ALGORITHMS FOR NP-HARD PROBLEMS

Part I – Heuristic Search

Lecture 1: Introduction to Heuristic Search

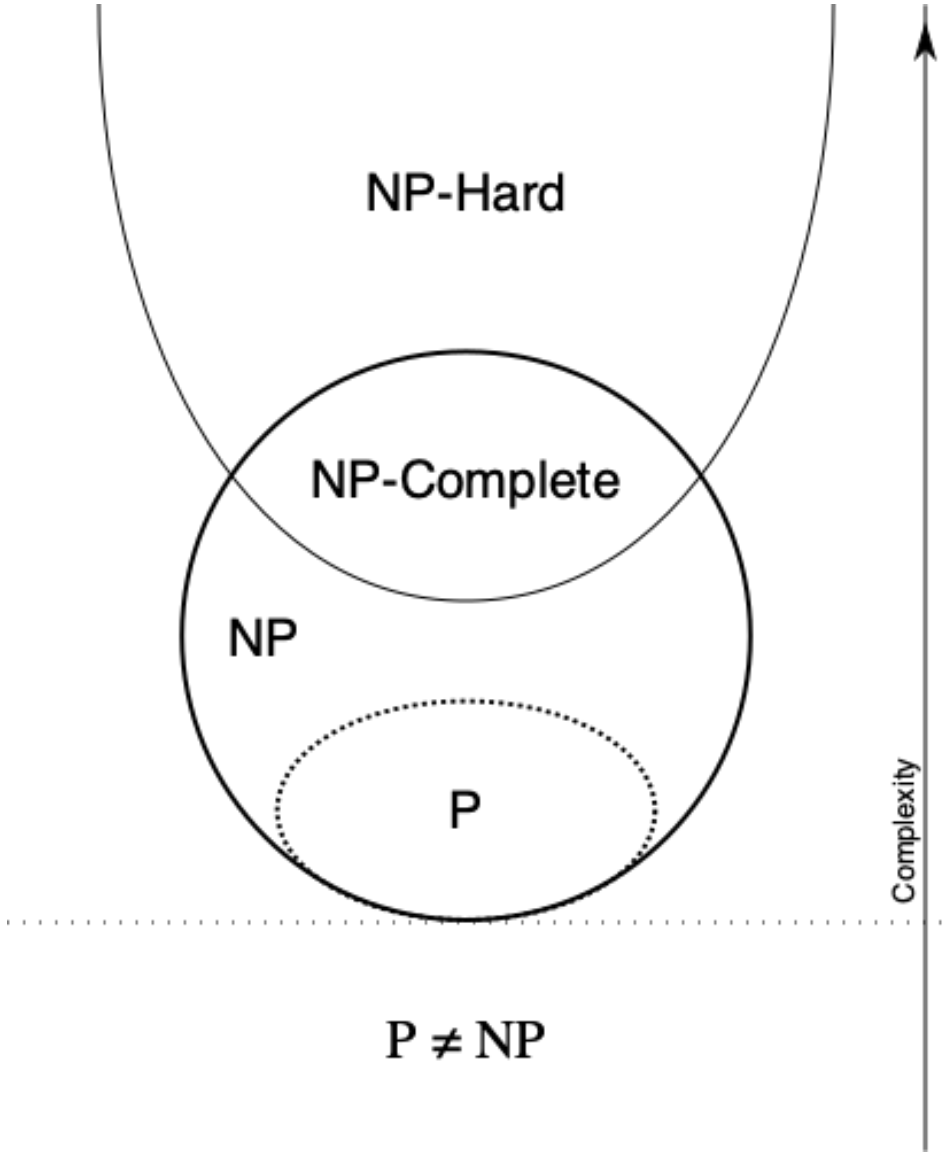
Neil Yorke-Smith

n.yorke-smith@tudelft.nl

February 2023

NP-hard

ACC



By Behnam Esfahbod, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3532181

NP-hard

Examples

Image by L.M. Ramirez-Elizondo,
https://repository.tudelft.nl/islandora/object/uuid:3e2cb6d7-3ba2-4b45-af71-2fa106b5d189

NP-hard

Examples

- making timetables (railways, schools, etc.)

Image by L.M. Ramirez-Elizondo,
https://repository.tudelft.nl/islandora/object/uuid:3e2cb6d7-3ba2-4b45-af71-2fa106b5d189

NP-hard

Examples

- making timetables (railways, schools, etc.)
- delivery of packages (e.g. TNT)

Image by L.M. Ramirez-Elizondo,
https://repository.tudelft.nl/islandora/object/uuid:3e2cb6d7-3ba2-4b45-af71-2fa106b5d189

NP-hard

Examples

- making timetables (railways, schools, etc.)
- delivery of packages (e.g. TNT)
- efficient routes for ride sharing (e.g. school bus, Valys)

Image by L.M. Ramirez-Elizondo,
https://repository.tudelft.nl/islandora/object/uuid:3e2cb6d7-3ba2-4b45-af71-2fa106b5d189

NP-hard

Examples

- making timetables (railways, schools, etc.)
- delivery of packages (e.g. TNT)
- efficient routes for ride sharing (e.g. school bus, Valys)
- scheduling processes (factories, hospitals, maintenance, CHPs in greenhouses, HVAC, etc.)

Image by L.M. Ramirez-Elizondo,
https://repository.tudelft.nl/islandora/object/uuid:3e2cb6d7-3ba2-4b45-af71-2fa106b5d189

NP-hard

Examples

- making timetables (railways, schools, etc.)
- delivery of packages (e.g. TNT)
- efficient routes for ride sharing (e.g. school bus, Valys)
- scheduling processes (factories, hospitals, maintenance, CHPs in greenhouses, HVAC, etc.)
- location of facilities (warehouses, shops, etc.)

Image by L.M. Ramirez-Elizondo,
https://repository.tudelft.nl/islandora/object/uuid:3e2cb6d7-3ba2-4b45-af71-2fa106b5d189

Algorithms?

- What algorithms can we use?

Algorithms?

- What algorithms can we use?
- Exponential time ☹

Algorithms?

- What algorithms can we use?
- Exponential time ☹
- This course gives us new algorithmic tools

Interactive block I: NP-hard

By Paddington_station_location_map.svg: OpenStreetMap contributors (up by Edward)derivative work: Maly LOLek (talk) - Paddington_station_location_map.svg, CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=6098976

About the course

Part I: Heuristic Search (Yorke-Smith)

How to get from A to B, probably as fast as you can.

About the course

Part I: Heuristic Search (Yorke-Smith)

How to get from A to B, probably as fast as you can.

Part II: Search and Inference (Demirović)

How to make exhaustive search less tiresome.

About the course

Part I: Heuristic Search (Yorke-Smith)

How to get from A to B, probably as fast as you can.

Part II: Search and Inference (Demirović)

How to make exhaustive search less tiresome.

Part III: Simulation-Based Planning (Lukina)

How to find approximate solutions under uncertainty.

About the course

Part I: Heuristic Search (Yorke-Smith)

How to get from A to B, probably as fast as you can.

Part II: Search and Inference (Demirović)

How to make exhaustive search less tiresome.

Part III: Simulation-Based Planning (Lukina)

How to find approximate solutions under uncertainty.

Part IV (guest lecture): Dynamic Programming (Robu)

How to use what you've done before.

Material, questions, assessment

- Content on Brightspace

Material, questions, assessment

- Content on Brightspace
- Questions on TUD Answers

Material, questions, assessment

- Content on Brightspace
- Questions on TUD Answers
- Assessment

Material, questions, assessment

- Content on Brightspace
- Questions on TUD Answers
- Assessment
  - Two optional lab sessions per part

Material, questions, assessment

- Content on Brightspace
- Questions on TUD Answers
- Assessment
  - Two optional lab sessions per part
  - One assignment per part (in pairs)

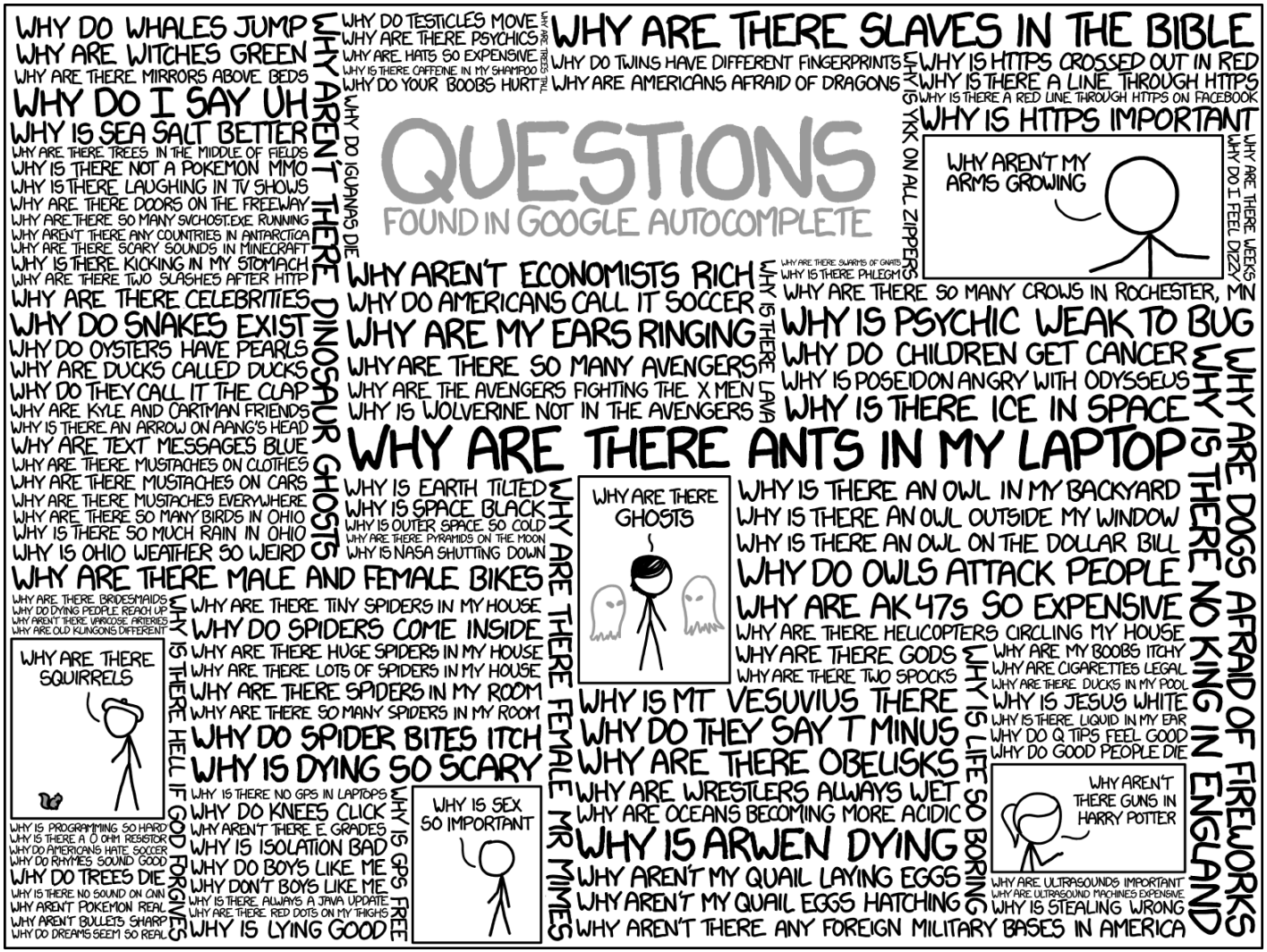
Material, questions, assessment

- Content on Brightspace
- Questions on TUD Answers
- Assessment
  - Two optional lab sessions per part
  - One assignment per part (in pairs)
  - One exam for all parts (individual, closed book)

Material, questions, assessment

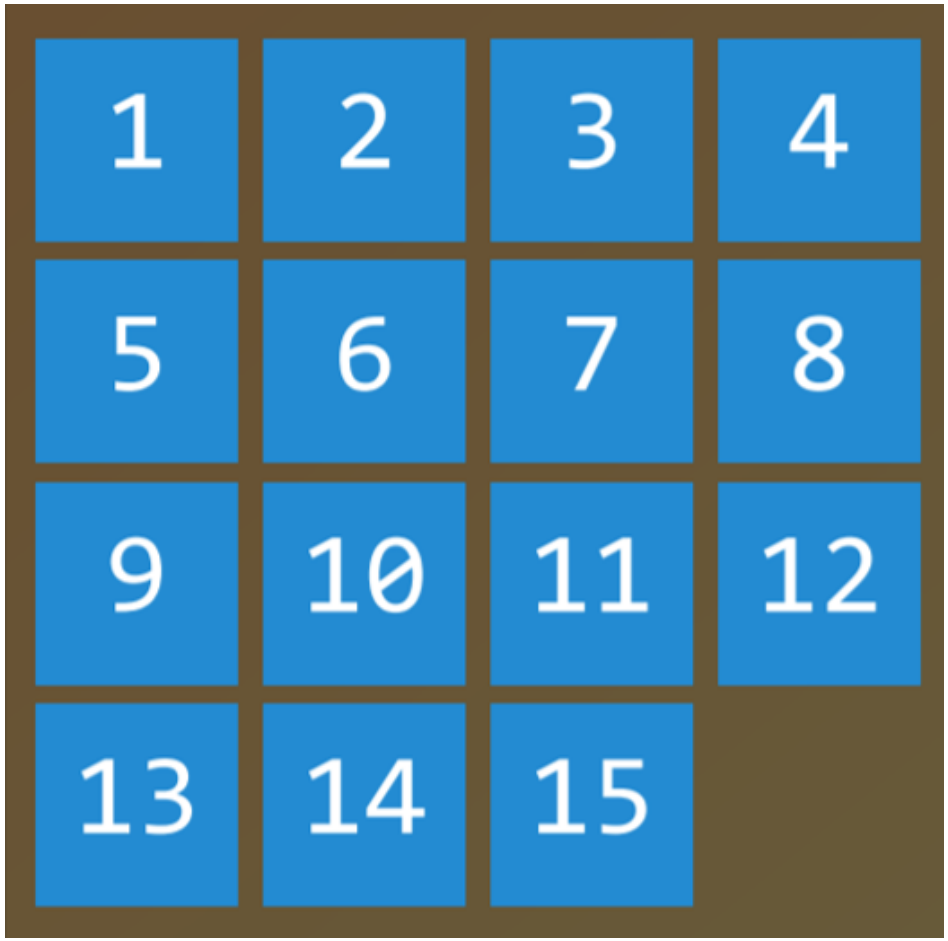
- Content on Brightspace
- Questions on TUD Answers
- Assessment
  - Two optional lab sessions per part
  - One assignment per part (in pairs)
  - One exam for all parts (individual, closed book)
  - Grade $= 0.3 \cdot A + 0.7 \cdot E$

Interactive block II: Questions



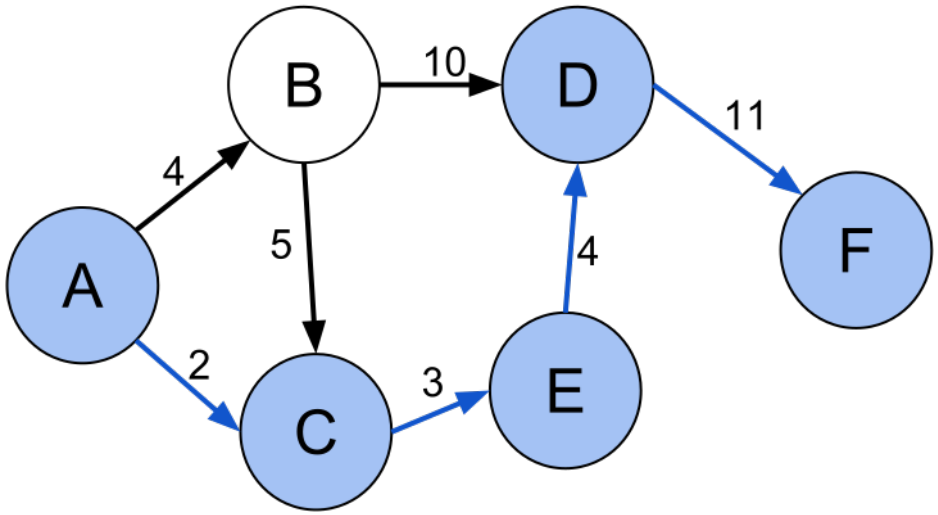
https://xkcd.com/1256/

Why search?



https://cs50.harvard.edu/ai/2020

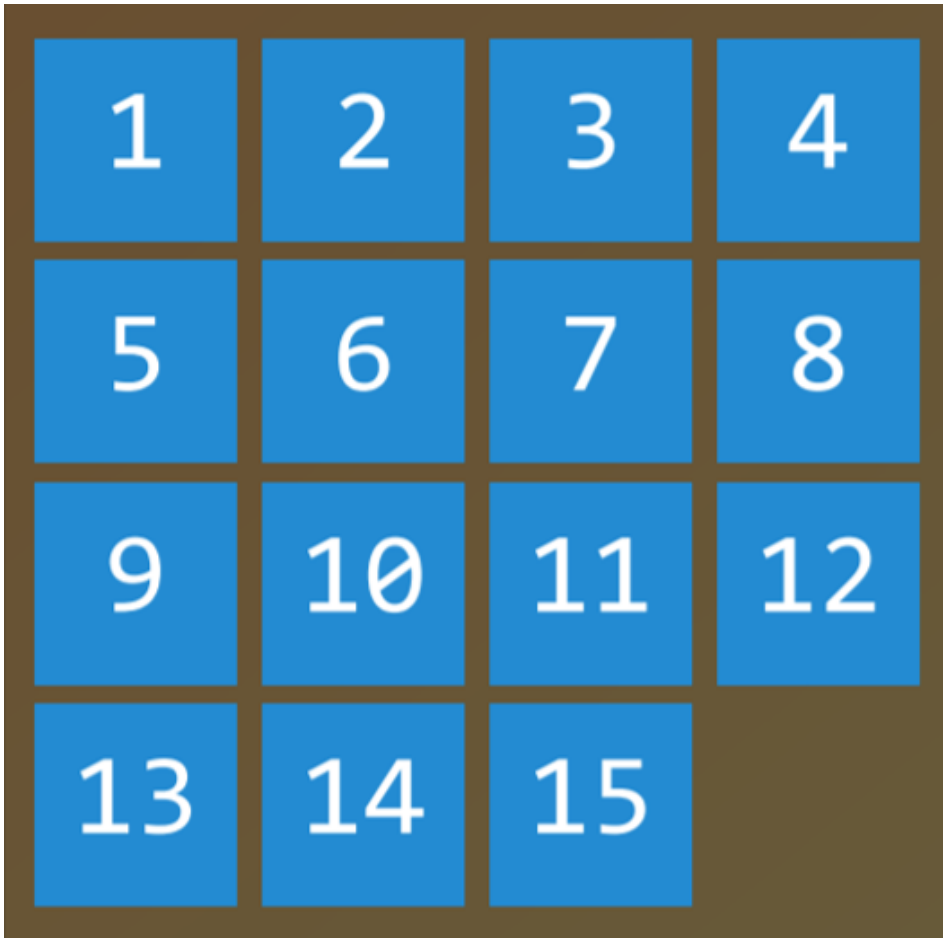
Heuristic search



By Artyom Kalinin - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=29980338

Search trees

Search is at the heart of AI



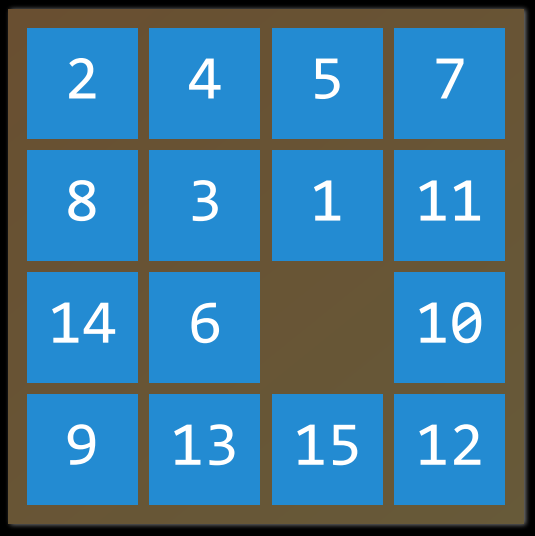
https://cs50.harvard.edu/ai/2020

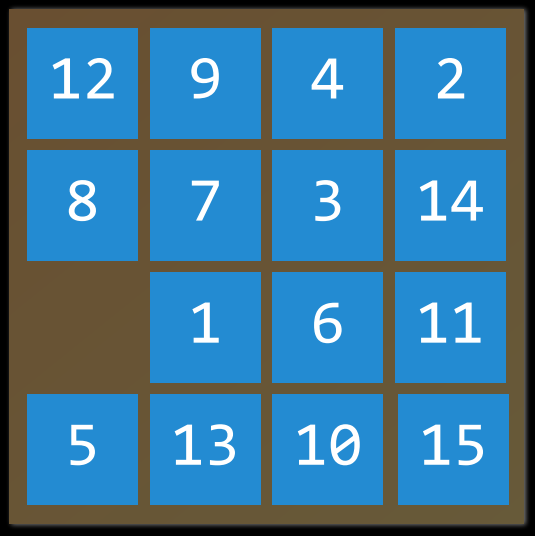
agent

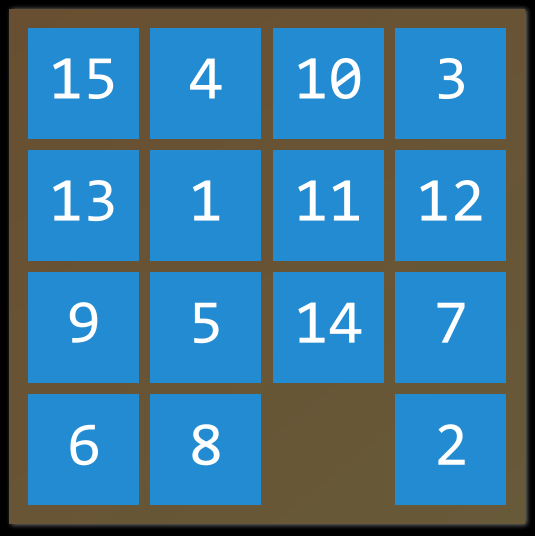
entity that perceives its environment
and acts upon that environment

state

a configuration of the agent and its environment


2 4 5 7
8 3 1 11
14 6 10
9 13 15 12

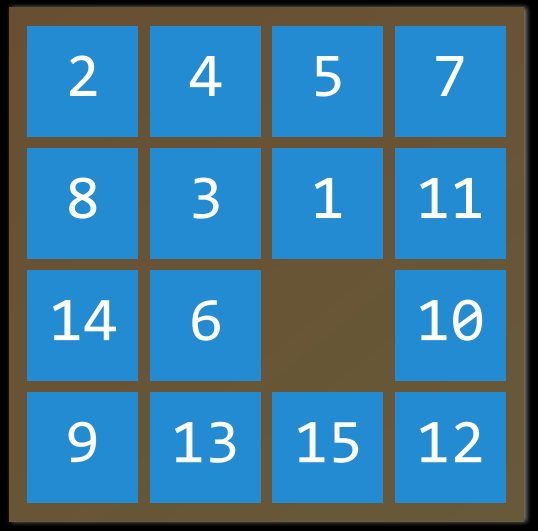

12 9 4 2
8 7 3 14
1 6 11
5 13 10 15


15 4 10 3
13 1 11 12
9 5 14 7
6 8 2

initial state

the state in which the agent begins

initial state



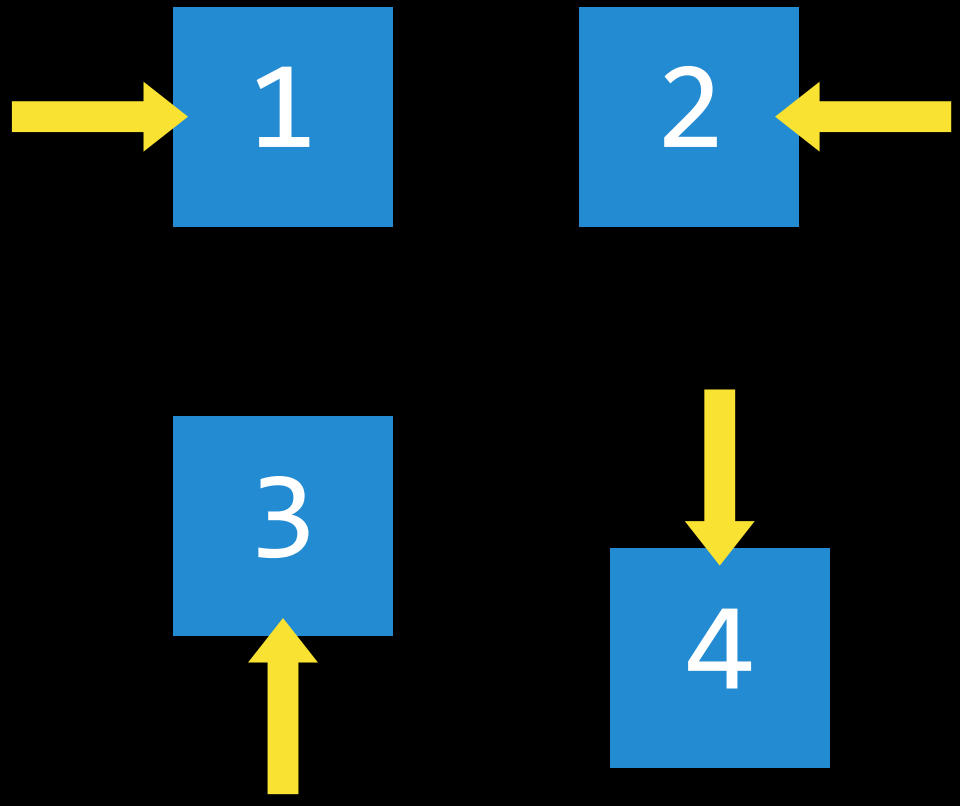
actions

choices that can be made in a state

actions

ACTIONS(s) returns the set of actions that can be executed in state s

actions



transition model

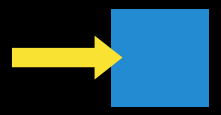
a description of what state results from performing any applicable action in any state

transition model

RESULT(s, a) returns the state resulting from performing action a in state s

RESULT(

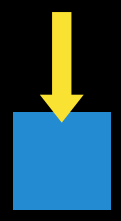
2	4	5	7
8	3	1	11
14	6	10	12
9	13	15	

, ) =

2	4	5	7
8	3	1	11
14	6	10	12
9	13		15

RESULT(

2	4	5	7
8	3	1	11
14	6	10	12
9	13	15	

, ) =

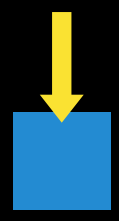
2	4	5	7
8	3	1	11
14	6	10	
9	13	15	12

transition model

RESULT(

2	4	5	7
8	3	1	11
14	6	10	12
9	13	15	

,

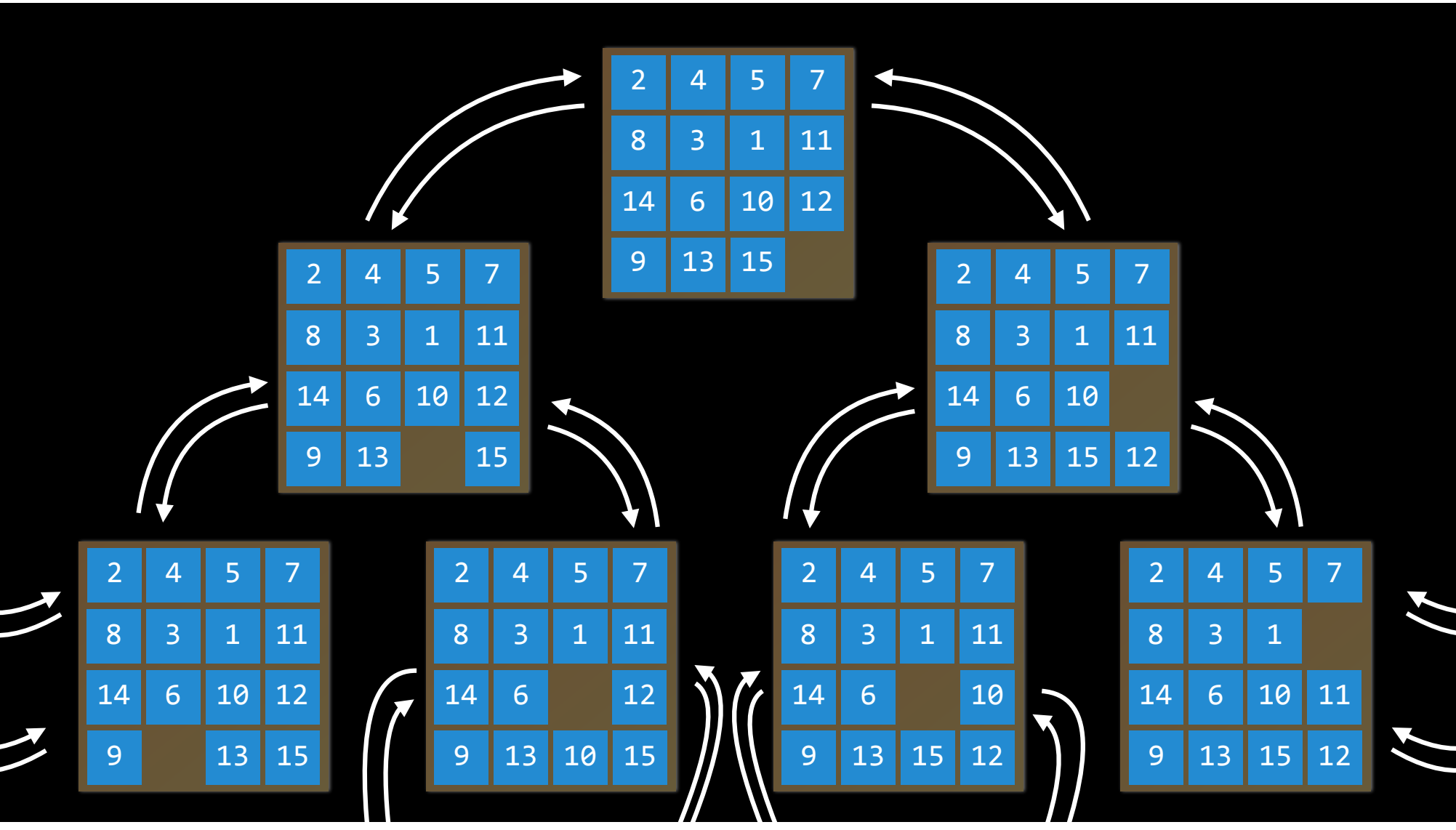


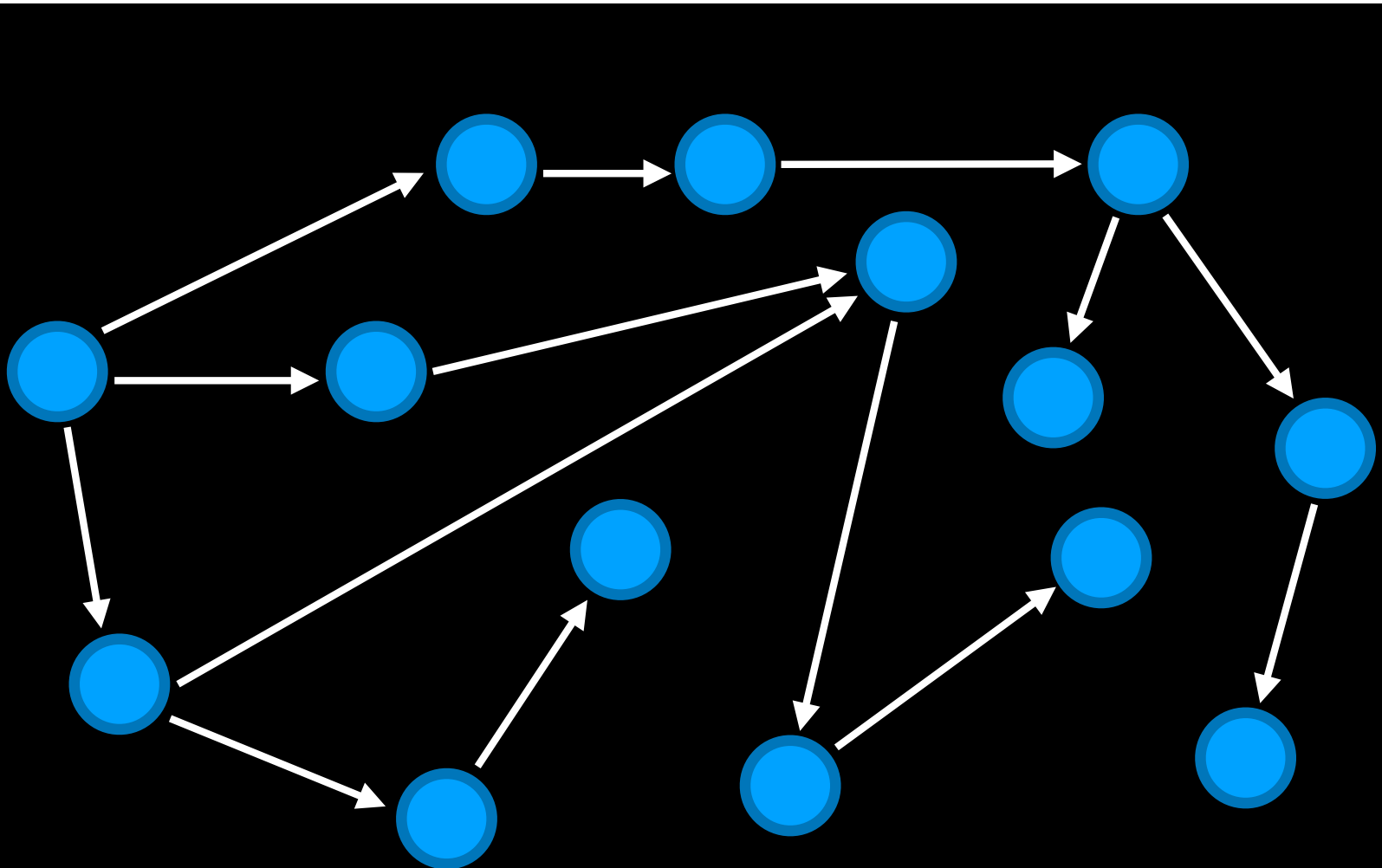
) =

2	4	5	7
8	3	1	11
14	6	10	
9	13	15	12

state space

the set of all states reachable from the initial state by any sequence of actions


