Elements of DSAI

AlphaGo Part 1: Introduction, Single-Agent Search

Search & Learn: A Recipe for AI Action Decisions

Jörg Hoffmann

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Winter Term 2019/20

Game Playing

Why AI Game Playing?

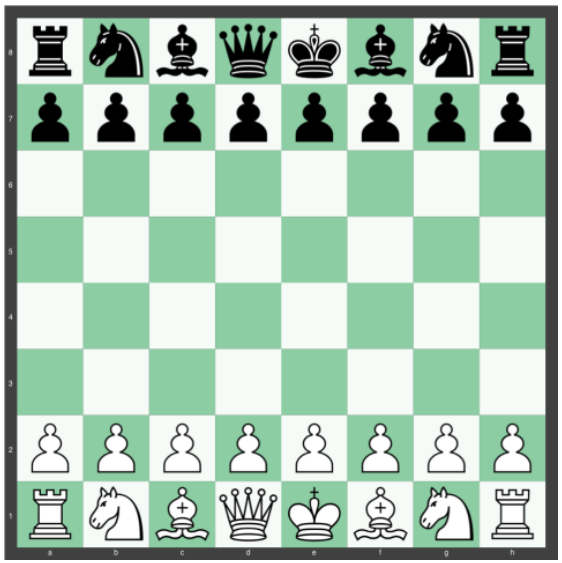
- Playing a game well is commonly viewed to require a form of "intelligence".
- Games capture a pure form of competition between opponents.
- Games are abstract and precisely defined, thus very easy to formalize and implement in a computer.

→ Game playing is one of the oldest topics in AI.

→ The dream of a machine that plays Chess is much older than AI!

Why is Game Playing Hard?

Number of possible board positions: (let's do some combinatorics)



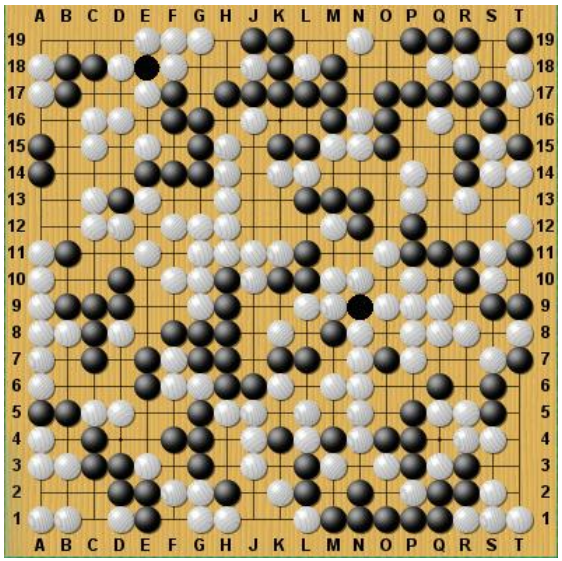
- Choose position for white king: 64
- Choose position for white queen: 63
- 2 positions for bishops (Läufer): $\binom{62}{2}$
- Knights (Springer) $\binom{60}{2}$, rooks (Türme) $\binom{58}{2}$
- 8 positions for pawns (Bauern): $\binom{56}{8}$
- Similar for black pieces on 48 remaining squares.

$\rightarrow 4.6 * 10^{42}$. But that's just those positions where all pieces are present!

Estimated number of legal positions: ca. 10^{43} (estimated number of atoms in the universe: 10^{82}).

Why is Game Playing Hard?

Number of possible board positions:



- 3 possibilities per square: white, black, empty
- $19 * 19 = 361$ squares

$\rightarrow 3^{361} \approx 1.7 * 10^{172}$. (Reminder: 10^{82} atoms in the universe.)

Number of legal positions: ca. 10^{171} (actually it's
208168199381979984699478633344862770286522453884530548425639456820927419612738015378525648451698519643907259916015628128546089888314427129715319317557736620397247064840935, see `https://tromp.github.io/go/legal.html`)

State of the Art, 1 B.A. (before AlphaGo)

- **Checkers (Dame):** Since 1994, "Chinook" is world champion. In 2007, it was shown to be *unbeatable*: Checkers is solved, we know the optimal game strategy.
- **Chess:** In 1997, "Deep Blue" beat Garry Kasparov.

Duell Kasparow gegen Deep Blue (1997): *Demütigende Niederlage*

  - 6 games, final score 3.5 : 2.5.
  - Specialized Chess hardware, 30 nodes with 16 processors each.
  - Nowadays, standard PC hardware plays at world champion level.

- **Go:** Best computer players ("Zen", "Mogo", "Crazystone") at the level of good amateurs, using sampling-based search methods.

Computer Chess: Famous Quotes

Claude Shannon (1949)

The chess machine is an ideal one to start with, since

1. the problem is sharply defined both in allowed operations (the moves) and in the ultimate goal (checkmate),
2. it is neither so simple as to be trivial nor too difficult for satisfactory solution,
3. chess is generally considered to require thinking for skilful play, [. . .]
4. the discrete structure of chess fits well into the digital nature of modern computers.

Alexander Kronrod (1965)

Chess is the drosophila of Artificial Intelligence.

$\rightarrow$ For those not in the know: "drosophila" = fruit fly, a species heavily used for research in genetics.

Computer Chess: Another Famous Quote

John McCarthy (1997)

In 1965, the Russian mathematician Alexander Kronrod said, "Chess is the Drosophila of artificial intelligence."

However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts on breeding racing Drosophilae. *We would have some science, but mainly we would have very fast fruit flies.*

Very Fast Fruit Flies, 1 B.A.

Computer Chess: (mostly similar in other games)

- Exhaustive search (Alpha-Beta, see later).
- Fast game-position evaluation functions, fine-tuned by human experts and training.
- Large database of known game positions for drawing analogies (from 2 million games, in 1997).
- Very large game opening databases.
- Very large game termination databases.
- Fast hardware.

$\rightarrow$ A mixture of (a) very fast search and human expertise. Learning plays only a minor role!

State of the Art, 3 A.A. (Anno AlphaGo)

AlphaGo, March 2016:

https://www.netflix.com/de-en/title/80190844

- AlphaGo beats Lee Sedol (winner of 18 world titles).
- Sampling-based search with guidance information from neural networks (NN), trained by expert data and self-play.

AlphaGo Zero, early 2017:

https://deepmind.com/blog/alphago-zero-learning-scratch/

- AlphaGo Zero beats AlphaGo using NN trained without expert data.
- Attention: Training time in "days" requires Google computing power! Intensely computation-expensive but massively parallelizable.

AlphaZero, late 2017:

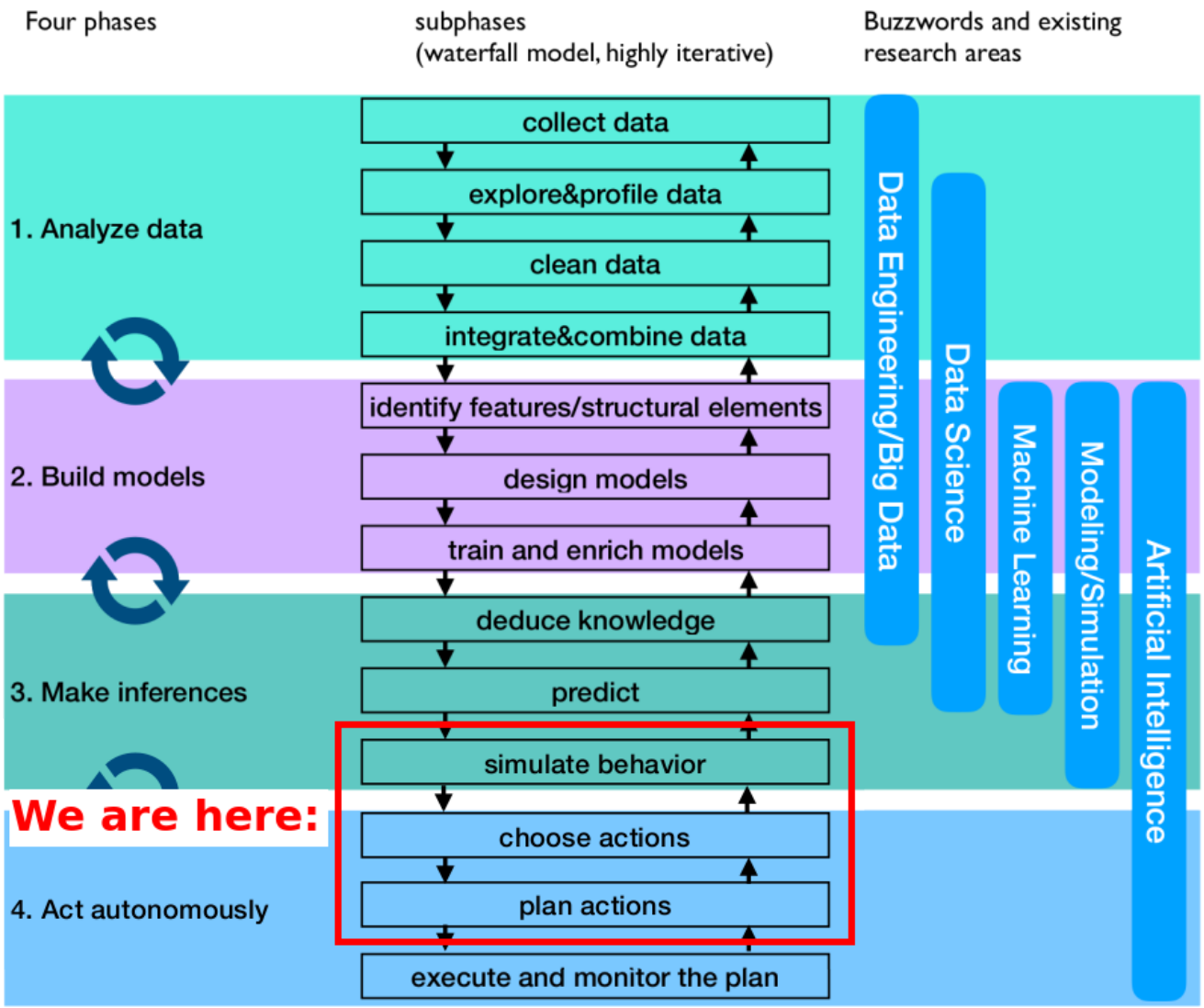
https://deepmind.com/blog/alphazero-shedding-new-light-grand-games-chess-shogi-and-go/

- AlphaZero beats world-class computer players in Go, chess, and shogi.

Agenda: Search Algorithms in Single-Agent Setting

- **Search Trees:** What do we mean by "search" here?
  → We look at an example to illustrate the concept.
- **Blind Search:** How to explore the search tree?
  → Simple algorithms first (we'll be quick on this).
- **Heuristic Search:** How to explore the search tree *intelligently*?
  → The simple algorithms usually don't work. We need to inform the search about where it is promising to explore. Here's how.
- **Monte-Carlo Tree Search (MCTS):** Can we explore the search tree more sparsely?
  → Sampling into the bargain. Recent successes, including AlphaGo.

