Artificial Intelligence
CE-417, Group 2
Computer Eng. Department
Sharif University of Technology

Fall 2020

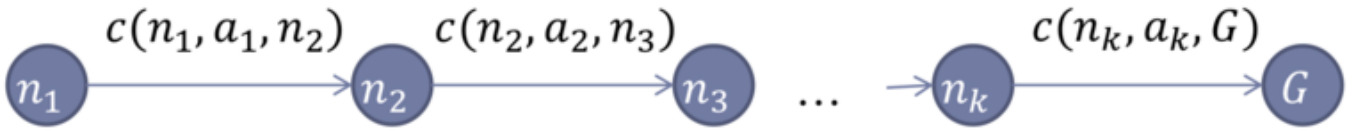
By Mohammad Hossein Rohban, Ph.D.

Courtesy: Most slides are adopted from CSE-573 (Washington U.), original slides for the textbook, and CS-188 (UC. Berkeley).

A* and
Heuristics and Pattern DB

Consistency implies admissibility

- Monotonic ⇒ Admissibility
  - All consistent heuristic functions are admissible
  - Nonetheless, most admissible heuristics are also consistent

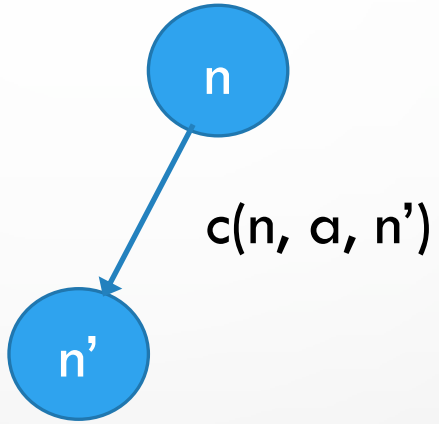


$$h(n_1) \le c(n_1, a_1, n_2) + h(n_2)$$
$$\le c(n_1, a_1, n_2) + c(n_2, a_2, n_3) + h(n_3)$$
$$\dots$$
$$\le \sum_{i=1}^{k} c(n_i, a_i, n_{i+1}) + h(G) \quad \Rightarrow h(n_1) \le \text{cost of (every) path from } n_1 \text{ to goal}$$
$$\le \text{cost of optimal path from } n_1 \text{ to goal}$$

Courtesy: Prof. Soleymani's slides – CE-417 – Spring 2018

How to make a heuristic function consistent?

- Take:



$$\bar{h}(n') = \max(h(n'), \bar{h}(n) - c(n, a, n'))$$

Heuristic dominance

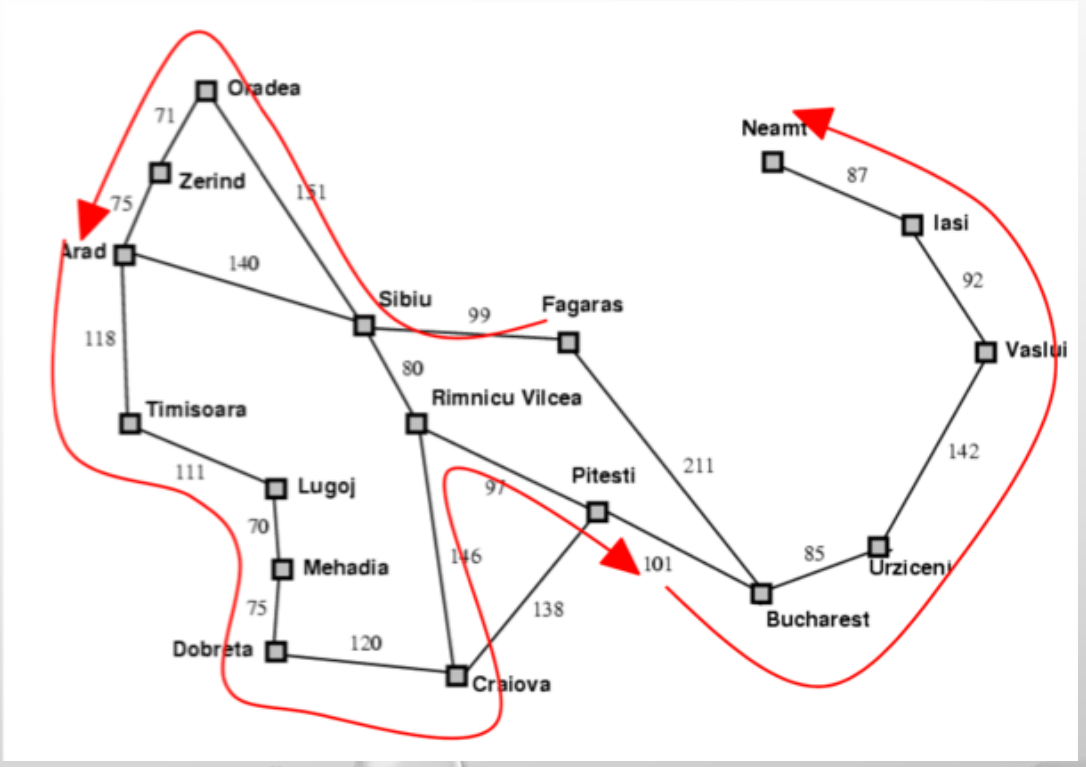
- Theorem: If h_2 dominates h_1, A* with h_2 expands at most the same number of nodes as when using h_1.
- A* opens all nodes with $f(n) < C$ and some with $f(n) = C$.
- Note that $h_1(n) \leq h_2(n)$:
- If node n is expanded by A* with h_2, $h_2(n) < C - g(n)$.
- $\Longrightarrow h_1(n) \leq h_2(n) < C - g(n) \Longrightarrow$ it would be expanded by A* with h_1 as well.

