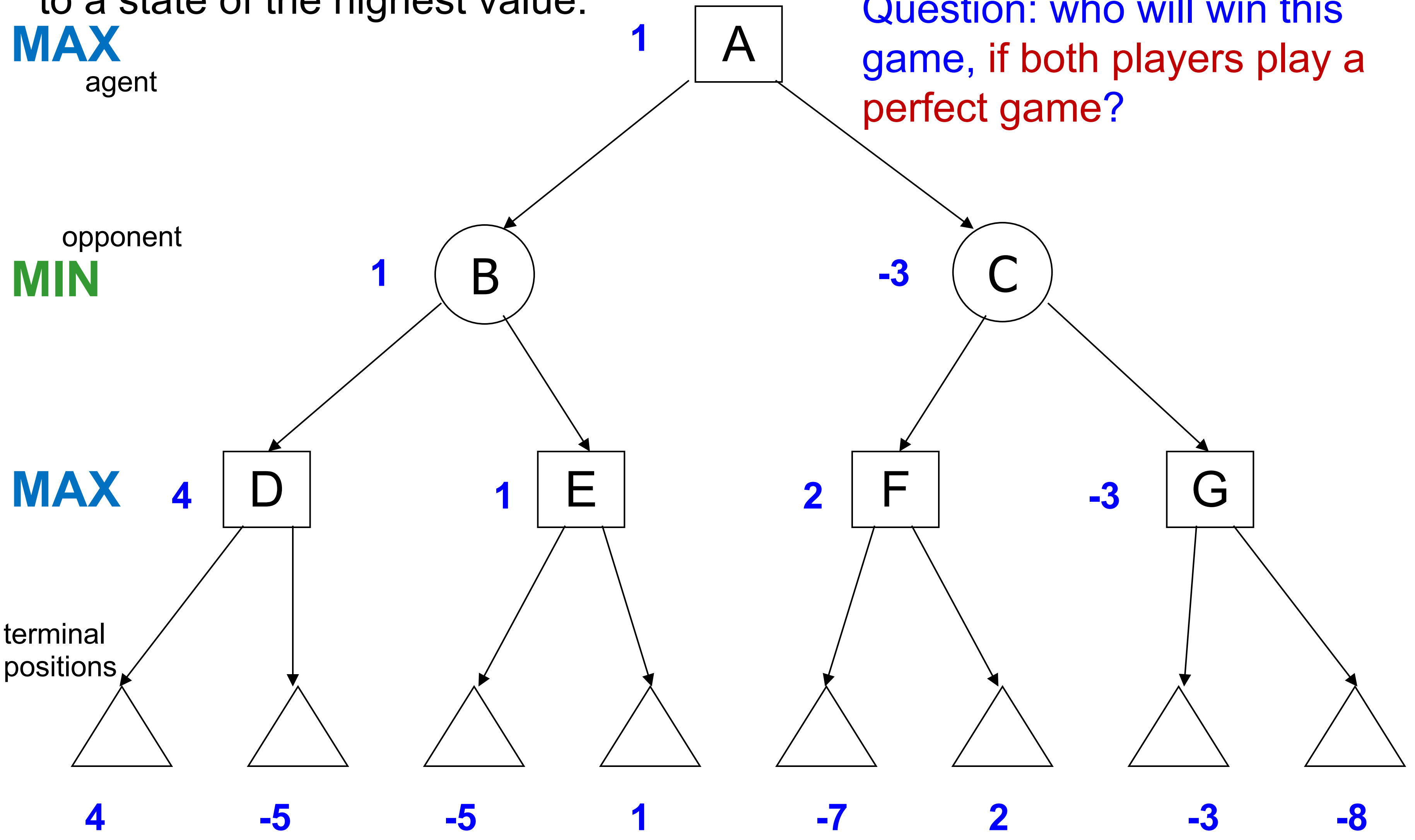
Values are propagated back up through the tree based on whose turn it is and whether they are trying to maximise or minimise at the point

Assume positive: computer wins

Now we can decide who will win the game

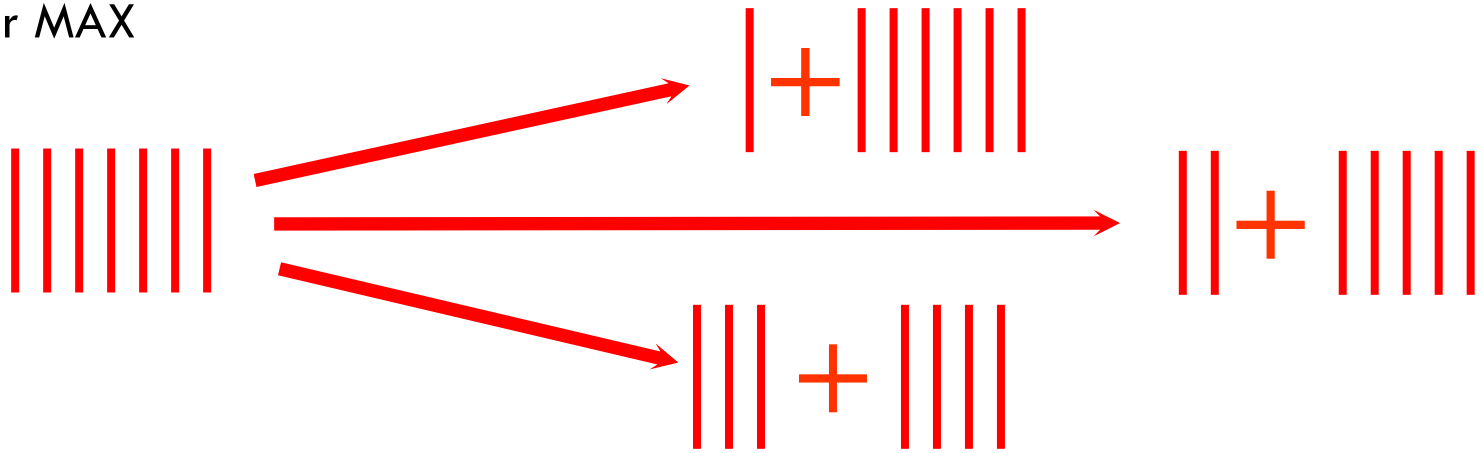
Now the computer is able to play a perfect game. At each move it'll move to a state of the highest value.