Positioning in the DSAI Phase Model



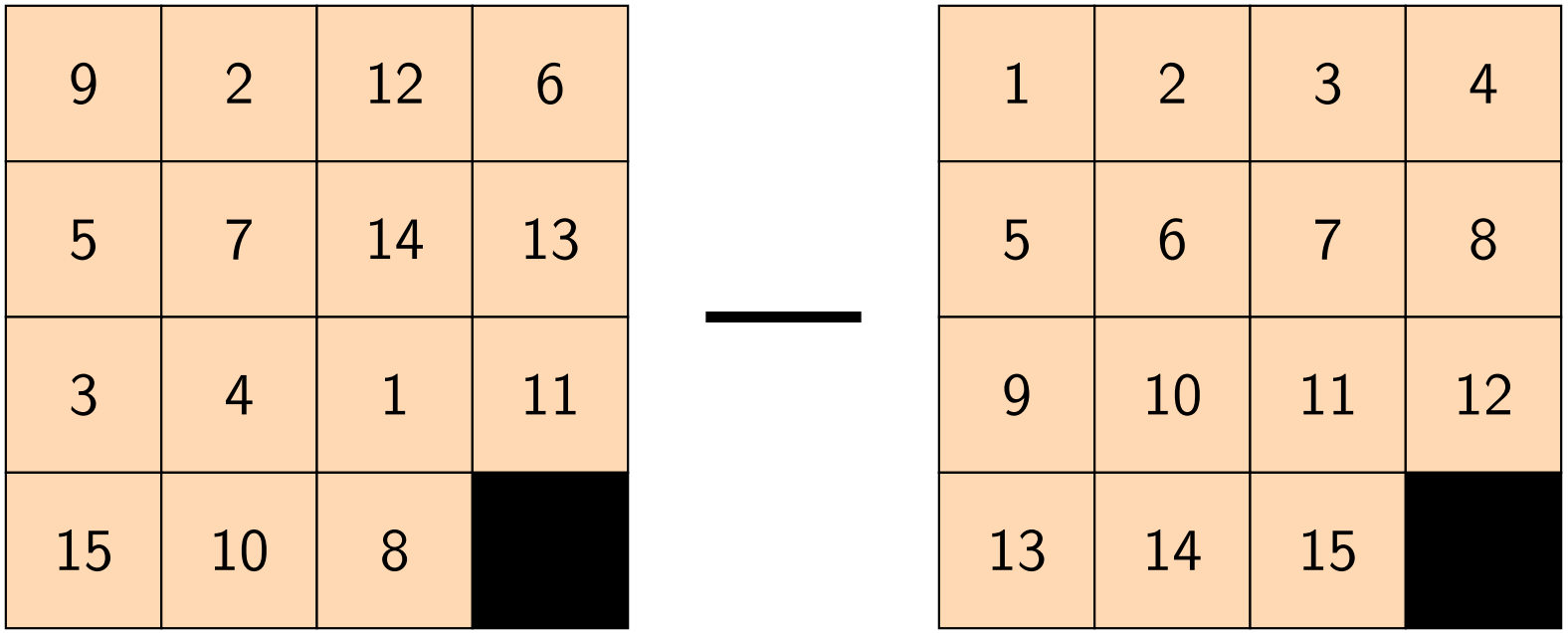
A Single-Agent Search Problem

→ Problem: Find a route to Moscow.

- Starting from an initial state . . . (SB)
- . . . apply actions . . . (Using a road segment)
- . . . to reach a goal state. (Moscow)
- Performance measure: Minimize summed-up action costs. (Road segment lengths)

Another Single-Agent Search Problem (The "15-Puzzle")

→ Problem: Move tiles to transform left state into right state.



- Starting from an initial state . . . (Left)
- . . . apply actions . . . (Moving a tile)
- . . . to reach a goal state. (Right)
- Performance measure: Minimize summed-up action costs. (Each move has cost 1, so we minimize the number of moves)

Single-Agent Search Problems

Restricted environment: (yet practical applications, see next slide)

- Single-agent.
- Finite numbers of states and actions (in particular: discrete).
- Fully observable (agent knows everything).
- Deterministic (each action has only one outcome).
- Static (if the agent does nothing, the world doesn't change).

$\rightarrow$ All of these restrictions can be removed. For example, many of the basic ideas generalize easily to two-player zero-sum games.

Single-Agent Search Applications

Just to name a few:

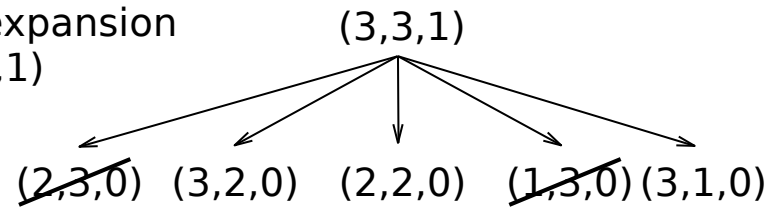
- Route planning (e.g. Google Maps).
- Detecting bugs in software and hardware.
- Non-player-characters in computer games.
- Robot assembly sequencing. Planning of the assembly of complex objects. Actions = robot activities.
- Attack planning. Finding a hack into a secured network. Used for regular security testing. Actions = exploits.
- Query optimization in databases. Actions = rewriting operations.
- Sequence alignment in Bioinformatics. Actions = re-alignment operations.
- Natural language sentence generation. Actions = add another word to a partial sentence.

How to Search?

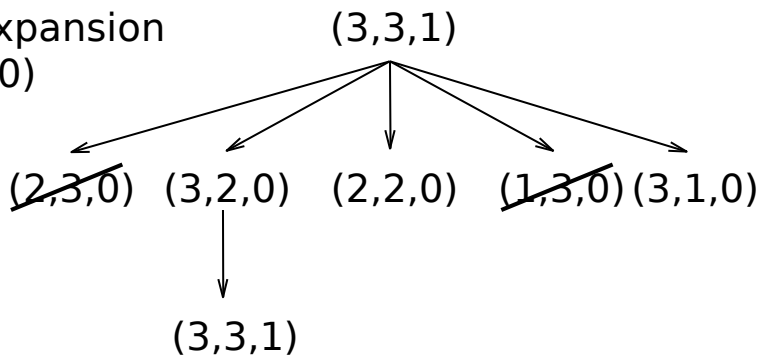
Start at the initial state. Then, step-by-step, expand a state by generating its successors . . . $\rightarrow$ Search space.

(a) initial state (3,3,1)

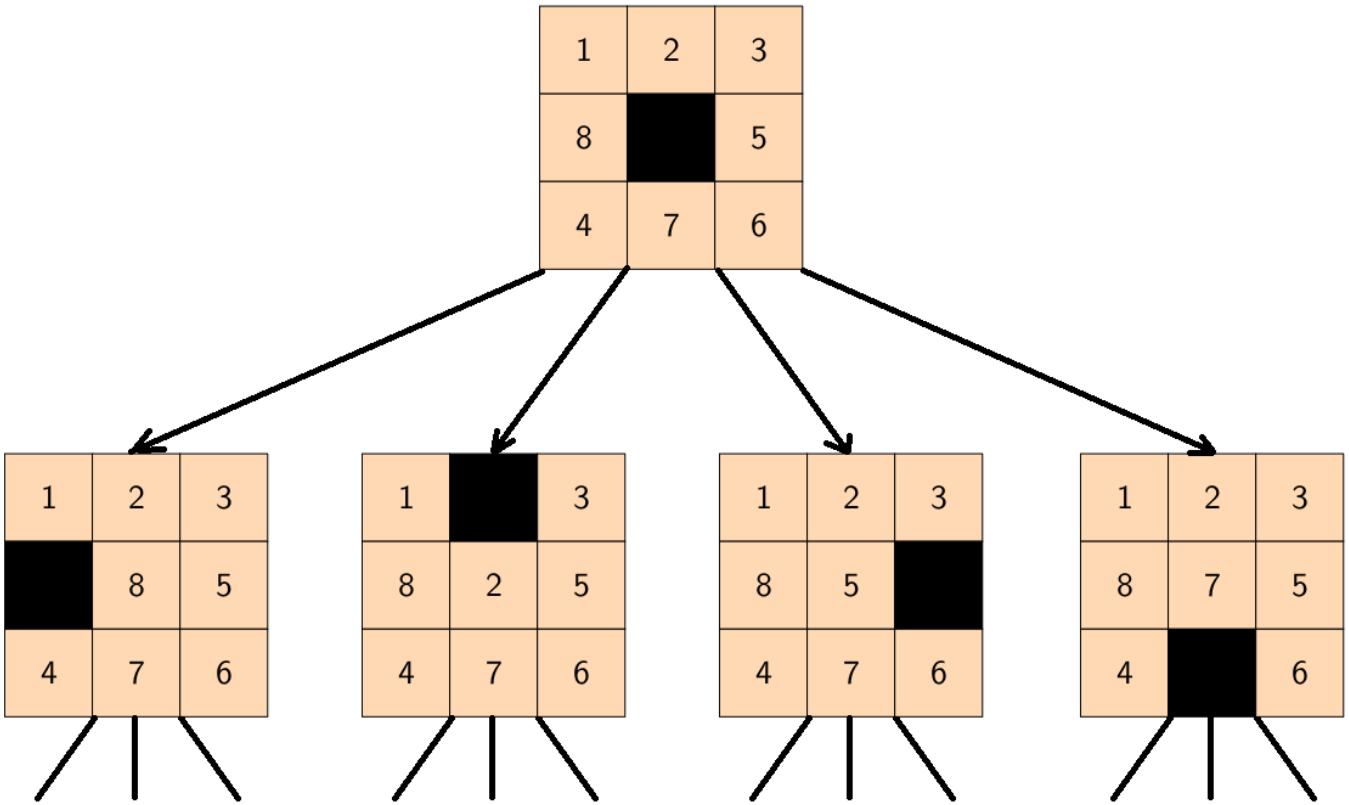
(b) after expansion of (3,3,1)



(c) after expansion of (3,2,0)



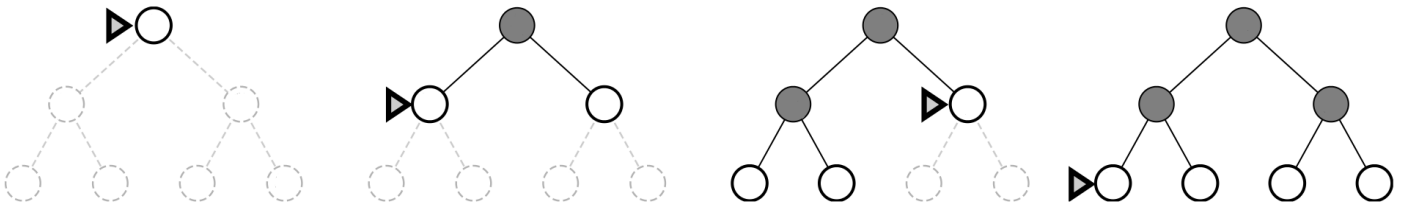
An Example Search Space: The "8-Puzzle"



