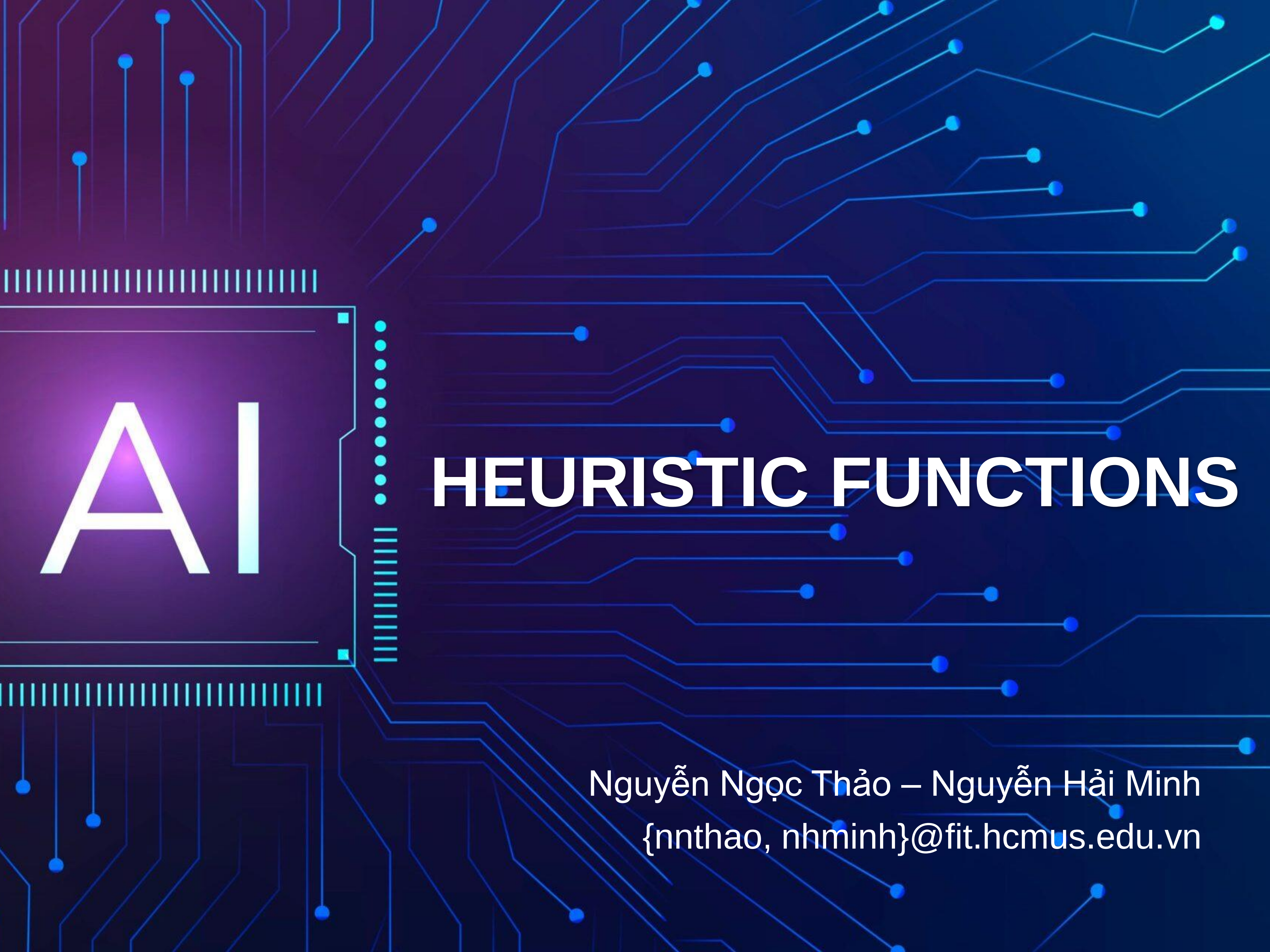
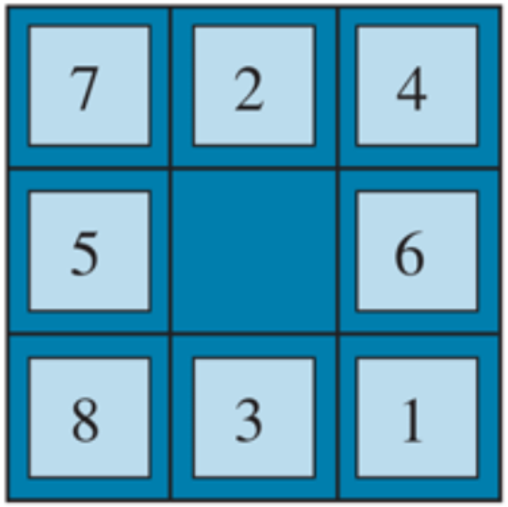
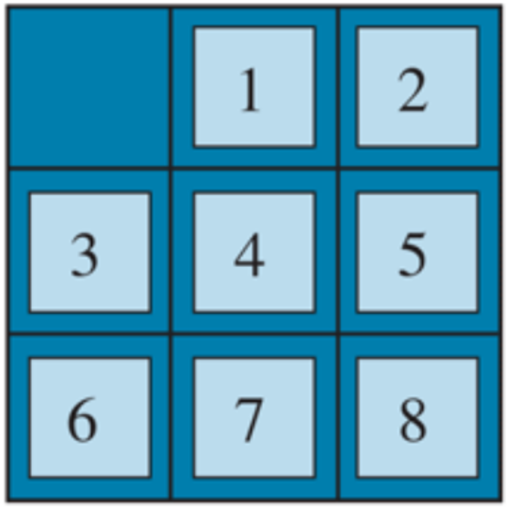


