Informed Search

Combinatorial Explosion

Depth	Nodes	Time	Memory
2	1100	.11 sec	1 meg
4	111,100	11 sec	106 meg
6	10^7	19 min	10 gig
8	10^9	31 hrs	1 tera
10	10^{11}	129 days	101 tera
12	10^{13}	35 yrs	10 peta
14	10^{15}	3523 yrs	1 exa

Rely only on problem description

Informed Methods: Heuristic Search

Idea: Informed search by using problem-specific knowledge.

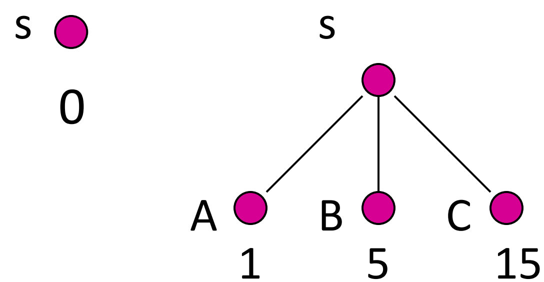
Best-First Search: Nodes are selected for expansion based on an *evaluation function*, *f*(*n*). Traditionally, *f* is a cost measure.

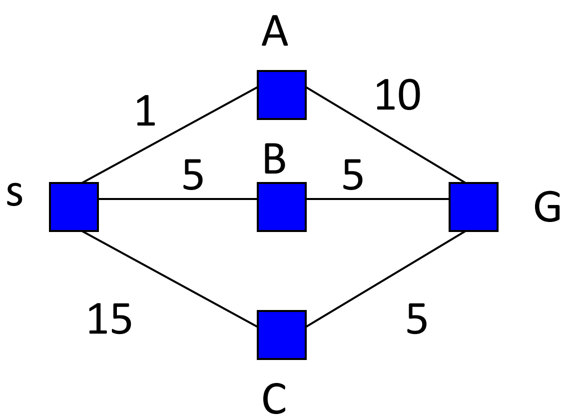
Heuristic: Problem specific knowledge that (tries to) lead the search algorithm faster towards a goal state. Often implemented via *heuristic function h(n).*

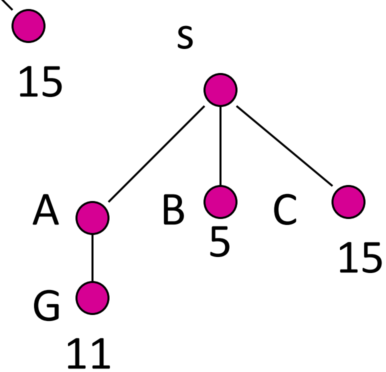
→ Heuristic search is an attempt to search the most promising paths first. Uses heuristics, or rules of thumb, to find the best node to expand next.

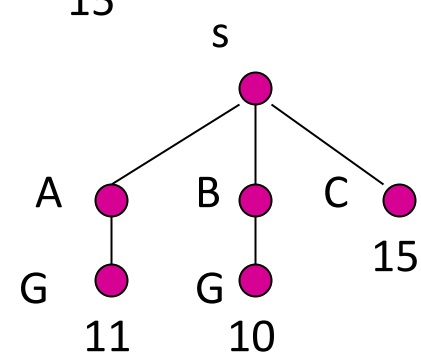
Uniform-cost search is NOT heuristic search

It only looks backwards; has no ability to predict future costs.









Requirement: $g(\text{Successor}(n)) \geq g(n)$

Always expand lowest cost node in open-list.
Goal-test only before expansion, not after generation.

Generic Best-First Search

1. Set L to be the initial node(s) representing the initial state(s).

2. If L is empty, fail. Let n be the node on L that is ``most promising'‘ according to f. Remove n from L.

3. If n is a goal node, stop and return it (and the path from the initial node to n).

4. Otherwise, add *successors*(n) to L. Return to step 2.

seemingly-best-first...

A good heuristic

- Heuristic cost should never overestimate the actual cost of a node
  – I.e. it must be “optimistic”
  – So that we never overlook a node that is actually good

$$\forall n, h(n) \leq C(n)$$

Greedy Best-First Search

Heuristic function *h(n):* estimated cost from node n to nearest goal node.

Greedy Search: Let $f(n) = h(n)$.

Example: 8-puzzle

5	4	
6	1	8
7	3	2

Start State

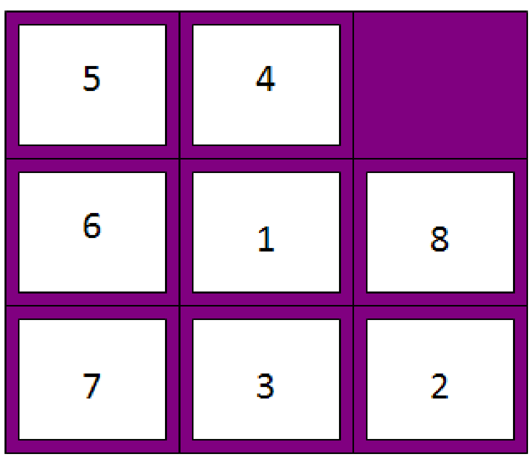
1	2	3
8		4
7	6	5

Goal State

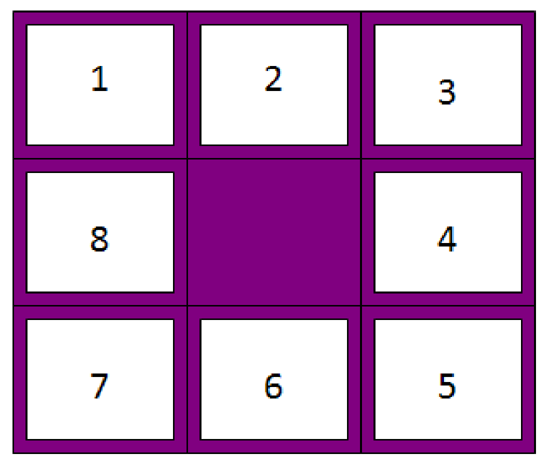
Which move is better?

