RN, Chapter
4.1 – 4.2

Heuristic Search

- Best-First
- A^*
- Heuristic Functions

Some material from: D Lin, J You, JC Latombe

Search Overview

- Introduction to Search
- Blind Search Techniques
- Heuristic Search Techniques
  - Best-First
  - A^*
  - Heuristic Functions
- Stochastic Algorithms
- Game Playing search
- Constraint Satisfaction Problems

Heuristic Search

- "Blind" methods only know Goal / NonGoal
- Often ∃ other problem-specific knowledge that can guide search:
- **Heuristic fn** *h(n): Nodes →ℜ*
  estimate of distance from *n* to a goal

  Eg: straight line on map,
  or "Manhattan distance",
  or …

5	4	3	4
4	3	2	3
			2
2	1	0	1

- Use: Given list of nodes to expand, choose node *n* with min'l *h(.)*

Heuristic Function

- h(n) estimates cost of cheapest path from node n to goal node
- Example: 8-puzzle

5		8
4	2	1
7	3	6

n

1	2	3
4	5	6
7	8	

goal

$h_1(n)$ = number of misplaced tiles
= 6

Heuristic Function

- h(n) estimates cost of cheapest path from node n to goal node
- Example: 8-puzzle

5		8
4	2	1
7	3	6

n

1	2	3
4	5	6
7	8	

goal

$h_1(n)$ = number of misplaced tiles

= 6

$h_2(n)$ = sum of the distances of every tile to its goal position

= 3 + 1 + 3 + 0 + 2 + 1 + 0 + 3

= 13

Greedy Best-First Search

BestF_Search(*start, operations, is_goal*): *path*

  L := makeList(*start*)

  loop

    n := arg $\min_{n_i \in L}$ $h(n_i)$

    ;; "most promising" node in L according to h(.)

    if [i*s_goal*(*n*)]

      return(*n*)

    S := successors(*n, operators*)

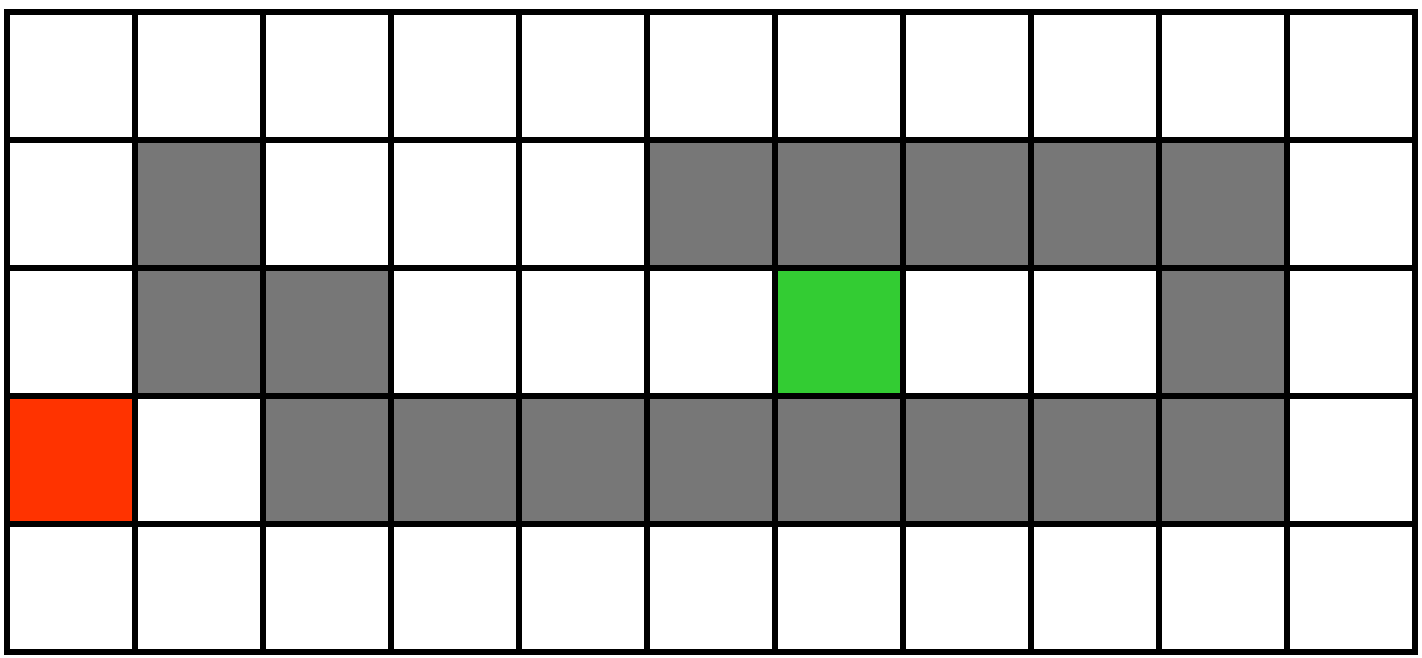
Idea: choose frontier node with smallest h-value

  ie, "closest to goal"

Can also return "path from start to *n*"