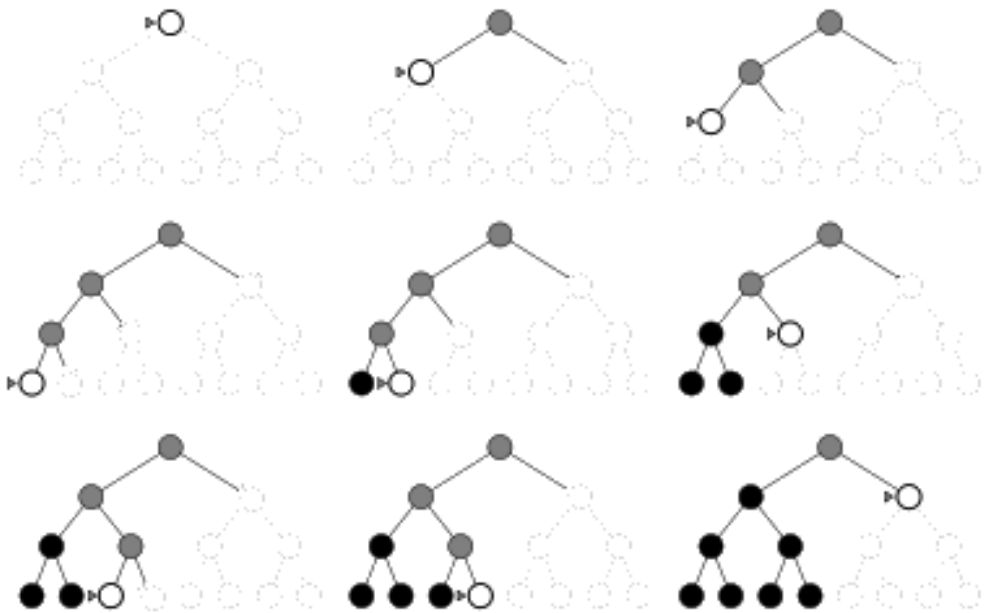
etc. . . . (Blackboard)

Breadth-First Search and Depth-First Search

Breadth-First Search: Expand nodes in the order they were generated.



Depth-First Search: Always expand the most recently generated node.



Algorithm Properties

Breadth-First Search:

- Does this guarantee to find a solution if there is one? Yes.
- Does this provide a guarantee on solution quality? Always finds a shortest solution.

Depth-First Search:

- Does this guarantee to find a solution if there is one? No, because search branches may be infinitely long (cycles).
- Does this provide a guarantee on solution quality? Trivially no.

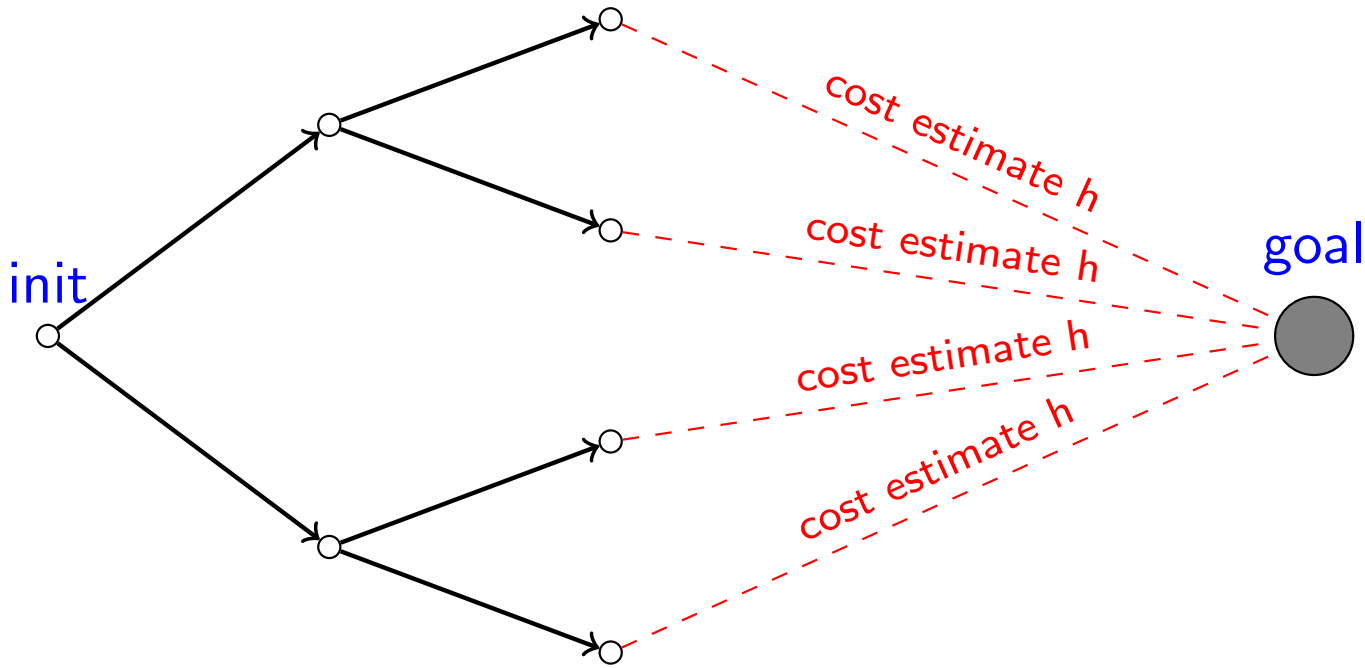
$\rightarrow$ There are many more algorithms improving over these in a variety of ways. See, for example, [Russell and Norvig (2010)].

(Not) Playing Stupid

→ Problem: Find a route to Moscow.

- "Look at all locations 10km distant from SB, look at all locations 20km distant from SB, . . ." = **Breadth-first search.**
- "Keep following down arbitrary roads until you hit an ocean, then back up . . ." = **Depth-first search.**
- "Focus on roads that go the right direction." = **Heuristic search!**

Heuristic Search: Basic Idea

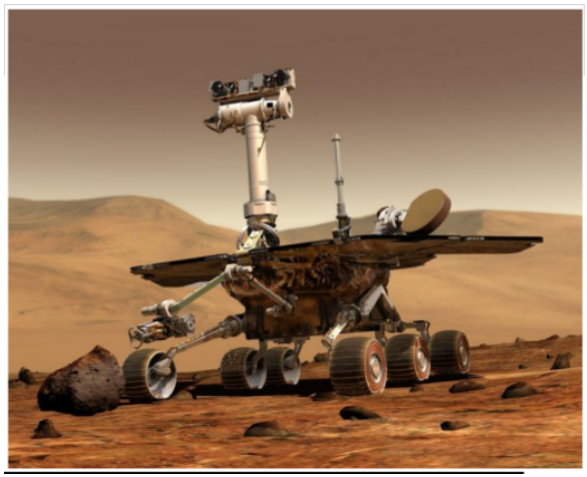


$\rightarrow$ Heuristic function h estimates the cost of an optimal path from a state s to the goal; search prefers to expand states s with small $h(s)$.

Some Applications

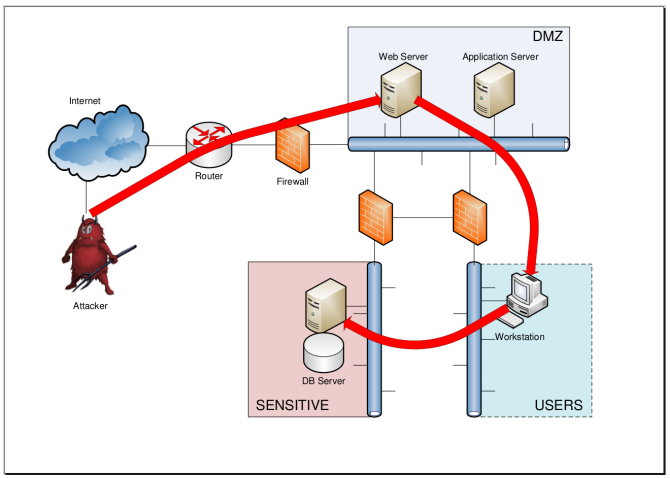
GPS

Robotics



Video Games

Security Testing



Heuristic Functions

Definition: *Given a single-agent search problem with states S, a heuristic is a function $h : S \mapsto \mathbb{R}_0^+$.*

Intended use:

- h estimates the cost of a cheapest path from s to a goal state.

Terminology:

- Ancient Greek $\varepsilon\upsilon\rho\iota\sigma\kappa\varepsilon\iota\nu$ ($=$ "I find"); aka: $\varepsilon\upsilon\rho\eta\kappa\alpha$!
- Same word often used in CS for "rule of thumb", "imprecise solution method".

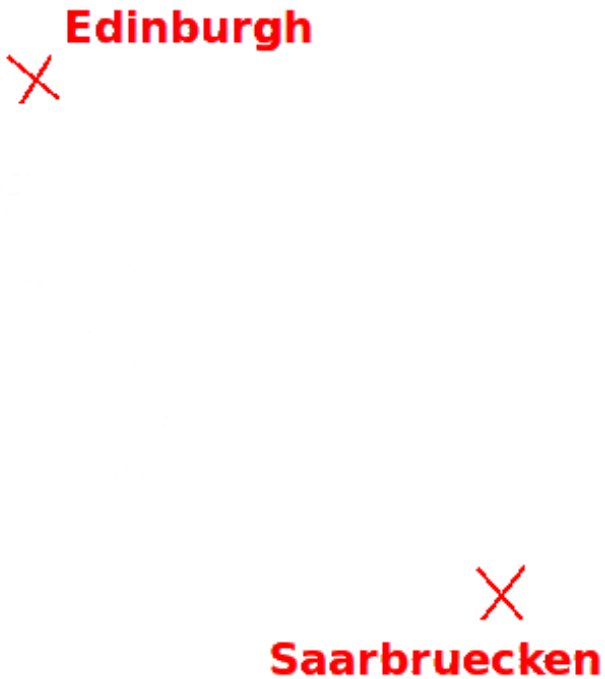
How to obtain h?

- Goal distance within a relaxed (i.e., simplified) problem.
- Or: learn from data, see later (games).

Heuristic Function from Relaxed Problem: Example

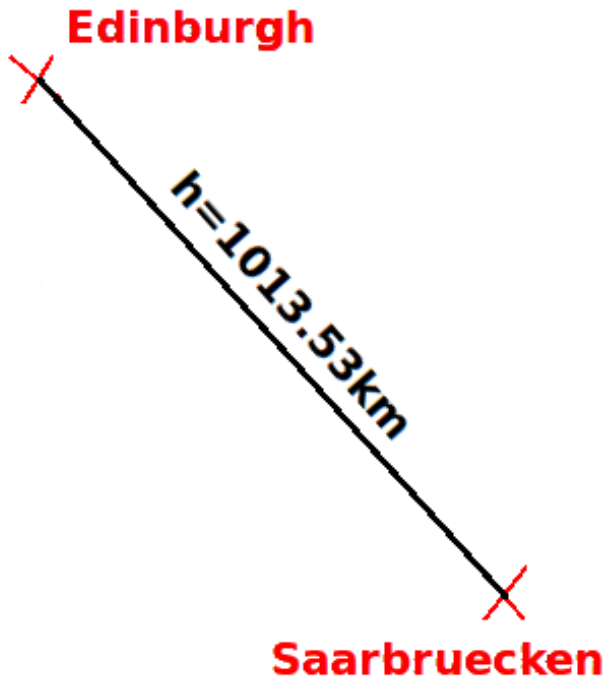
Problem: Find a route from Saarbrücken To Edinburgh.

