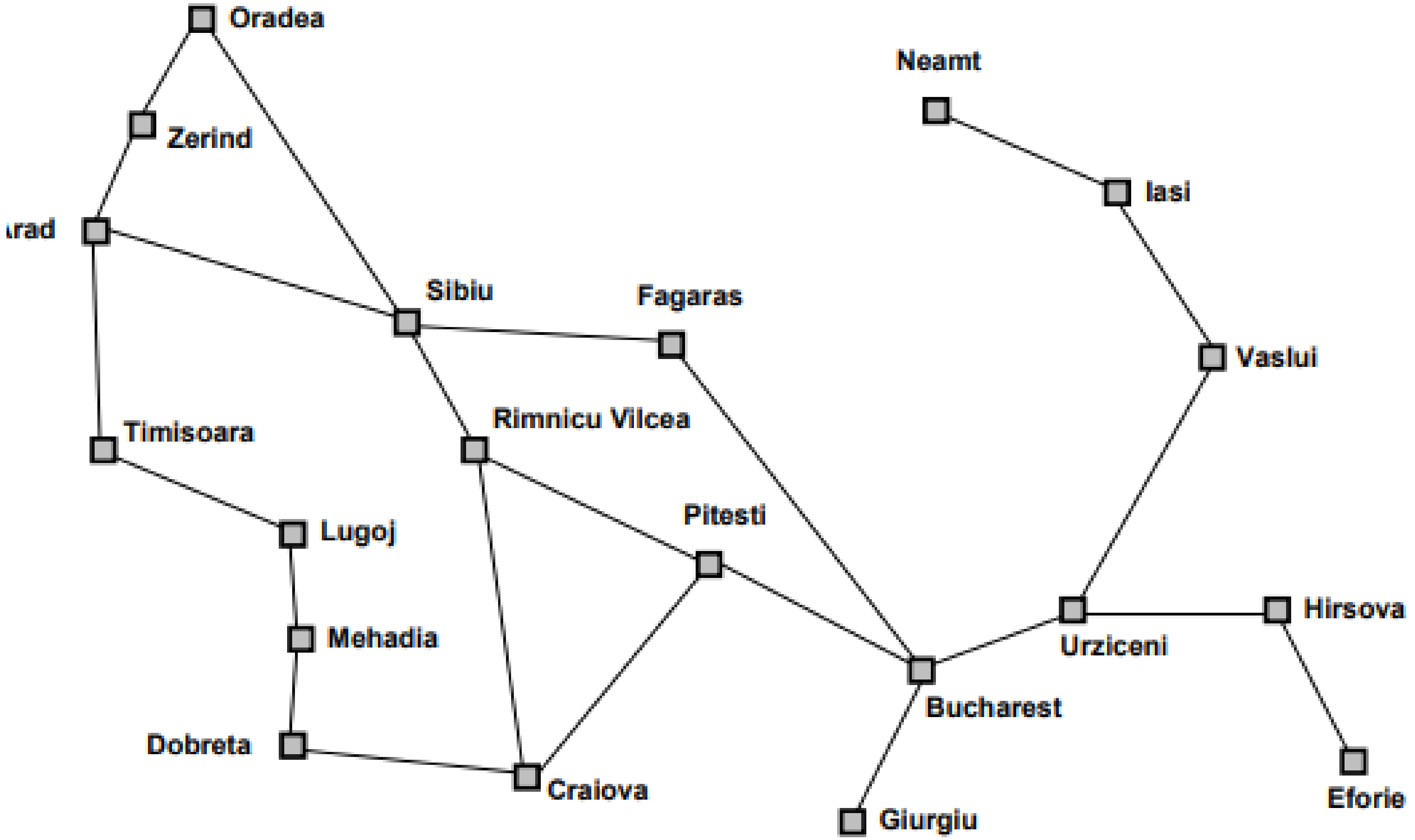
THE UNIVERSITY of TENNESSEE KNOXVILLE

Lecture 22: Heuristics in multistep decisions

Instructor: Sergei V. Kalinin

Route finding

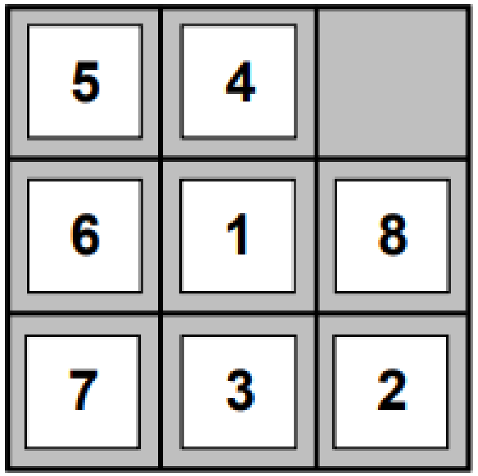


- Initial state: Arad
- Goal state: Bucharest
- Path cost: Number of intermediate cities, distance traveled, expected travel tin

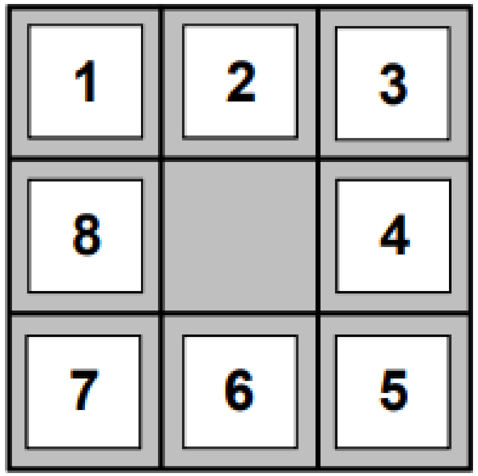
From agents to search

- Constraints and constraint-satisfaction problems

- 8-puzzle (sliding tile puzzle)

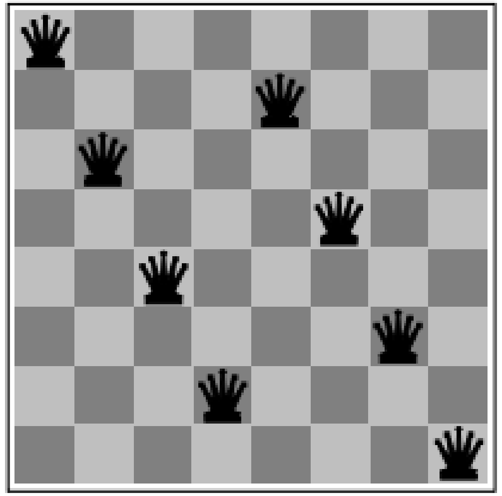


Start State



Goal State

- 8-queens problem (N-queens problem)



More realistic problems:

- Route finding (Google Maps)
- Travelling salesman problem
- Supply chain planning
- Integrated circuit design
- Experiment planning

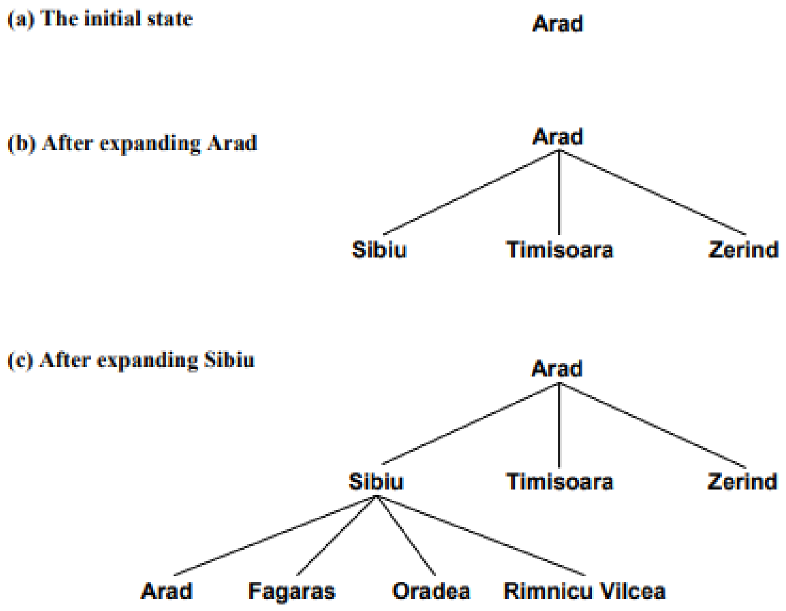
Search concepts

- A state can be expanded by generating all states that can be reached by applying a legal operator to the state

- State space can also be defined by a successor function that returns all states produced by applying a single legal operator

- A search tree is generated by generating search nodes by successively expanding states starting from the initial state as the root

- A search node in the tree can contain
    - Corresponding state
    - Parent node
    - Operator applied to reach this node
    - Length of path from root to node (depth)
    - Path cost of path from initial state to node



Search strategies

Properties of search strategies

- Completeness: does it always find solution if one exists?
- Time Complexity: number of nodes generated/expanded
- Space Complexity: maximum number of nodes in memory
- Optimality: does it always find least cost solution?