Make heuristics even better

- Lower gap between the heuristics function h and true remaining cost
  - lowers the number of expanded nodes
  - lowers running time of the search algorithm.
- Using simple heuristics (such as the Manhattan distance) is insufficient in certain cases:
  - 24-Puzzle takes 65000 years to be solved!

Recap: Relaxing the problem to define the heuristics Example 1: Traveling Salesman

- Input: undirected graph
- Output: connected path traversing each vertex ***exactly*** once
- As a search problem
  - States?
    - Graphs with partial paths
  - Operators?
    - Adding an edge to the path

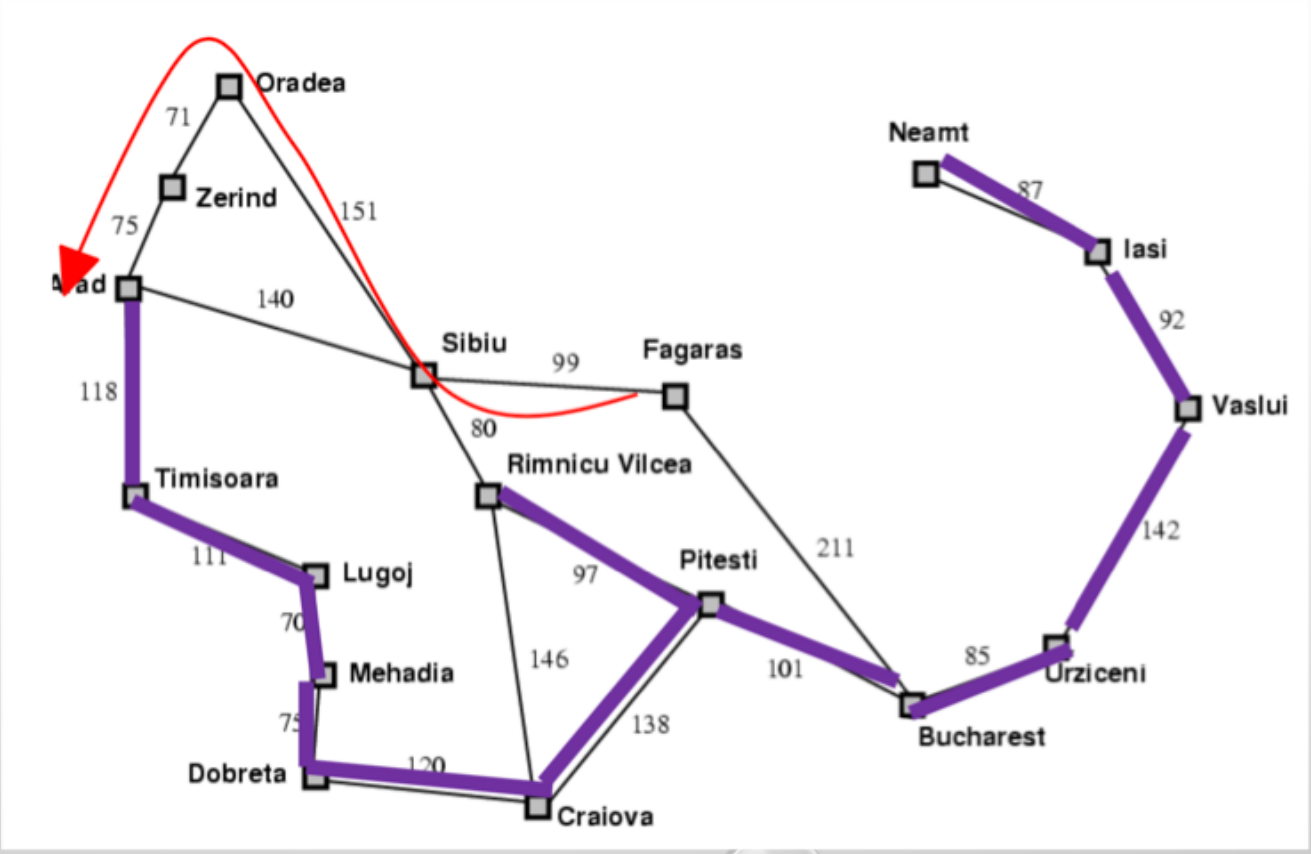


Recap: Relaxing the problem to define the heuristics Example 1: Traveling Salesman (cont.)

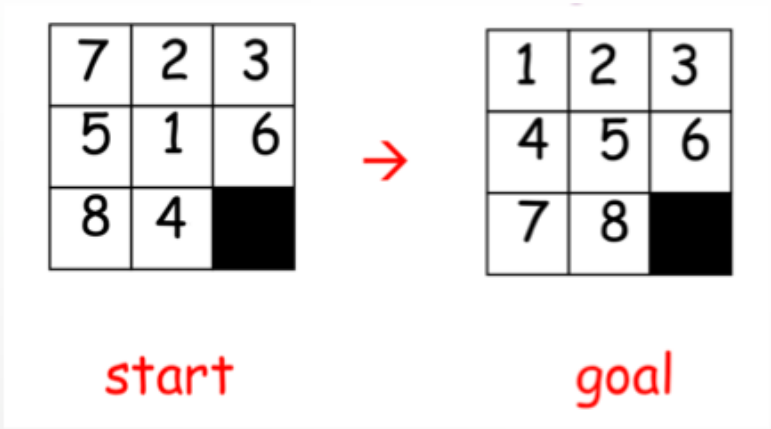
- Heuristic estimate of cost to complete a path?
  - What to relax?
  - What is a path?

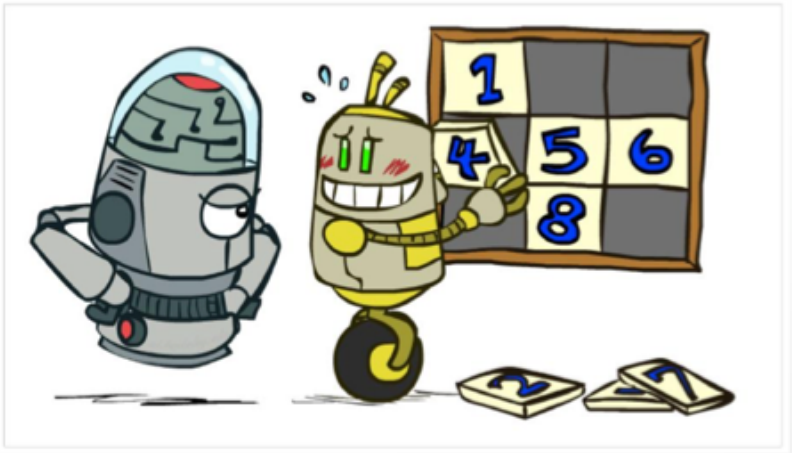
    Subgraph...

    Degree 2
  - Min spanning tree
    - $O(n^2)$



Recap: Relaxing the problem to define the heuristics
Example 2: 8-Puzzle (cont.)