Heuristic Function from Relaxed Problem: Example



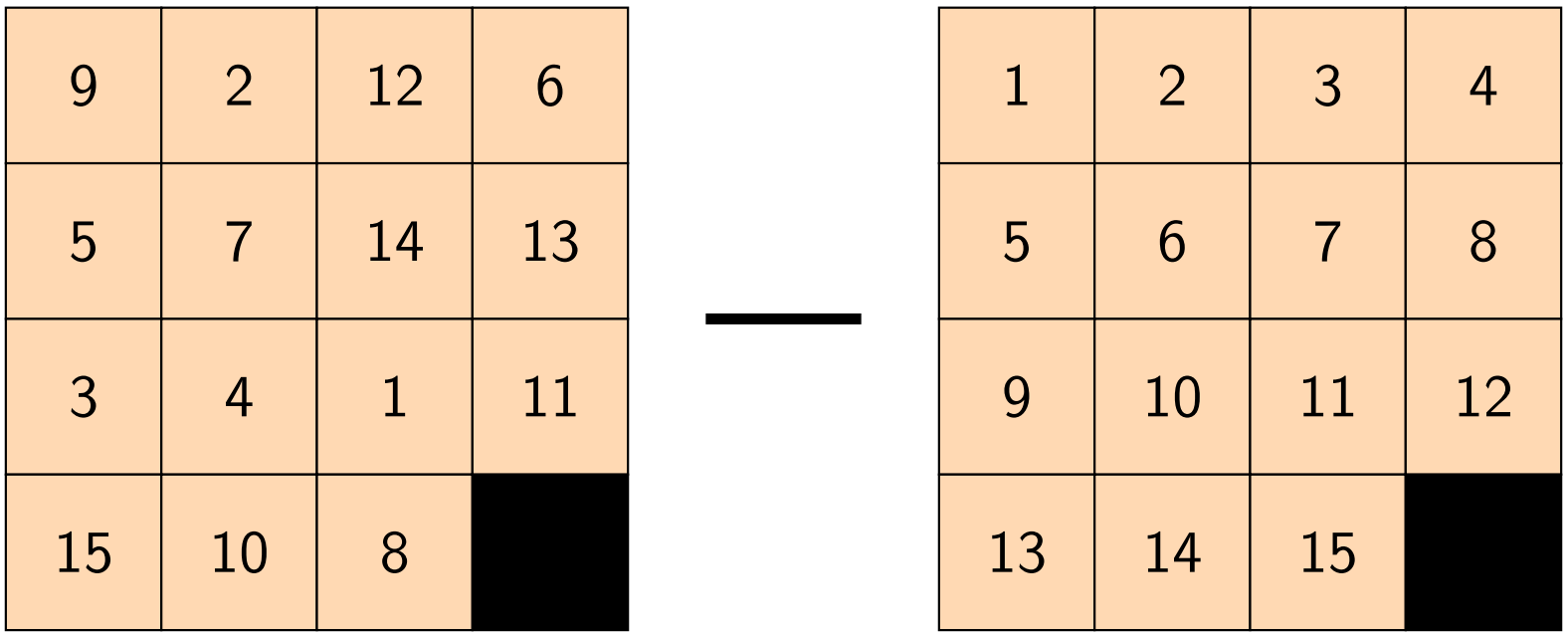
Relaxed Problem: Throw away the map.

Heuristic Function from Relaxed Problem: Example



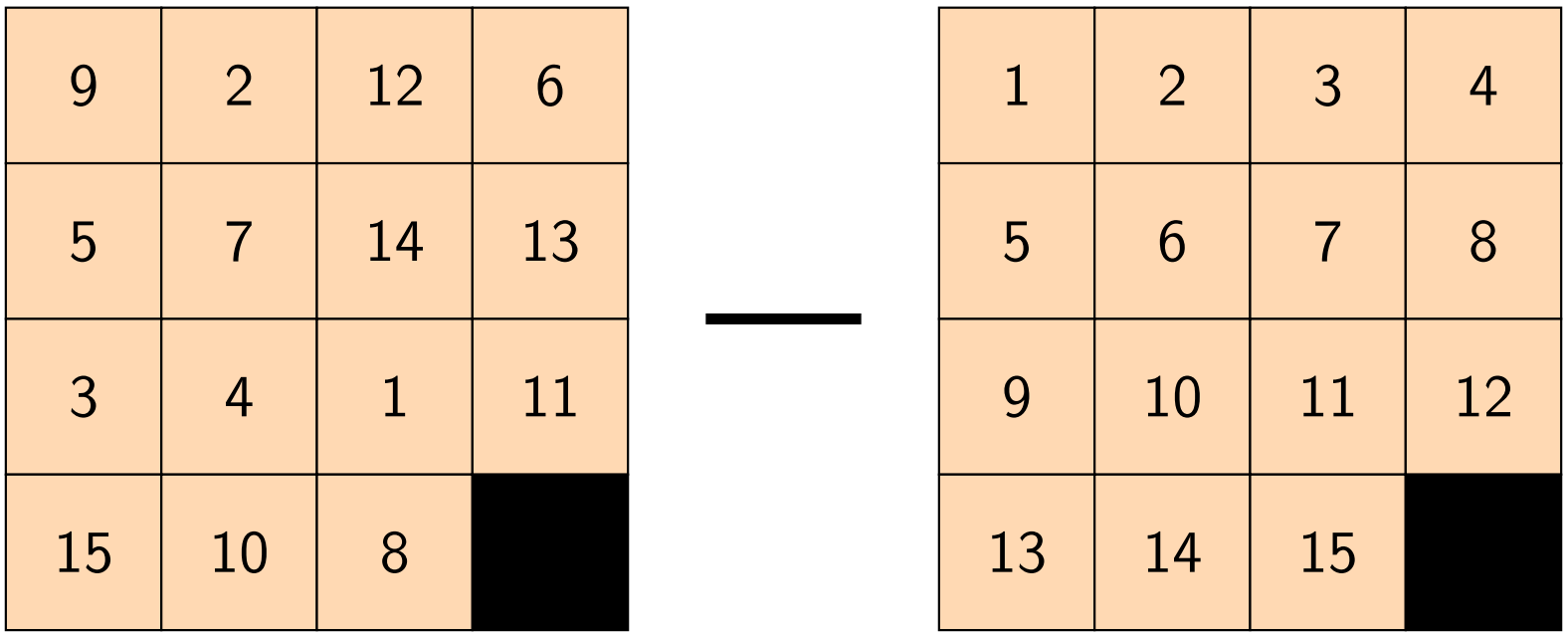
Heuristic function h: Straight line distance.

h from Relaxed Problem: Another Example



- Problem: Move tiles to transform left state into right state.
- Relaxed Problem: Allow to move each tile to *any cell* (in a single move), regardless of the situation.
- Heuristic function h: Number of misplaced tiles. (Here: 13)

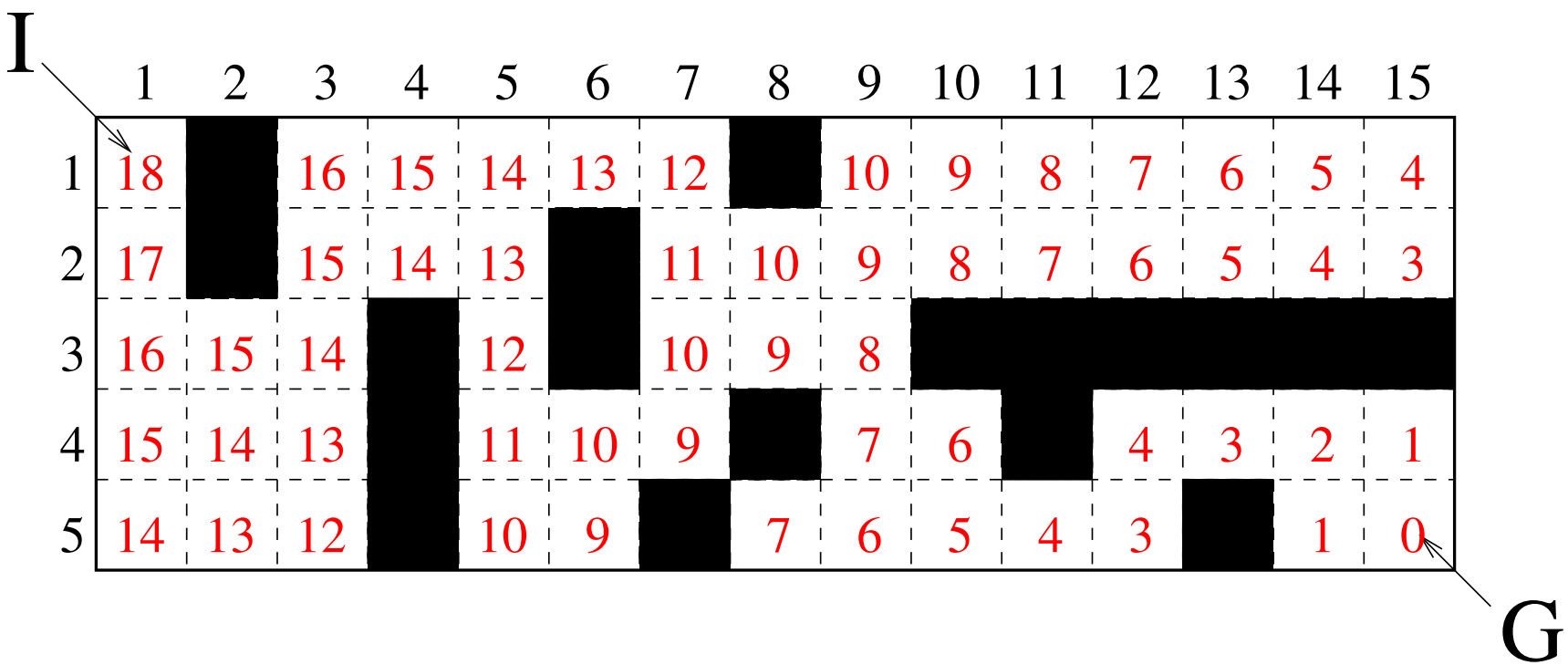
h from Relaxed Problem: Another Example



- Problem: Move tiles to transform left state into right state.
- Relaxed Problem: Allow to move each tile to *any neighbor cell*, regardless of the situation.
- Heuristic function h: Manhattan distance. (Here: 36)