2 4 5 7
8 3 1 11
14 6 10 12
9 13 15
2 4 5 7
8 3 1 11
14 6 10 12
9 13 15
2 4 5 7
8 3 1 11
14 6 10
9 13 15 12
2 4 5 7
8 3 1 11
14 6 12
9 13 10 15
2 4 5 7
8 3 1 11
14 6 10 12
9 13 15
2 4 5 7
8 3 1 11
14 6 10
9 13 15 12
2 4 5 7
8 3 1
14 6 10 11
9 13 15 12

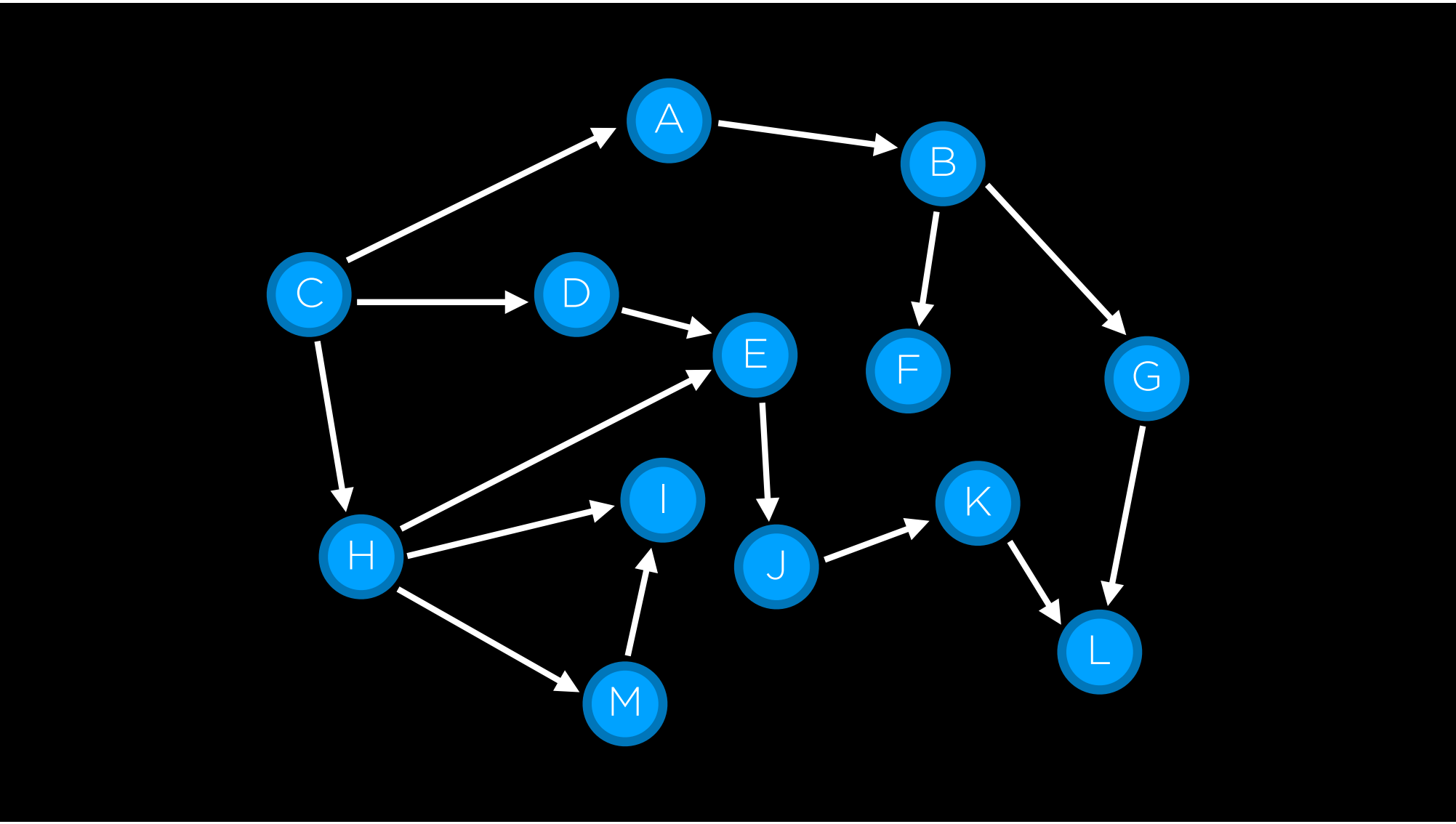


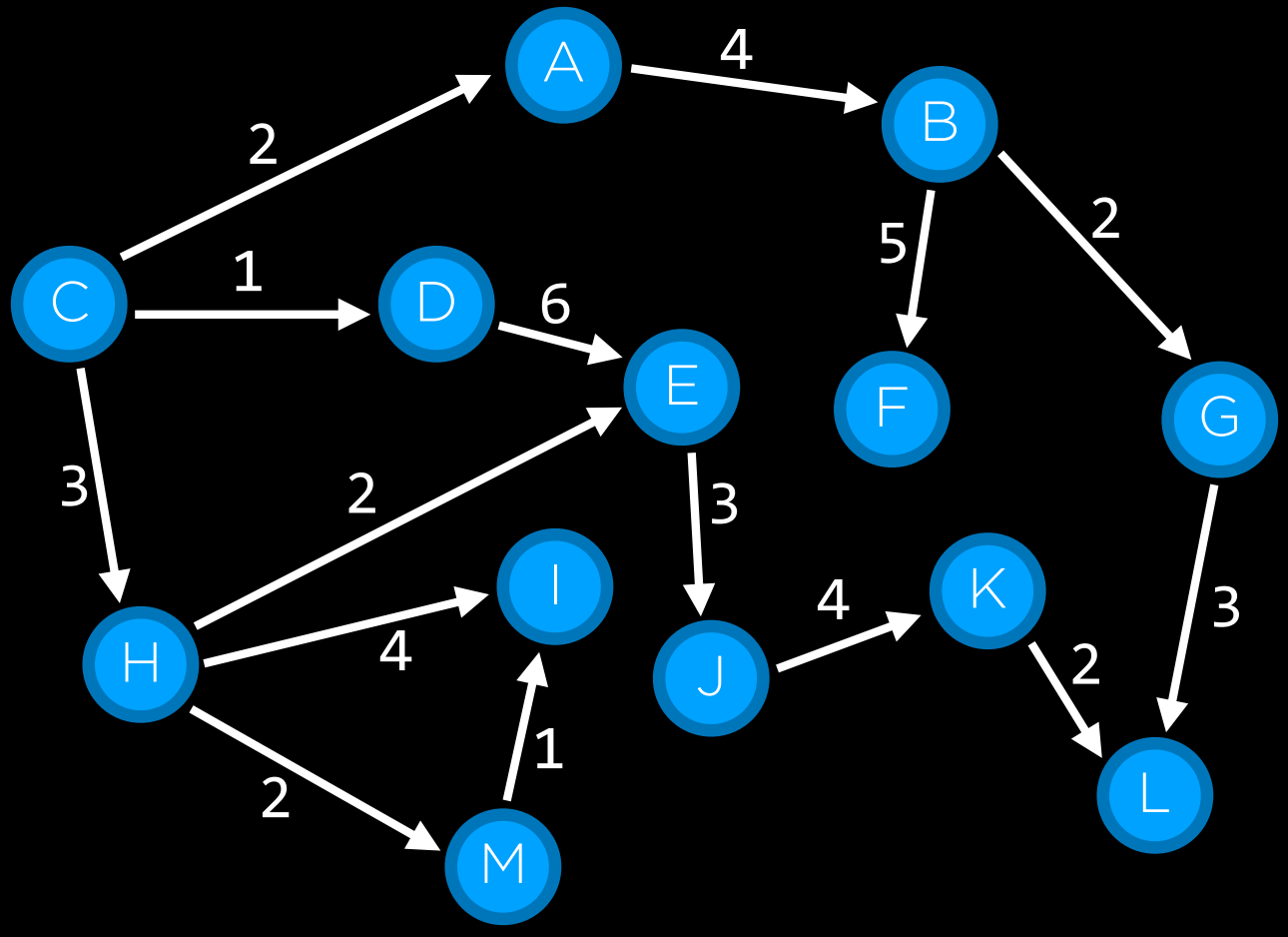
goal test

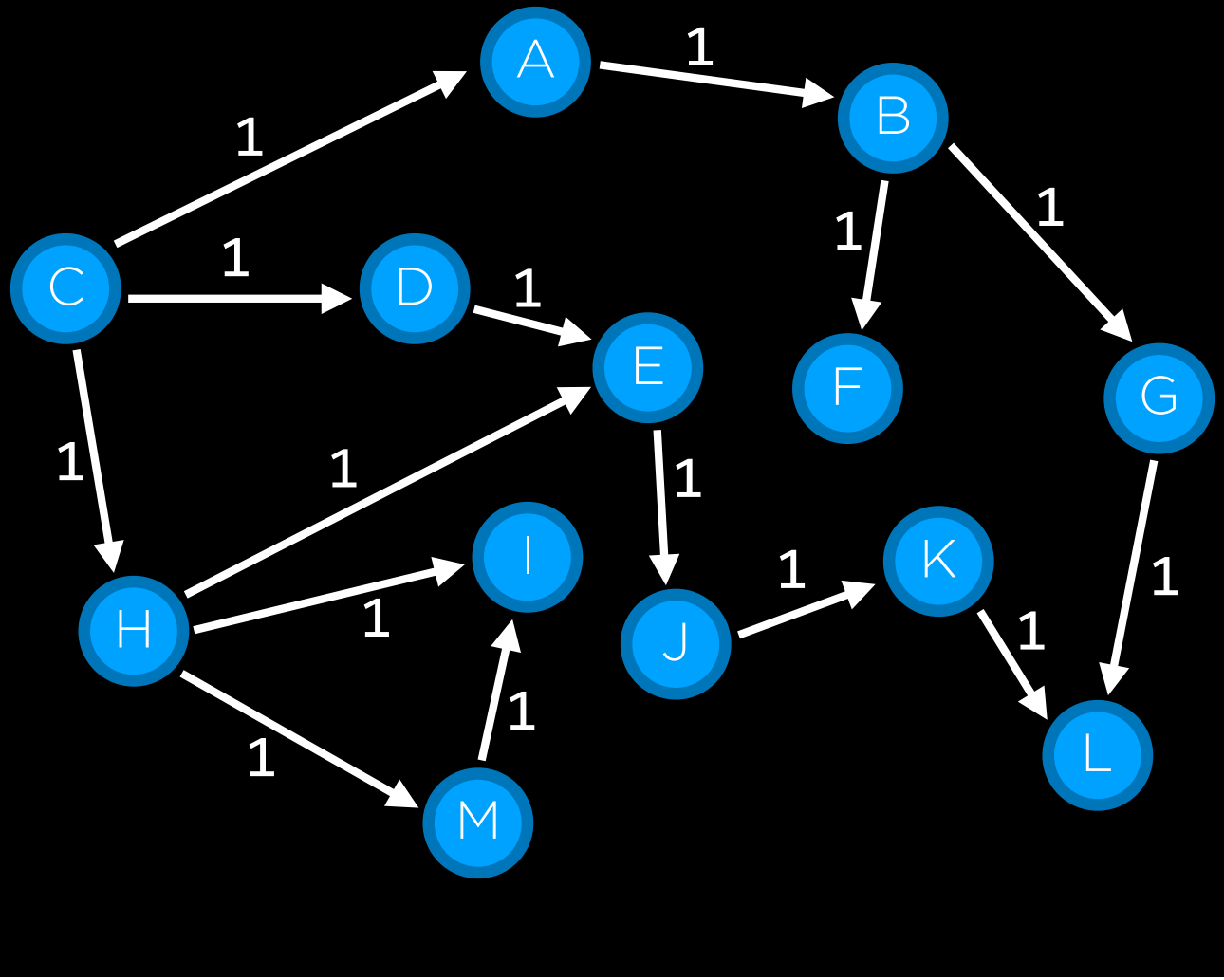
way to determine whether a given state is a goal state

path cost

numerical cost associated with a given path


A
B
C
D
E
F
G
H
I
J
K
L
M


A
B
C
D
E
F
G
H
I
J
K
L
M
4
2
1
6
5
2
3
2
3
4
4
3
2
1
2

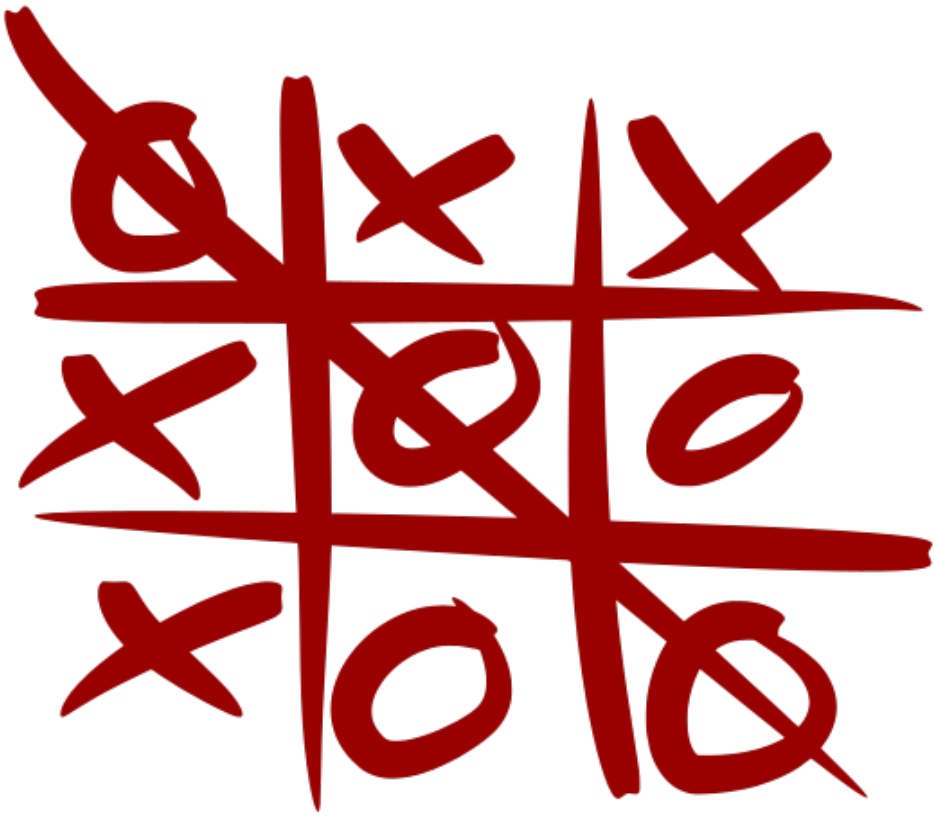

A
B
C
D
E
F
G
H
I
J
K
L
M
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1

Search Problems

- initial state
- actions
- transition model
- goal test
- path cost function

Noughts-and-crosses

A game to analyse



By Symode09 - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=2064271

Interactive block III: Search tree for OXO

Suppose X goes first, in the top-left corner

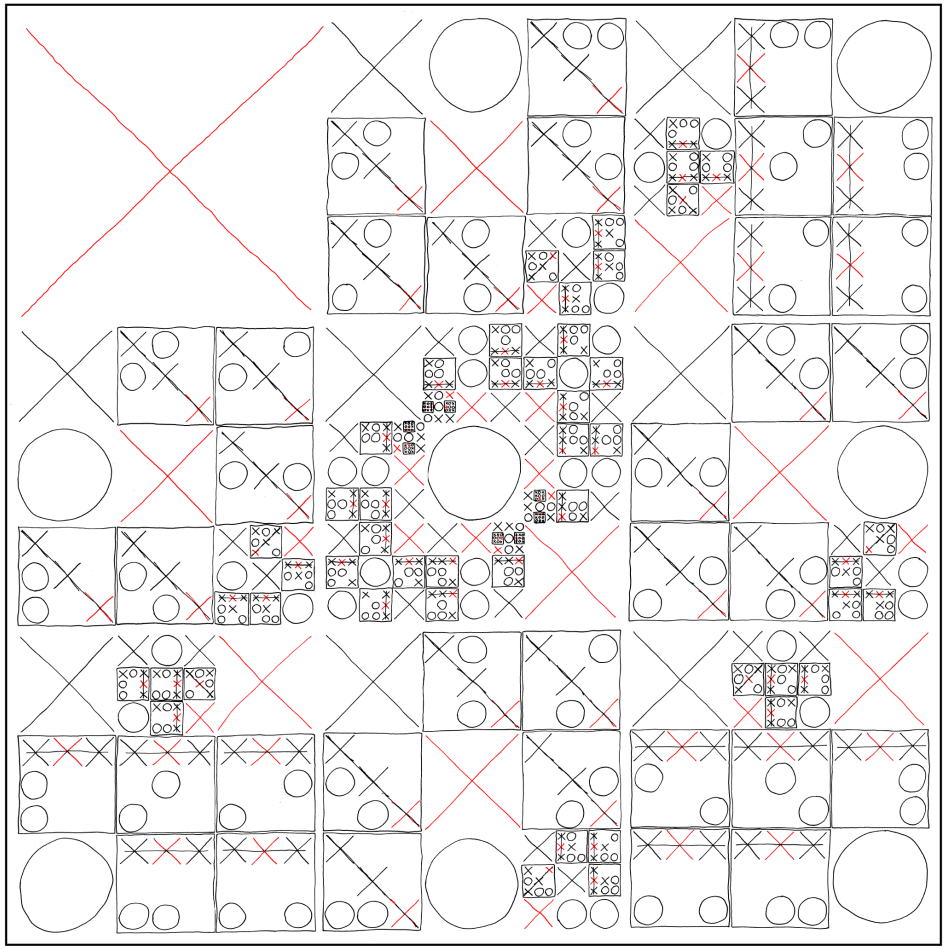
Write down the whole search tree for every possible move of O and X.

Interactive block III: Search tree for OXO

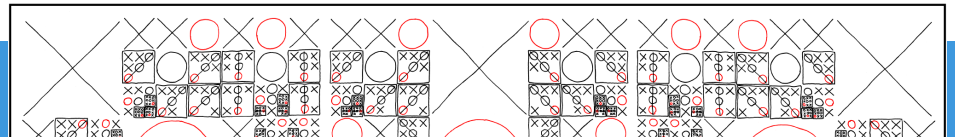
COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES

YOUR MOVE IS GIVEN BY THE POSITION OF THE LARGEST RED SYMBOL ON THE GRID. WHEN YOUR OPPONENT PICKS A MOVE, ZOOM IN ON THE REGION OF THE GRID WHERE THEY WENT. REPEAT.

MAP FOR X:



MAP FOR O:

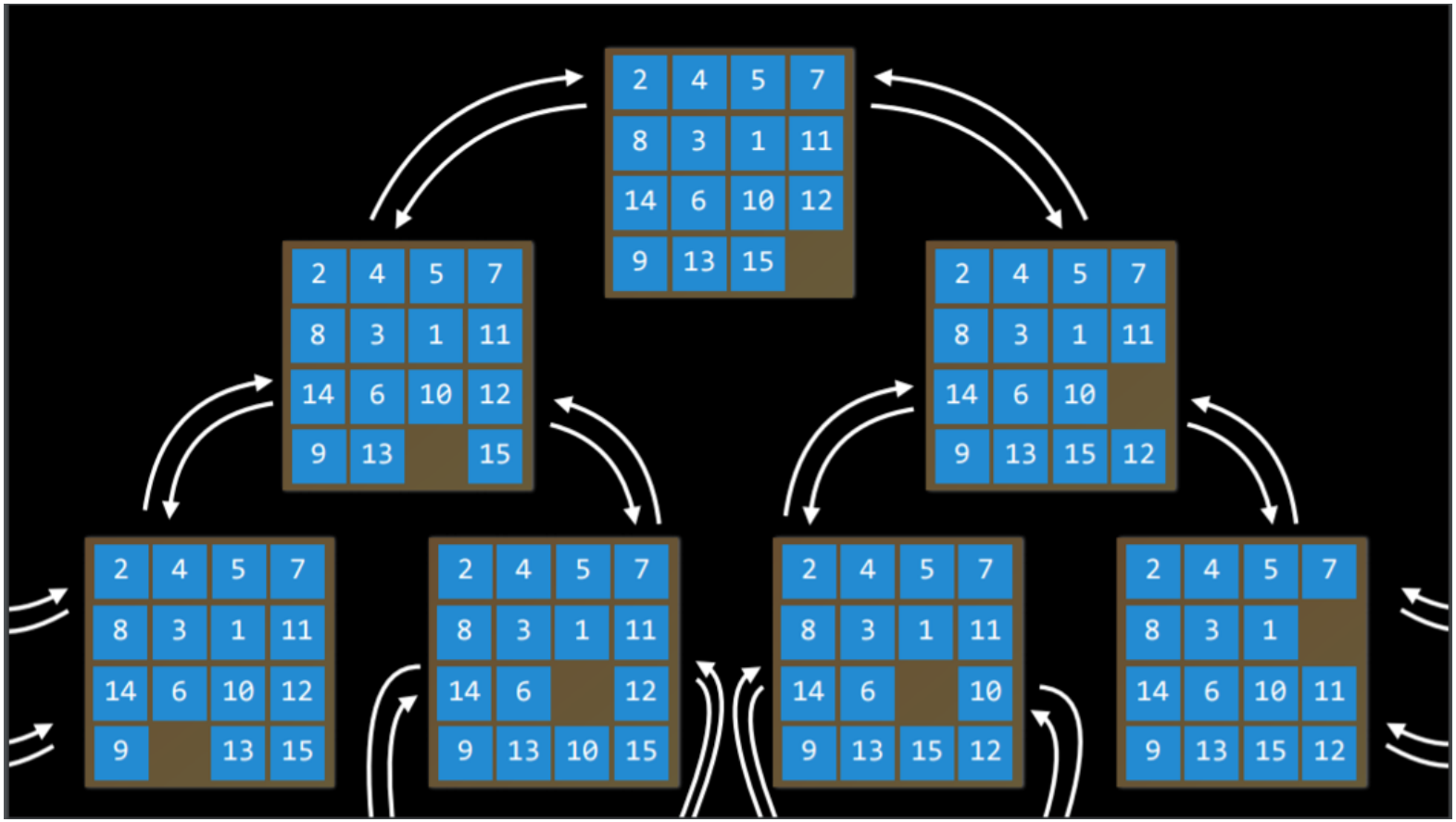


https://xkcd.com/832/

15 minute(s) break!

Search trees

Search is at the heart of AI



https://cs50.harvard.edu/ai/2020

Depth-First Search

Go deep

- Always explore a deepest node in the frontier

Depth-First Search

Go deep

- Always explore a deepest node in the frontier
- Frontier = stack

Depth-First Search

Go deep

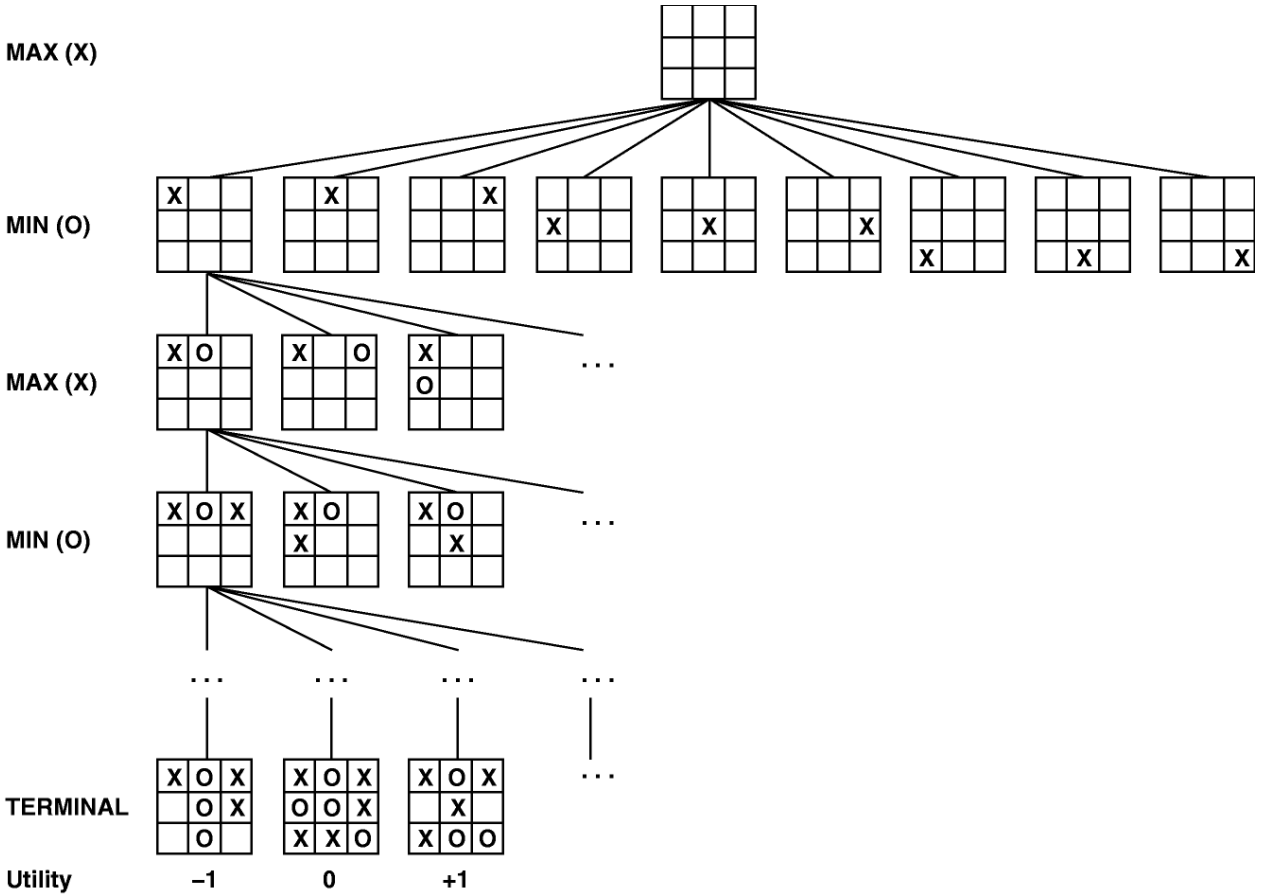
- Always explore a deepest node in the frontier
- Frontier = stack
- Advantages?

Interactive block IV: DFS for OXO

Suppose X goes first, in the top-left corner

Show a DFS traversal of your search tree.

