CS 480

Introduction to Artificial Intelligence

February 3, 2022

Announcements / Reminders

- Please follow the Week 04 TO DO List
- Quiz #01: due on Sunday (02/06) at 11:00 PM CST
- Written Assignment #1 will be posted this weekend
- Programming Assignment #1 will be posted within 1.5 - 2 weeks
- Exam dates (consider fixed):
  - Midterm: February 24, 2022 during lecture time
  - Final: April 28, 2022 during lecture time

Plan for Today

- **A* Heuristics revisited**
- **Problem Solving: Adversarial Search**

A* Algorithm: Evaluation Function

Calculate / obtain:

f(n) = **g(State$_n$)** + **h(State$_n$)**

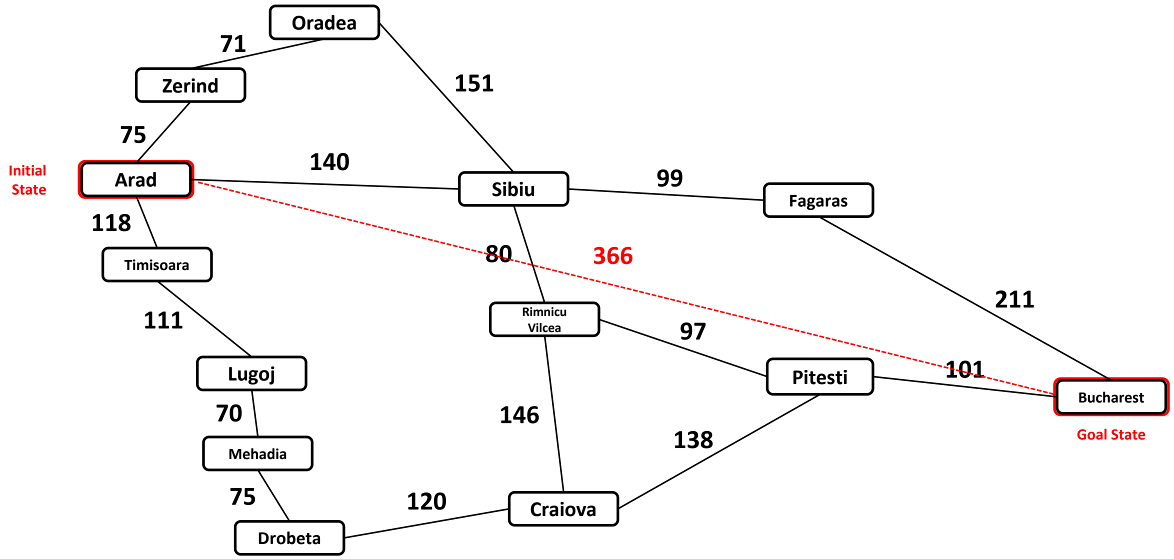
where:

- **g(n) - initial node to node n path cost**
- **h(n) - estimated cost of the best path that continues from node n to a goal node**

A state n with minimum (maximum) f(n) should be chosen for expansion

What Made A* Work Well?

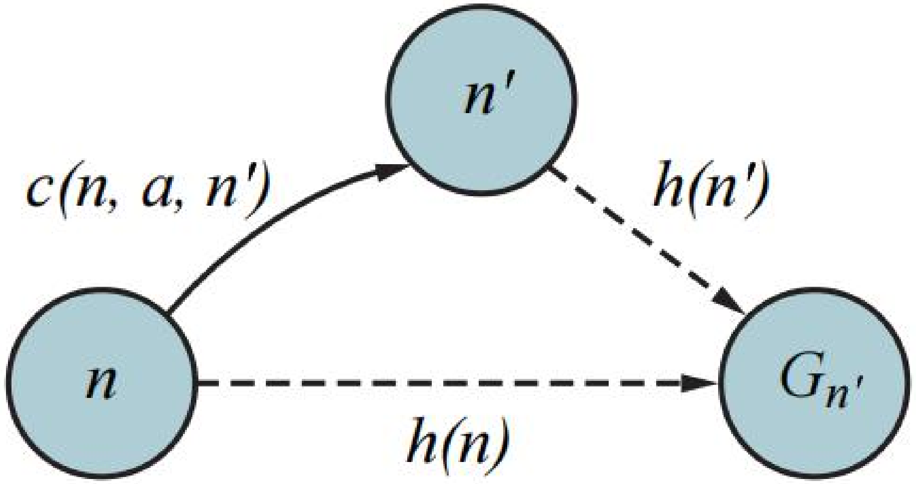
- Straight-line heuristics is admissible: it never overestimates the cost.



- An admissible heuristics is guaranteed to give you the optimal solution

What Made A* Work Well?

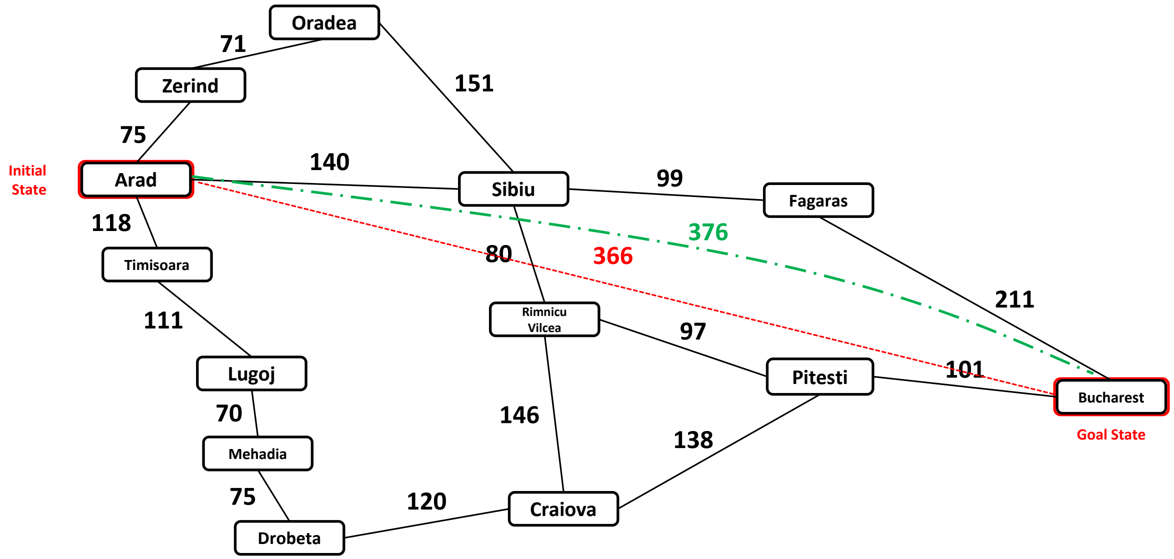
- **Straight-line heuristics is consistent: its estimate is getting better and better as we get closer to the goal**



- **Every consistent heuristics is admissible heuristics, but <u>not the other way around</u>**

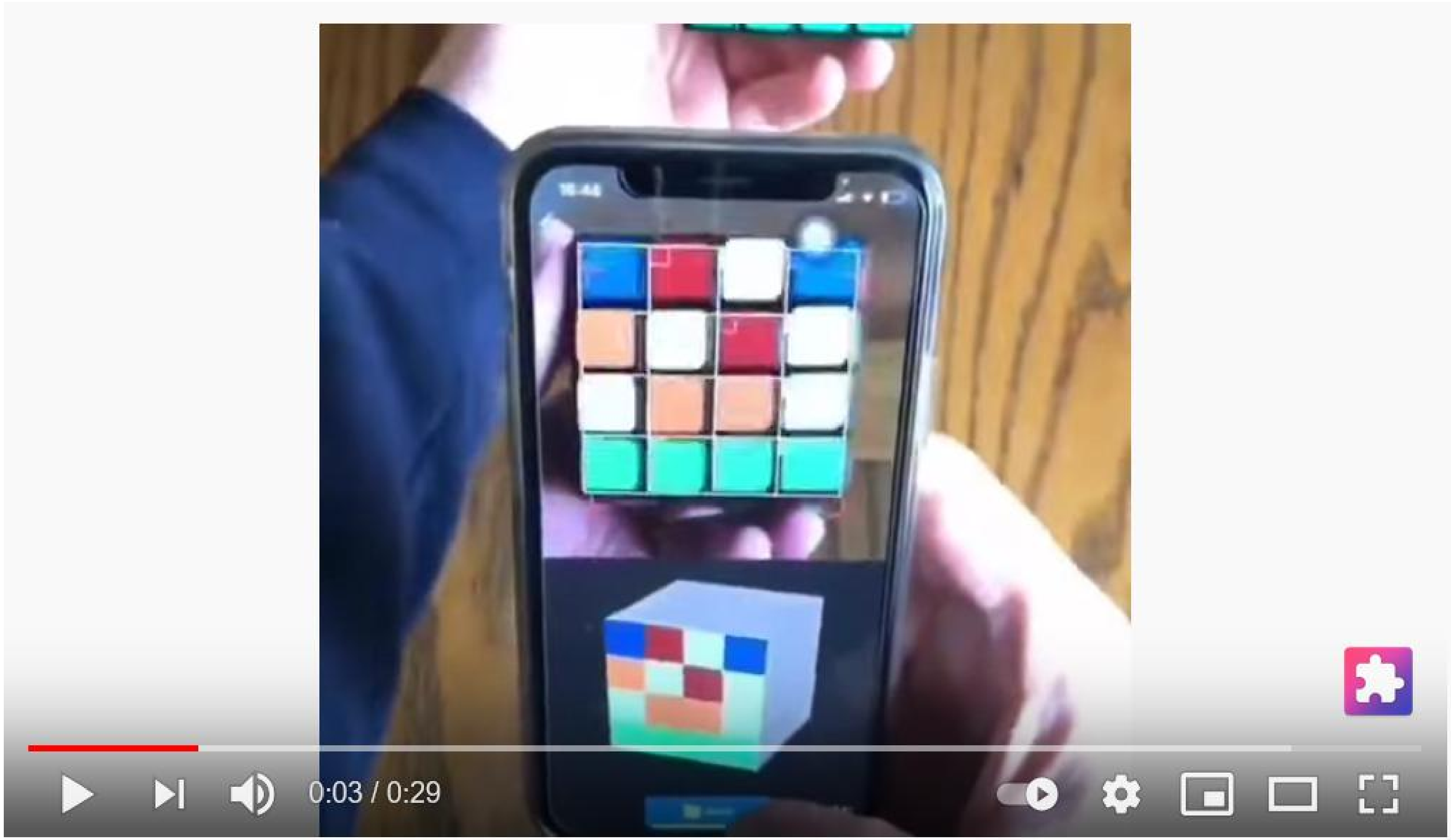
Dominating Heuristics

- We can have more than one available heuristics. For example $h_1(n)$ and $h_2(n)$.



- Heuristics $h_2(n)$ estimate is closer to actual cost than $h_1(n)$. $h_2(n)$ dominates $h_1(n)$. Use $h_2(n)$.

Informed Search: Application Example



#shorts - Solve Any Rubik Cube Using this Augmented Reality App

2,116 views • Apr 2, 2021 61 5 SHARE SAVE ...

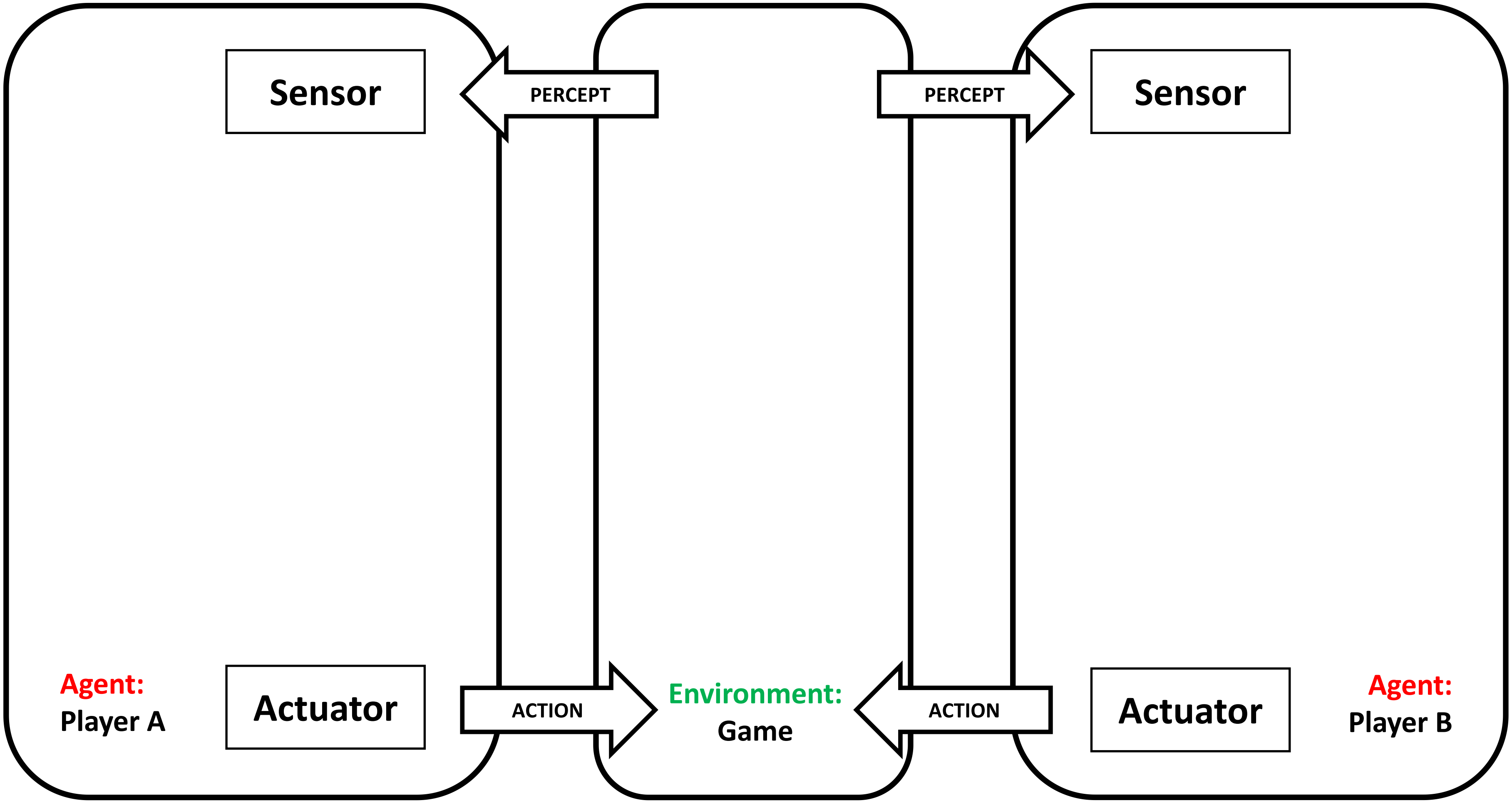
Source: https://www.youtube.com/watch?v=Pxbv2gEhnMk

Selected Searching Algorithms

- Searching Algorithms
  - **Uninformed** Search
    - Breadth First Search
    - Depth First Search
    - Brute Force Search
    - Uniform Cost Search
  - **Informed** Search
    - Greedy Best First Search
    - Branch and Bound
    - A* Algorithm
    - Hill Climbing Algorithm
    - Constraint Satisfaction Search
    - α - β Pruning
    - Min-Max Search

Uninformed search: an agent has no estimate how far it is from the goal
Informed search: an agent can estimate how far it is from the goal

Two-player Games



Perfect Information Zero Sum Games

- **Perfect information = fully observable**
- **Multiagent: number of players is 2 or more**
- **Multiagent: agents are competitve**
- **Zero-sum: "winner takes all"**
- **Examples:**
  - **Tic Tac Toe**
  - **Chess**

Two Player Games: Env Assumptions

Works with a "Simple Environment":

- Fully observable
- ~~Single agent~~ Mulitagent (competitive!)
- Deterministic
- Static
- Episodic / sequential
- Discrete
- Known to the agent