HEURISTIC FUNCTIONS

Nguyễn Ngọc Thảo – Nguyễn Hải Minh
{nnthao, nhminh}@fit.hcmus.edu.vn

The 8-puzzle problem





Start state

Goal state

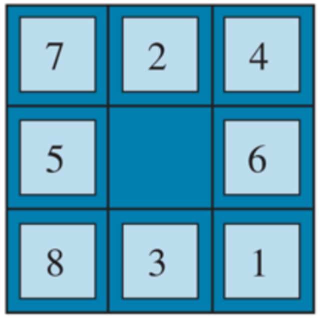
A typical instance of the 8-puzzle. The solution is 26 steps long.

- Average solution cost: about 22 steps, branching factor ~ 3.
- 15-puzzle: 16! / 2—over 10 trillion states
- 24-puzzle: around 10^{25} states

Admissible heuristics for 8-puzzle

- *$h_1(n)$ = number of misplaced numbered tiles (Hamming distance).*

Hamming distance

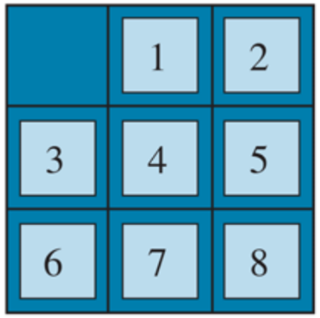


Start state

1	2	3	4	5	6	7	8	8
✓	✓	✓	✓	✓	✓	✓	✓	

1	2	3	4	5	6	7	8	18
3	1	2	2	2	3	3	2	

Manhattan distance



Goal state

- *$h_2(n)$ = sum of the (Manhattan) distance of every numbered tile to its goal position.*

- Neither of these overestimates the true solution cost—26.

Quiz 03: Admissible heuristics

- For 8-puzzle, which of the following heuristics is admissible?