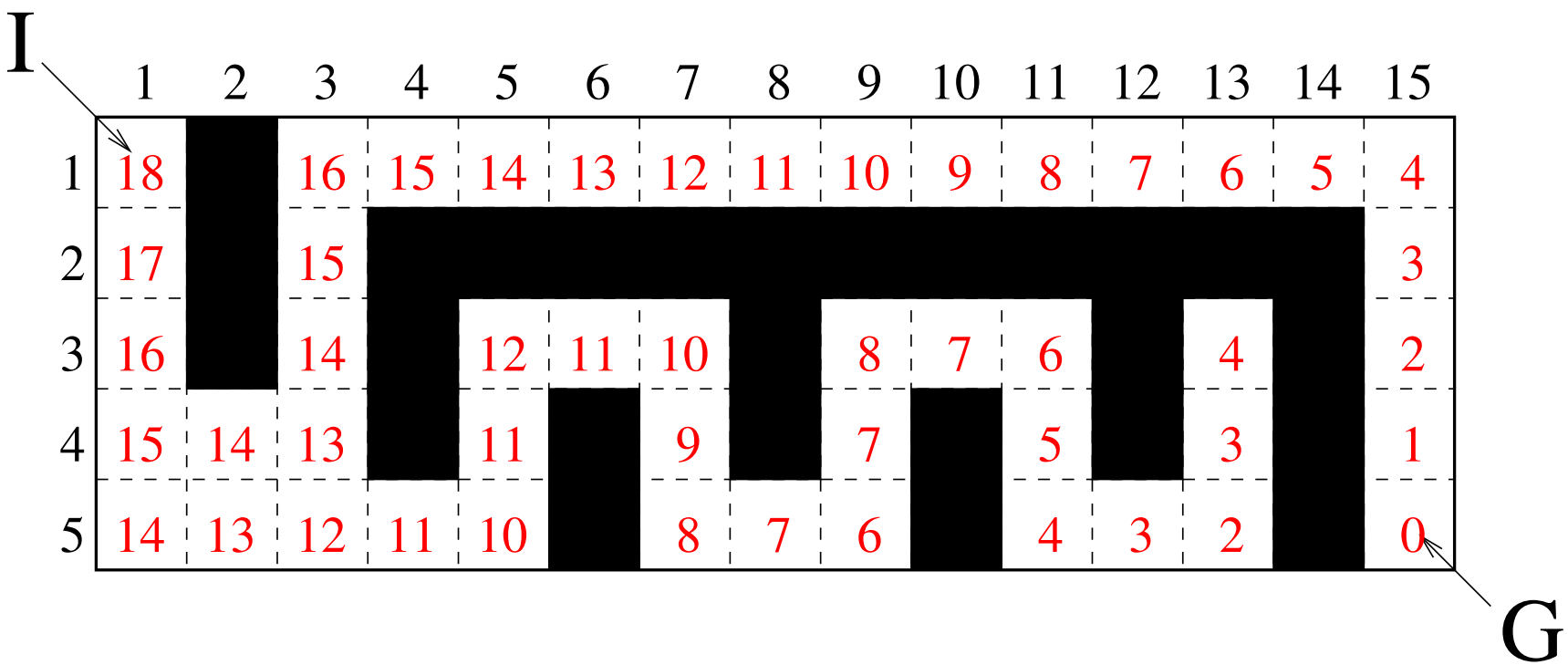
Heuristic Functions: Illustration Path Planning

Manhattan Distance, good case:



Heuristic Functions: Illustration Path Planning

Manhattan Distance, bad case:



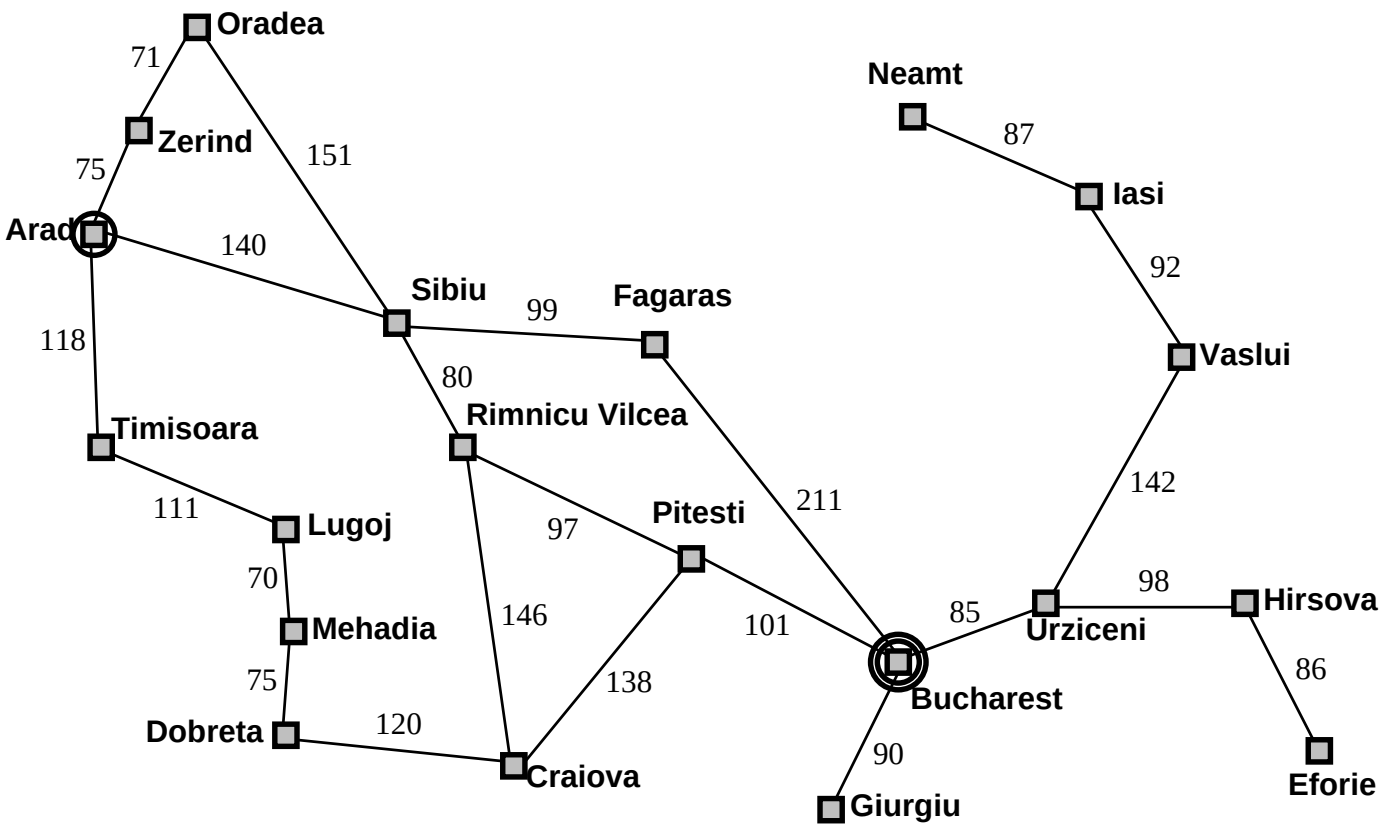
Greedy Best-First Search

Search nodes ordered by ascending h:

function Greedy Best-First Search(*problem*) **returns** a solution, or failure
  node $\leftarrow$ a node n with n.*State*=*problem.InitialState*
  frontier $\leftarrow$ a priority queue ordered by ascending h, only element n
  loop do
    if *Empty?(frontier)* **then return** failure
    $n \leftarrow$ *Pop(frontier)*
    if *problem.GoalTest*(n.*State*) **then return** *Solution*(n)
    for each *action* a **in** *problem.Actions*(n.*State*) **do**
      $n' \leftarrow$ *ChildNode(problem,*n*,*a*)*
      Insert(n', $h(n')$, *frontier*)

- Typically used: duplicates pruning. Insert n' only if n.State has not been seen before.

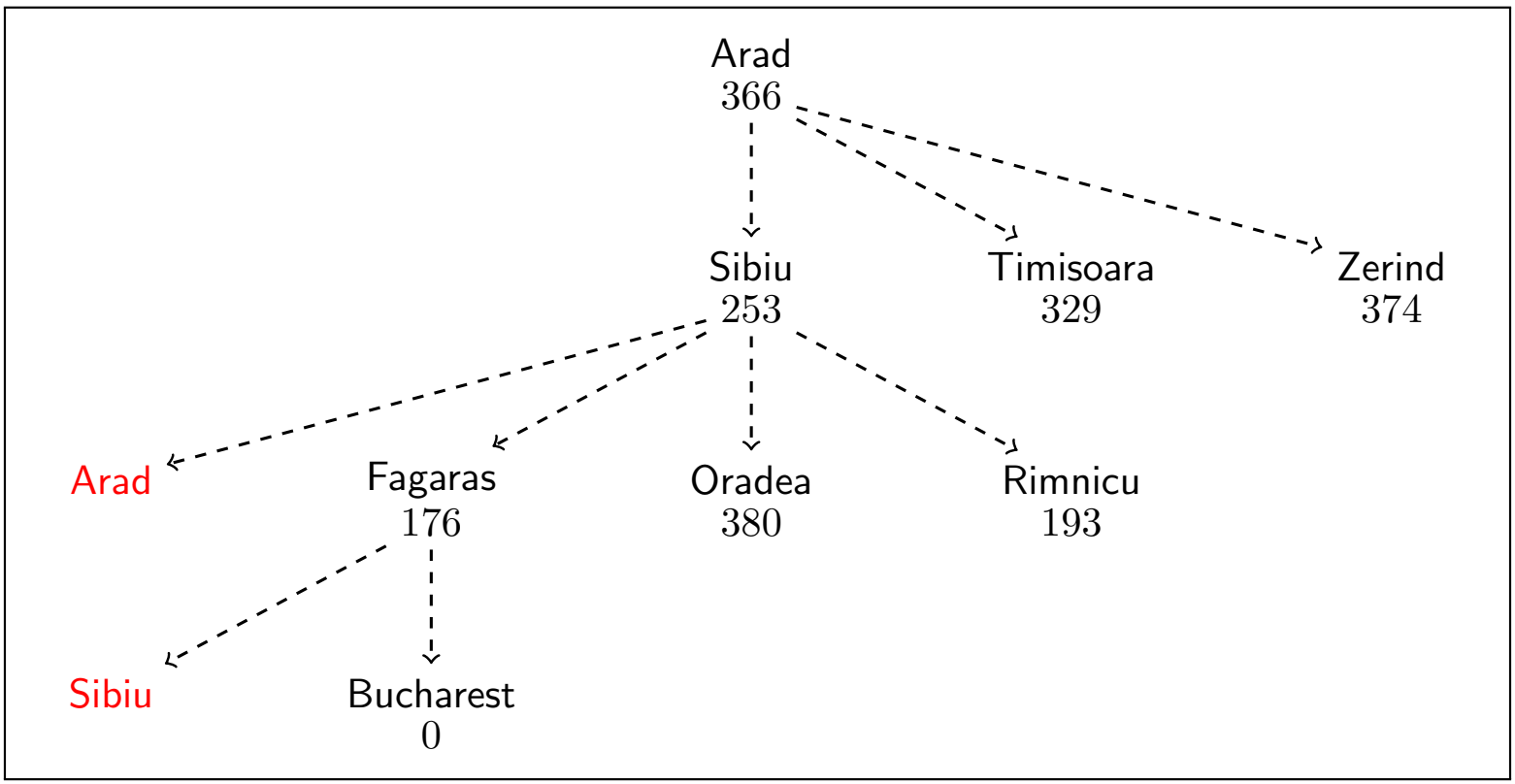
Greedy Best-First Search: Route to Bucharest



Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-First Search: Route to Bucharest

Subscripts: h. Red nodes: removed by duplicate pruning.



