CSC384 Summer 2018

Week 3 - Informed Search

Ilir Dema

University of Toronto

May 14, 2018

University of Toronto Mississauga

Overview

Heuristic Search

- In uninformed search, we don't try to evaluate which of the nodes on OPEN are most promising. We never "look-ahead" to the goal.

  E.g., in uniform cost search we always expand the cheapest path. We don't consider the cost of getting to the goal from the end of the current path.
- Often we have some other knowledge about the merit of nodes, e.g., going the wrong direction in Romania.

Heuristic Search

Merit of an OPEN node: different notions of merit.

- If we are concerned about the cost of the solution, we might want to consider how costly it is to get to the goal from the end state of that node.
- If we are concerned about minimizing computation in search we might want to consider how easy it is to find the goal from the end state of that node.
- We will focus on the “cost of solution” notion of merit.

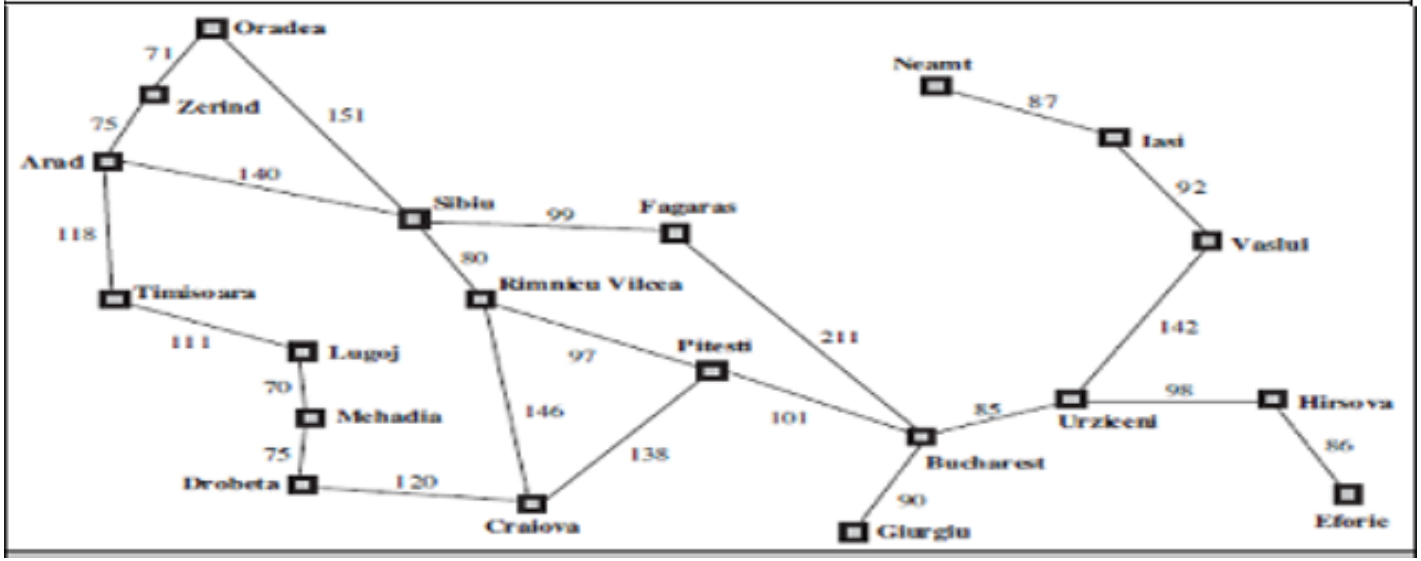
Heuristic Search

- The idea is to develop a domain specific heuristic function h(n).
- h(n) guesses the cost of getting to the goal from node n (i.e., from the terminal state of the path represented by n).
- There are different ways of guessing this cost in different domains.
  - heuristics are domain specific.

EXAMPLE: EUCLIDEAN DISTANCE

Planning a path from Arad to Bucharest, we can utilize the straight line distance from each city to our goal. This lets us plan our trip by picking cities at each time point that minimize the distance to our goal.

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



Heuristic Search

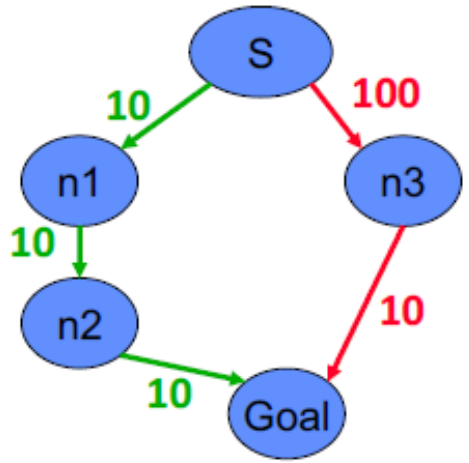
- $h(n_1) < h(n_2)$ this means that we guess that it is cheaper to get to the goal from n_1 than from n_2.
- We require that
  - $h(n) = 0$ for every node n whose terminal state satisfies the goal.
  - Zero cost of achieving the goal from node that already satisfies the goal.