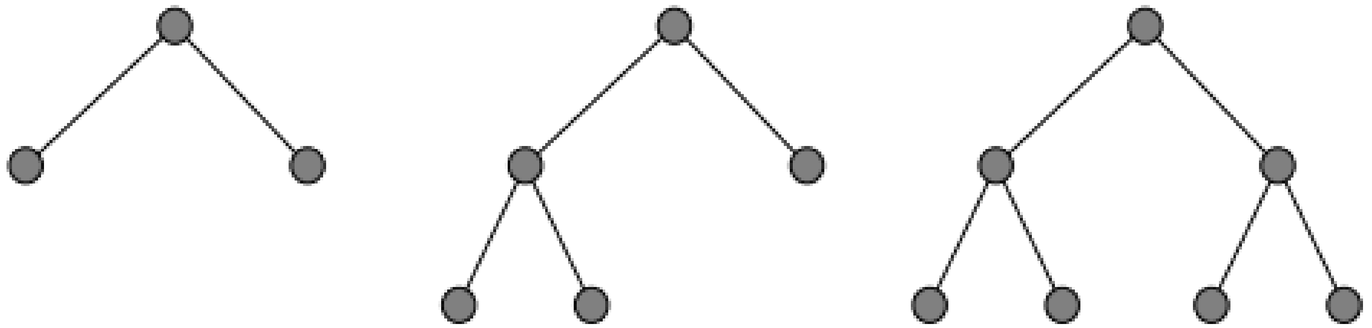
Informed vs. uninformed:

• Uninformed search strategies (blind, exhaustive, brute force) do not guide the search with any additional information about the problem.

• Informed search strategies (heuristic, intelligent) use information about the problem (estimated distance from a state to the goal) to guide the search

Breadth-first search

- Expands search nodes level by level, all nodes at level *d* are expanded before expanding nodes at level *d*+1



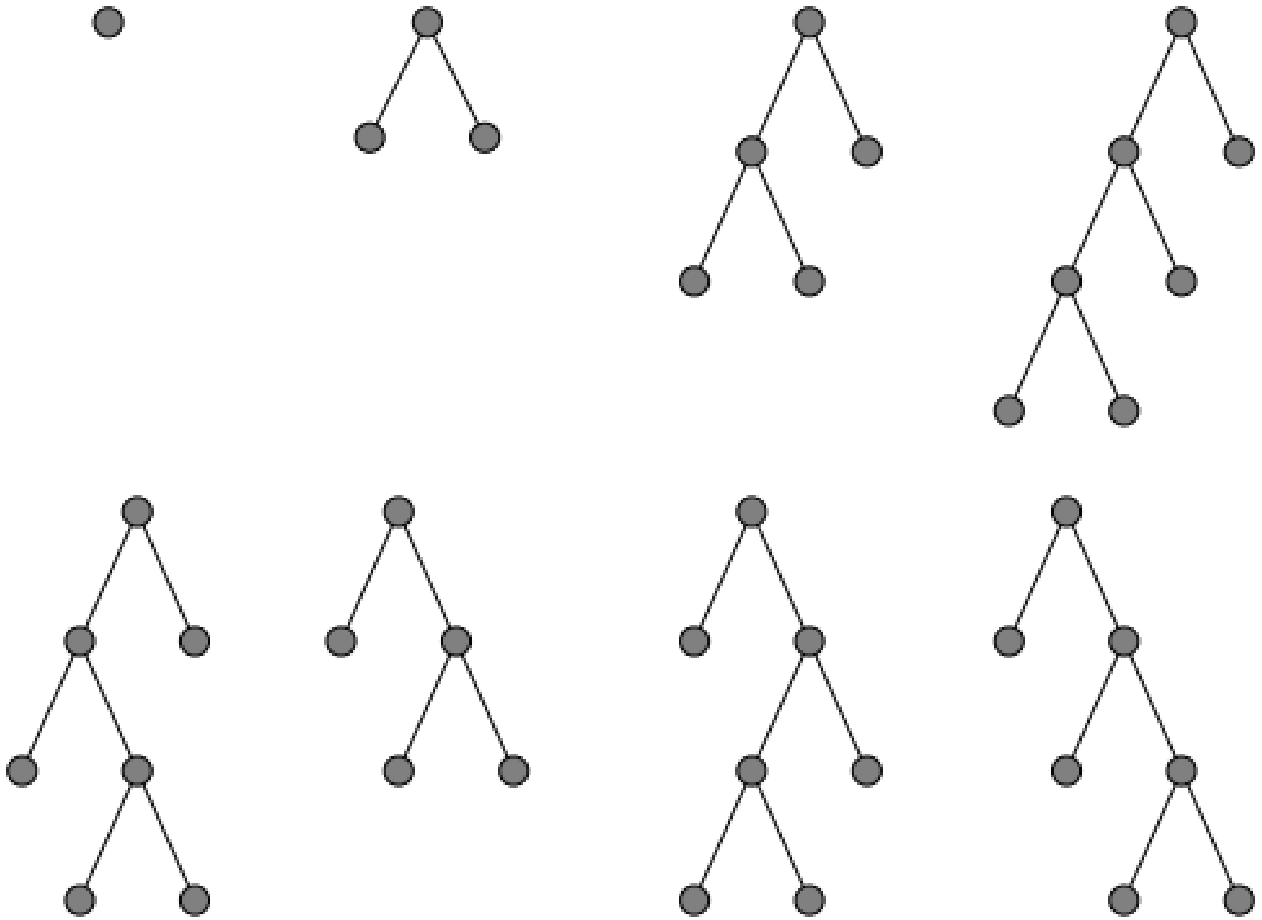
- Implemented by adding new nodes to the end of the queue

- Since eventually visits every node to a given depth, guaranteed to be complete

- Optimal provided path cost is a nondecreasing function of the depth of the node (e.g. all operators of equal cost) since nodes explored in depth order

Depth-first search

- Always expand node at deepest level of the tree, i.e. one of the most recently generated nodes. When hit a dead-end, backtrack to last choice



Heuristic Search

- Heuristic or informed search exploits additional knowledge about the problem that helps direct search to more promising paths.

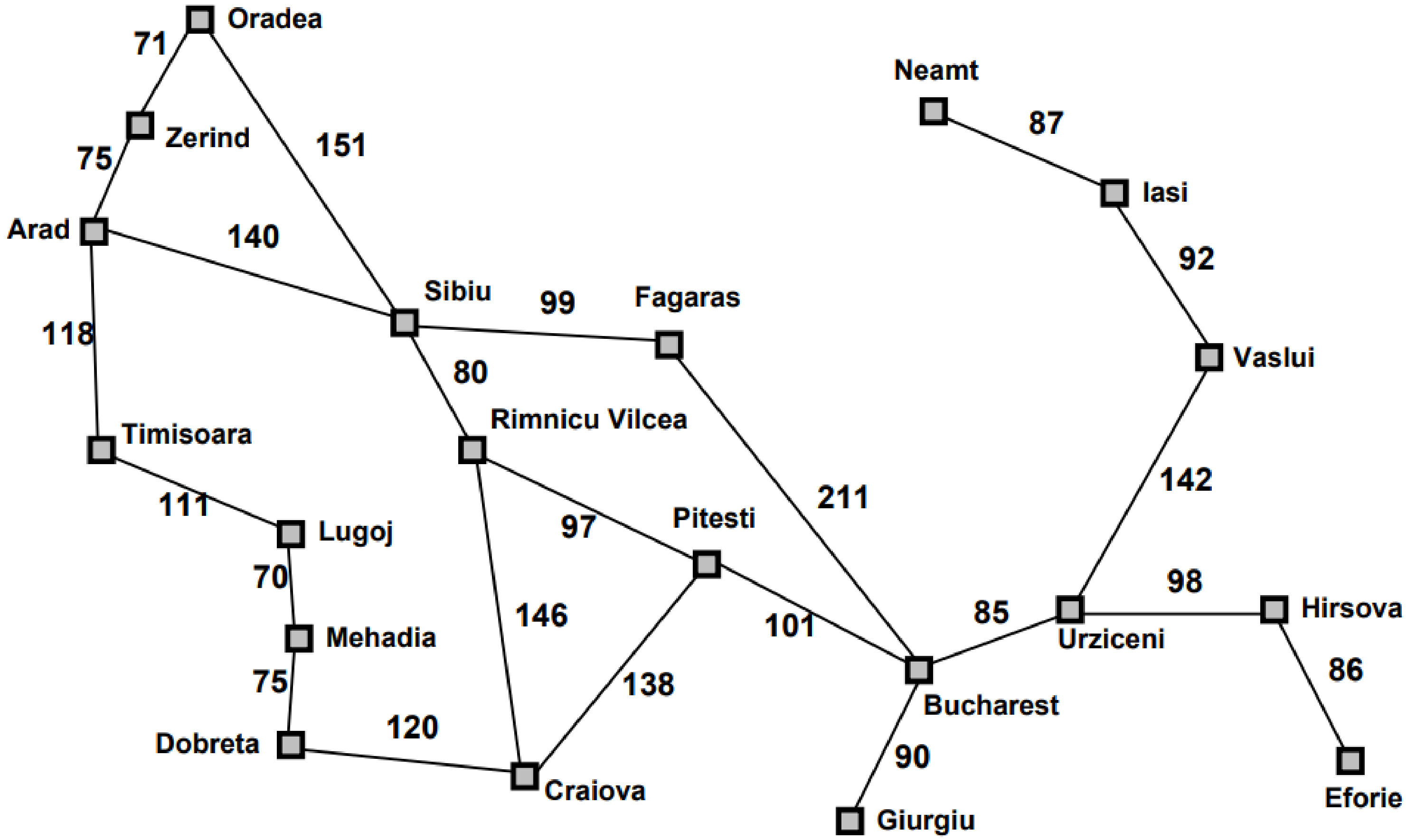
- A **heuristic function**, $h(n)$, provides an estimate of the cost of the path from a given node to the closest goal state