A*

Ordered by ascending $g + h$: ($g(n) =$ cost from initial state to n)

function A*(*problem*) **returns** a solution, or failure
  node $\leftarrow$ a node n with n.*State*=*problem.InitialState*
  frontier $\leftarrow$ a priority queue ordered by ascending $g + h$, only element n
  loop do
    if *Empty?(frontier)* **then return** failure
    $n \leftarrow$ *Pop(frontier)*
    if *problem.GoalTest*(n.*State*) **then return** *Solution*(n)
    for each *action* a **in** *problem.Actions*(n.*State*) **do**
      $n' \leftarrow$ *ChildNode(problem,*n*,*a*)*
      Insert(n', $g(n') + h(n')$, *frontier*)

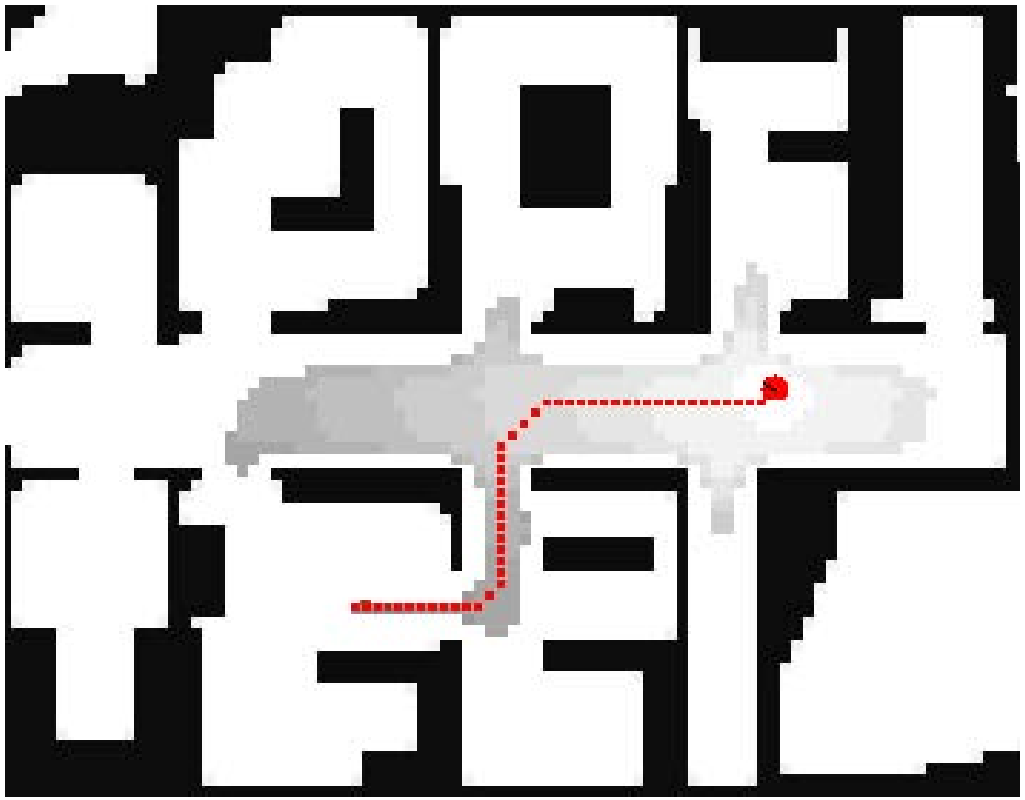
- Optimal if h is admissible, i.e. never over-estimates the true cost-to-goal.
- Duplicates pruning: possible but must take care to not affect optimality.

Performance: A^* in the 8-Puzzle

Without Duplicate Elimination; d = length of solution:

	Number of search nodes generated		
	Iterative	A^* with	
d	Deepening Search	misplaced tiles h	Manhattan distance h
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	3644035	227	73
14	-	539	113
16	-	1301	211
18	-	3056	363
20	-	7276	676
22	-	18094	1219
24	-	39135	1641

Empirical Performance: A^* in Path Planning



Live illustration:

`http://qiao.github.io/PathFinding.js/visual/`

Monte-Carlo Sampling

Setting: Online, fixed time budget. Decide which action to take.

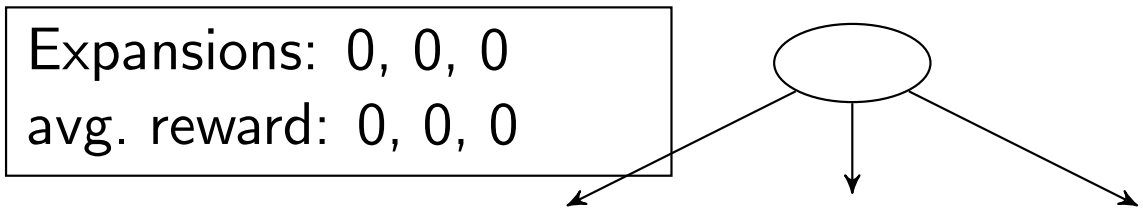
Algorithm: Current state s, choose action a

while time not up **do**
  select a transition $s \xrightarrow{a} s'$
  rollout from s' until terminal state t
  increment $\#expansions(a)$, update average $reward(a)$ with $reward(t)$
return an a for s with maximal average $reward(a)$

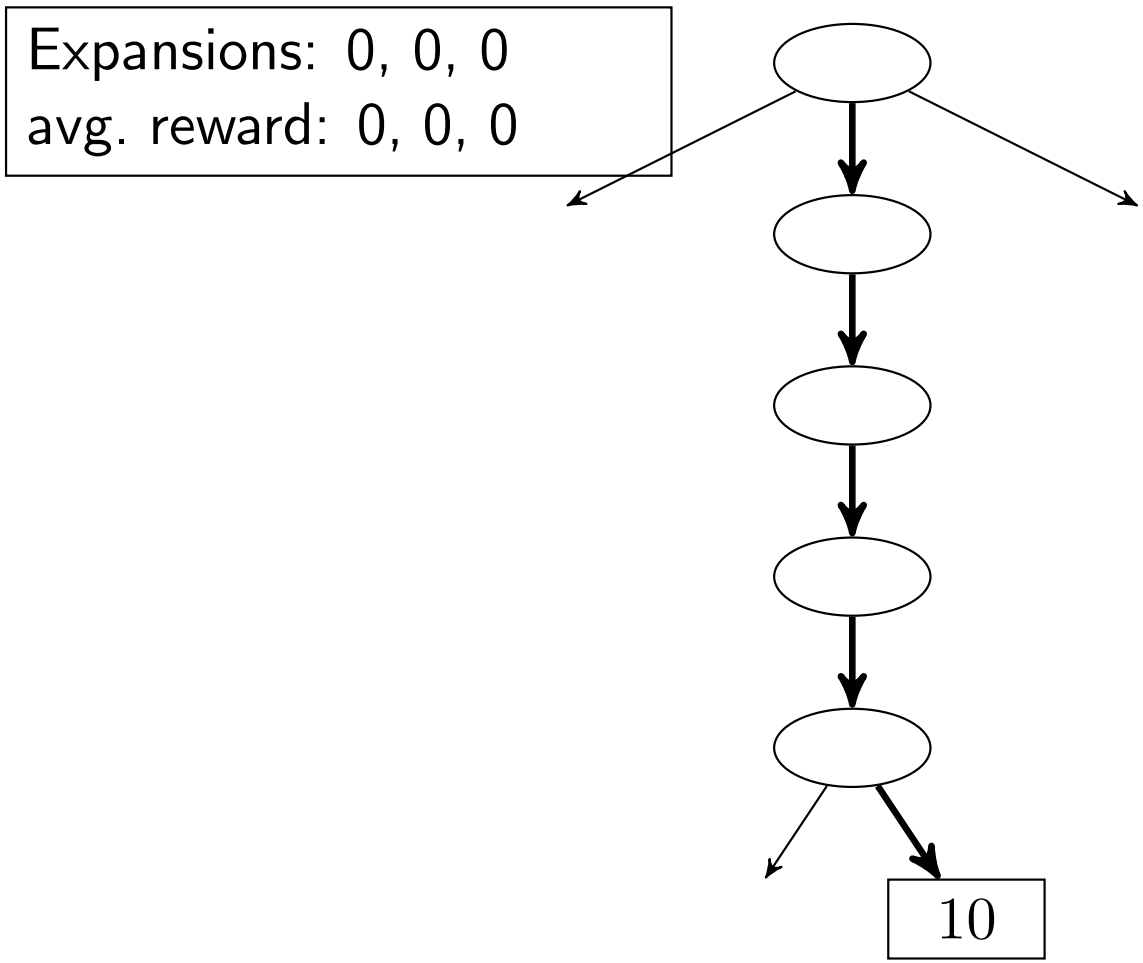
Notes:

- Sparse search, rollouts go deep fast.
- Guidance information can be injected into "select" and "rollout".
- Rollouts require "terminal state" to be reached in finite number of transitions. Intrinsic in games; not as clear in general.
- "Reward" at terminal state: intrinsic in games; here, based on path success (goal reached?) and, in the successful cases, plan cost.

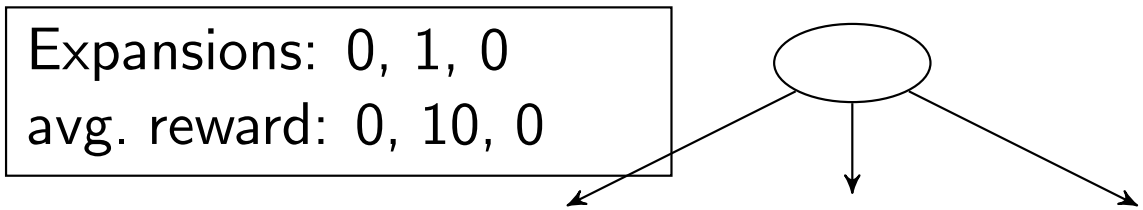
Monte-Carlo Sampling: Illustration



Monte-Carlo Sampling: Illustration

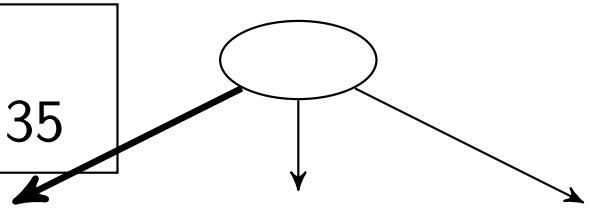


Monte-Carlo Sampling: Illustration

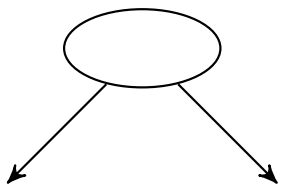


Monte-Carlo Sampling: Illustration

Expansions: 2, 2, 2
avg. reward: **60**, 55, 35



Monte-Carlo Sampling: Illustration



Expansions: 0, 0
avg. reward: 0, 0

Monte-Carlo Tree Search (MCTS)