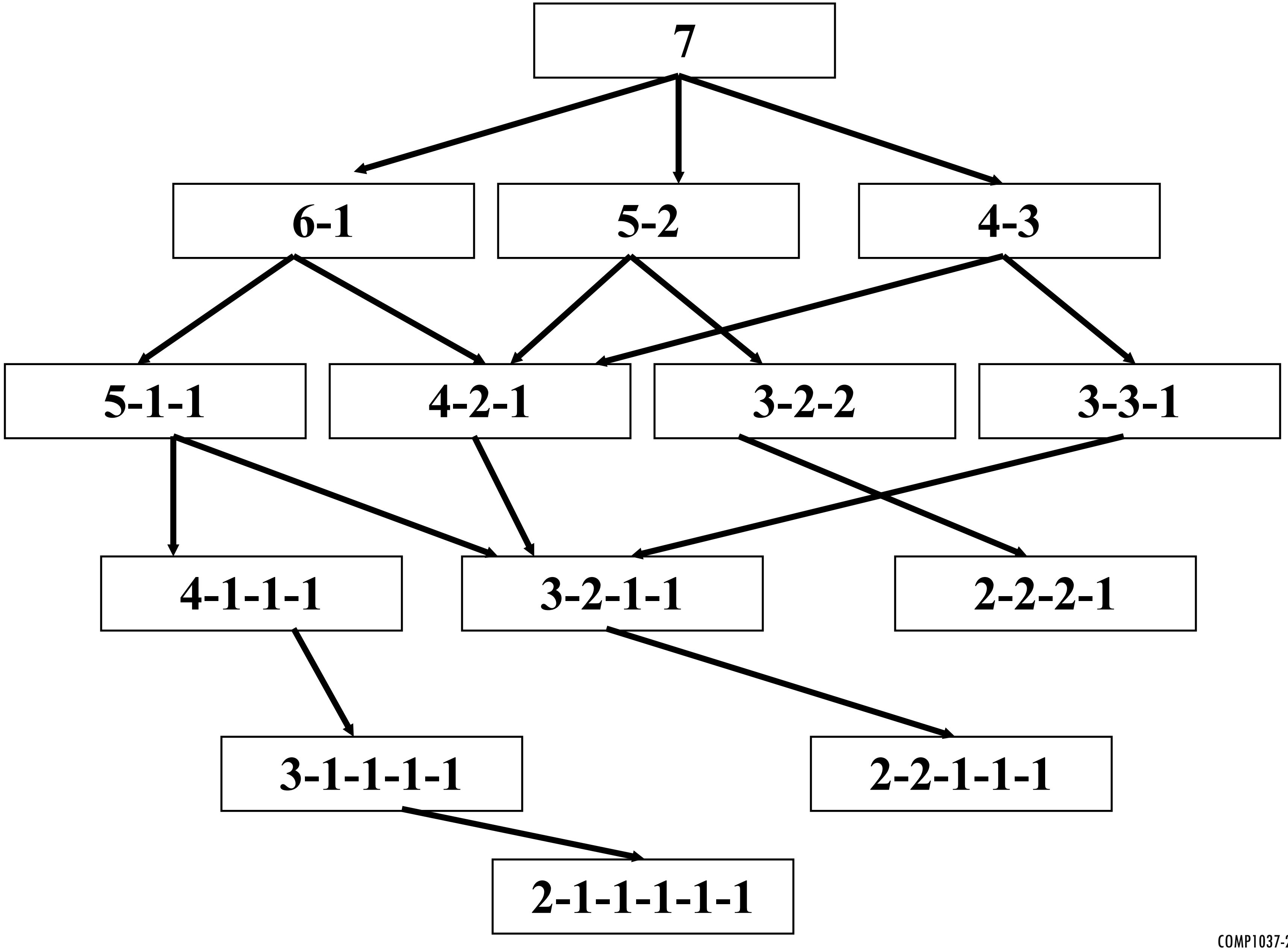
Question: who will win this game, if both players play a perfect game?