  - $h_1(n) = \text{total number of misplaced tiles}$
  - $h_2(n) = \text{total Manhattan distance}$
  - $h_3(n) = 0$
  - $h_4(n) = 1$
  - $h_5(n) = h^*(n)$
  - $h_6(n) = \min(2, h^*(n))$
  - $h_7(n) = \max(2, h^*(n))$

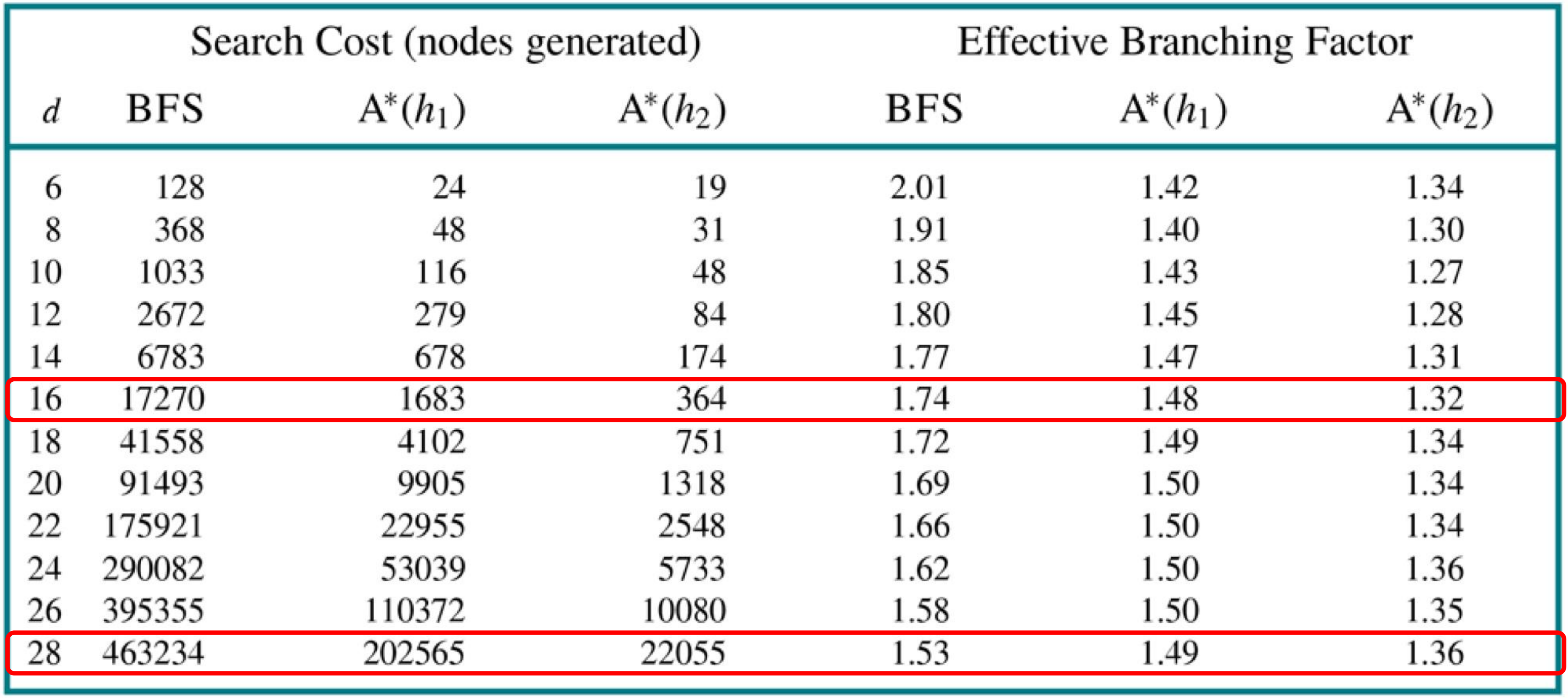
The effect of heuristic on performance

- One way to characterize the quality of a heuristic is the effective branching factor b^*.
- Let the total number of nodes generated by A* for a particular problem be N and the solution depth be d.
- $\boldsymbol{b}^*$ is the branching factor that a uniform tree of depth would have to have to contain $N + 1$ nodes.
- Thus, $\boldsymbol{N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d}$
  - E.g., A* finds a solution at depth 5 using 52 nodes $\rightarrow b^* = 1.92$