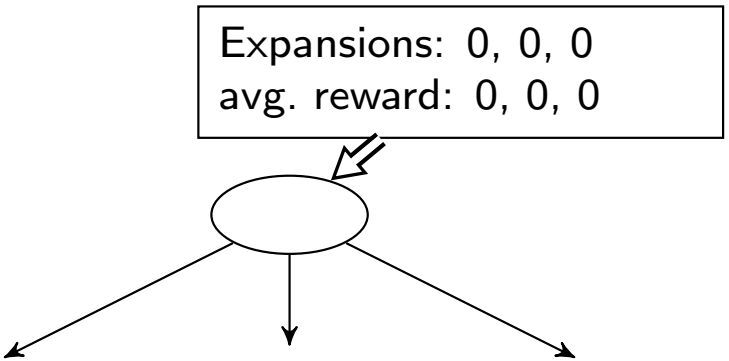
Algorithm: Same, but maintain a search tree T

while time not up **do**
  select actions within T up to a state s' and $s' \xrightarrow{a'} s''$ s.t. $s'' \notin T$
    with bias to maximize reward
  rollout from s'' until terminal state t
  add s'' to T
  update, from a' up to root, #expansions and average rewards
return an a for s with maximal average $reward(a)$
When executing a, keep the part of T below a

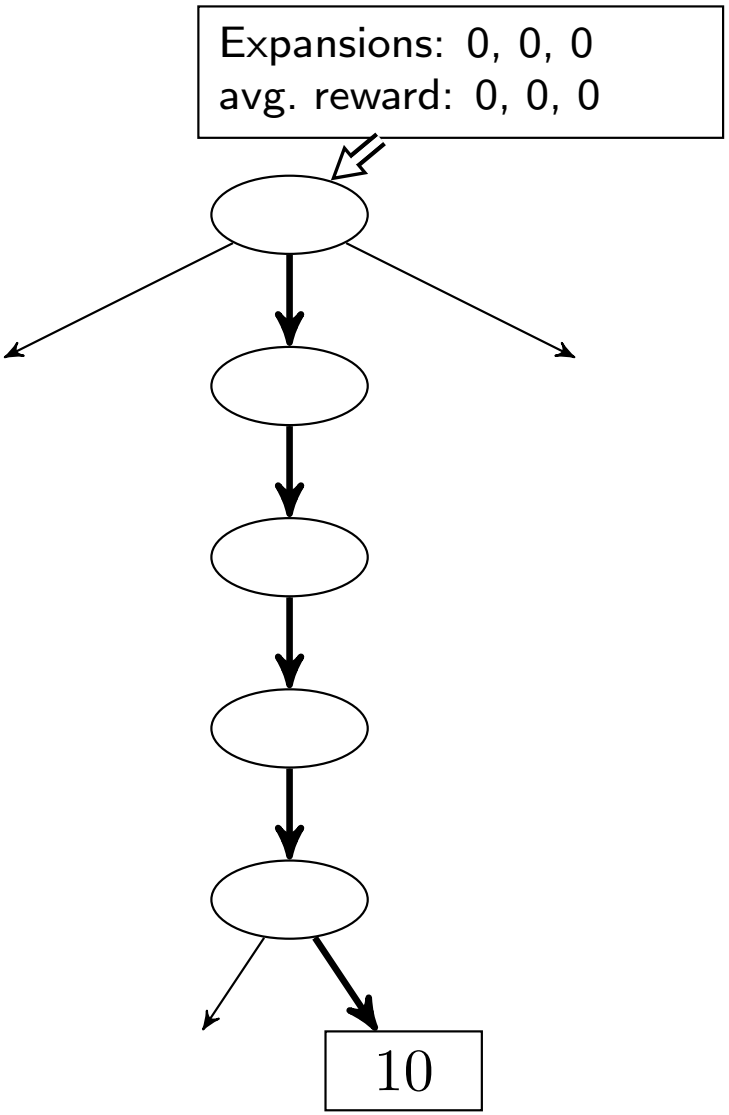
Notes:

- Search sparsity, information from deep samples, requirement terminal states and reward: as in Monte-Carlo Sampling.
- Action decisions (select) within tree. Maximize reward: exploitation; only bias, not exclusive: exploration, convergence to optimal choices.
- Different strategies/guidance within tree vs. rollouts.
- Tree remembers previous work, reduces redundant work later.