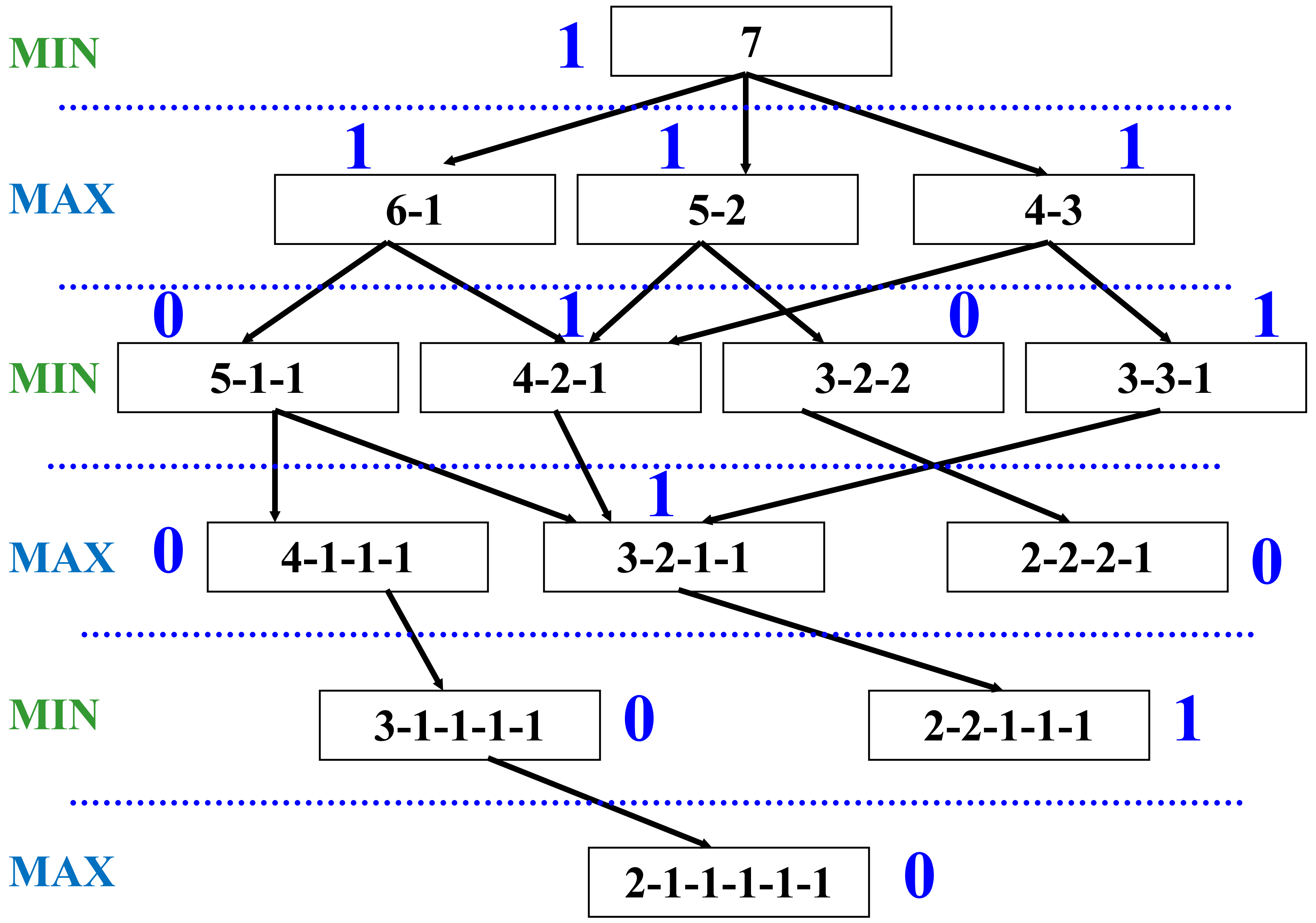


GAME PLAYING - MINIMAX

- Nim
  - Start with a pile of tokens, at each move the player must divide the tokens into two non-empty, non-equal piles
  - Starting with 7 tokens, draw the complete search tree
- Assume that a utility function of
  - 0 = a win for MIN
  - 1 = a win for MAX




7
6-1
5-2
4-3
5-1-1
4-2-1
3-2-2
3-3-1
4-1-1-1
3-2-1-1
2-2-2-1
3-1-1-1-1
2-2-1-1-1
2-1-1-1-1-1


MIN
MAX
MIN
MAX
MIN
MAX
1
7
1
6-1
1
5-2
1
4-3
0
5-1-1
1
4-2-1
0
3-2-2
1
3-3-1
0
4-1-1-1
1
3-2-1-1
2-2-2-1
0
3-1-1-1-1
0
2-2-1-1-1
1
2-1-1-1-1-1
0