. . . by identifying each node with path

Robot Navigation



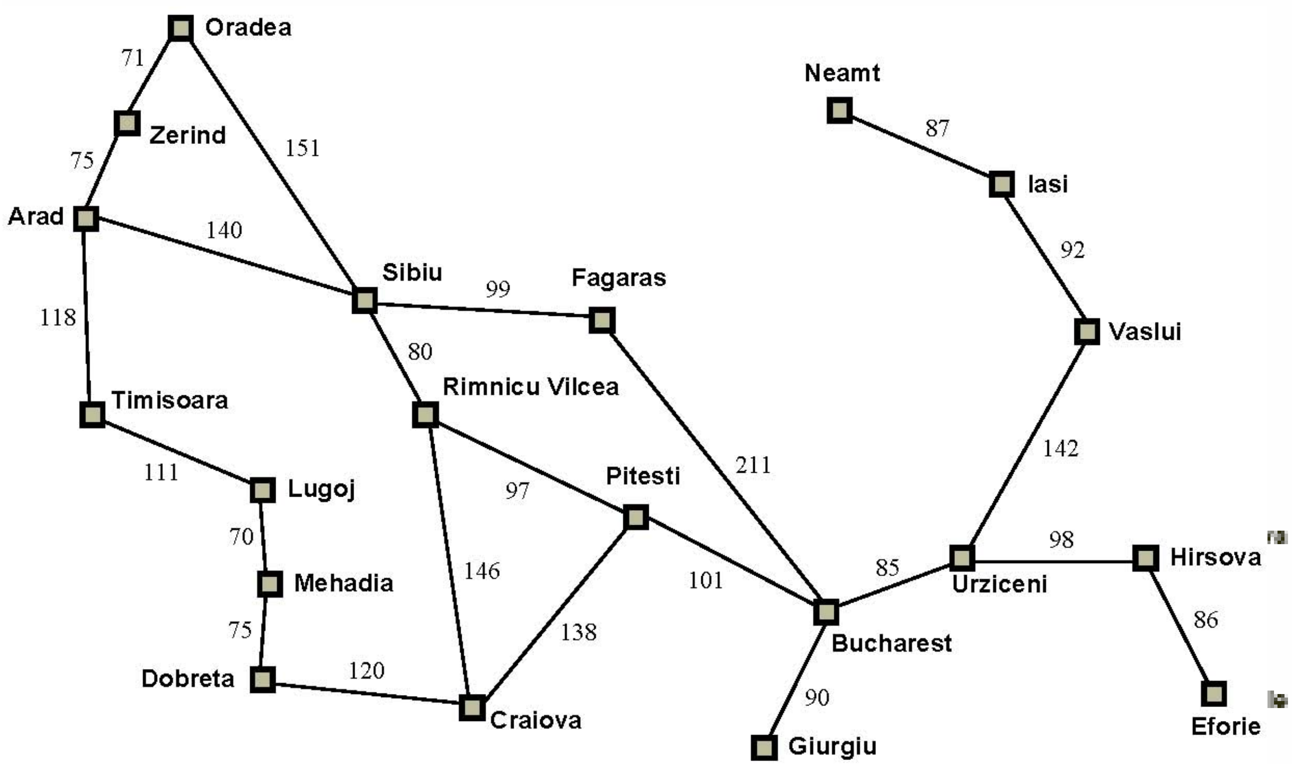
Robot Navigation

Edmonton

$h(n)$ = Manhattan distance to the goal

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

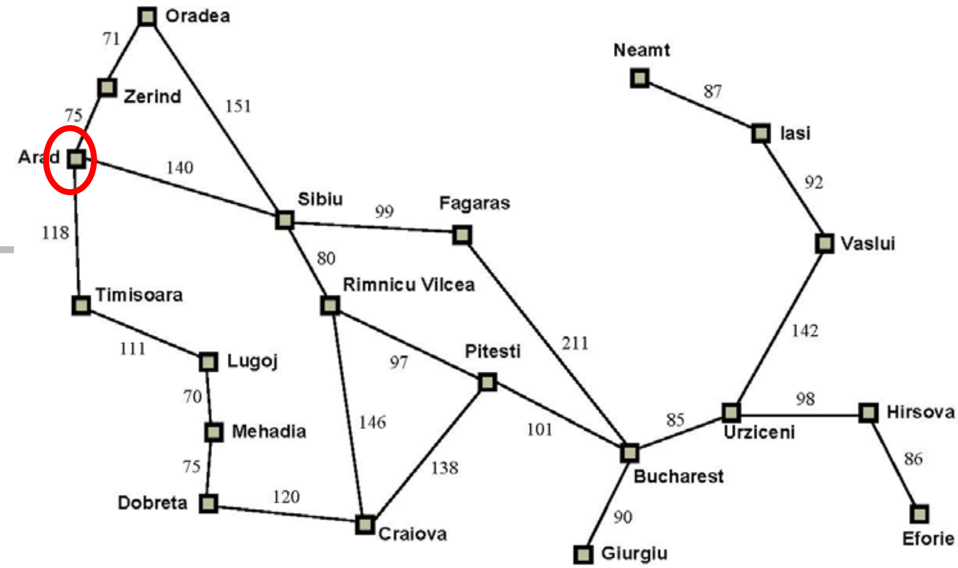
Heuristic Function – Bulgaria



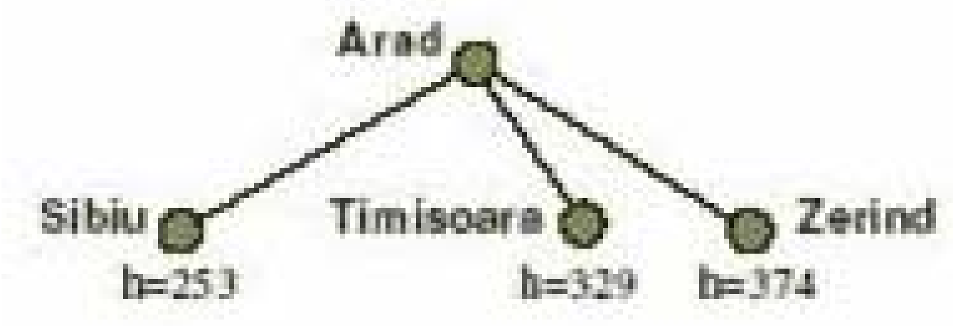
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

$h_{SLD}(n)$ is straight-line distance from n to goal (Bucharest)

Best First

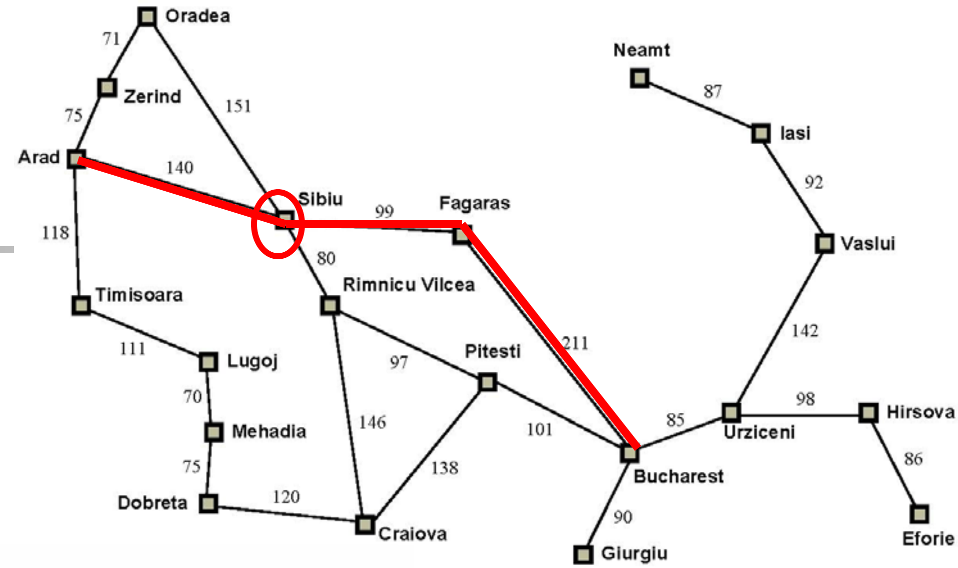


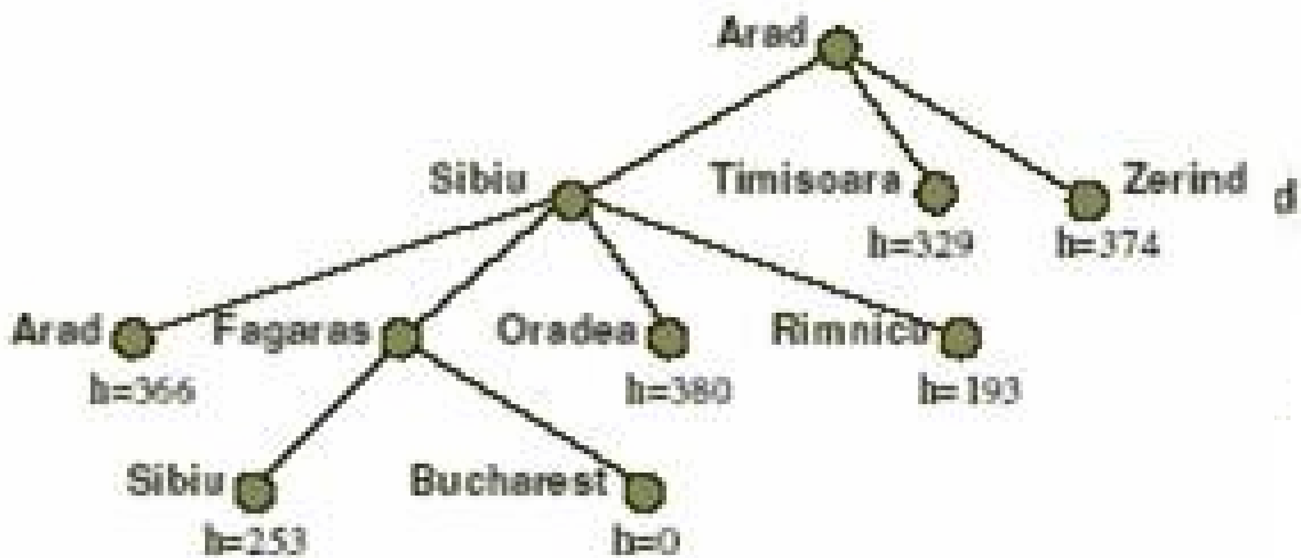
Oradea
Neamt
Zerind
Iasi
Arad
Sibiu
Fagaras
Vaslui
Timisoara
Rimnicu Vilcea
Pitesti
Lugoj
Mehadia
Urziceni
Hirsova
Bucharest
Dobreta
Craiova
Giurgiu
Eforie
71
151
75
87
140
92
99
118
80
211
142
111
97
70
146
101
85
98
75
138
86
120
90



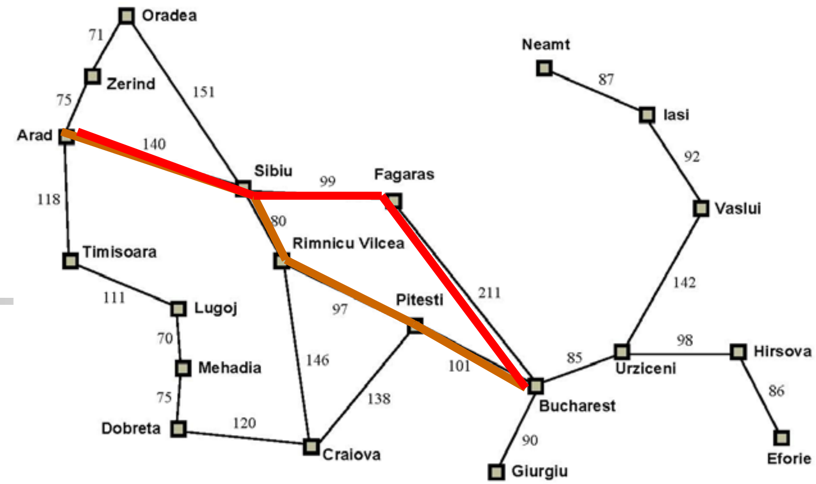
Arad
Sibiu
Timisoara
Zerind
h=253
h=329
h=374

Best First


Oradea
Neamt
71
Zerind
87
151
Iasi
75
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Lugoj
97
Pitesti
211
70
98
Mehadia
146
85
Hirsova
101
Urziceni
75
138
86
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu

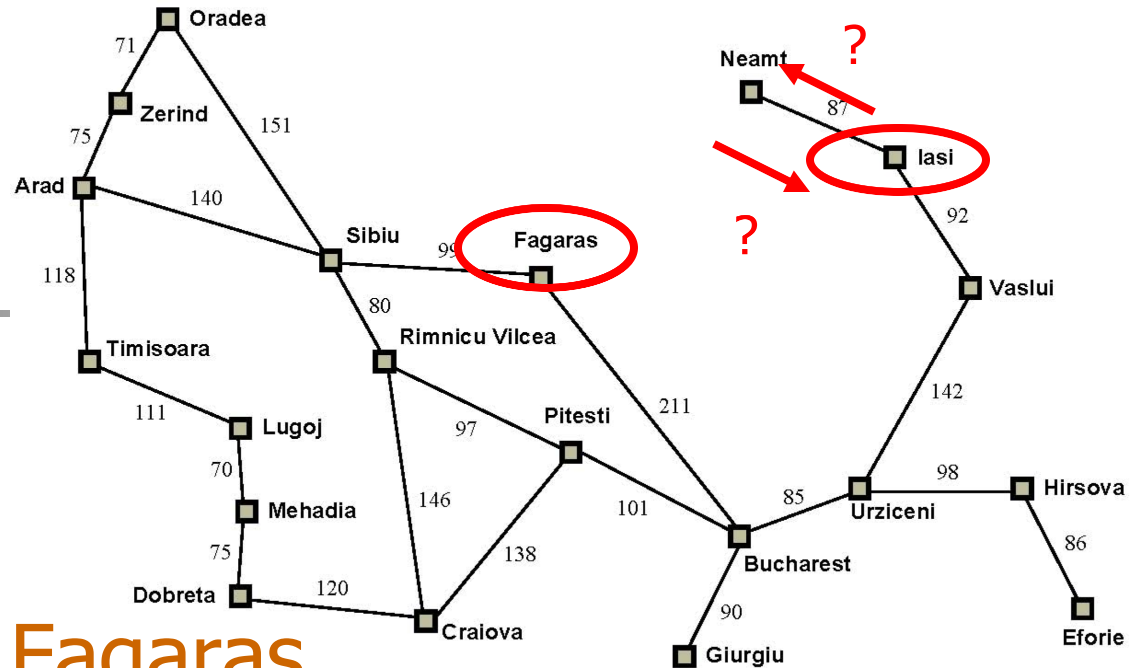

Arad
Sibiu
Timisoara
h=329
Zerind
h=374
Arad
h=366
Fagaras
Oradea
h=380
Rimnicu
h=193
Sibiu
h=253
Bucharest
h=0