Interactive block IV: Search tree for OXO



https://www.massey.ac.nz/~mjjohnso/

Breadth-First Search

Go wide

- Always explore a shallowest node in the frontier

Breadth-First Search

Go wide

- Always explore a shallowest node in the frontier
- Frontier = queue

Breadth-First Search

Go wide

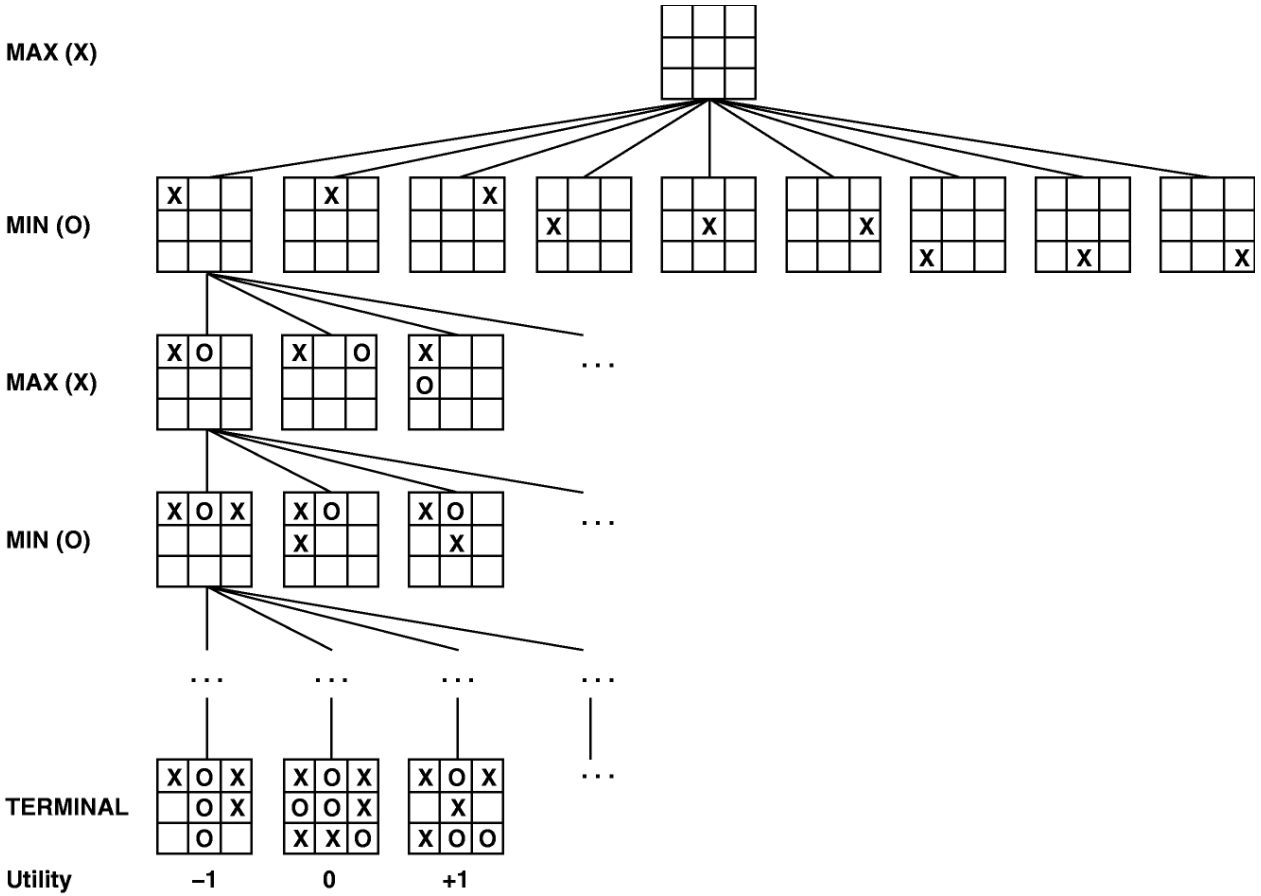
- Always explore a shallowest node in the frontier
- Frontier = queue
- Advantages?

Interactive block V: BFS for OXO

Suppose X goes first, in the top-left corner

Show a BFS traversal of your search tree.

Interactive block V: Search tree for OXO



https://www.massey.ac.nz/~mjjohnso/

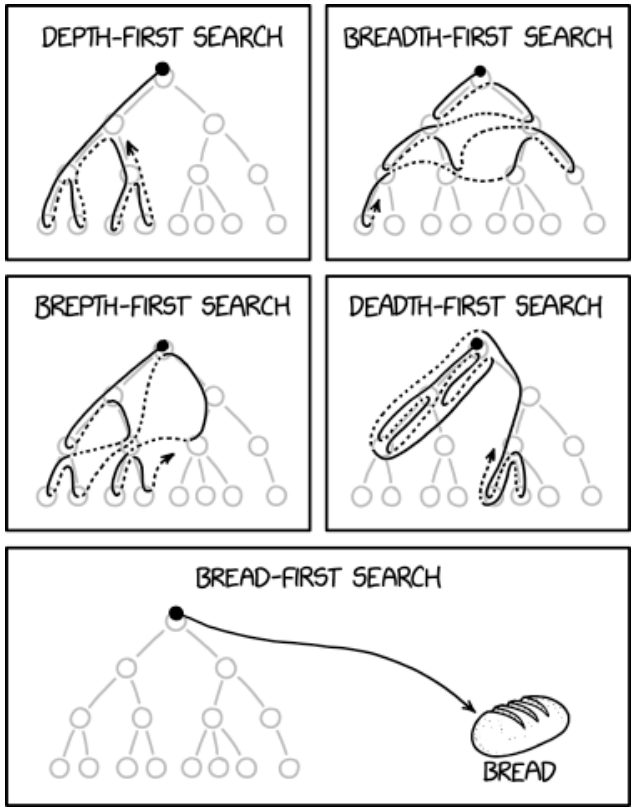
Now what?

Now what?

- DFS or BFS with smarter data structures?

Now what?

- DFS or BFS with smarter data structures?
- Combining DFS and BFS?



https://xkcd.com/2407/

Now what?

- DFS or BFS with smarter data structures?
- Combining DFS and BFS?
- DFS etc on graphs (instead of trees)?

Now what?

- DFS or BFS with smarter data structures?
- Combining DFS and BFS?
- DFS etc on graphs (instead of trees)?
- DFS etc on hyper-graphs (instead of planar)?

Now what?

- DFS or BFS with smarter data structures?
- Combining DFS and BFS?
- DFS etc on graphs (instead of trees)?
- DFS etc on hyper-graphs (instead of planar)?
- DFS etc in parallel?

Now what?

- DFS or BFS with smarter data structures?
- Combining DFS and BFS?
- DFS etc on graphs (instead of trees)?
- DFS etc on hyper-graphs (instead of planar)?
- DFS etc in parallel?
- ...

Now what?

iBFS: Concurrent Breadth-First Search on GPUs

Hang Liu H. Howie Huang Yang Hu
Department of Electrical and Computer Engineering
George Washington University
{asherliu, howie, huyang}@gwu.edu

ABSTRACT

Breadth-First Search (BFS) is a key graph algorithm with many important applications. In this work, we focus on a special class of graph traversal algorithm - concurrent BFS - where multiple breadth-first traversals are performed simultaneously on the same graph. We have designed and developed a new approach called *iBFS* that is able to run i concurrent BFSes from i distinct source vertices, very efficiently on Graphics Processing Units (GPUs). iBFS consists of three novel designs. First, iBFS develops a single GPU kernel for *joint traversal* of concurrent BFS to take advantage of shared frontiers across different instances. Second, *outdegree-based GroupBy rules* enables iBFS to selectively run a group of BFS instances which further maximizes the frontier sharing within such a group. Third, iBFS brings additional performance benefit by utilizing highly optimized *bitwise operations* on GPUs, which allows a single GPU thread to inspect a vertex for concurrent BFS instances. The evaluation on a wide spectrum of graph benchmarks shows that iBFS on one GPU runs up to 30× faster than executing BFS instances sequentially, and on 112 GPUs achieves near linear speedup with the maximum performance of 57,267 billion traversed edges per second (TEPS).

1. INTRODUCTION

Graph-based representations are ubiquitous in many applications such as social networks [1–3], metabolic networks [4],

Depending on the value of i, iBFS actually becomes a number of different problems. Formally, in a graph with $|V|$ vertices, iBFS is

- single source shortest path (SSSP) *if* $i = 1$ [6];
- multi-source shortest path (MSSP) *if* $i \in (1, |V|)$ [7,8];
- all-pairs shortest path (APSP) *if* $i = |V|$ [9,10].

Moreover, iBFS can be utilized in many other graph algorithms such as betweenness centrality [11, 12] and closeness centrality [13]. For example, one can leverage iBFS to construct the index for answering graph reachability queries, that is, whether there exists a path from vertex s to t with the number of edges in-between less than k [14–16]. We will show in this paper this step can be an order of magnitude faster with iBFS. In all, a wide variety of applications, e.g., network routing [17–19], network attack detection [20], route planning [21–23], web crawling [24,25], etc., can benefit from high-performance iBFS.

This work proposes a new approach for running iBFS on GPUs that consists of three novel techniques: joint traversal, GroupBy, and bitwise optimization. Prior work has proposed to combine the execution of different BFS instances mostly on multi-core CPUs [26–28]. The performance improvement, however, is limited. For one, none of early projects has attempted to group the BFS instances to improve the frontier sharing during the traversal. Further, bottom-up BFS provides new additional challenges. For example, while MS-BFS [26] supports bottom-up, it does not provide early

Now what?

Now what?

Informed search

What if we have some problem-specific information?

(Greedy) Best-First Search

Go smart

- Always explore a best node in the frontier

(Greedy) Best-First Search

Go smart

- Always explore a best node in the frontier
- Best $=$ measured by an estimate $h(n)$ of how close node n is to the goal

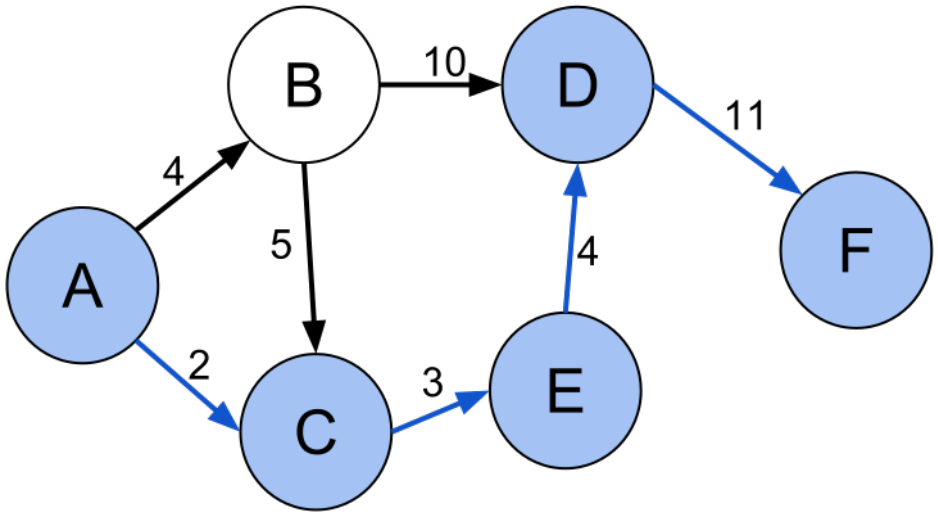
(Greedy) Best-First Search

Go smart

- Always explore a best node in the frontier
- Best $=$ measured by an estimate $h(n)$ of how close node n is to the goal
- Advantages?

Best node = closest to goal

According to heuristic $h(n)$



By Artyom Kalinin - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=29980338

Interactive block VI: BestFS for OXO

Suppose X goes first, in the top-left corner

Use the heuristic function:

$$h(n) = \sum_{x \in \text{squares}} \begin{cases} 3 & x = \text{middle} \\ 2 & x = \text{corner} \\ 1 & x = \text{otherwise} \end{cases} - \sum_{o \in \text{squares}} \begin{cases} 3 & o = \text{middle} \\ 2 & o = \text{corner} \\ 1 & o = \text{otherwise} \end{cases}$$

Show a BestFS traversal of your search tree.

Interactive block VI: BestFS for OXO

Example heuristic function

https://cs50.harvard.edu/ai/2020

Interactive block VI: BestFS for OXO

Example heuristic function

$h(n) =$

https://cs50.harvard.edu/ai/2020

Interactive block VI: BestFS for OXO

Example heuristic function

$$h(n) = 3 + 2 + 1 -$$

https://cs50.harvard.edu/ai/2020

Interactive block VI: BestFS for OXO

Example heuristic function

$$h(n) = 3 + 2 + 1 - (2 + 2 + 1) =$$

https://cs50.harvard.edu/ai/2020

Interactive block VI: BestFS for OXO

Example heuristic function

$$h(n) = 3 + 2 + 1 - (2 + 2 + 1) = 1$$

https://cs50.harvard.edu/ai/2020

Interactive block VI: BestFS for OXO

Example heuristic function

$$h(n) = 3 + 2 + 1 - (2 + 2 + 1) = 1$$

What terminal states is this heuristic designed to find?

https://cs50.harvard.edu/ai/2020

Next lecture

- Today we saw basic tree search

Next lecture

- Today we saw basic tree search
  - Heuristics for game playing ☞ Lectures 3 & 4

Next lecture

- Today we saw basic tree search
  - Heuristics for game playing ☞ Lectures 3 & 4
  - Exhaustive tree search will return in Part 2

Next lecture

- Today we saw basic tree search
  - Heuristics for game playing ☞ Lectures 3 & 4
  - Exhaustive tree search will return in Part 2
- Next time: smarter greedy best-first search

Next lecture

- Today we saw basic tree search
  - Heuristics for game playing ☞ Lectures 3 & 4
  - Exhaustive tree search will return in Part 2
- Next time: smarter greedy best-first search
- Preparation: see Brightspace

Next lecture

- Today we saw basic tree search
  - Heuristics for game playing ☞ Lectures 3 & 4
  - Exhaustive tree search will return in Part 2
- Next time: smarter greedy best-first search
- Preparation: see Brightspace
- Lab on Monday

Next lecture

- Today we saw basic tree search
  - Heuristics for game playing ☞ Lectures 3 & 4
  - Exhaustive tree search will return in Part 2
- Next time: smarter greedy best-first search
- Preparation: see Brightspace
- Lab on Monday
- Questions? TUD Answers

What is still unclear?

After every lecture...

Give us some homework:
tell us what is still unclear.

ALGORITHMS FOR NP-HARD PROBLEMS

Part I – Heuristic Search

Lecture 2: Greedy Best-First Search

Neil Yorke-Smith

n.yorke-smith@tudelft.nl

February 2023

Where are we?

- Last time: systematically searching a tree
- Today: (more) informed search

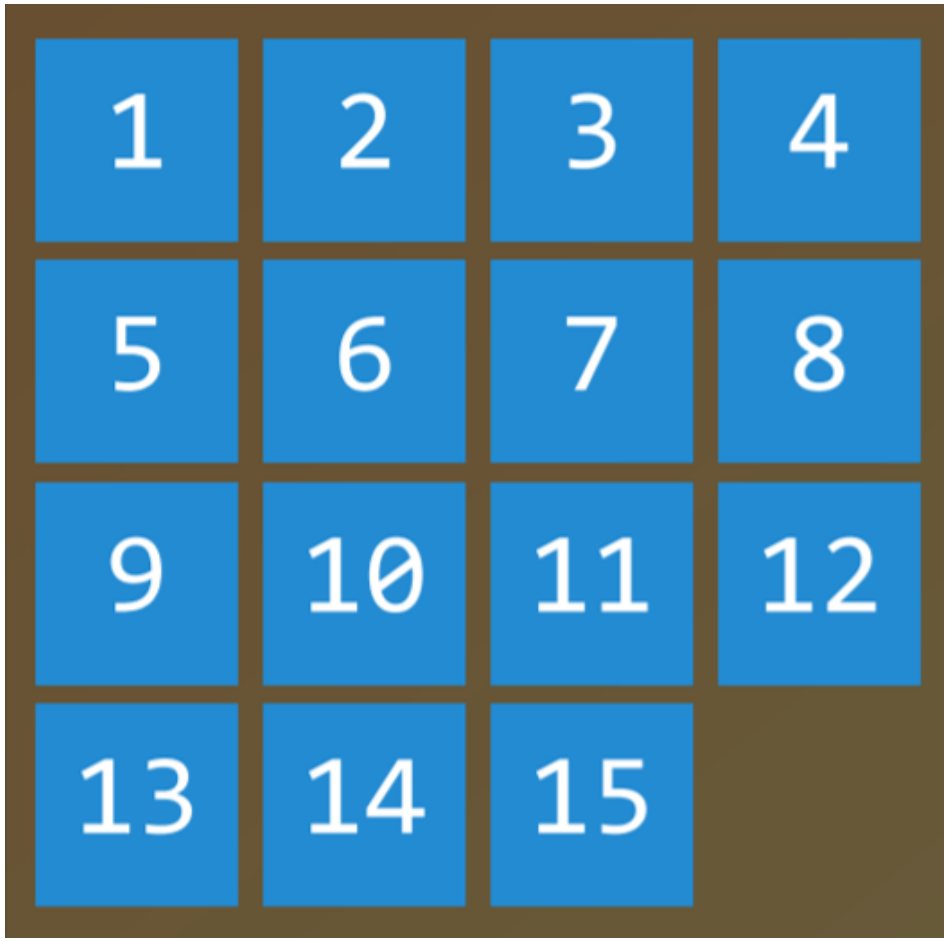
Where are we?

- Last time: systematically searching a tree
- Today: (more) informed search
- You prepared:
  - Video on A* search

Questions from Lecture 1

Uninformed search

Uses no problem-specific knowledge



https://cs50.harvard.edu/ai/2020

Informed search

Uses problem-specific knowledge

Informed search

Heuristic function estimates

Informed search

Uses problem-specific knowledge

Informed search

Heuristic function estimates distance of node to goal.

Informed search

Uses problem-specific knowledge

Informed search

Heuristic function estimates distance of node to goal.

Is informed better than uninformed?

Informed search

Uses problem-specific knowledge

Informed search

Heuristic function estimates distance of node to goal.

Is informed better than uninformed?

Not always.

Dijkstra's algorithm

aka Uniform Cost Search

By Hamilton Richards - manuscripts of Edsger W. Dijkstra, University Texas at Austin, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4204157

Dijkstra's algorithm

aka Uniform Cost Search

- Expands from starting node

By Hamilton Richards - manuscripts of Edsger W. Dijkstra, University Texas at Austin, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4204157

Dijkstra's algorithm

aka Uniform Cost Search

- Expands from starting node
- Uniformly, using cost-from-start-to-current-node $g(n)$

By Hamilton Richards - manuscripts of Edsger W. Dijkstra, University Texas at Austin, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4204157

Dijkstra's algorithm

aka Uniform Cost Search

- Expands from starting node
- Uniformly, using cost-from-start-to-current-node $g(n)$
- Guarantees shortest path (unlike Greedy BestFS)

By Hamilton Richards - manuscripts of Edsger W. Dijkstra, University Texas at Austin, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=4204157

A* search

- Look back and look forward (with estimate)

A* search

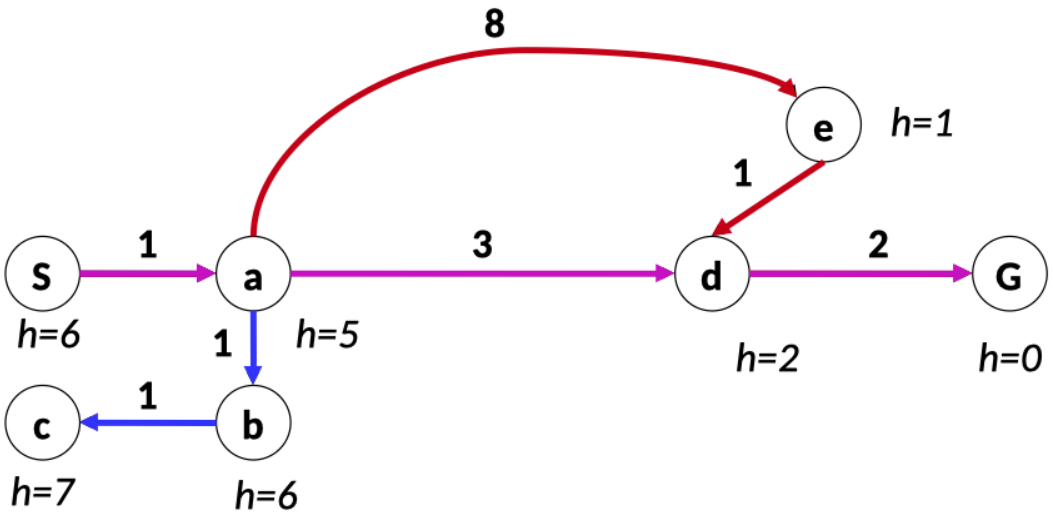
- Look back and look forward (with estimate)
- Use $g(n) + h(n)$

A* search

- Look back and look forward (with estimate)
- Use $g(n) + h(n)$
- Advantages?

A* search

Example

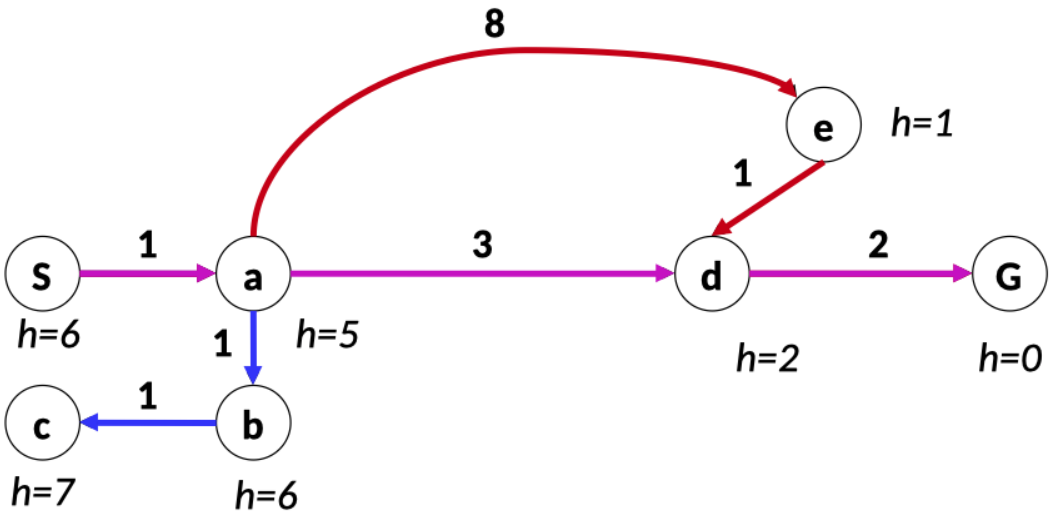


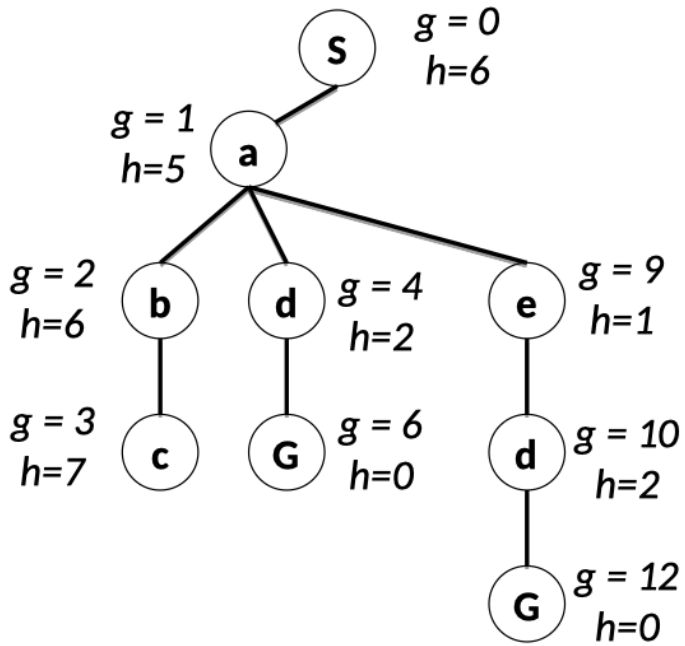
Dijkstra (UCS) $g(n)$ + Greedy $h(n)$ = A* $g(n) + h(n)$

Example: Teg Grenager

A* search

Example

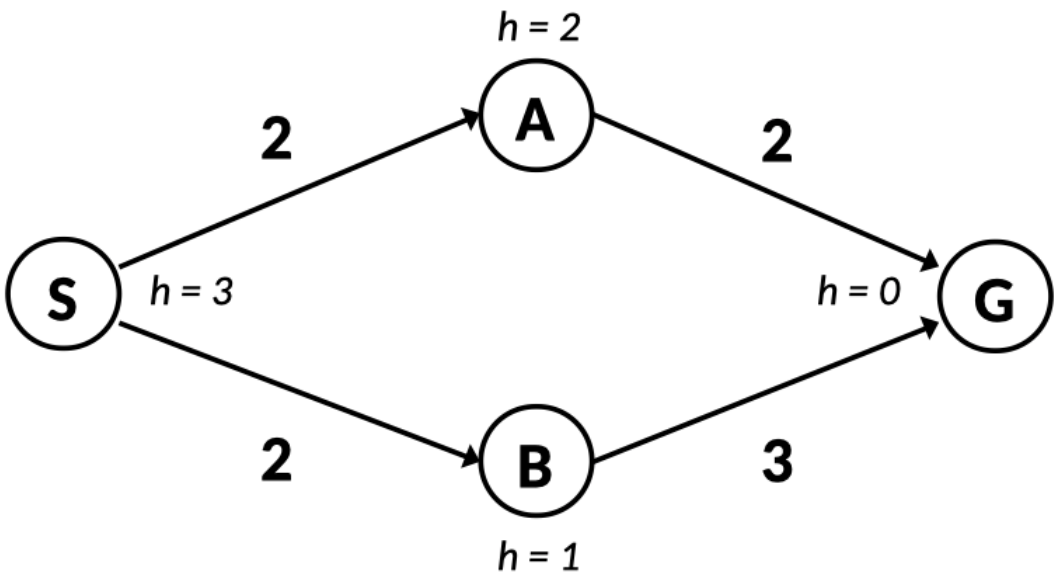




Dijkstra (UCS) $g(n)$ + Greedy $h(n)$ = A* $g(n) + h(n)$

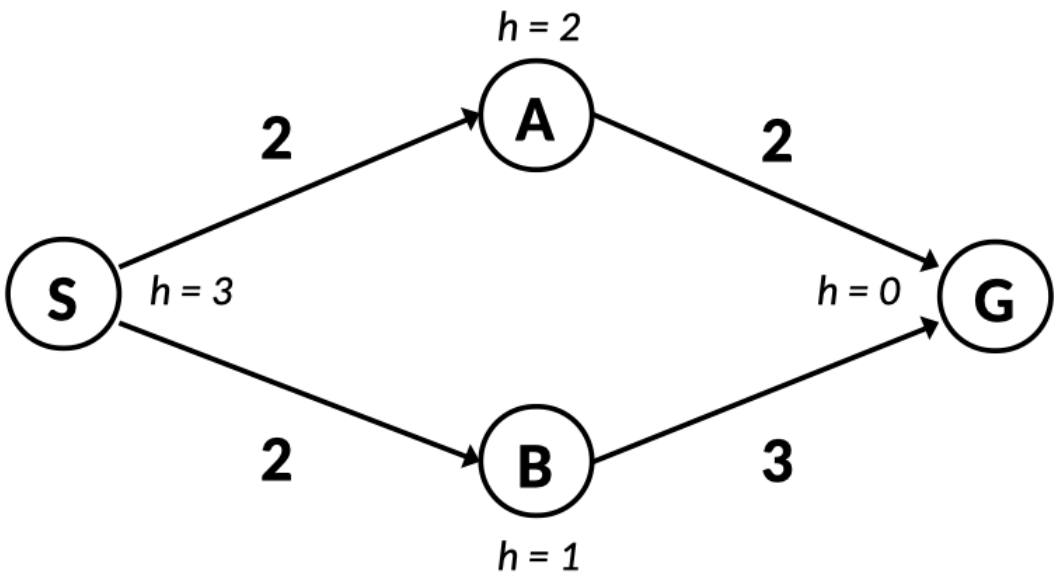
Example: Teg Grenager

A* termination



https://inst.eecs.berkeley.edu/~cs188/su20/

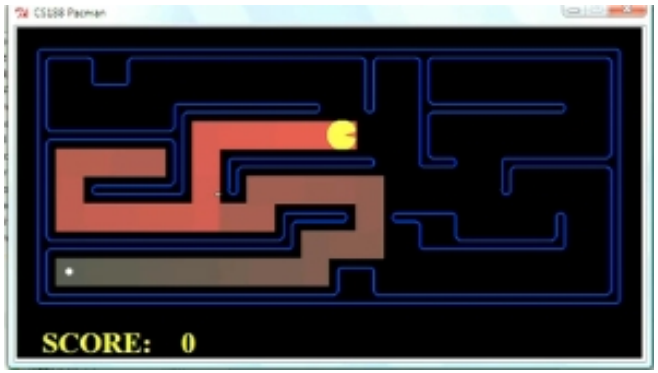
A* termination

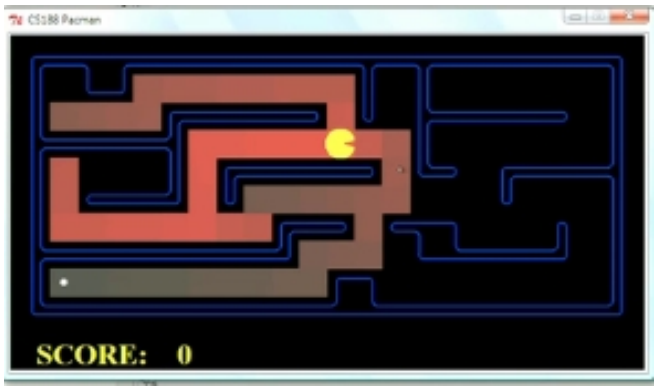


Stop when dequeue a goal

https://inst.eecs.berkeley.edu/~cs188/su20/

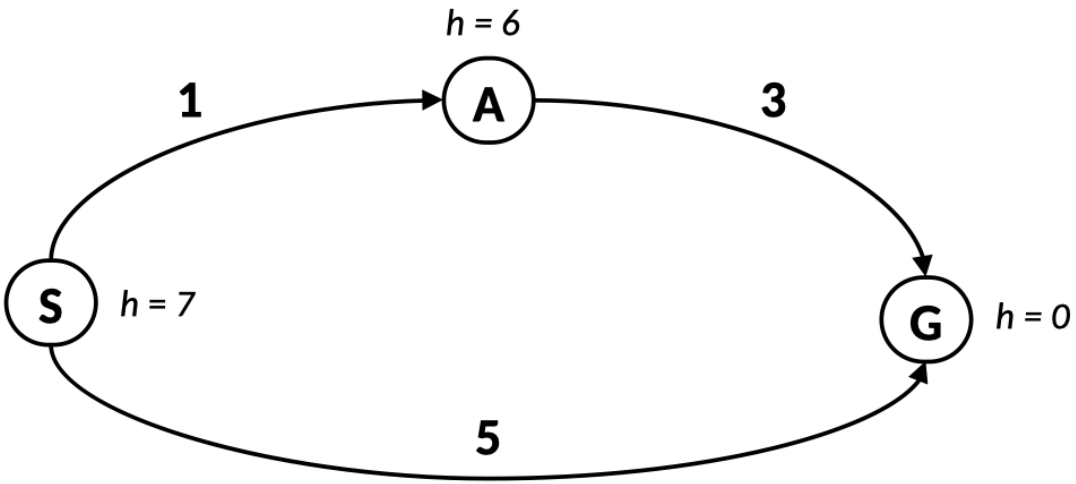
Greedy vs. UCS vs. A*





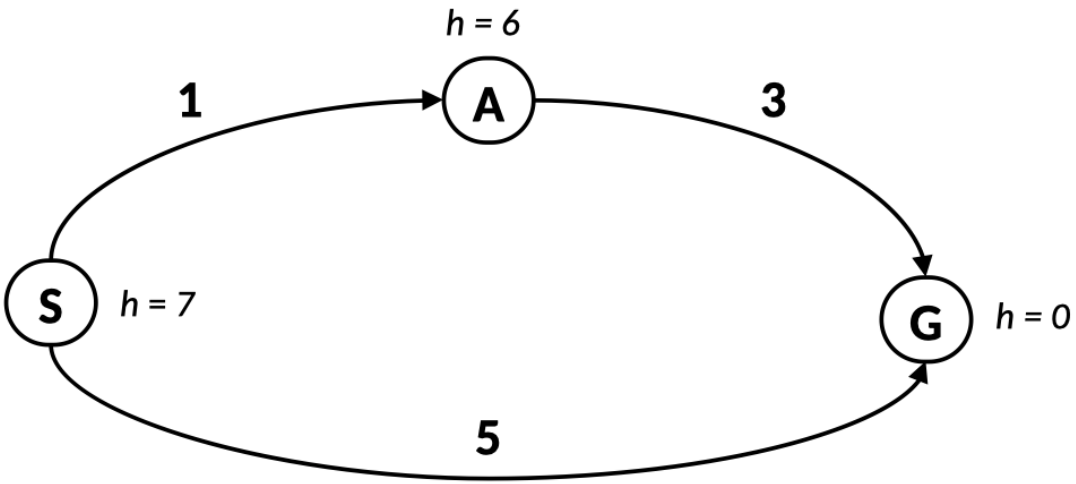
https://inst.eecs.berkeley.edu/~cs188/su20/

Interactive block I: Is A* optimal?



https://inst.eecs.berkeley.edu/~cs188/su20/

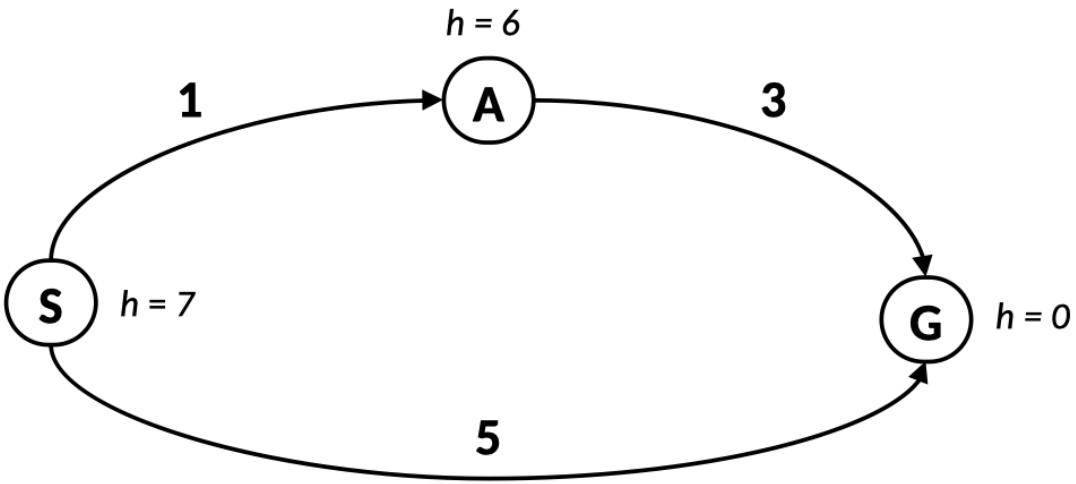
Interactive block I: Is A* optimal?