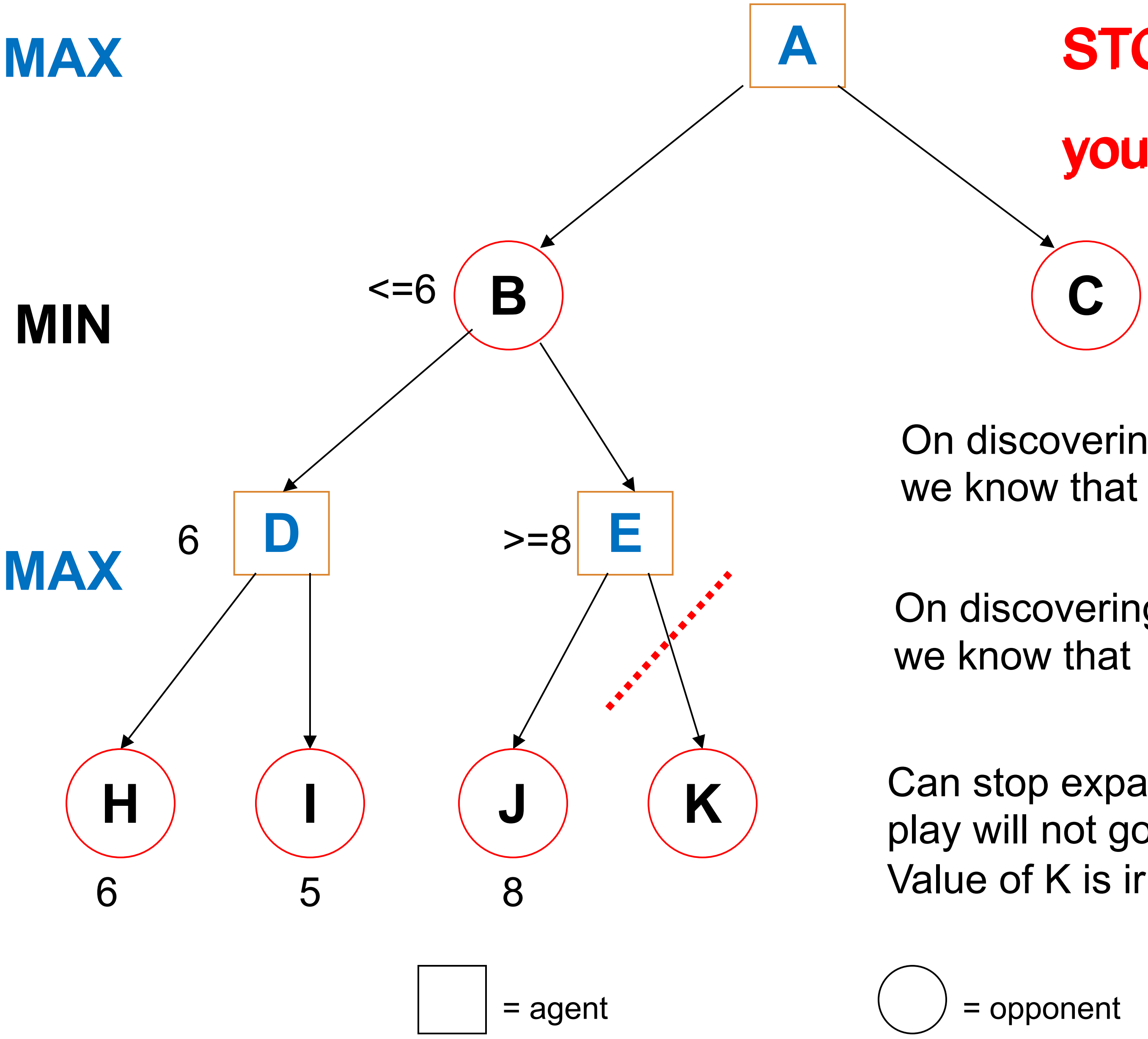
GAME PLAYING - MINIMAX

- Efficiency of the search
  - Game trees are very big
  - Evaluation of positions is time-consuming

- How can we reduce the number of nodes to be evaluated?
  - alpha-beta pruning based on minimax, Deep Blue
  - Better estimation of utility values (possibility of winning), i.e. heuristic, ANN, etc.

GAME PLAYING – ALPHA-BETA PRUNING

- **Pruning** allow us to **ignore** portions of the search tree that make no difference to the final choice
  - The number of nodes grow exponentially,
  - It is possible to compute the correct minimax decision without looking at every node in the game tree
  - Use the idea of pruning to eliminate large parts of the tree from consideration

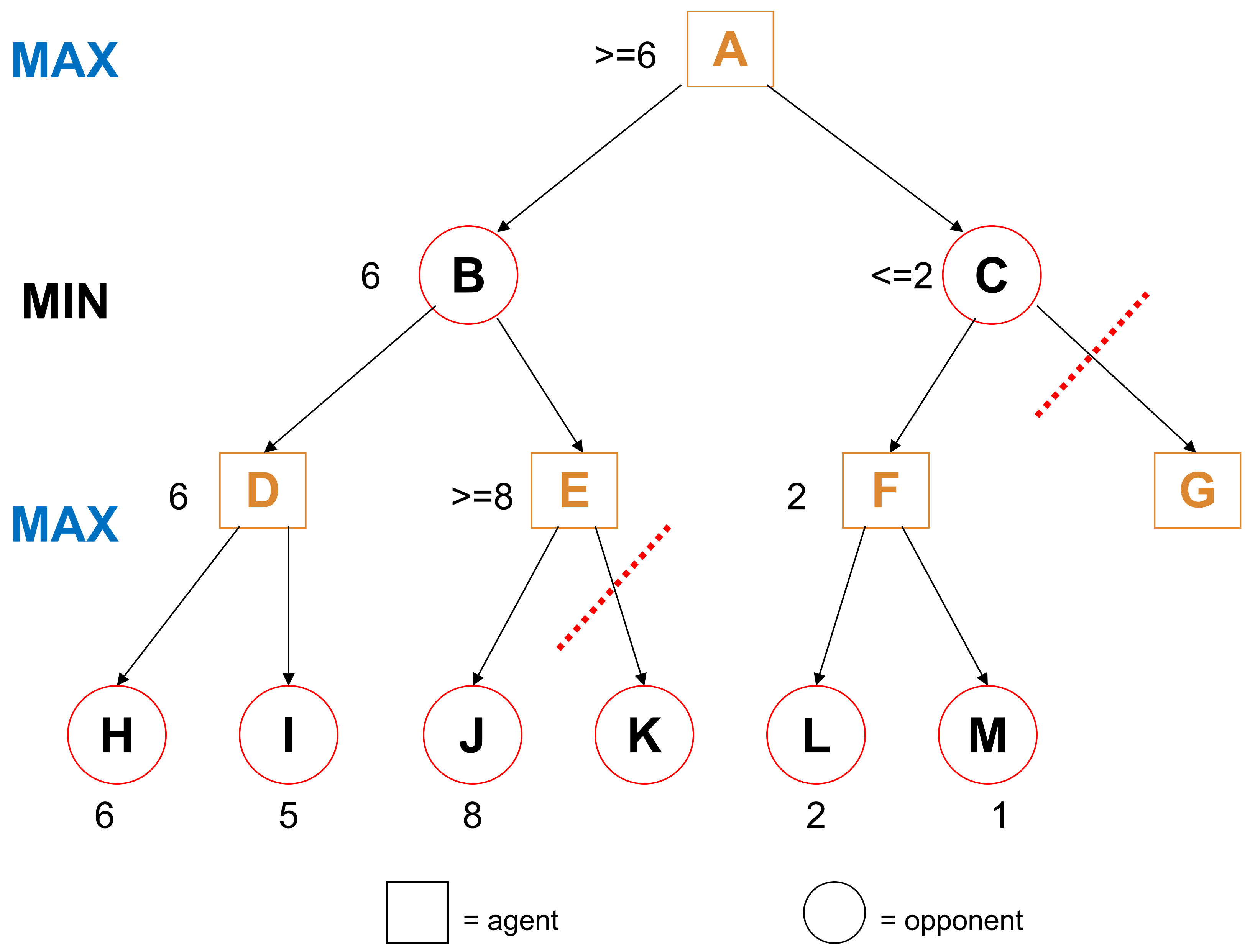


STOP! What else can you deduce now!?