Using only h(n): Greedy best-first search

- We use $h(n)$ to rank the nodes on OPEN

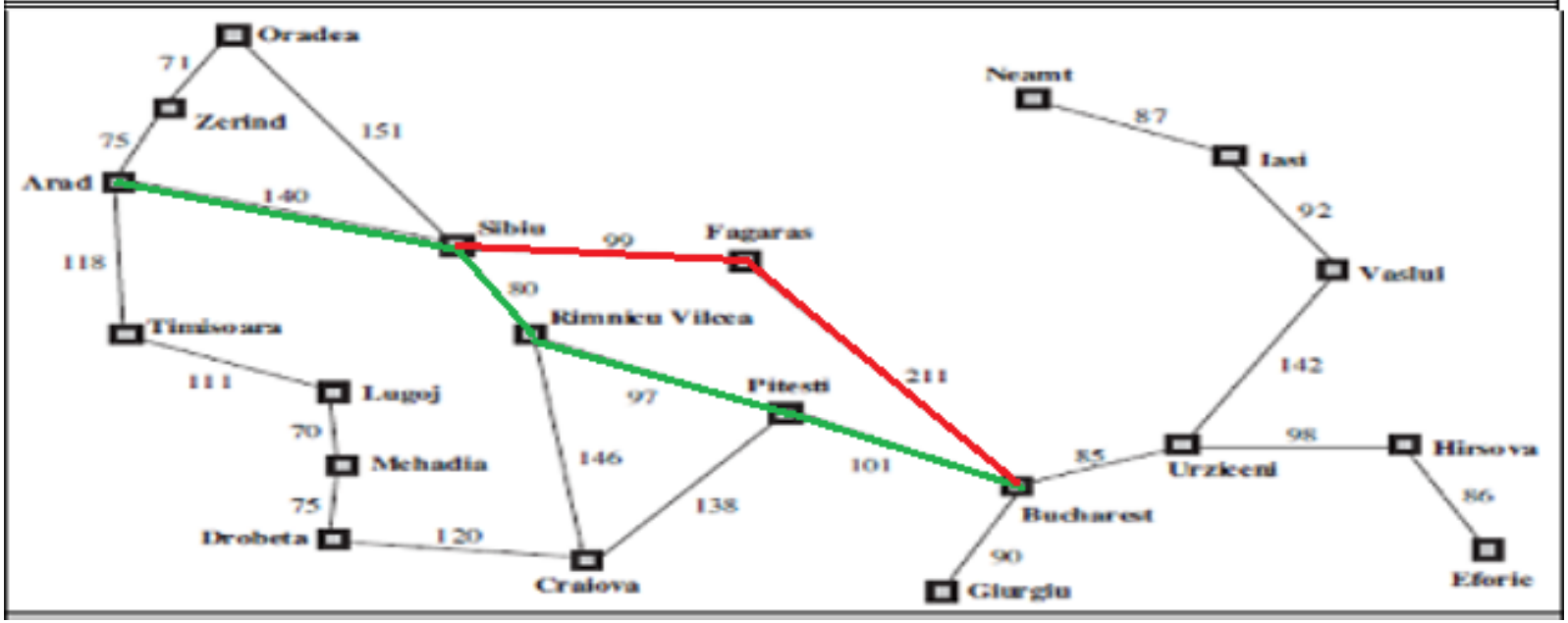
  Always expand node with lowest h-value.
- We are greedily trying to achieve a low cost solution.
- However, this method ignores the cost of getting to n, so it can be lead astray exploring nodes that cost a lot but seem to be close to the goal:

$\rightarrow$ step cost $= 10$
$\rightarrow$ step cost $= 100$
$h(n_1) = 20$
$h(n_3) = 10$



GREEDY BEST-FIRST SEARCH EXAMPLE

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



Greedy best-first search example

Goal: Bucharest

- When you're at Sibiu and contemplating whether to go to Fagaras or RV, the heuristic value of the successor nodes, i.e., the h value guess of the cost is: h(Fagaras) = 178 and h(RV) =193), so Fagaras looks like the better choice, but ...
- Actual Cost(Arad-Sibiu-RV-Pitesli-Bucharest): 140+80+97+101 = 140 + 278 = 418
- Actual Cost (Arad-Sibiu-Fagaras-Bucharest): 140+99+211 = 140 + 310 = 450

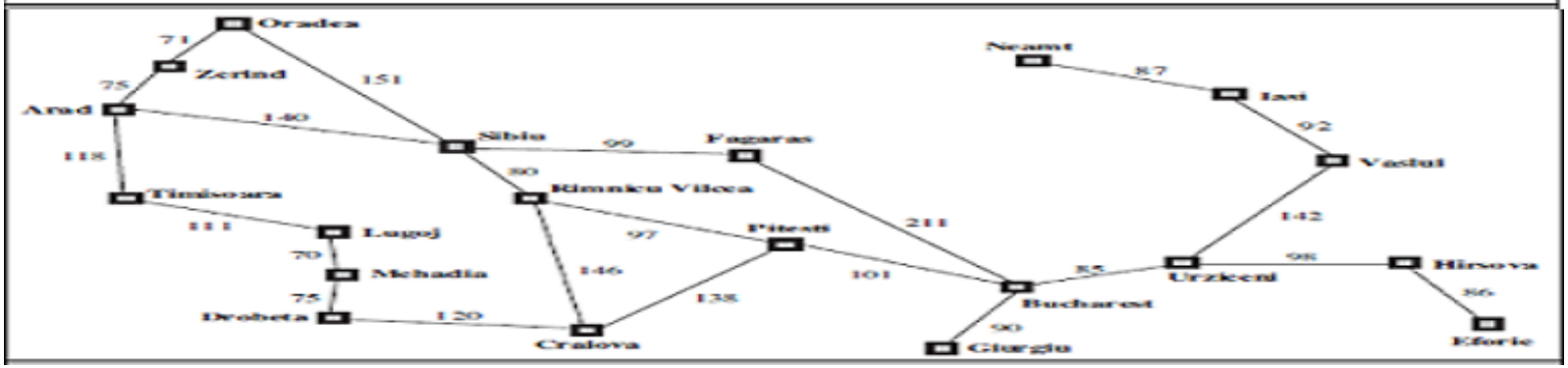
A^* SEARCH

- Take into account the cost of getting to the node as well as our estimate of the cost of getting to the goal from the node.
- Define an evaluation function $f(n)$

  $f(n) = g(n) + h(n)$
  - $g(n)$ is the cost of the path represented by node n
  - $h(n)$ is the heuristic estimate of the cost of achieving the goal from n.
- Always expand the node with lowest f-value on OPEN.
- The f-value, $f(n)$ is an estimate of the cost of getting to the goal via the node (path) n.
  - I.e., we first follow the path n then we try to get to the goal. $f(n)$ estimates the total cost of such a solution.