The effect of heuristic on performance

- $\boldsymbol{b}^*$ can vary across problem instances, but constant across all nontrivial instances in a specific domain.
- Experiments of $\boldsymbol{b}^*$ on a small set of problems can provide a good guide to the heuristic's overall usefulness.
- A well-designed heuristic would have a value of $\boldsymbol{b}^*$ close to 1, allowing large problems to be solved at reasonable cost.

Search cost vs. Branching factor



	Search Cost (nodes generated)			Effective Branching Factor		
d	BFS	$A^*(h_1)$	$A^*(h_2)$	BFS	$A^*(h_1)$	$A^*(h_2)$
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

Comparison of the search costs and effective branching factors for 8-puzzle problems using BFS, A* with misplaced tiles (h_1), and A* with Manhattan distance (h_2). Data are averaged over 100 puzzles for each solution length from 6 to 28.

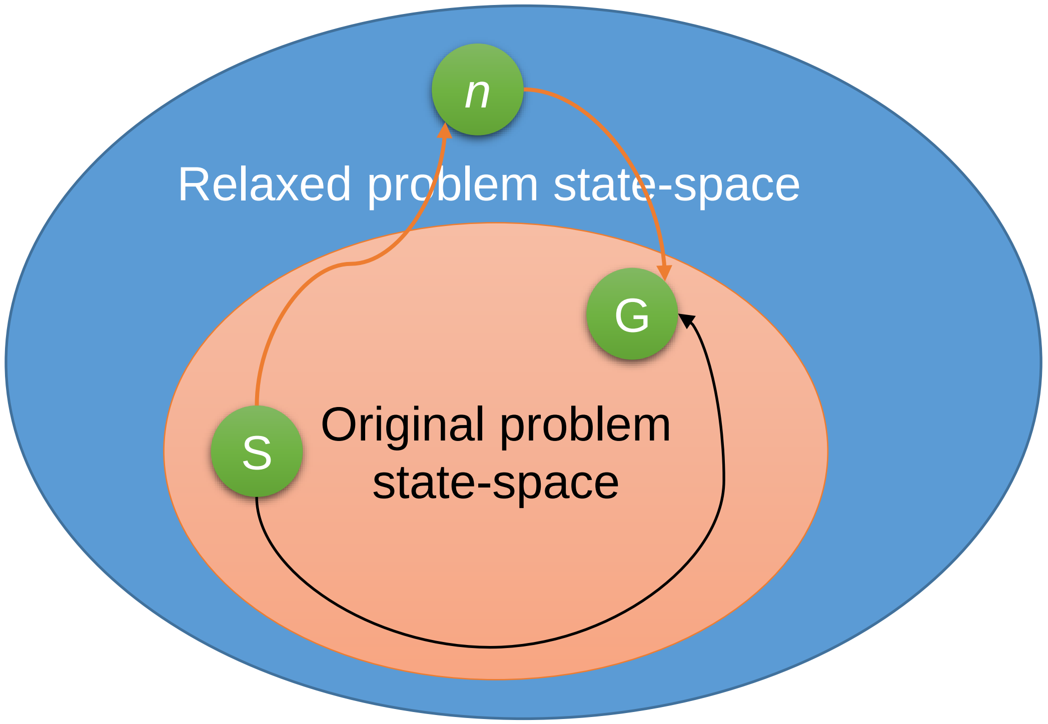
Heuristic dominance

- Consider two heuristics, h_1 and h_2.
- h_2 dominates h_1 if $h_2(n) \geq h_1(n)$, for all n.
- A* using h_2 will never expand more nodes than A* using h_1.
    - Assume that h_1 and h_2 are both consistent and $h_2(n) \geq h_1(n)$, $\forall n$.
    - For A*, every node with $f(n) < C^*$, i.e., $h(n) < C^* - g(n)$, will surely be expanded when h is consistent.
    - Thus, every node surely expanded by A* with h_2 is also surely expanded with h_1, and h_1 might cause other nodes to be expanded.
- A heuristic with higher values is more useful, provided it is consistent and its computational time is not too long.