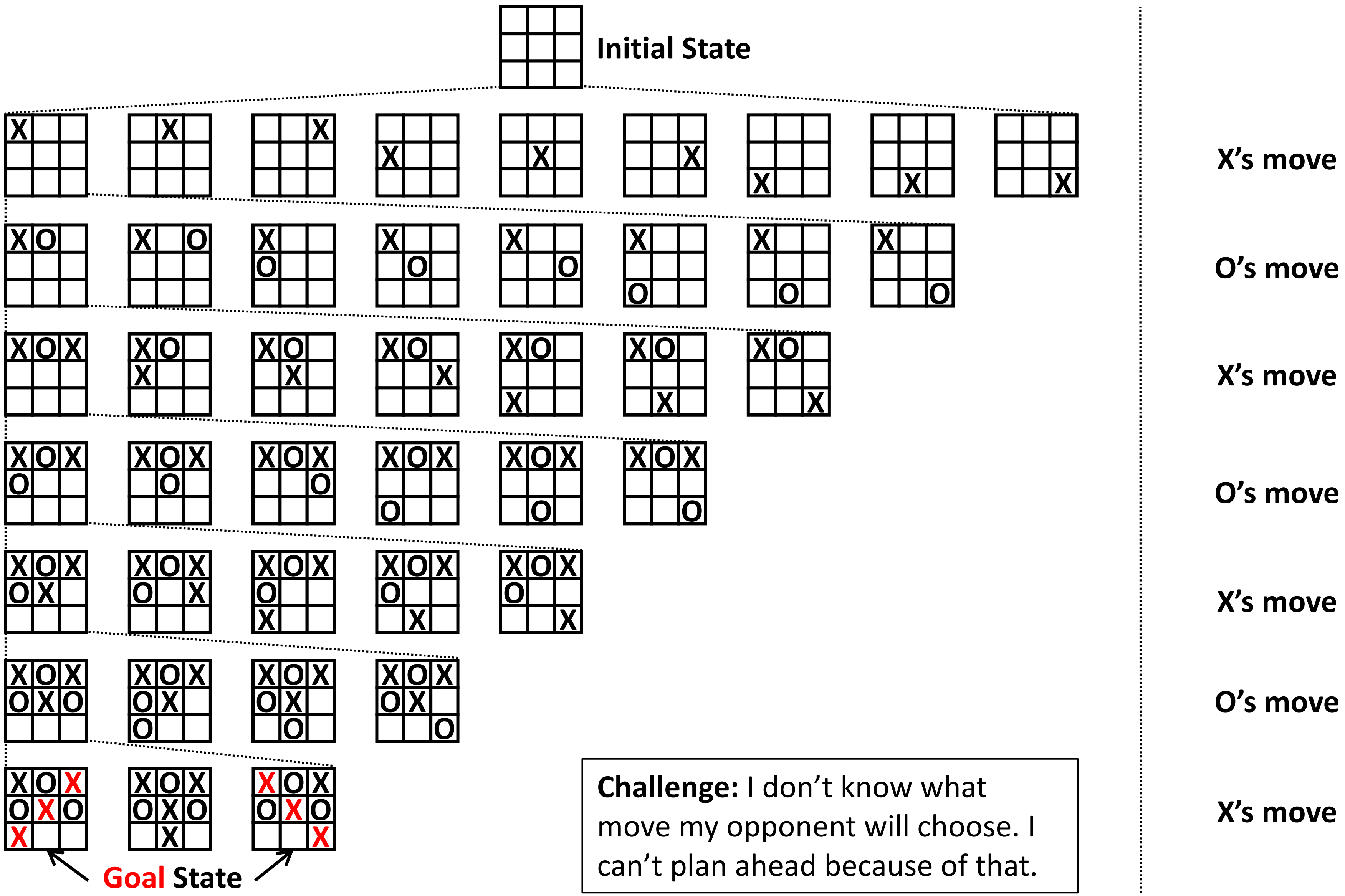
Defining Zero Sum Game Problem

- Define a set of possible states: State Space
- Specify how will you track Whose Move / Turn it is
- Specify Initial State
- Specify Goal State(s) (there can be multiple)
- Define a FINITE set of possible Actions (legal moves) for EACH state in the State Space
- Come up with a Transition Model which describes what each action does
- Come up with a Terminal Test that verifies if the game is over
- Specify the Utility (Payoff / Objective) Function: a function that defines the final numerical value to player p when the game ends in terminal state s

MinMax Algorithm: the Idea

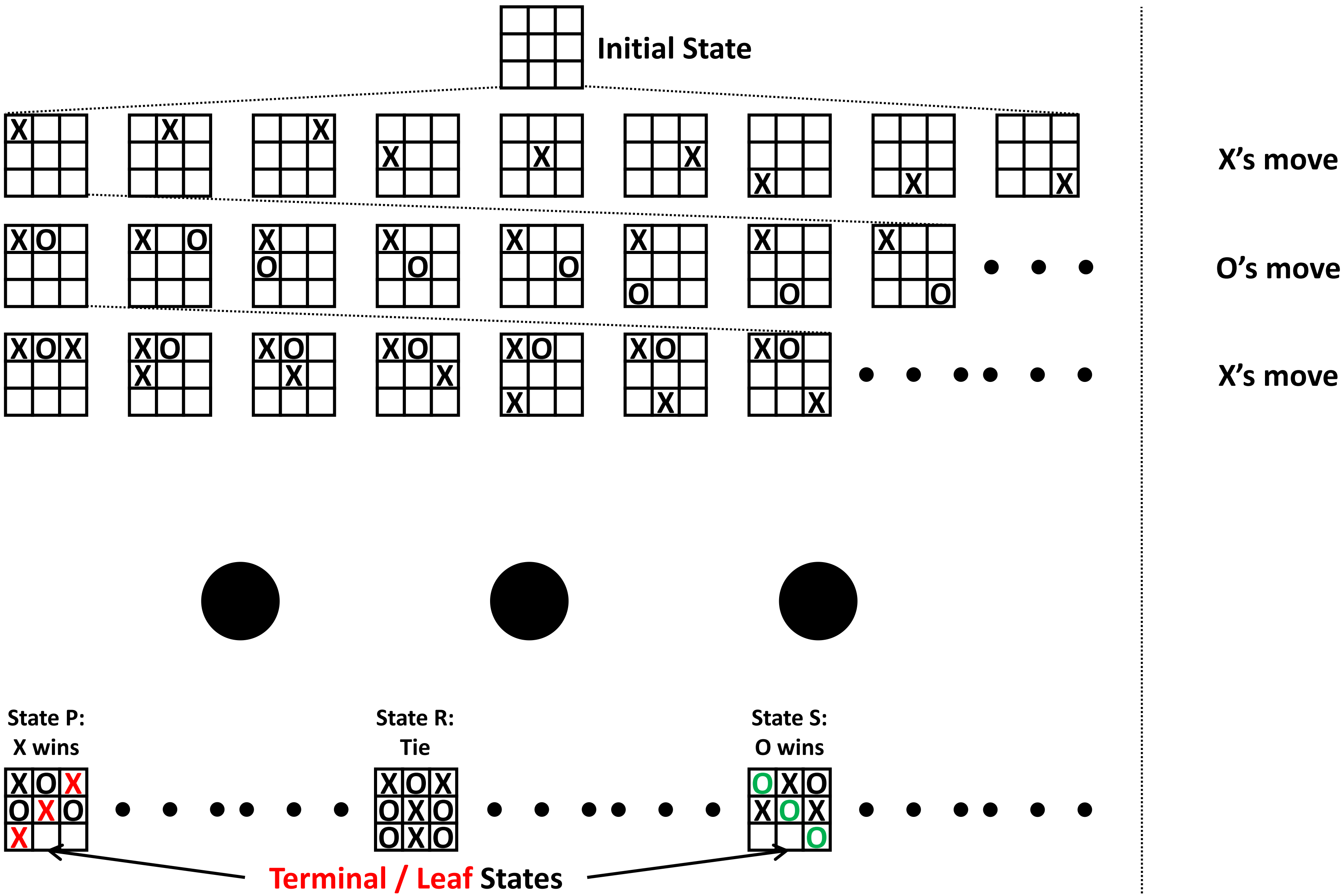
I don't know what move my opponent will choose, but I am going to ASSUME that it is going to be the best / optimal option

Tic Tac Toe: Zero Sum Game (2 Players)

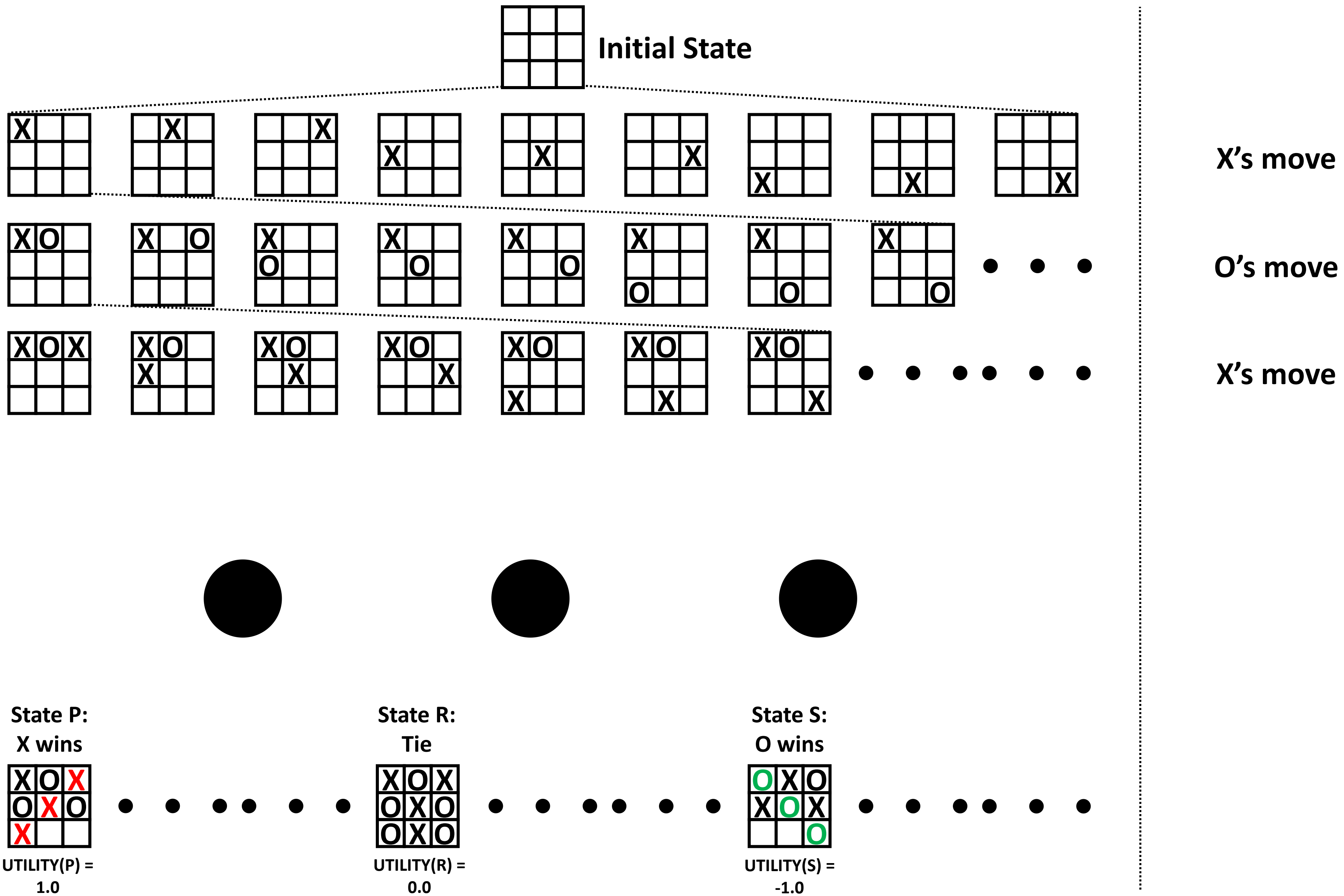


Challenge: I don't know what move my opponent will choose. I can't plan ahead because of that.

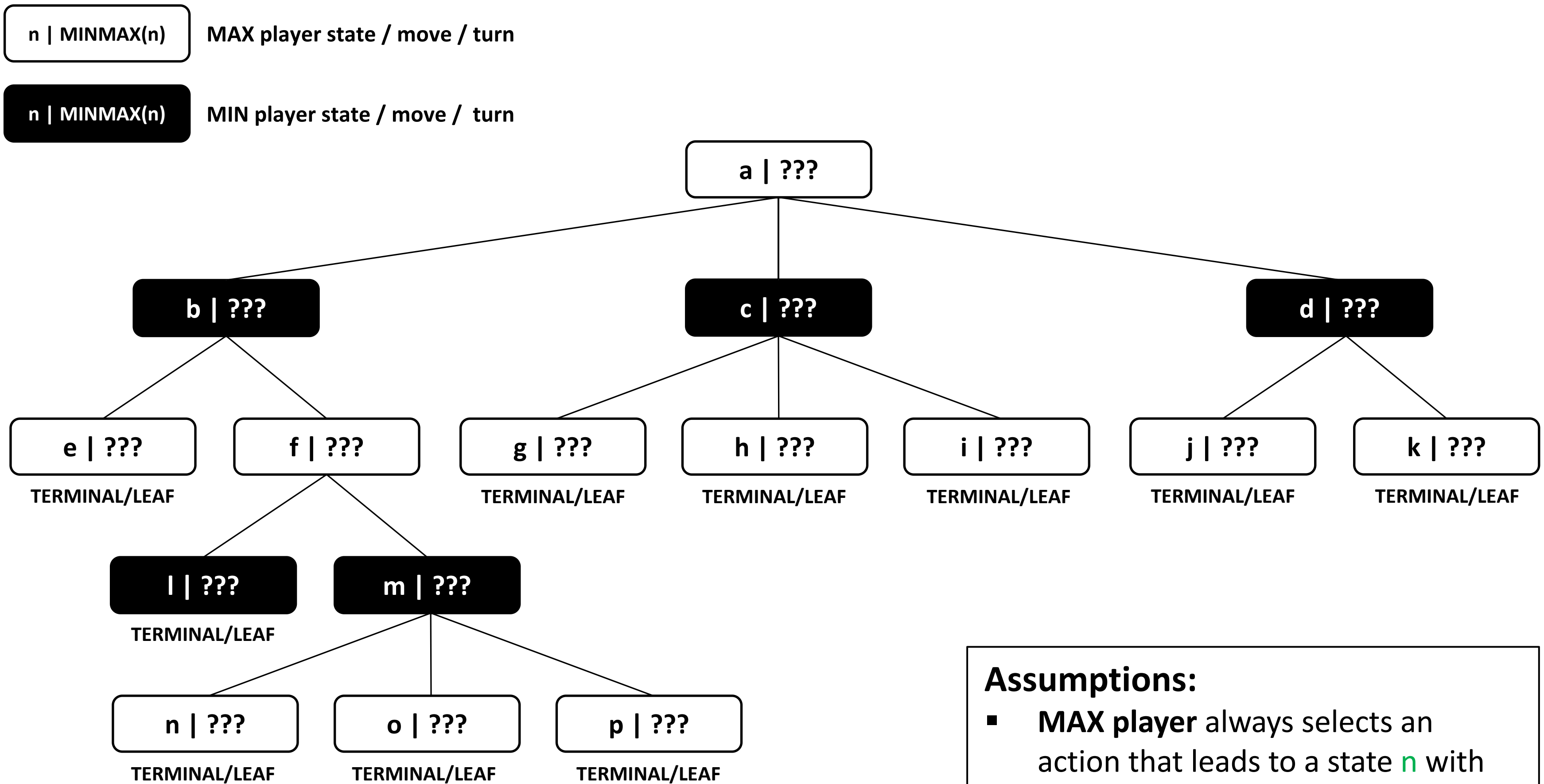
Tic Tac Toe: Zero Sum Game (2 Players)



Tic Tac Toe: Zero Sum Game (2 Players)



Example MinMax Search Tree



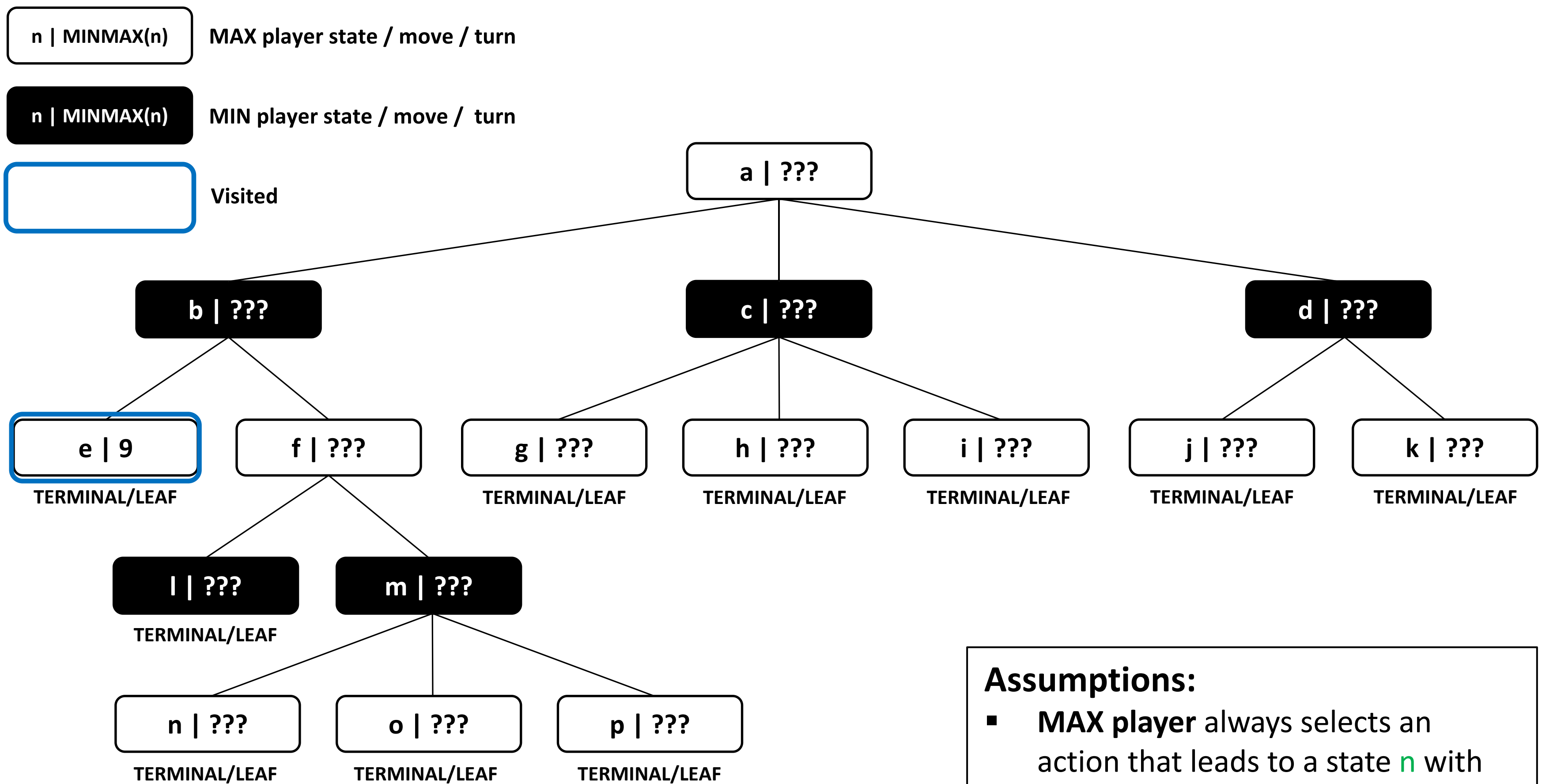
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), \text{ if } ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \text{ if } TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \text{ if } TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