BestFirst is SubOptimal



- h_{SLD} finds path:
  Arad → Sibiu → Fagaras → Bucharest
  $(\text{Cost} = 140 + 99 + 211 = 450)$
- Not optimal!
  C(Arad → Sibiu → Rimnicu → Pitesti → Bucharest)
  $= 140 + 80 + 97 + 101 = 418$
  $<$ h_{SLD}'s solution!
- BestFirst is greedy:
  takes BIGGEST step each time…

BestFirst can Loop



- Consider: Iasi → Fagaras

  h_{SLD} suggests: Iasi → Neamt
- Worse: Unless search alg detects repeated states, BestFirst will oscillate:

  Iasi → Neamt → Iasi → Neamt → …
- Loops are a real problem…

Properties of Greedy Best-First Search

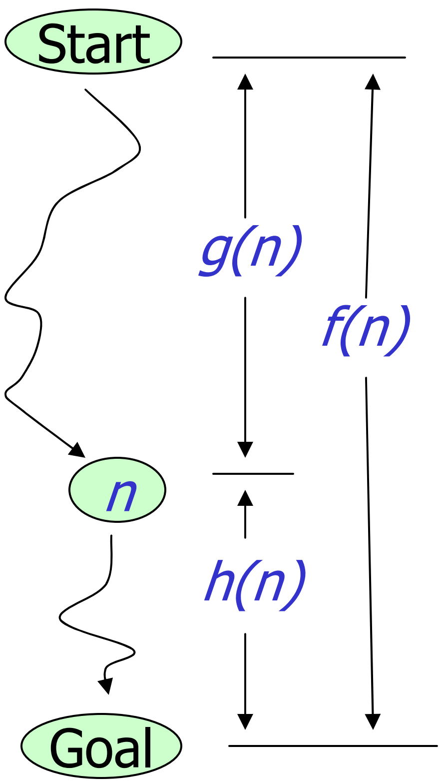
- If state space is finite and
  we avoid repeated states,
  THEN Best-First search is complete,
  but in general is not optimal
- If state space is finite and
  we do ***not*** avoid repeated states,
  THEN Best-First search is not complete.
- If the state space is infinite,
  THEN Best-First search is not complete.

Analysis of Greedy BestFirst

- **Complete?** **No**
  …can go down ∞-path (oscillate)
- **Optimal?** **No**
  … may not find shortest path
- **Time:** $O(b^m)$
- **Space:** $O(b^m)$
  (if $h(.) \equiv 0$, could examine entire space)

- Worst of both worlds
  - ≈DFS: too greedy!
  - ≈BFS: too much space!

A* Search



- Find cheapest path, quickly
- Consider both:
  - Path from start to *n*:

    g(n) = cost of path found to *n*
  - Path from *n* to goal (est.):

    h(n) = estimate of cost from *n* to a goal
- *f(n) = g(n)+h(n)*
  - est of cost of path from start to goal, via *n*

$$\underbrace{\underbrace{start \xrightarrow{\text{actual}} n}_{g(n)}\ \underbrace{\xrightarrow{\text{estimate}} goal}_{h(n)}}_{f(n)}$$

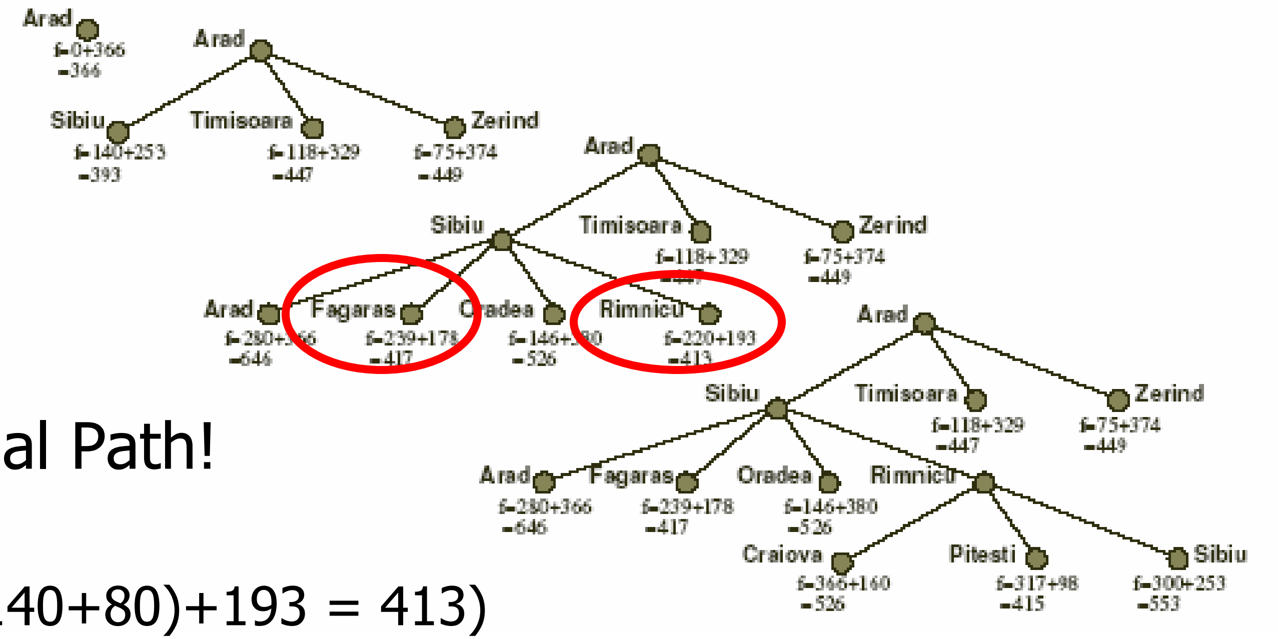
A* Search, con't

(so far) (to go)

$$\underbrace{\underbrace{start \xrightarrow{\text{actual}} n}_{g(n)} \underbrace{\xrightarrow{\text{estimate}} goal}_{h(n)}}_{f(n)}$$

- A* selects node with min'l *f(n)*
  - …ie, node with lowest estimated distance from
  - start to goal, constrained to go via that node
- … mix of $\left\{ \begin{array}{c} \text{lowest-cost-first} \\ \text{best-first} \end{array} \right\}$ searches!

Example of A*



Note: Finds Optimal Path!

- A* expands
  - Rimnicu (f = (140+80)+193 = 413)

  over
  - Faragas (f = (140+99)+178 = 417)
- Why?

  Fagaras is closer to Bucharest (than Rimnicu)

  but

  path taken to get to Fargaras
  is not as efficient at getting close to Bucharest
  … as Rimnicu

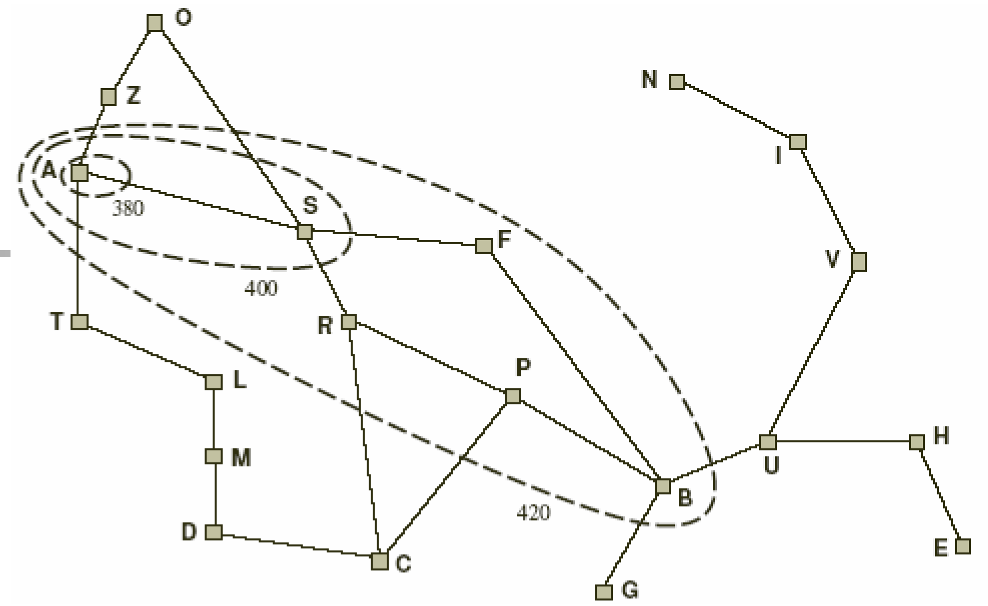
Robot Navigation

Edmonton

$f(n) = g(n)+h(n)$, with $h(n)$ = Manhattan distance to goal

8+3	7+4	6+3	5+6	4+7	3+8	2+9	3+10	4	5	6
7+2		5+6	4+7	3+8						5
6+1				2+9	1+10	0+11	1	2		4
7+0	6+1									5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4+9	5	6

How A* Searches



- Contour-lines of "equal-f values"
- A* expands nodes with increasing *f(n)* values
- If use *h(.)≡ 0* (UniformCost) get Circles

  ⇒ more nodes expanded (in general)!