Heuristics from relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem.

The state-space graph of the relaxed problem is a *supergraph* of the original state space.



- The cost of an optimal solution to a relaxed problem is a consistent heuristic for the original problem.

Heuristics from the relaxed 8-puzzle

A tile can move from square A to square B if A is horizontally or vertically adjacent to B and B is blank

- Relaxed problems are generated by removing one or both conditions
  - 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.

Manhattan distance

Misplaced tiles

Relaxed problems

- Relaxed problems generated by removing restrictions must be solved essentially without search.
- This allows the original problem to be decomposed into several independent subproblems.
  - E.g., decomposing the 8-puzzle problem yields eight independent subproblems, one per tile.
- If the relaxed problem is hard to solve, the values of the corresponding heuristic will be expensive to obtain.

Composite heuristics

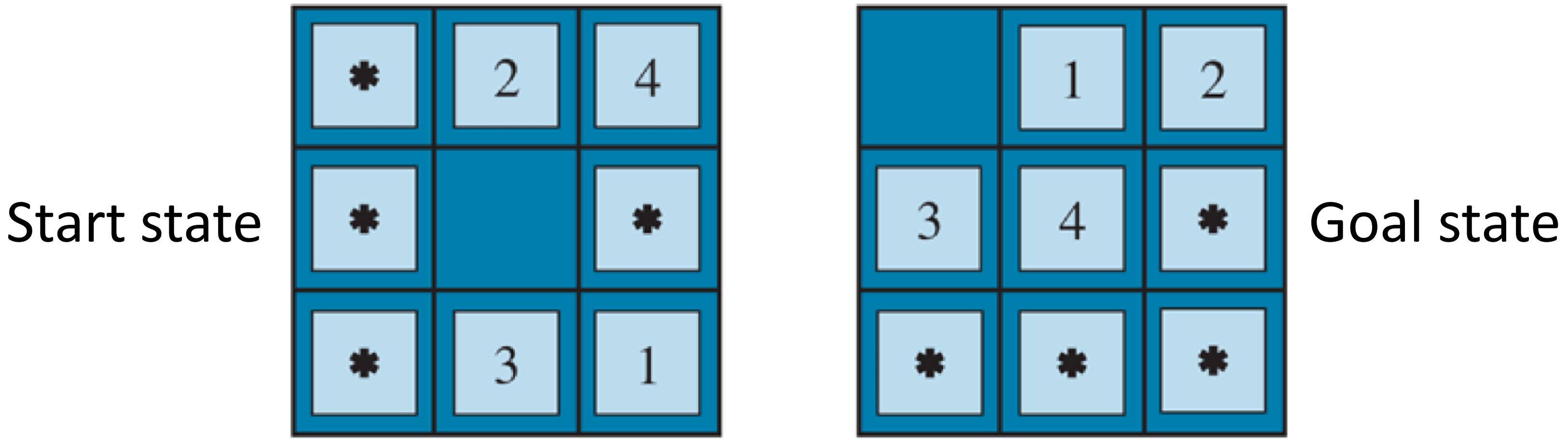
- Consider a collection of admissible heuristics for a problem, $\boldsymbol{h_1}, \boldsymbol{h_2},\ldots, \boldsymbol{h_m}$, none is clearly better than the others.
- The composite heuristic is defined as follows.

$$\boldsymbol{h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\}}$$

- This heuristic is consistent and dominates all component heuristics, yet it takes longer to compute.
  - Alternative: randomly select one of the heuristics at each evaluation or use a ML algorithm to predict which heuristic will be best.
  - This can result in an inconsistent heuristic, yet practically leading to faster problem solving.