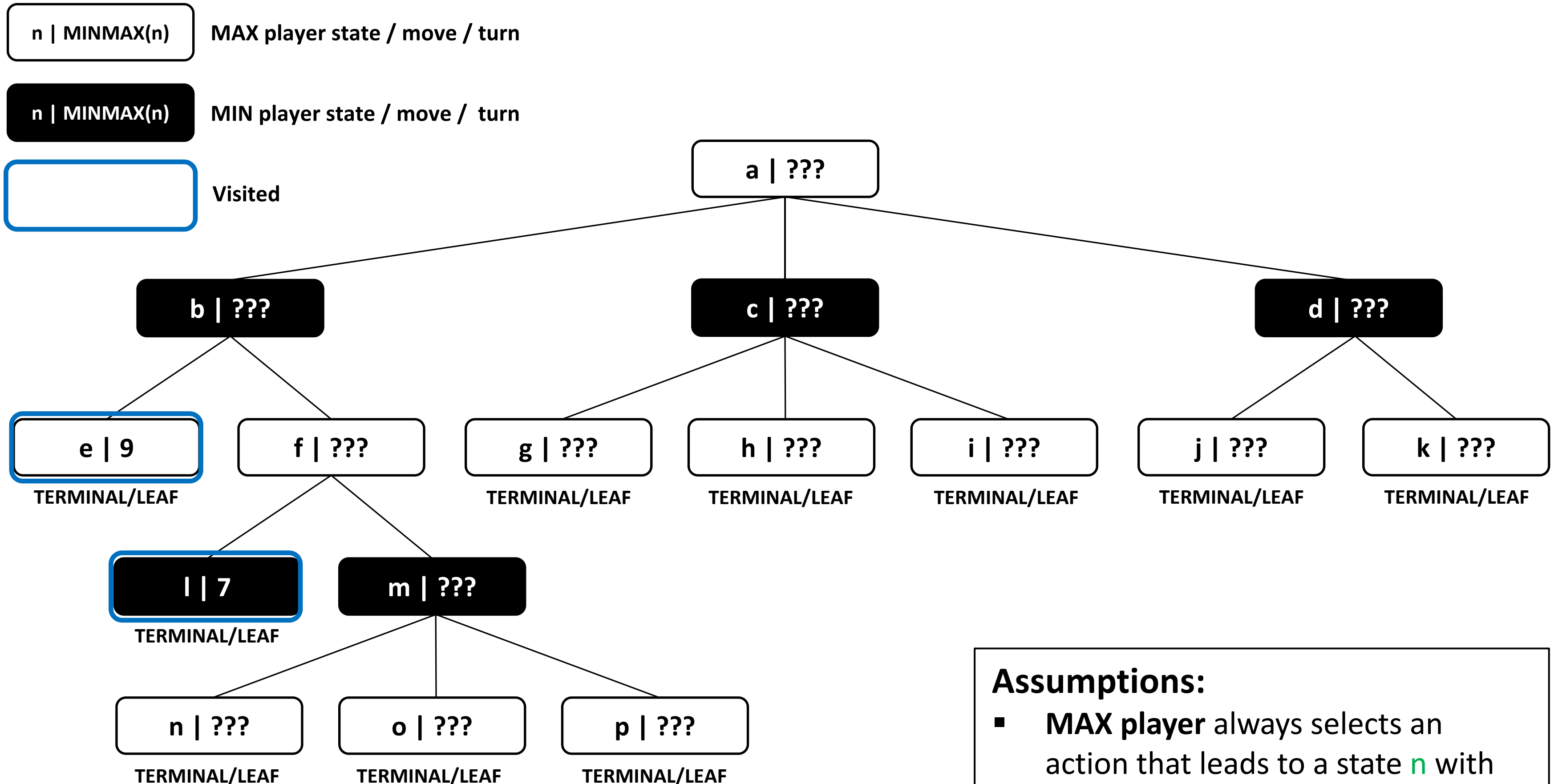
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), if\ ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), if\ TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), if\ TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



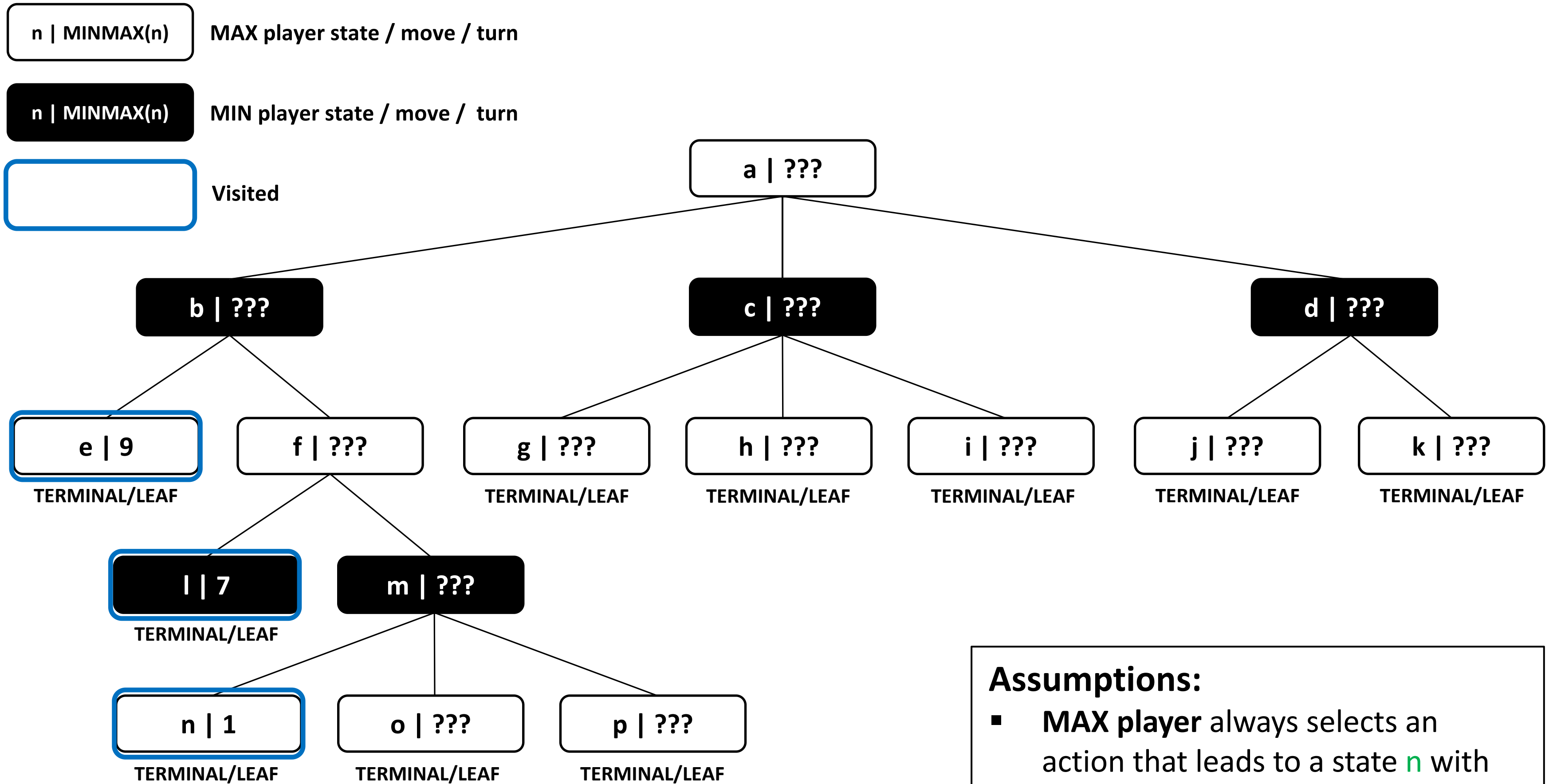
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), \text{ if } ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \text{ if } TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \text{ if } TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



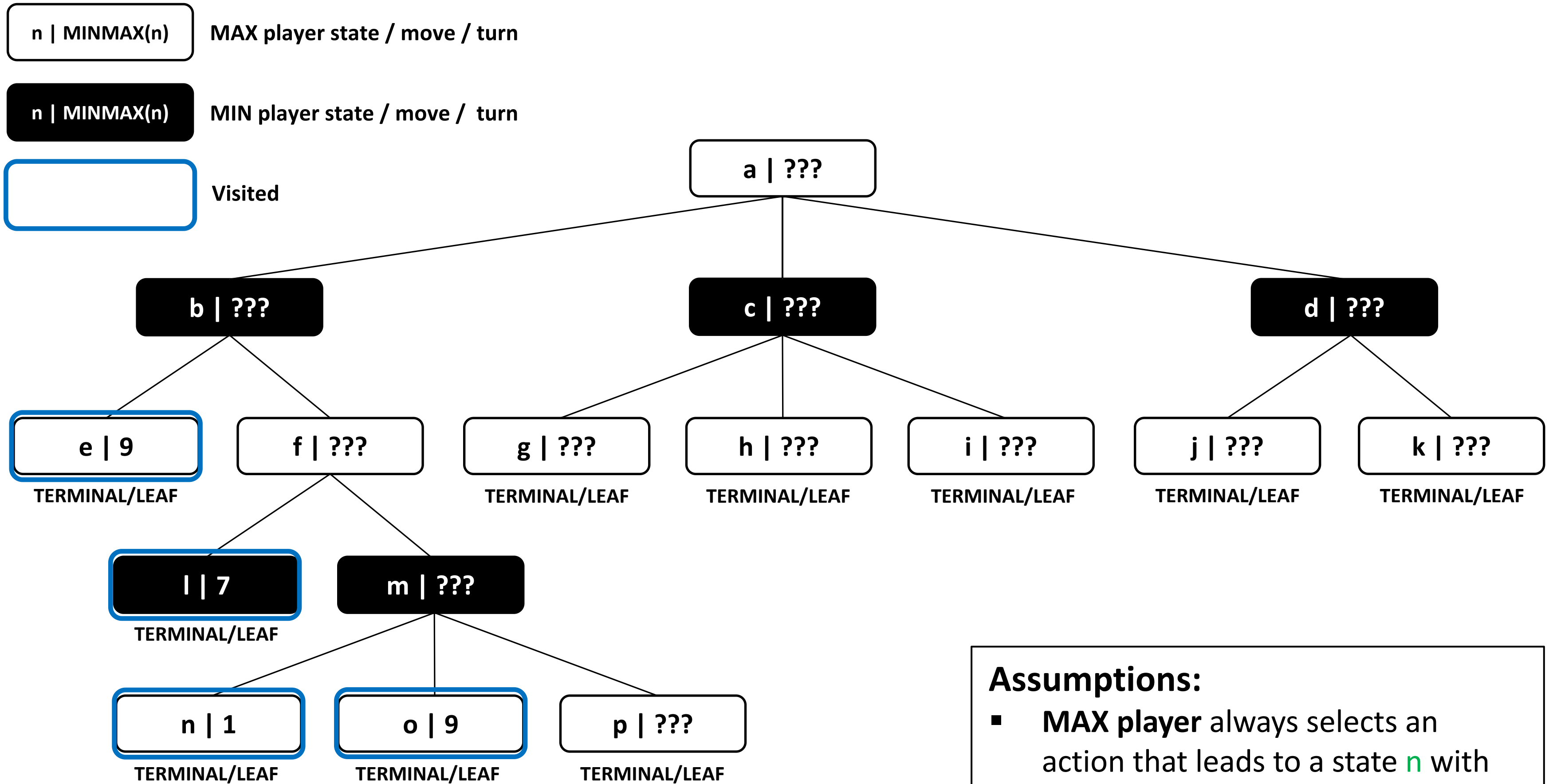
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), \textit{ if } ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \textit{ if } TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \textit{ if } TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



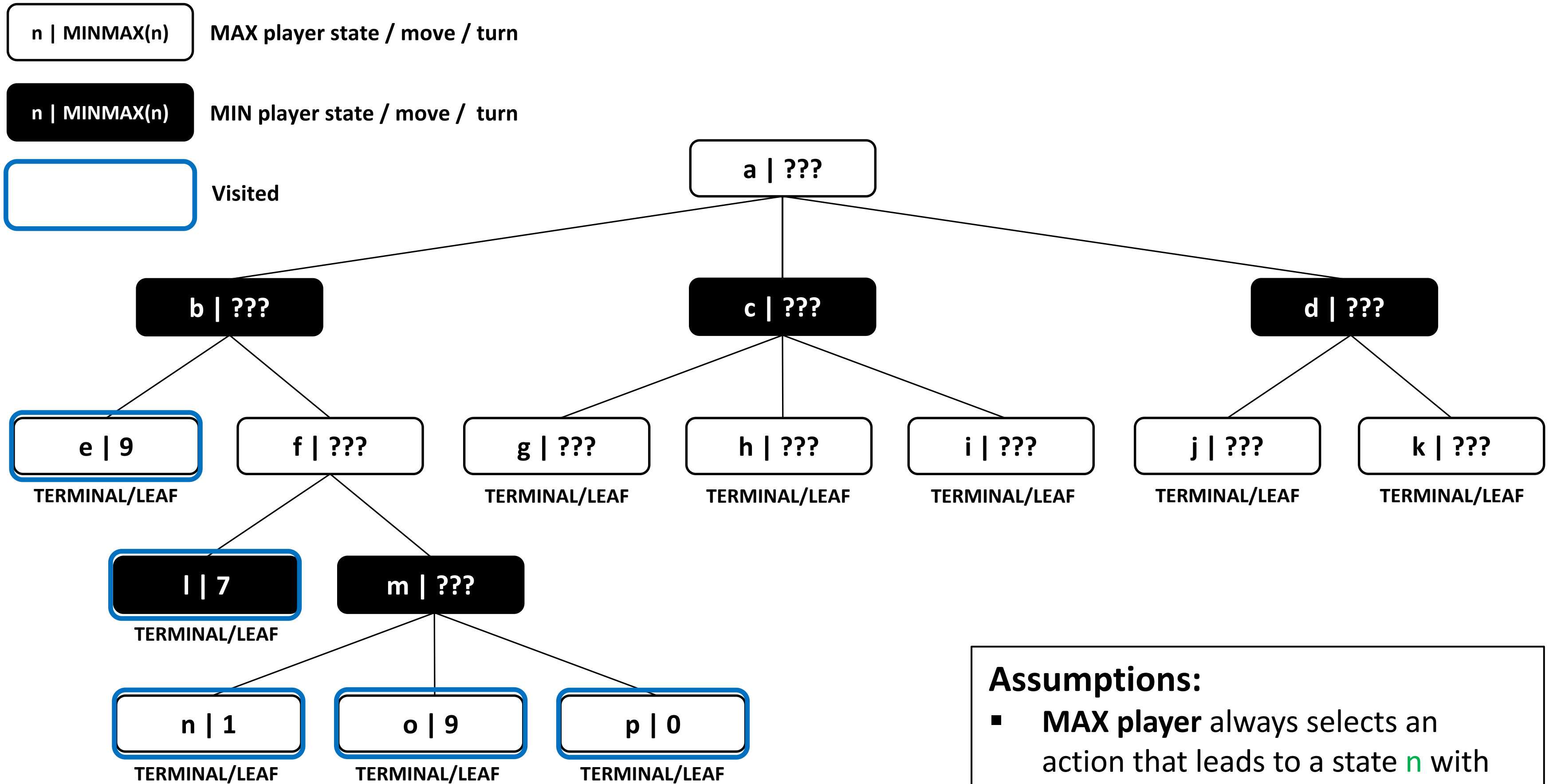
$$MINMAX(n) \; = \; MINMAX(State_n)$$

$$MINMAX(n) \; = \begin{cases} UTILITY(n, MAX), if\ ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n, a), if\ TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n, a), if\ TOMOVE(s) \; = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



