Informed search algorithms

Chapter 3

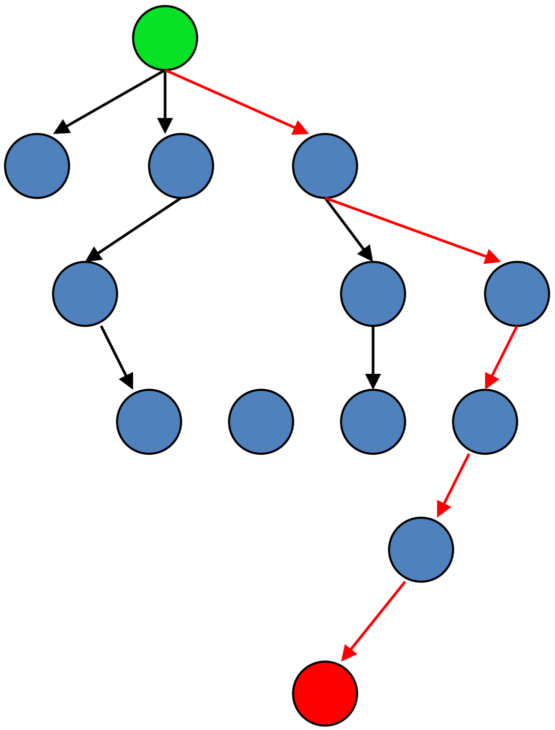
(Based on Slides by Stuart Russell, Richard Korf, Subbarao Kambhampati, and UW-AI faculty)

“Intuition, like the rays of the sun, acts only in an inflexibly straight line; it can guess right only on condition of never diverting its gaze; the freaks of chance disturb it.”

-- Honore de Balzac

Informed (Heuristic) Search

Idea: be **smart** about what paths to try.



Blind Search vs. Informed Search

- What's the difference?

- How do we formally specify this?

  A node is selected for expansion based on an evaluation function that estimates cost to goal.

General Tree Search Paradigm

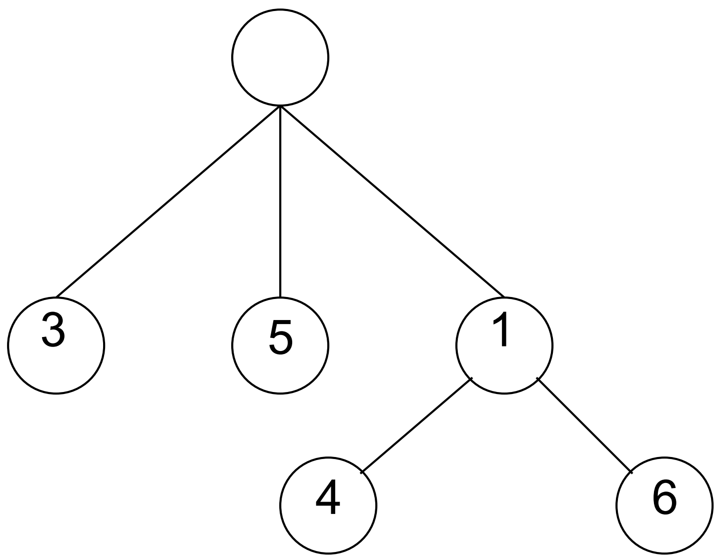
```
function tree-search(root-node)
  fringe ← successors(root-node)
  while ( notempty(fringe) )
        {node ← remove-first(fringe) //lowest f value
            state ← state(node)
            if goal-test(state) return solution(node)
            fringe ← insert-all(successors(node),fringe) }
  return failure
end tree-search
```

General Graph Search Paradigm

```
function tree-search(root-node)
  fringe ← successors(root-node)
  explored ← empty
  while ( notempty(fringe) )
       {node ← remove-first(fringe)
          state ← state(node)
          if goal-test(state) return solution(node)
          explored ← insert(node,explored)
          fringe ← insert-all(successors(node),fringe, if node not in explored)
       }
  return failure
end tree-search
```

Best-First Search

- Use an evaluation function f(n) for node n.
- Always choose the node from fringe that has the lowest f value.