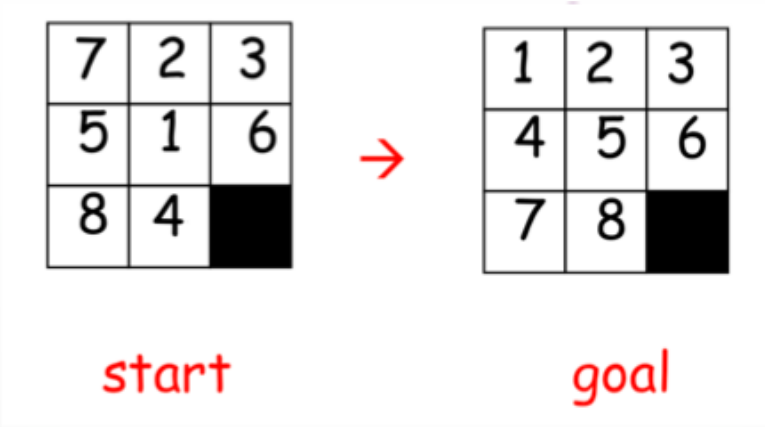
$$h^*(s_0) = \min_{n,a_1,\ldots,a_n,s_1,\ldots,s_n} n$$

$$\text{s.t. } s_n \in G, \qquad a_i \in A_{s_i}, \qquad s_i \in S, \qquad s_1 = s_0$$

$$s_{i+1} = succ(s_i, a_i)$$

$$A_{s_i} = \{(j,k) | 1 \le j \le 8, 1 \le k \le 9, j \in N_{empty}(s_i), k = I_{empty}(s_i)\}$$

Recap: Relaxing the problem to define the heuristics
Example 2: 8-Puzzle (cont.)



$h_1(n)$ = number of misplaced items in n

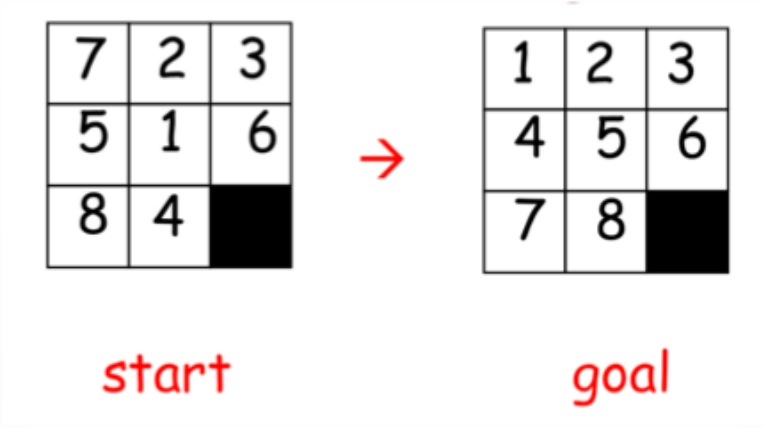
$$h(s_0) = \min_{n,a_1,\ldots,a_n,s_1,\ldots,s_n} n$$

$$\text{s.t. } s_n \in G, \qquad a_i \in A'_{s_i}, \qquad s_i \in S', \qquad s_1 = s_0$$

$$s_{i+1} = succ(s_i, a_i)$$

$$A'_{s_i} = \{(j,k) | 1 \le j \le 8, 1 \le k \le 9\}$$

Recap: Relaxing the problem to define the heuristics
Example 2: 8-Puzzle (cont.)



$h_2(n)$ = sum of Manhattan dist. of each tile to its correct location

$$h(s_0) = \min_{n,a_1,\ldots,a_n,s_1,\ldots,s_n} n$$

$$\text{s.t. } s_n \in G, \qquad a_i \in A''_{s_i}, \qquad s_i \in S'', \qquad s_1 = s_0$$

$$s_{i+1} = succ(s_i, a_i)$$

$$A''_{s_i} = \{(j,k) | 1 \le j \le 8, 1 \le k \le 9, k \in N(j)\}$$

Importance of Heuristics

h1 = number of tiles in wrong place

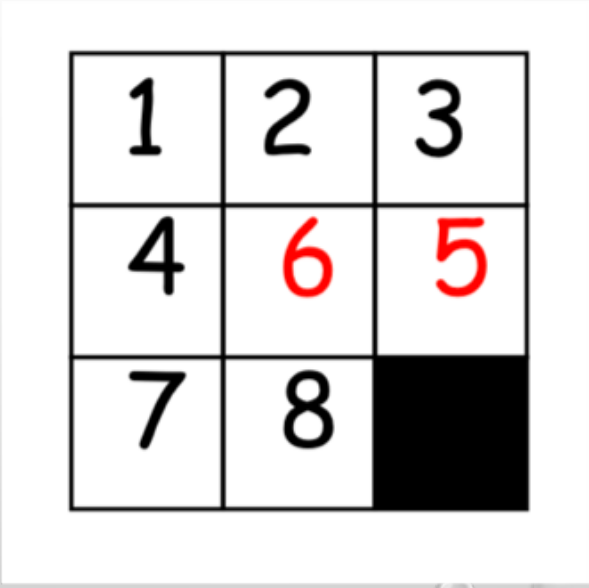
h2 = Σ distances of tiles from correct loc