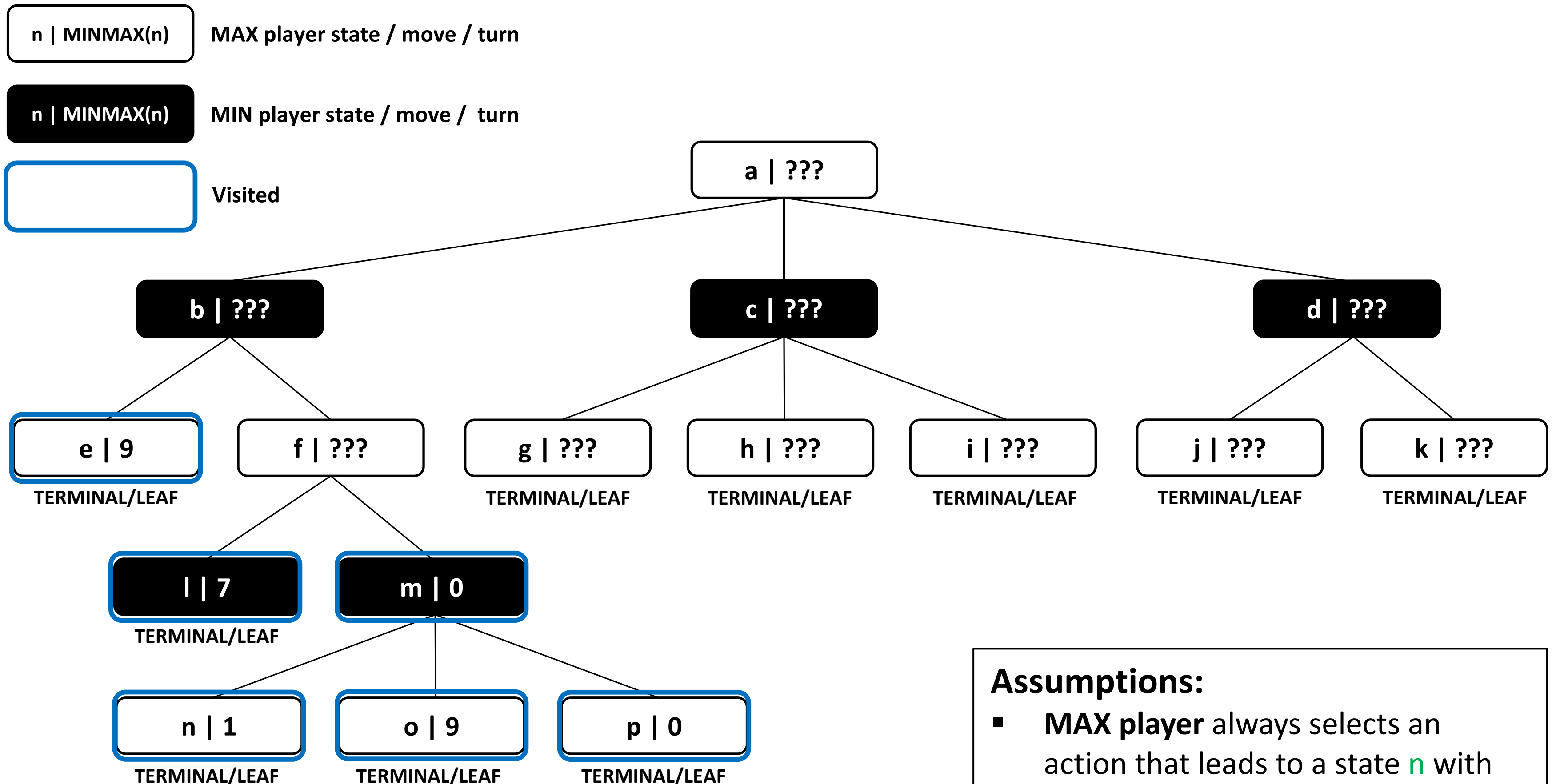
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), \text{ } if \text{ } ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n, a), \text{ } if \text{ } TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n, a), \text{ } if \text{ } TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



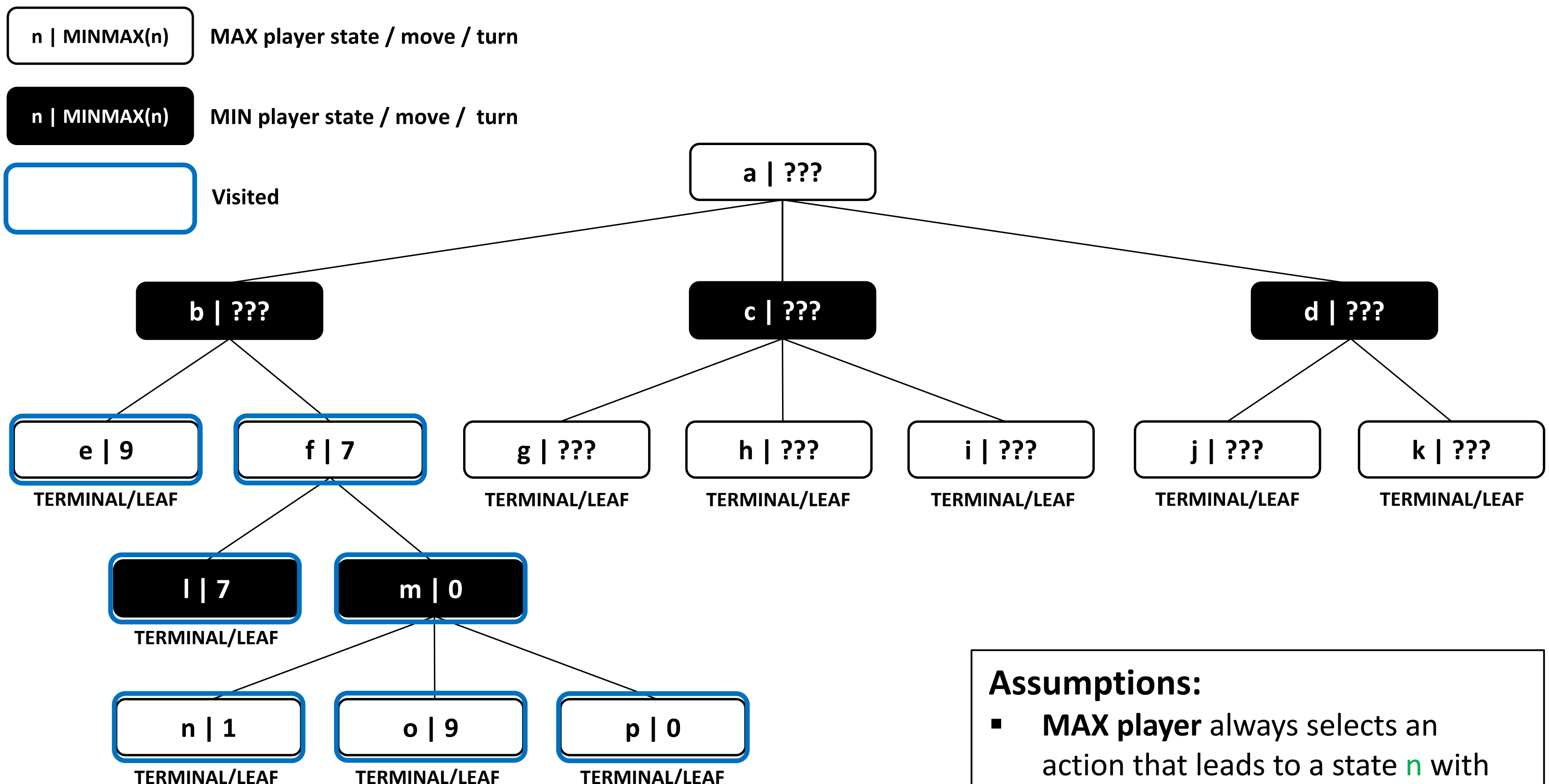
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), \text{ if } ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \text{ if } TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \text{ if } TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



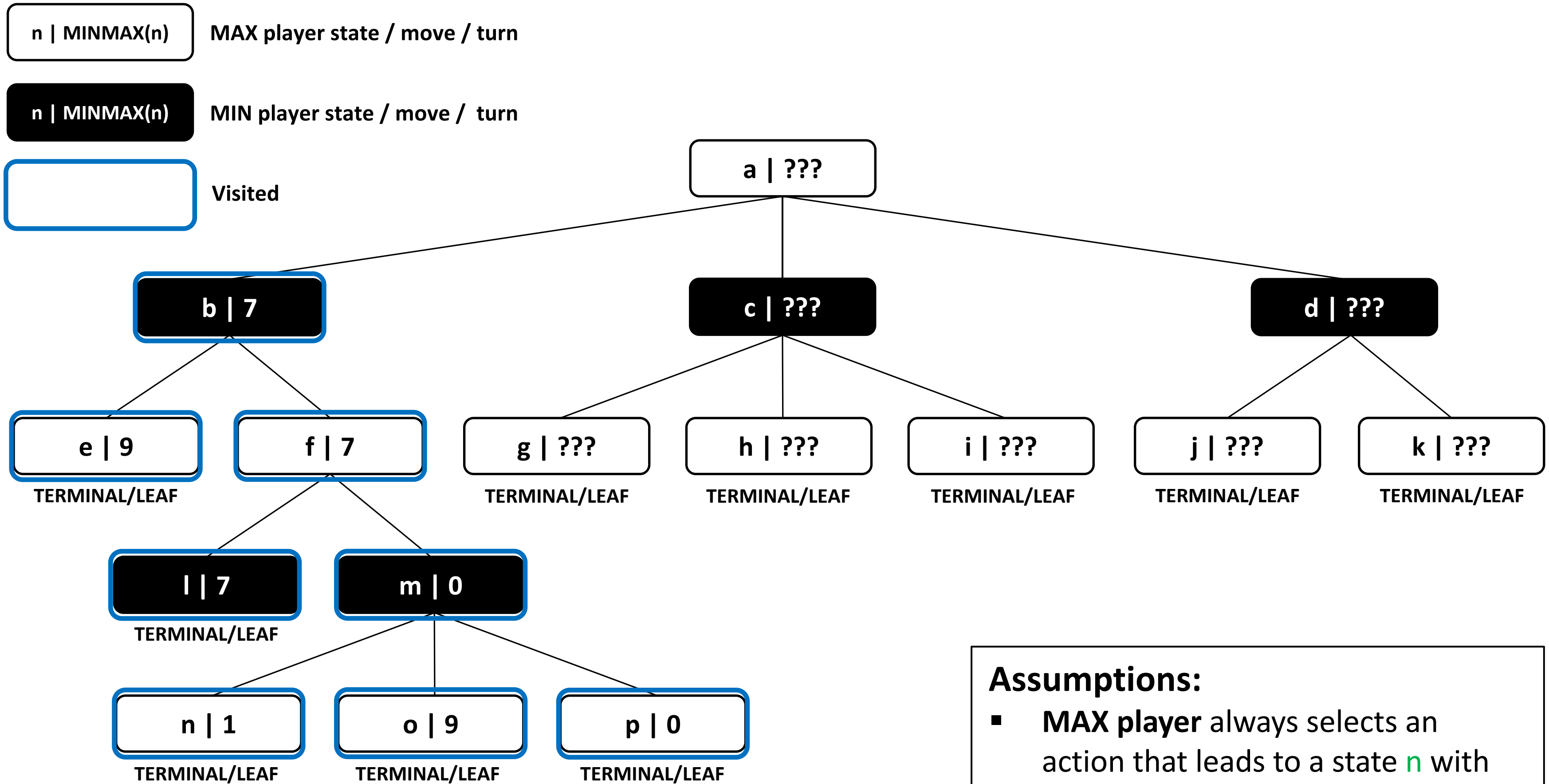
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), if\ ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), if\ TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), if\ TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



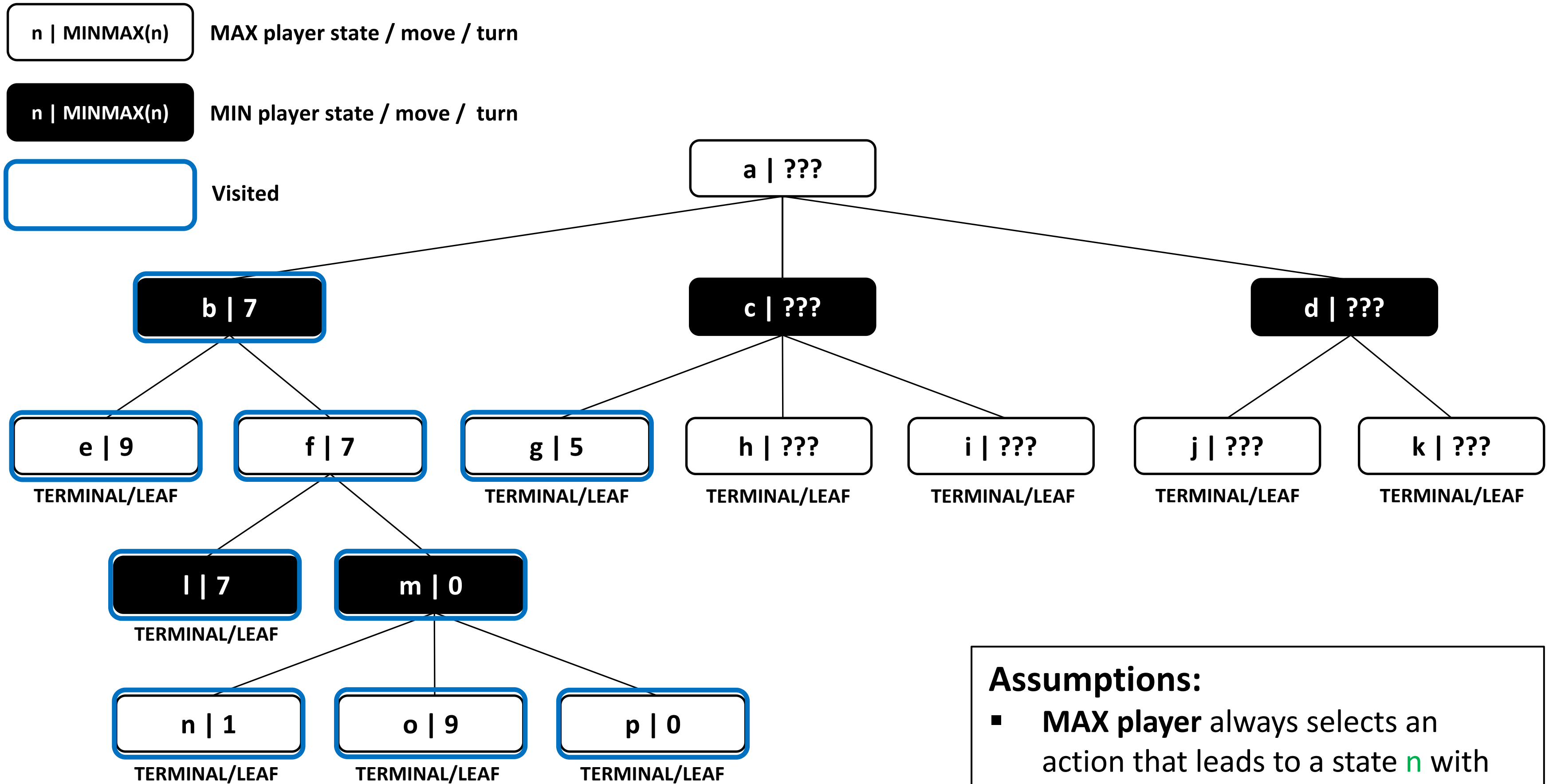
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), if\ ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n, a), if\ TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n, a), if\ TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



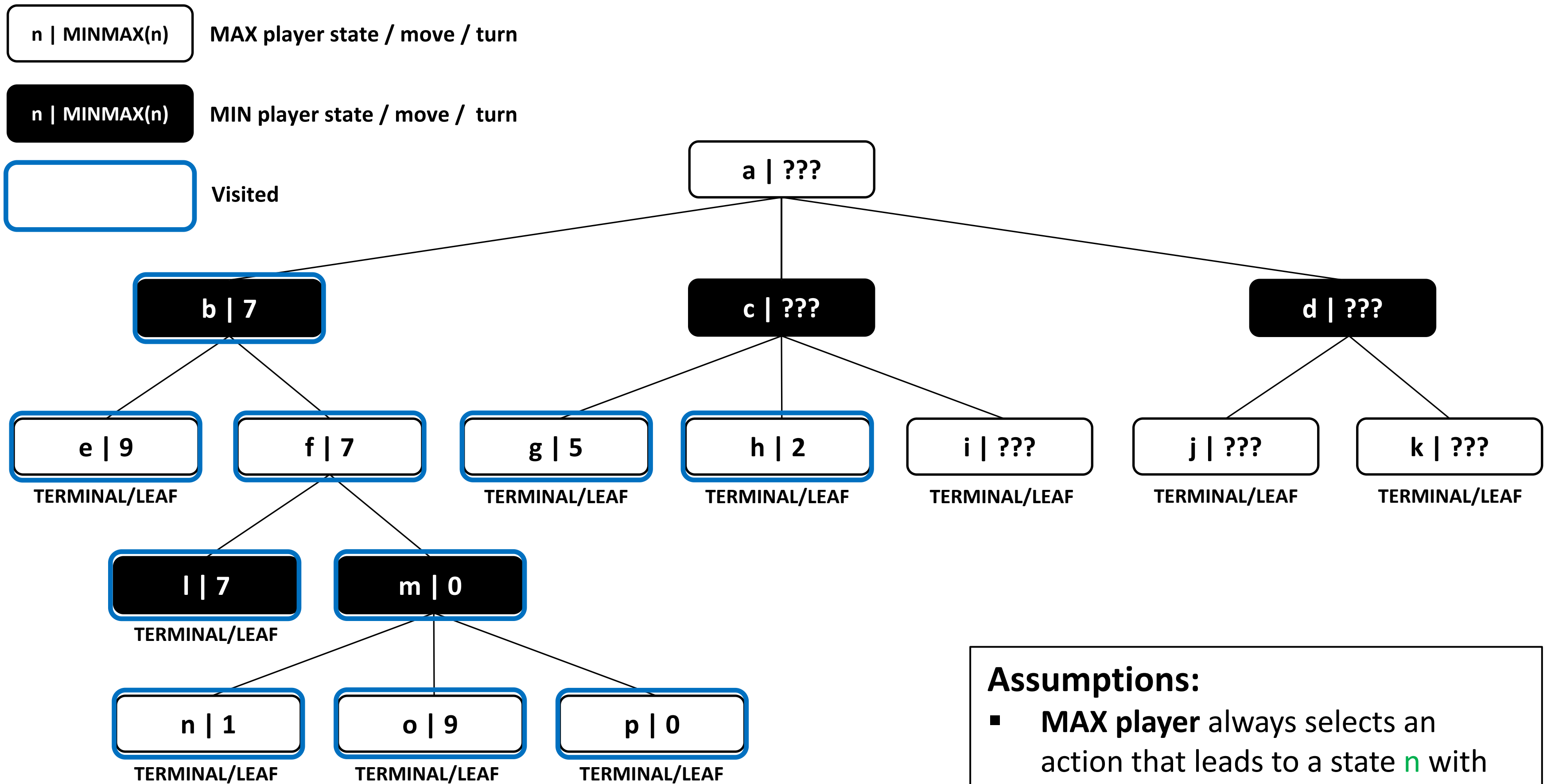
$$MINMAX(n) \ = \ MINMAX(State_n)$$

$$MINMAX(n) \ = \begin{cases} UTILITY(n, MAX), if \ ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), if \ TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), if \ TOMOVE(s) \ = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