- Must be zero if node represents a goal state.
  - Example: Straight-line distance from current location to the goal location in a road navigation problem

- Many search problems are NP-complete so in the worst case still have exponential time complexity; however a good heuristic can:
  - Find a solution for an average problem efficiently.
  - Find a reasonably good but not optimal solution efficiently.

Best first search

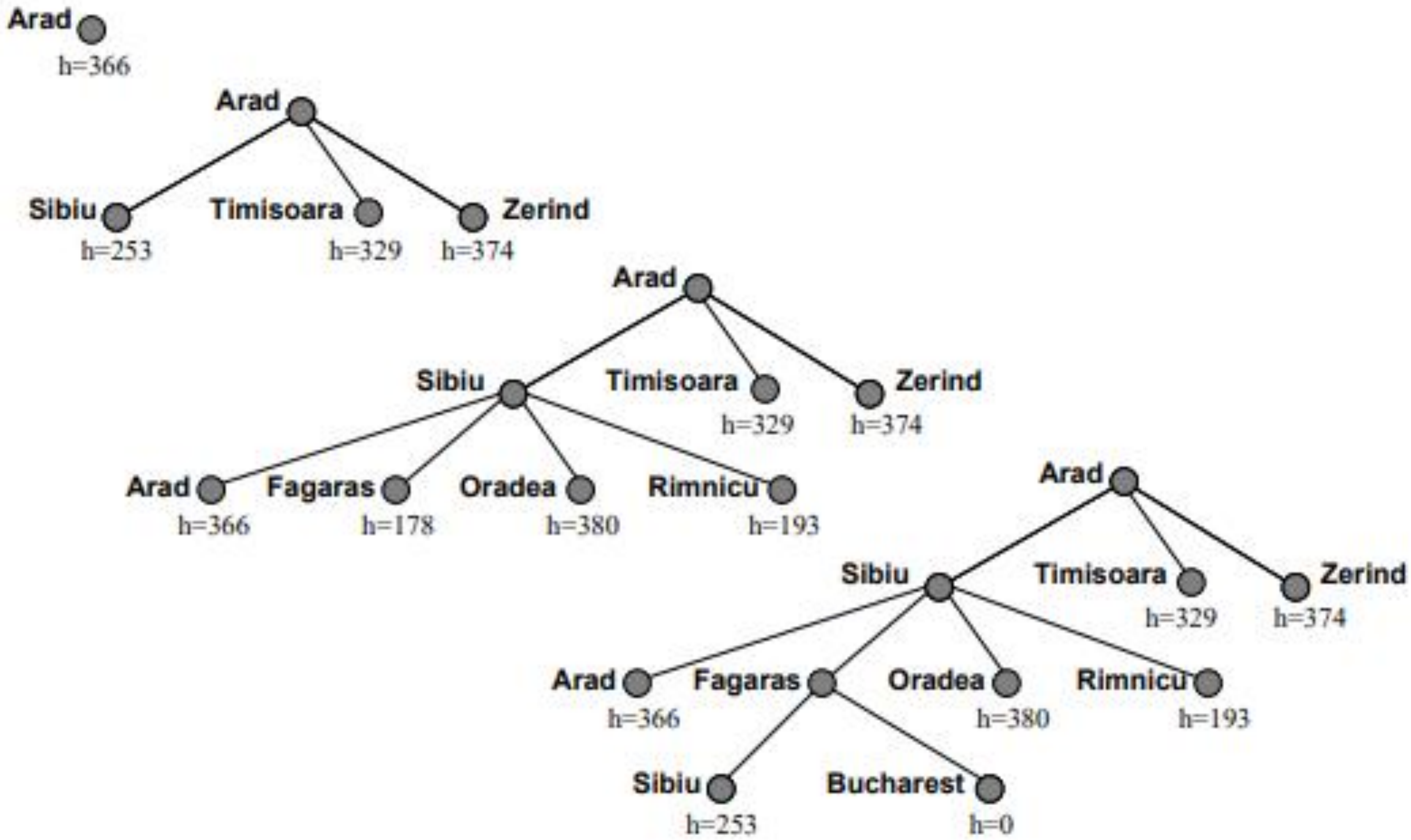
- At each step, best-first search sorts the queue according to a heuristic function



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Best first search



- Does not find shortest path to goal (through Rimnicu) since it is only focused on the cost remaining rather than the total cost

Best first search properties

- Not complete, may follow infinite path if heuristic rates each state on such a path as the best option. Most reasonable heuristics will not cause this problem however

- Worst case time complexity is still $O(b^m)$ where m is the maximum depth.

- Since must maintain a queue of all unexpanded states, space-complexity is also $O(b^m)$

- However, a good heuristic will avoid this worst-case behavior for most problems.