- Actual bad goal cost $\leq$ estimated good goal cost

https://inst.eecs.berkeley.edu/~cs188/su20/

Interactive block I: Is A* optimal?



- Actual bad goal cost $\leq$ estimated good goal cost
- We need estimates to be less than actual costs!

https://inst.eecs.berkeley.edu/~cs188/su20/

Properties of A* heuristic

Necessary and sufficient for optimality

Properties of A* heuristic

Necessary and sufficient for optimality

Admissible

Never over-estimating the true cost.

Properties of A* heuristic

Necessary and sufficient for optimality

Admissible

Never over-estimating the true cost.

Sufficient but not necessary condition:

Consistent

Never over-estimating the growth of the path cost.

Properties of A* heuristic

Necessary and sufficient for optimality

Admissible

$0 \leq h(n) \leq h^*(n)$

Properties of A* heuristic

Necessary and sufficient for optimality

Admissible

$0 \leq h(n) \leq h^*(n)$

Consistent

$h(n) \leq h(n') + c(n, n')$

Properties of A* heuristic

Necessary and sufficient for optimality

Admissible

$0 \leq h(n) \leq h^*(n)$

Consistent

$h(n) \leq h(n') + c(n, n')$

Exercise: prove that consistent implies admissible

Properties of A* heuristic

Necessary and sufficient for optimality

Admissible

$0 \leq h(n) \leq h^*(n)$

Consistent

$h(n) \leq h(n') + c(n, n')$

Exercise: prove that consistent implies admissible

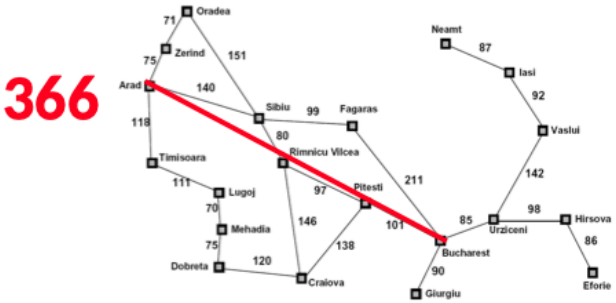
Exercise: give a heuristic that is admissible but not consistent

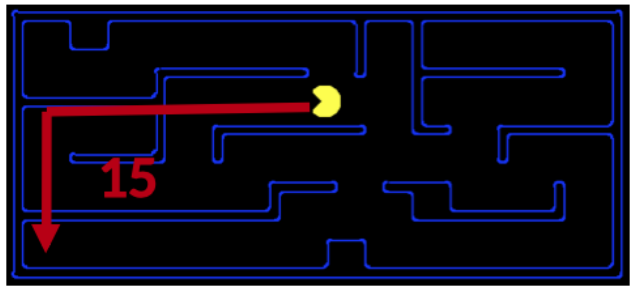
15 minute(s) break!

How to find an admissible heuristic?

This is the challenge

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics



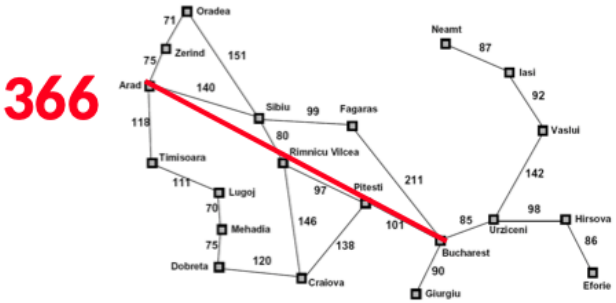


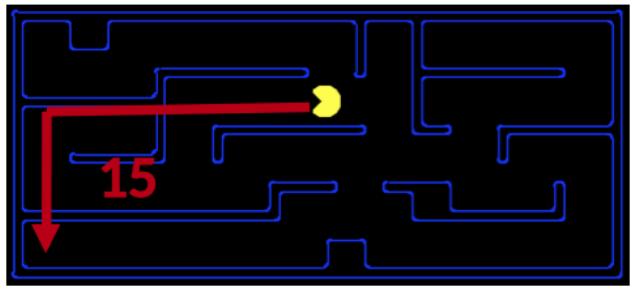
https://inst.eecs.berkeley.edu/~cs188/su20/

How to find an admissible heuristic?

This is the challenge

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to relaxed problems, where new actions are available



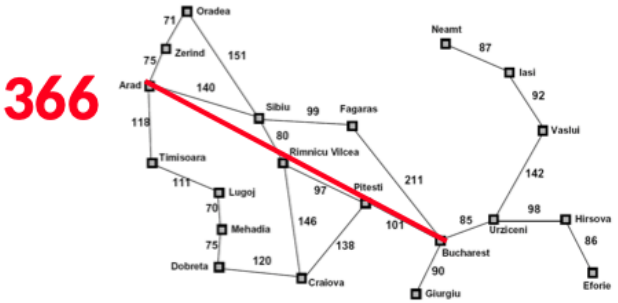


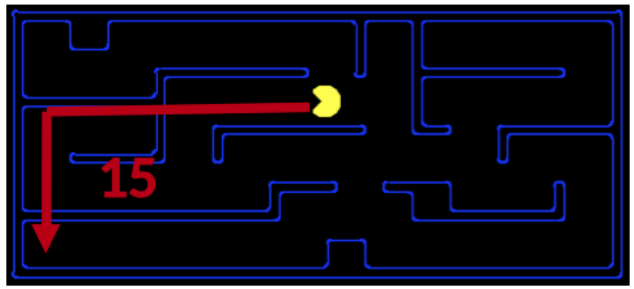
https://inst.eecs.berkeley.edu/~cs188/su20/

How to find an admissible heuristic?

This is the challenge

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to relaxed problems, where new actions are available
- Inadmissible heuristics are often useful too



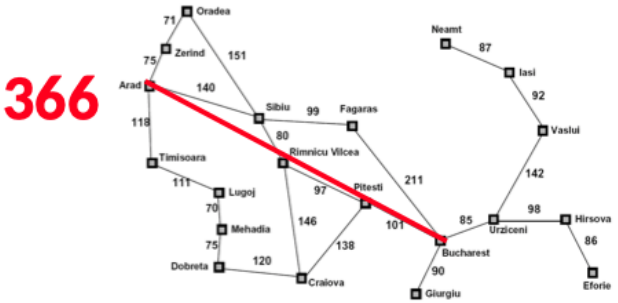


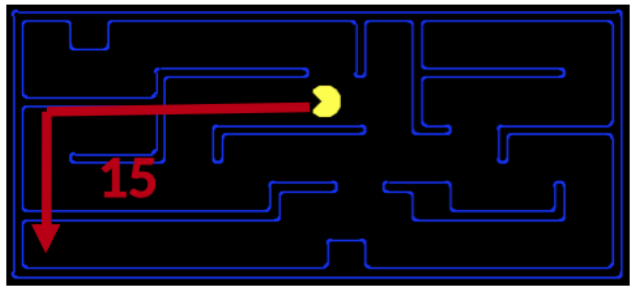
https://inst.eecs.berkeley.edu/~cs188/su20/

How to find an admissible heuristic?

This is the challenge

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to relaxed problems, where new actions are available
- Inadmissible heuristics are often useful too



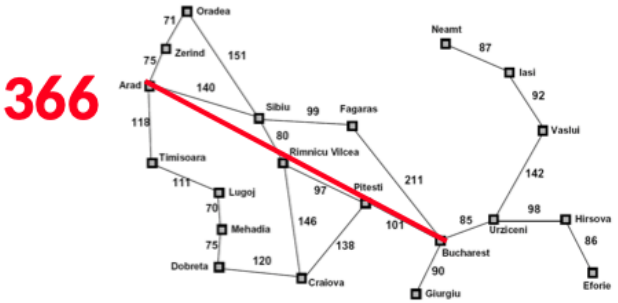


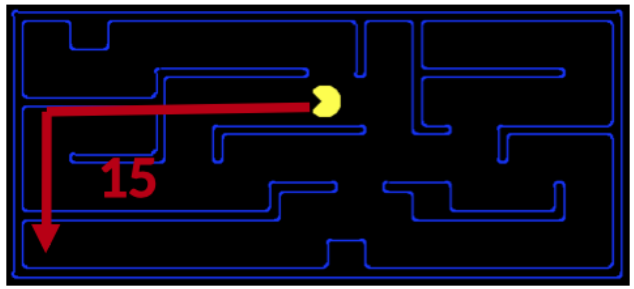
https://inst.eecs.berkeley.edu/~cs188/su20/

How to find an admissible heuristic?

This is the challenge

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to relaxed problems, where new actions are available
- Inadmissible heuristics are often useful too





https://inst.eecs.berkeley.edu/~cs188/su20/

Optimality revisited

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case ($h = 0$)

Optimality revisited

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case ($h = 0$)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal ($h = 0$ is consistent)

Optimality revisited

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case ($h = 0$)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal ($h = 0$ is consistent)
- Consistency implies admissibility, converse is false
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems

Optimal efficiency

for more details see Dechter & Pearl (1985)

Optimal efficiency

Optimality

Finds a shortest path.

A* is optimal if the heuristic is admissible.

for more details see Dechter & Pearl (1985)

Optimal efficiency

Optimality

Finds a shortest path.

A* is optimal if the heuristic is admissible.

Optimally efficient

Does the least work of any such algorithm.

A* is optimally efficient if the heuristic is consistent.

for more details see Dechter & Pearl (1985)

Optimal efficiency

Optimality

Finds a shortest path.

A* is optimal if the heuristic is admissible.

Optimally efficient

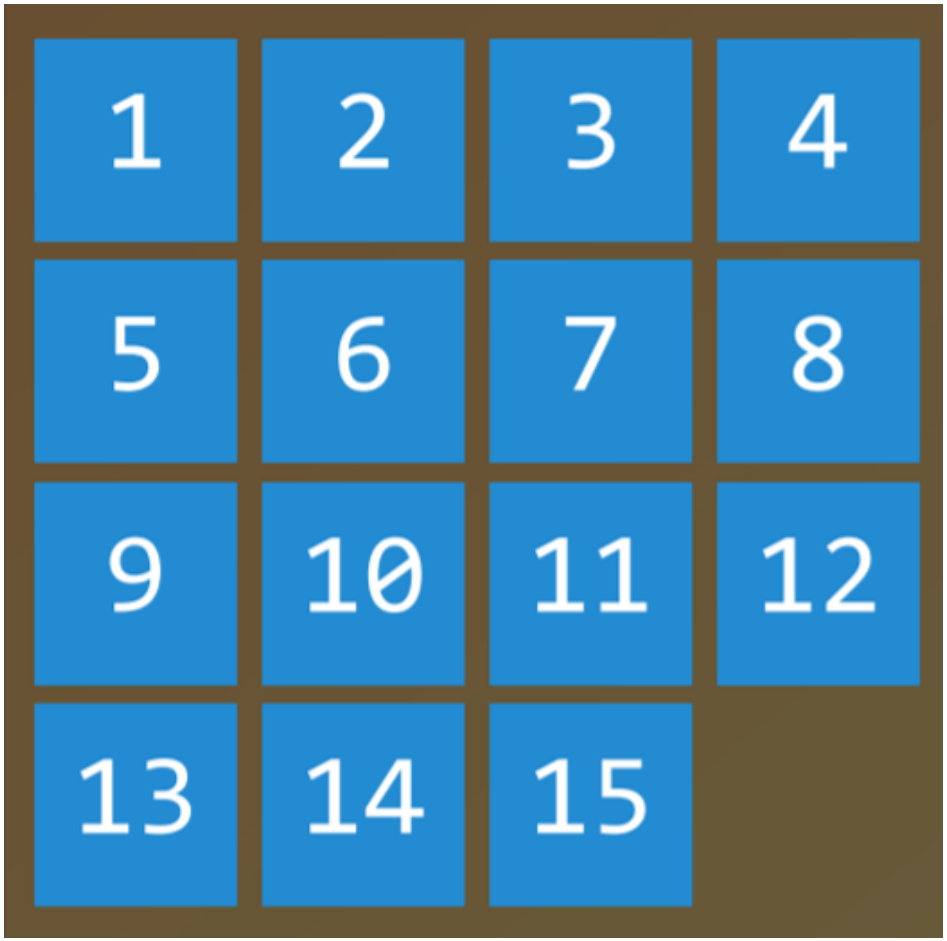
Does the least work of any such algorithm.

A* is optimally efficient if the heuristic is consistent.

Efficiency = set of nodes expanded, not number of node expansions

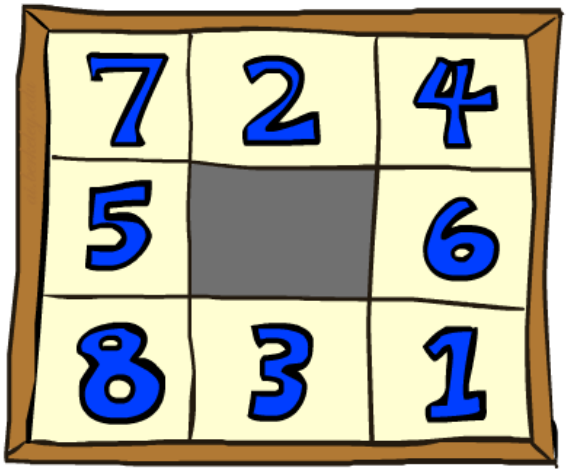
for more details see Dechter & Pearl (1985)

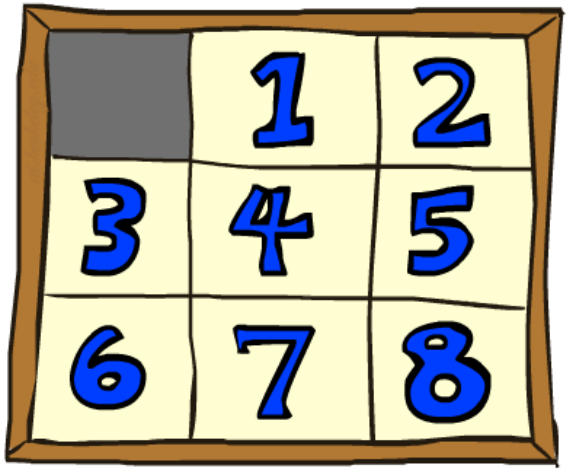
Interactive block II: Admissible heuristic for sliding puzzle



https://cs50.harvard.edu/ai/2020

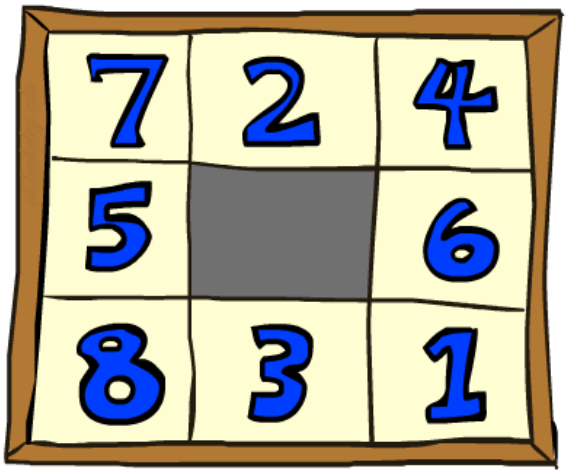
Interactive block II: Admissible heuristic for sliding puzzle

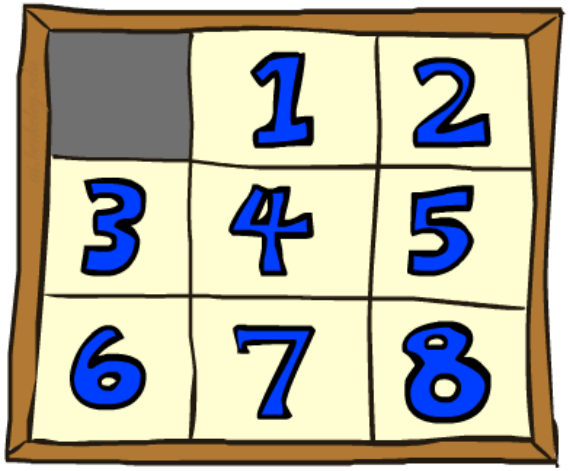




https://inst.eecs.berkeley.edu/~cs188/su20/

Interactive block II: Admissible heuristic for sliding puzzle

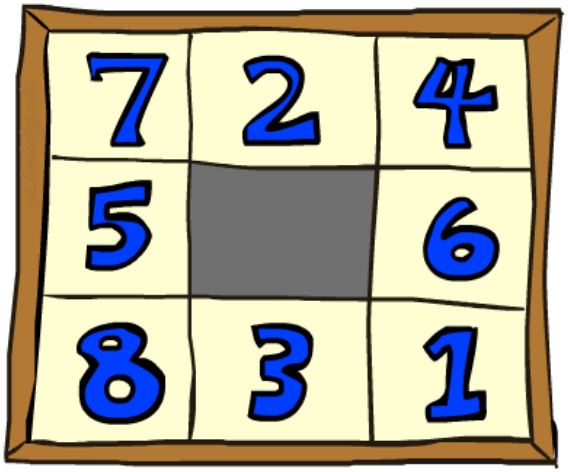


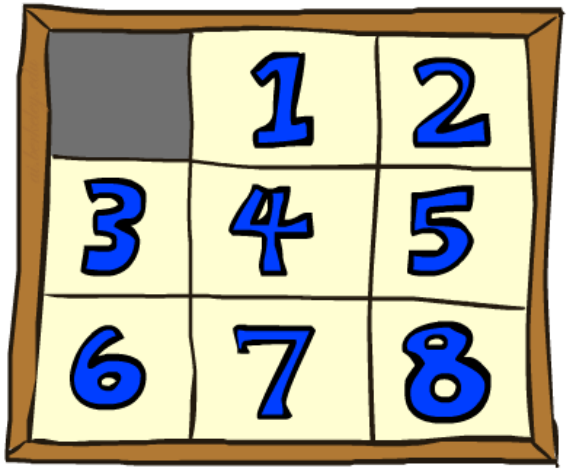


- Number of tiles misplaced

https://inst.eecs.berkeley.edu/~cs188/su20/

Interactive block II: Admissible heuristic for sliding puzzle

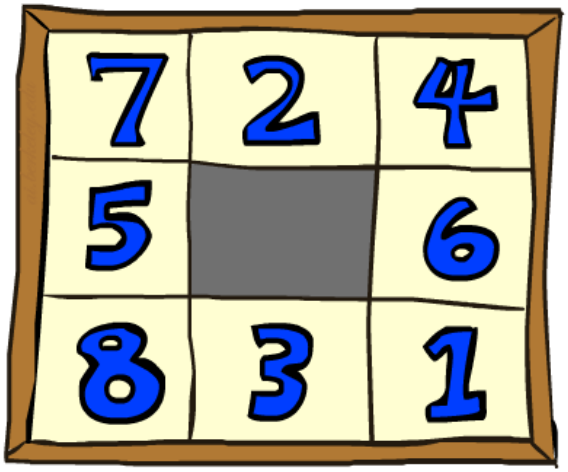


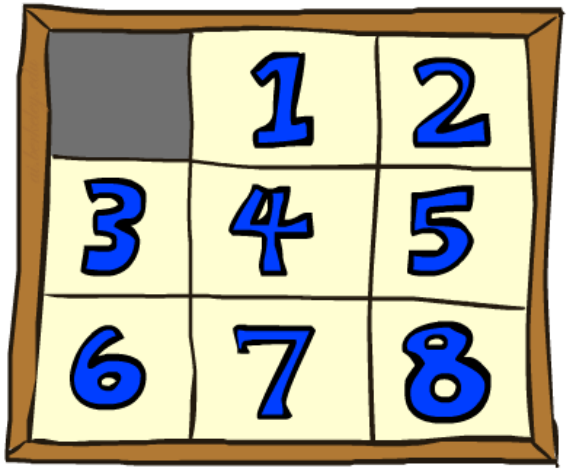


- Number of tiles misplaced
- Total Manhattan distance

https://inst.eecs.berkeley.edu/~cs188/su20/

Interactive block II: Admissible heuristic for sliding puzzle

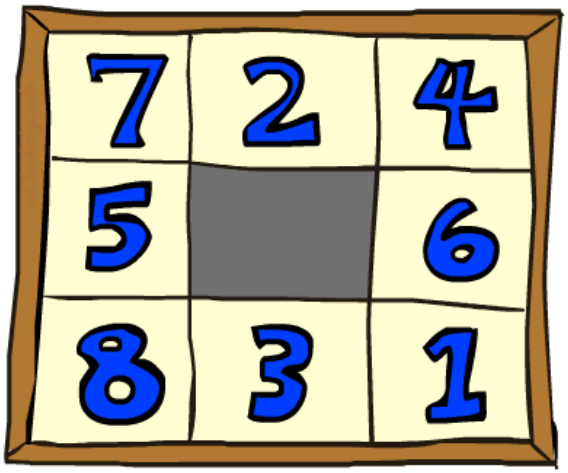


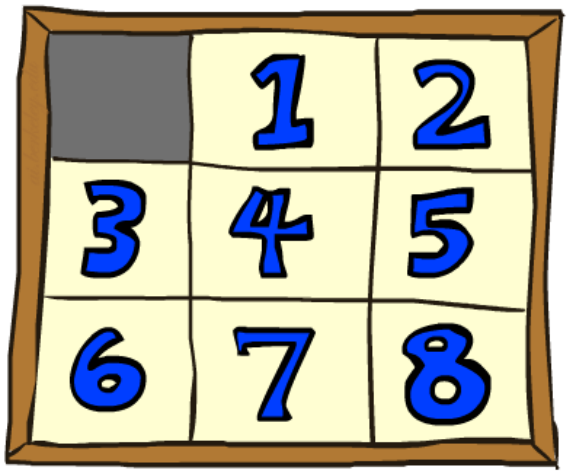


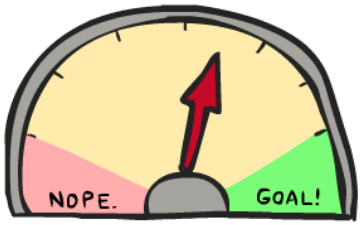
- Number of tiles misplaced
- Total Manhattan distance
- Actual cost to solution?

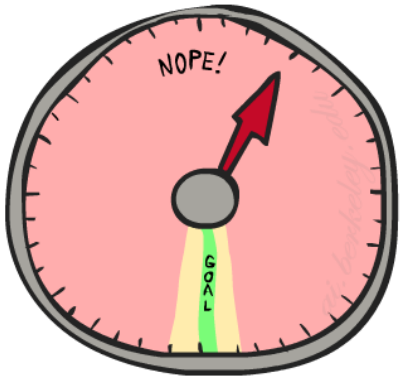
https://inst.eecs.berkeley.edu/~cs188/su20/

Interactive block II: Admissible heuristic for sliding puzzle









https://inst.eecs.berkeley.edu/~cs188/su20/

A* extensions

- Anytime A*
- Block A*
- D*
- Field D*
- Fringe
- Fringe Saving A* (FSA*)
- Generalised Adaptive A* (GAA*)
- Incremental heuristic search
- Reduced A*
- **Iterative deepening A*** (IDA*)
- Jump point search
- Lifelong Planning A* (LPA*)
- New Bidirectional A* (NBA*)
- Simplified Memory bounded A* (SMA*)
- Theta*

Next lecture

- Today we saw A* search

Next lecture

- Today we saw A* search
  - Sorry, no OXO today ☺

Next lecture

- Today we saw A* search
  - Sorry, no OXO today ☺
  - A* is optimal + optimally efficient, but space...

Next lecture

- Today we saw A* search
  - Sorry, no OXO today ☺
  - A* is optimal + optimally efficient, but space...
- Next time: adversarial search

Next lecture

- Today we saw A* search
  - Sorry, no OXO today ☺
  - A* is optimal + optimally efficient, but space...
- Next time: adversarial search
- Preparation: see Brightspace

Next lecture

- Today we saw A* search
  - Sorry, no OXO today ☺
  - A* is optimal + optimally efficient, but space...
- Next time: adversarial search
- Preparation: see Brightspace
- Lab on Monday

Next lecture

- Today we saw A* search
  - Sorry, no OXO today ☺
  - A* is optimal + optimally efficient, but space...
- Next time: adversarial search
- Preparation: see Brightspace
- Lab on Monday
- Assignment 1 due: see Brightspace

Next lecture

- Today we saw A* search
  - Sorry, no OXO today ☺
  - A* is optimal + optimally efficient, but space...
- Next time: adversarial search
- Preparation: see Brightspace
- Lab on Monday
- Assignment 1 due: see Brightspace
- Questions? TUD Answers

What is still unclear?

After every lecture...

Give us some homework:
tell us what is still unclear.

ALGORITHMS FOR NP-HARD PROBLEMS

Part I – Heuristic Search

Lecture 3: Adversarial Search

Neil Yorke-Smith

n.yorke-smith@tudelft.nl

February 2023

Where are we?

- Last time: A* search
- Today: minimax search

Where are we?

- Last time: A* search
- Today: minimax search
- You prepared:
  - Paper about Deep Blue

Adversary!

Regular search

Find an answer to a question.

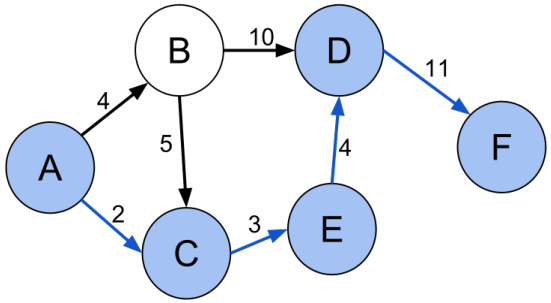


Photo by Mesh on Unsplash

Adversary!

Regular search

Find an answer to a question.

Adversarial search

But now an opponent that tries to achieve the opposite goal.

Photo by Mesh on Unsplash

Adversary!

Regular search

Find an answer to a question.

Adversarial search

But now an opponent that tries to achieve the opposite goal.