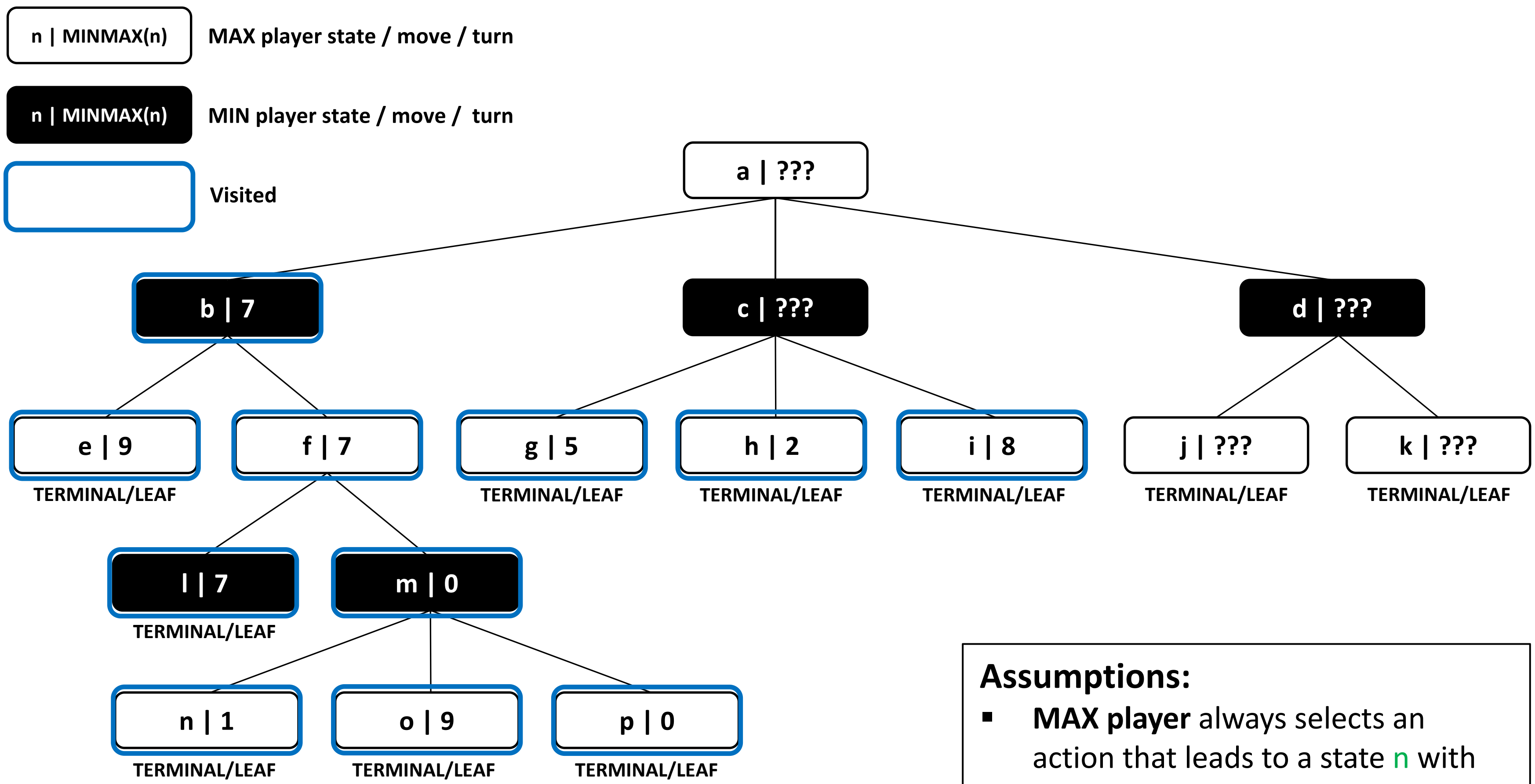
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), if\ ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), if\ TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), if\ TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



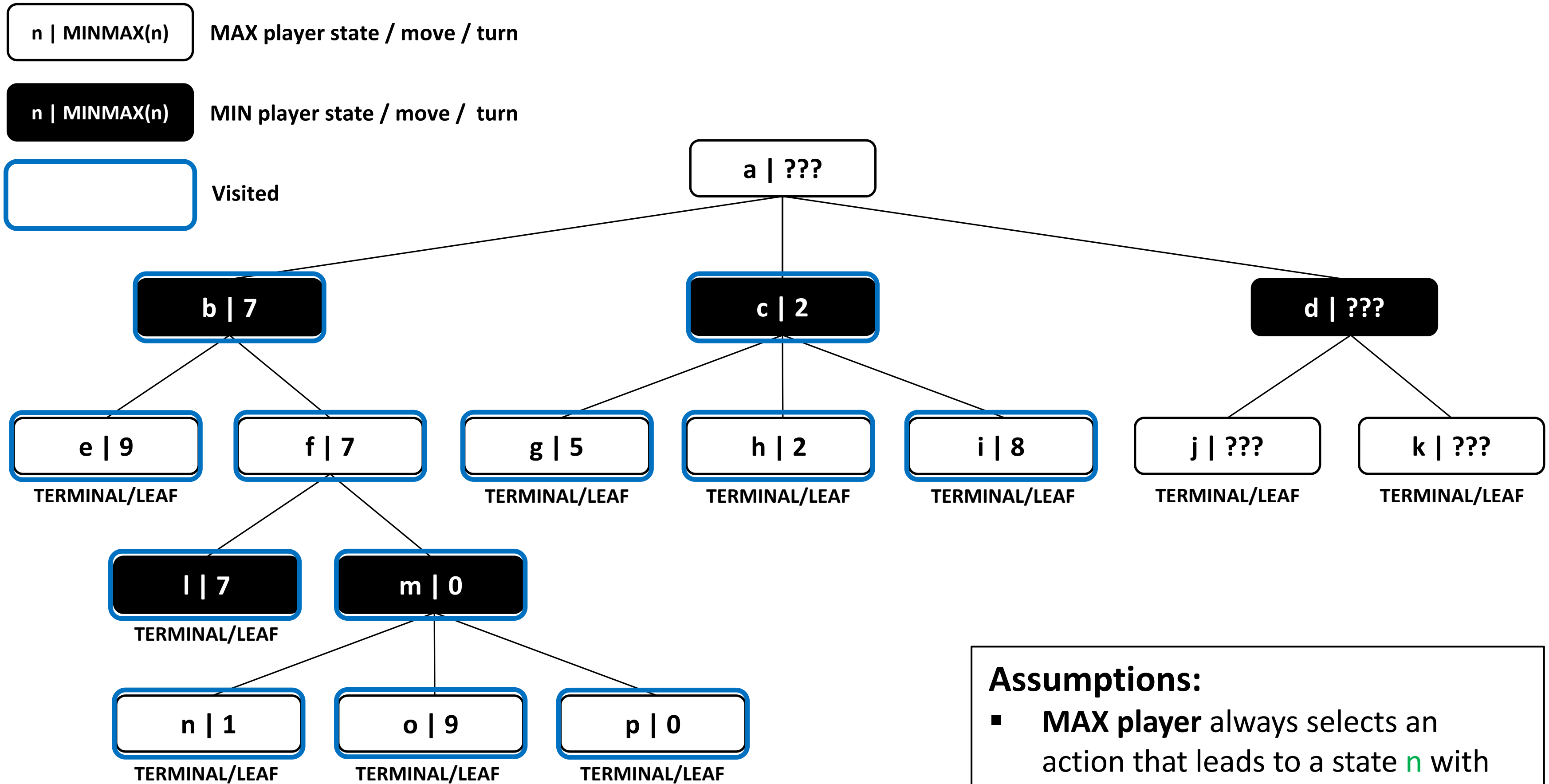
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), if\ ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), if\ TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), if\ TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



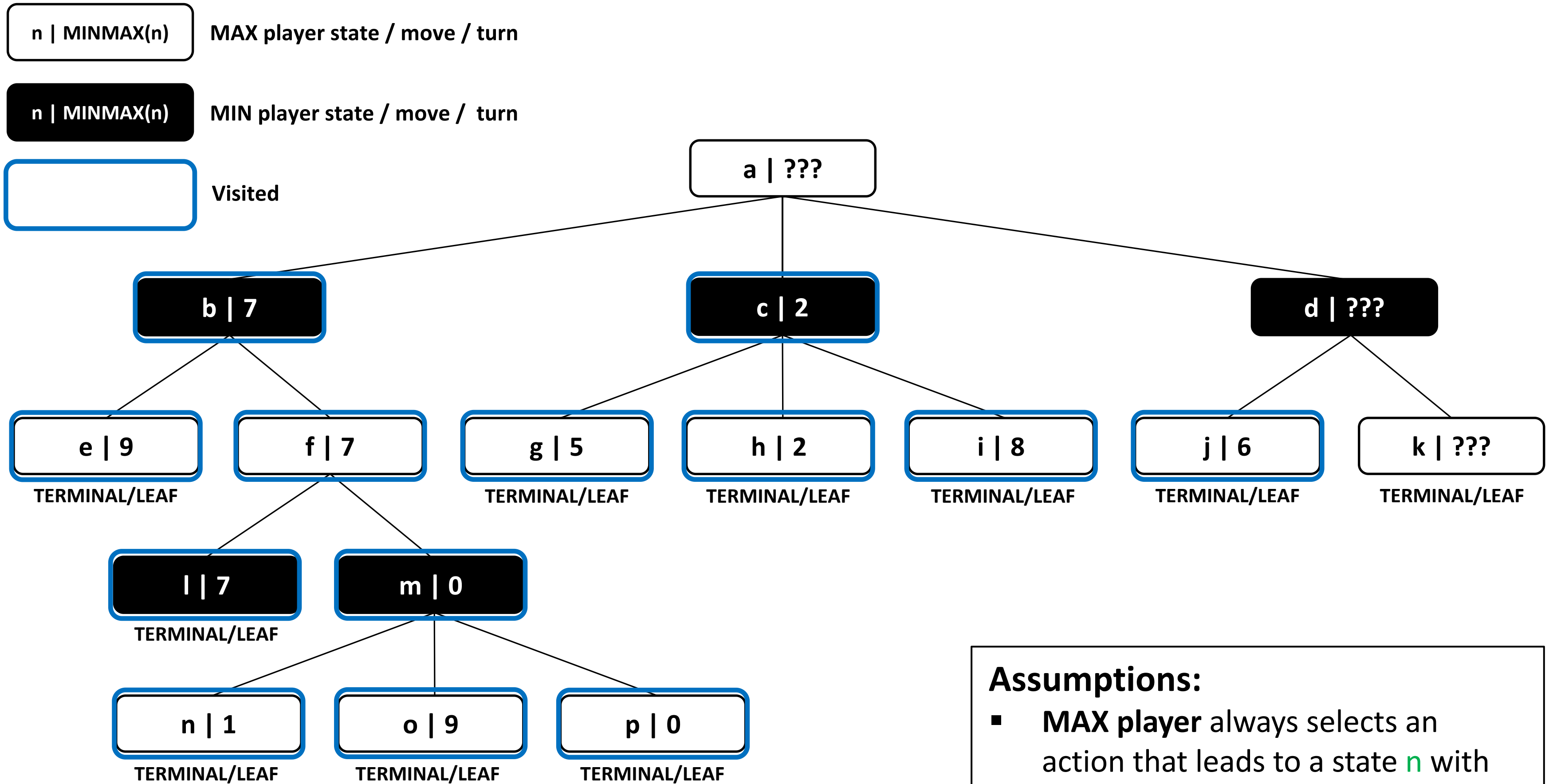
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), \text{ if } ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n, a), \text{ if } TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n, a), \text{ if } TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



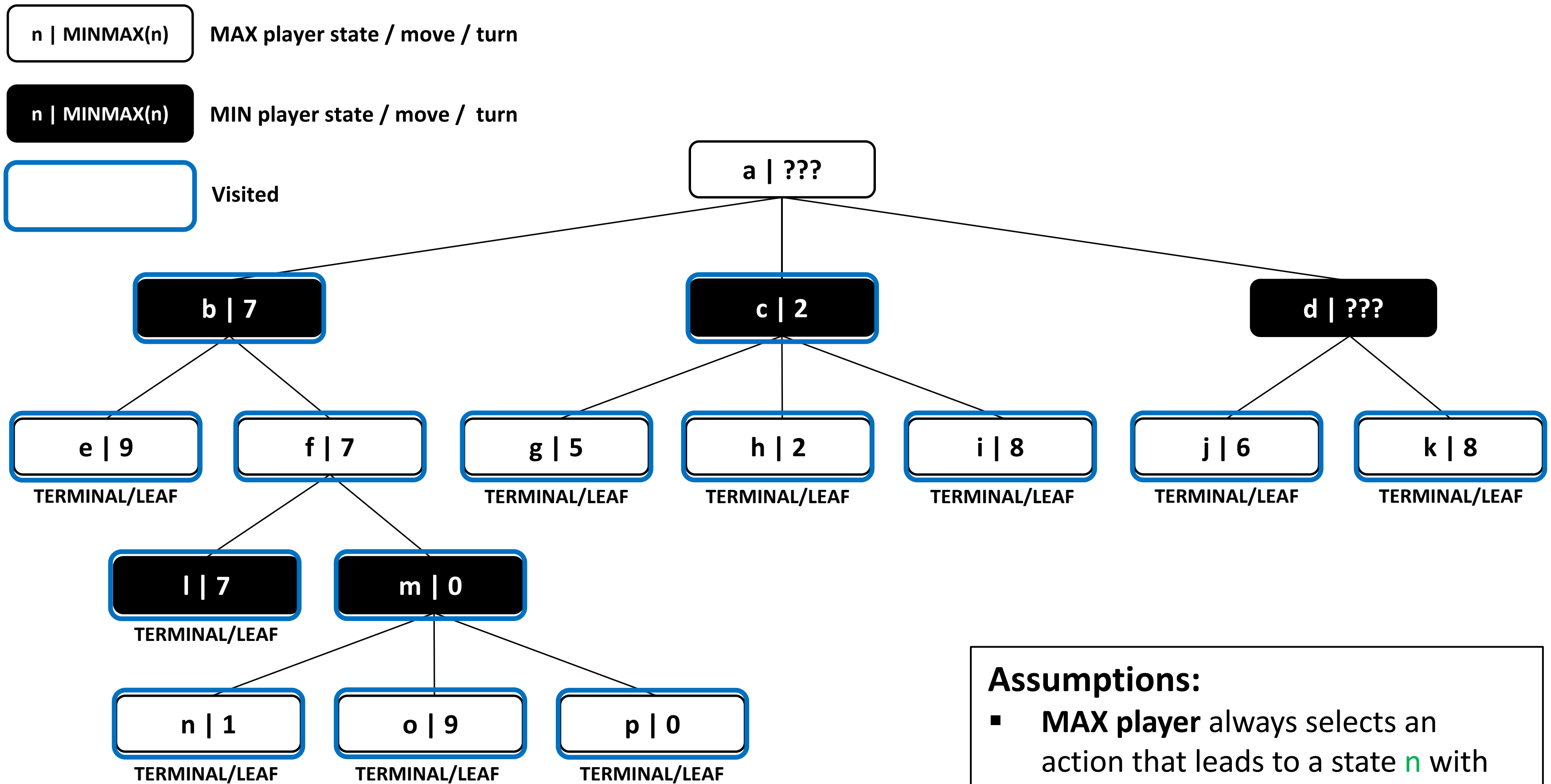
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), \text{ if } ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a)), \text{ if } TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a)), \text{ if } TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