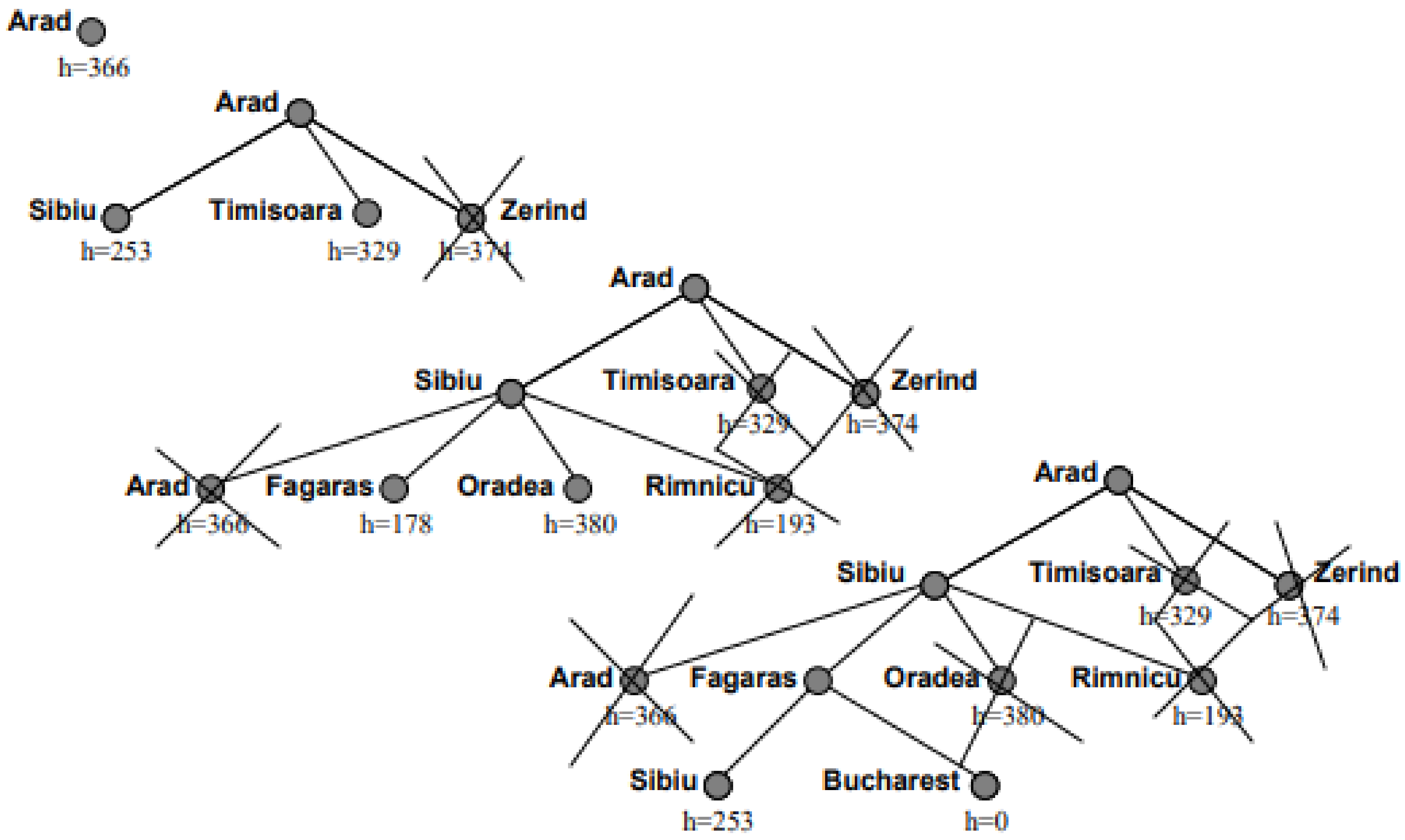
Beam search

- Space and time complexity of storing and sorting the complete queue can be too inefficient

- Beam search trims queue to the best *n* options (*n* is called the beam width) at each point

- Focuses search more but may eliminate solution even for finite search graphs

Example for n=2



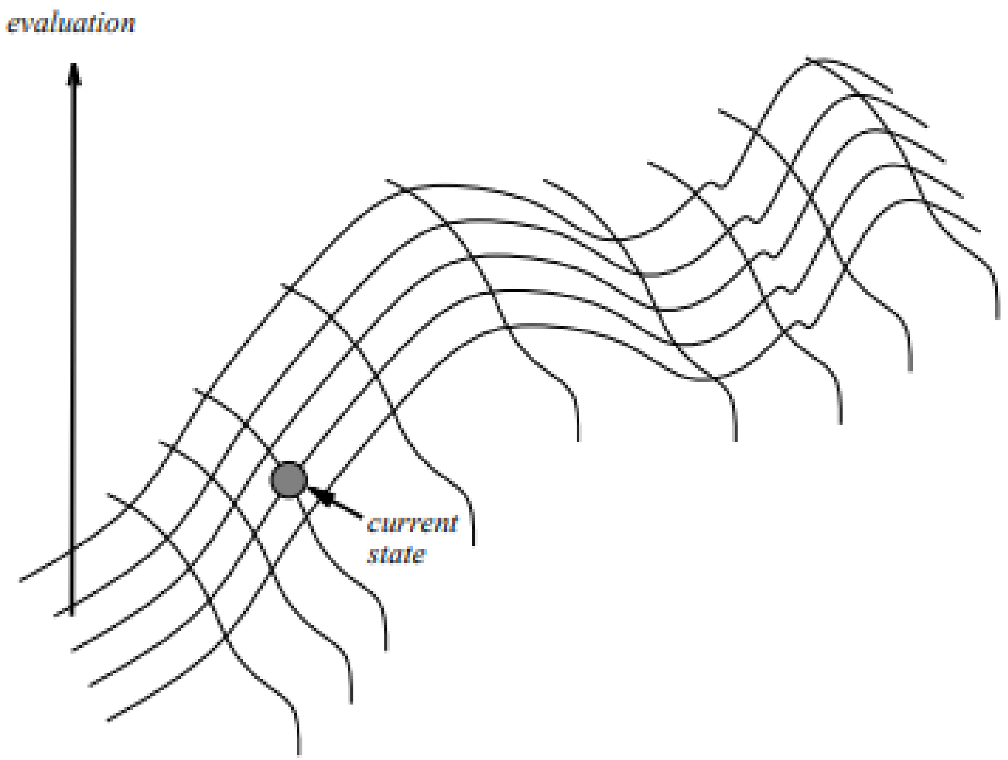
AIMA Slides ©Stuart Russell and Peter Norvig, 1998

Hill climbing

- Beam search with a beam-width of 1 is called hill climbing

- Pursues locally best option at each point, i.e. the best successor to the current best node.

- Subject to local maxima, plateaux, and ridges.



AIMA Slides ©Stuart Russell and Peter Norvig, 1998

Minimizing total cost: A^* Search

- A^* combines features of uniform cost search (complete, optimal, inefficient) with best-first (incomplete, non-optimal, efficient).
- Sort queue by estimated total cost of the completion of a path:

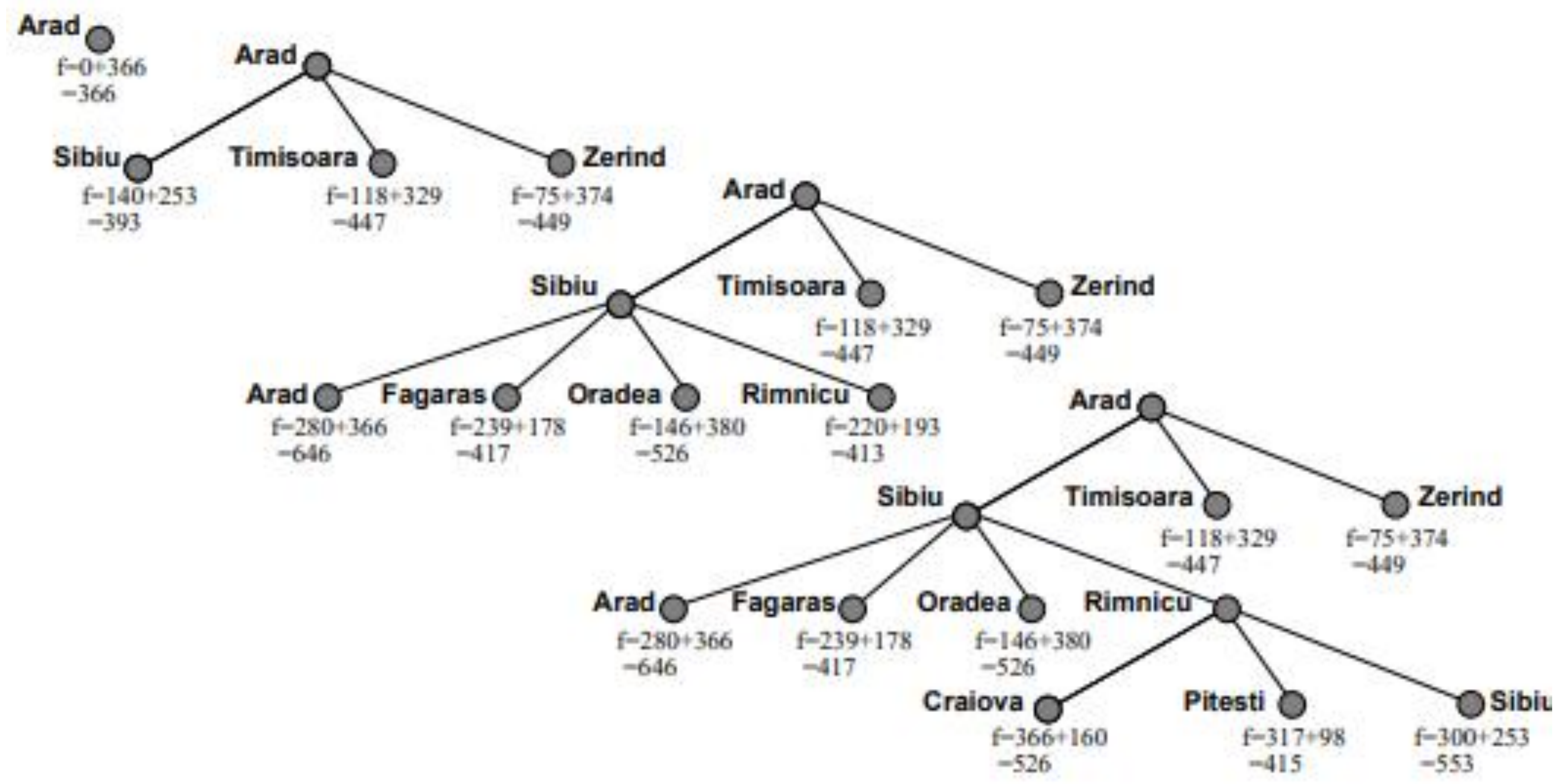
$$f(n) = g(n) + h(n)$$

Cost to this moment   Estimated cost to the goal

- If the heuristic function always underestimates the distance to the goal, it is said to be admissible.
- If h is admissible, then $f(n)$ never overestimates the actual cost of the best solution through n