Heuristic: # of incorrect tiles

- A: Move space Left
- B: Move space Down
- C: Both the same



Start State

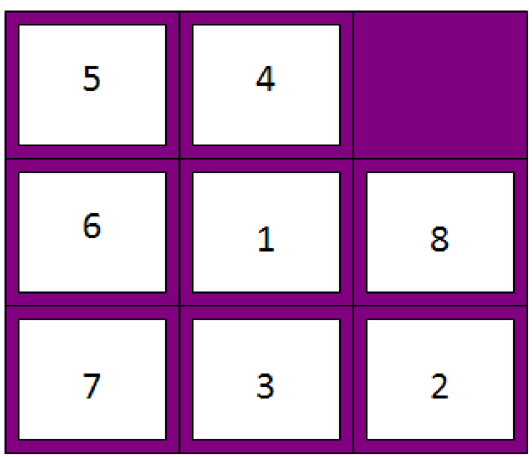


Goal State

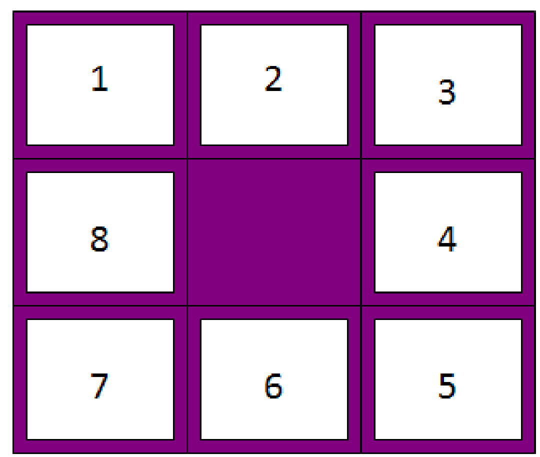
Which move is better?

Heuristic: Manhattan distance of tiles from correct location

- A: Move space Left
- B: Move space Down
- C: Both the same



Start State



Goal State

8-Puzzle

1. h_C = number of misplaced tiles
2. h_M = Manhattan distance

Are they admissible?

Which one should we use?