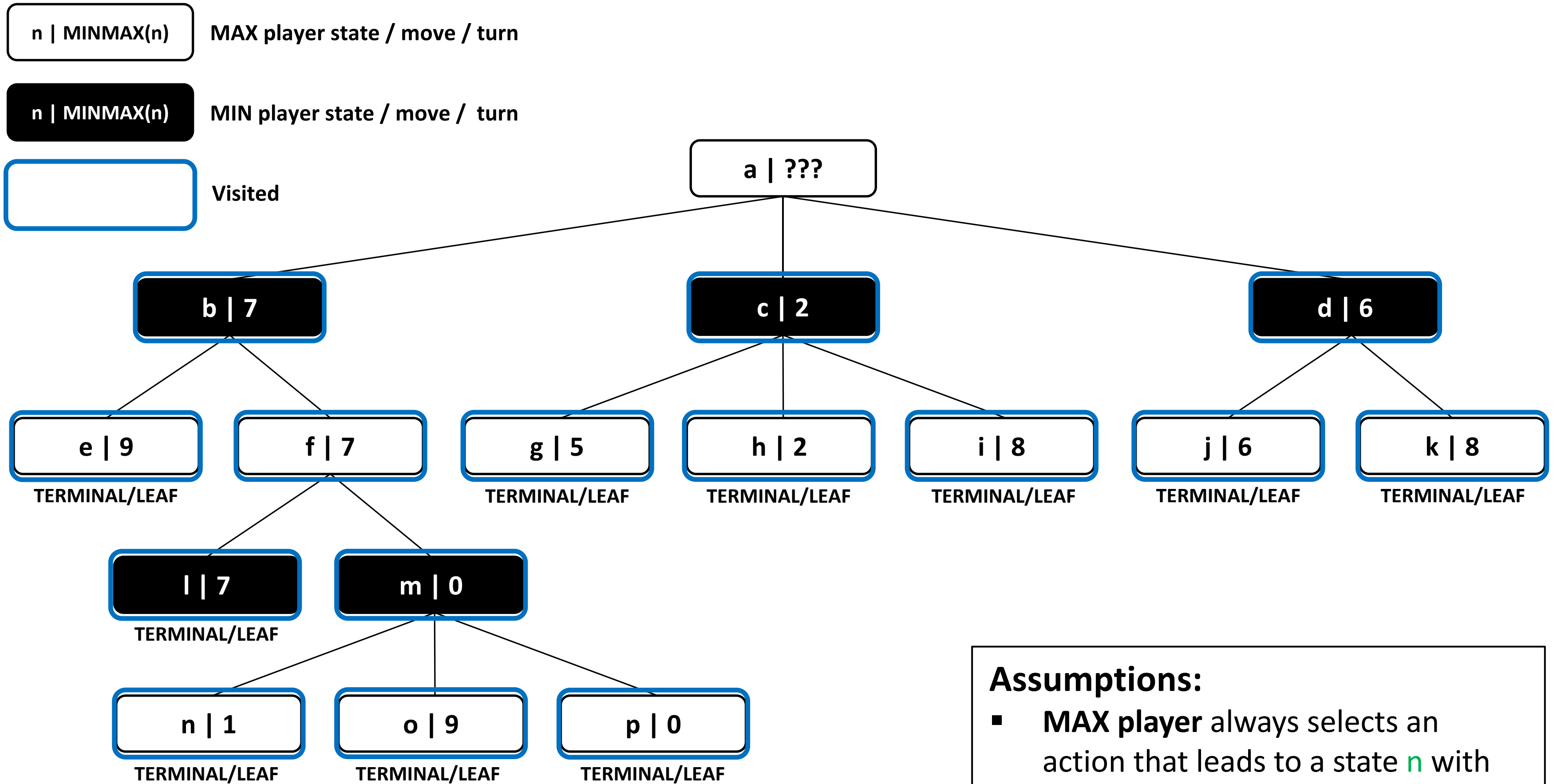
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), \text{ if } ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \text{ if } TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \text{ if } TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



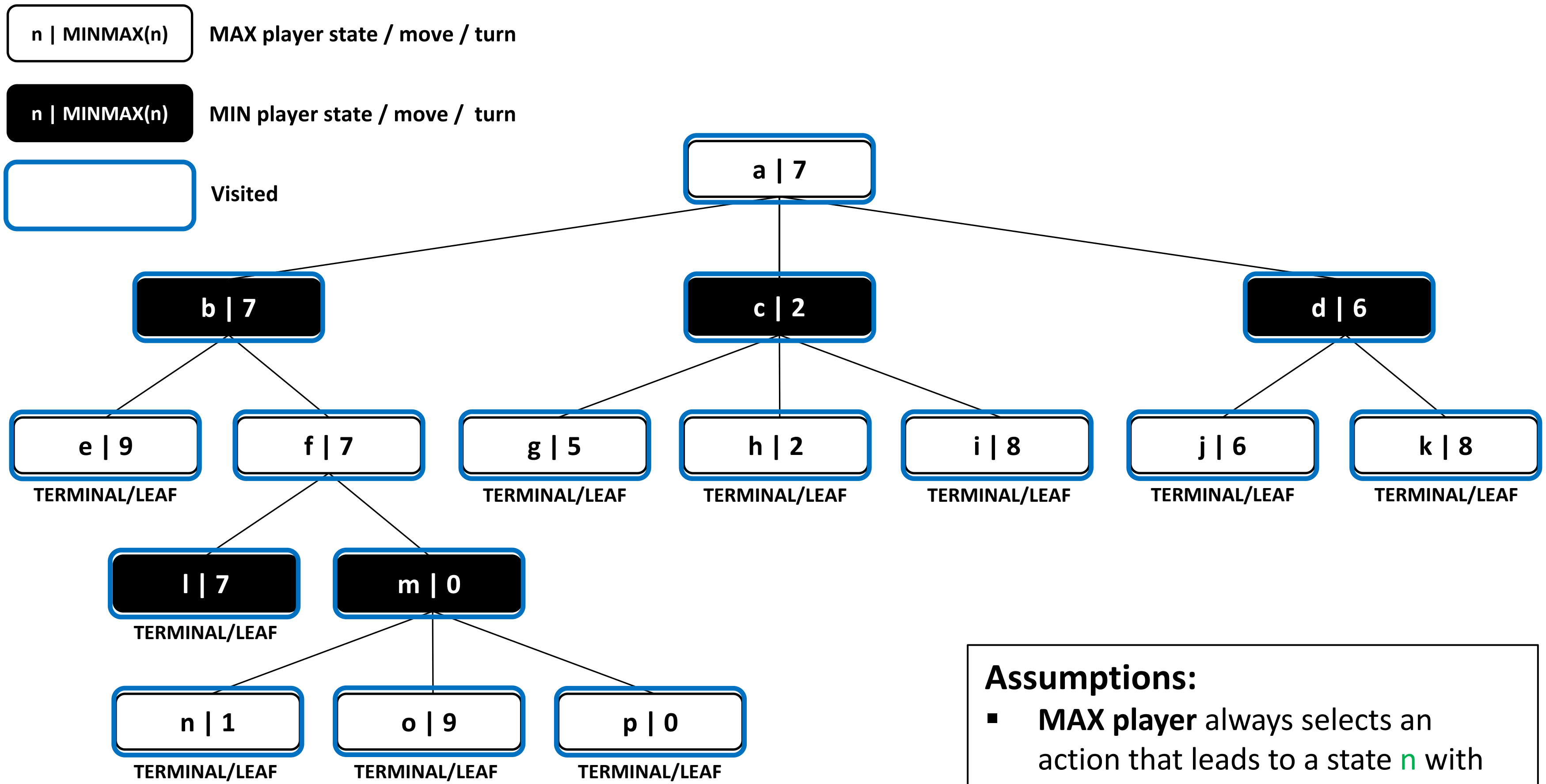
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), \text{ if } ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a)), \text{ if } TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a)), \text{ if } TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



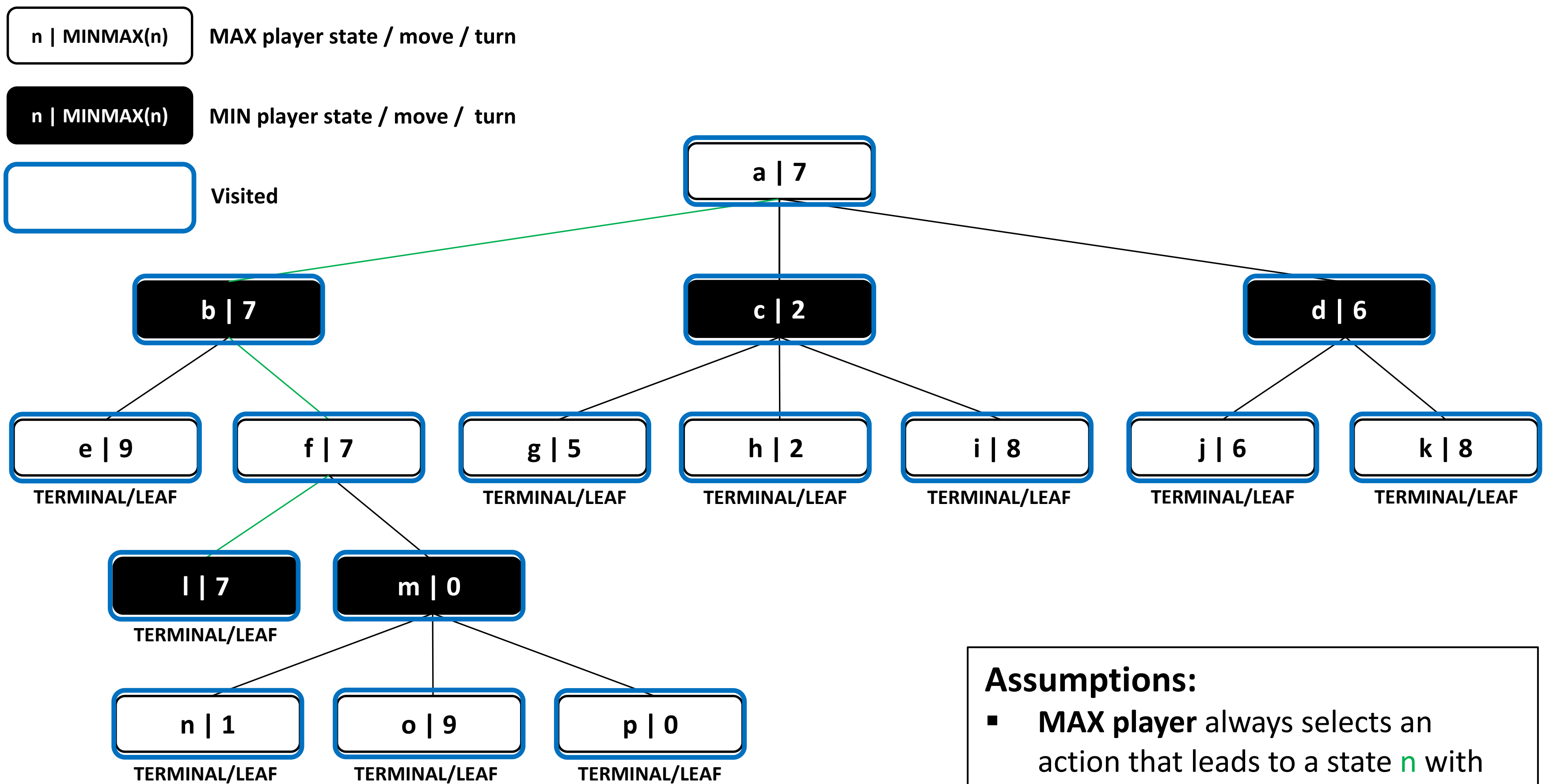
$$MINMAX(n) = MINMAX(State_n)$$

$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), \; if \; ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \; if \; TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \; if \; TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

Example MinMax Search Tree



$$MINMAX(n) = MINMAX(State_n)$$

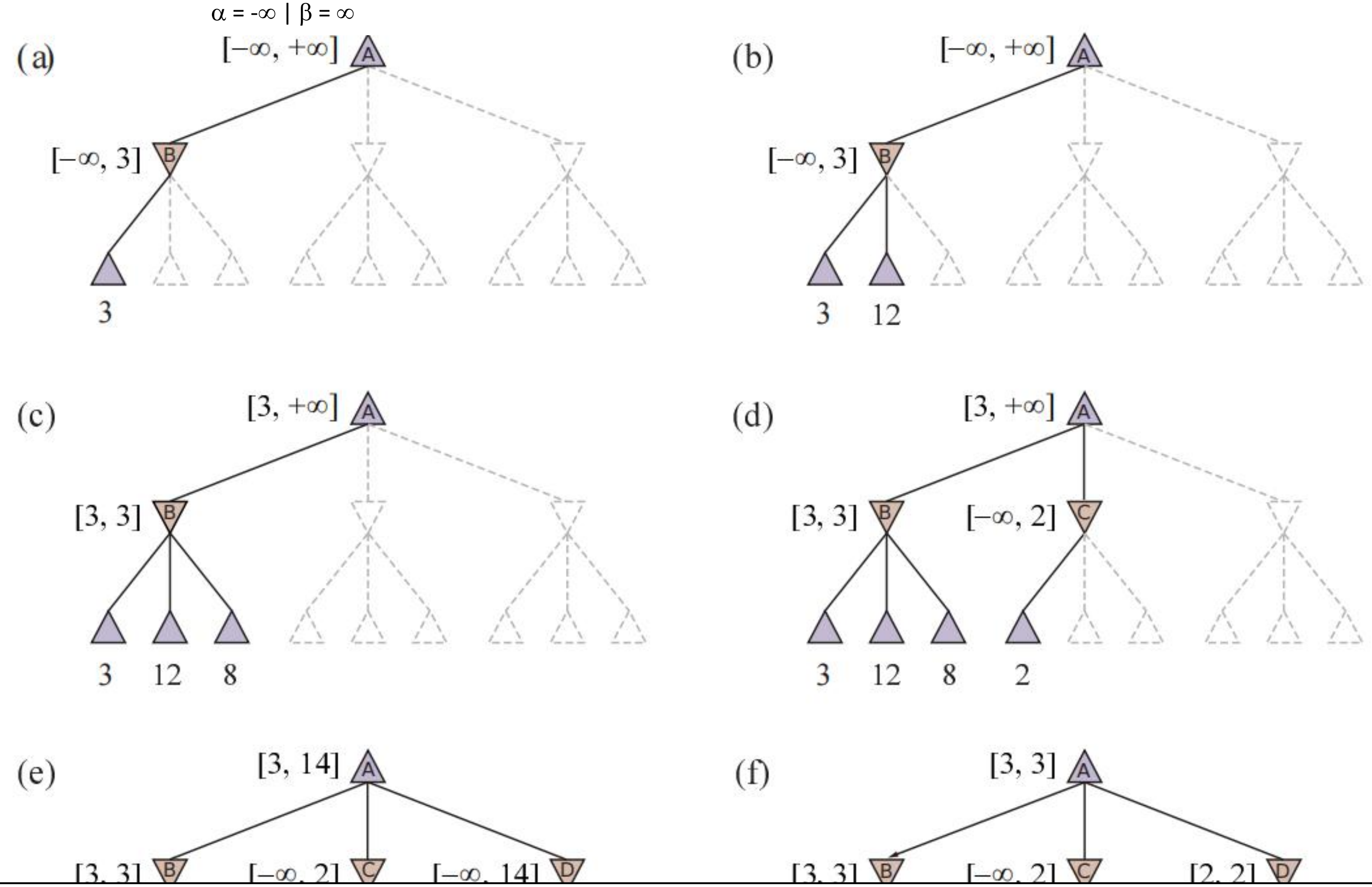
$$MINMAX(n) = \begin{cases} UTILITY(n, MAX), \text{ if } ISTERMINAL(n) \\ max_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \text{ if } TOMOVE(s) = MAX \\ min_{a \in ACTIONS(n)} MINMAX(RESULT(n,a), \text{ if } TOMOVE(s) = MIN \end{cases}$$

Assumptions:

- **MAX player** always selects an action that leads to a state n with **maximum** MINIMAX(n) value
- **MIN player** always selects an action that leads to a state n with **minimum** MINIMAX(n) value
- **BOTH players** always play optimally

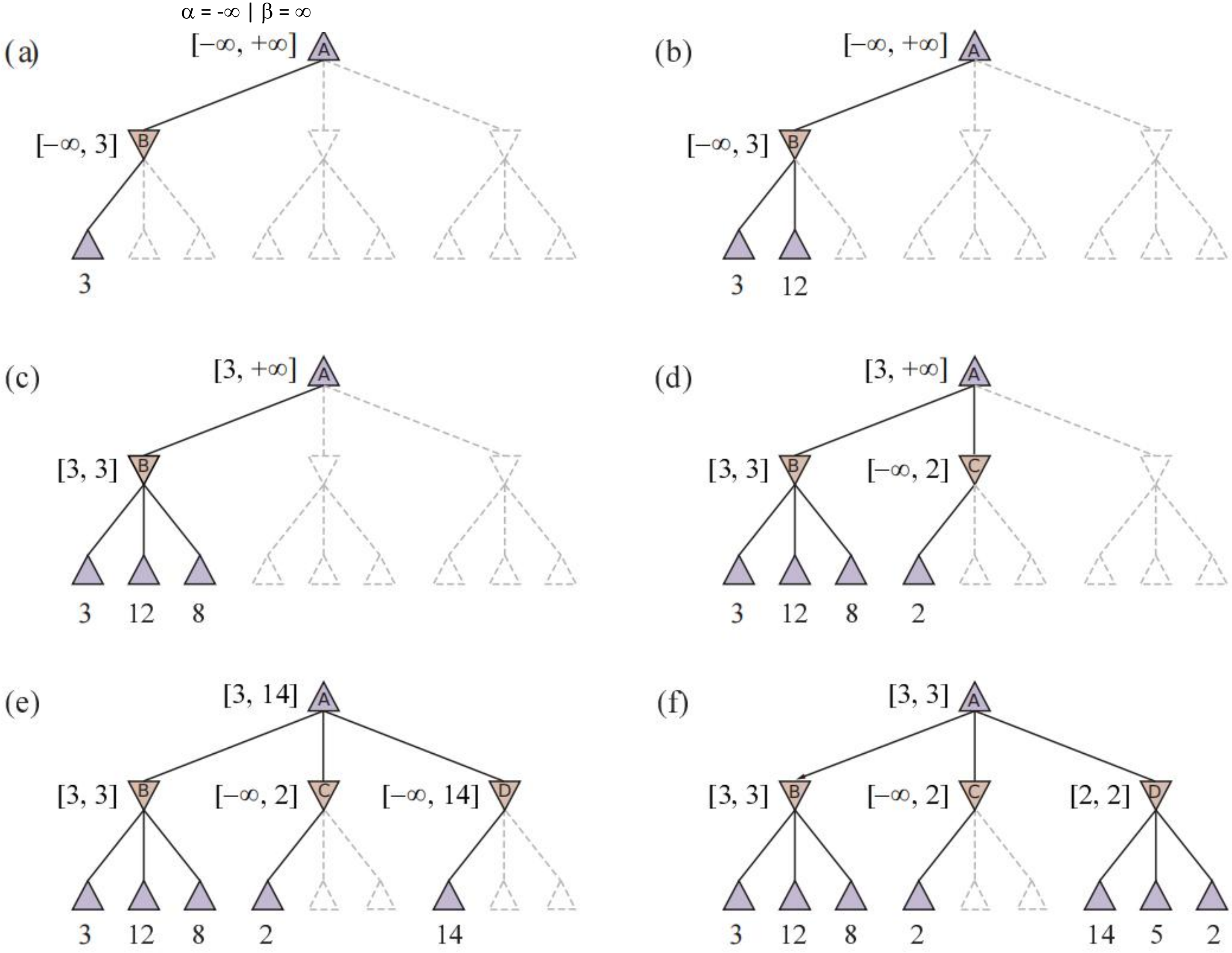
MinMax: What is the Challenge?