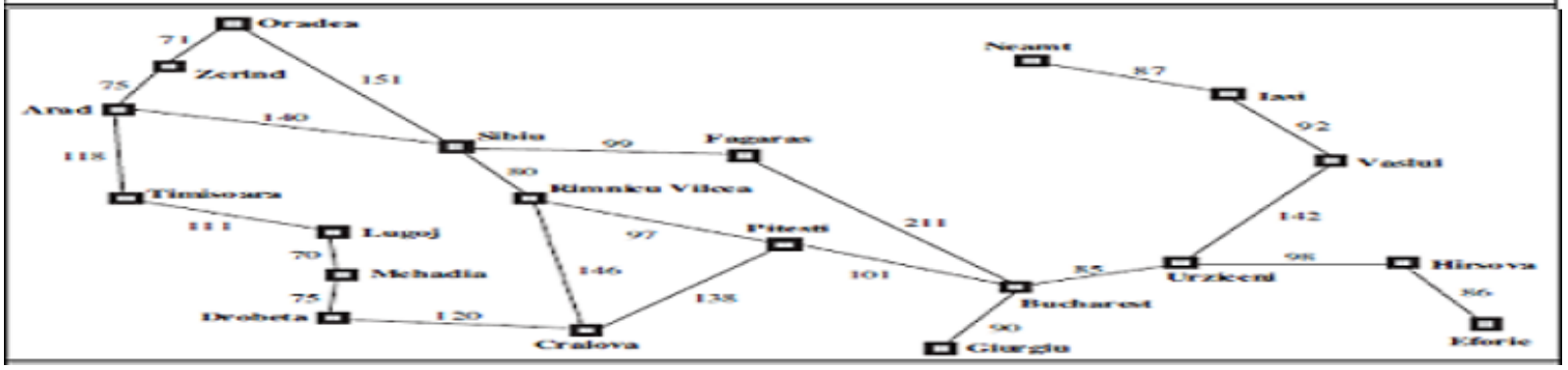
A^* EXAMPLE

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

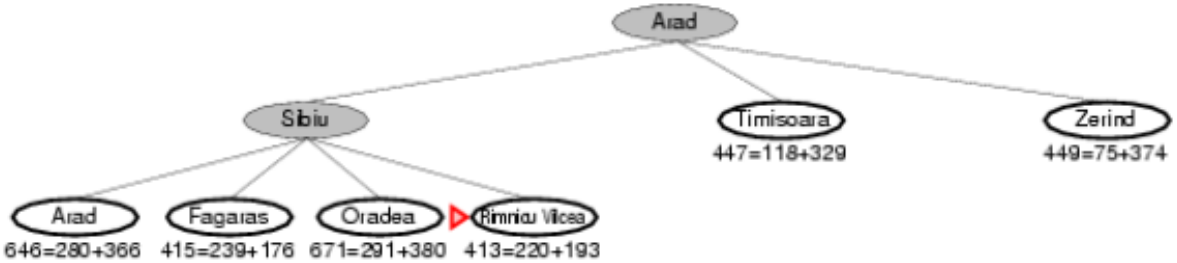


A^* EXAMPLE

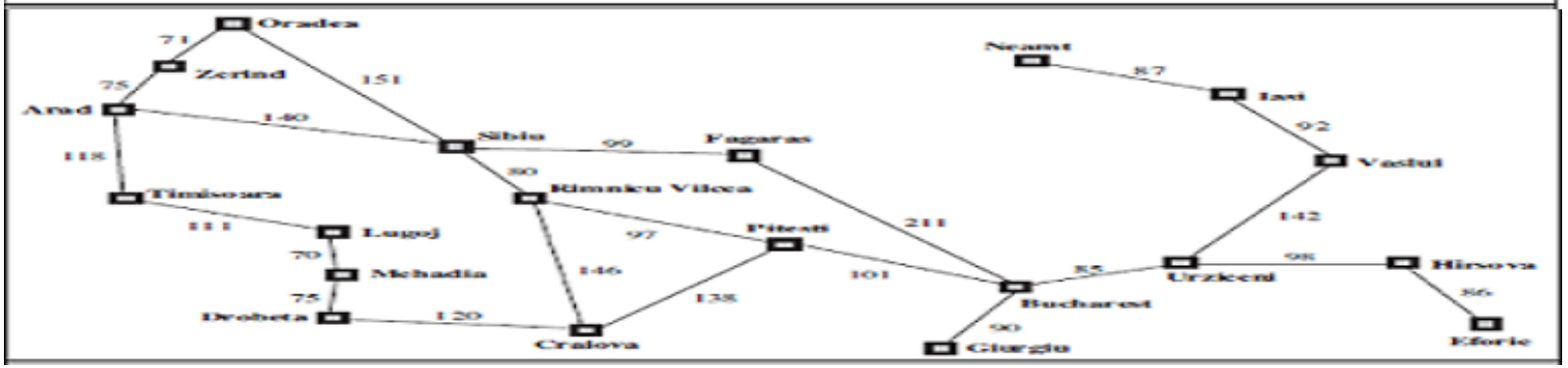
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