A* Search Example

AIMA Slides ©Stuart Russell and Peter Norvig, 1998

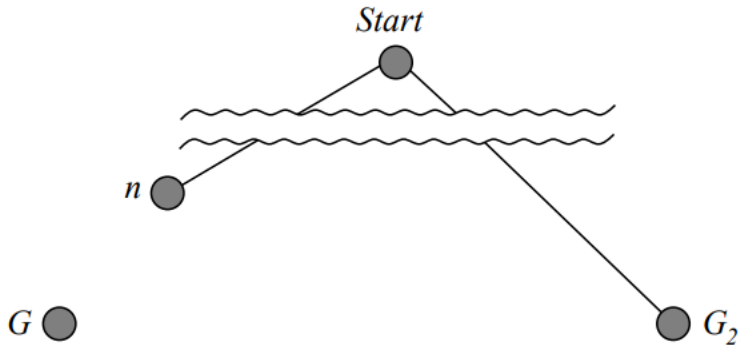


- Finds the optimal path to Bucharest through Rimnicu and Pitesti

Optimality of A*

If h is admissible, A* will always find a least cost path to the goal

- Proof by contradiction:
  - Let *G* be an optimal goal state with a path cost f*
  - Let G_2 be a suboptimal goal state supposedly found by A*
  - Let *n* be a current leaf node on an optimal path to *G*



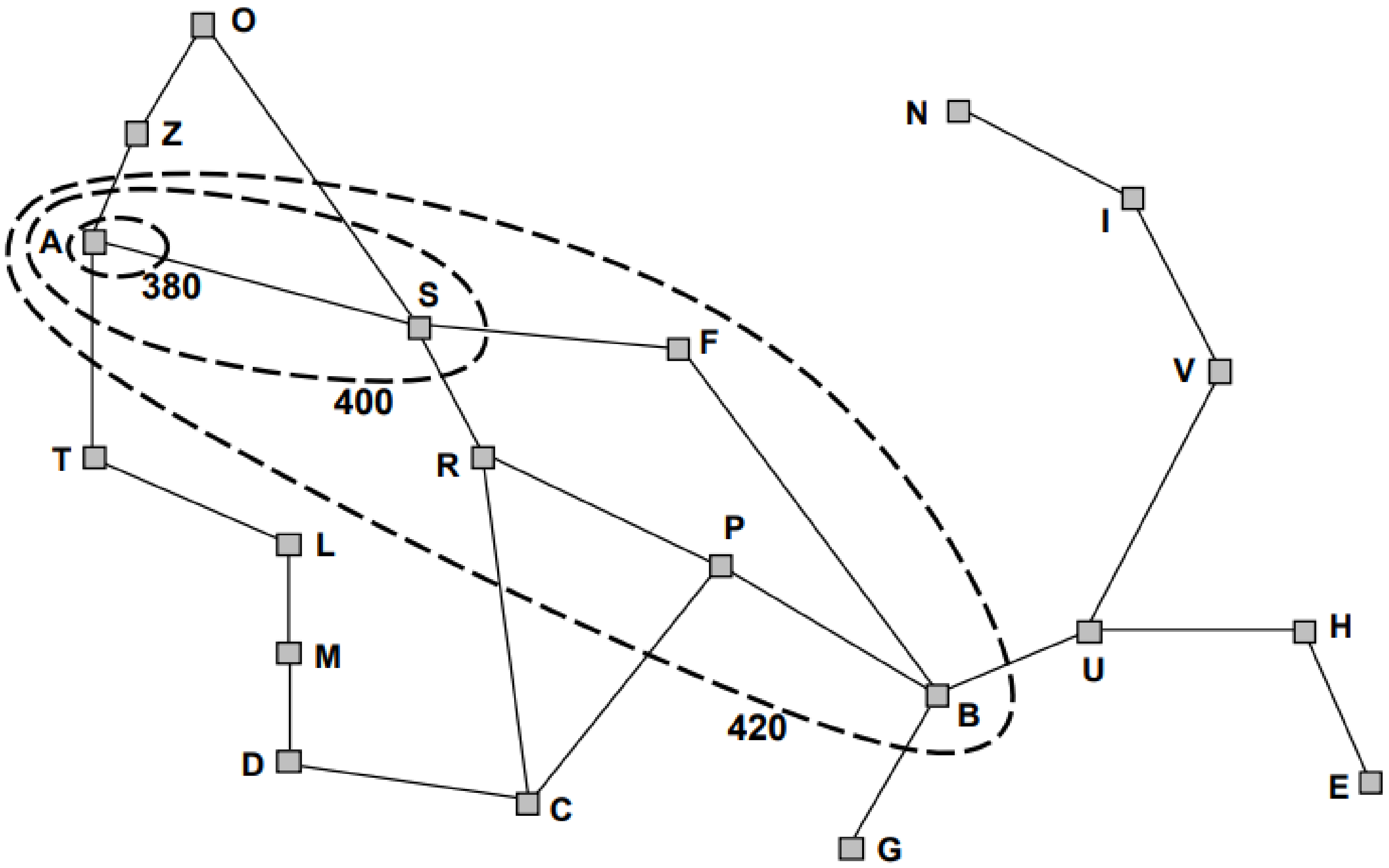
- Since h is admissible: f* >= f(*n*)
- If G2 is chosen for expansion over n then: f(*n*) >= f(G_2)
- Therefore: f* >= f(G_2)

- Since G_2 is a goal state, h(G_2)=0, therefore f(G2) = g(G_2) and f* >= g(G_2)

- Therefore G_2 is optimal. Contradiction.

Lemma: A* expands nodes in order of increasing f value

Gradually adds "f-contours" of nodes (cf. breadth-first adds layers)
Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



Other properties of A*

- A* is complete as long as
  - Branching factor is always finite
  - Every operator adds cost at least $\delta > 0$

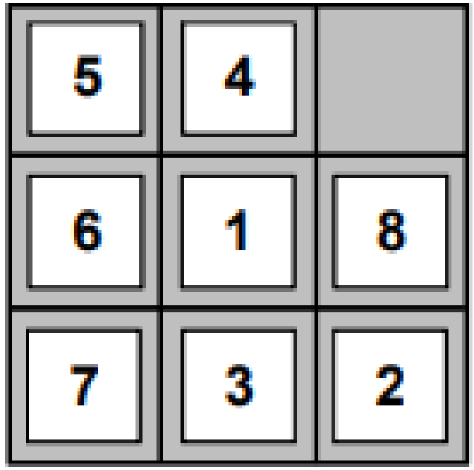
- Time and space complexity still $O(b^m)$ in the worst case, since the search must maintain and sort complete queue of unexplored options.

- However, with a good heuristic can find optimal solutions for many problems in reasonable time

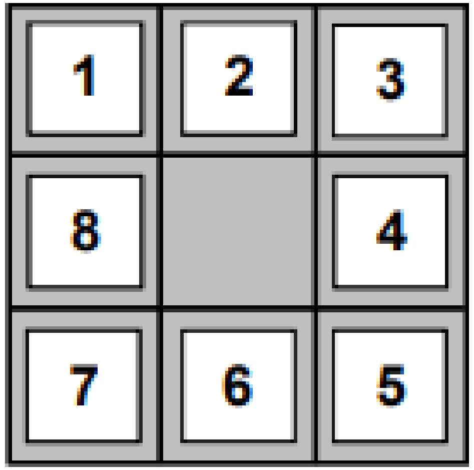
- Again, space complexity is a worse problem than time.

But what about heuristic functions?