D	IDS	A*(h1)	A*(h2)
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
18		3056	363
24		39135	1641

Decrease effective branching factor

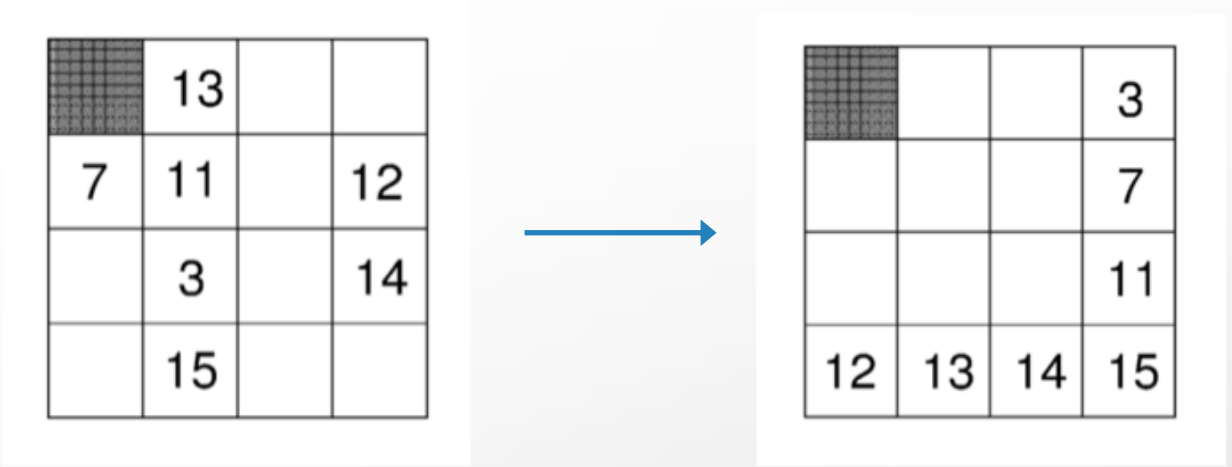
Sub-goal Interactions

- Manhattan distance assumes
  - Each tile can be moved independently of others
- Underestimates because
  - Doesn't consider interactions between tiles



Pattern Databases

- Pick any subset of tiles
  - e.g., 3, 7, 11, 12, 13, 14, 15
- Precompute a table
  - Optimal cost of solving just these tiles
  - For all possible configurations
    - 57 Million in this case
  - What search algorithm to use to fill the DB?
    - State = position of just these tiles (& blank)

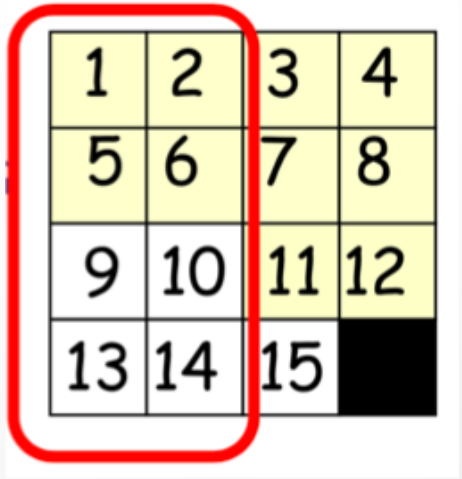


Using a Pattern Database

- As each state is generated
  - Use position of chosen tiles as index into DB
  - Use lookup value as heuristic, h(n)
- Admissible?
- Monotonic?