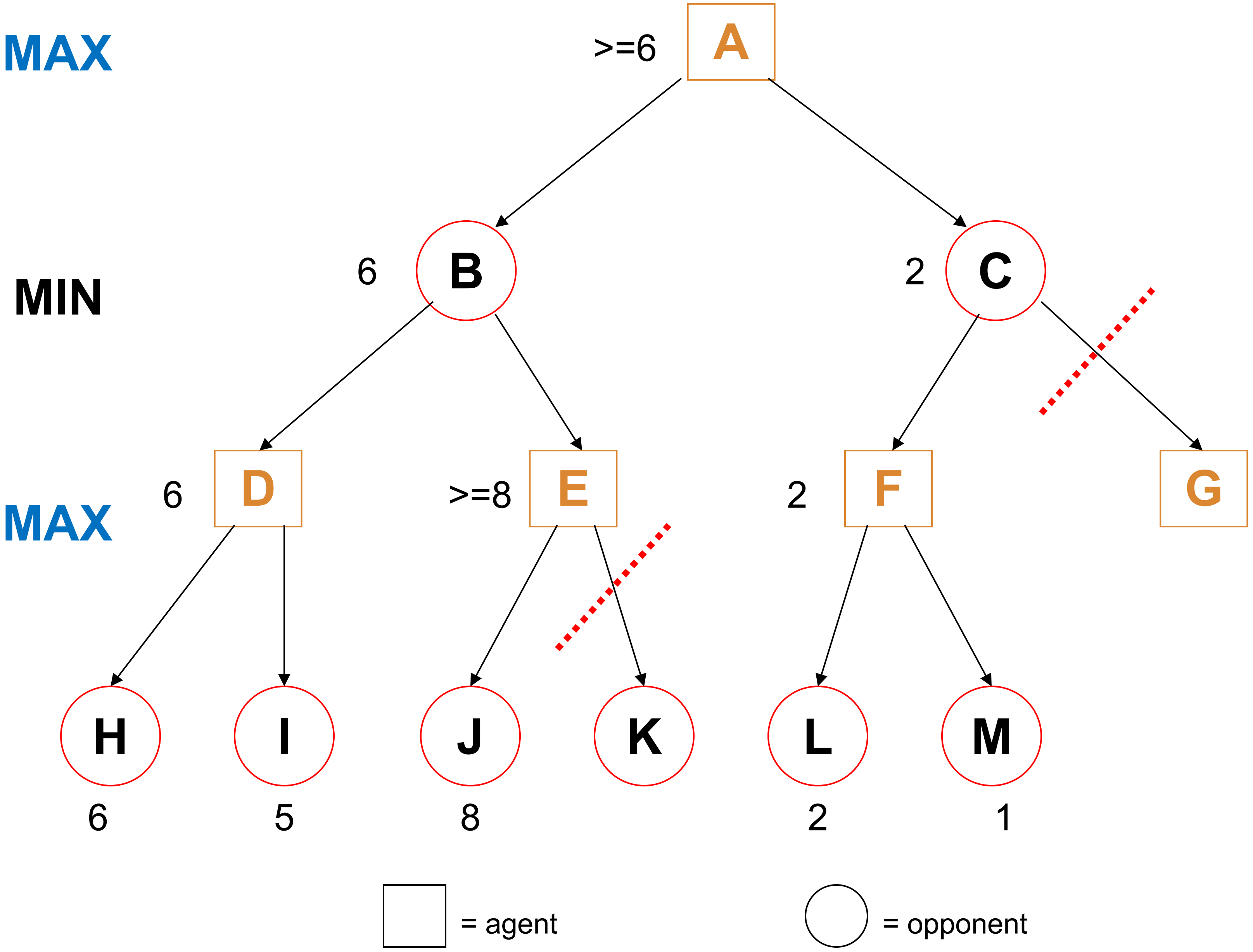
On discovering util(D) = 6
we know that util(B) <= 6

On discovering util(J) = 8
we know that util(E) >= 8

Can stop expansion of E as best play will not go via E
Value of K is irrelevant – prune it!