Two player, sequential deterministic moves, perfect information, discrete zero-sum

Photo by Mesh on Unsplash

Terminology of games

- single/**two**/multi-player game
  - Solitaire/Chess/Settlers-of-Catan

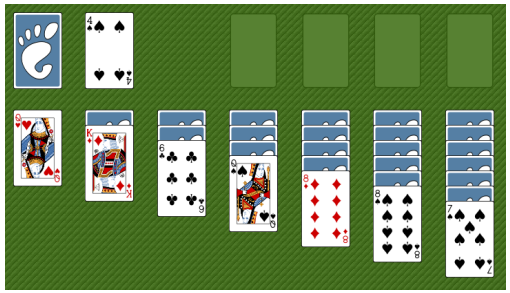


Image sources: `wikimedia.org`

Terminology of games

- single/**two**/multi-player game
  - Solitaire/Chess/Settlers-of-Catan
- simultaneous/**sequential** moves
  - Rock-paper-scissors/Go

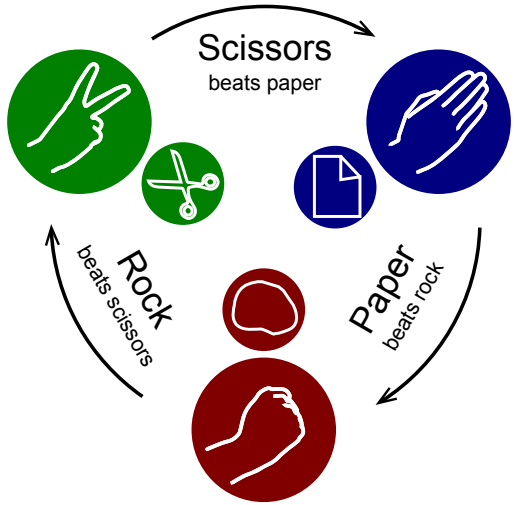


Image sources: `wikimedia.org`

Terminology of games

- single/**two**/multi-player game
  - Solitaire/Chess/Settlers-of-Catan
- simultaneous/**sequential** moves
  - Rock-paper-scissors/Go
- stochastic/**deterministic** moves
  - Poker/Chess

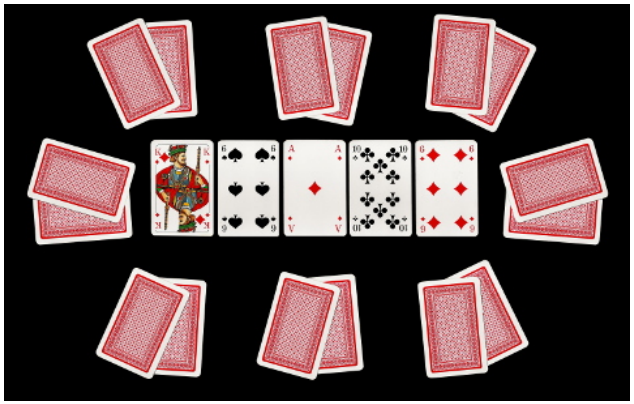


Image sources: `wikimedia.org`

Terminology of games

- single/**two**/multi-player game
  - Solitaire/Chess/Settlers-of-Catan
- simultaneous/**sequential** moves
  - Rock-paper-scissors/Go
- stochastic/**deterministic** moves
  - Poker/Chess
- partial/**perfect** information
  - Card-games/Connect-four

Image sources: `wikimedia.org`

Terminology of games

- single/**two**/multi-player game
  - Solitaire/Chess/Settlers-of-Catan
- simultaneous/**sequential** moves
  - Rock-paper-scissors/Go
- stochastic/**deterministic** moves
  - Poker/Chess
- partial/**perfect** information
  - Card-games/Connect-four
- **discrete**/continual state/actions/time
  - turn-based/real-time strategy

Image sources: `wikimedia.org`, Samvelyan et al. (2019)

Terminology of games

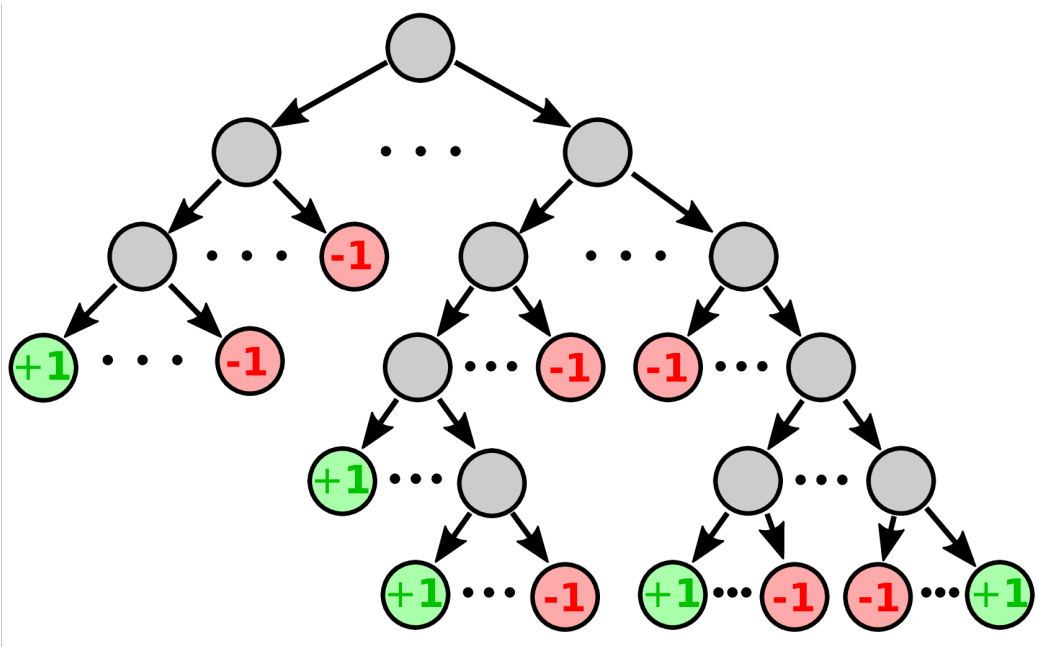
- single/**two**/multi-player game
  - Solitaire/Chess/Settlers-of-Catan
- simultaneous/**sequential** moves
  - Rock-paper-scissors/Go
- stochastic/**deterministic** moves
  - Poker/Chess
- partial/**perfect** information
  - Card-games/Connect-four
- **discrete**/continual state/actions/time
  - turn-based/real-time strategy
- **zero-sum**/general-sum game
  - Chess/Settlers-of-Catan

Image sources: wikimedia.org

Minimax evaluation

First, search trees as we did so far

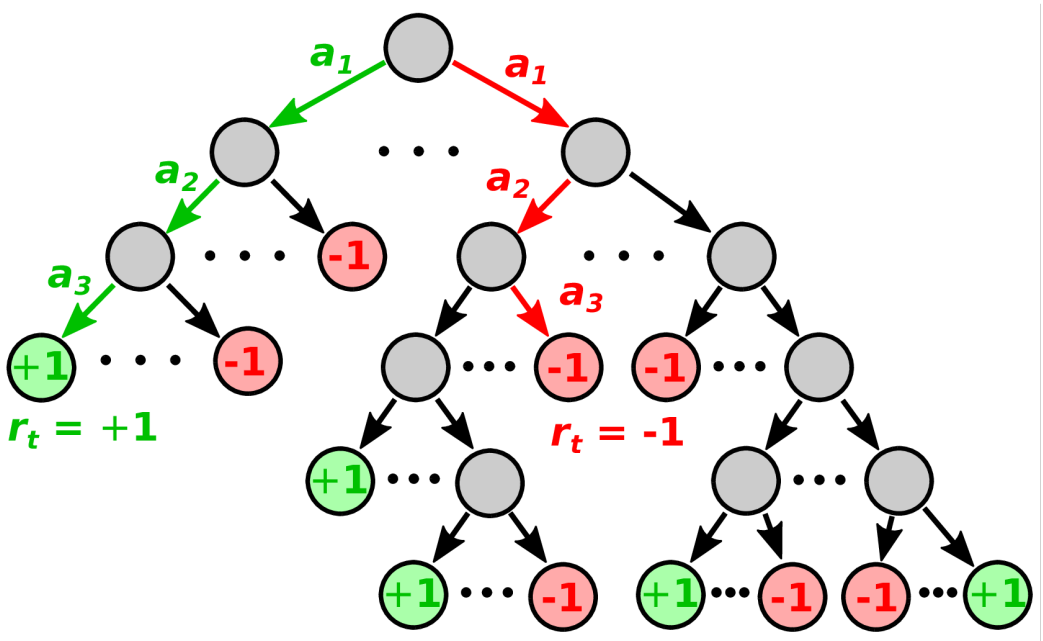
Suppose value $+1$ for a win, -1 for a lose



Minimax evaluation

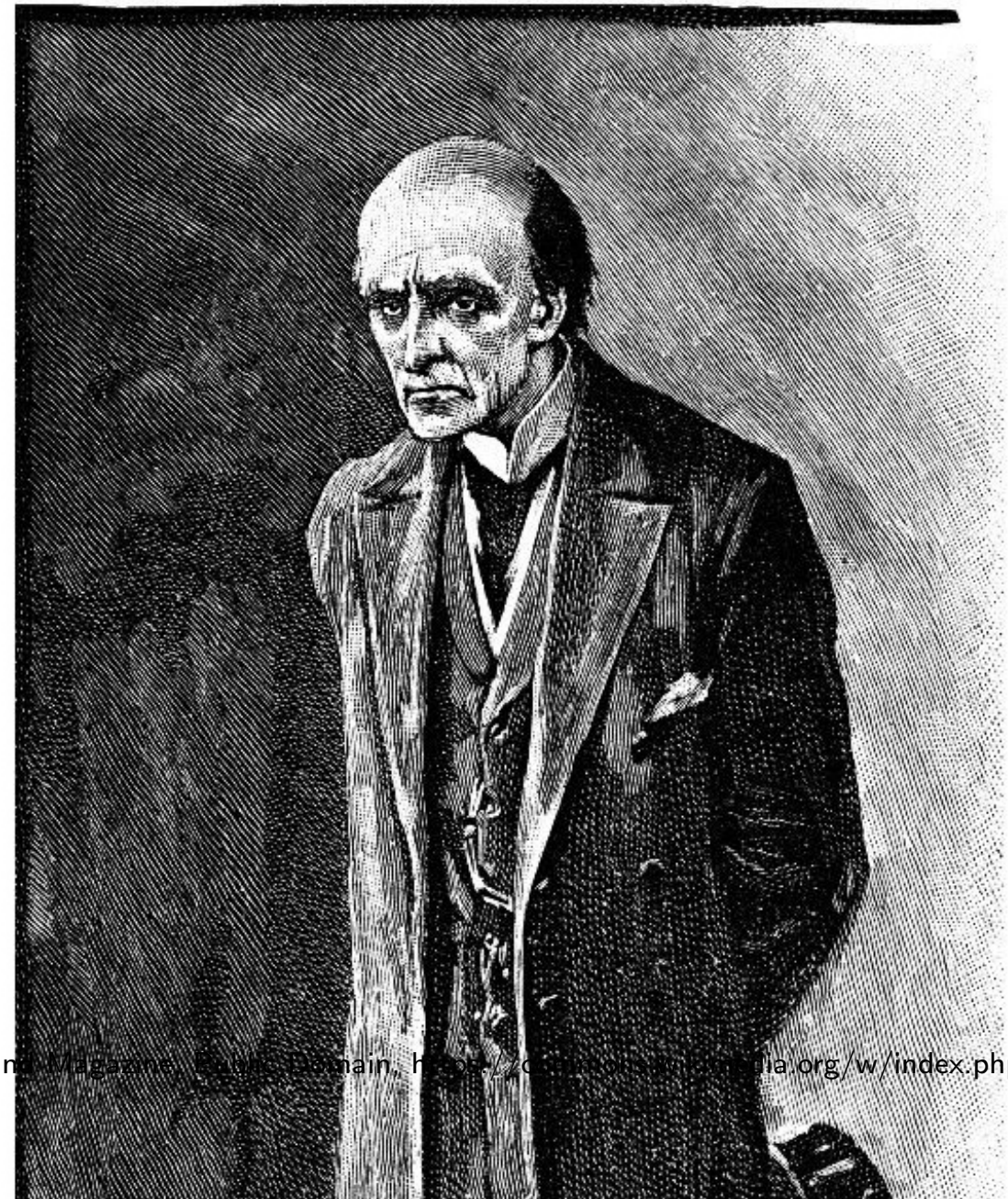
First, search trees as we did so far

Winning path, losing path



Minimax evaluation

Adversary moves

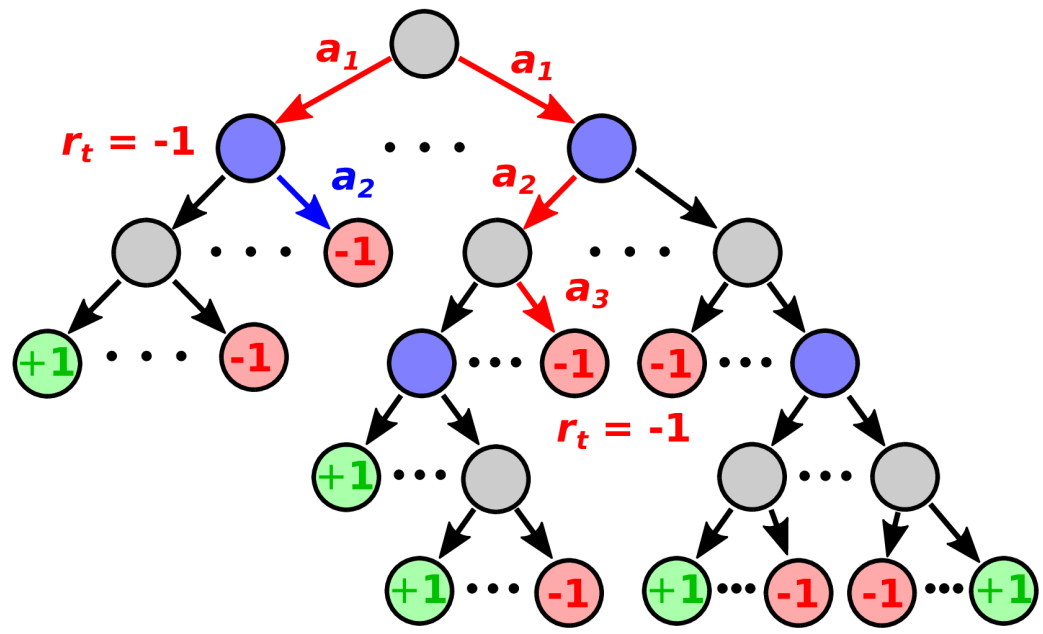


By Sidney Paget - The Strand Magazine, Public Domain, https://commons.wikimedia.org/w/index.php?curid=69209162

Minimax evaluation

Adversary moves

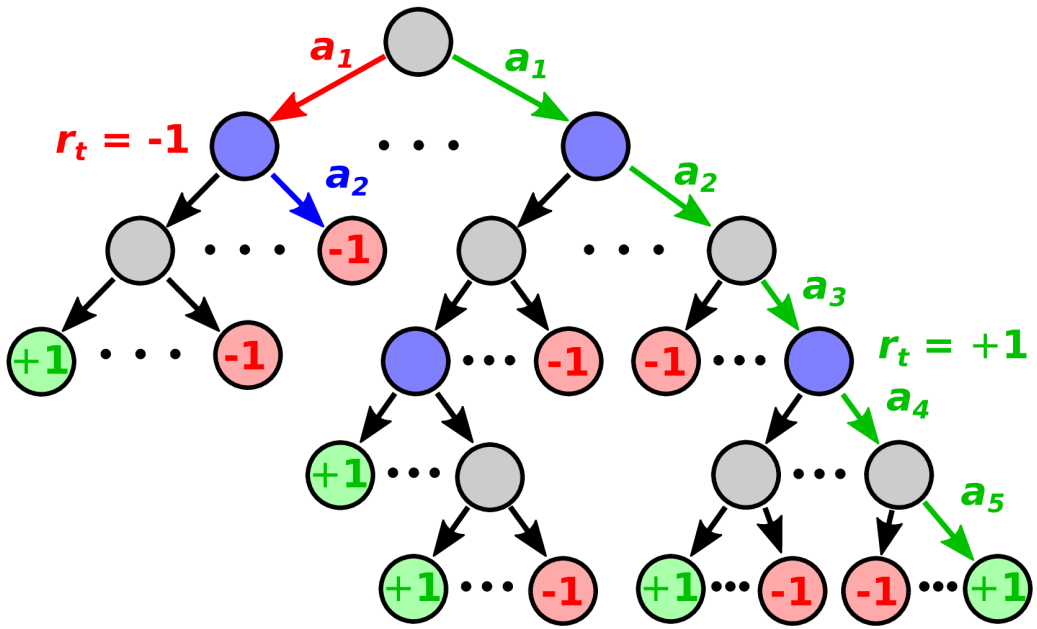
Adversary's plays $\Longrightarrow$ left-hand a_1 is now a losing move!



Minimax evaluation

Adversary moves

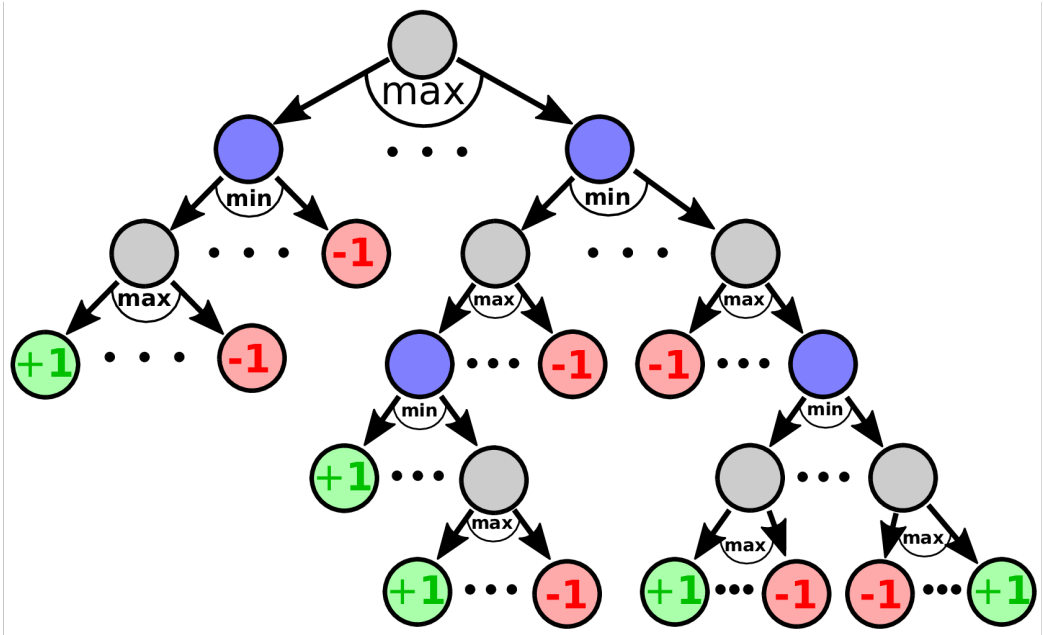
So our moves must account for his. How?



Minimax evaluation

Max for me, min for him

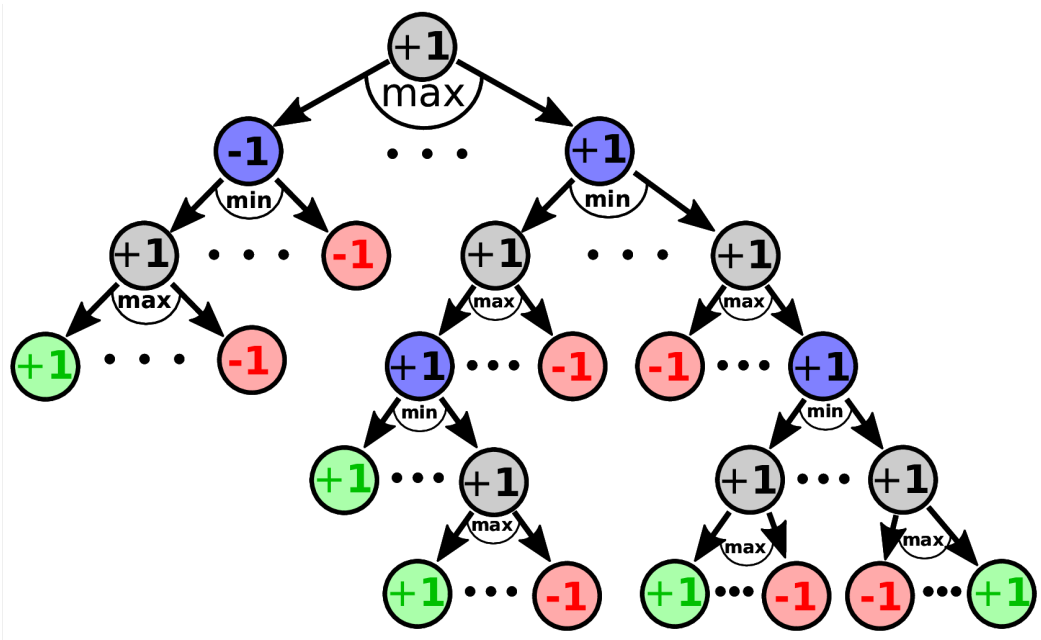
Key idea: play value-maximising moves, assuming that adversary plays value-*minimising* moves.



Minimax evaluation

Max for me, min for him

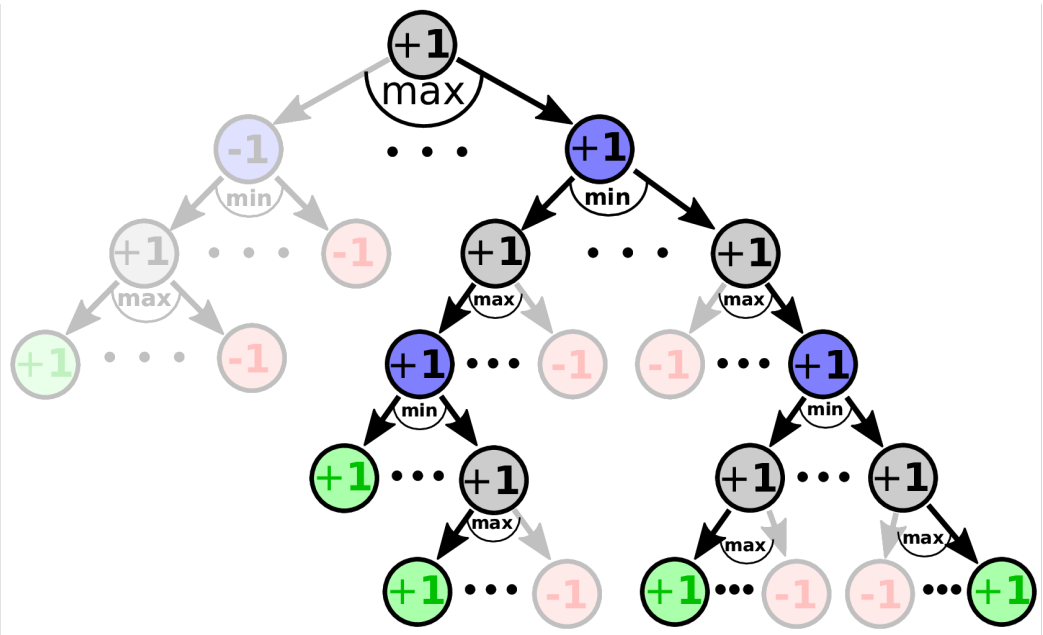
At each node, compute minimax score from node's children



Minimax evaluation

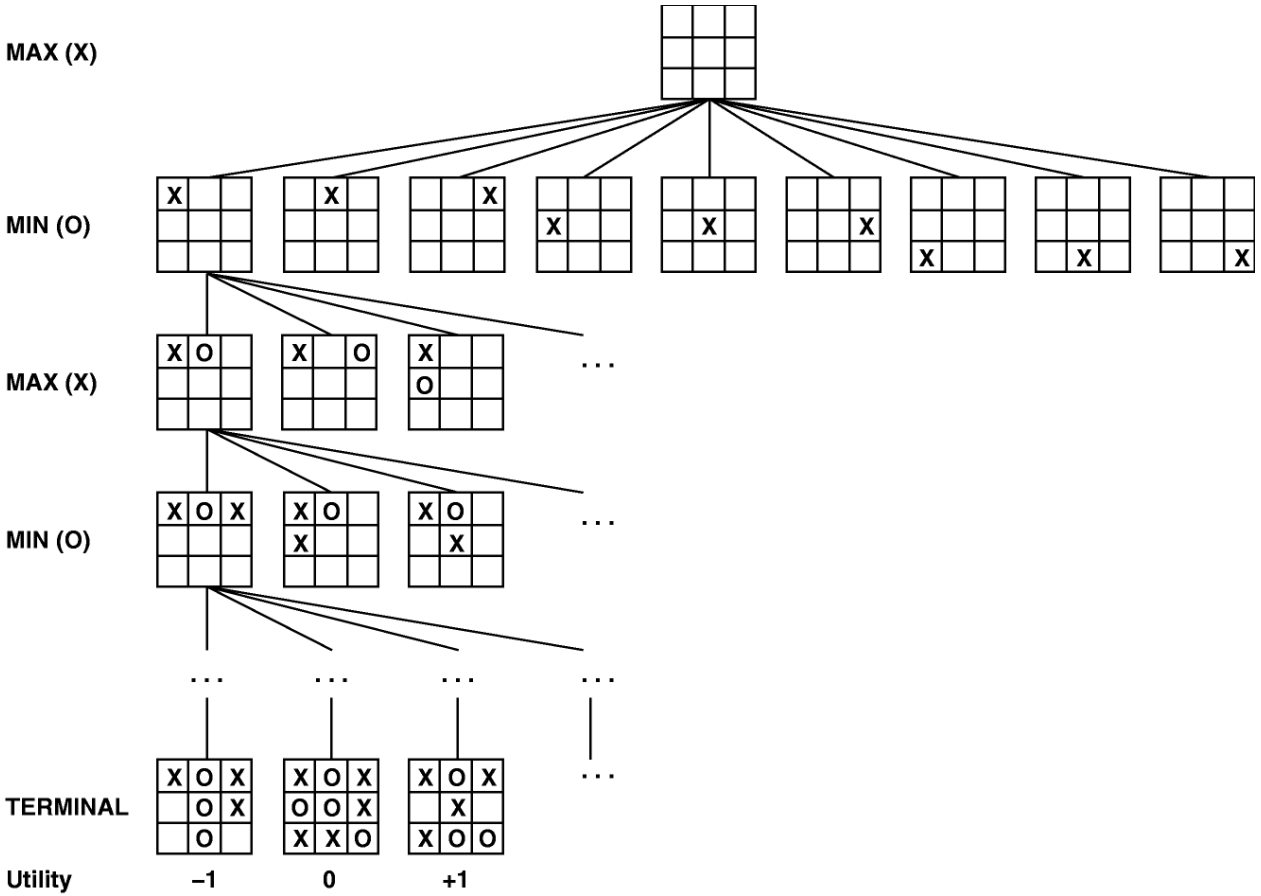
Max for me, min for him

Choose moves that lead to maximum score



Search tree for OXO

Remember this?



https://www.massey.ac.nz/~mjjohnso/

Interactive block I: Minimax evaluation for OXO

Suppose X goes first, in the top-left corner

Write down the whole search tree, assuming an adversary who plays optimally.

Improving minimax

What now?

Minimax search doesn't scale

Works for OXO, not for chess.

Improving minimax

What now?

Minimax search doesn't scale

Works for OXO, not for chess. Or shogi, or go, or poker, ...

Improving minimax

What now?

Minimax search doesn't scale

Works for OXO, not for chess. Or shogi, or go, or poker, ...

Ideas

- Don't fathom the tree – depth-limited

Improving minimax

What now?

Minimax search doesn't scale

Works for OXO, not for chess. Or shogi, or go, or poker, ...

Ideas

- Don't fathom the tree – depth-limited
- Prune useless subtrees – alpha-beta pruning

Improving minimax

What now?

Minimax search doesn't scale

Works for OXO, not for chess. Or shogi, or go, or poker, ...

Ideas

- Don't fathom the tree – depth-limited
- Prune useless subtrees – alpha-beta pruning
- Search stochastically – MCTS ☞ Lecture 4

Depth-limited minimax search

Don't explore the whole tree

```
fun minimax(n: node): int =
   if leaf(n) then return evaluate(n)
   if n is a max node
      v := L
      for each child of n
         v' := minimax (child)
         if v' > v, v:= v'
      return v
   if n is a min node
      v := W
      for each child of n
         v' := minimax (child)
         if v' < v, v:= v'
      return v
```

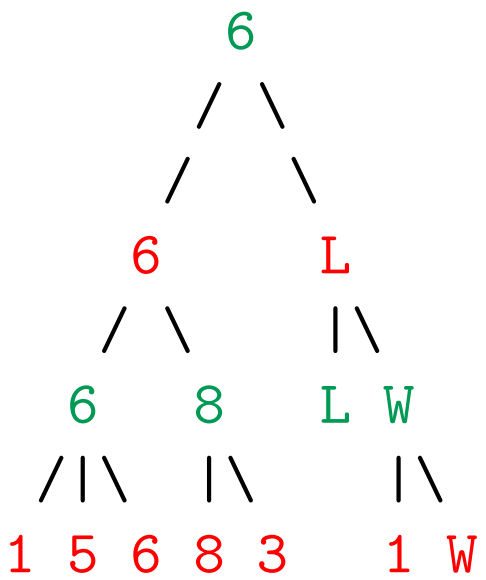
Depth-limited minimax search

Don't explore the whole tree

```
fun minimax(n: node, d: int): int =
   if leaf(n) or d=0 return evaluate(n)
   if n is a max node
      v := L
      for each child of n
         v' := minimax (child,d-1)
         if v' > v, v:= v'
      return v
   if n is a min node
      v := W
      for each child of n
         v' := minimax (child,d-1)
         if v' < v, v:= v'
      return v
```

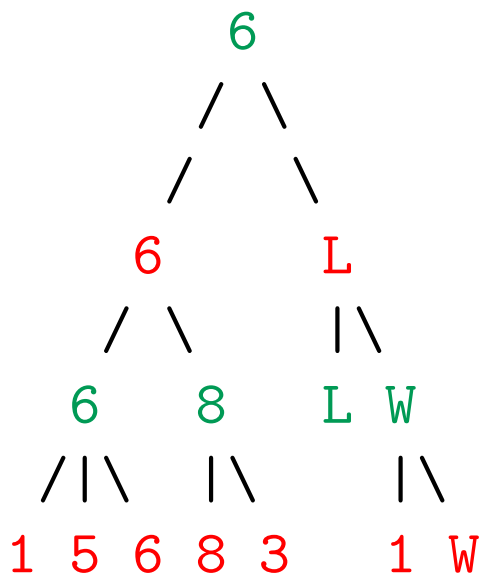
Depth-limited minimax search

Don't explore the whole tree



Depth-limited minimax search

Don't explore the whole tree



How deep to search?

15 minute(s) break!

Improving minimax

What now?