Admissible heuristic

- *h*(n)* = cost of *optimal path* from *n* to a goal node
- Heuristic *h(n)* is admissible if:

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

- Admissible heuristic is always *optimistic*
- True for
  - Straight Line [map traversal]
  - Manhattan distances [8-puzzle]
  - Number of attacking queens [n-queens] [place all queens, then move]

$\Rightarrow$ *f(.)* is *under-estimate*

Heuristics for 8-Puzzle

5		8
4	2	1
7	3	6

n

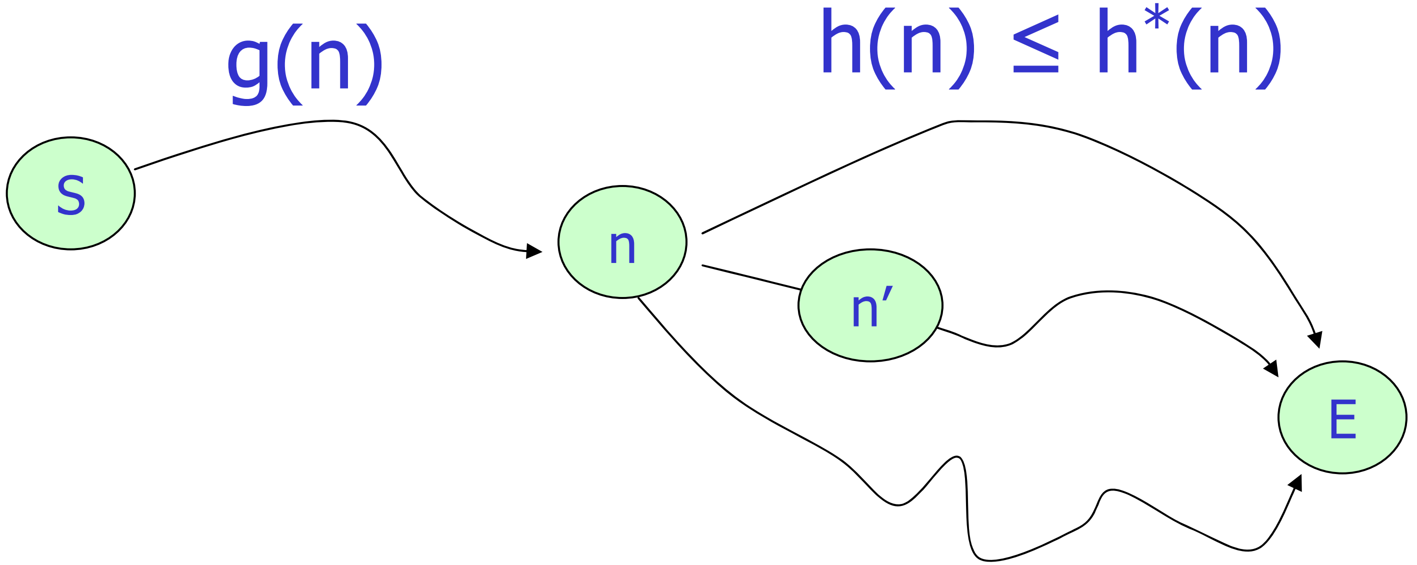
1	2	3
4	5	6
7	8	

goal

Admissible??

+ • $h_1(n)$ = number of misplaced tiles … = 6
+ • $h_2(n)$ = sum of distances of each tile to goal posn … = 13
− • $h_3(n) = h_1(n) + 3 \times h_2(n)$ … = 45
+ • $h_4(n) \equiv 0$ … $= 0$
+ • $h_5(n) = \min\{ h_1(n), h_2(n) \}$ … = 6

f(n) is monotonic



- $f(n) \leq f(n')$, as from-S-to-E-via-n is less constrained than from-S-to-E-via-n-n'

Monotonic *f(.)*

- f(.) is "monotonic" $\equiv$
  f(Successor(n)) $\geq$ f(n)

- Always true if
  $| h(n) - h(m) | \leq d(n,m)$
  … *d(n,m)* is distance from *n* to *m*

- If true:
  first path that A* finds to node, is always shortest

- If f (.) not monotonic, can modify to be:

Eg, n′ $\in$ Successor(n)

f(n) = g(n)+h(n) = 3+4 = 7

f(n′) = g(n′)+h(n′) = 4+2 = 6

- But… any path through *n′* is also path through *n*,
  so f(n) must be $\geq$ 7

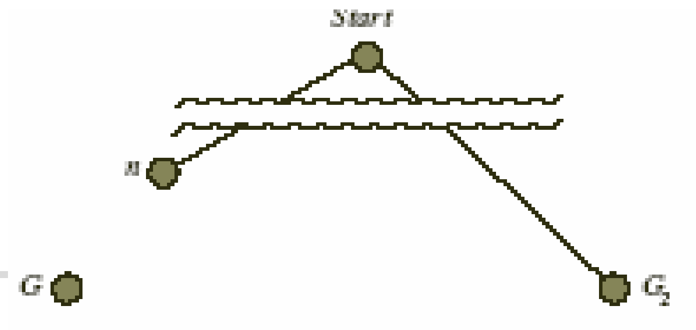
$\Rightarrow$ should reset f(n′) = 7

$\Rightarrow$ use

$$f(n') = \max\{ f(n), g(n')+h(n') \}$$

Called "path-max equation"
… ignores misleading numbers in heuristic

A^* is OPTIMAL



Thrm: A^* always returns optimal solution if

- $\exists$ solution
- *h(n)* is under-estimate

PROOF:

Let G be optimal goal, with $f(G) = g(G) = f$

G_2 be suboptimal goal, with $f(G_2) = g(G_2) > f$

If A^* returns $G_2 \Rightarrow$

G_2 is chosen over n, where n is node on optimal path to G

This only happens if $f(G_2) \leq f(n)$

As f is monotonically increasing along every path,

$\Rightarrow f = f(G) \geq f(n)$

Hence, $f \geq f(G_2)$ … ie, if $g(G) \geq g(G_2)$

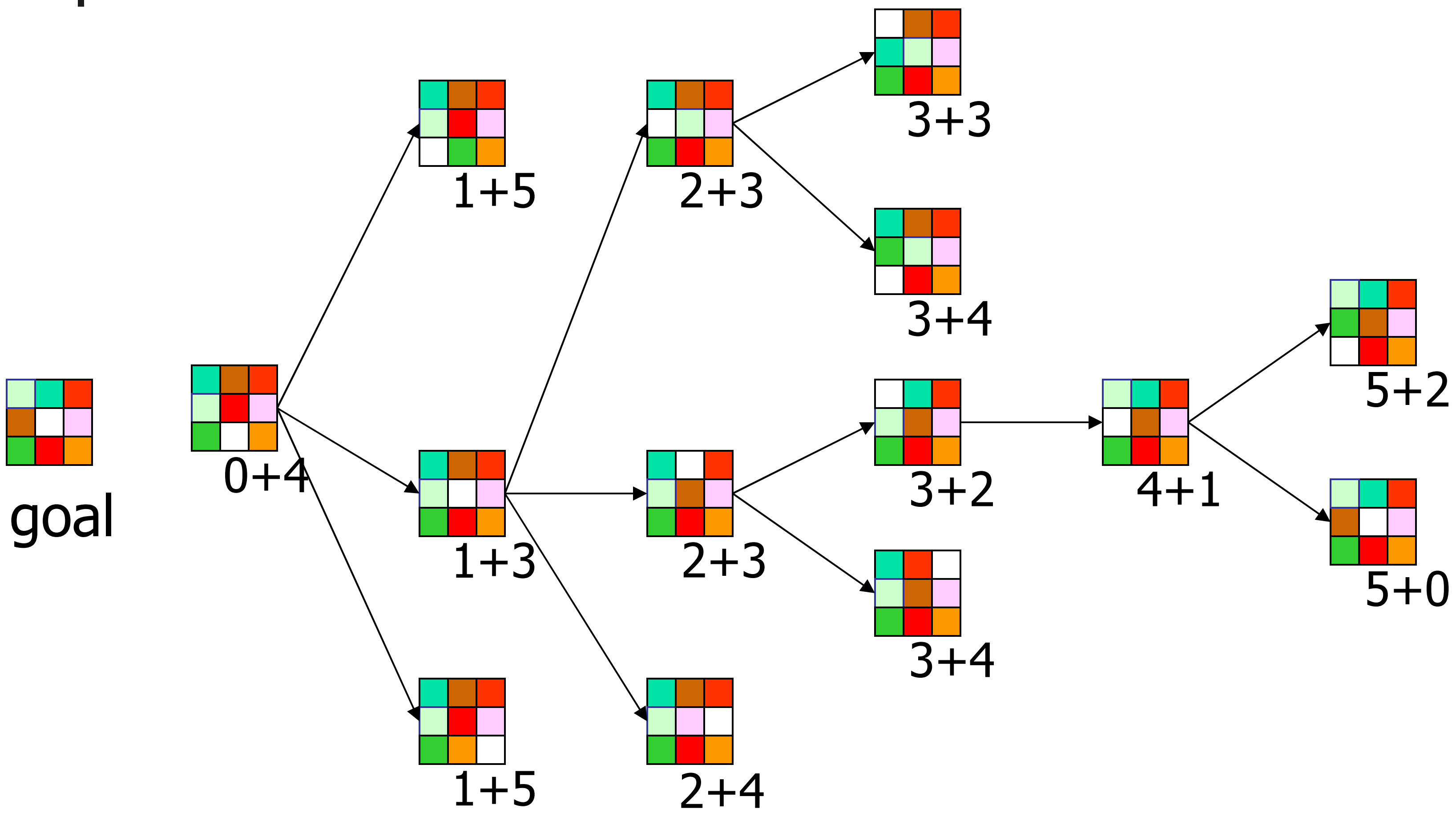
… contradicting claim that G_2 is suboptimal! []