$$h_C \leq h_M \leq h^*$$

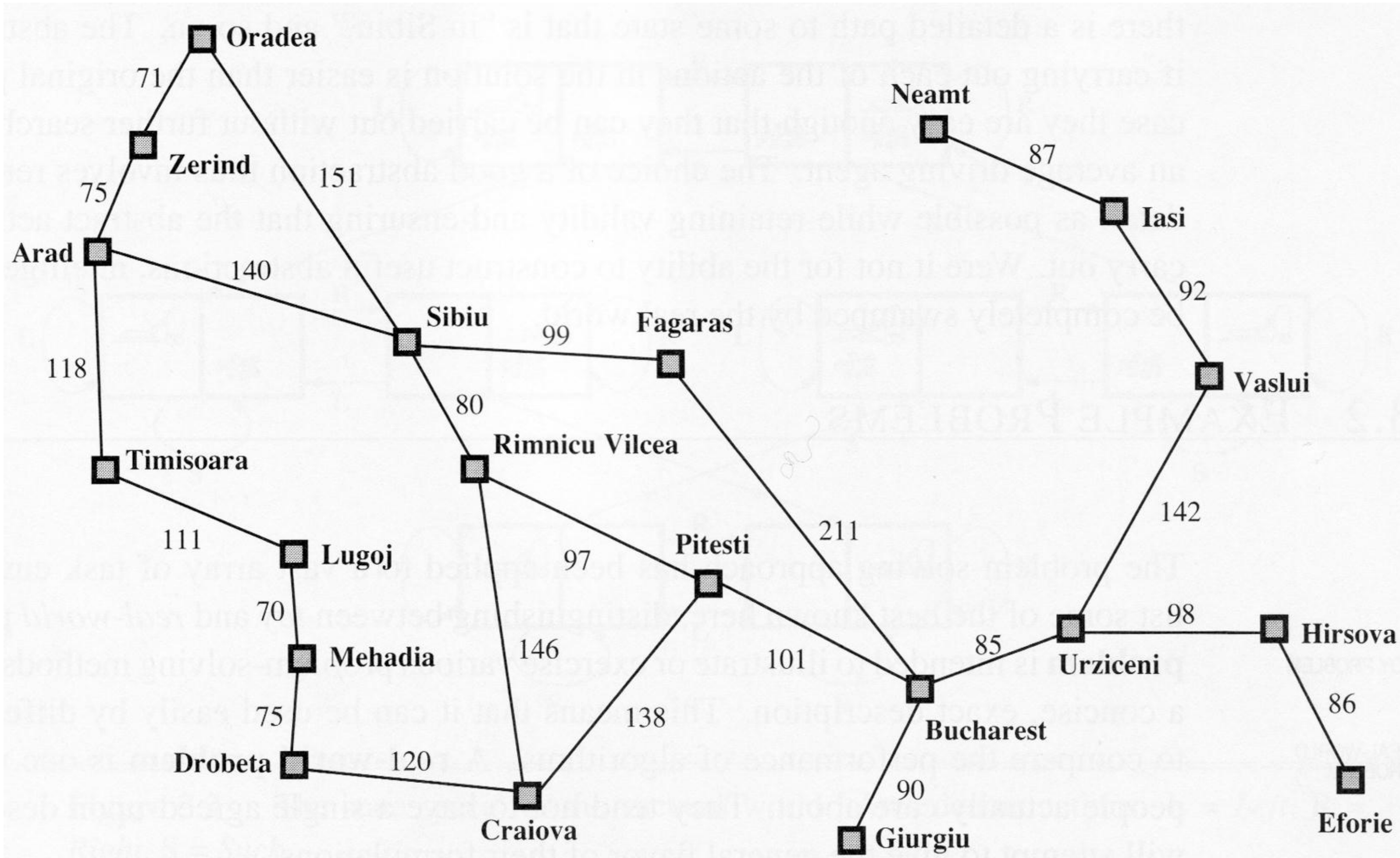
Search Costs on 8-Puzzle

h1: number of misplaced tiles
h2: Manhattan distance

		Search Cost		
d		IDS	$A^*(h_1)$	$A^*(h_2)$
2		10	6	6
4		112	13	12
6		680	20	18
8		6384	39	25
10		47127	93	39
12		364404	227	73
14		3473941	539	113
16		-	1301	211
18		-	3056	363
20		-	7276	676
22		-	18094	1219
24		-	39135	1641

Route Planning

Greedy Best-first search

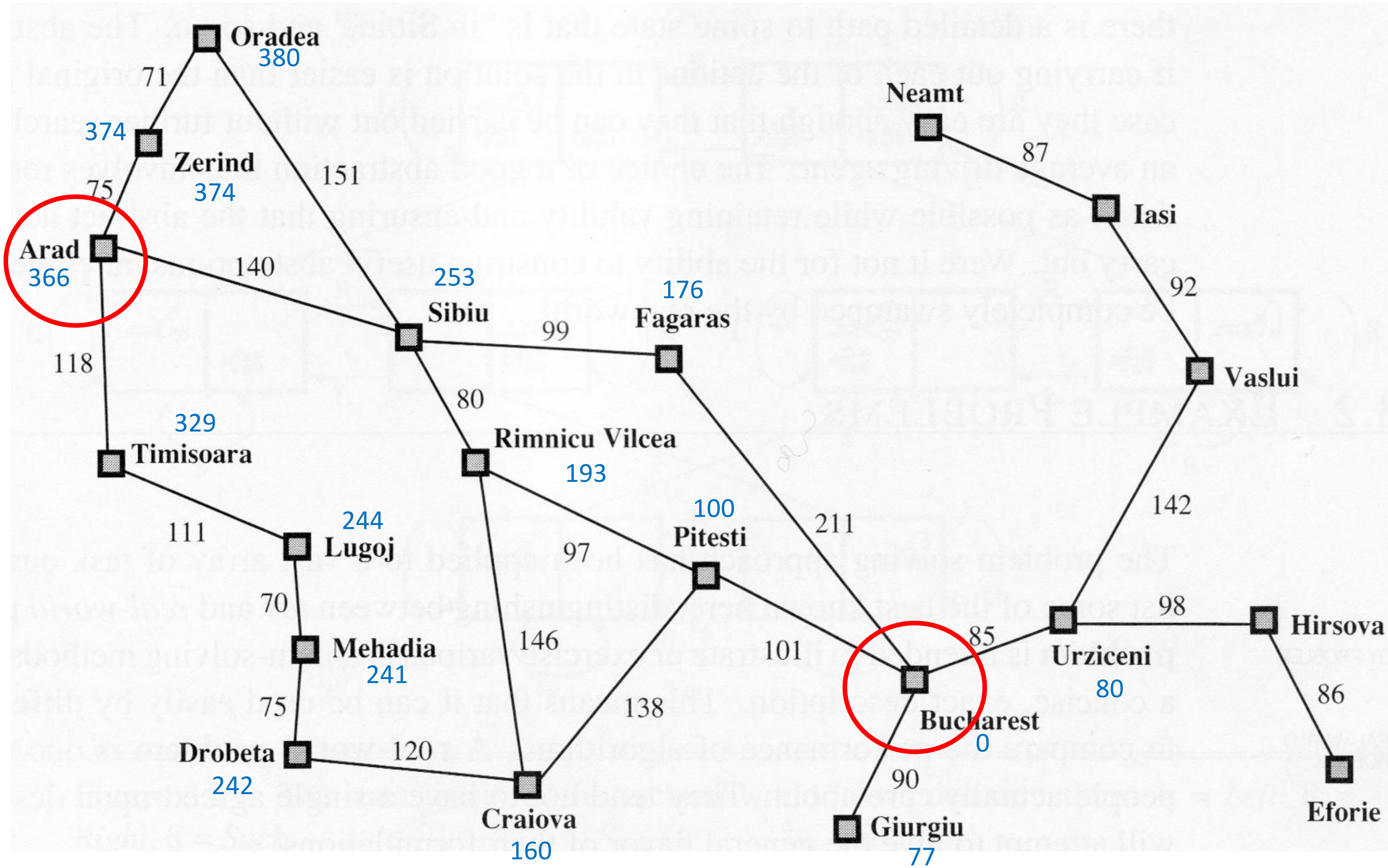


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Straight line distances to Bucharest

Route Planning

Greedy Best-first search



Blue – Straight line distance to Bucharest

A* Search

Idea: Use total estimated solution cost:

$g(n)$: Cost of reaching node $\underline{n}$ from initial node

$h(n)$: Estimated cost from node n to nearest goal

A* evaluation function: $f(n) = g(n) + h(n)$
→ $f(n)$ is estimated cost of cheapest solution through n.