Minimax search doesn't scale

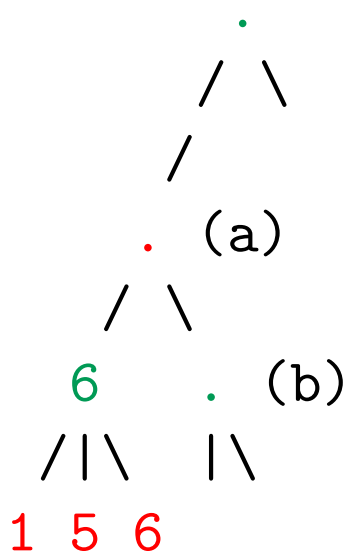
Works for OXO, not for chess. Or shogi, or go, or poker, ...

Ideas

- Don't fathom the tree – depth-limited
- Prune useless subtrees – alpha-beta pruning
- Search stochastically – MCTS ☞ Lecture 4

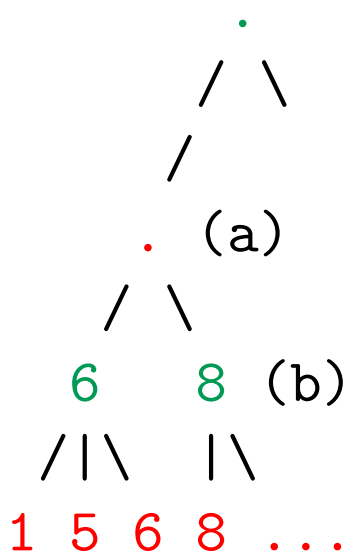
Alpha-beta pruning

Prune useless subtrees



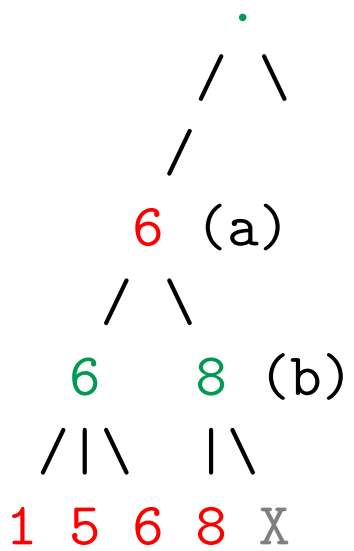
Alpha-beta pruning

Prune useless subtrees



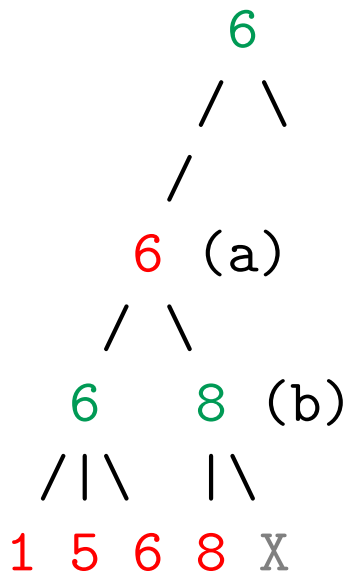
Alpha-beta pruning

Prune useless subtrees



Alpha-beta pruning

Prune useless subtrees



Alpha-beta pruning

Prune useless subtrees

```
fun minimax(n: node, d: int, min: int, max: int): int =
  if leaf(n) or d=0 return evaluate(n)
  if n is a max node
     v := min
     for each child of n
        v' := minimax (child,d-1,...,...)
        if v' > v, v:= v'
        if v > max return max
     return v
  if n is a min node
     v := max
     for each child of n
        v' := minimax (child,d-1,...,...)
        if v' < v, v:= v'
        if v < min return min
     return v
```

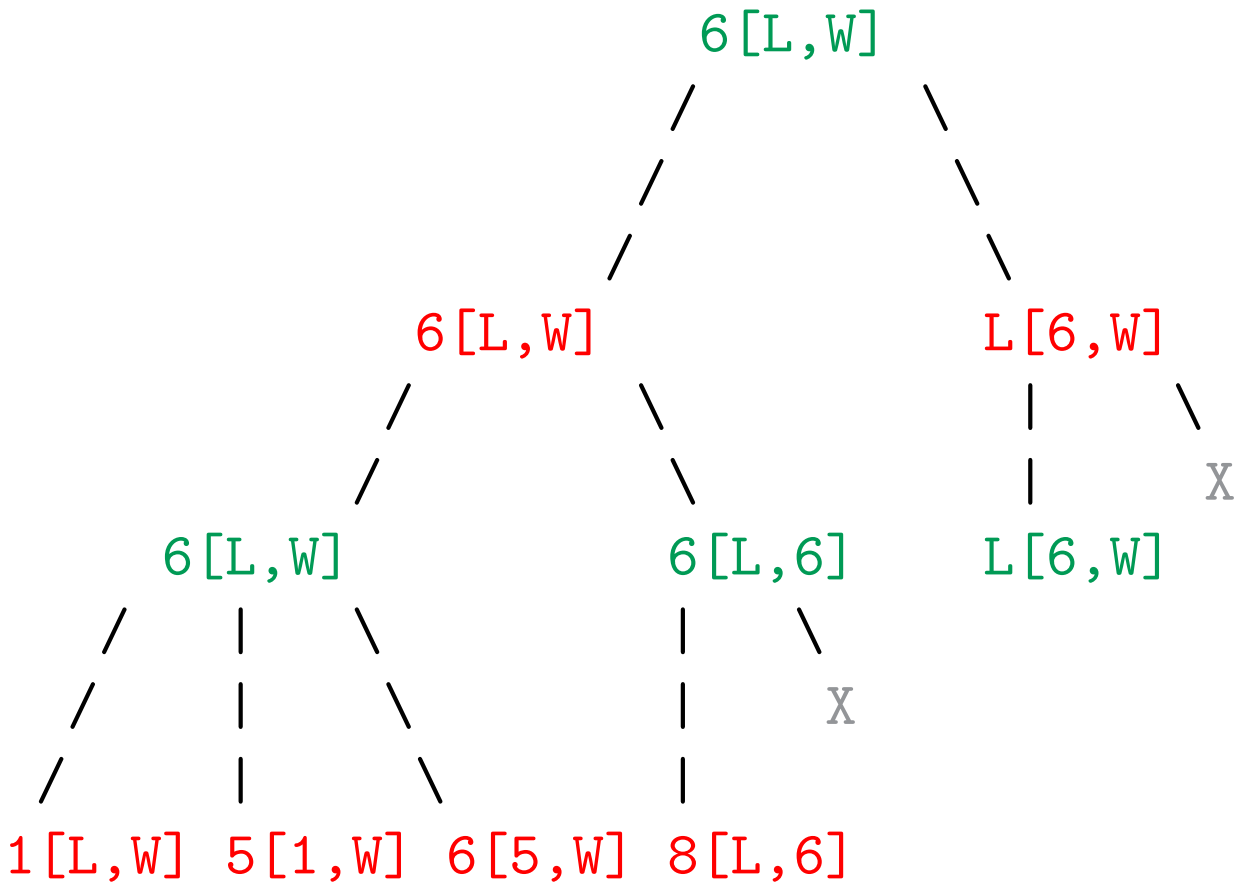
Alpha-beta pruning

Prune useless subtrees

```
fun minimax(n: node, d: int, min: int, max: int): int =
  if leaf(n) or d=0 return evaluate(n)
  if n is a max node
     v := min
     for each child of n
        v' := minimax (child,d-1,v,max)
        if v' > v, v:= v'
        if v > max return max
     return v
  if n is a min node
     v := max
     for each child of n
        v' := minimax (child,d-1,min,v)
        if v' < v, v:= v'
        if v < min return min
     return v
```

Alpha-beta pruning

Prune useless subtrees



Alpha-beta pruning

Prune useless subtrees

How to prune effectively?

Alpha-beta pruning

Prune useless subtrees

How to prune effectively?

- Order well the nodes (high max, low min)

Alpha-beta pruning

Prune useless subtrees

How to prune effectively?

- Order well the nodes (high max, low min)
- Previous (partial) searches of the tree

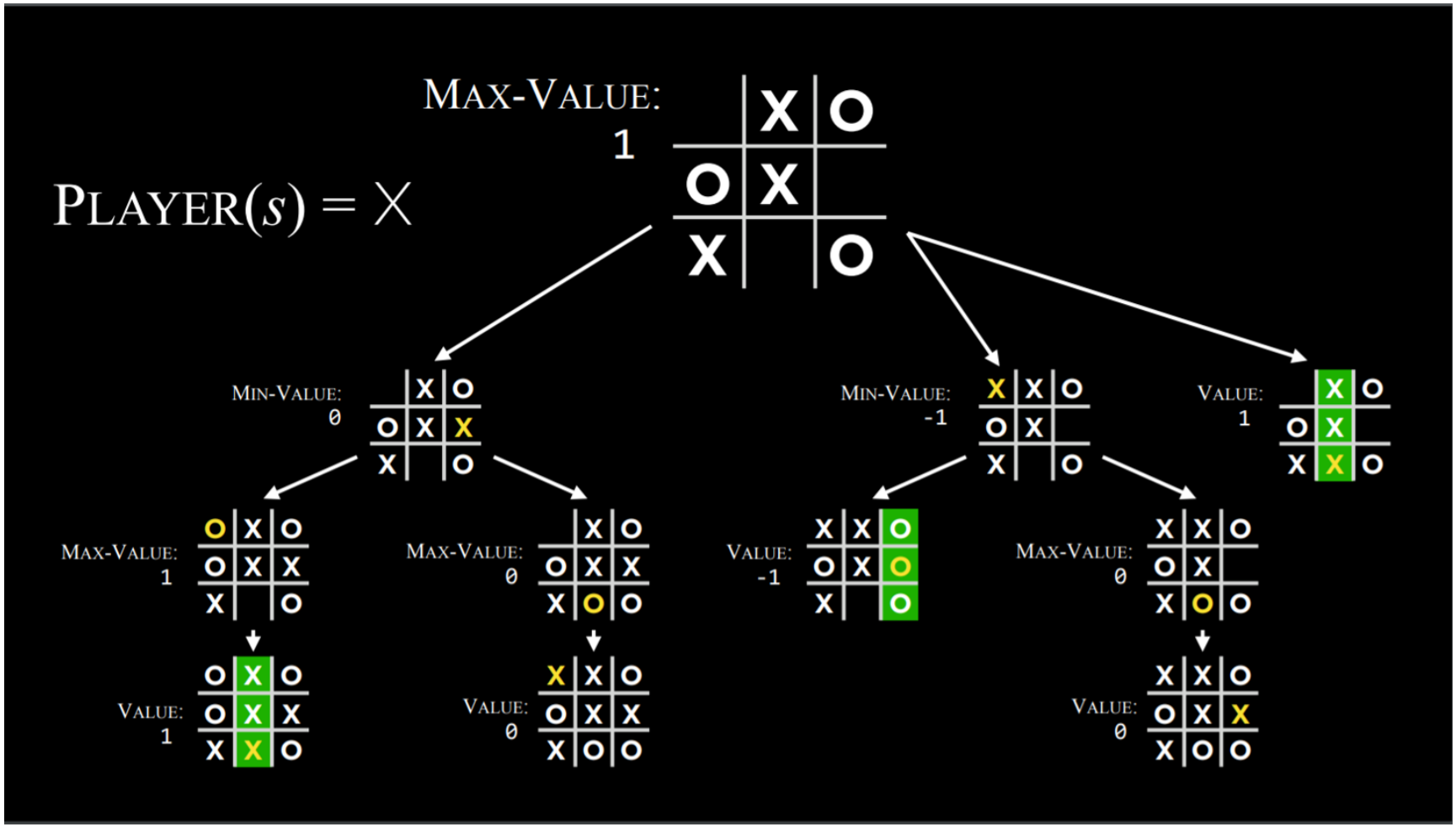
Alpha-beta pruning

Prune useless subtrees

How to prune effectively?

- Order well the nodes (high max, low min)
- Previous (partial) searches of the tree
- Good heuristic evaluation function

Interactive block II: Does α-β help here?



https://cs50.harvard.edu/ai/2020

Example: Deep Blue

Famous in 1997

By James the photographer - https://www.flickr.com/photos/[illegible]53761@N00/592436598/, CC BY 2.0, https://commons.wikimedia.org/w/index.php?curid=3511068

Learned heuristic function

Evaluation functions are subtle

Hard to code a good evaluation for complex states.

Learned heuristic function

Evaluation functions are subtle

Hard to code a good evaluation for complex states.

Example: Chess

Material + movement + king safety + passed pawns + ...

Learned heuristic function

Evaluation functions are subtle

Hard to code a good evaluation for complex states.

Example: Chess

Material + movement + king safety + passed pawns + ...

Idea: Machine learning

Efficient neural networks (NNUE).

Example: Stockfish 12

Famous in 2020

Stockfish 12 has added neural network evaluation abilities.

As of October 2020, Stockfish is the highest-rated engine according to the computer chess rating list (CCRL) with a rating of 3514—it is the only engine with a rating above 3500. According to the July 2020 Swedish Chess Computer Association (SSDF) rating list, Stockfish 9 is ranked #3, Stockfish 10 is ranked #2, and Stockfish 11 is ranked #1 with a rating of 3558. Taking the top three spots with three different versions is quite impressive.

According to this great video on the strongest chess engines of all time (based on the SSDF rating lists), Stockfish is the strongest engine of all time—a sentiment that is widely shared in the chess community.

`https://www.chess.com/terms/stockfish-chess-engine`

Next lecture

- Today we saw minimax search

Next lecture

- Today we saw minimax search
  - With enhancements and NN evaluation = state of the art for chess

Next lecture

- Today we saw minimax search
  - With enhancements and NN evaluation = state of the art for chess
- Next time: monte carlo tree search

Next lecture

- Today we saw minimax search
  - With enhancements and NN evaluation = state of the art for chess
- Next time: monte carlo tree search
  - With enhancements and NN evaluation = state of the art for go

Next lecture

- Today we saw minimax search
  - With enhancements and NN evaluation = state of the art for chess
- Next time: monte carlo tree search
  - With enhancements and NN evaluation = state of the art for go
- Preparation: see Brightspace

Next lecture

- Today we saw minimax search
  - With enhancements and NN evaluation = state of the art for chess
- Next time: monte carlo tree search
  - With enhancements and NN evaluation = state of the art for go
- Preparation: see Brightspace
- Lab on Monday

Next lecture

- Today we saw minimax search
  - With enhancements and NN evaluation = state of the art for chess
- Next time: monte carlo tree search
  - With enhancements and NN evaluation = state of the art for go
- Preparation: see Brightspace
- Lab on Monday
- Assignment 1 due: see Brightspace

Next lecture

- Today we saw minimax search
  - With enhancements and NN evaluation = state of the art for chess
- Next time: monte carlo tree search
  - With enhancements and NN evaluation = state of the art for go
- Preparation: see Brightspace
- Lab on Monday
- Assignment 1 due: see Brightspace
- Questions? TUD Answers

What is still unclear?

After every lecture...

Give us some homework:
tell us what is still unclear.

References I

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

ALGORITHMS FOR NP-HARD PROBLEMS

Part I – Heuristic Search

Lecture 4: Monte Carlo Tree Search

Wendelin Böhmer and Neil Yorke-Smith

n.yorke-smith@tudelft.nl

February 2023

Where are we?

- Last time: minimax search
- Today: exploiting stochasticity

Where are we?

- Last time: minimax search
- Today: exploiting stochasticity
- You prepared:
  - Paper about AlphaGo

Terminology of games

- single/**two**/multi-player game
  - Solitaire/Chess/Settlers-of-Catan
- simultaneous/**sequential** moves
  - Rock-paper-scissors/Go
- stochastic/**deterministic** moves
  - Poker/Chess
- partial/**perfect** information
  - Card-games/Connect-four
- **discrete**/continual state/actions/time
  - turn-based/real-time strategy
- **zero-sum**/general-sum game
  - Chess/Settlers-of-Catan

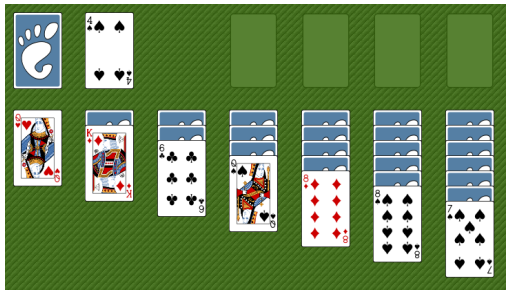


Image sources: `wikimedia.org`

Formal definition

Markov Decision Process

- Deterministic episodic two-player MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{T}, \mathcal{R} \rangle$
  - $s \in \mathcal{S}$ denotes all game-states [discrete, perfect information]
  - $a \in \mathcal{A}(s)$ are all actions that can be played in s [discrete, sequential]
  - $\mathcal{P}(s, a) = s'$ determines next state $s' \in \mathcal{S}$ [deterministic moves]
  - $\mathcal{T}(s) \in \{\top, \bot\}$ whether the game terminates (ends) in s [episodic]
  - $\mathcal{R}(s) \in \mathbb{R}$ returns the reward for terminal state s
  - $p(s) \in \{-1, 1\}$ returns which player's move it is [two sequential players]

- Note that we only consider *reward* at the end of the game
  - `+1` for winning, `0` for draw, `-1` for loss
  - other definitions are possible, but can change solution

State interface (1)

- two-player (1 vs. -1) zero-sum games with perfect information

```python
class State:
    def __init__(self, player=1):
        self.player = player          # p(s) ∈ {−1, 1}

    def actions(self):                # A(s)
        return []     # list of legal actions

    def transition(self, action):     # P(s, a) ∈ S
        return None   # new state after action

    def terminal(self):               # T(s) ∈ {⊤, ⊥}
        return None   # boolean whether the game ends

    def reward(self):                 # R(s) ∈ ℝ
        return None   # 1 for won, 0 for draw and -1 for lost
```

Action interface

- two-player (1 vs. -1) zero-sum games with perfect information
- discrete sequential deterministic moves

```
 1 class Action:
 2     def __init__(self, player, x, ...):
 3         self.player = player
 4         self.x = x
 5         # more parameters describing the action
 6 
 7     def __str__(self):          # to look better
 8         return str(self.x)
 9 
10     def __eq__(self, other): # for comparisons ('if a in list')
11         return self.player == other.player and self.x == other.x
12 
13     def __hash__(self):         # for use in dictionaries (hash maps)
14         return hash((self.player, self.x))
```

Example: Noughts-and-crosses (OXO)

```
 1 class NoughtsAndCrosses (State):
 2     def __init__(self, player=1):
 3         super().__init__(player=player)
 4         self.board = [[0 for _ in range(3)] for _ in range(3)]
 5
 6     def actions(self):                    # A(s)
 7         # returns a list with an action for each 0 in self.board
 8
 9     def transition(self, action):         # P(s,a) ∈ S
10         state = deepcopy(self)
11         state.board[action.x][action.y] = action.player
12         state.player *= -1
13         return state
14
15     def reward(self):                     # R(s) ∈ ℝ
16         # checks all rows, columns and diagonals <x>:
17         # return +1 if sum(x) == +3
18         # return -1 if sum(x) == -3
19         # return  0 if no x triggered a return
20
21     def terminal(self):                   # T(s) ∈ {⊤,⊥}
22         return self.reward() != 0 or len(self.actions()) == 0
```

Interactive block I: two-player games

Multiple (or no) answers may be correct!

- How many distinct states has noughts-and-crosses (OXO)?
  - A) $n = 2^9$
  - B) $n = 3^9$
  - C) $n = 9!$

- Which *properties* has the game backgammon?
  - E) simultaneous moves
  - F) deterministic moves
  - G) perfect information
  - H) continuous time
  - I) discrete states

image source `wikimedia.org`

Interactive block I: two-player games

Multiple (or no) answers may be correct!

- How many distinct states has noughts-and-crosses (OXO)?
  - A $n = 2^9$
  - B $n = 3^9$
  - C $n = 9!$

X	X	X

not reachable
$\Rightarrow n \neq x^9$

X		X
	O	

reachable by two paths
$\Rightarrow n \neq 9!$

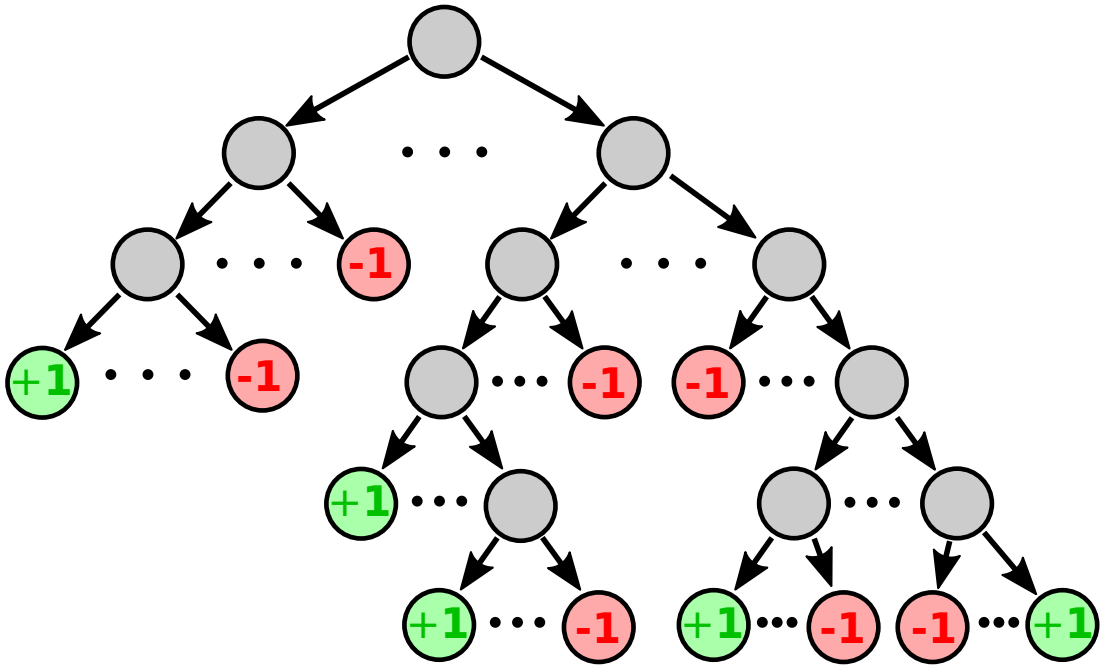
- Which *properties* has the game backgammon?
  - E simultaneous moves
  - F deterministic moves
  - G perfect information $\surd$
  - H continuous time
  - I discrete states $\surd$

image source `wikimedia.org`

Decision trees (1)

Recall last lecture

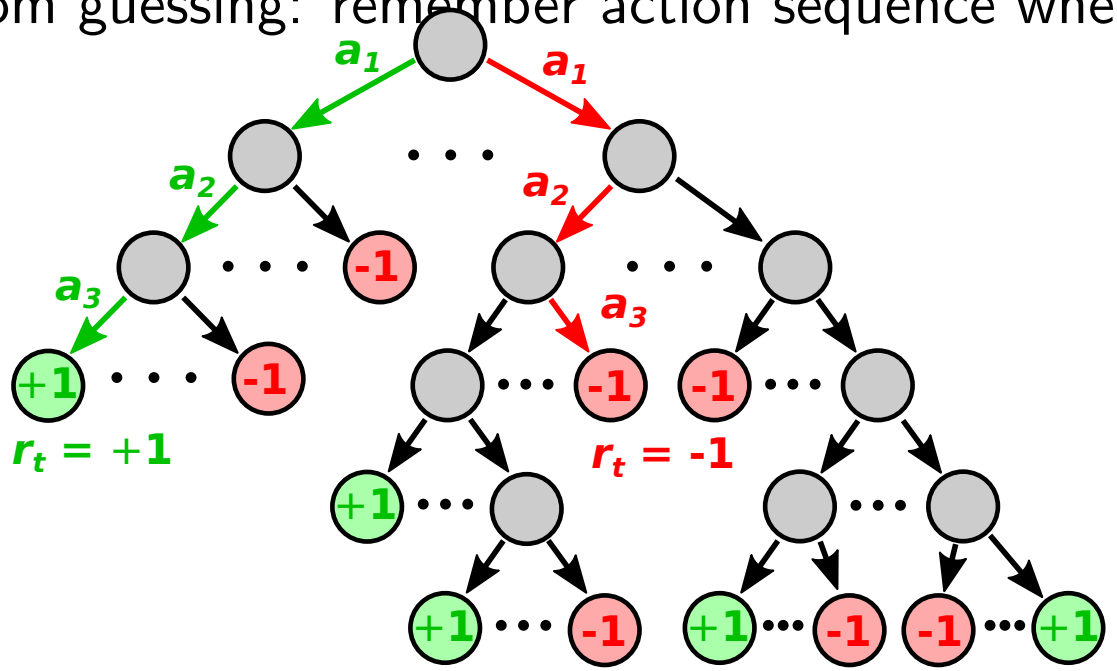
- Find a sequence of actions $a_1, \ldots, a_n$ that leads to a win
  - standard search problem for single player games
  - many possible solutions: easy problem



Decision trees (1)

Recall last lecture

- Find a sequence of actions $a_1, \ldots, a_n$ that leads to a win
  - standard search problem for single player games
  - many possible solutions: easy problem
- Random guessing: remember action sequence when $r = +1$



Decision trees (1)

Recall last lecture

- Find a sequence of actions $a_1, \ldots, a_n$ that leads to a win
  - standard search problem for single player games
  - many possible solutions: easy problem
- Random guessing: remember action sequence when $r = +1$

```
1 def random_guessing(initial_state):
2     while True:
3         state, actions = initial_state, []
4         while not state.terminal():
5             actions.append(random.choice(state.actions()))
6             state = state.transition(actions[-1])
7         if state.reward() == +1:
8             return actions
```

Decision trees (1)

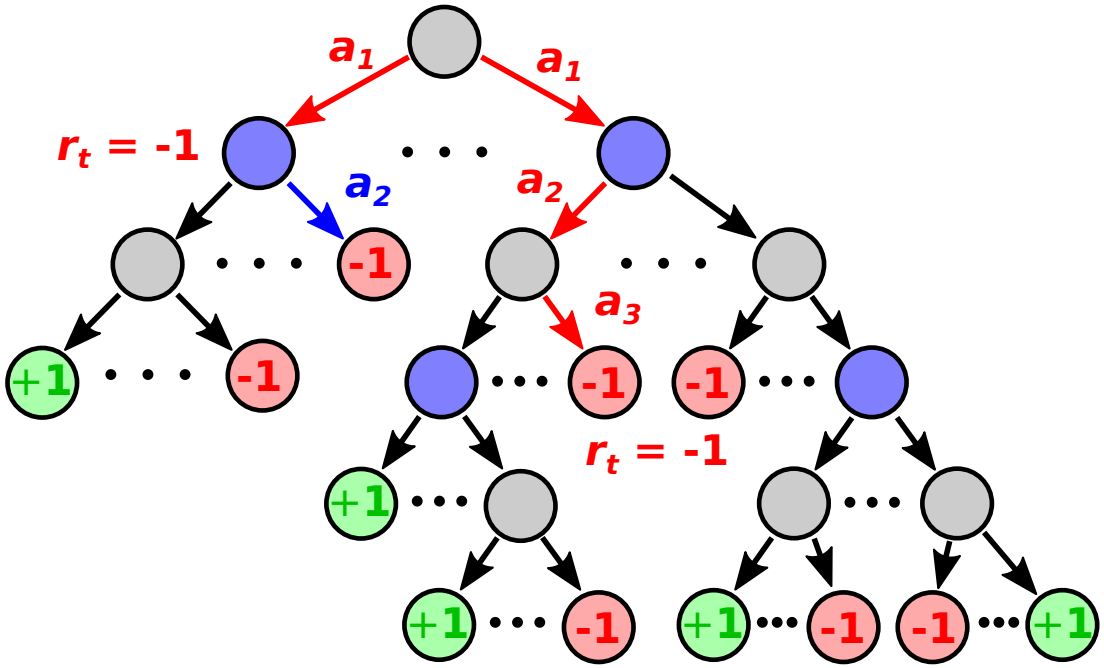
Recall last lecture

- Find a sequence of actions $a_1, \ldots, a_n$ that leads to a win
  - standard search problem for single player games
  - many possible solutions: easy problem
- Random guessing: remember action sequence when $r = +1$
  - random execution time

Decision trees (2)

Recall last lecture

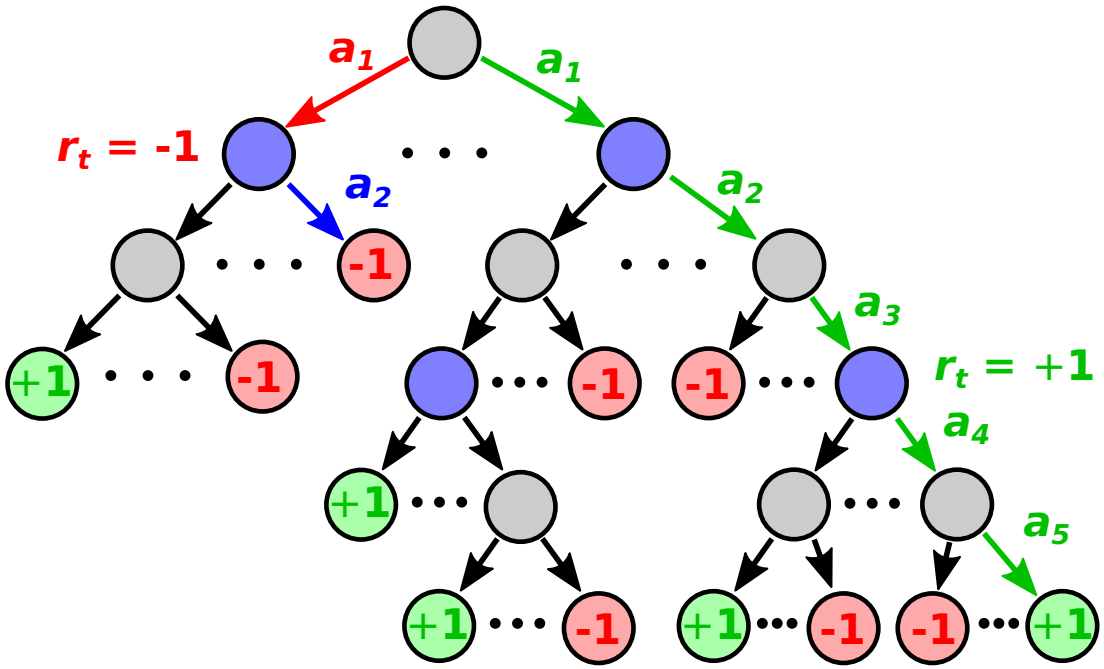
- But the opponent chooses some of the actions!
  - opponent can easily prevent random solutions



Decision trees (2)

Recall last lecture

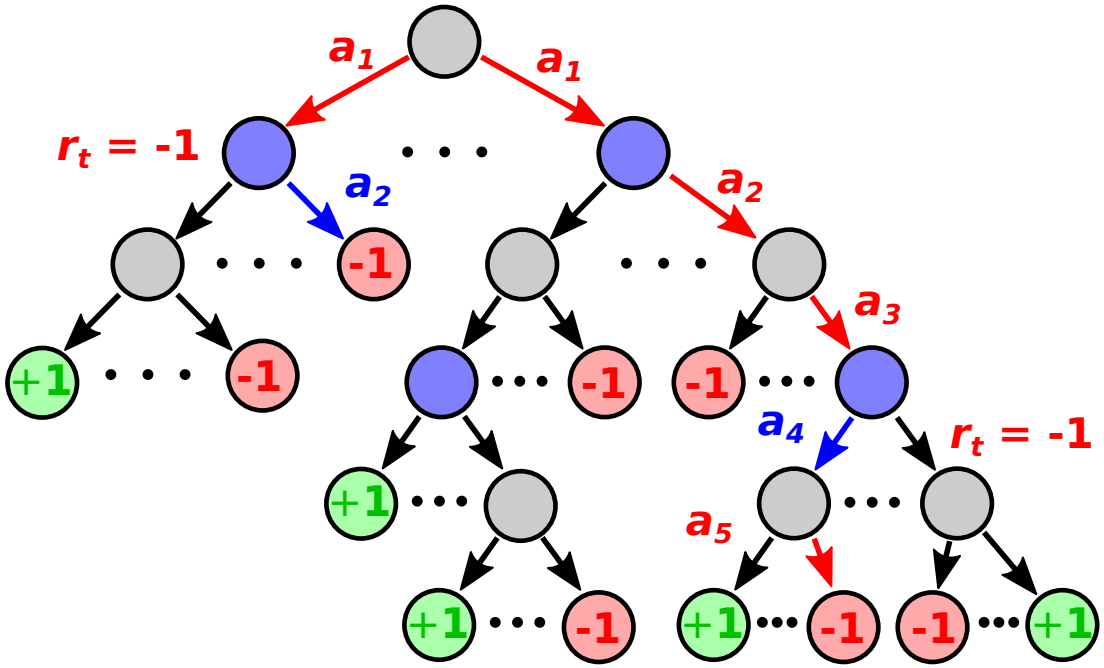
- But the opponent chooses some of the actions!
  - opponent can easily prevent random solutions
  - in fact, all winning action sequence in this tree ...