- 8-puzzle search space
  - Typical solution length: 20 steps
  - Average branching factor: 3
  - Exhaustive search: $3^{20}=3.5 \times 10^9$
  - Bound on unique states: $9! = 362{,}880$



Start State



Goal State

- Admissible Heuristics:
  - Number of tiles out of place (h1): 7
  - City-block (Manhattan) distance (h2): 2+3+3+2+4+2+0+2=18

Using heuristics:

- Experiments on sample problems can determine the number of nodes searched and CPU time for different strategies

- One other useful measure is effective branching factor: If a method expands N nodes to find solution of depth d, and a uniform tree of depth d would require a branching factor of b* to contain N nodes, the effective branching factor is b*, so that N = 1 + b* + (b*)2 + ...+ (b*)d

Experimental Results on 8-puzzle problems

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Quality of heuristics

- Since A* expands all nodes whose f value is less than that of an optimal solution, it is always better to use a heuristic with a higher value as long as it does not over-estimate.

- Therefore h_2 is uniformly better than h_1 or h_2 dominates h_1

- A heuristic should also be easy to compute, otherwise the overhead of computing the heuristic could outweigh the time saved by reducing search (e.g. using full breadth-first search to estimate distance wouldn't help).

Inventing Heuristics

- Many good heuristics can be invented by considering relaxed versions of the problem (abstractions)

- **For 8-puzzle:** A tile can move from square A to B if A is adjacent to B and B is blank
  - (a) A tile can move from square A to B if A is adjacent to B
  - (b) A tile can move from square A to B if B is blank
  - (c) A tile can move from square A to B

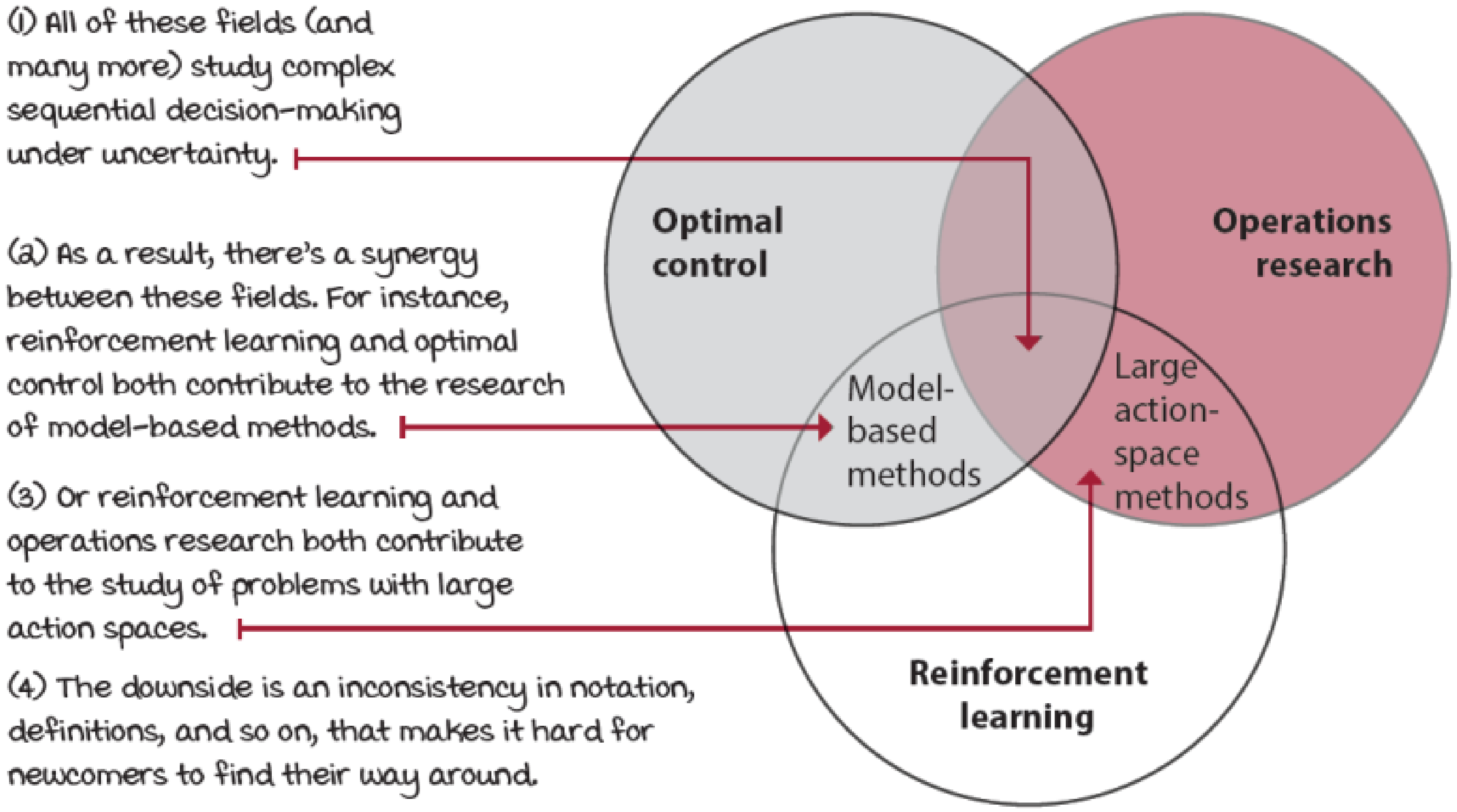
- If there are a number of features that indicate a promising or unpromising state, a weighted sum of these features can be useful. Learning methods can be used to set weights.

(A Peek at) Reinforcement Learning

- **Supervised learning**
  - Classification
  - Regression
- **Unsupervised learning**
  - Clustering
  - Dimensionality reduction
- **Reinforcement learning**
  - more general than supervised/unsupervised learning
  - learn from interaction with environment to achieve a goal

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

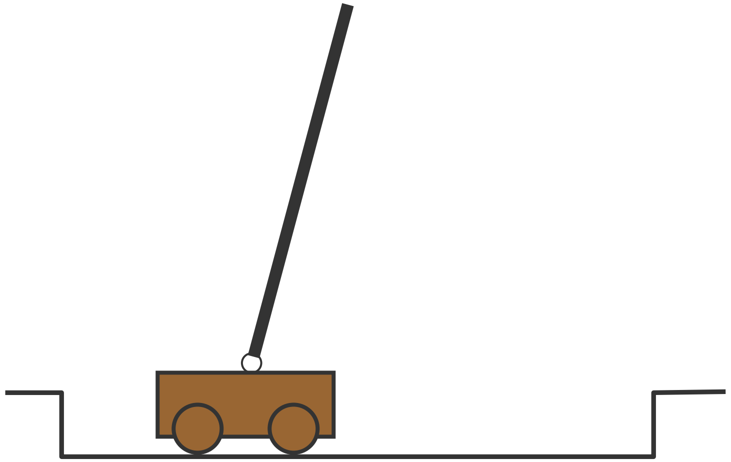
Where is reinforcement learning in ML world?



https://livebook.manning.com/book/grokking-deep-reinforcement-learning/chapter-1/31

Examples

- Frozen lake
- Pole-balancing
- TD-Gammon [Gerry Tesauro]
- Helicopter [Andrew Ng]



- no teacher who would say “good” or “bad”
  – is reward “10” good or bad?
  – rewards could be delayed

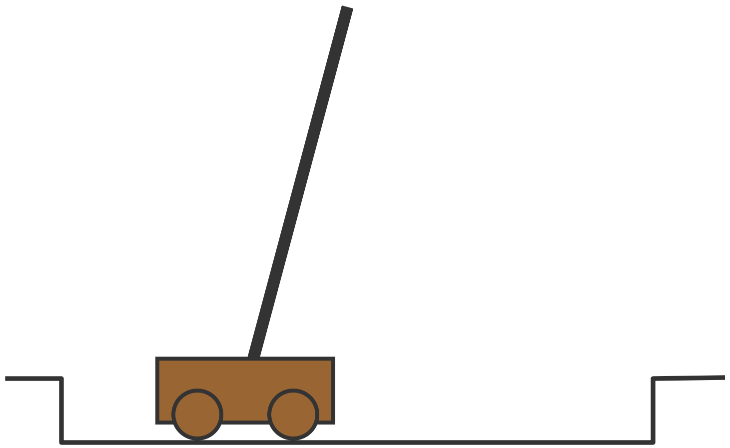
- similar to control theory
  – more general, fewer constraints

- explore the environment and learn from experience
  – not just blind search, try to be smart about it