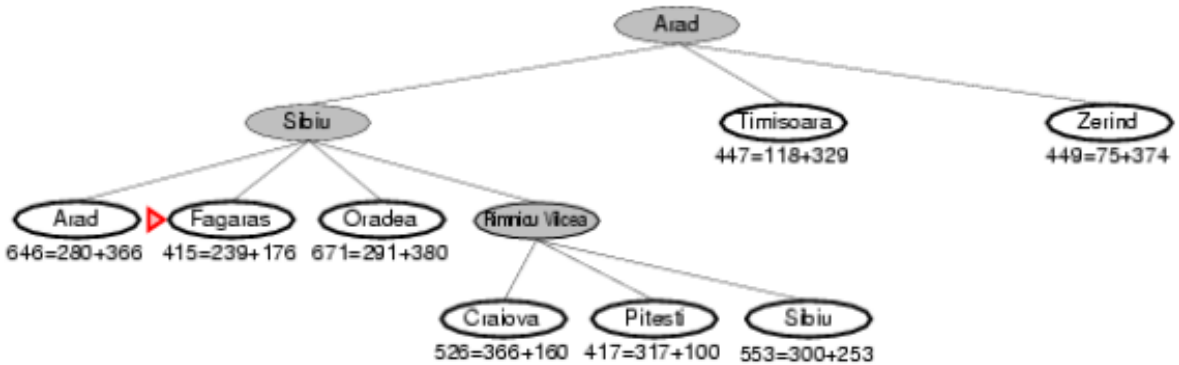
A^* EXAMPLE



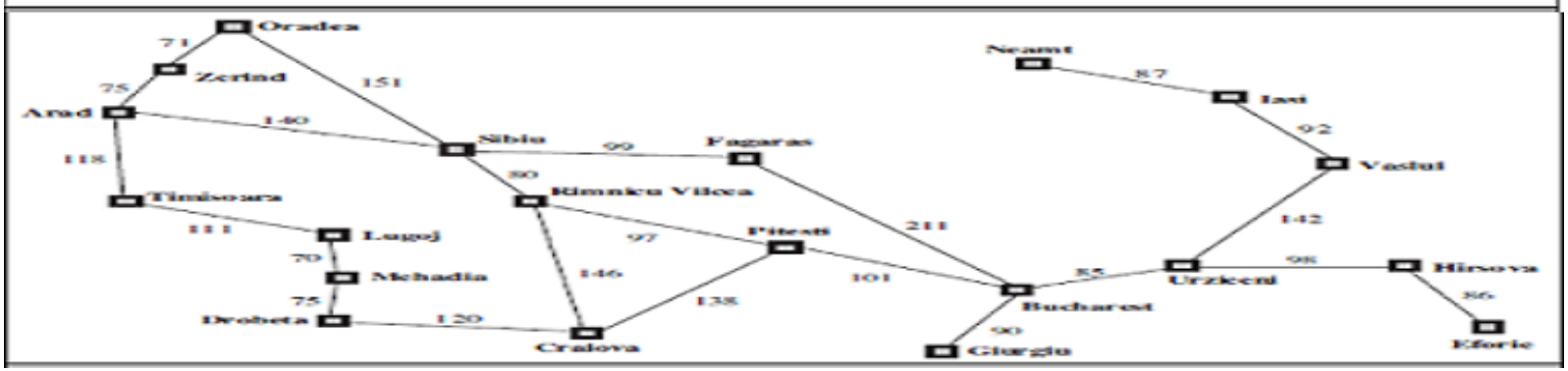
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



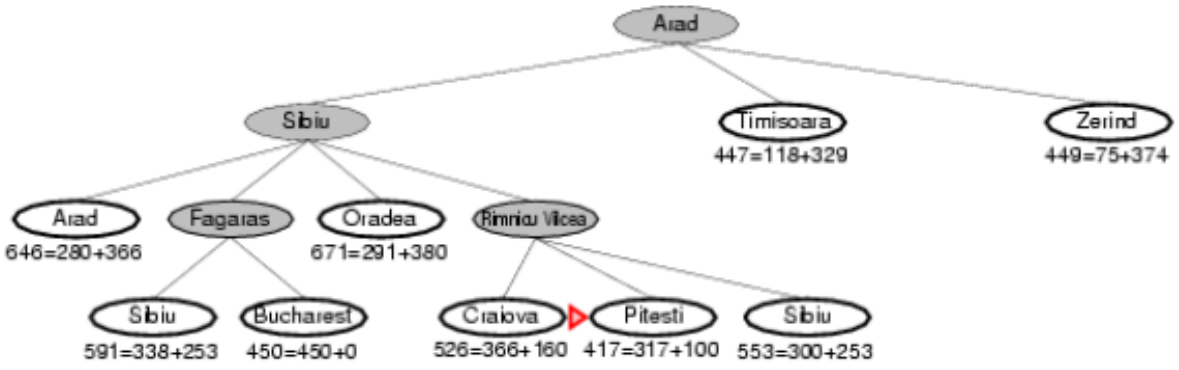
A^* EXAMPLE



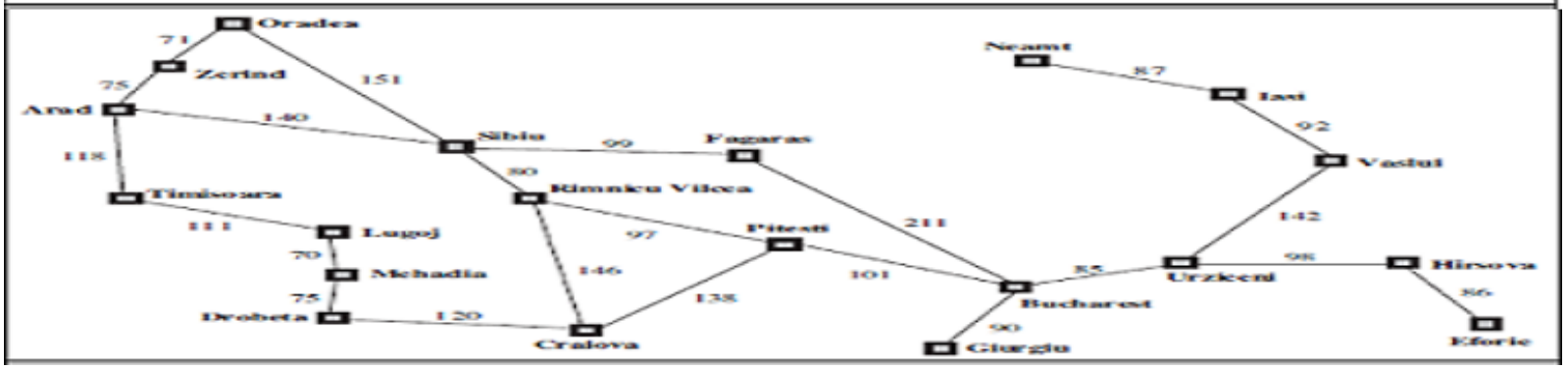
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