Admissibility

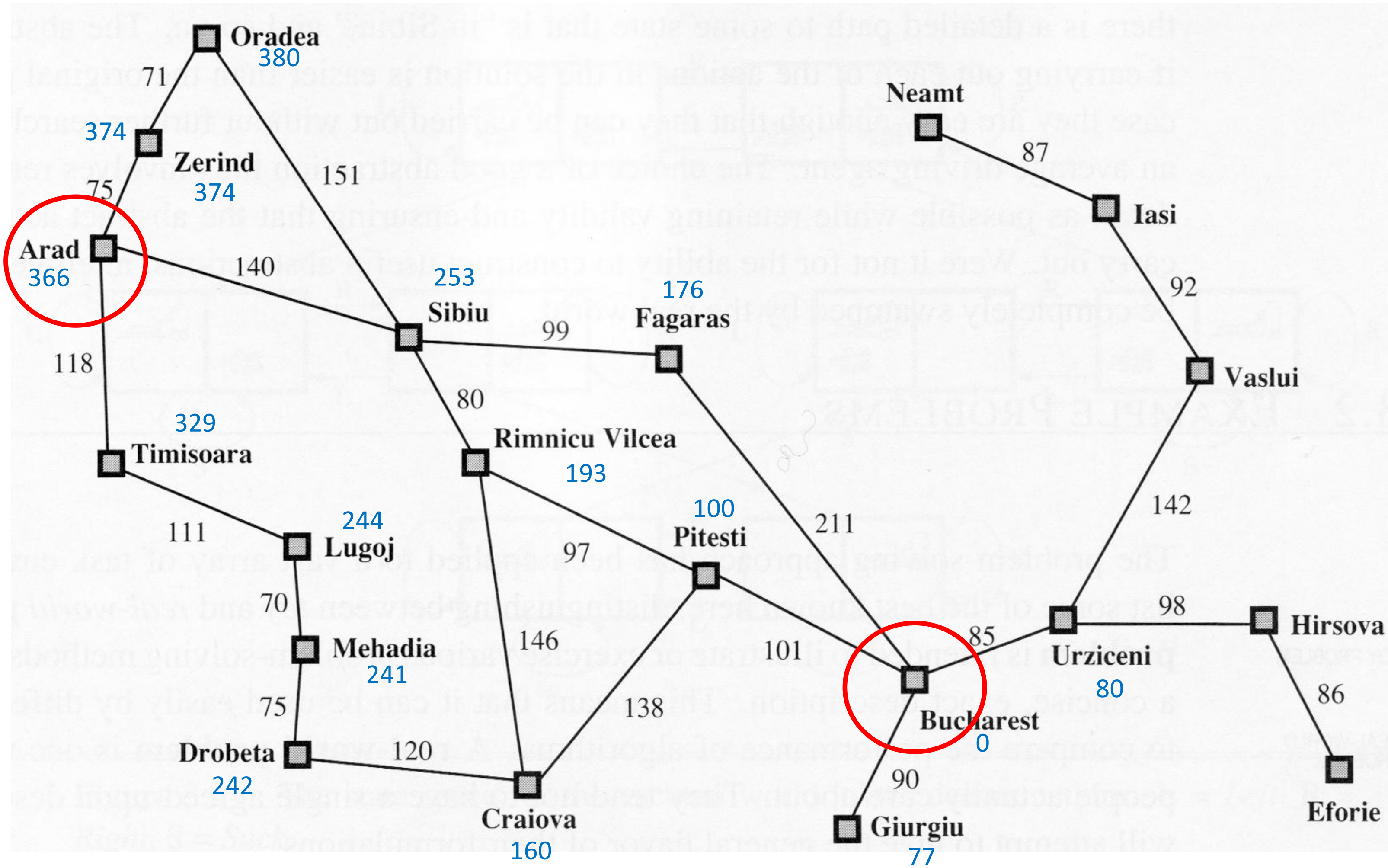
$h^*(n)$ *Actual* cost to reach a goal from n.

Definition: A heuristic function h is **optimistic** or **admissible** if $h(n) \leq h^*(n)$ for all nodes n. (h **never overestimates** the cost of reaching the goal.)

Theorem: If h is admissible, then the A* algorithm will never return a suboptimal goal node.

Route Planning

A*



Blue – Straight line distance to Bucharest

Proof of the optimality of A*

Assume: *h* admissible; *f* non-decreasing along any path.

Proof:

Assume C* is cost of optimal solution, G_2 is suboptimal goal (so $h(G_2)=0$)

$f(G_2)=g(G_2)+h(G_2)=g(G_2) > C^*$

Assume node *n* is some node on the optimal path

$f(n)=g(n)+h(n) \leq C^*$

So $f(n) \leq C^* < f(G_2)$ so *n* will always be expanded before G_2.

A*

Optimal: yes

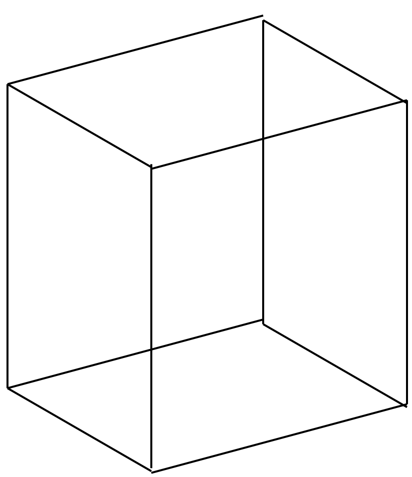
Complete: Unless there are infinitely many nodes with
f(n)<f*.
Assume locally finite:
(1) finite branching, (2) every operator costs at least $\delta > 0$