Decision trees (2)

Recall last lecture

- But the opponent chooses some of the actions!
  - opponent can easily prevent random solutions
  - in fact, all winning action sequence in this tree ...
  - ... that can be easily disturbed



Two-player zero-sum games

- Problem very complex for *general-sum* games
  - sometimes cooperation, sometimes competition
  - very complex Nash equilibria (☞ Game Theory)

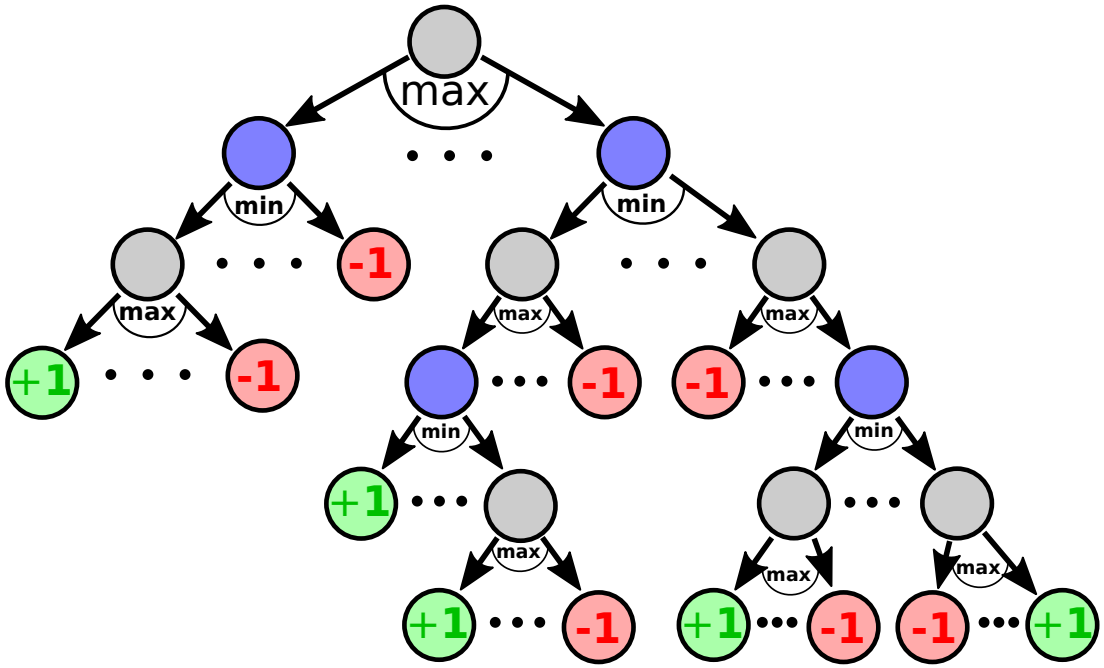
- Zero-sum: **"one players' gain is the other's loss"**
  - every reward r for one player is a punishment $-r$ for the other
  - two-player zero sum game can be expressed by *one* function
  - player `1` *maximizes* function, player `-1` *minimizes* function

Image sources: `wikimedia.org`

Minmax value function

Formalised in MDP framework

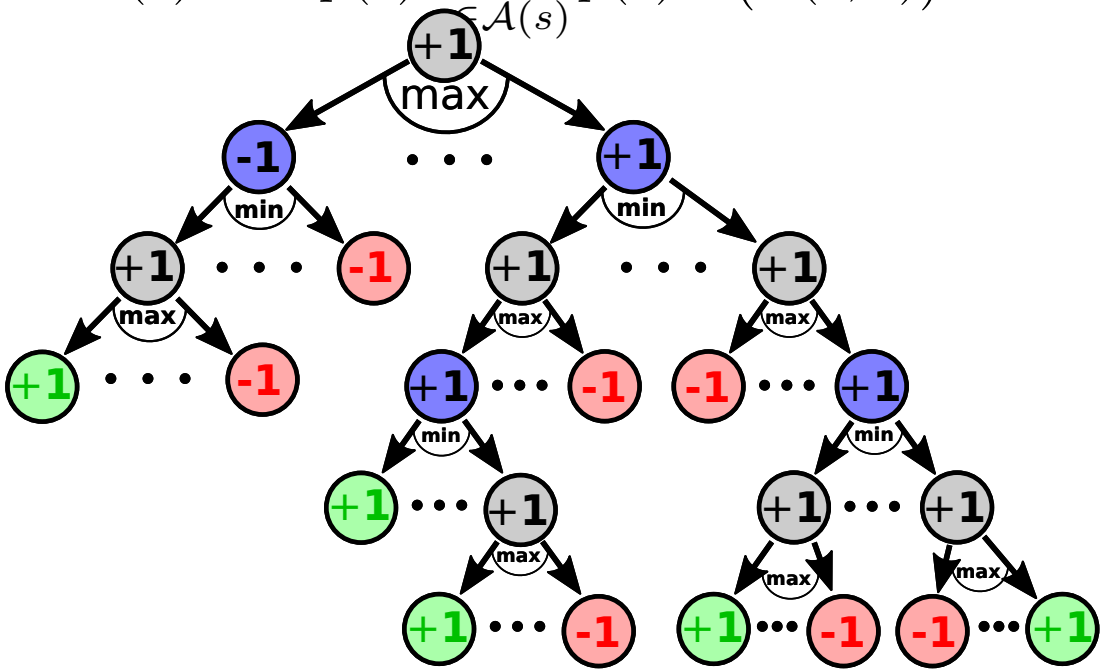
- The value function $V(s)$ of a state $s \in \mathcal{S}$
  - maximizes return for $p(s) = 1$ (grey nodes)
  - minimizes return for $p(s) = -1$ (blue nodes)



Minmax value function

Formalised in MDP framework

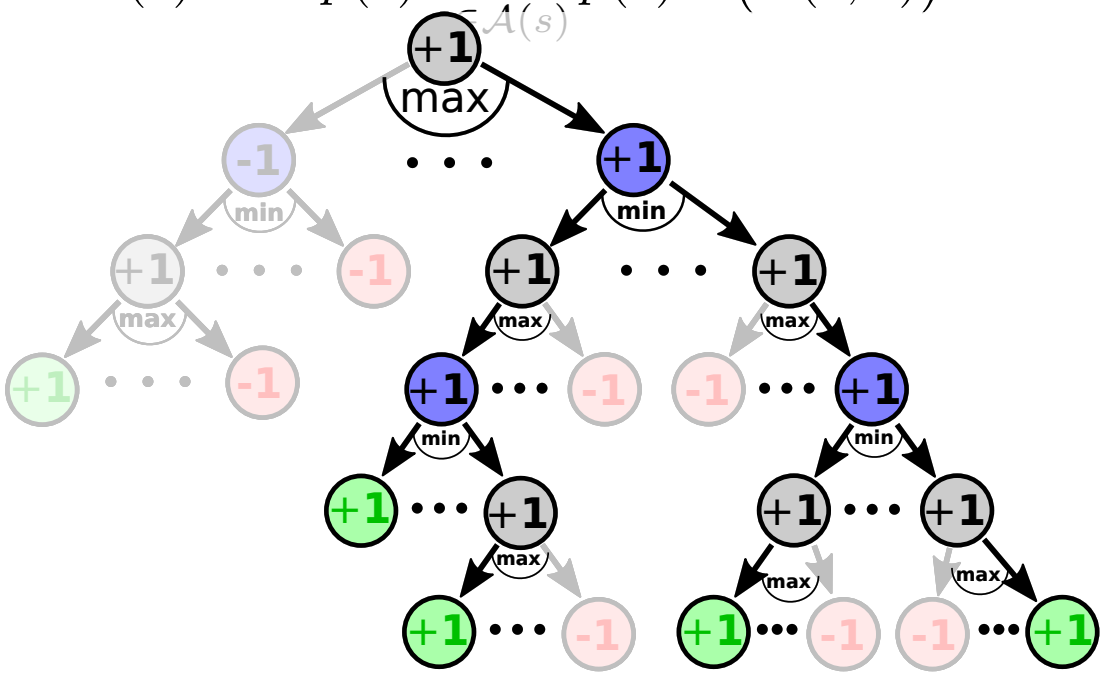
- The value function $V(s)$ of a state $s \in \mathcal{S}$
  - maximizes return for $p(s) = 1$ (grey nodes)
  - minimizes return for $p(s) = -1$ (blue nodes)
  - if $\mathcal{T}(s)$ then $V(s) := \mathcal{R}(s)$
  - else $V(s) := p(s) \max_{a \in \mathcal{A}(s)} p(s)\, V\big(\mathcal{P}(s,a)\big)$



Minmax value function

Formalised in MDP framework

- The value function $V(s)$ of a state $s \in \mathcal{S}$
  - maximizes return for $p(s) = 1$ (grey nodes)
  - minimizes return for $p(s) = -1$ (blue nodes)
  - if $\mathcal{T}(s)$ then $V(s) := \mathcal{R}(s)$
  - else $V(s) := p(s) \max_{a \in \mathcal{A}(s)} p(s)\, V\big(\mathcal{P}(s,a)\big)$



Decision tree implementation

```python
class Node:
    def __init__(self, state):
        self.state = state                      # s ∈ S
        self.children = {}                      # {a : P(s, a), ...}
        self.total_value = 0                    # Ṽ(s) = 1/N(s) Σ_{t=1}^{N(s)} r_t
        self.num_visits = 0                     # N(s)

class DecisionTree:
    def __init__(self, initial_state):
        self.root = Node(initial_state) # root of the tree

    def node_value(self, node, parent=None, **kwargs):
        value = node.total_value / node.num_visits
        if parent is not None:
            value *= parent.state.player
        return value                            # p(parent.s) Ṽ(node.s)
```

Interactive block II: zero-sum games

Multiple (or no) answers may be correct!

Let the **Minmax-value** for this question be defined like this:

$$V(s) = \begin{cases} \alpha\,\mathcal{R}(s) - \beta & \text{, if } \mathcal{T}(s) \\ \gamma\, p(s) \max\limits_{a \in \mathcal{A}(s)} p(s)\, V\big(\mathcal{P}(s,a)\big) - \delta & \text{, if } \neg\mathcal{T}(s) \end{cases}$$

- Which combination(s) will return the **shortest winning path**?

A) $\alpha = 1,\ \beta = 0,\ \gamma = 1,\ \delta = 0$
B) $\alpha = \frac{1}{2},\ \beta = 0,\ \gamma = 1,\ \delta = 0$
C) $\alpha = 1,\ \beta = \frac{1}{2},\ \gamma = 1,\ \delta = 0$
D) $\alpha = \frac{1}{2},\ \beta = \frac{1}{2},\ \gamma = 1,\ \delta = 0$
E) $\alpha = 1,\ \beta = 0,\ \gamma = \frac{1}{2},\ \delta = 0$
F) $\alpha = \frac{1}{2},\ \beta = 0,\ \gamma = \frac{1}{2},\ \delta = 0$
G) $\alpha = 1,\ \beta = \frac{1}{2},\ \gamma = \frac{1}{2},\ \delta = 0$
H) $\alpha = \frac{1}{2},\ \beta = \frac{1}{2},\ \gamma = \frac{1}{2},\ \delta = 0$
I) $\alpha = 1,\ \beta = 0,\ \gamma = 1,\ \delta = \frac{1}{2}$
J) $\alpha = \frac{1}{2},\ \beta = 0,\ \gamma = 1,\ \delta = \frac{1}{2}$
K) $\alpha = 1,\ \beta = \frac{1}{2},\ \gamma = 1,\ \delta = \frac{1}{2}$
L) $\alpha = \frac{1}{2},\ \beta = \frac{1}{2},\ \gamma = 1,\ \delta = \frac{1}{2}$
M) $\alpha = 1,\ \beta = 0,\ \gamma = \frac{1}{2},\ \delta = \frac{1}{2}$
N) $\alpha = \frac{1}{2},\ \beta = 0,\ \gamma = \frac{1}{2},\ \delta = \frac{1}{2}$
O) $\alpha = 1,\ \beta = \frac{1}{2},\ \gamma = \frac{1}{2},\ \delta = \frac{1}{2}$
P) $\alpha = \frac{1}{2},\ \beta = \frac{1}{2},\ \gamma = \frac{1}{2},\ \delta = \frac{1}{2}$

Interactive block II: zero-sum games

Multiple (or no) answers may be correct!

Let the **Minmax-value** for this question be defined like this:

$$V(s) = \begin{cases} \alpha\,\mathcal{R}(s) - \beta & \text{, if } \mathcal{T}(s) \\ \gamma\,p(s) \max\limits_{a \in \mathcal{A}(s)} p(s)\,V\big(\mathcal{P}(s,a)\big) - \delta & \text{, if } \neg\mathcal{T}(s) \end{cases}$$

- Which combination(s) will return the **shortest winning path**?

A) $\alpha = 1,\ \beta = 0,\ \gamma = 1,\ \delta = 0$

B) $\alpha = \frac{1}{2},\ \beta = 0,\ \gamma = 1,\ \delta = 0$

C) $\alpha = 1,\ \beta = \frac{1}{2},\ \gamma = 1,\ \delta = 0$

D) $\alpha = \frac{1}{2},\ \beta = \frac{1}{2},\ \gamma = 1,\ \delta = 0$

E) $\alpha = 1,\ \beta = 0,\ \gamma = \frac{1}{2},\ \delta = 0$ ✓

F) $\alpha = \frac{1}{2},\ \beta = 0,\ \gamma = \frac{1}{2},\ \delta = 0$ ✓

G) $\alpha = 1,\ \beta = \frac{1}{2},\ \gamma = \frac{1}{2},\ \delta = 0$ ✓

H) $\alpha = \frac{1}{2},\ \beta = \frac{1}{2},\ \gamma = \frac{1}{2},\ \delta = 0$

I) $\alpha = 1,\ \beta = 0,\ \gamma = 1,\ \delta = \frac{1}{2}$

J) $\alpha = \frac{1}{2},\ \beta = 0,\ \gamma = 1,\ \delta = \frac{1}{2}$

K) $\alpha = 1,\ \beta = \frac{1}{2},\ \gamma = 1,\ \delta = \frac{1}{2}$

L) $\alpha = \frac{1}{2},\ \beta = \frac{1}{2},\ \gamma = 1,\ \delta = \frac{1}{2}$

M) $\alpha = 1,\ \beta = 0,\ \gamma = \frac{1}{2},\ \delta = \frac{1}{2}$

N) $\alpha = \frac{1}{2},\ \beta = 0,\ \gamma = \frac{1}{2},\ \delta = \frac{1}{2}$

O) $\alpha = 1,\ \beta = \frac{1}{2},\ \gamma = \frac{1}{2},\ \delta = \frac{1}{2}$

P) $\alpha = \frac{1}{2},\ \beta = \frac{1}{2},\ \gamma = \frac{1}{2},\ \delta = \frac{1}{2}$

15 minute(s) break!

Playing a game

Back to the lab ☺

- We want to play a game between two solvers like this:

```
 1 initial_state = State()                          # the game played
 2 us = GameSolver(initial_state)                   # player's solver
 3 them = AnotherGameSolver(initial_state)          # opponent's solver
 4 while not us.terminal():                         # until game finished
 5     us.solve(), them.solve()                     # both players think
 6     action = us.choose() if us.player() == 1 \
 7              else them.choose()                  # one player chooses
 8     us.play(action), them.play(action)           # both players act
 9 out = us.outcome()                               # who won the game
10 print("won" if out > 1 else "lost" if out < -1 else "draw")
```

Game interface (1)

- Our `GameSolver`'s need to follow the `Game` interface

```
 1 class Game:
 2     def reset(self):
 3         pass              # reverts game back to initial state
 4
 5     def play(self, action):
 6         pass              # changes state of the current game
 7
 8     def choose(self):
 9         return None       # ∈ 𝒜(s); chooses an action
10
11     def terminal(self):
12         return True       # ∈ {⊤, ⊥}; whether game is finished
13
14     def outcome(self):
15         return 0          # ∈ ℝ; a finished game's outcome
16
17     def player(self):
18         return 1          # ∈ {−1, 1}; which player's move it is
```

Game implementation (1)

- The following `DecisionTree` can be played as a `Game`

```
 1 class GameTree (DecisionTree, Game):
 2     def __init__(self, initial_state):
 3         super().__init__(initial_state)
 4         self.initial_state = initial_state
 5
 6     def reset(self):
 7         self.root = Node(self.initial_state)
 8
 9     def play(self, action):
10         self.root = self.get_child(self.root, action)
11
12     def get_child(self, node, action):
13         child = node.children.get(action)
14         if child is None:
15             child = Node(node.state.transition(action))
16         return child
```

Game implementation (2)

```
 1     def choose(self):
 2         selected = self.best_child(self.root)
 3         for action, child in self.root.children.items():
 4             if child is selected:
 5                 return action
 6
 7     def best_child(self, node, **kwargs):
 8         best_nodes, best_value = [], float("-inf")
 9         for child in node.children.values():
10             value = self.node_value(child, parent=node, **kwargs)
11             if value == best_value:
12                 best_nodes.append(child)
13             if value > best_value:
14                 best_nodes, best_value = [child], value
15         return random.choice(best_nodes)
16
17     def terminal(self):
18         return self.root.state.terminal()
19
20     def outcome(self):
21         return self.root.state.reward()
```

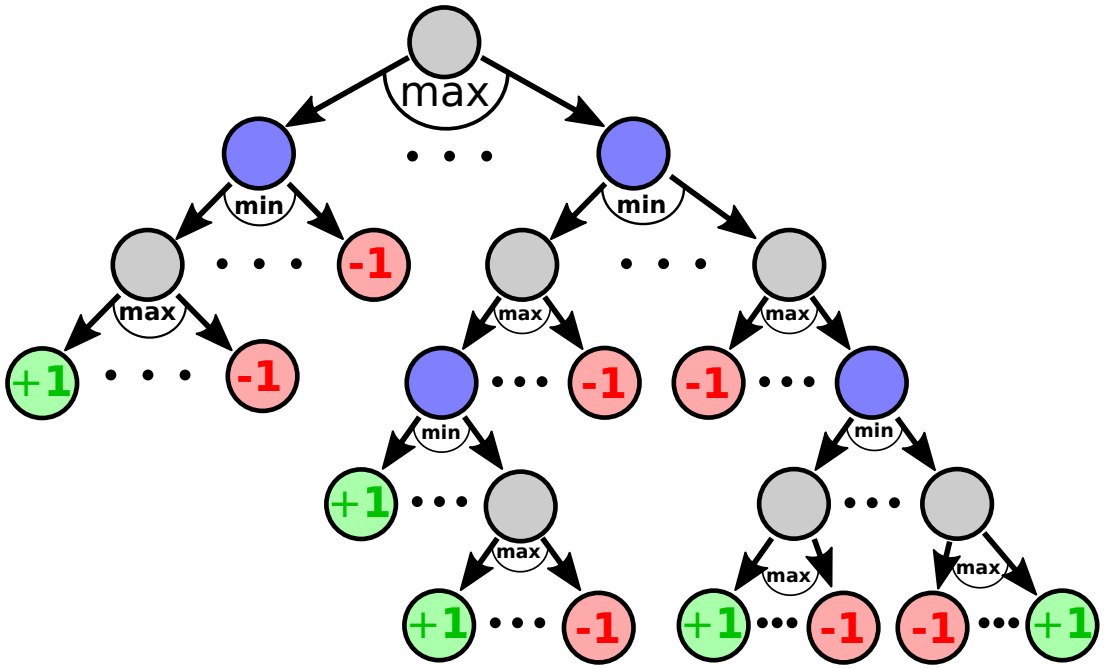
Game solvers

- A solver populates the `DecisionTree` by calling `solve()`

```
1  class Solver:
2      def __init__(self, iteration_limit, **kwargs):
3          self.iteration_limit = iteration_limit
4          self.iteration_counter = 0
5
6      def solve(self):                # populate the decision tree
7          while self.within_budget():
8              self.search()
9
10     def within_budget(self):        # checks the available budget
11         self.iteration_counter += 1
12         done = self.iteration_counter > self.iteration_limit
13         if done:
14             self.iteration_counter = 0
15         return not done
16
17     def search(self):               # solvers override this
18         raise Exception("Not implemented yet.")
```

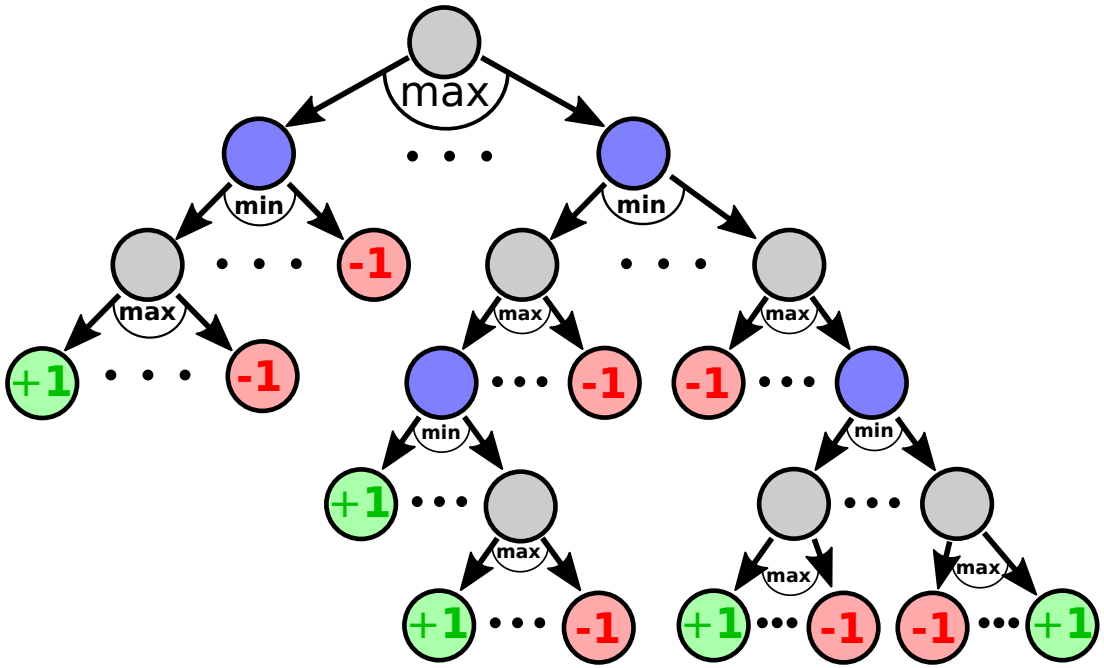
Random search

- Computing $V(s)$ recursively can take too long



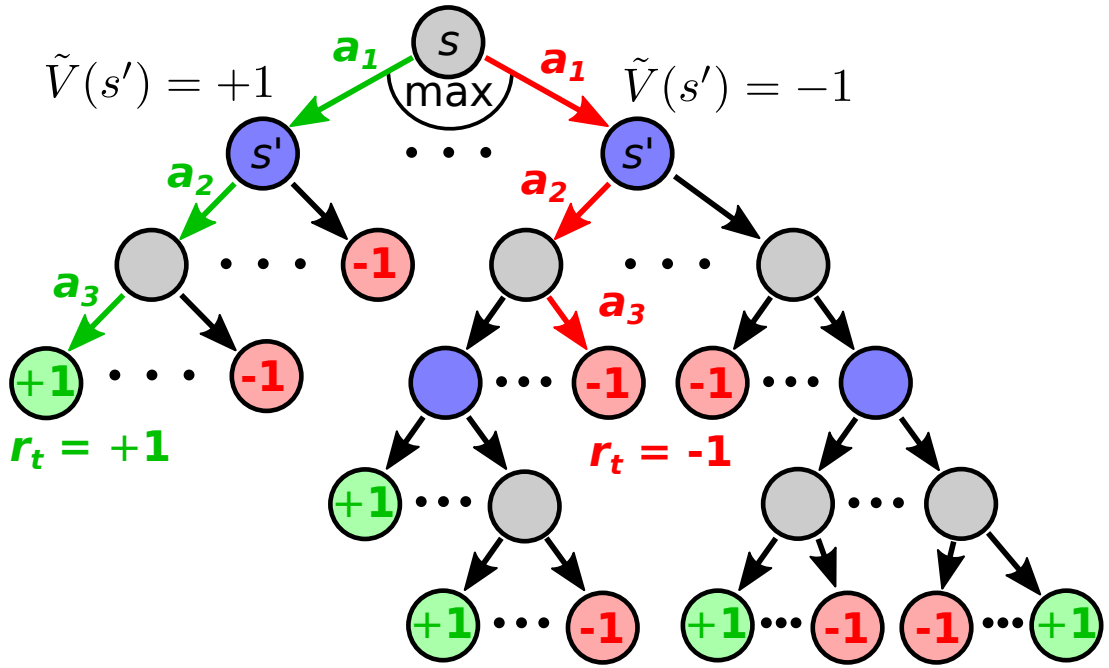
Random search

- Computing $V(s)$ recursively can take too long
- Minimax search (plus pruning, etc) is deterministic



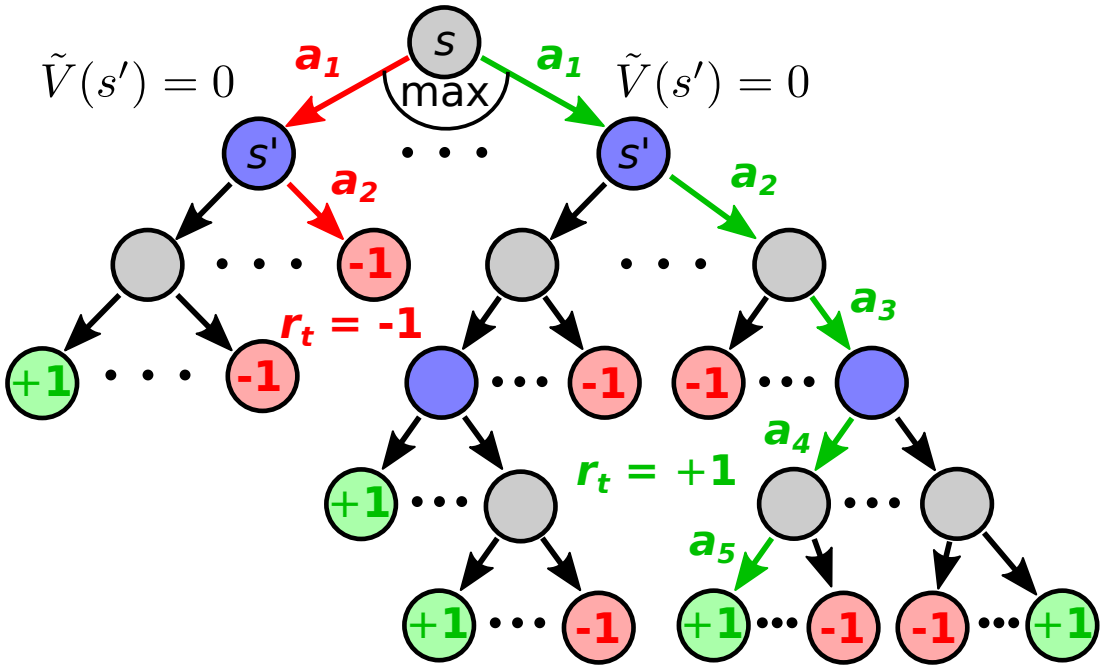
Random search

- Computing $V(s)$ recursively can take too long
- Idea: Approximate $V(\mathcal{P}(s,a)), \forall a \in \mathcal{A}(s)$, with *random* rollouts
  - keep track of $\tilde{V}_{\mathsf{R}}(s') = \frac{1}{N(s')} \sum_{t=1}^{N(s')} r_t, \forall s' \in \{\mathcal{P}(s,a) | a \in \mathcal{A}(s)\}$