Properties of A*

- **A* is Optimally Efficient**
  Given the information in *h(.)*,
  no other optimal search method can expand fewer nodes.
  Non-trivial and quite remarkable!
- **A* is Complete**
  … unless there are ∞ nodes w/ $f(n) < f^*$
  - A* is Complete
    if branching factor is finite & arc costs bounded above zero
    ($\exists \varepsilon > 0$ s.t. $c(a_i) \geq \varepsilon$)
- **Time/Space Complexity**:
  Still exponential as ≈breadth-first.
  … unless $|h(n) - h(n^*)| \leq O(\log(h(n^*))$
  $h(n^*)$ = true cost of getting from *n* to goal

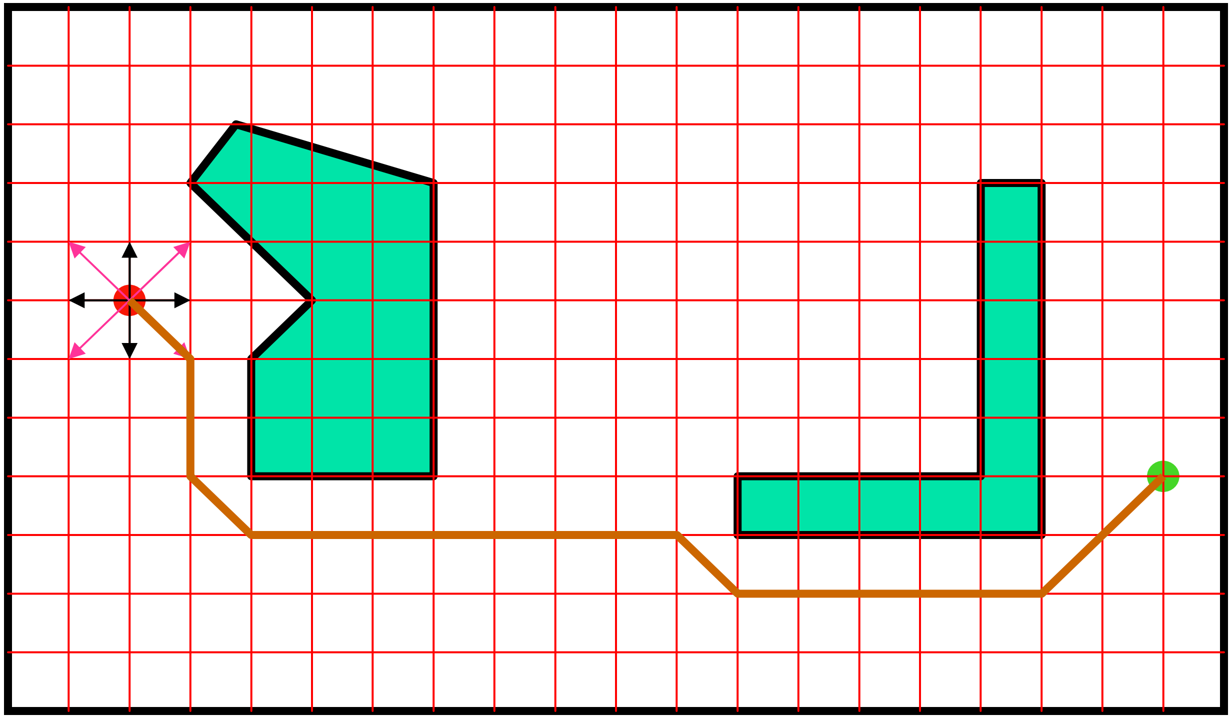
8-Puzzle

$f(n) = g(n) + h(n)$
with $h(n)$ = number of misplaced tiles



Robot navigation

f(n) = g(n) + h(n), with *h(n)* = straight-line distance from *n* to goal



Cost of one horizontal/vertical step = 1

Cost of one diagonal step = $\sqrt{2}$

A^* Topics

- Which heuristic?
- Avoiding Loops
- Iterative Deepening A^*

Heuristics for 8-Puzzle

5		8
4	2	1
7	3	6

n

1	2	3
4	5	6
7	8	

goal

Admissible??

+ • $h_1(n)$ = number of misplaced tiles … = 6

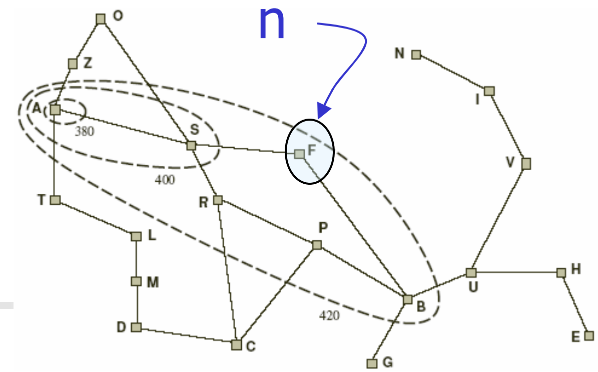
+ • $h_2(n)$ = sum of distances of each tile to goal posn …= 13

– • ~~$h_3(n) = h1(n) + 3 \times h2(n)$ … = 45~~

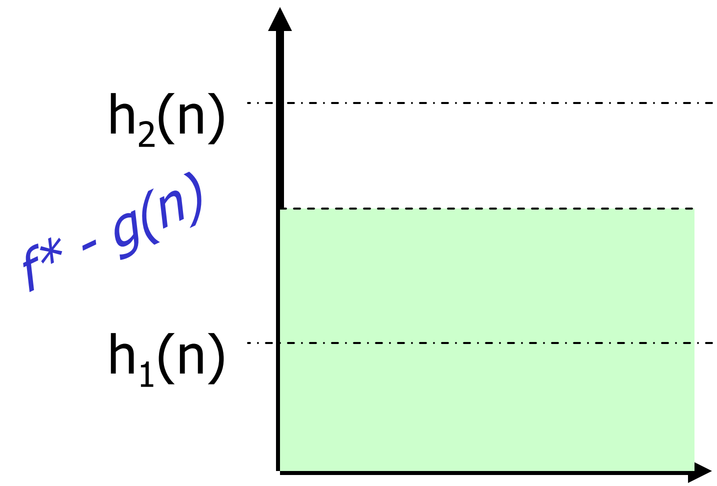
+ • $h_4(n) == 0$ … = 0

Many admissible heuristics … which to use??

Importance of *h(.)*



- $A^*(\,h_i\,)$ expands all nodes with
  $f(n) = g(n)+h_i(n) < f^*$
  … ie, with $h_i(n) < f^* - g(n)$

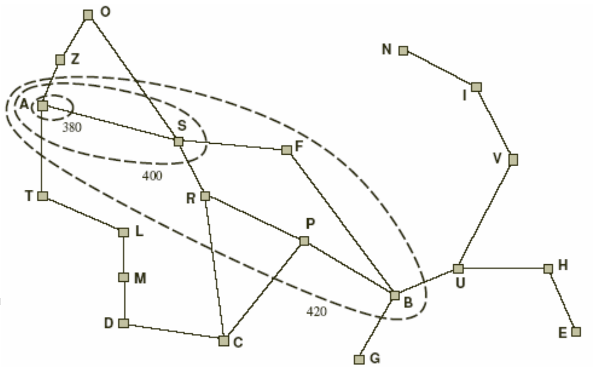


- $h_1(n) < h_2(n) \quad \Rightarrow$
  If $A^*(h_2)$ expands n,
  then $A^*(h_1)$ expands n!
  . . . but not vice versa

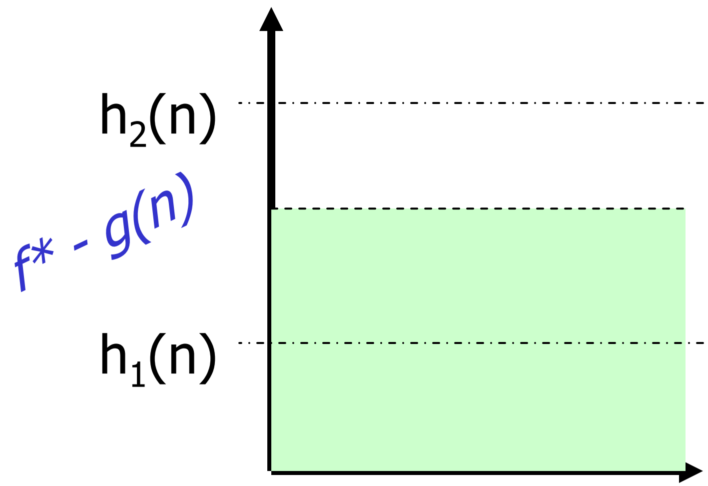
  $A^*(h_2)$ might expand FEWER nodes

- So LARGER $h_i()$ means fewer n's expanded!

Importance of *h(.)*



- LARGER $h_i()$ means fewer n's expanded!



- As $h_C \leq h_M \leq h^*$, prefer h_M !