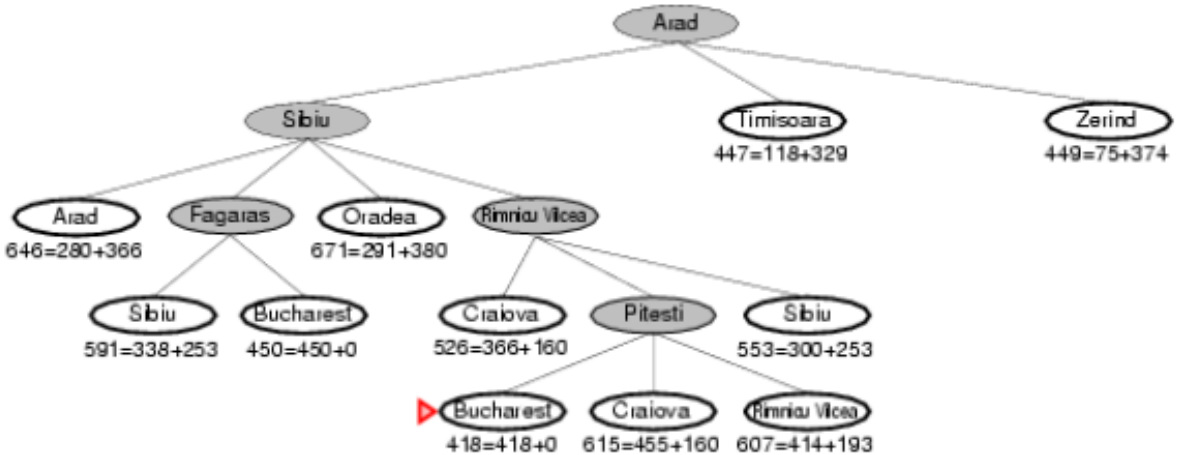
A^* EXAMPLE



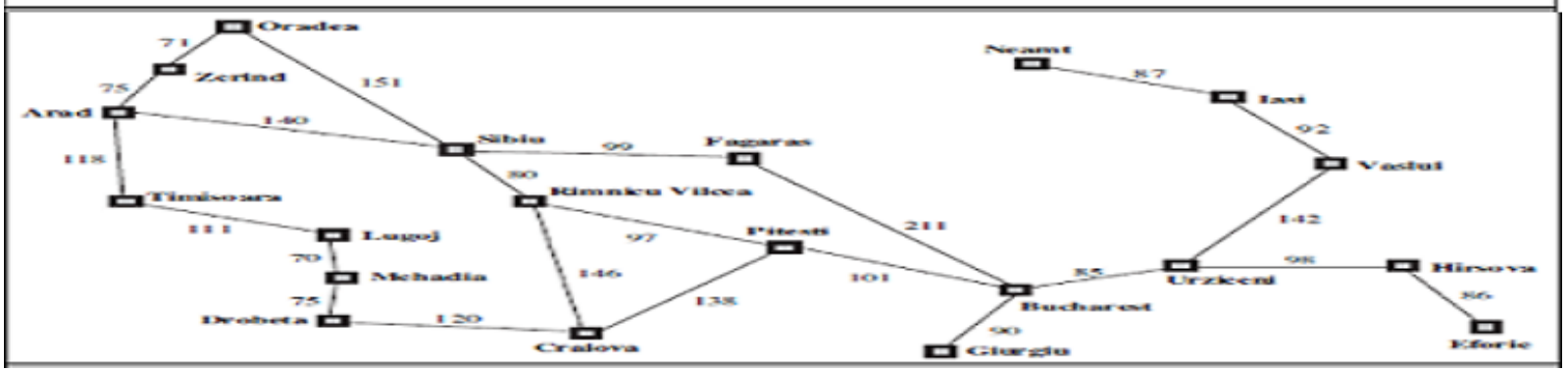
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



A^* EXAMPLE



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



Properties of A^* depend on conditions on $h(n)$

- We want to analyze the behavior of the resultant search.
  - Completeness, time and space, optimality?
- To obtain such results we must put some further conditions on the heuristic function $h(n)$ and the search space.

Conditions on $h(n)$: Admissible

- We always assume that $C(s_1, a, s_2) \geq \varepsilon > 0$ for any two states s_1, s_2 and any action a: the cost of any transition is greater than zero and can't be arbitrarily small.
- Let $h^*(n)$ be the cost of an optimal path from n to a goal node (∞ if there is no path). Then an admissible heuristic satisfies the condition

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

  - an admissible heuristic never over-estimates the cost to reach the goal, i.e., it is optimistic
- Hence $h(g) = 0$, for any goal node g
- Also $h^*(n) = \infty$ if there is no path from n to a goal node

Consistency (aka monotonicity)

- A stronger condition than $h(n) \leq h^*(n)$.
- A monotone/consistent heuristic satisfies the triangle inequality: for all nodes n_1, n_2 and for all actions a

$$h(n_1) \leq C(n_1, a, n_2) + h(n_2)$$

$C(n_1, a, n_2)$ means the cost of getting from the terminal state of n_1 to the terminal state of n_2 via action a.

- Note that there might be more than one transition (action) between n_1 and n_2, the inequality must hold for all of them.
- Monotonicity implies admissibility.

$$\forall n_1, n_2, a,\ h(n_1) \leq C(n_1, a, n_2) + h(n_2) \implies \forall n,\ h(n) \leq h^*(n)$$

Intuition

Admissibility

$h(n) \leq h^*(n)$ means we won't miss any promising paths.

- If it is really cheap to get to a goal via n (i.e both $g(n)$ and $h^*(n)$ are low), then $f(n) = g(n) + h(n)$ will also be low, and the search won't ignore n in favour of more expensive options. optimality.
- This can be formalized to show that admissibility implies optimality.

Monotonicity

$h(n_1) \leq C(n_1, a, n_2) + h(n_2)$

- This says something similar, but in addition we cannot be "locally" mislead.

Consistency $\implies$ Admissible

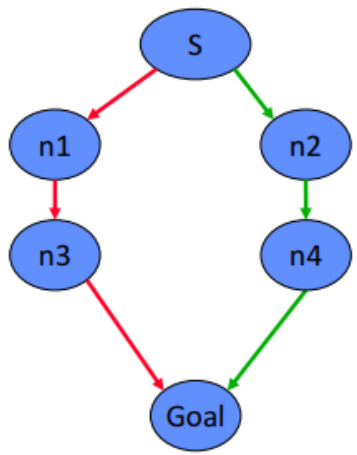
$$\forall n_1, n_2, a,\ h(n_1) \leq C(n_1, a, n_2) + h(n_2) \implies \forall n,\ h(n) \leq h^*(n)$$

Proof:
If no path exists from n to a goal then $h^*(n) = \infty$ and $h(n) \leq h^*(n)$.
Else let $n \to n_1 \to \cdots \to n^*$ be an OPTIMAL path from n to a goal (with actions $a_1, a_2, \ldots$). Note that the cost of this path is $h^*(n)$, and each subpath $n_i \to \cdots \to n^*$ has cost equal to $h^*(n_i)$.
Prove $h(n) \leq h^*(n)$ by induction on the length of this optimal path.
Base case: $n = n^*$
by our conditions on h, $h(n) = 0 \leq h(n^*) = 0$
Induction hypothesis: $h(n_1) \leq h^*(n_1)$
$h(n) \leq C(n, a_1, n_1) + h(n_1) \leq C(n, a_1, n_1) + h^*(n_1) = h^*(n)$

THE CONVERSE IS NOT TRUE

That means, in general, we cannot hope that every admissible heuristic is also monotonic.
In the example below, h is not consistent, since $h(n_2) > c(n_2 \to n_4) + h(n_4)$. However h is admissible (check this!). Then we find the optimal path. But we are mislead into ignoring n_2 after we expand n_1.