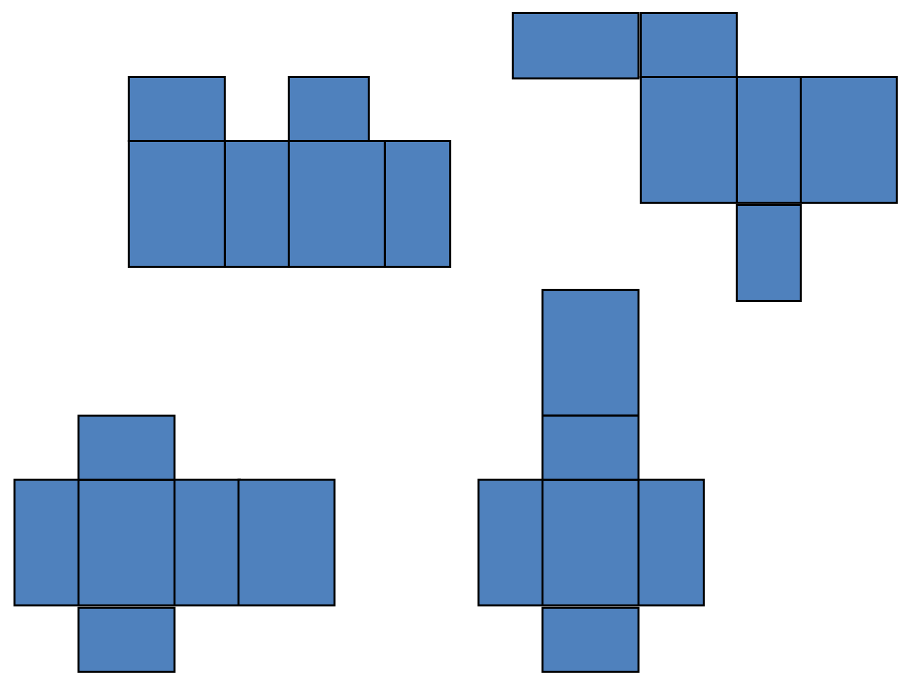
Complexity (time and space): Still exponential because of breadth-first nature. Unless $|h(n) - h^*(n)| \leq O(\log(h^*(n)))$, with h^* true cost of getting to goal.

A* is optimally efficient: given the information in *h*, no other optimal search method can expand fewer nodes.