- Gen'l:
  Want largest *h()* that is under-estimate

Effect of Different Heuristic Functions

	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_C)$	$A^*(h_M)$	IDS	$A^*(h_C)$	$A^*(h_M)$
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

- "Effective Branching Factor" b is solution to

$$N = 1+(b^*)+(b^*)^2 + (b^*)^3 + \ldots + (b^*)^d$$

where N is # of nodes searched

d is solution depth

About Heuristics

- Heuristics are intended to orient the search along promising paths
- Time spent evaluating heuristic function must be recovered by a better search
  - "Perfect heuristic function" would mean NO search!
- Deciding which node to expand ≡ "*meta-reasoning*"
- Heuristics…
  - may not always look like numbers
  - may involve large amount of knowledge

Inventing Heuristics

- Solve problem, then compute backwards...
- If $\{h_1, \ldots h_k\}$ all underestimates,
  use $h_{max}(n) = \max \{ h_i(n) \}$
  (Still an under-estimate, but larger …)
- **Relaxation**:
  Consider SIMPLER version of problem.
  As heuristic, use
  - "exact answer to approx problem"

Inventing Heuristics

- Original:

  Can move tile from sq A to sq B if
  … A is adjacent to B and B is blank.

- Relaxed version#1:

  Can move tile from sq A to sq B if
  … ~~A is adjacent to B and~~ B is blank.

  - Ie, can TELEPORT tile to blank

  ⇒ # of misplaced tiles h_C

5		8
4	2	1
7	3	6

- Relaxed version#2:

  Can move tile from sq A to sq B if
  … A is adjacent to B ~~and B is blank~~.

  - Ie, can walk over non-blank tile

  ⇒ Manhattan distance h_M

5		8
4	2	1
7	3	6

Other Tricks

- Patterns Databases
- Learning from part experiences

Avoiding Repeated States in A*

If the heuristic *h(.)* is monotonic, then:

- Let *CLOSED* be the list of states associated with expanded nodes
- When a new node *n* is generated:
  - If its state is in *CLOSED*, then discard *n*
  - If it has the same state as another node in the fringe, then discard the node with the largest *f(.)*

Complexity of Consistent A*

- $s = |S|$
  - size of the state space
- $r = |A|$
  - max number of states that can be reached by applying any operator, from any state
- Assume test if state $s \in$ CLOSED is O(1)

$\Rightarrow$ Time complexity of A*: $O(s\, r \log s)$

Iterative Deepening A* (IDA*)

- Use $f(n) = g(n) + h(n)$
  with admissible, consistent $h(.)$
- Each iteration is depth-first with cutoff on the value of $f(n)$ of expanded nodes