$\to$ step cost $= 100$
$\to$ step cost $= 200$
$h(n_1) = 50, h(n_2) = 200$
$h(n_3) = 50, h(n_4) = 50$
$\{S\} \to \{n_1[200+50=250], n_2[200+100=300]\}$
$\to \{n_2[100+200=300], n_3[400+40=450]\}$
$\to \{n_4[200+50=250], n_3[400+50=450]\}$
$\to \{goal[300+0=300], n_3[400+50=450]\}$



Consequences of monotonicity

1. The f-values of nodes along any path must be non-decreasing.

Let $\langle \text{Start} \to s_1 \to s_2 \to \cdots \to s_k \rangle$ be a path. Let n_i be the subpath $\langle \text{Start} \to s_1 \to s_2 \to \cdots \to s_i \rangle$. We claim that

$$f(n_i) \leq f(n_{i+1})$$

Proof.

$$\begin{aligned} f(n_i) &= C(\text{Start} \to \cdots \to n_i) + h(n_i) \\ &\leq C(\text{Start} \to \cdots \to n_i) + C(n_i \to n_{i+1}) + h(n_{i+1}) \\ &\leq C(\text{Start} \to \cdots \to n_i \to n_{i+1}) + h(n_{i+1}) \\ &\leq g(n_{i+1}) + h(n_{i+1}) = f(n_{i+1}) \end{aligned}$$

Consequences of monotonicity

2. If n_2 is expanded immediately after n_1, then $f(n_1) \leq f(n_2)$ (the f-value of expanded nodes is monotonic non-decreasing)
Proof.

- If n_2 was on OPEN when n_1 was expanded, then $f(n_1) \leq f(n_2)$ otherwise we would have expanded n_2.
- If n_2 was added to OPEN after expansion of n_1, then n_2 extends the path of n_1. That is, the path represented by n_1 is a prefix of the path represented by n_2. By property (1) we have $f(n_1) \leq f(n_2)$ as the f-values along a path are non-decreasing.

CONSEQUENCES OF MONOTONICITY

3. Corollary: the sequence of f-values of the nodes expanded by A^* is non-decreasing. I.e, If n_2 is expanded after (not necessarily immediately after) n_1, then
$f(n_1) \leq f(n_2)$
(the f-value of expanded nodes is monotonic non-decreasing)
Proof.

- If n_2 was on OPEN when n_1 was expanded, then $f(n_1) \leq f(n_2)$ otherwise we would have expanded n_2.
- If n_2 was added to OPEN after expansion of n_1, then let n be an ancestor of n_2 that was present when n_1 was being expanded (this could be n_1 itself). We have $f(n_1) \leq f(n)$ since A^* chose n_1 while n was present on OPEN. Also, since n is along the path to n_2, by property (1) we have $f(n) \leq f(n_2)$. So, we have $f(n_1) \leq f(n_2)$.

Consequences of monotonicity

4.When n is expanded every path with lower f-value has already been expanded.

Proof.

Assume by contradiction that there exists a path $\langle \text{Start}, n_0, n_1, \ldots, n_{i-1}, n_i, n_{i+1}, \ldots, n_k \rangle$ with $f(n_k) < f(n)$ and n_i is its last expanded node.

- n_{i+1} must be on OPEN while n is expanded, so
  - (A) by (1) $f(n_{i+1}) \leq f(n_k)$ since they lie along the same path.
  - (B) since $f(n_k) < f(n)$ so we have $f(n_{i+1}) < f(n)$
  - (C) by (2) $f(n) \leq f(n_{i+1})$ because n is expanded before n_{i+1}.
- Contradiction from b&c!

Consequences of monotonicity

5. With a monotone heuristic, the first time A^* expands a state, it has found the minimum cost path to that state.
Proof.

- Let $PATH1 = \langle \text{Start}, s_0, s_1, \ldots, s_k, s \rangle$ be the first path to a state s found. We have $f(path1) = c(PATH1) + h(s)$.
- Let $PATH2 = \langle \text{Start}, t_0, t_1, \ldots, t_j, s \rangle$ be another path to s found later. we have $f(path2) = c(PATH2) + h(s)$.
- Note $h(s)$ is dependent only on the state s (terminal state of the path) it does not depend on how we got to s.
- By property (3), $f(path1) \leq f(path2)$
- hence: $c(PATH1) \leq c(PATH2)$

Consequences of monotonicity

Complete

Yes, consider a least cost path to a goal node

- $SolutionPath = \langle \text{Start} \rightarrow n_1 \rightarrow \cdots \rightarrow G \rangle$ with cost $c(SolutionPath)$. Since $h(G) = 0$, this means that $f(SolutionPath) = cost(SolutionPath)$
- Since each action has a cost $\geq \varepsilon > 0$, there are only a finite number of paths that have f-value < c(SolutionPath). None of these paths lead to a goal node since SolutionPath is a least cost path to the goal.
- So eventually SolutionPath, or some equal cost path to a goal must be expanded.

Consequences of monotonicity

Time and Space complexity.

- When h(n) = 0, for all n h is monotone.
- A* becomes uniform-cost search!
- It can be shown that when h(n) > 0 for some n and still admissible, the number of nodes expanded can be no larger than uniform-cost.
- Hence the same bounds as uniform-cost apply. (These are worst case bounds). Still exponential unless we have a very good h!
- In real world problems, we sometimes run out of time and memory. IDA* can sometimes be used to address memory issues, but IDA* isn't very good when many cycles are present.