Example MinMax with α-β Pruning

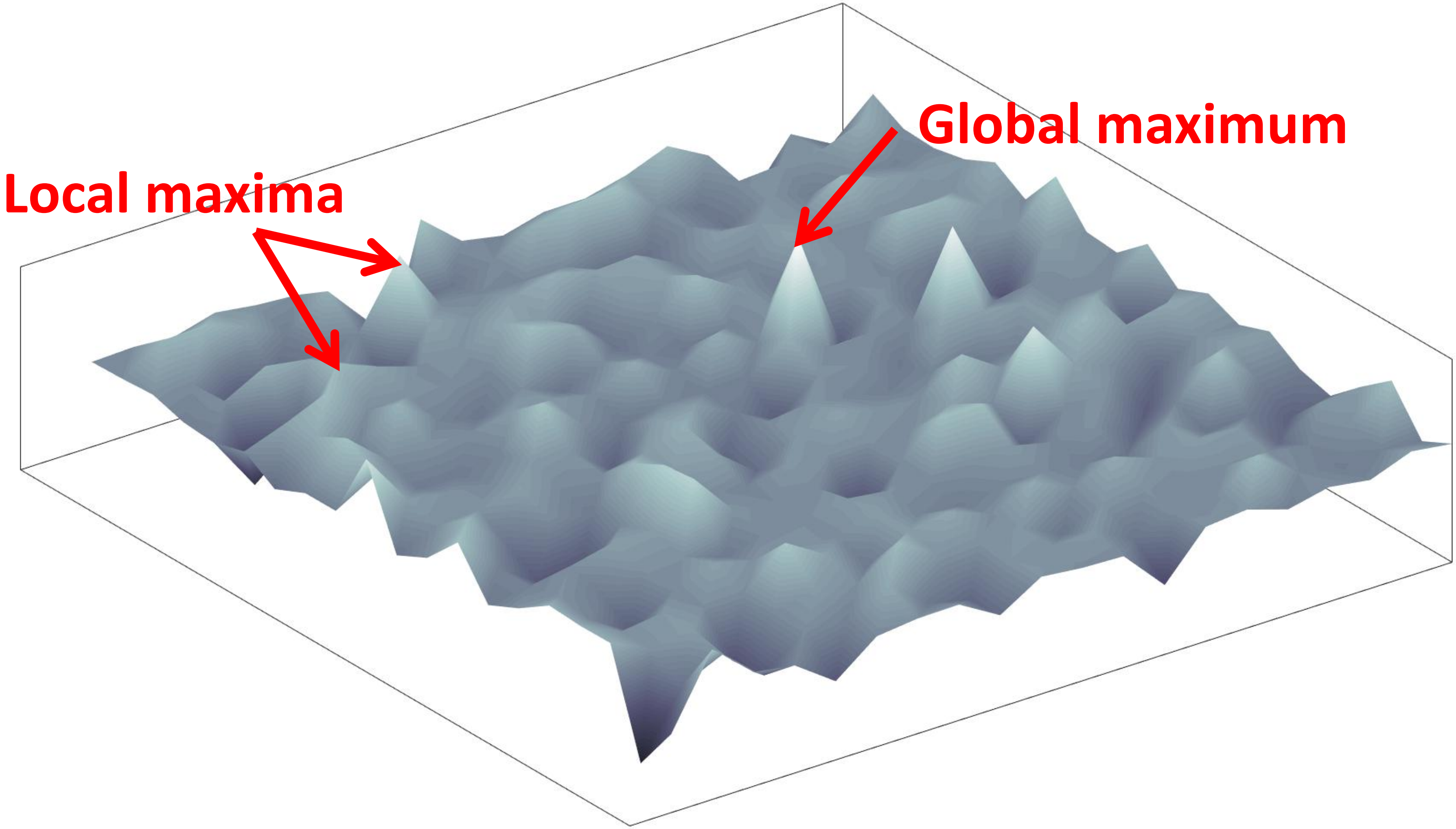


α: the value of the best (highest-value) choice we have found so far at any choice point along the path for MAX player ("at least")
β: the value of the best (lowest-value) choice we have found so far at any choice point along the path for MIN player ("at most")

Example MinMax with α-β Pruning

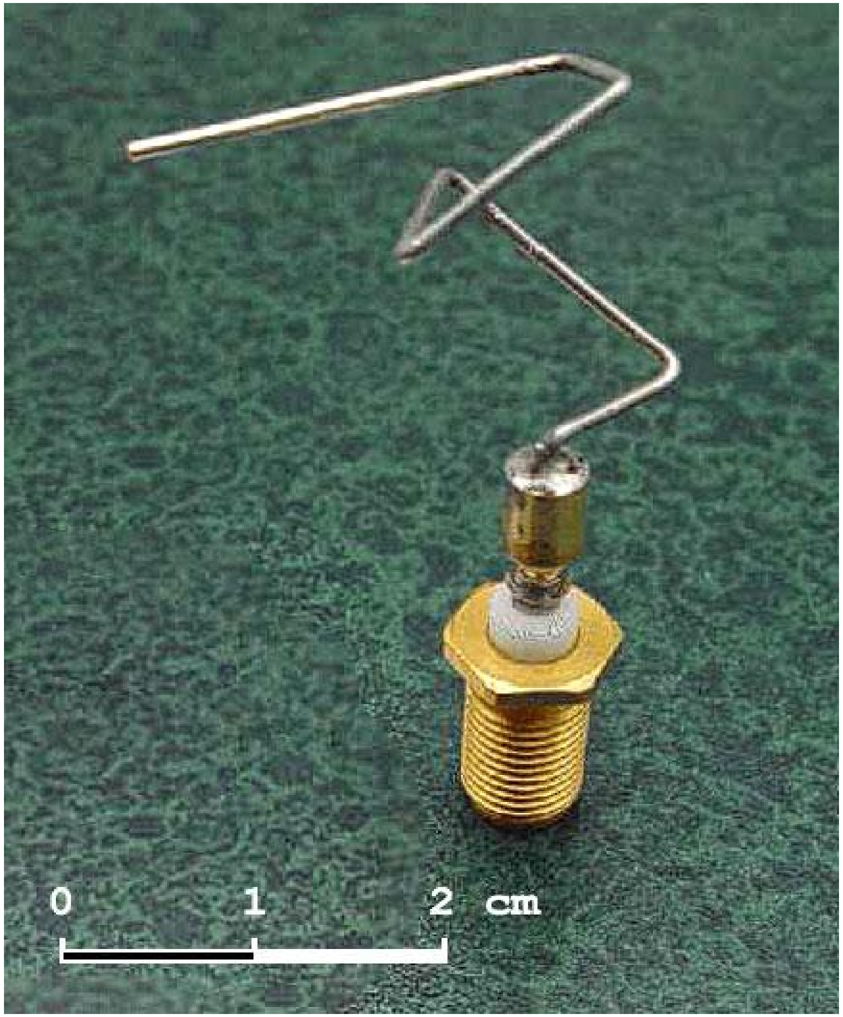


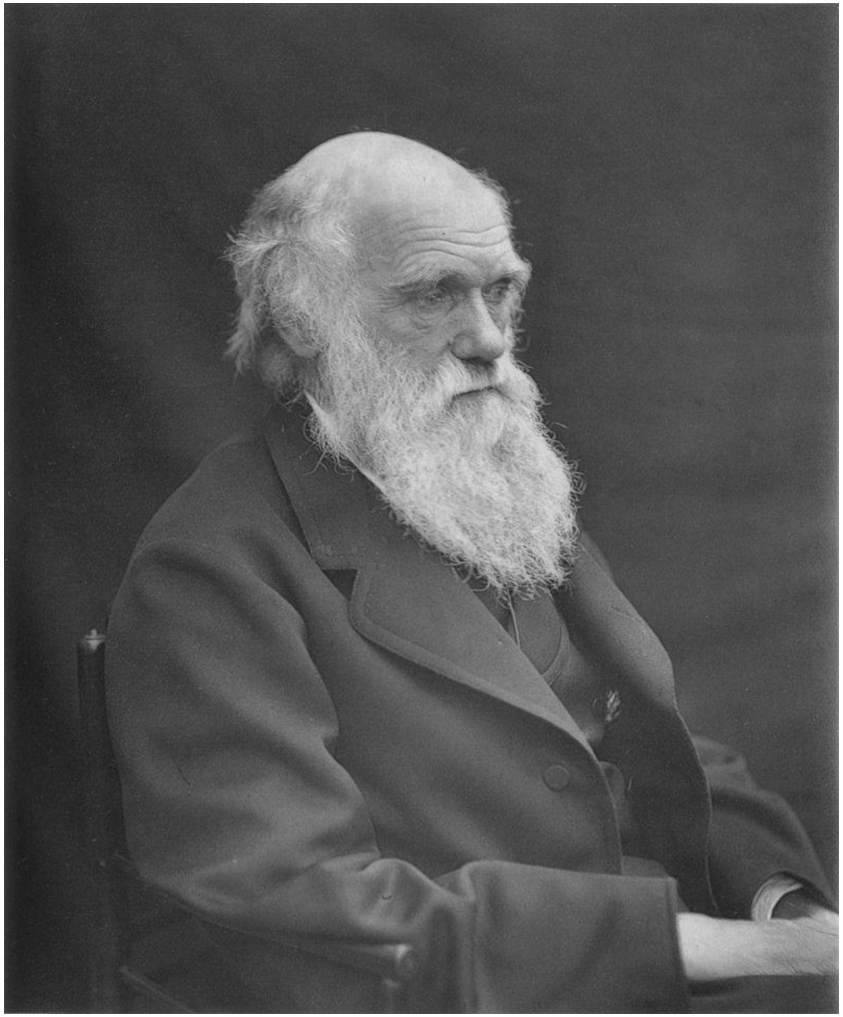
Complex Environments



Bonus Material
Chapter 4 - related
(NOT ON EXAMS!)
Search in Complex Environments

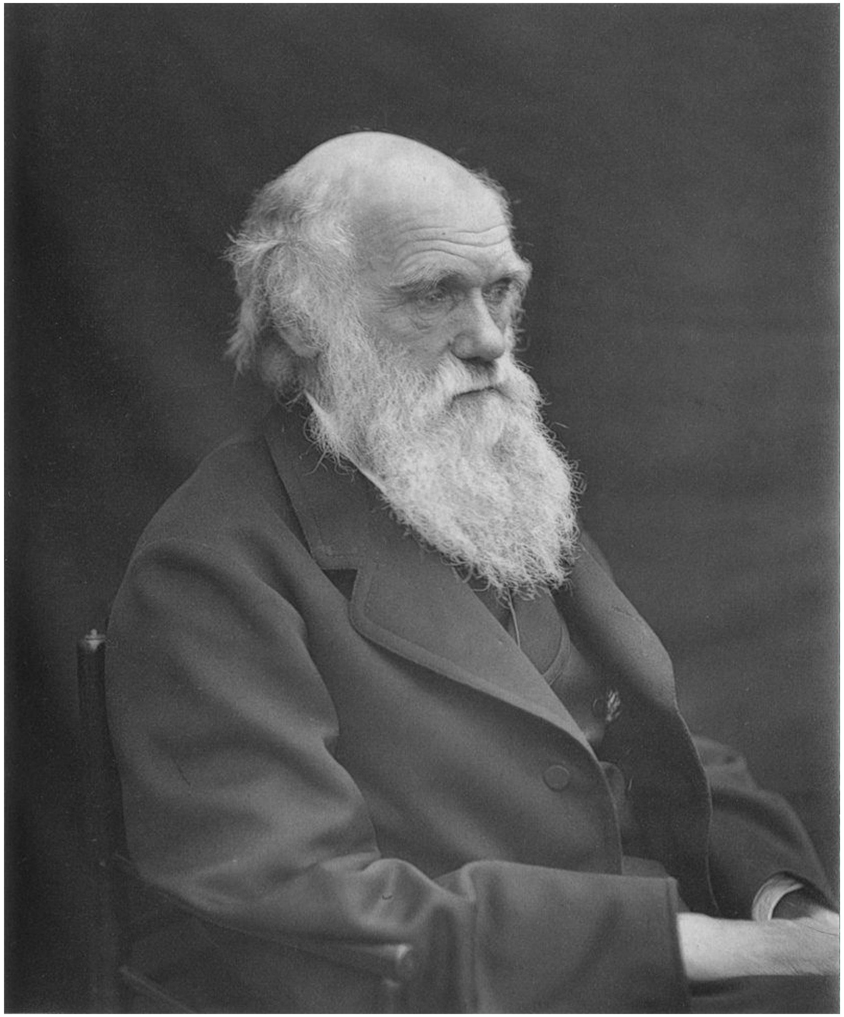
What's the Connection Here?





Source: https://wikipedia.org/

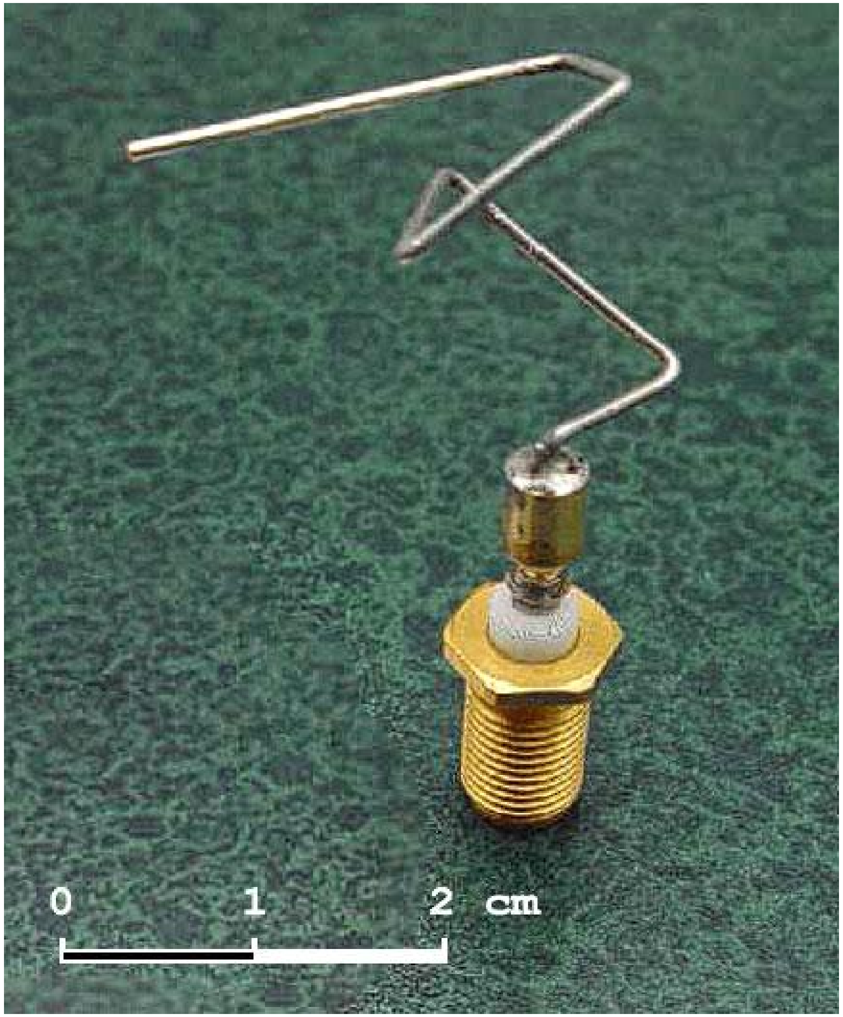
Charles Darwin



Source: https://wikipedia.org/

Charles Robert Darwin was an English naturalist, geologist and biologist, best known for his contributions to the science of evolution. His proposition that all species of life have descended over time from common ancestors is now widely accepted, and considered a foundational concept in science.