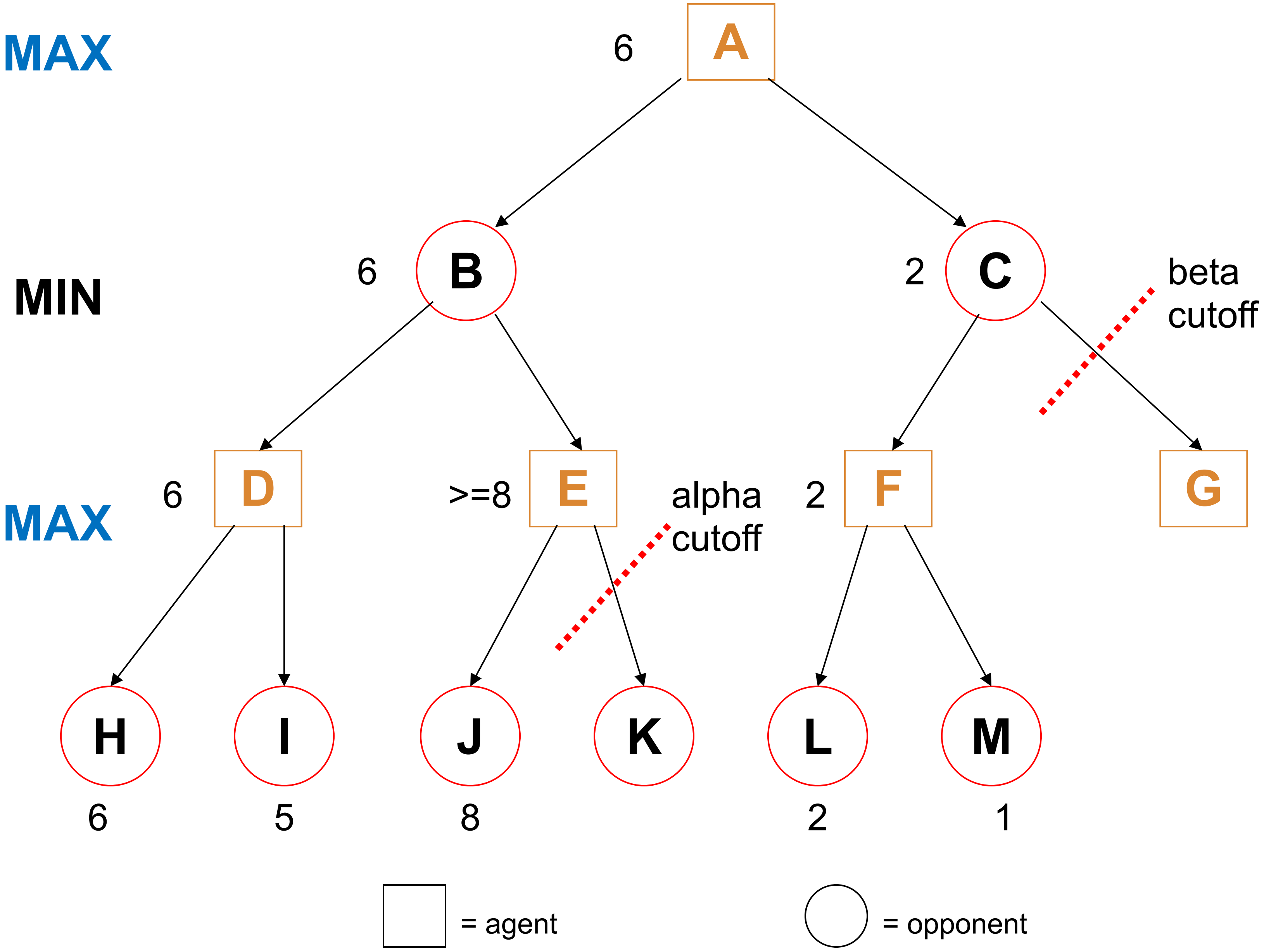


MAX
MIN
MAX
>=6
A
6
B
<=2
C
6
D
>=8
E
2
F
G
H
I
J
K
L
M
6
5
8
2
1
= agent
= opponent



MAX
MIN
MAX
>=6
A
6
B
2
C
6
D
>=8
E
2
F
G
H
I
J
K
L
M
6
5
8
2
1
= agent
= opponent

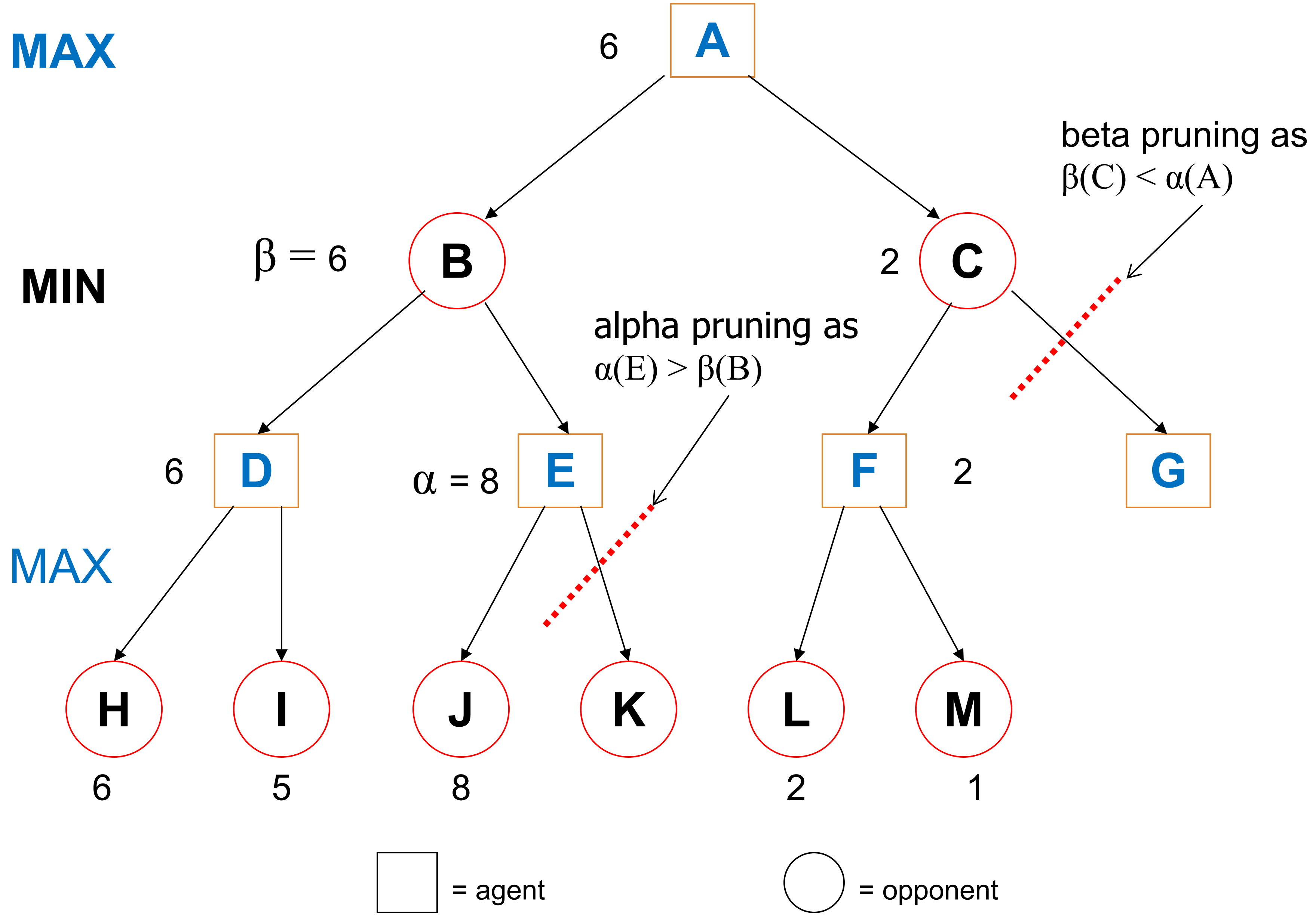
Alpha-beta Pruning



GAME PLAYING - ALPHA-BETA PRUNING

- If this is done well then alpha-beta search can effectively double the depth of search tree that is searchable in a given time
  - Effectively reduces the branching factor in chess from about 30 to about 8
  - This is an enormous improvement!
- These bounds are stored in terms of two parameters
  - alpha α: α values are stored with each MAX node
  - the highest-value we have found so far at any choice point along the path of MAX
  - beta β: values are stored with each MIN node
  - the lowest-value we have found so far at any choice point along the path of MIN

Alpha-beta Pruning



GAME PLAYING - ALPHA-BETA PRUNING

- If we were doing BFS, would you still be able to prune nodes in this fashion?
  - NO! Because the pruning on node D is made by evaluating the tree underneath D
  - This form of pruning relies on doing a DFS

- To maximise pruning: first expand the best children
  - cannot know which ones are really best
  - use heuristics for the "best-first" ordering
  - **Heuristic evaluation function** allow us to approximate the true utility of a state without doing a complete search.