AIxploratorium
http://www.cs.ualberta.ca/~aixplore

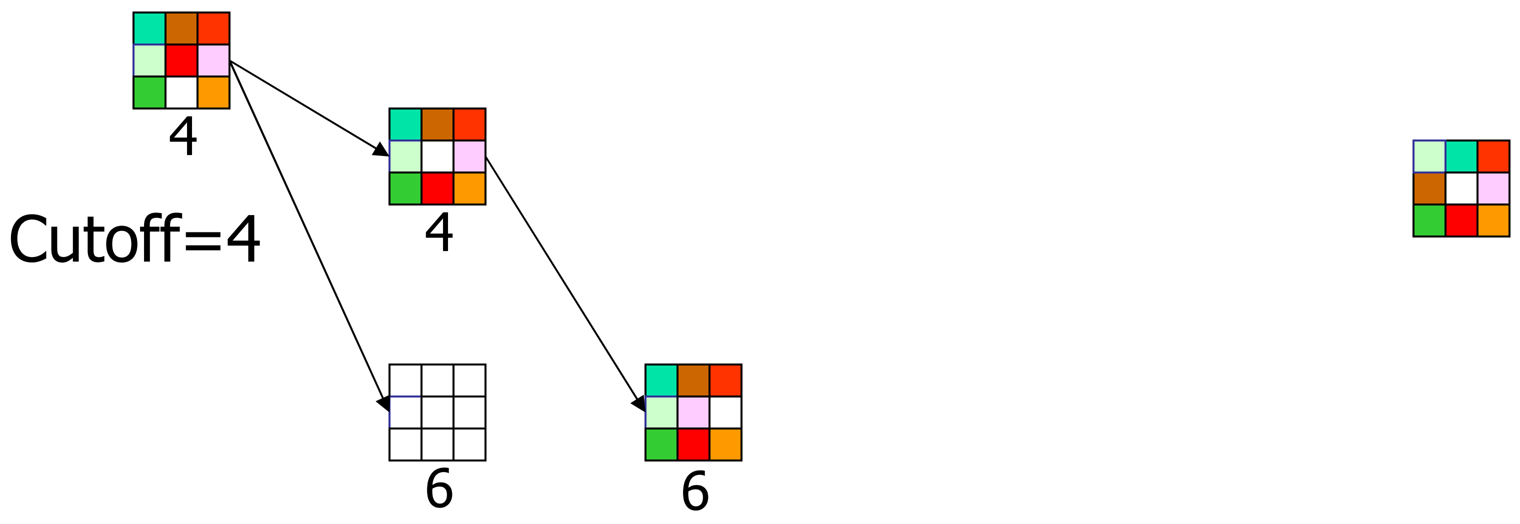
8-Puzzle

f(n) = g(n) + h(n)
with h(n) = number of misplaced tiles

Cutoff=4

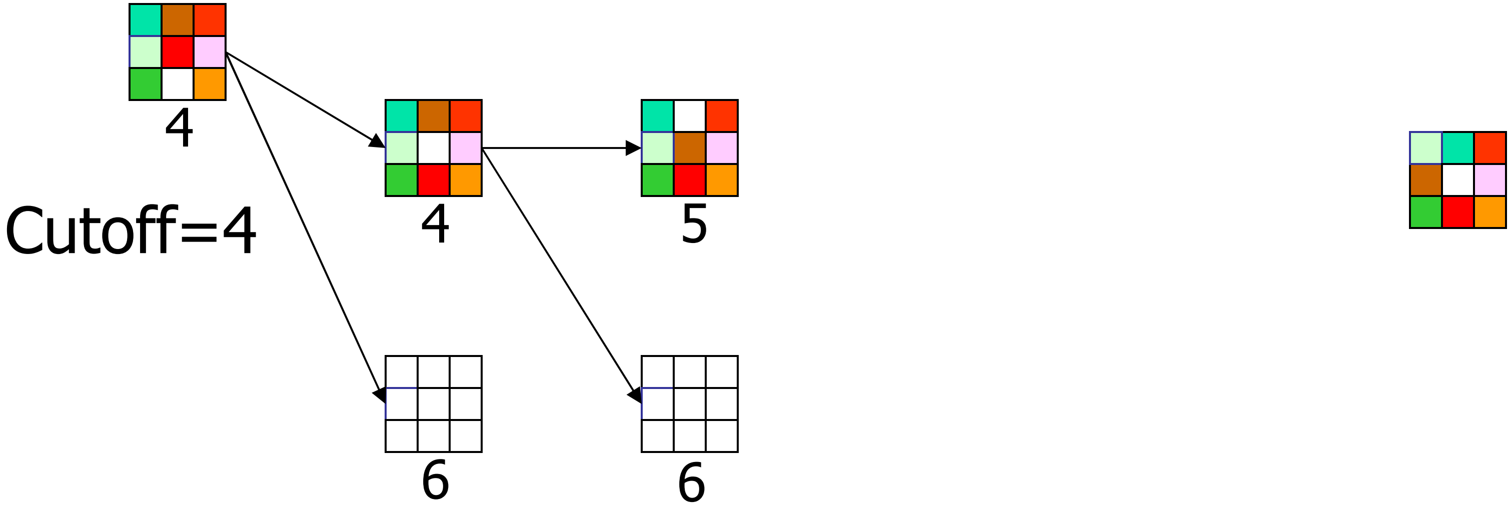
8-Puzzle

$f(n) = g(n) + h(n)$
with $h(n)$ = number of misplaced tiles



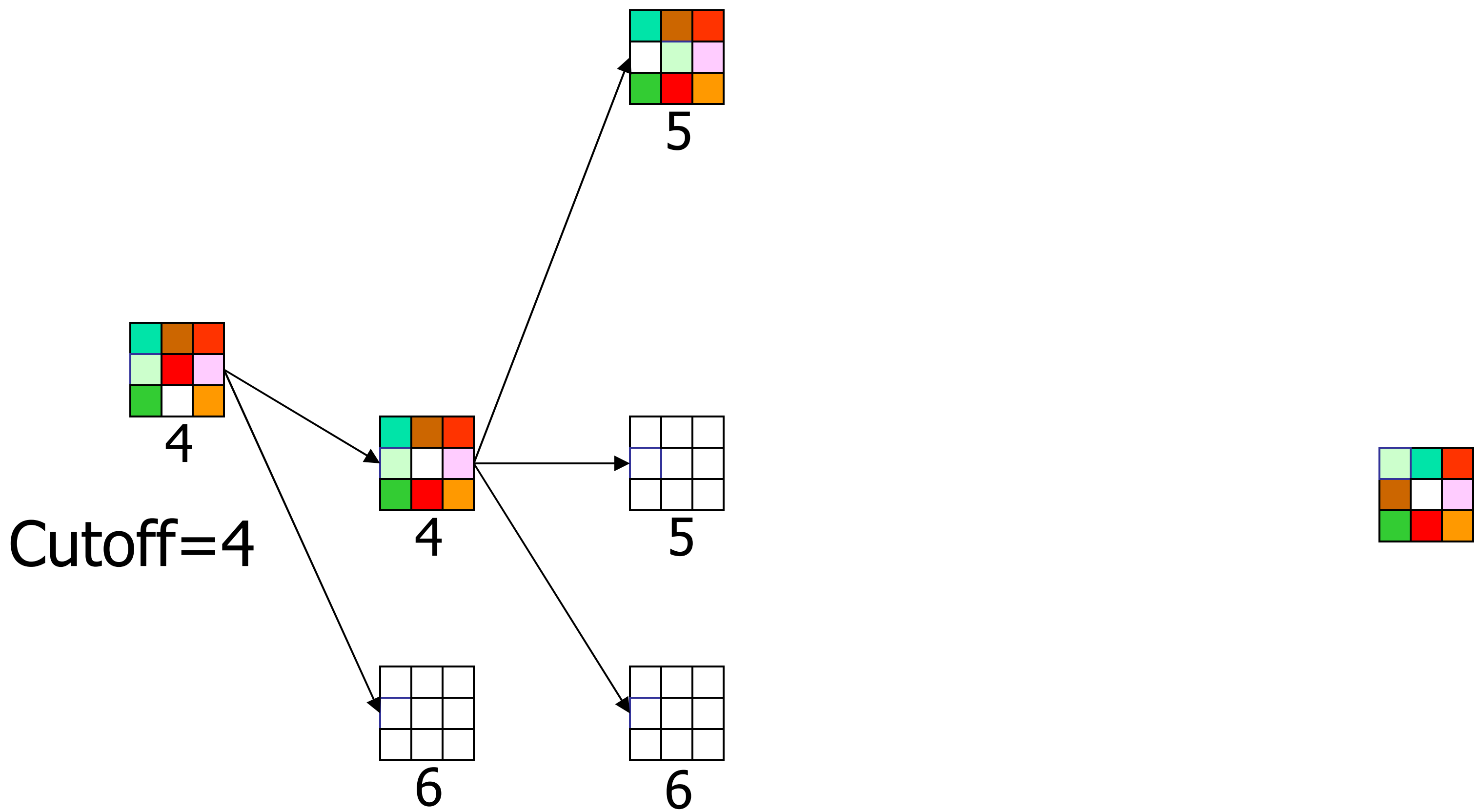
8-Puzzle

f(n) = g(n) + h(n)
with h(n) = number of misplaced tiles



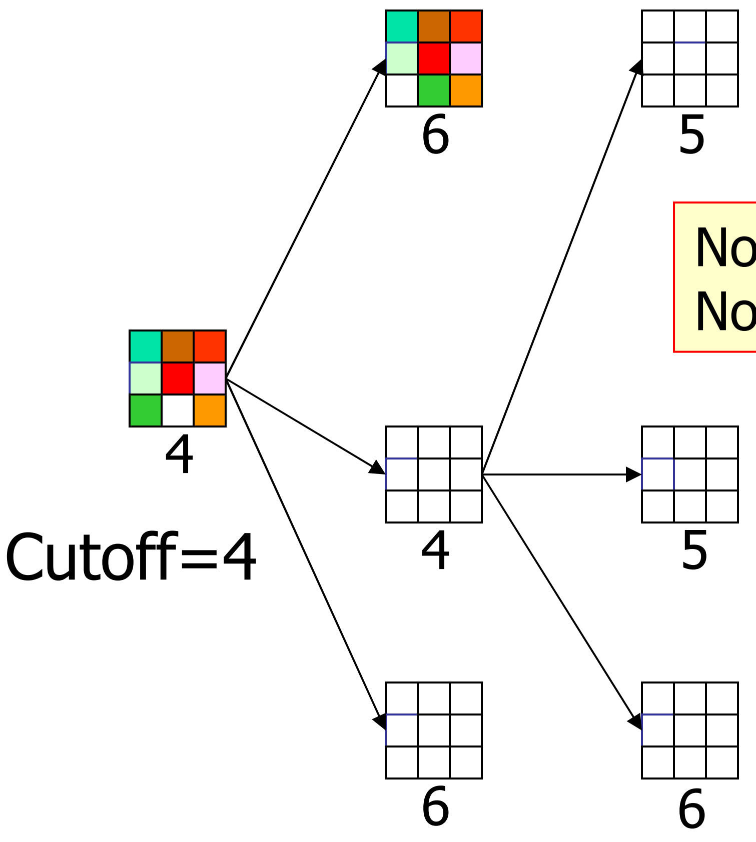
8-Puzzle

$f(n) = g(n) + h(n)$
with $h(n)$ = number of misplaced tiles

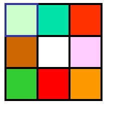


8-Puzzle

$f(n) = g(n) + h(n)$
with $h(n)$ = number of misplaced tiles



No more nodes to expand with Cutoff =4
Now consider Cutoff = 5

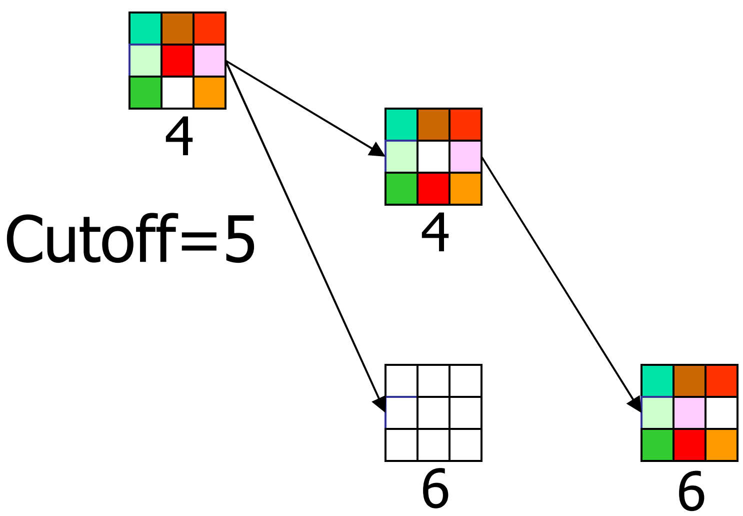


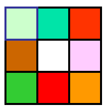
8-Puzzle

$f(n) = g(n) + h(n)$
with $h(n)$ = number of misplaced tiles

8-Puzzle

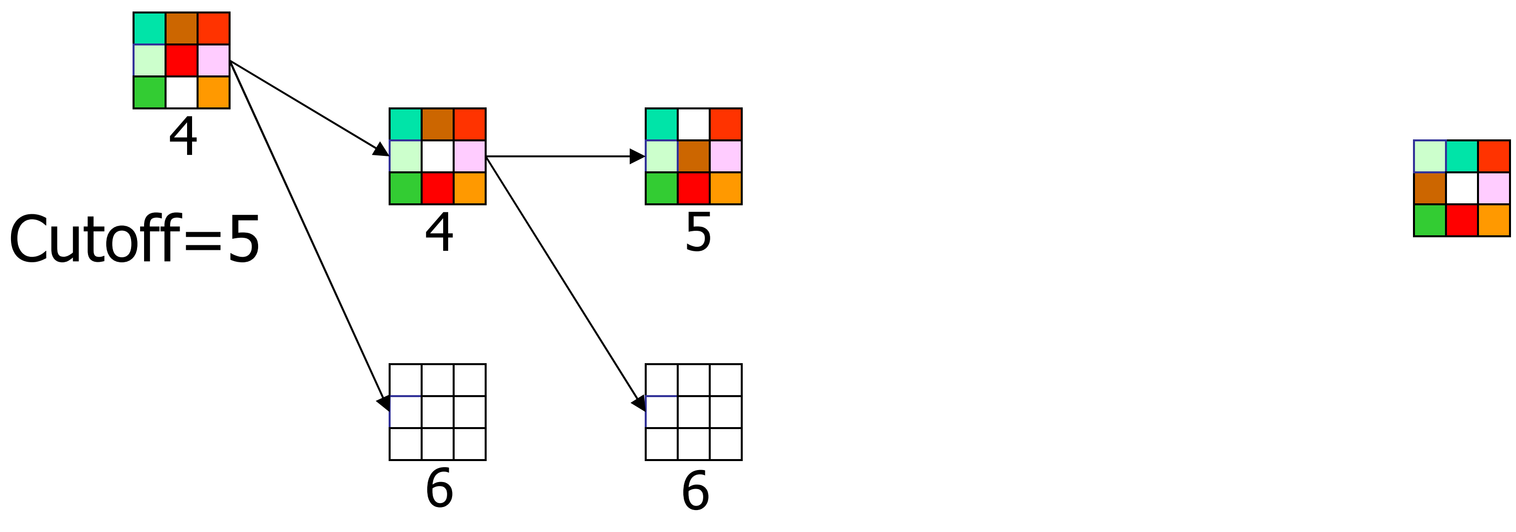
$f(n) = g(n) + h(n)$
with $h(n)$ = number of misplaced tiles