Heuristics from subproblems

- Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem.
  - This cost is a lower bound on the cost of the complete problem.
- More accurate than Manhattan distance in some cases.



The task is to get tiles 1, 2, 3, 4, and the blank into their correct positions, without worrying about what happens to the other tiles.

Heuristics from subproblems

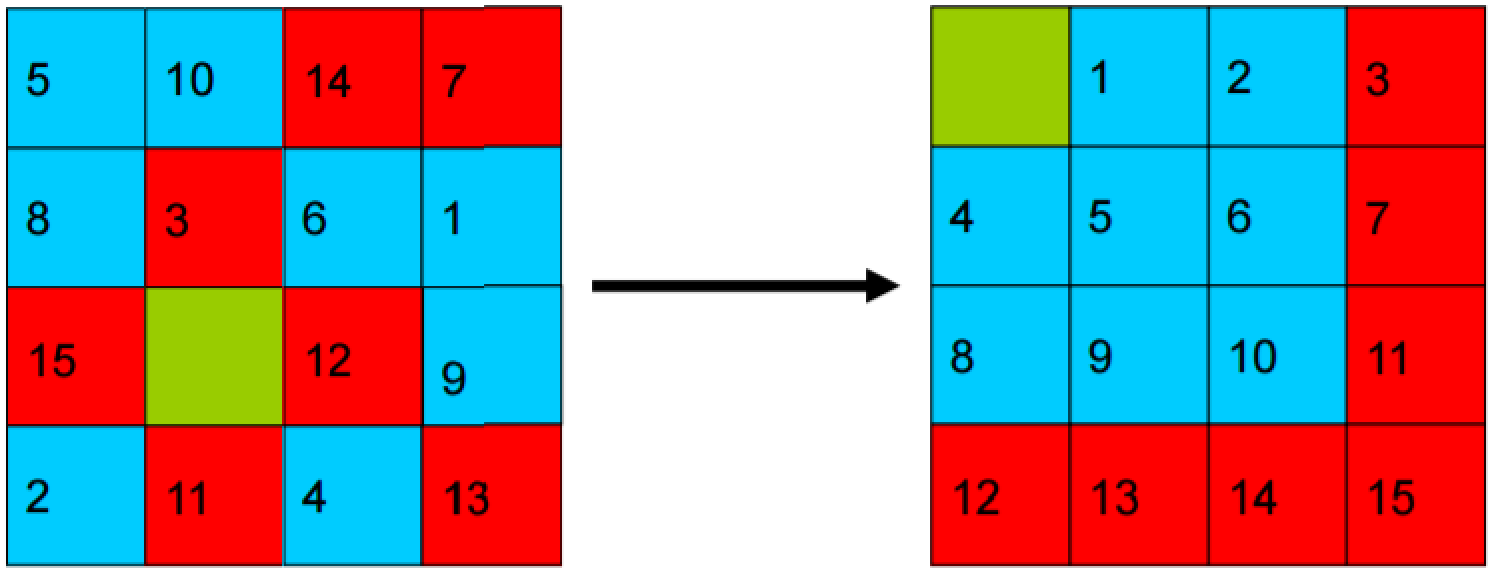
- Pattern databases (PDB) store the exact solution costs for every possible subproblem instances.
  - E.g., the previous 8-puzzle configuration has $9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 = 15{,}120$ patterns—every possible arrangement of the four tiles and the blank.
- The database is built by searching back from the goal and recording the cost of each new pattern encountered.
  - The search expense is amortized over subsequent instances, and thus it makes sense if we solve many problems.
- We determine heuristic values for states by referencing the relevant subproblem configuration in the database.