Consequences of monotonicity

Optimality

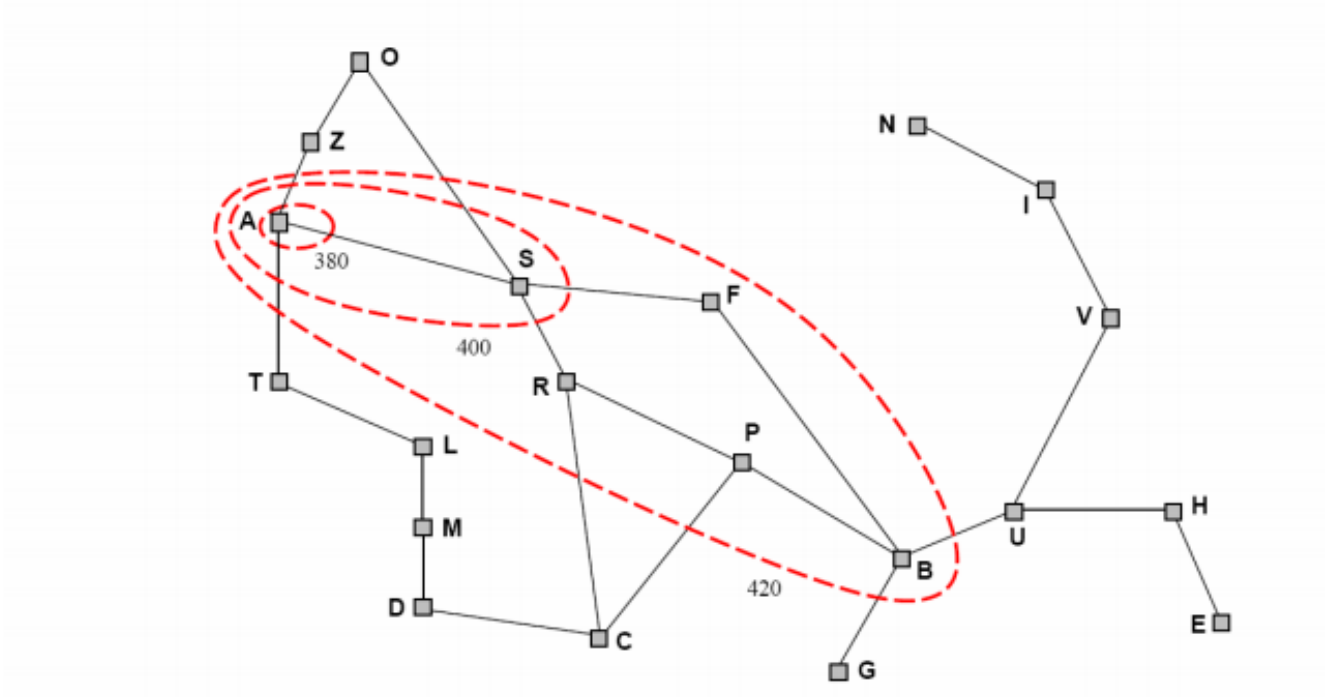
Yes, by (5) the first path to a goal node must be optimal.
5. With a monotone heuristic, the first time A* expands a state, it has found the minimum cost path to that state.

Cycle Checking

We can use a simple implementation of cycle checking (multiple path checking) - Just reject all search nodes visiting a state already visited by a previously expanded node. By property (5), we need keep only the first path to a node, rejecting all subsequent paths.

SEARCH GENERATED BY MONOTONICITY

Gradually adds “f-contours” of nodes. Inside each contour, the f values are less or equal to the contour value.



- For uniform cost search, bands are circular
- Better heuristics generate bands that stretch more toward the goal.

Admissibility without monotonicity

When "h" is admissible but not monotonic.

- Time and Space complexity remain the same. Completeness holds.
- Optimality still holds (without cycle checking), but need a different argument: don't know that paths are explored in order of cost.

Proof (By contradiction)

- Assume the goal path $\langle S, \ldots, G \rangle$ found by A^* has cost bigger than the optimal cost, i.e. $C^*(G) < f(G)$.
- There must exist a node n in the optimal path that is still in the frontier (Exercise: why?)
- We have $f(n) = g(n) + h(n) \leq g(n) + h^*(n) = C^*(G) < f(G)$.
- Then n must have been selected before G by A^*. Contradiction. (Exercise: Show exactly where the contradiction is).

Admissibility without monotonicity

What about cycle checking?

- No longer guaranteed we have found an optimal path to a node the first time we visit it.
- So, cycle checking might not preserve optimality. To fix this, for previously visited nodes, must remember cost of previous path. If new path is cheaper, must explore again.

Space Problems with A*

- A* has the same potential space problems as BFS or UCS
- IDA* - Iterative Deepening A* is similar to Iterative Deepening Search and addresses space issues, but because of f-values vary more than node depths it can require many iterations.

IDA* - Iterative Deepening A*

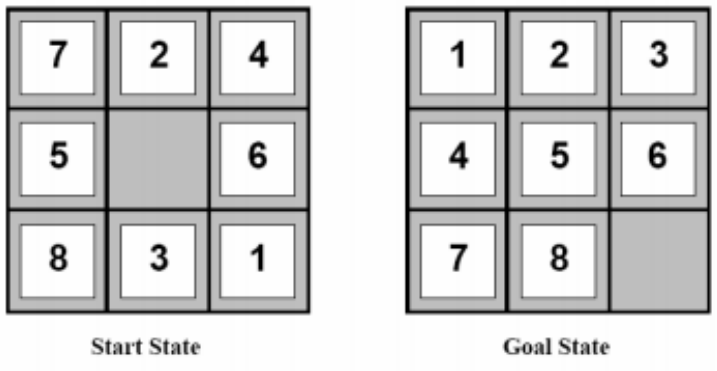
Objective: reduce memory requirements for A*

- Like iterative deepening, but now the "cutoff" is the f-value (g+h) rather than the depth
- At each iteration, the cutoff value is the smallest f-value of any node that exceeded the cutoff on the previous iteration
- Avoids overhead associated with keeping a sorted queue of nodes.
- Two new parameters:
  - curBound (any node with a bigger f-value is discarded)
  - smallestNotExplored (the smallest f-value for discarded nodes in a round) when OPEN becomes empty, the search starts a new round with this bound.
    - Easier to expand all nodes with f-value EQUAL to the f-limit. This way we can compute "smallestNotExplored" more easily.

Building Heuristics: Relaxed Problem

- One useful technique is to consider an easier problem, and let h(n) be the cost of reaching the goal in the easier problem.