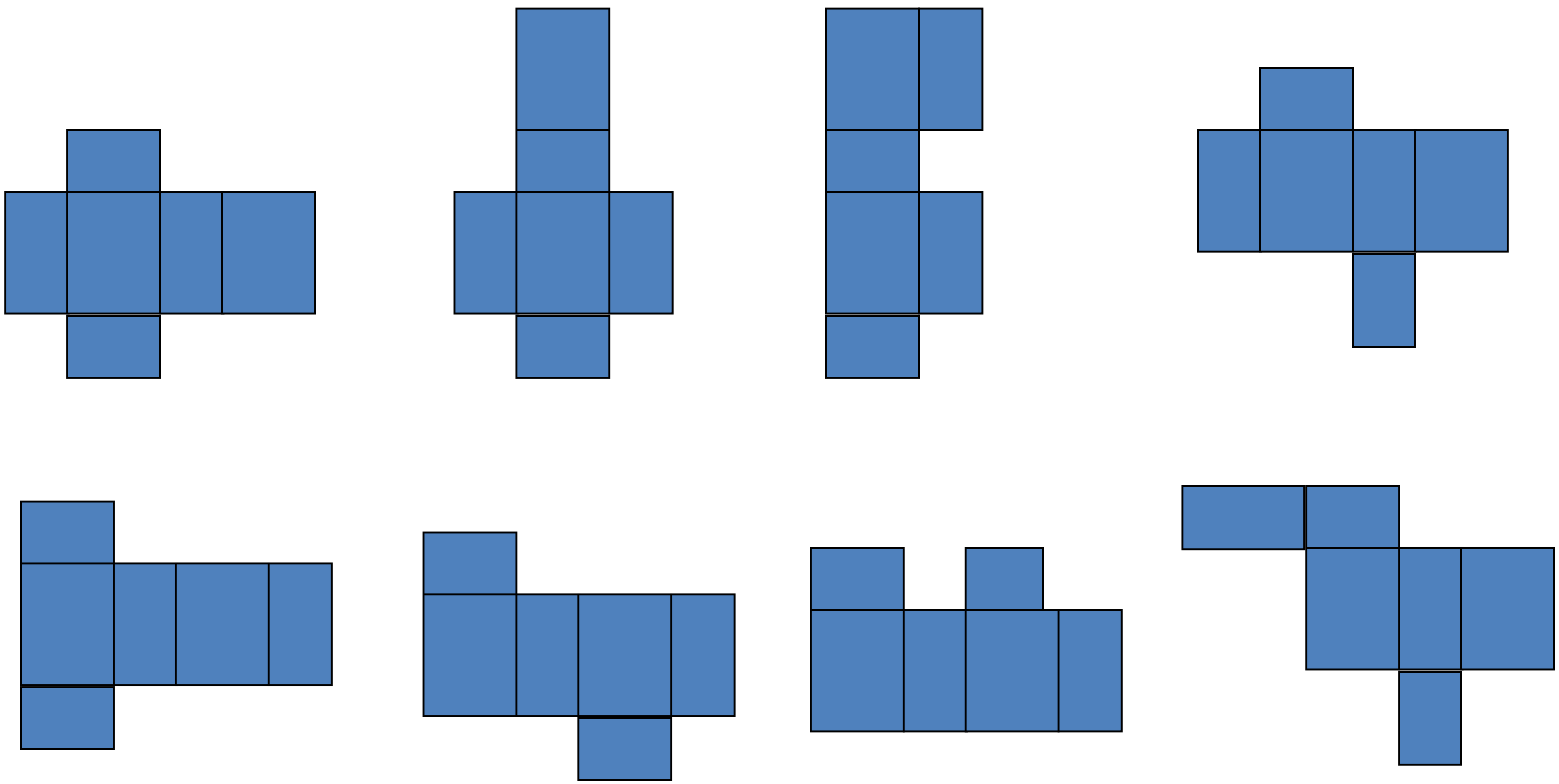
Example:
Optimal flat pattern problem

Find the flat pattern that minimizes total welding length



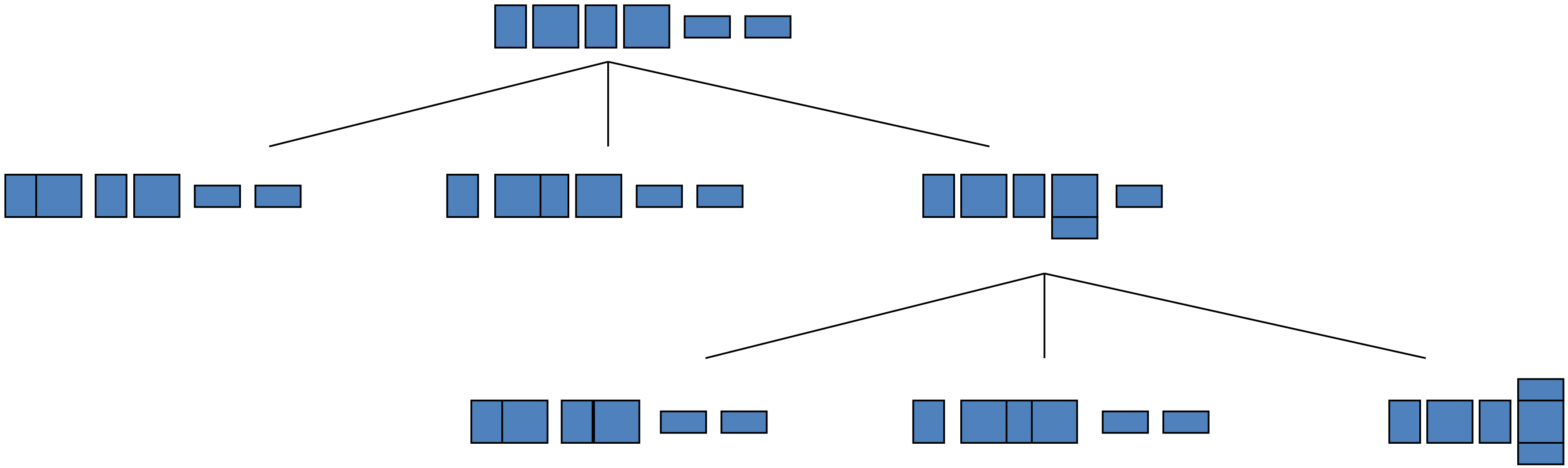


Permutation space



Lots more…

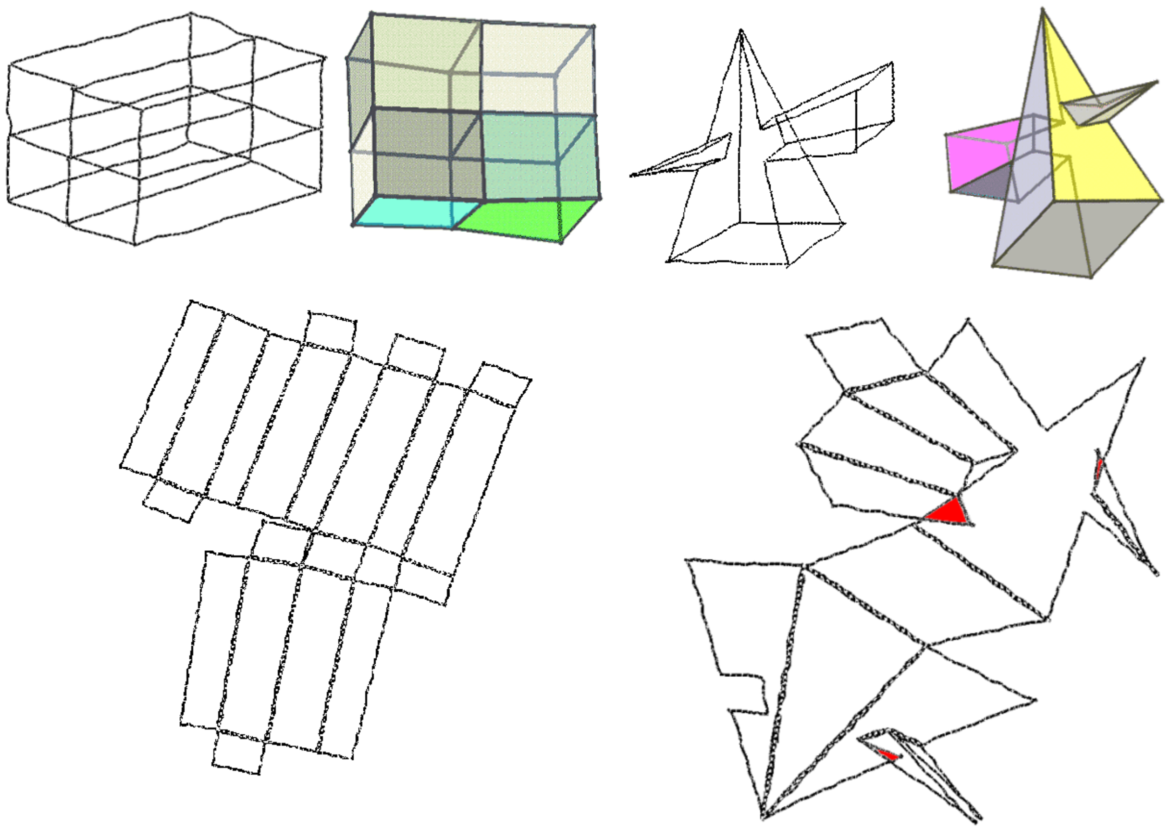
Searching the permutation space



• **Searching routes**: Reduce travel distance. Heuristic estimates remaining min distance (optimistic: does not *overestimate* remaining distance)

• **Searching flat patterns**: Reduce welding length. Heuristic estimates max savings (optimistic: does not *underestimate* remaining savings)

Finding optimal unfolding



Effective Branching Factor

$N=1+b^*+(b^*)^2+\ldots+(b^*)^d = (1- b^{d+1})/(1-b)$
$b^* =\sim N^{1/d}$

		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

Branching factor of chess is about 35

IDA*

Memory is a problem for the A* algorithms.