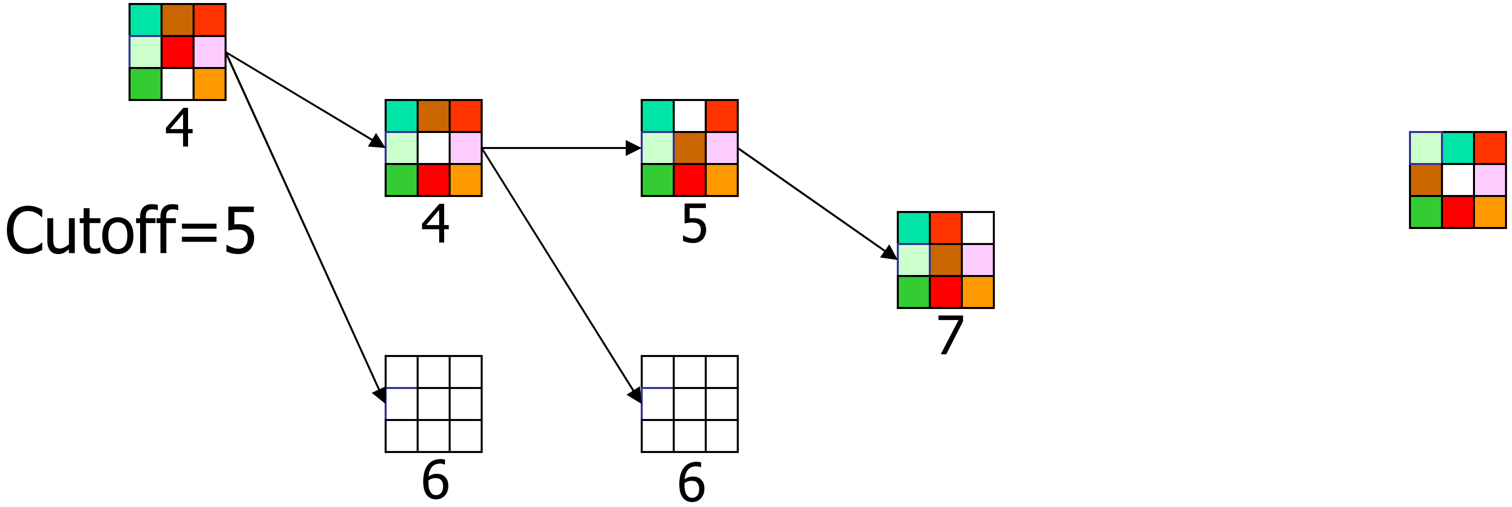
8-Puzzle

$f(n) = g(n) + h(n)$
with $h(n)$ = number of misplaced tiles



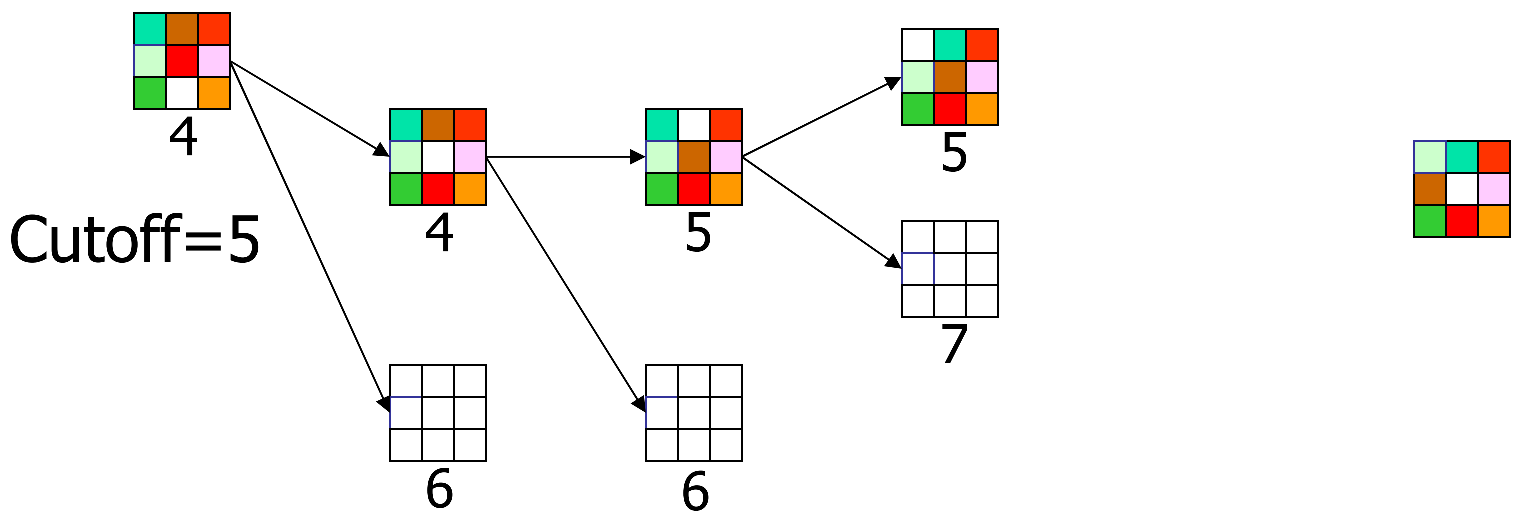
8-Puzzle

$f(n) = g(n) + h(n)$
with $h(n)$ = number of misplaced tiles



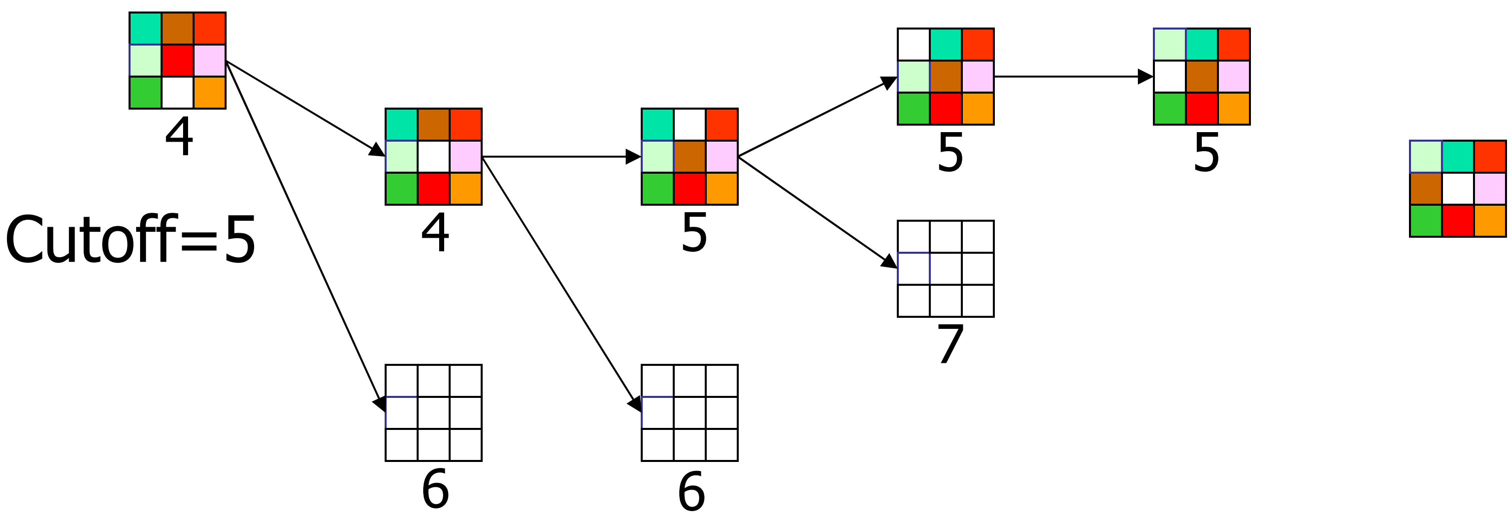
8-Puzzle

f(n) = g(n) + h(n)
with h(n) = number of misplaced tiles



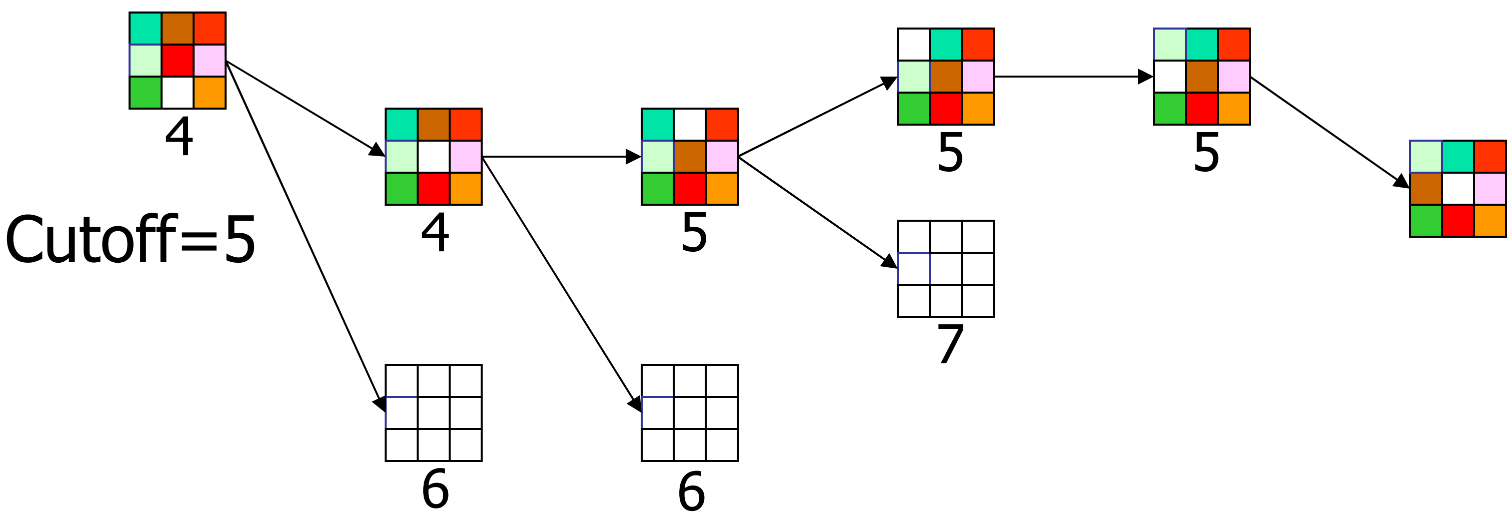
8-Puzzle

$f(n) = g(n) + h(n)$
with $h(n)$ = number of misplaced tiles



8-Puzzle

$f(n) = g(n) + h(n)$
with $h(n)$ = number of misplaced tiles



Summary

- Heuristic function
- Greedy Best-first search
- Admissible heuristic
- A* is complete and optimal
  - Optimally efficient !
- Consistent heuristic and repeated states
- Inventing Heuristics
- IDA*