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

State Representation

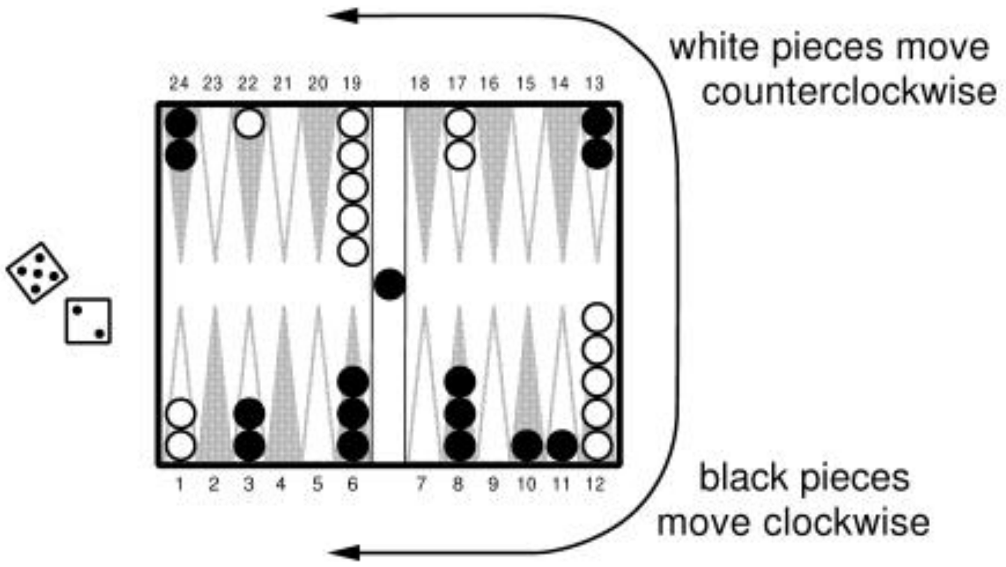
- pole-balancing
  – move car left/right to keep the pole balanced
- state representation
  – position and velocity of car
  – angle and angular velocity of pole
- what about *Markov property*?
  – would need more info
  – noise in sensors, temperature, bending of pole
- solution
  – coarse discretization of 4 state variables
    - left, center, right
  – totally non-Markov, but still works



From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

Backgammon

- rules
  - 30 pieces, 24 locations
  - roll 2, 5: move 2, 5
  - hitting, blocking
  - branching factor: 400
- implementation
  - use TD(λ) and neural nets
  - 4 binary features for each positio
  - no BG expert knowledge
- results
  - TD-Gammon 0.0: trained against itself (300,000 games)
    - as good as best previous BG computer program (also by Tesauro)
    - lot of expert input, hand-crafted features
  - TD-Gammon 1.0: add special features
  - TD-Gammon 2 and 3 (2-ply and 3-ply search)
    - 1.5M games, beat human champion



From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

Designing Rewards

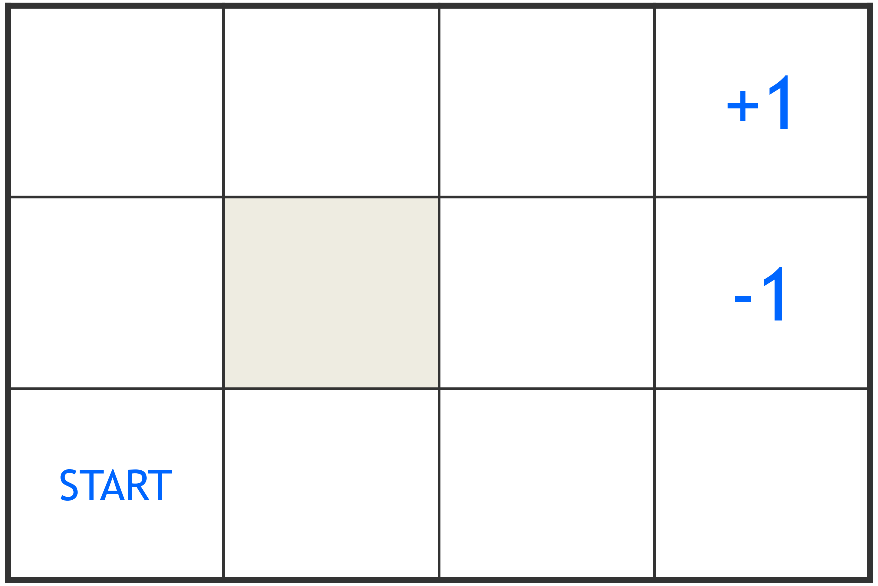
- robot in a maze
  – episodic task, not discounted, +1 when out, 0 for each step

- chess
  – GOOD: +1 for winning, -1 losing
  – BAD: +0.25 for taking opponent’s pieces
    • high reward even when lose

- rewards
  – rewards indicate what we want to accomplish
  – NOT how we want to accomplish it

- shaping
  – positive reward often very “far away”
  – rewards for achieving subgoals (domain knowledge)
  – also: adjust initial policy or initial value function

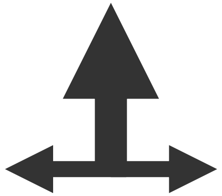
Robot in the room



actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT

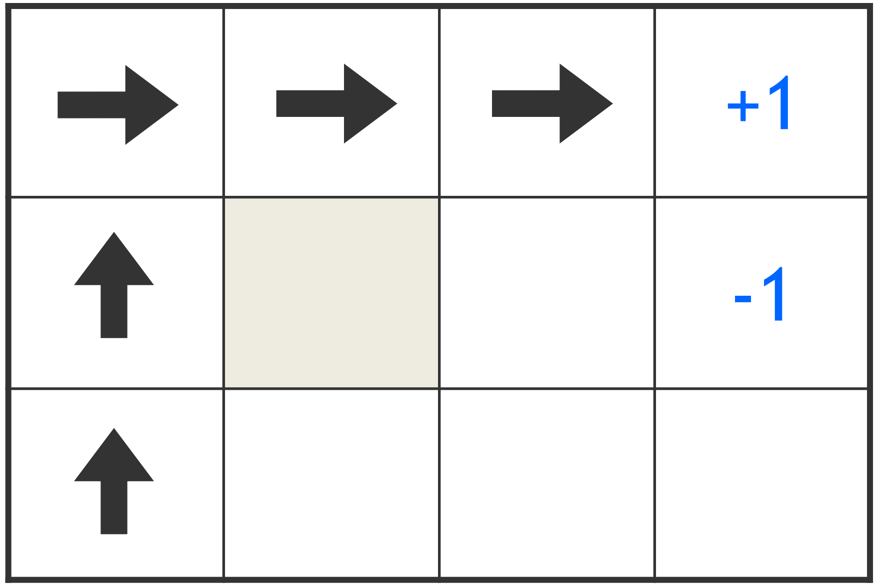


- Reward +1 at [4,3], -1 at [4,2]
- Reward -0.04 for each step

- What's the strategy to achieve max reward?
- What if the actions were deterministic?

From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

Is this a solution?

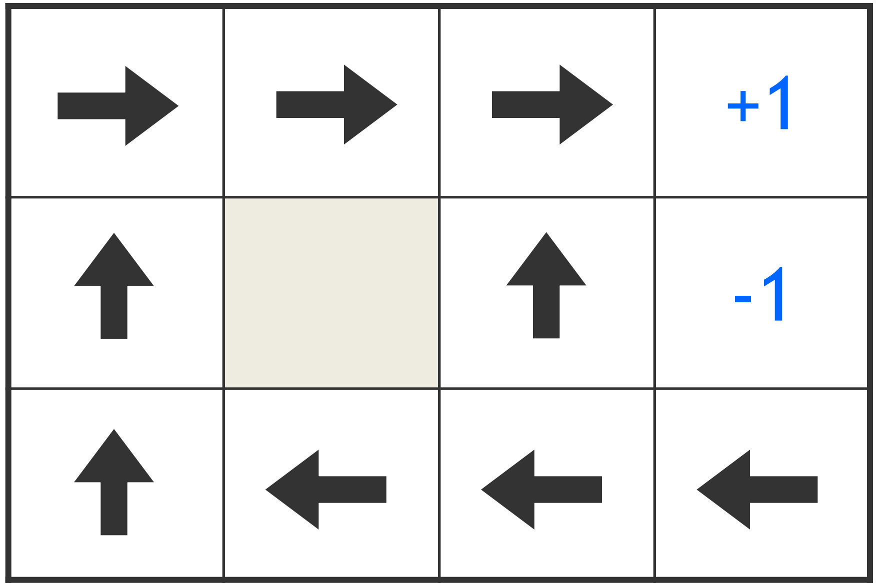


- only if actions deterministic
- not in this case (actions are stochastic)