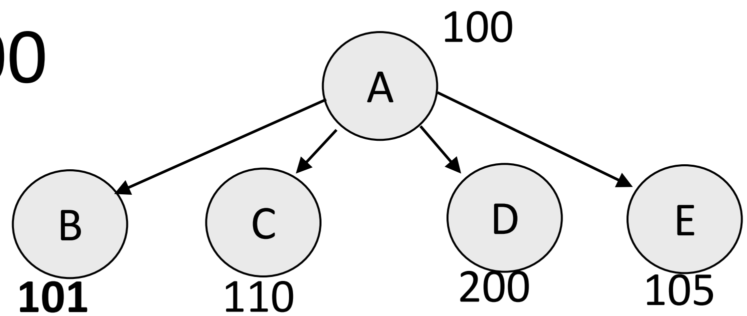
IDA* is like iterative deepening, but uses an *f*-cost limit rather than a depth limit.

At each iteration, the cutoff value is the smallest *f*-cost of any node that exceeded the cutoff on the previous iteration.

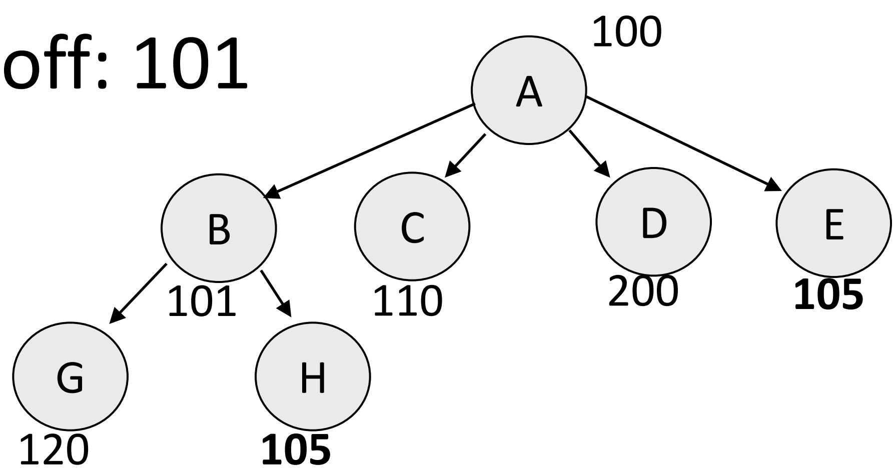
Each iteration uses conventional depth-first search.

Example: IDA*

- Initial state: A, f=100
- Cutoff: 100



- Cutoff: 101



- Cutoff: 105

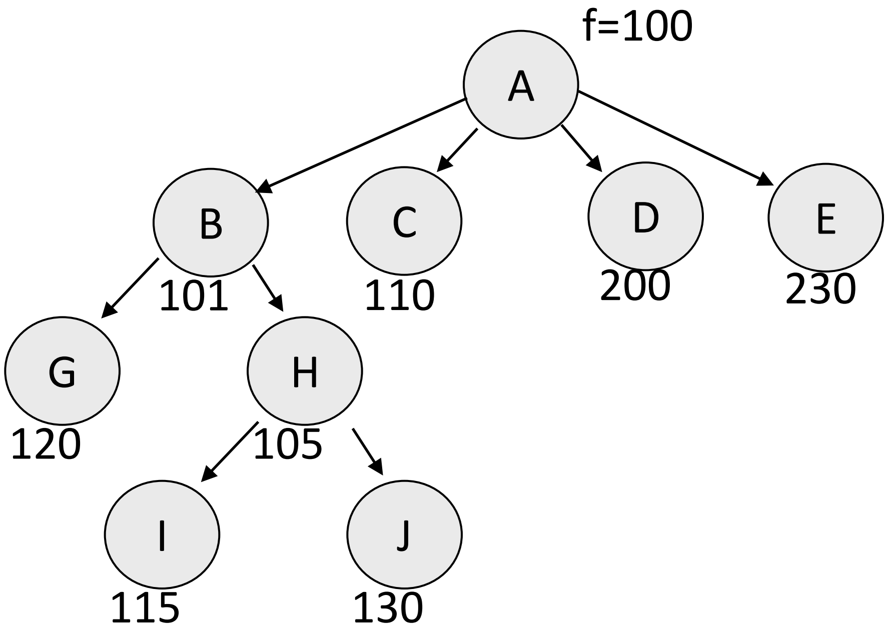
Recursive best-first search (RBFS)

Similar to a DFS, but keeps track of the *f*-value of the best alternative path available from any ancestor of the current node.

If current node exceeds this limit, recursion unwinds back to the alternative path, replacing the *f*-value of each node along the path with the best (highest, most accurate estimate) *f*-value of its children.