Publication on pattern databases

- Culberson, Joseph C., and Jonathan Schaeffer. "Pattern databases." Computational Intelligence 14.3 (1998): 318-334.
- Korf, Richard E., and Ariel Felner. "Disjoint pattern database heuristics." Artificial intelligence 134.1-2 (2002): 9-22.
- Felner, Ariel, Richard E. Korf, and Sarit Hanan. "Additive pattern database heuristics." *Journal of Artificial Intelligence Research* 22 (2004): 279-318.

Traditional pattern databases

- Heuristics can be combined by taking the maximum value.
- However, with each additional DB, there are diminishing returns and increased memory and computation costs.



31 moves needed to solve red tiles. 22 moves needed to solve blue tiles

→ Overall heuristic is the maximum value—31 moves

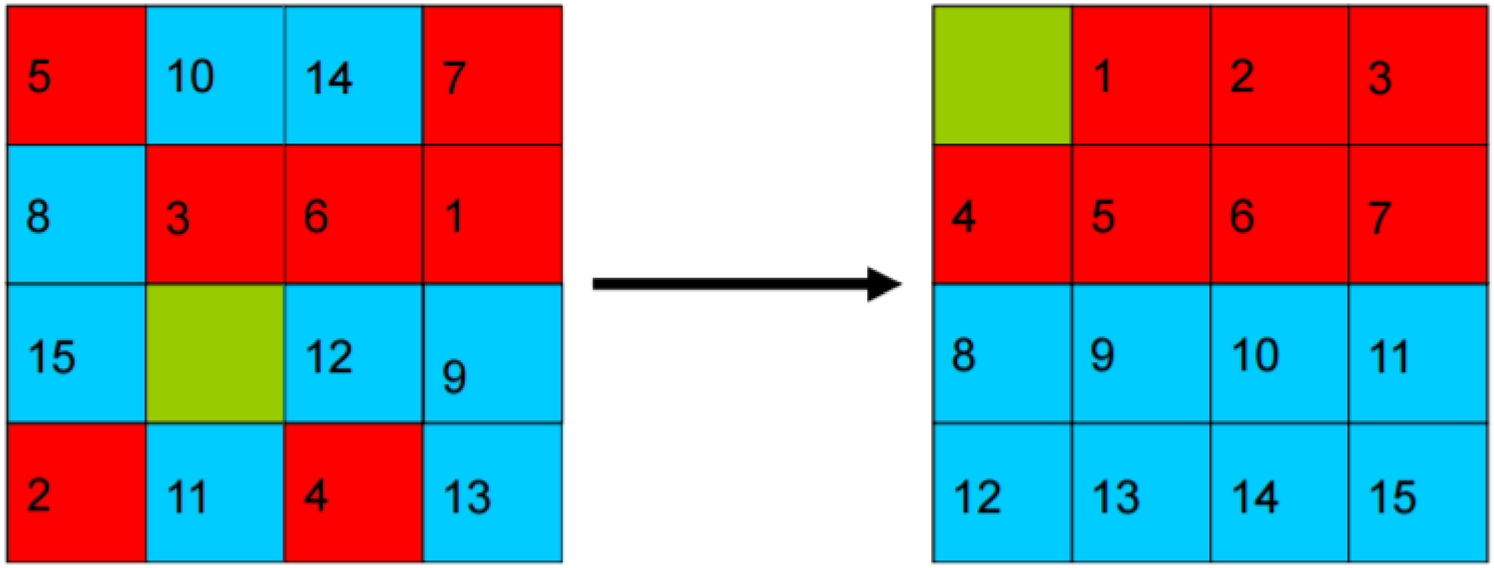
Image credit: CSE473 course

Disjoint pattern databases

- Assume that no tile belongs to more than one pattern.
- Disjoint pattern databases counts only moves of the pattern tiles while ignoring non-pattern moves.
  - That is, we record not the total cost of solving the 1-2-3-4 subproblem, but just the number of moves involving 1-2-3-4.
- It is possible to sum up the costs, which is still a lower bound on the cost of solving the entire problem.