- solution/policy
- mapping from each state to an action

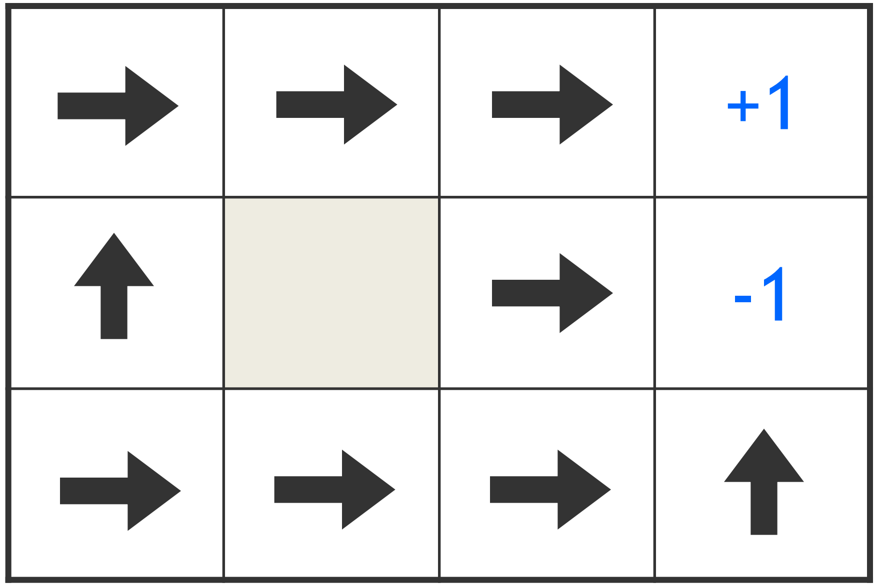
From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

Optimal policy



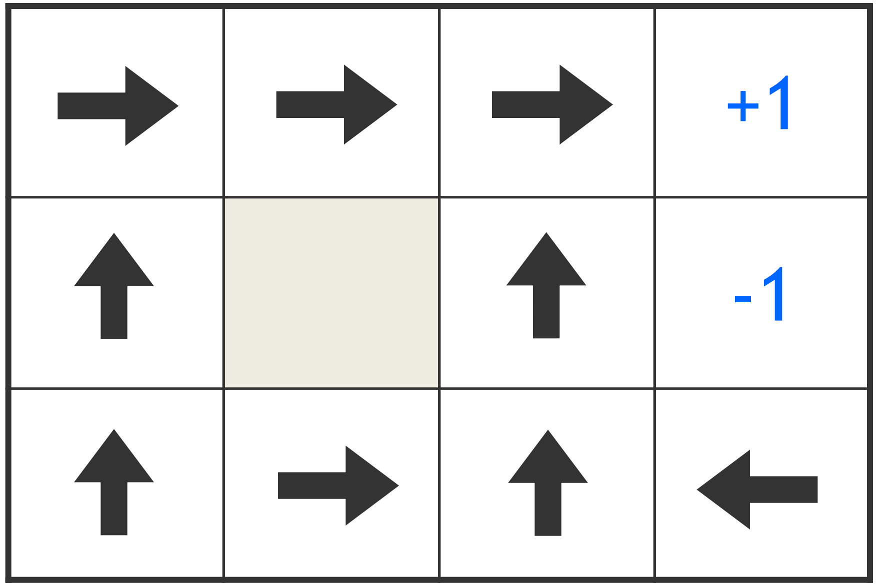
From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

What if the reward for each step is -2?



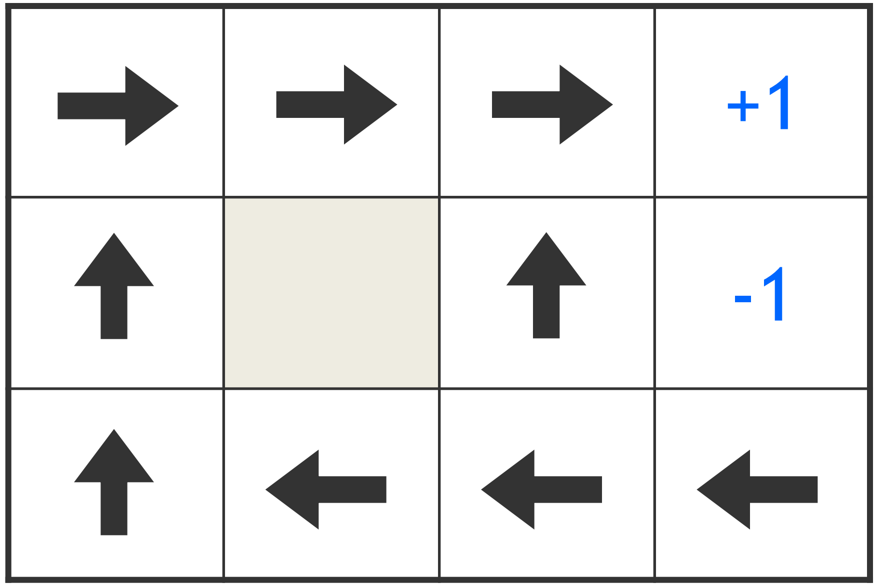
From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

Reward for each step is -0.1



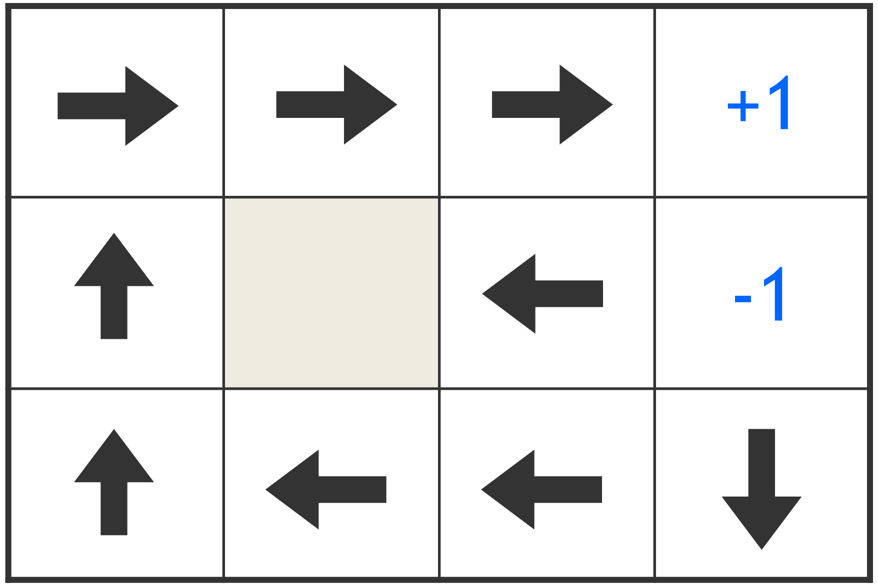
From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

Reward for each step is -0.04



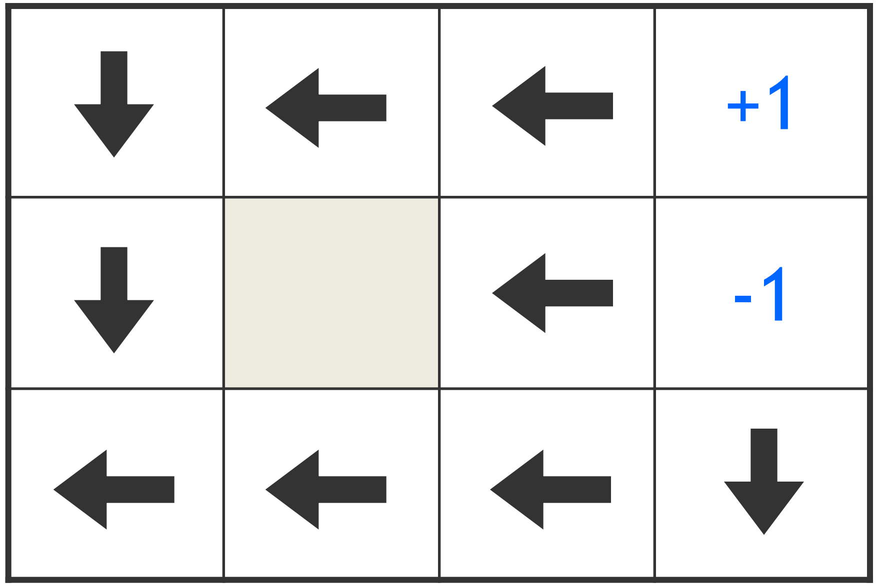
From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

Reward for each step is -0.01



From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley

Reward for each step is +0.01



From Reinforcement Learning Tutorial, Peter Bodík, RAD Lab, UC Berkeley