Combining Multiple Databases

- Can choose another set of tiles.
  - Precompute multiple tables
- How combine table values?
  - Min, Max, Sum?
- E.g. Optimal solutions to Rubik's cube
  - First found w/ IDA* using pattern DB heuristics
  - Multiple DBs were used (diff. cubie subsets)
  - Most problems solved optimally in 1 day
  - Compare with *574,000 years* for iterative deepening



Drawbacks of Standard Pattern DBs

- Since we can only take *max*
  - Diminishing returns on additional DBs
  - Consider bigger problem instances.
    - Subproblems should be small to be scalable.
    - If $h_i(n)$ from each database is at most x, then $\max_i h_i$ would be at most x.
- Would like to be able to ***add*** values

Disjoint Pattern DBs

- What if we make patterns be disjoint sets?
  - Can we take summation of heuristics by pattern DBs as an admissible heuristic?
- How to fix this?
  - Take number of moves made to the specified tiles as h_i instead.
- Why does summation result in an admissible heuristic in that case?

$$h^*(s_0) = \min_{n,a_1,\ldots,a_n,s_1,\ldots,s_n} n_1 + \cdots + n_m$$

$$\text{s.t. } s_n \in G, \qquad a_i \in A_{s_i}, \qquad s_i \in S, \qquad s_1 = s_0$$

$$s_{i+1} = succ(s_i, a_i)$$

$$n_i = |\{a_k | s_k(a_k(1)) \in P_i\}|$$

$$A_{s_i} = \{(j,k) | 1 \le j \le 8, 1 \le k \le 9, j \in N_{empty}(s_i), k = I_{empty}(s_i)\}$$

$$h^*(s_0) \geq \min_{n,a_1,\ldots,a_n,s_1,\ldots,s_n,} n_1 + \cdots + \min_{n,a_1,\ldots,a_n,s_1,\ldots,s_n} n_m$$

$$\text{s.t. } s_n \in G, \qquad a_i \in A_{s_i}, \qquad s_i \in S, \qquad s_1 = s_0$$

$$s_{i+1} = succ(s_i, a_i)$$

$$n_i = |\{a_k | s_k(a_k(1)) \in P_i\}|$$

$$A_{s_i} = \{(j,k) | 1 \leq j \leq 8, 1 \leq k \leq 9, j \in N_{empty}(s_i), k = I_{empty}(s_i)\}$$

$$h^*(s_0) \geq \min_{n,a_1,\ldots,a_n,s_1,\ldots,s_n,s_n \in G_1} n_1 + \cdots + \min_{n,a_1,\ldots,a_n,s_1,\ldots,s_n,s_n \in G_m} n_m$$

$$\text{s.t.} \quad a_i \in A_{s_i}, \qquad s_i \in S, \qquad s_1 = s_0$$

$$s_{i+1} = succ(s_i, a_i)$$

$$n_i = |\{a_k | s_k(a_k(1)) \in P_i\}|$$

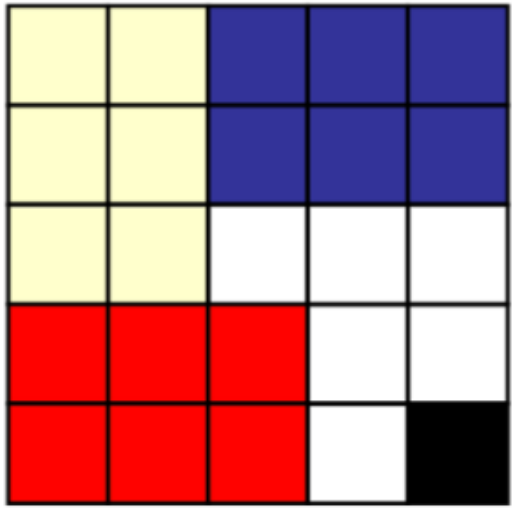
$$A_{s_i} = \{(j,k) | 1 \leq j \leq 8, 1 \leq k \leq 9, j \in N_{empty}(s_i), k = I_{empty}(s_i)\}$$

Disjoint Pattern DBs (cont.)

- Manhattan dist. is a trivial example of a disjoint DBs, where each group contains only a single tile. Why?
- As a general rule, when partitioning the tiles, we want to group together tiles that are near each other in the goal state, since these tiles will **interact the most** with one another.

Performance

- **15 Puzzle**: 2000x speedup *vs* Manhattan dist.
  - IDA* with the two DBs shown previously solves 15 Puzzles optimally in 30 milliseconds.
- **24 Puzzle**: 12 million x speedup *vs* Manhattan
  - IDA* can solve random instances in 2 days.
  - Requires 4 DBs as shown
    - Each DB has 128 million entries
  - Without PDBs: 65,000 years



Alternative Approach...

- Optimality is nice to have, but...
- Sometimes space is too vast!
  - Find suboptimal solution using local search.