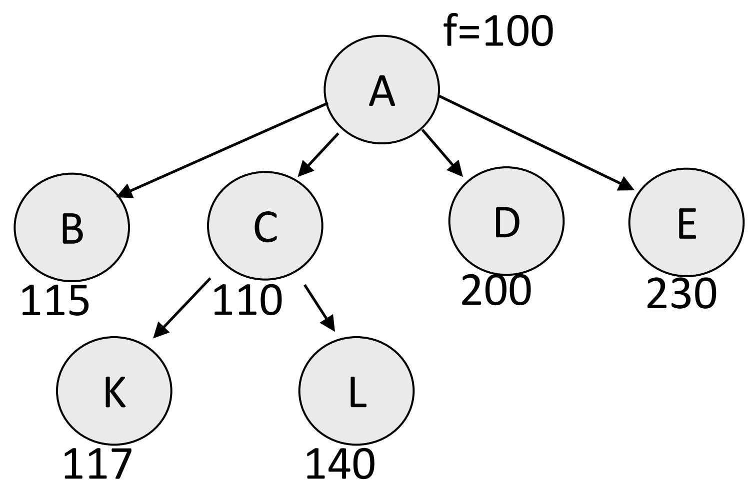
(RBFS remembers the *f*-value of the best leaf in the forgotten subtree.)

Example: RBFS



L=[(C,110), (I,115), (G,120), (J,130), (D,200), (E,230)]

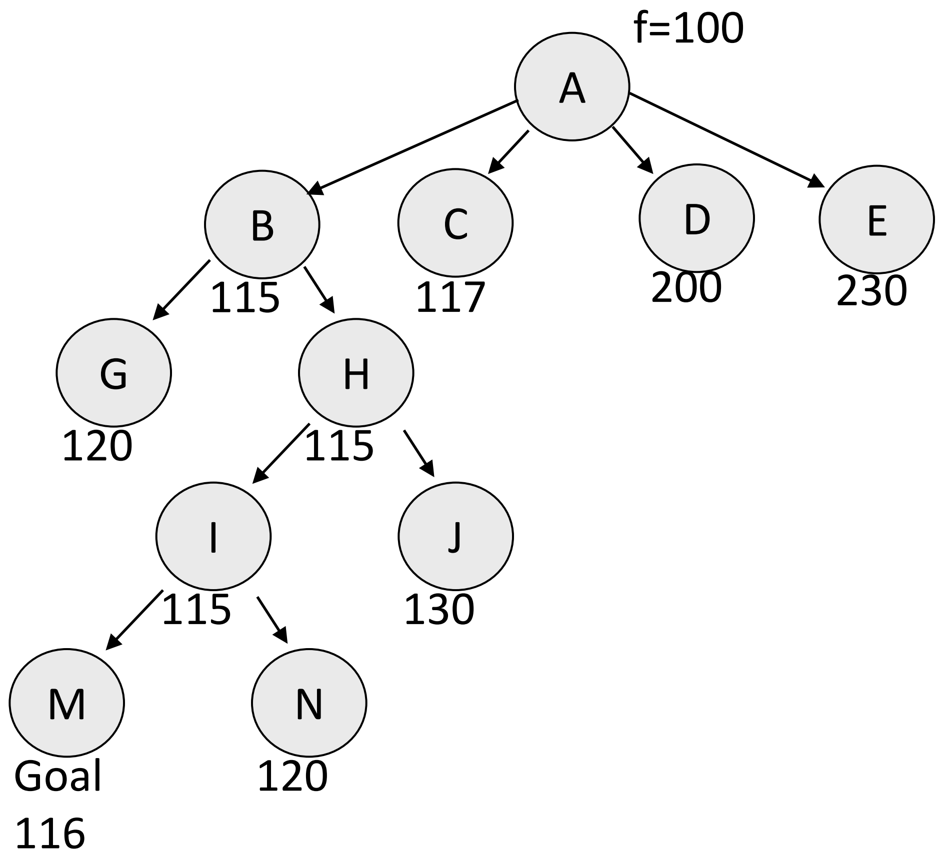
L=[(C,110), (B,115), (D,200), (E,230)]



L=[(B,115), (K,117), (L,140), (D,200), (E,230)]

L=[(B,115), (C,117), (D,200), (E,230)]

Example: RBFS (continued)



SMA*

Simplified Memory-Bounded A* Search:

- While memory available, proceeds just like A*, expanding the best leaf.
- If memory is full, drops the worst leaf node - the one the highest *f*-cost; and stores this value in its parent node.

(Won't know which way to go from this node, but we will have some idea of how worthwhile it is to explore the node.)

Constructing Admissible Heuristics

Deriving admissible heuristics automatically

Combining Heuristics

- $h(n)=\max(h1(n),h2(n),\ldots,h_m(n))$

Relaxed problems

– A problem with less restrictions on its operators is called a relaxed problem.
– The optimal solution of the original problem is also a solution to the relaxed problem and must therefore be at least as expensive as the optimal solution to the relaxed problem

Relaxed problems

- A tile can move from square A to square B if **A is adjacent to B** and **B is blank**.
- A tile can move from square A to square B if **A is adjacent to B**.
- A tile can move from square A to square B if **B is blank**.
- A tile can move from square A to square B.

Sub-Problems

- Cost of solutions to sub-problems are admissible
  - Pattern databases can store exact of the problem.

*	2	4
*		*
*	3	1

Start State

	1	2
3	4	*
*	*	*

Goal State

Sub-Problems

- Cost of solutions to sub-problems are admissible
  - Pattern databases can store exact of the problem
  - 4-tiles more effective than Manhattan distance
    - 1000 factor reduction on 15-puzzle
  - Disjoint patterns can be added
    - 10,000 factor reduction on 15-puzzle
    - 1M factor for 24-puzzle

Learning from experience

- Learn from prior searches
- Use inductive learning methods
  - Calculate actual cost for 1000 random samples
  - x_1 = Manhattan distance
  - Discover that when $x_1(n)=5$, actual cost is 14
  - x_2 = Number of relatively-correct pairs
  - $h(n)=c_1*x_1(n)+c_2*x_2(n)$
  - Loose admissibility…