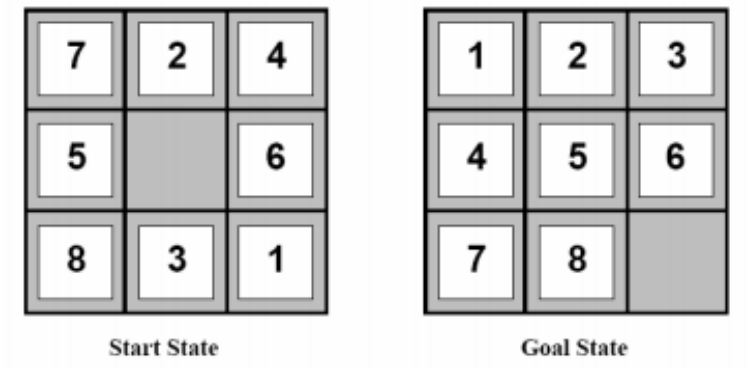
8-Puzzle



- Can move a tile from square A to B if
  - A is adjacent (left, right, above, below) to B
  - and B is blank

BUILDING HEURISTICS: RELAXED PROBLEM

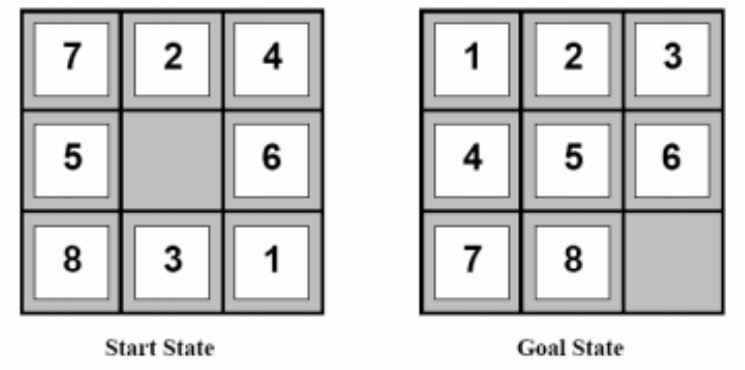
8-Puzzle



- Can relax some of these conditions
  - 1. can move from A to B if A is adjacent to B (ignore whether or not position is blank)
  - 2. can move from A to B if B is blank (ignore adjacency)
  - 3. can move from A to B (ignore both conditions).

BUILDING HEURISTICS: RELAXED PROBLEM

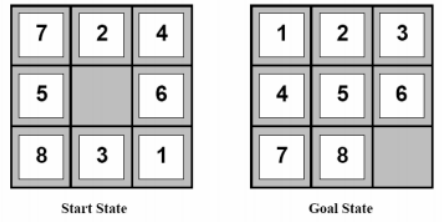
8-Puzzle



- #3 “can move from A to B (ignore both conditions)”. leads to the misplaced tiles heuristic.
  - To solve the puzzle, we need to move each tile into its final position.
  - Number of moves = number of misplaced tiles.
  - Clearly h(n) = number of misplaced tiles $\leq$ the h*(n) the cost of an optimal sequence of moves from n.

BUILDING HEURISTICS: RELAXED PROBLEM

8-Puzzle



- #1 "can move from A to B if A is adjacent to B (ignore whether or not position is blank)"

  leads to the Manhattan distance heuristic.
  - To solve the puzzle we need to slide each tile into its final position.
  - We can move vertically or horizontally.
  - Number of moves = sum over all of the tiles of the number of vertical and horizontal slides we need to move that tile into place.
  - Again h(n) = sum of the Manhattan distances $\leq$ h*(n)
    - in a real solution we need to move each tile at least that far and we can only move one tile at a time.

BUILDING HEURISTICS: RELAXED PROBLEM

Comparison of IDS and A* (average total nodes expanded):

Depth	IDS	A*(Misplaced) **h1**	A*(Manhattan) **h2**
10	47,127	93	39
14	3,473,941	539	113
24	---	39,135	1,641

Let h1=Misplaced, h2=Manhattan
Does h2 always expand fewer nodes than h1?

- Yes! Note that h2 dominates h1, i.e. for all n: $h1(n) \leq h2(n)$. From this you can prove h2 is better than h1 (once both are admissible).
- Therefore, among several admissible heuristic the one with highest value is better (will cause fewer nodes to be expanded).

Building Heuristics: Relaxed Problem

- The optimal cost to nodes in the relaxed problem is an admissible heuristic for the original problem!
- **Proof Idea:** the optimal solution in the original problem is a (not necessarily optimal) solution for relaxed problem, therefore it must be at least as expensive as the optimal solution in the relaxed problem.
- So admissible heuristics can sometimes be constructed by finding a relaxation whose optimal solution can be easily computed.

BUILDING HEURISTICS: PATTERN DATABASES

- Try to generate admissible heuristics by solving a subproblem and storing the exact solution cost for that subproblem
- By searching backwards from these goal states, we can compute the distance of any configuration of these tiles to their goal locations. We are ignoring the identity of the other tiles.
- For any state n, the number of moves required to get these tiles into place, form a lower bound on the cost of getting to the goal from n.

- Here are goals for two sub-problems (called Corner and Fringe) of 15-puzzle.
- Note the location of BLANK!

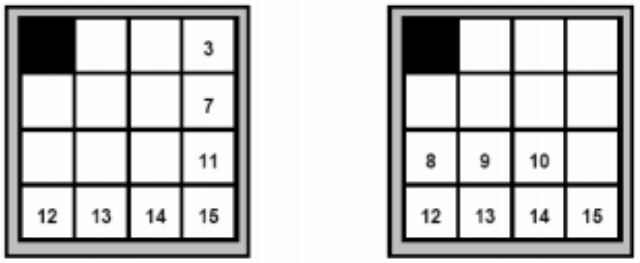


Fig. 2. The Fringe and Corner Target Patterns.

Building Heuristics: Pattern databases

- These configurations are stored in a database, along with the number of moves required to move the tiles into place.
- The maximum number of moves taken over all of the databases can be used as a heuristic.
- On the 15-puzzle
  - The fringe database yields about 345 fold decrease in the search tree size
  - The corner database yields about 437 fold decrease
- Sometimes disjoint patterns can be found, then the number of moves can be added rather than taking the max (if we only count moves of the target tiles).

Exercise

Consider a search algorithm with evaluation function

$$f(n) = ag(n) + bh(n)$$

given $h(n)$ an admissible heuristic and $a > 0, b > 0$ are numeric constants.
What conditions should a and b satisfy so the algorithm is guaranteed to be complete?
Hint: Write $f(n) = a(g(n) + \frac{b}{a}h(n))$.

THANK YOU

Some of the slides were created by Dan Klein and Pieter Abbeel at UC Berkeley and Fahiem Bacchus and other UofT instructors..