MINIMAX USAGE
- GENERATIVE ADVERSARIAL NETWORK

- Problem: Generative Model
- Given training data, generate new samples from same distribution

Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

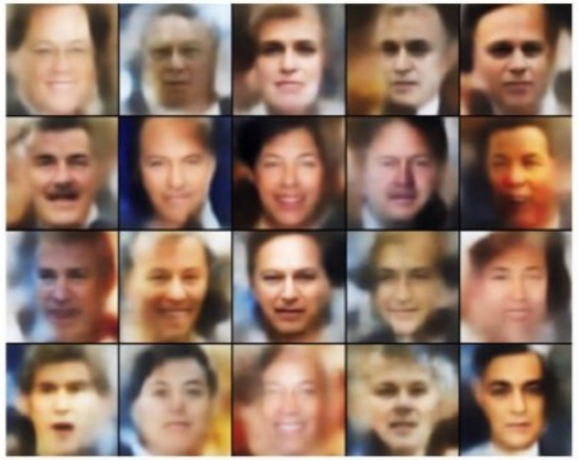
MINIMAX USAGE
- GENERATIVE ADVERSARIAL NETWORK

- Why Generative Model?
  - Generative models of time-series data can be used for simulation and planning
  - Training generative models can also enable inference of latent representations that can be useful as general features



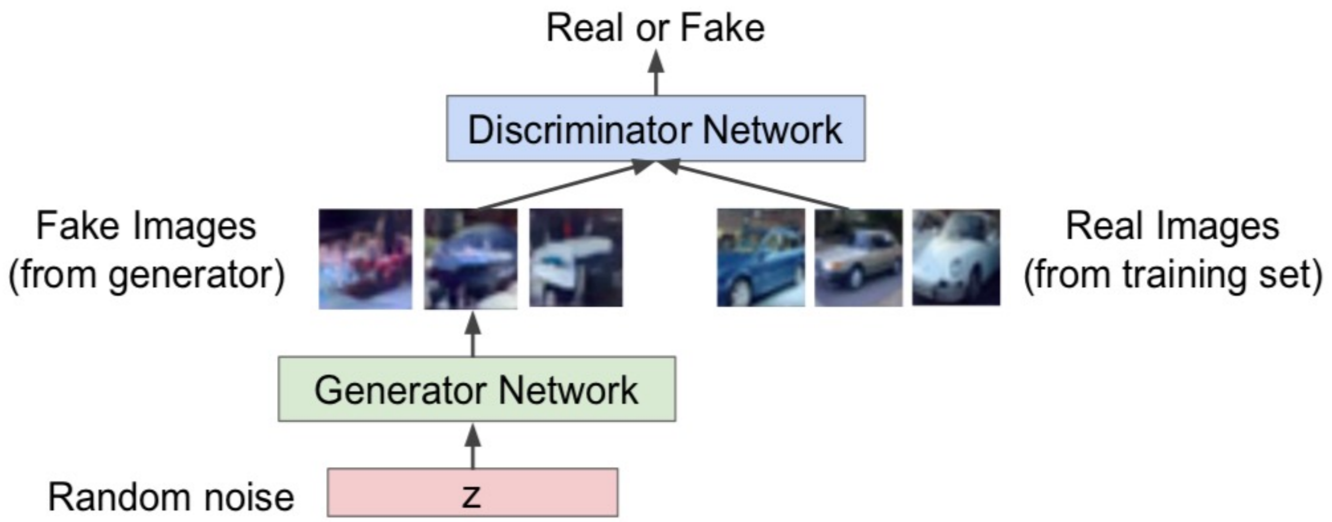
MINIMAX USAGE
- GENERATIVE ADVERSARIAL NETWORK

- GANs:
  - Instead of learning a explicitly density function p_{model}(x)
  - It takes game-theoretic approach: learn to generate from training distribution through 2-player game
  - Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014



MINIMAX USAGE
- GENERATIVE ADVERSARIAL NETWORK

- Training GANs: Two-player game
  - Generator network: try to fool the discriminator by generating real-looking images
  - Discriminator network: try to distinguish between real and fake images



MINIMAX USAGE
- GENERATIVE ADVERSARIAL NETWORK

- Training GANs: Two-player game
  - Generator network: try to fool the discriminator by generating real-looking images
  - Discriminator network: try to distinguish between real and fake images
  - Train jointly in minimax game