Best-first search

- A search strategy is defined by picking the order of node expansion

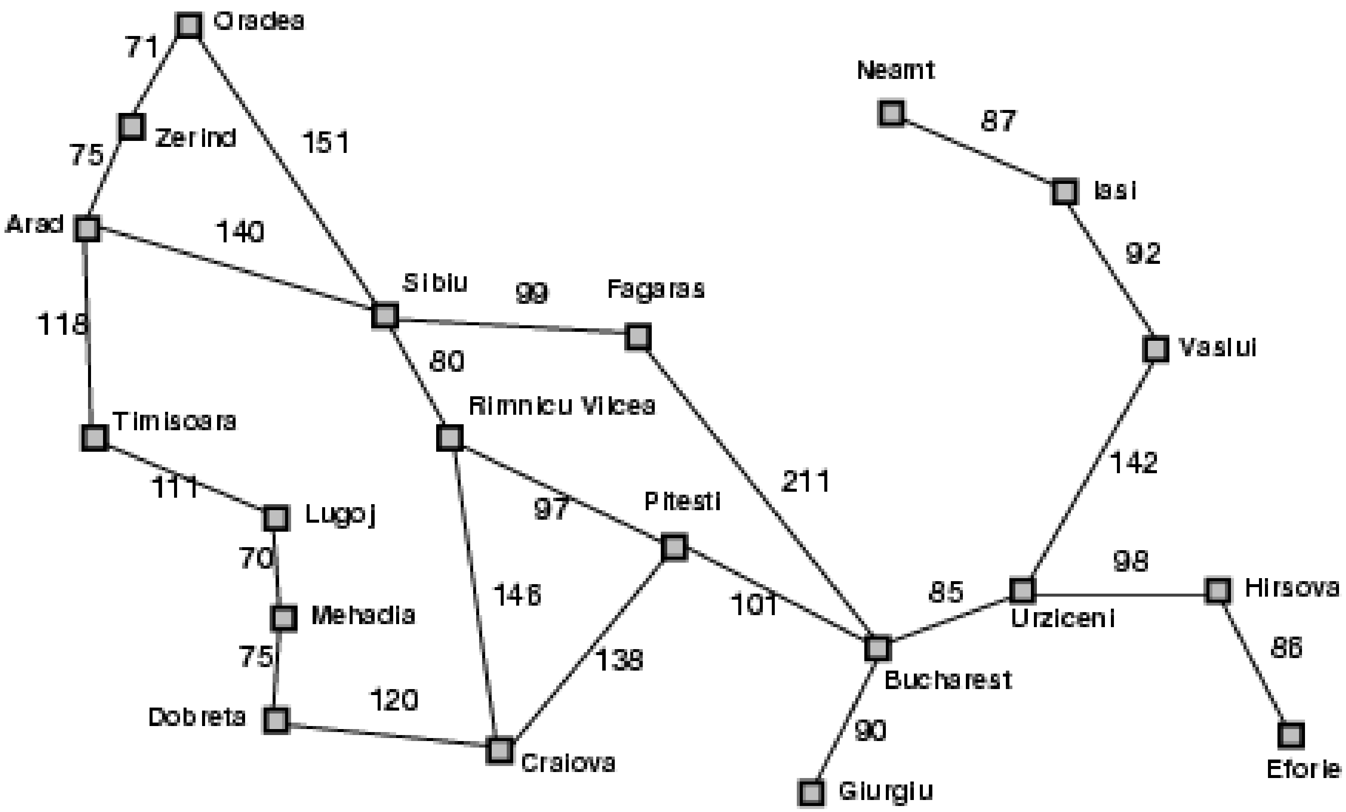
- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability“

  → Expand most desirable unexpanded node

- Implementation:
  Order the nodes in fringe in decreasing order of desirability

- Special cases:
  - greedy best-first search
  - A^* search

Romania with step costs in km



Old (Uninformed) Friends

- Breadth First =
  - Best First
  - with f(n) = depth(n)

- Uniform cost search =
  - Best First
  - with f(n) = the sum of edge costs from start to n

g(n)

Greedy best-first search

- Evaluation function $f(n) = h(n)$ (heuristic function) = estimate of cost from n to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

Properties of greedy best-first search

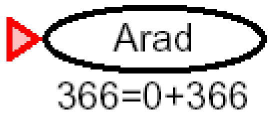
- Complete?
  - No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →
- Time?
  - $O(b^m)$, but a good heuristic can give dramatic improvement
- Space?
  - $O(b^m)$ -- keeps all nodes in memory
- Optimal?
  - No