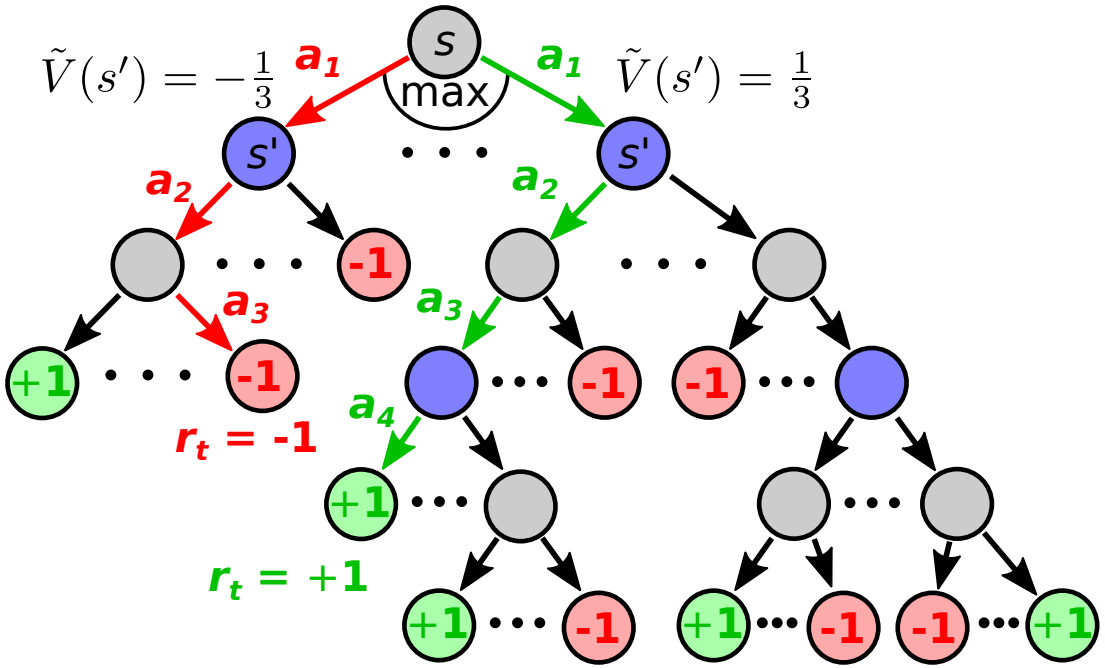
Random search

- Computing $V(s)$ recursively can take too long
- Idea: Approximate $V(\mathcal{P}(s,a)), \forall a \in \mathcal{A}(s)$, with *random* rollouts
  - keep track of $\tilde{V}_{\mathsf{R}}(s') = \frac{1}{N(s')} \sum_{t=1}^{N(s')} r_t, \forall s' \in \{\mathcal{P}(s,a) | a \in \mathcal{A}(s)\}$



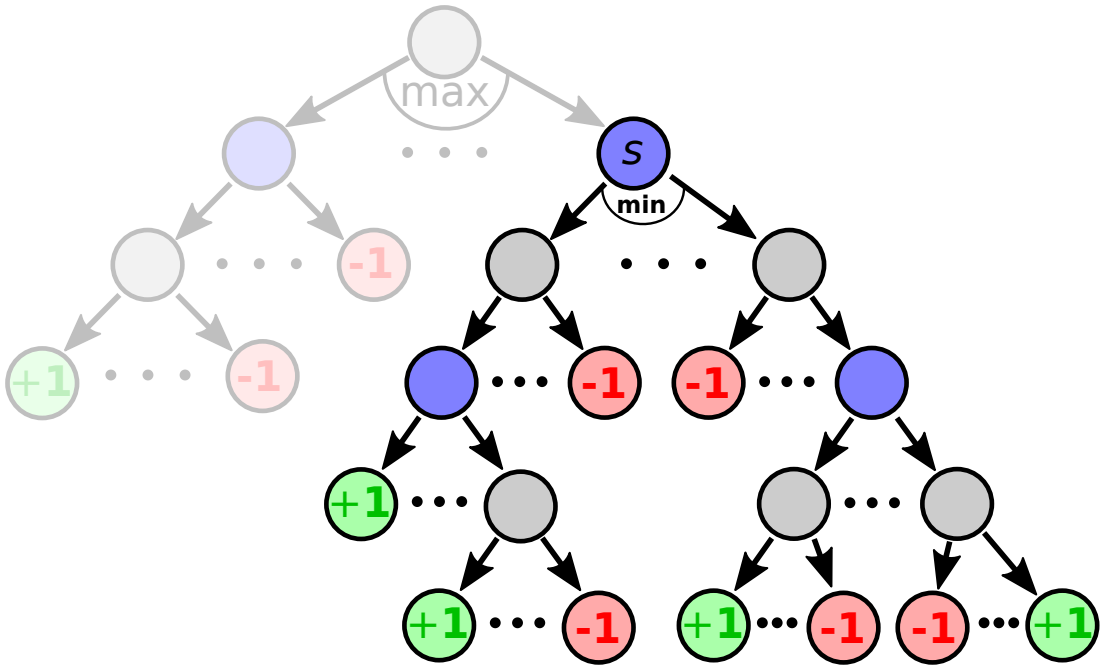
Random search

- Computing $V(s)$ recursively can take too long
- Idea: Approximate $V(\mathcal{P}(s,a)), \forall a \in \mathcal{A}(s)$, with *random* rollouts
  - keep track of $\tilde{V}_{\mathsf{R}}(s') = \frac{1}{N(s')} \sum_{t=1}^{N(s')} r_t, \forall s' \in \{\mathcal{P}(s,a) | a \in \mathcal{A}(s)\}$
  - good for shallow sub-trees with similar outcomes



Random search

- Computing $V(s)$ recursively can take too long
- Idea: Approximate $V(\mathcal{P}(s,a)), \forall a \in \mathcal{A}(s)$, with *random* rollouts
  - keep track of $\tilde{V}_{\mathsf{R}}(s') = \frac{1}{N(s')} \sum_{t=1}^{N(s')} r_t, \forall s' \in \{\mathcal{P}(s,a) | a \in \mathcal{A}(s)\}$
  - good for shallow sub-trees with similar outcomes
  - minmax selection: $a_t := \underset{a \in \mathcal{A}(s_t)}{\arg\max}\ p(s_t)\, \tilde{V}_{\mathsf{R}}\big(\mathcal{P}(s_t,a)\big)$



Random search

- Computing $V(s)$ recursively can take too long
- Idea: Approximate $V(\mathcal{P}(s,a)), \forall a \in \mathcal{A}(s)$, with *random* rollouts
  - keep track of $\tilde{V}_{\mathsf{R}}(s') = \frac{1}{N(s')} \sum_{t=1}^{N(s')} r_t, \forall s' \in \{\mathcal{P}(s,a) | a \in \mathcal{A}(s)\}$
  - good for shallow sub-trees with similar outcomes
  - minmax selection: $a_t := \underset{a \in \mathcal{A}(s_t)}{\arg\max}\ p(s_t)\, \tilde{V}_{\mathsf{R}}\big(\mathcal{P}(s_t,a)\big)$

- Often works in practice, but is *not* Minmax Value:

$$\lim_{N(s)\to\infty} \tilde{V}_{\mathsf{R}}(s) \quad = \quad \begin{cases} \mathcal{R}(s) & \text{, if } \mathcal{T}(s) \\ \frac{1}{\mathcal{A}(s)} \sum\limits_{a \in \mathcal{A}(s)} \tilde{V}_{\mathsf{R}}\big(\mathcal{P}(s,a)\big) & \text{, otherwise} \end{cases}$$

Random search implementation

```python
class RandomSearch(GameTree, Solver):
    def search(self):                          # rollout random action
        all_actions = self.root.state.actions()
        action = random.choice(all_actions)
        child = self.add_child(self.root, action)
        child.total_value += self.rollout(child)
        child.num_visits += 1

    def add_child(self, node, action):  # potentially new node
        child = self.get_child(node, action)
        node.children[action] = child
        return child

    def rollout(self, node):                   # evaluate random policy
        state = node.state
        while not state.terminal():
            action = random.choice(state.actions())
            state = state.transition(action)
        return state.reward()
```

Interactive block III: random search

Multiple (or no) answers may be correct!

How can we exploit the tree-structure better?

- By building a decision tree using...
  - A) random rollouts from the root
  - B) random rollouts from the root's children
  - C) minmax rollouts within the tree
  - D) minmax rollouts at the tree's leafs
  - E) random rollouts at the tree's leafs
- The following nodes should be updated after a rollout:
  - F) only the root
  - G) only the root's children
  - H) all nodes along the chosen path
  - I) all nodes along the current best path
  - J) all nodes along the chosen path that differ from the current best path

Interactive block III: random search

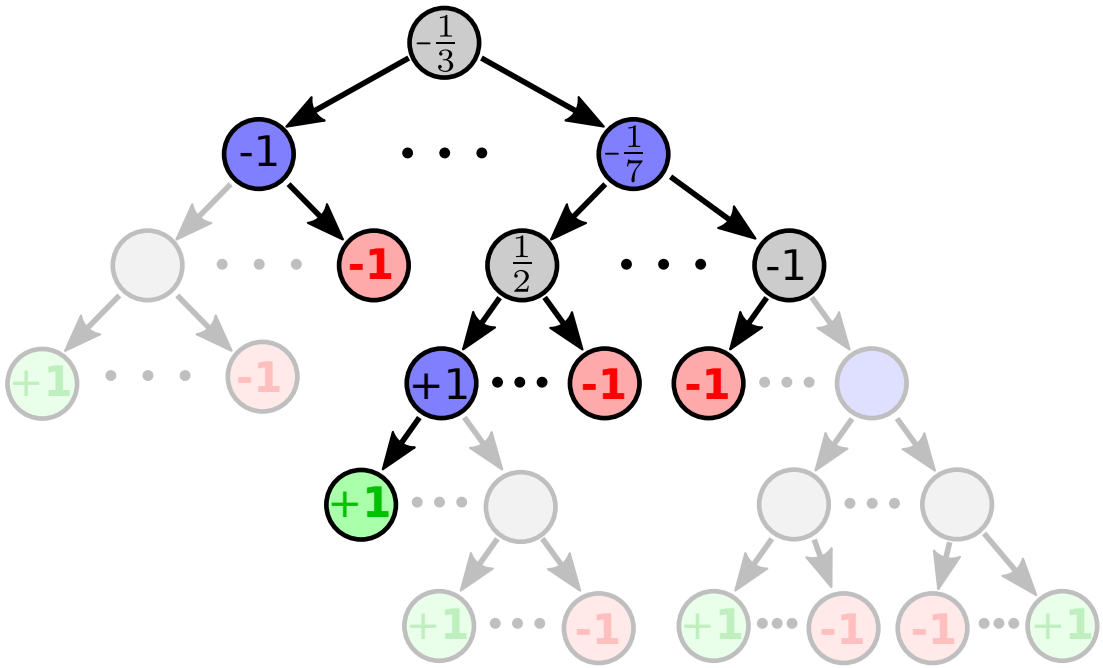
Multiple (or no) answers may be correct!

How can we exploit the tree-structure better?

- By building a decision tree using...
  - A) random rollouts from the root
  - B) random rollouts from the root's children
  - C) minmax rollouts within the tree √
  - D) minmax rollouts at the tree's leafs
  - E) random rollouts at the tree's leafs √
- The following nodes should be updated after a rollout:
  - F) only the root
  - G) only the root's children
  - H) all nodes along the chosen path √
  - I) all nodes along the current best path
  - J) all nodes along the chosen path that differ from the current best path

Monte Carlo Tree Search (MCTS)

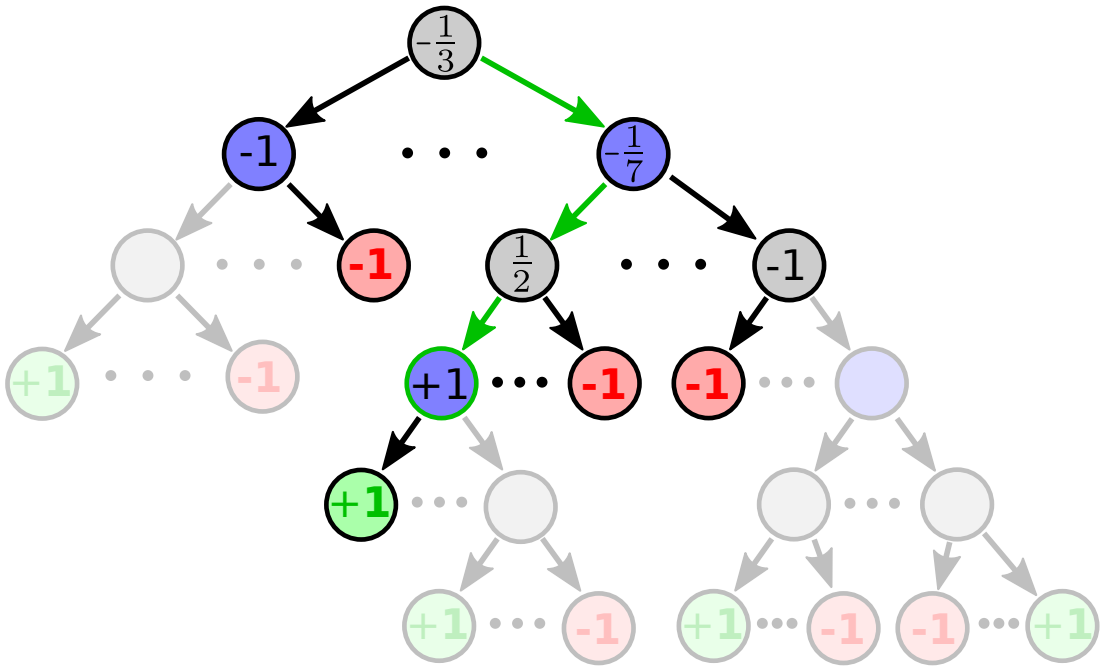
- Idea: build partial tree $\mathcal{B} \subseteq \mathcal{S}$



for details on MCTS see Browne et al. (2012)

Monte Carlo Tree Search (MCTS)

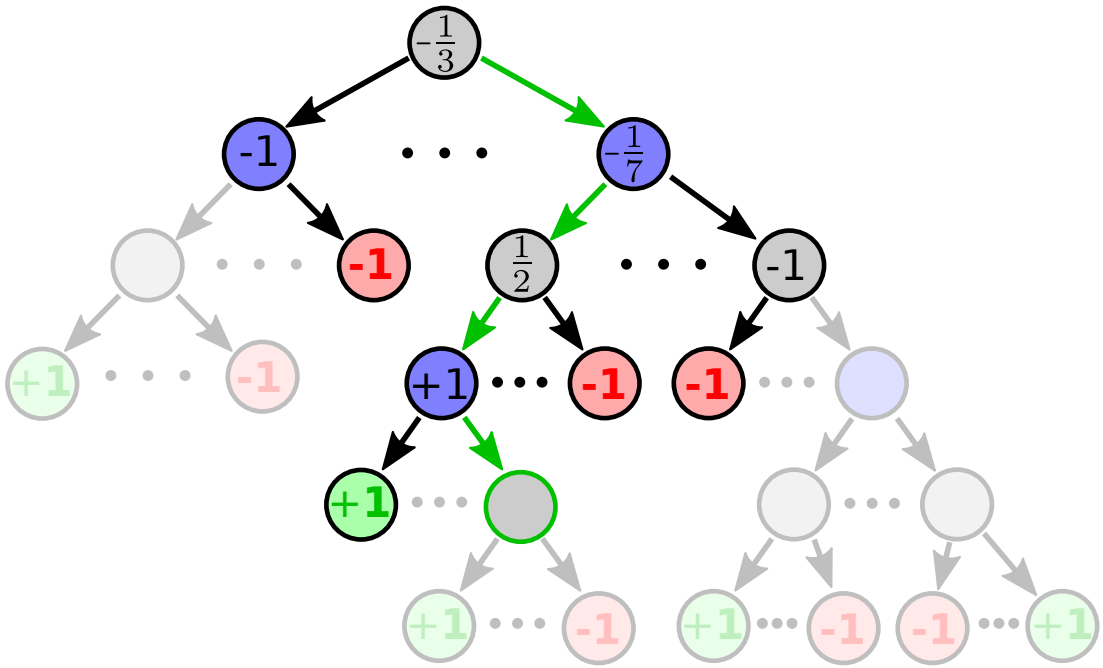
- Idea: build partial tree $\mathcal{B} \subseteq \mathcal{S}$
  - use minmax-selection in partial tree



for details on MCTS see Browne et al. (2012)

Monte Carlo Tree Search (MCTS)

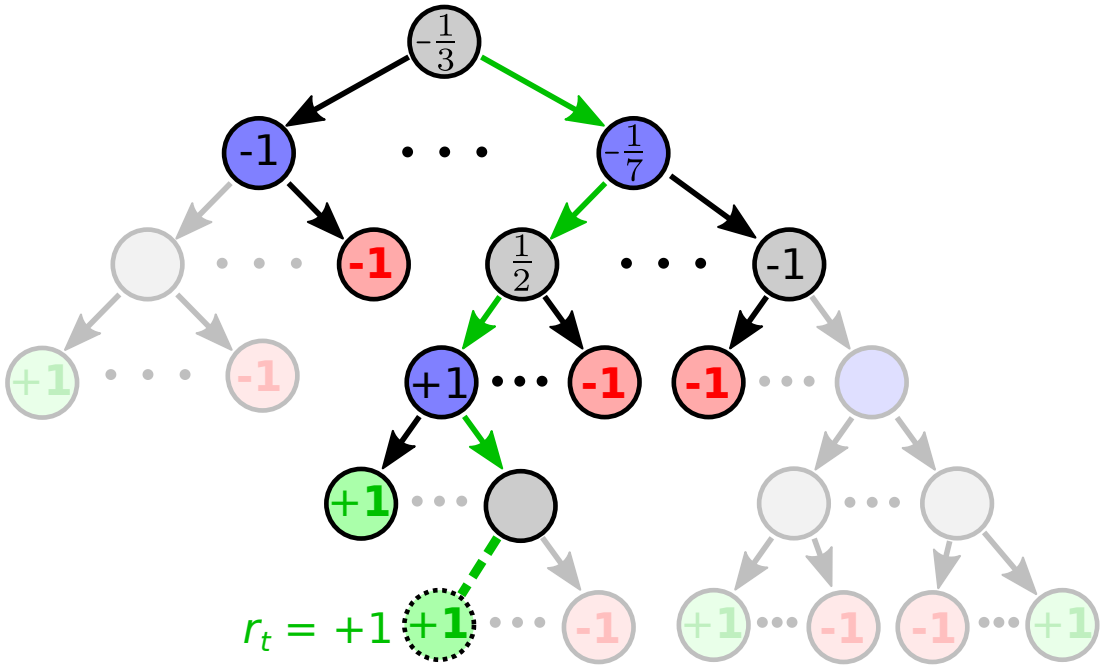
- Idea: build partial tree $\mathcal{B} \subseteq \mathcal{S}$
  - use minmax-selection in partial tree
  - expand tree with new leaf



for details on MCTS see Browne et al. (2012)

Monte Carlo Tree Search (MCTS)

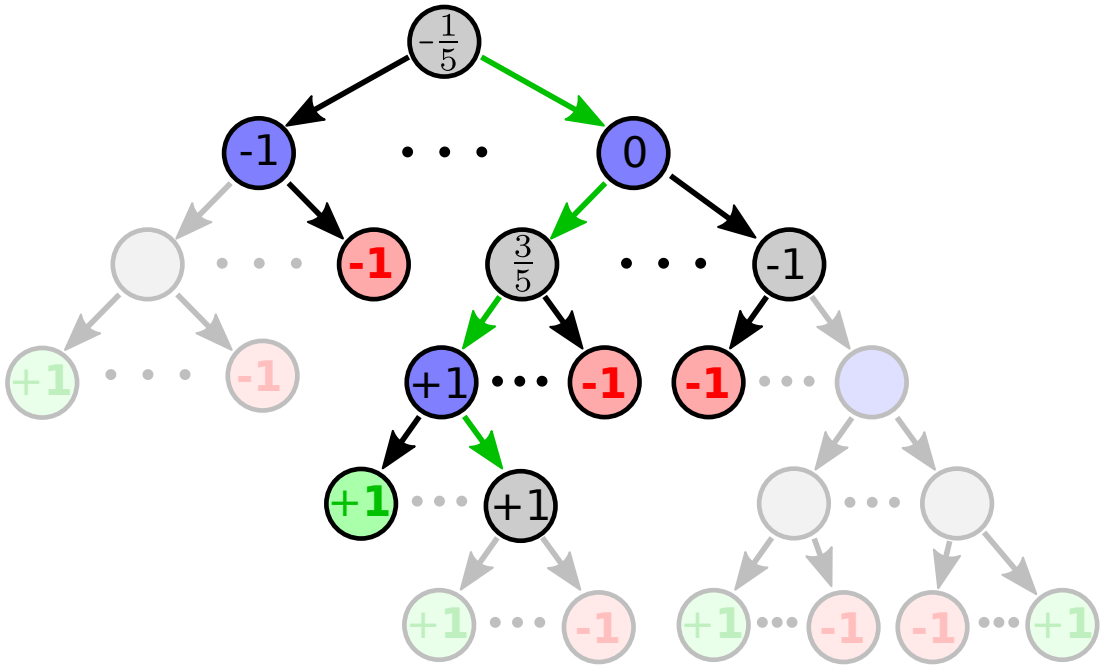
- Idea: build partial tree $\mathcal{B} \subseteq \mathcal{S}$
  - use minmax-selection in partial tree
  - expand tree with new leaf
  - approximate leaf values with random rollouts



for details on MCTS see Browne et al. (2012)

Monte Carlo Tree Search (MCTS)

- Idea: build partial tree $\mathcal{B} \subseteq \mathcal{S}$
  - use minmax-selection in partial tree
  - expand tree with new leaf
  - approximate leaf values with random rollouts
  - backup the resulting reward to all parents



for details on MCTS see Browne et al. (2012)

Monte Carlo Tree Search (MCTS)

- Idea: build partial tree $\mathcal{B} \subseteq \mathcal{S}$
  - use minmax-selection in partial tree
  - expand tree with new leaf
  - approximate leaf values with random rollouts
  - backup the resulting reward to all parents

- $\mu(a|s)$ fraction of action a chosen in state s
  - mostly minmax, but sensitive to initial rewards

$$\tilde{V}_{\mathsf{M}}(s) = \begin{cases} \mathcal{R}(s) & \text{, if } \mathcal{T}(s) \\ \sum_{a \in \mathcal{A}(s)} \mu(a|s)\, \tilde{V}_{\mathsf{M}}\big(\mathcal{P}(s,a)\big) & \text{, if } s \in \mathcal{B} \\ r_t & \text{, otherwise} \end{cases}$$

for details on MCTS see Browne et al. (2012)

MCTS implementation (1)

- uses a stack (`list`) to keep track of visited nodes
  1. `select` minmax policy
  2. `expand` adds a new leaf
  3. `rollout` evaluates leaf
  4. `backup` updates values

```
 1 class MCTS (RandomSearch):
 2     def __init__(self, initial_state, iteration_limit, **kwargs):
 3         super().__init__(initial_state, iteration_limit, **kwargs)
 4         self.stack = []
 5
 6     def init_stack(self, node):          # initialize stack
 7         self.stack = [node]
 8
 9     def search(self):
10         leaf = self.select(self.root)    # push path on stack
11         if not leaf.state.terminal():    # unless this is a leaf
12             leaf = self.expand(leaf)     # create new leaf
13         reward = self.rollout(leaf)      # evaluate leaf
14         self.backup(reward)              # pop path from stack
```

MCTS implementation (2)

```
 1     def select(self, node):                  # selects leaf node
 2         self.init_stack(node)
 3         while len(node.children) == len(node.state.actions()) \
 4                 and not node.state.terminal():
 5             node = self.best_child(node)
 6             self.stack.append(node)
 7         return node
 8
 9     def expand(self, node):                  # adds unexplored leaf
10         for action in node.state.actions():
11             if action not in node.children:
12                 leaf = self.add_child(node, action)
13                 self.stack.append(leaf)
14                 return leaf
15
16     def backup(self, reward):
17         while len(self.stack) > 0:
18             node = self.stack.pop()
19             node.total_value += reward
20             node.num_visits += 1
```

```
1 def search(self):
2   leaf = self.select(self.root)
3   if not leaf.state.terminal():
4     leaf = self.expand(leaf)
5   reward = self.rollout(leaf)
6   self.backup(reward)
```

MCTS characteristics

- In MCTS μ depends on initial rewards
  - there are ways to do proper *exploration*

$$\tilde{V}_{\mathsf{M}}(s) = \begin{cases} \mathcal{R}(s) & \text{, if } \mathcal{T}(s) \\ \sum_{a \in \mathcal{A}(s)} \mu(a|s)\, \tilde{V}_{\mathsf{M}}\big(\mathcal{P}(s,a)\big) & \text{, if } s \in \mathcal{B} \\ r_t & \text{, otherwise} \end{cases}$$

- Important characteristics of MCTS
  - aheuristic: no game-heuristics necessary
  - anytime: `within_budget()` can use time limit
  - asymmetric: explores promising sub-trees first

for more details on MCTS see Browne et al. (2012)

MCTS characteristics

Excellent for game playing

- In MCTS μ depends on initial rewards
  - there are ways to do proper *exploration*

$$\tilde{V}_{\mathsf{M}}(s) \quad = \quad \begin{cases} \mathcal{R}(s) & \text{, if } \mathcal{T}(s) \\ \sum\limits_{a \in \mathcal{A}(s)} \mu(a|s)\, \tilde{V}_{\mathsf{M}}\big(\mathcal{P}(s,a)\big) & \text{, if } s \in \mathcal{B} \\ r_t & \text{, otherwise} \end{cases}$$

- Important characteristics of MCTS ☺
  - aheuristic: no game-heuristics necessary
  - anytime: `within_budget()` can use time limit
  - asymmetric: explores promising sub-trees first

for more details on MCTS see Browne et al. (2012)

Example: AlphaGo

Famous in 2016

- Go is unsolvable with search alone

Image source: `wikimedia.org`

Example: AlphaGo

Famous in 2016

- Go is unsolvable with search alone
  - $\mathcal{A}(s) \leq 361$, decision tree with $\approx 10^{768}$ nodes
  - long time horizons between action and payoff
  - more a game of pattern recognition than logic

Image source: `wikimedia.org`

Example: AlphaGo

Famous in 2016

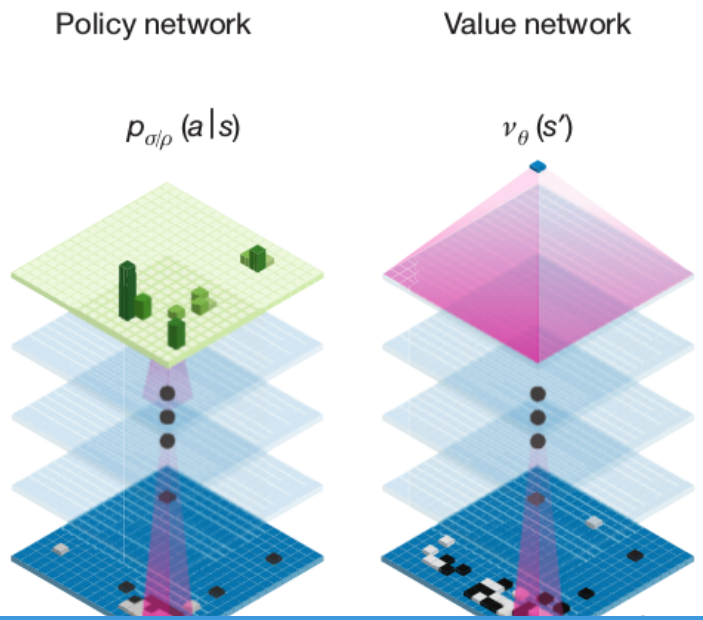
- Go is unsolvable with search alone
- AlphaGo combines search with deep neural networks:

images from Silver et al. (2016)

Example: AlphaGo

Famous in 2016

- Go is unsolvable with search alone
- AlphaGo combines search with deep neural networks:
  - MCTS with bounds (UCT)
  - neural networks as heuristics
  - behaviour cloning from human tournaments
  - further refinement through self-play

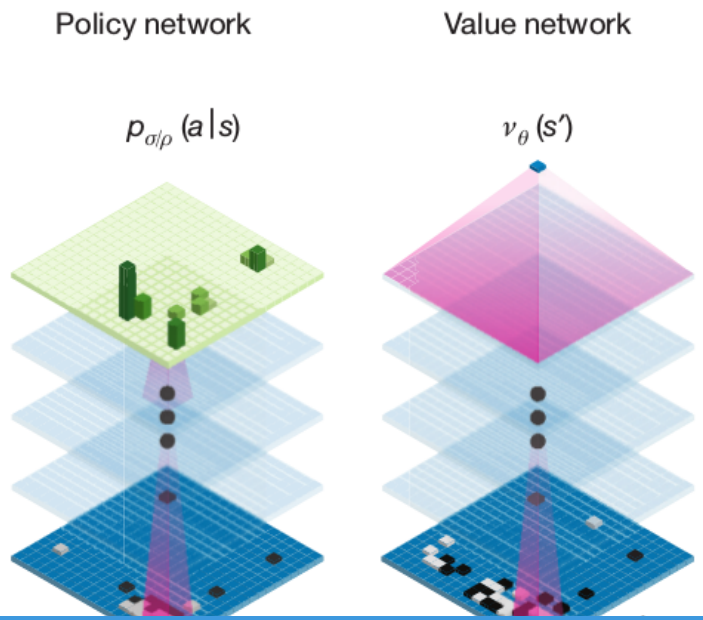


images from Silver et al. (2016)

Example: AlphaGo

Famous in 2016

- Go is unsolvable with search alone
- AlphaGo combines search with deep neural networks:
  - MCTS with bounds (UCT)
  - neural networks as heuristics
  - ~~behaviour cloning from human tournaments~~
  - further refinement through self-play



images from Silver et al. (2016)

Part II: Search and inference

- Today we saw monte carlo tree search

Part II: Search and inference

- Today we saw monte carlo tree search
  - With enhancements and NN evaluation = state of the art for go

Part II: Search and inference

- Today we saw monte carlo tree search
  - With enhancements and NN evaluation = state of the art for go
- Part II: constraints, propagation, search

Part II: Search and inference

- Today we saw monte carlo tree search
  - With enhancements and NN evaluation = state of the art for go
- Part II: constraints, propagation, search
  - state of the art for sudoku

Part II: Search and inference

- Today we saw monte carlo tree search
  - With enhancements and NN evaluation = state of the art for go
- Part II: constraints, propagation, search
  - state of the art for sudoku
- Lab on Monday

Part II: Search and inference

- Today we saw monte carlo tree search
  - With enhancements and NN evaluation = state of the art for go
- Part II: constraints, propagation, search
  - state of the art for sudoku
- Lab on Monday
- Assignment 1 due: see Brightspace

Part II: Search and inference

- Today we saw monte carlo tree search
  - With enhancements and NN evaluation = state of the art for go
- Part II: constraints, propagation, search
  - state of the art for sudoku
- Lab on Monday
- Assignment 1 due: see Brightspace
- Questions? TUD Answers

What is still unclear?

After every lecture...

Give us some homework:
tell us what is still unclear.

References I

Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.