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data G(z)}}) \right]$$

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

MINIMAX USAGE
- GENERATIVE ADVERSARIAL NETWORK

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

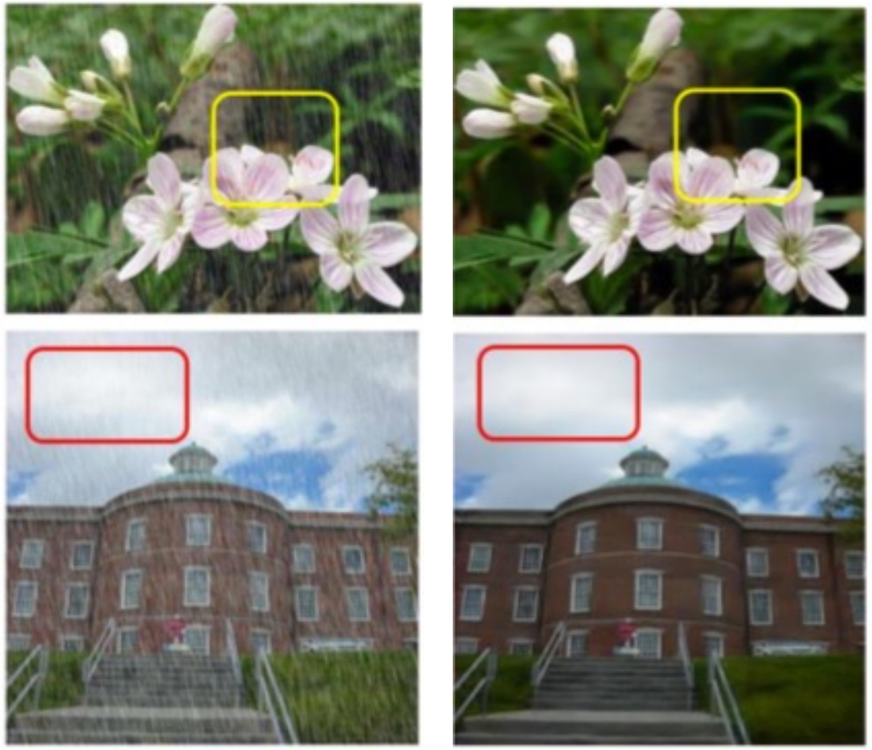
1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

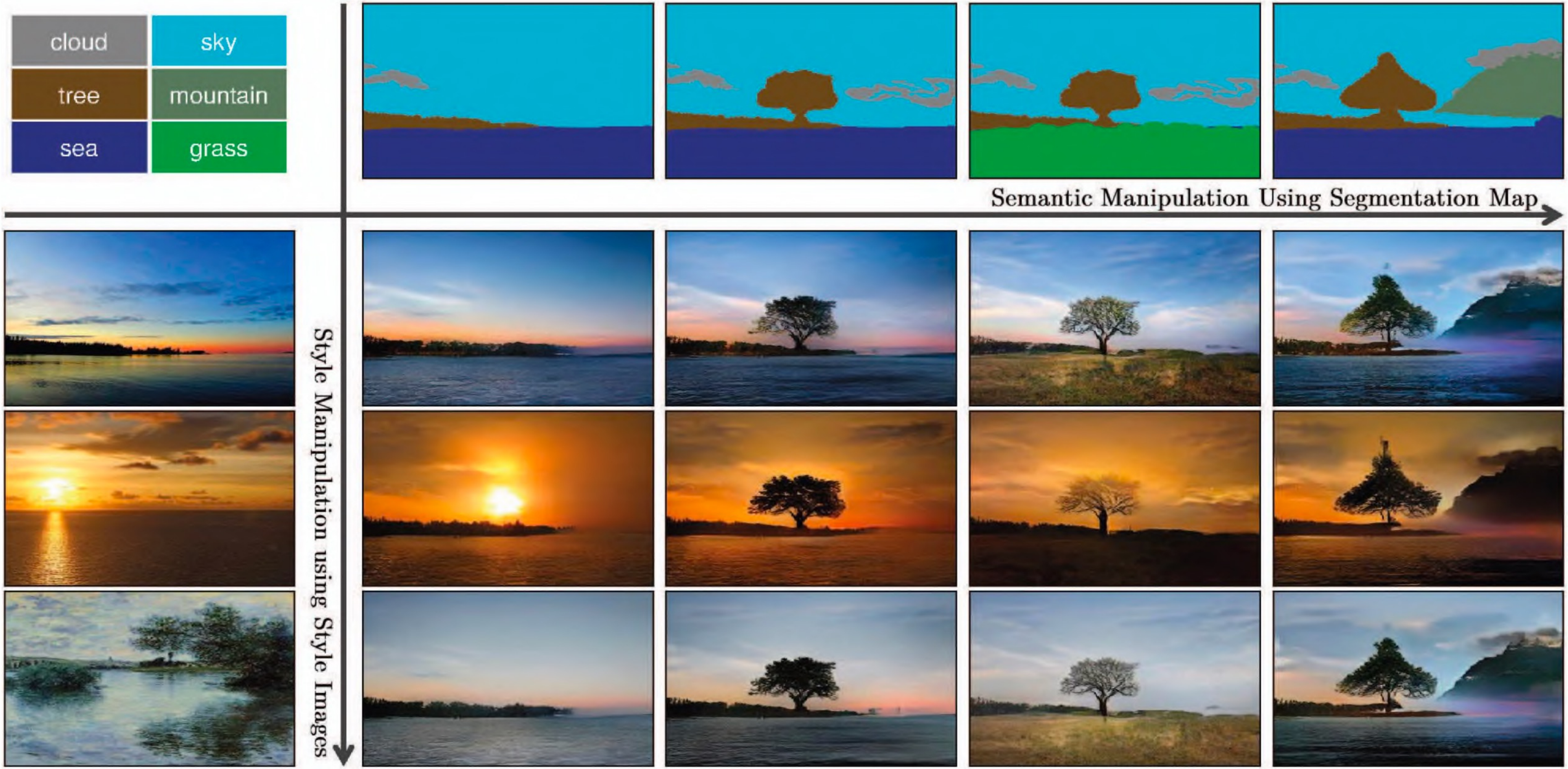
2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$


Input
Our result
Ground truth
Input
Our result
Ground truth
Zoom in




Source B
Source A
Coarse styles from source B


cloud
sky
tree
mountain
sea
grass
Semantic Manipulation Using Segmentation Map
Style Manipulation using Style Images

SUMMARY – GAME PLAYING

- Definitions of Game and adversarial search
- Techniques
  - Minimax
  - Alpha-beta pruning
- Game classifications
- Further reading: AIMA Adversarial search(5.1-5.3)