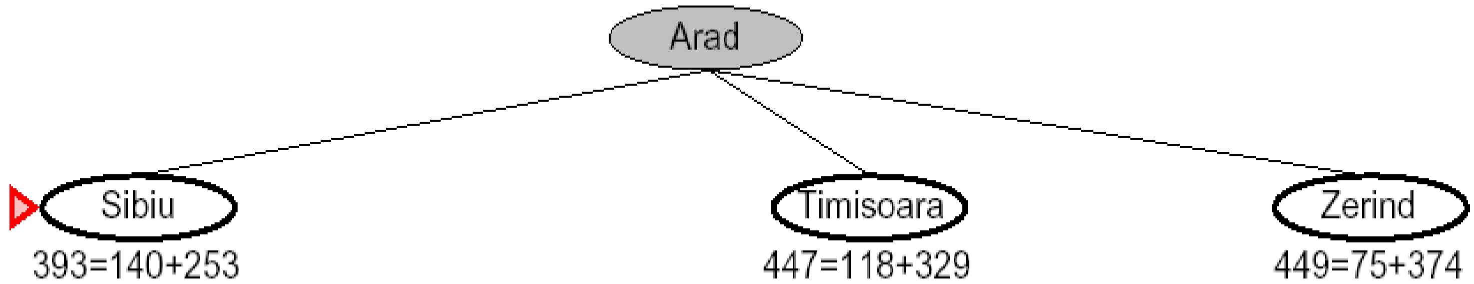
A^* search

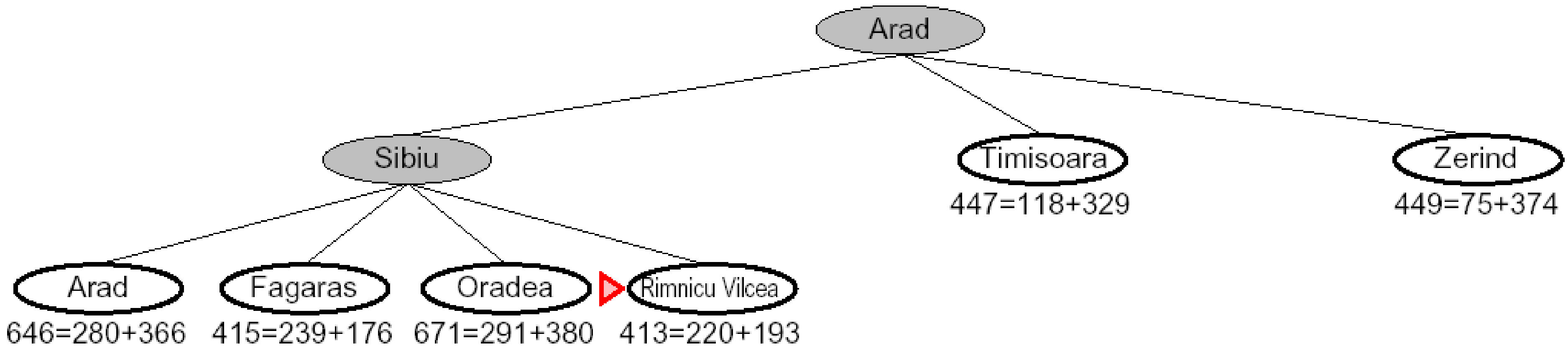
- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$