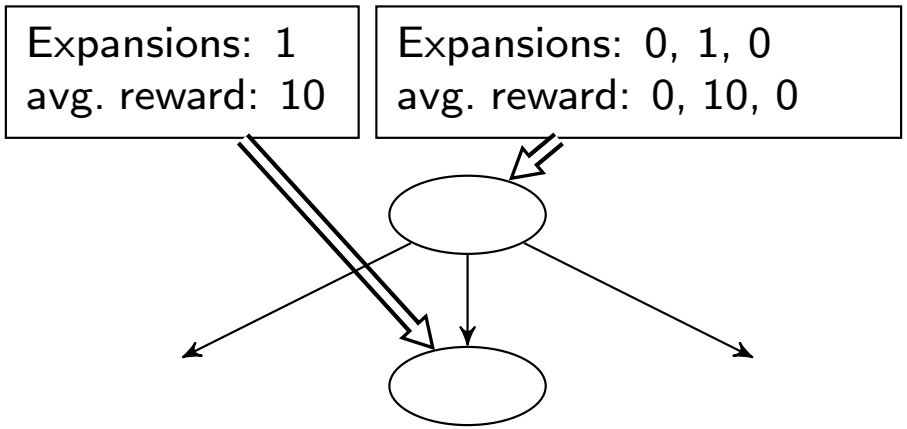
MCTS: Illustration



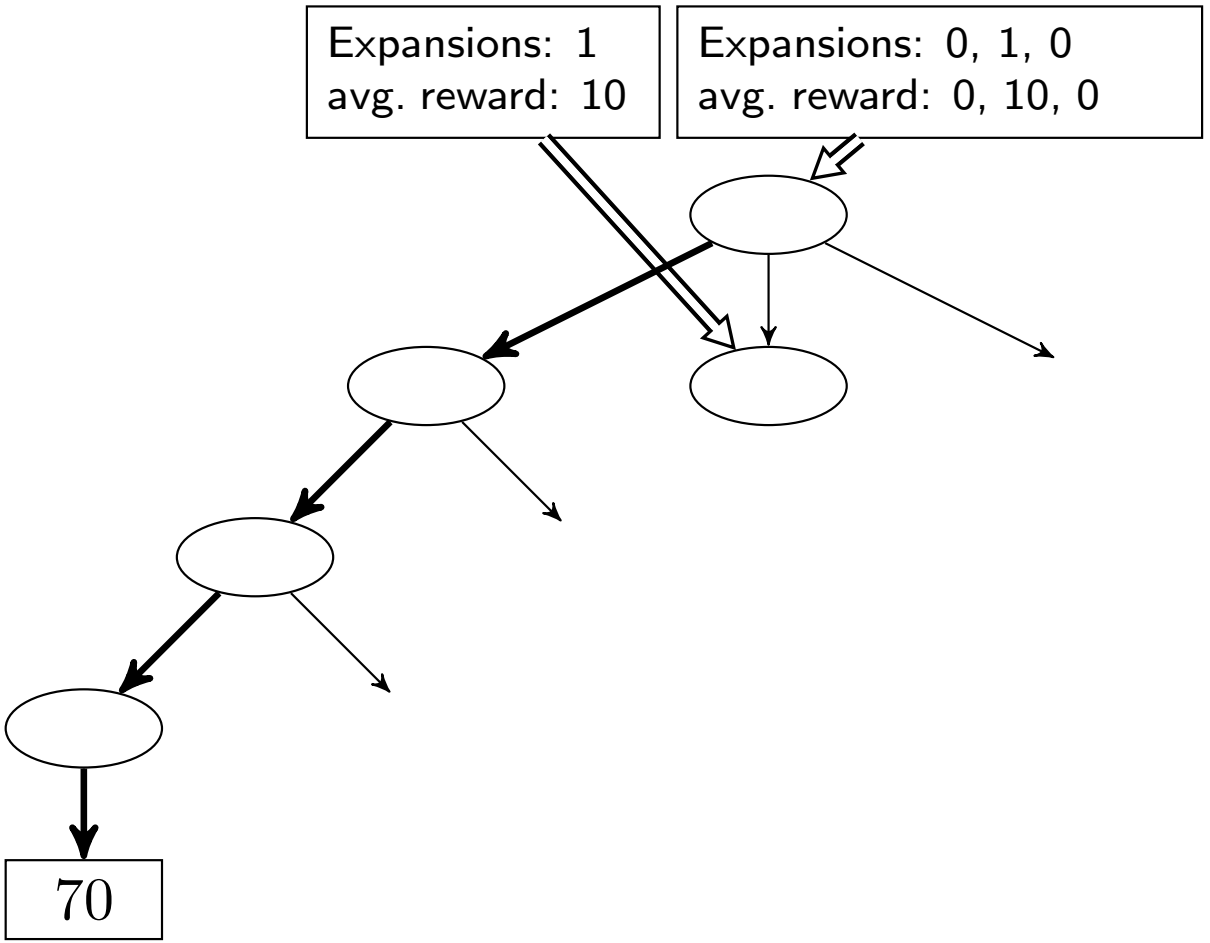
MCTS: Illustration



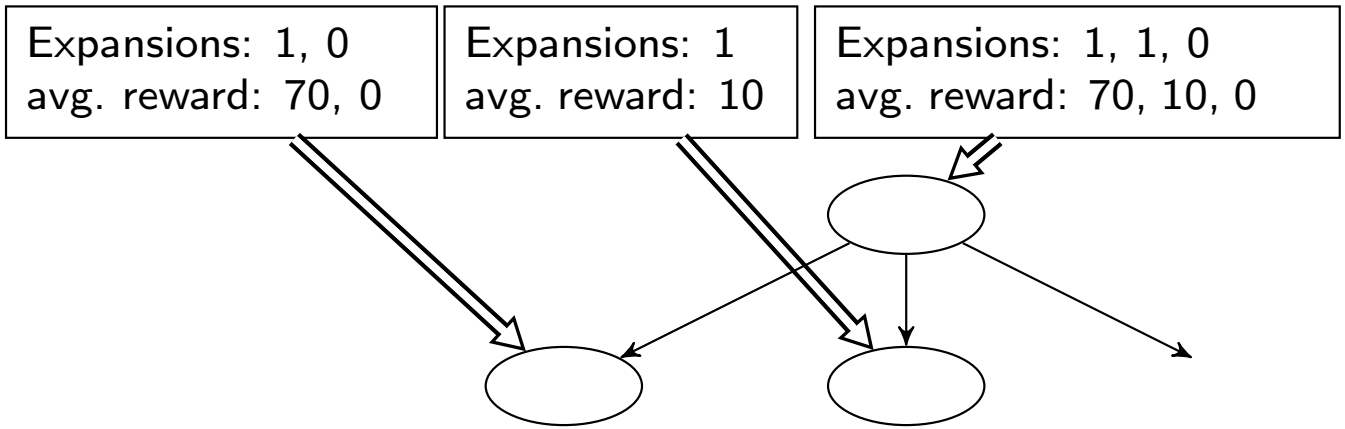
MCTS: Illustration



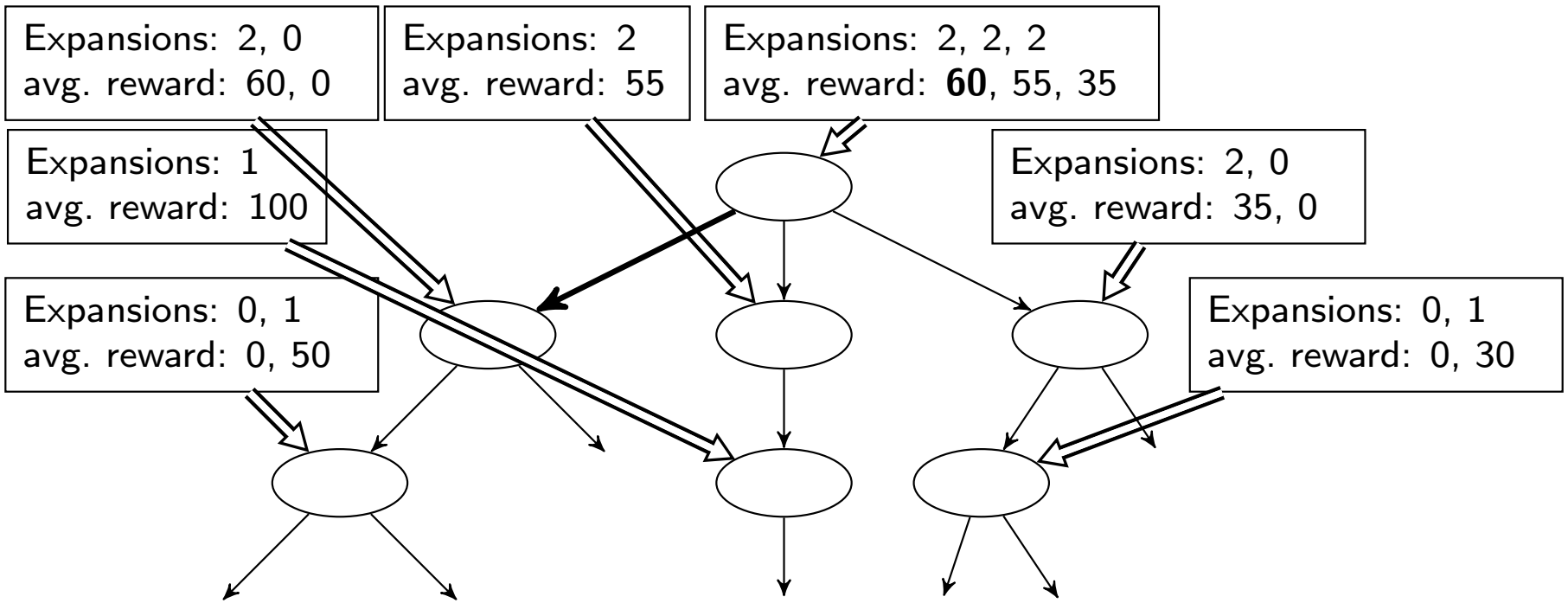
MCTS: Illustration



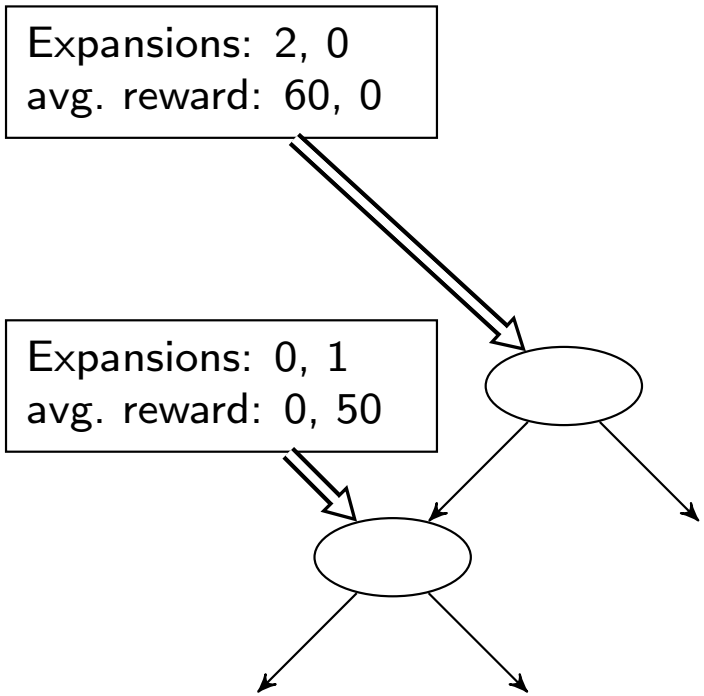
MCTS: Illustration



MCTS: Illustration



MCTS: Illustration



How to Guide MCTS? "Select" and "Sample"

Basic strategy: Exploitation vs. exploration

- Exploitation: Actions with high average → Focus on "what works".
- Exploration: Actions not tried a lot yet → Try to find better solutions.
- Balance the two, e.g. UCT [Kocsis and Szepesvári (2006)].

→ Strong link to reinforcement learning: exploitation = "use the action currently believed to be best"; exploration = "use some other action".

Additional information: Incorporate search guidance!

- Heuristic function: State quality estimate to guide "select" and/or "rollout".
- Approximate policy (action-choice function): Same, as action-selection bias.
- Based on problem relaxations (→ no training phase needed), or on learning (→ reinforcement learning across search iterations).

Summary

- Game playing is a traditional benchmark for AI. The AlphaGo/Zero line of works has dramatically advanced the state of the art, but there are many challenges still.
- The success of AlphaGo is based on a combination of search with learned search guidance.
- Single-agent search problems require to find a path of actions leading from an initial state to a goal state.
- Heuristic functions h map states to an estimate of goal distance. Greedy best-first search and A^* use h for ordering search nodes.
- Monte-Carlo tree search (MCTS) is a sparse form of search, going deep through rollouts. One needs to balance exploitation vs. exploration, similarly as in reinforcement learning. Heuristic functions and approximate policies can be integrated/learned.

Reading

- *Chapter 3: Solving Problems by Searching*, Sections 3.1 – 3.4 [Russell and Norvig (2010)].

  Content: Covers search trees and blind search in more detail.

- *Chapter 3: Solving Problems by Searching*, Sections 3.5 and 3.6 [Russell and Norvig (2010)].

  Content: Covers heuristic search in more detail.

- `https://en.wikipedia.org/wiki/Monte_Carlo_tree_search`

  Content: Quite a nice read giving lots of background. (MCTS is not covered by Russel/Norvig yet)

- *Trial-based Heuristic Tree Search for Finite Horizon MDPs* [Russell and Norvig (2010)].

  Content: Nice overview paper systematizing the space of MCTS-style algorithms in (probabilistic) AI Planning.

Quiz: Video Games

Quiz

When was the first game-playing computer built?

(A): 1941 (B): 1950

(C): 1958 (D): 1965

→ In 1941, a small box beat humans at Nim (take away objects from heaps, player taking the last object looses).

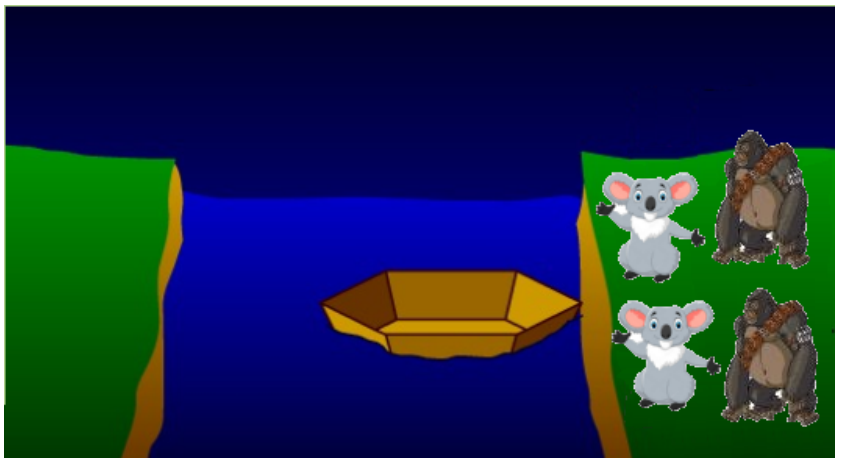
Quiz

Does the video game industry attempt to make the computer opponents as intelligent as possible?

(A): Yes (B): No

→ In some cases, yes (I suppose). In general, no. For example, in Ego-Shooter games, if your computer opponents did the best they can, you'd be shot immediately and always (similar: Rambo movies et al).

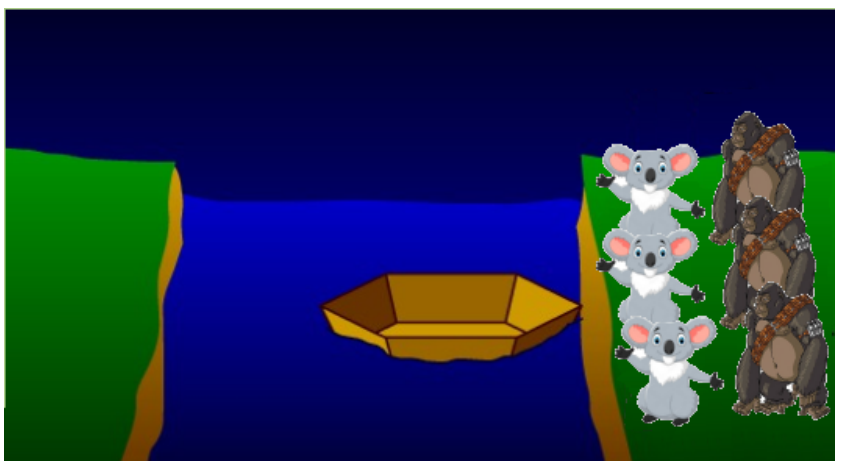
Quiz: Gorillas and Koalas



Quiz

How to solve this?

1. 1K,1G; 2. 1K; 3. 2K; 4. 1K; 5. 1K,1G.

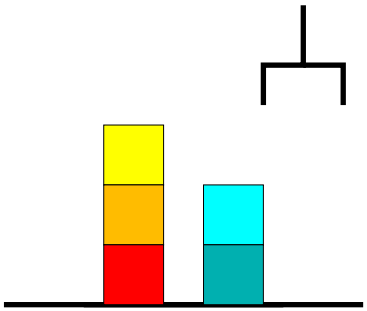


Quiz

And this?

Video: `http://www.youtube.com/watch?v=W9NEWxabGmg`

Quiz: Heuristic Error in the Blocksworld



- n blocks, 1 hand.
- A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.
- The goal is a set of statements "on(x,y)".

Quiz

Consider $h :=$ number of goal statements that are not currently true. Is the error (difference from true goal distance) bounded by a constant?

(A): Yes. (B): No.

$\rightarrow$ No. There are examples where the error grows linearly in n. Example: Block b_1 is currently beneath a stack of $b_n, \ldots, b_2$ and the goal is on(b_1, b_2). Then $h(s) = 1$ but true goal distance is n as we have to move $b_n, \ldots, b_2$ away first.

References I

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293, 2006.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Third Edition)*. 2010.