Disjoint pattern databases

- During the search, we look up heuristic values for each set and add the values without overestimating.



20 moves needed to solve red tiles. 25 moves needed to solve blue tiles.

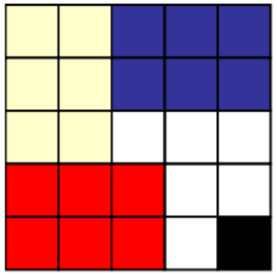
$\rightarrow$ Overall heuristic is 20 + 25 = 45 moves

Image credit: CSE473 course

Heuristics from subproblems

- We may organize multiple pattern databases, each of which is huge in size.
  - E.g., there are two disjoint pattern databases for 15-puzzles, one contains 58 million entries and the other has 519 entries.

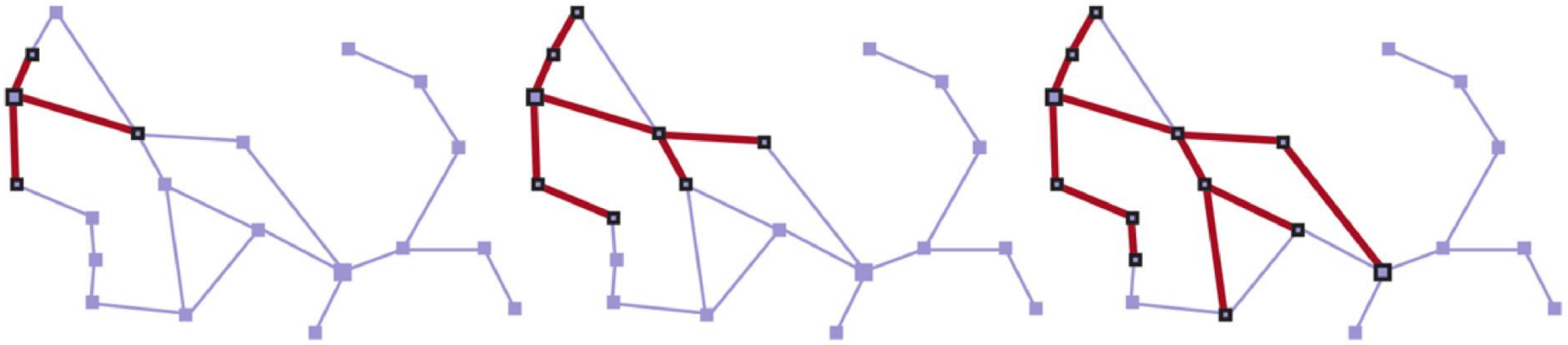
15-puzzle	• 2000× speedup vs. Manhattan distance • IDA* with the two DBs solves 15-puzzles optimally in 30 milliseconds
24-puzzle	• 12 million × speedup vs. Manhattan, without PDBs: 65,000 years • IDA* can solve random instances in 2 days. • Requires 4 DBs, each has 128 million entries



Learning to search better

- *Could an agent learn how to search better?* **YES**
- Each state in the metalevel state space captures the internal (computational) state of a program that is searching in an object-level state space.
  - The internal state of the search algorithm is the current search tree.
- An action is a computation step that alters the internal state.
  - E.g., expands a leaf node and adds its successors to the tree

Learning to search better



The figure depicts a path in the metalevel state space where each state on the path (subfigure) is an object-level search tree.

- The expansion of Fagaras is not helpful $\rightarrow$ harder problems may even include more such missteps.
- A metalevel learning algorithm gains from these experiences to avoid exploring unpromising subtrees $\rightarrow$ reinforcement learning.

Learning heuristics from experience

- Experience means solving a lot of instances of a problem.
    - E.g., solving lots of 8-puzzles
- Each optimal solution to a problem instance provides examples from which $h(n)$ can be learned
- Learning algorithms
    - Neural nets
    - Decision trees
    - Inductive learning
    - …

...the end.