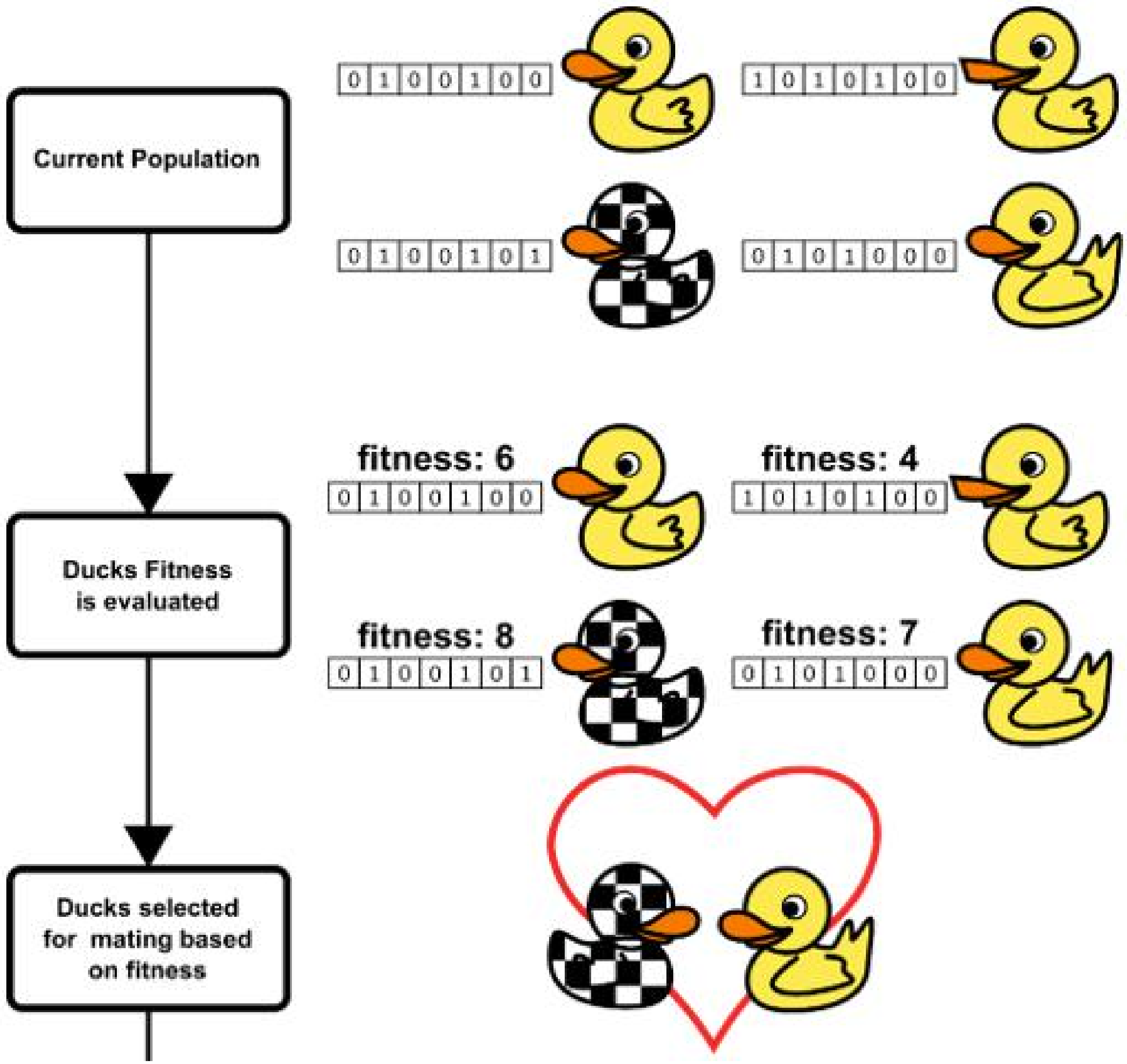
Evolved Antenna



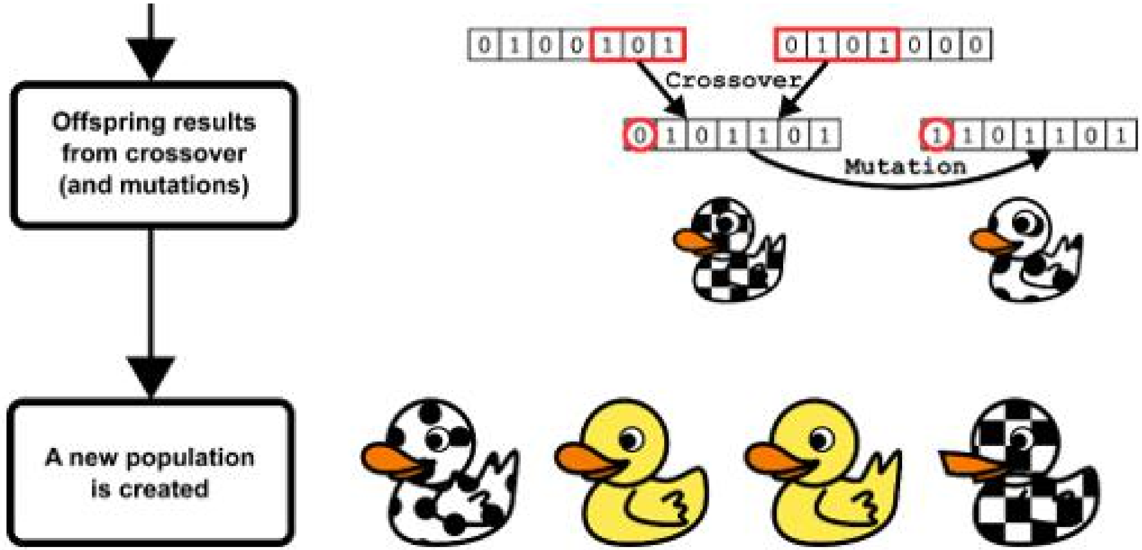
Source: https://wikipedia.org/

An evolved antenna is an antenna designed fully or substantially by an automatic computer design program that uses an evolutionary algorithm that mimics Darwinian evolution.

Genetic Algorithm: The Idea



Genetic Algorithm: The Idea



Source: https://livebook.manning.com/book/algorithms-and-data-structures-in-action/chapter-18/v-14/102

Genetic Algorithm: Example

Population of points (solutions)

x y

0	1	0	1	0	1

x y

0	1	1	1	0	1

x y

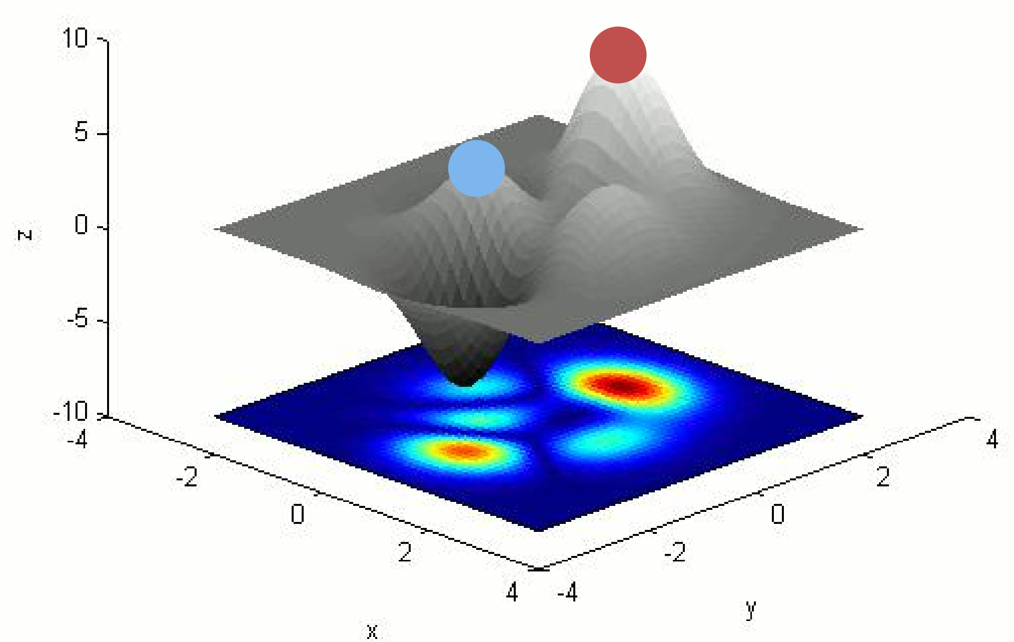
0	0	0	1	0	1

x y

0	1	0	1	0	0

x y

0	0	1	0	0	0

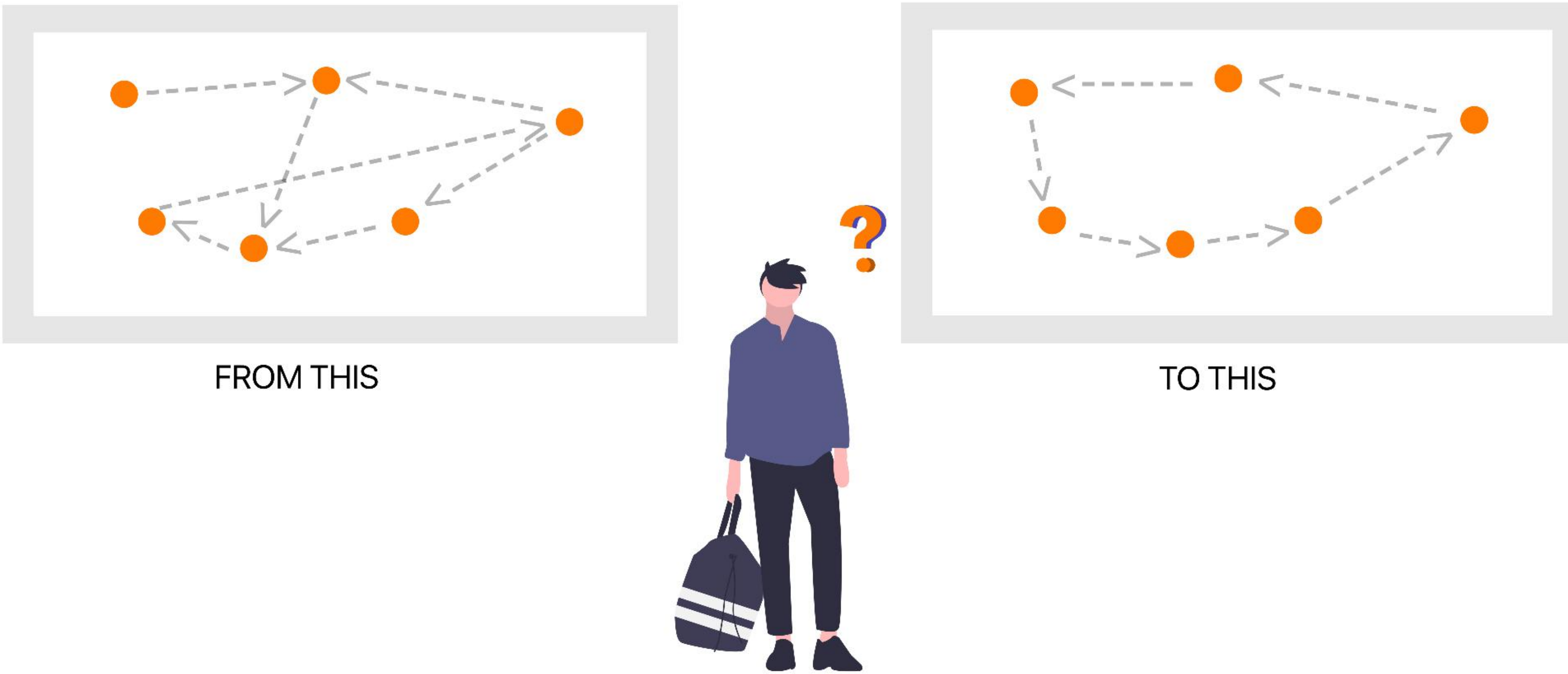


y = f(x,y) - fitness function

"Good enough" / local maximum

Best / global maximum

Traveling Salesman Problem



A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. What is the shortest possible route that he visits each city exactly once and returns to the origin city?

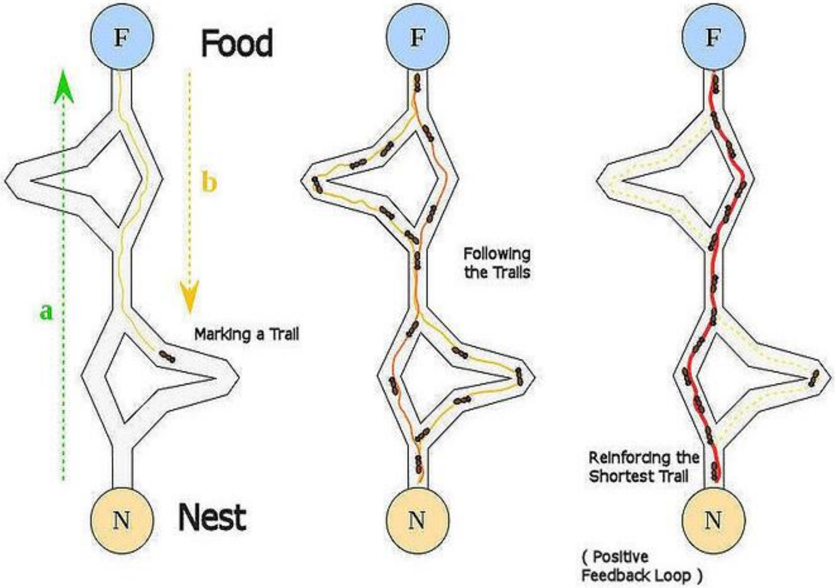
N cities → (N-1)!/2 paths | 15 cities → 43589145600 paths

Source: https://medium.com/ivymobility-developers/traveling-salesman-problem-9ab623c88fab

Example: Genetic Algorithm

http://ostap0207.github.io/web-ga-tsp/

Ant Colony Optimization: The Idea

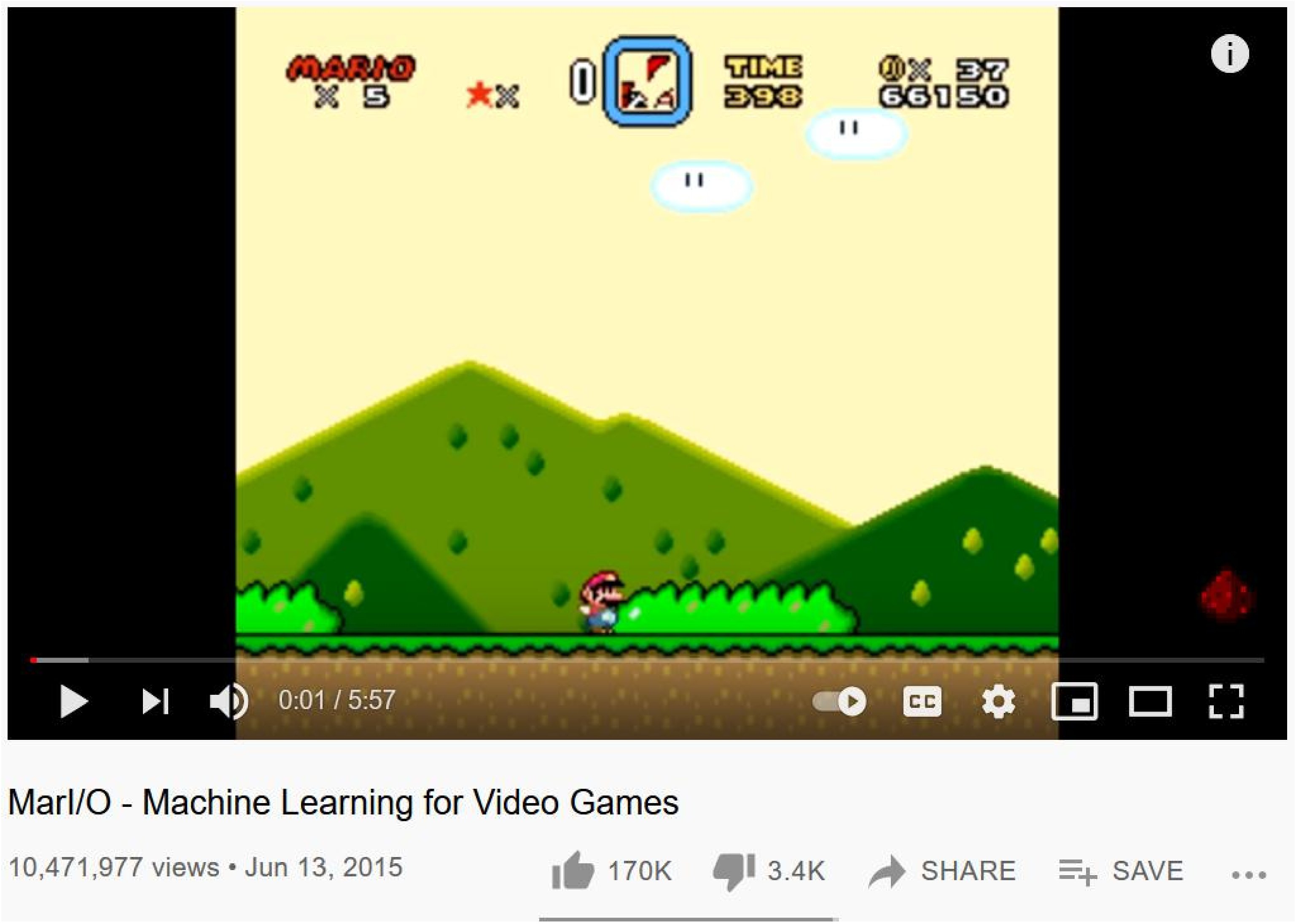


Source: https://wikipedia.org/

Example: Ant Colony Optimization

https://courses.cs.ut.ee/demos/visual-aco/

Genetic Algorithm in Action



Source: https://www.youtube.com/watch?v=qv6UVOQ0F44

Bonus DEFINITELY OPTIONAL Material