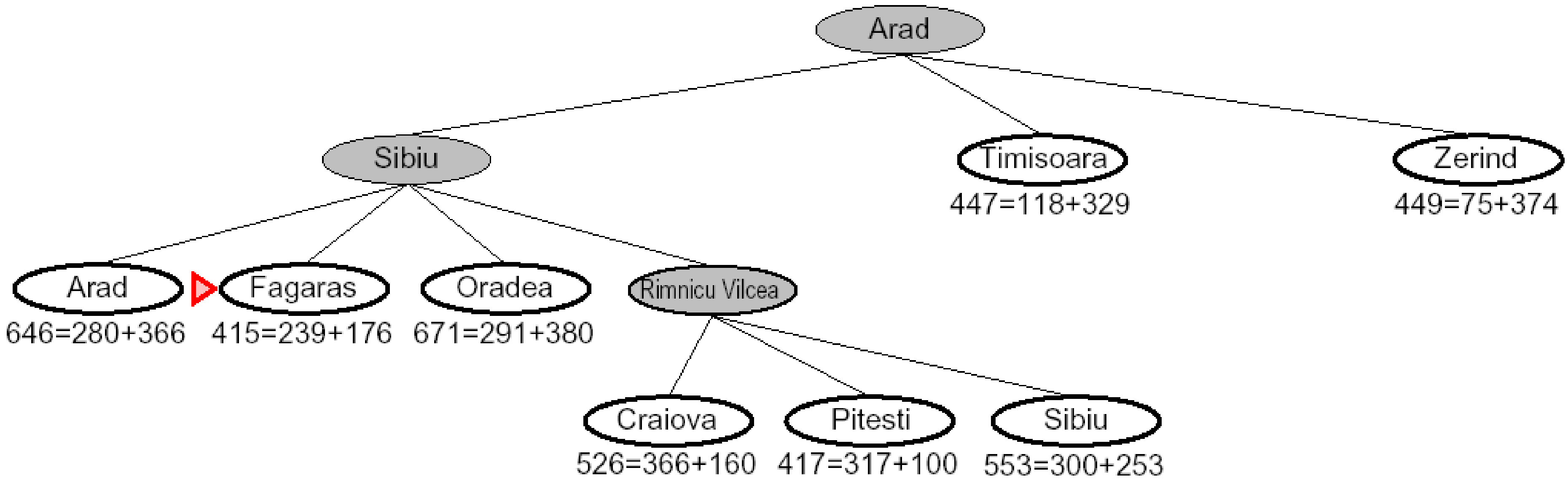
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

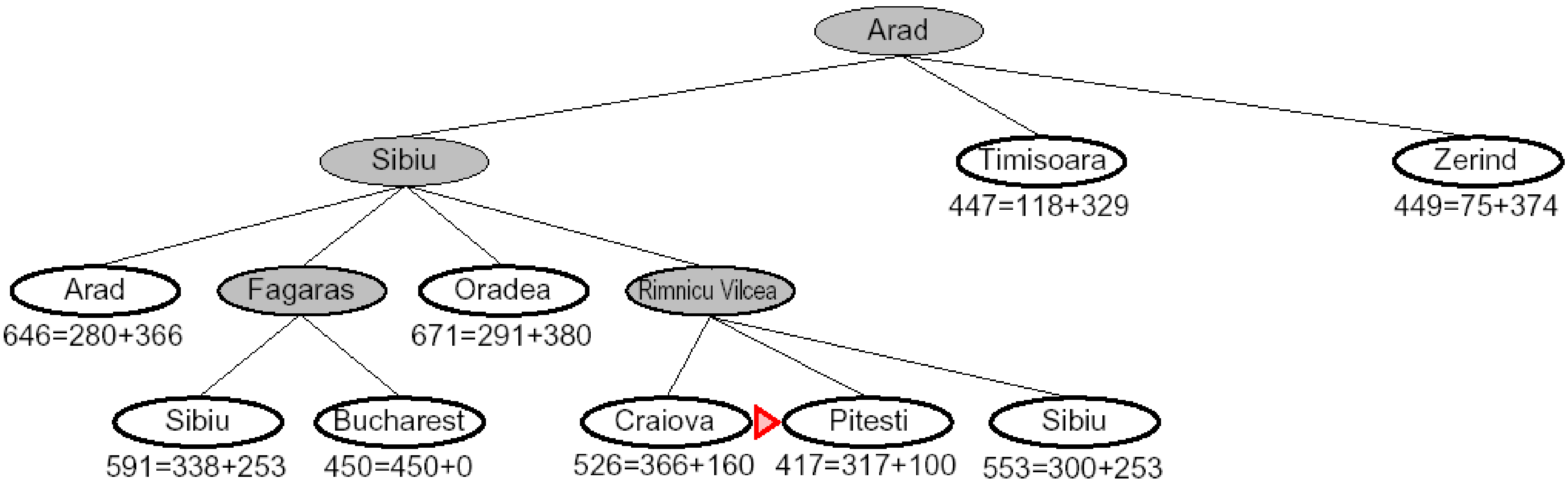
A* for Romanian Shortest Path



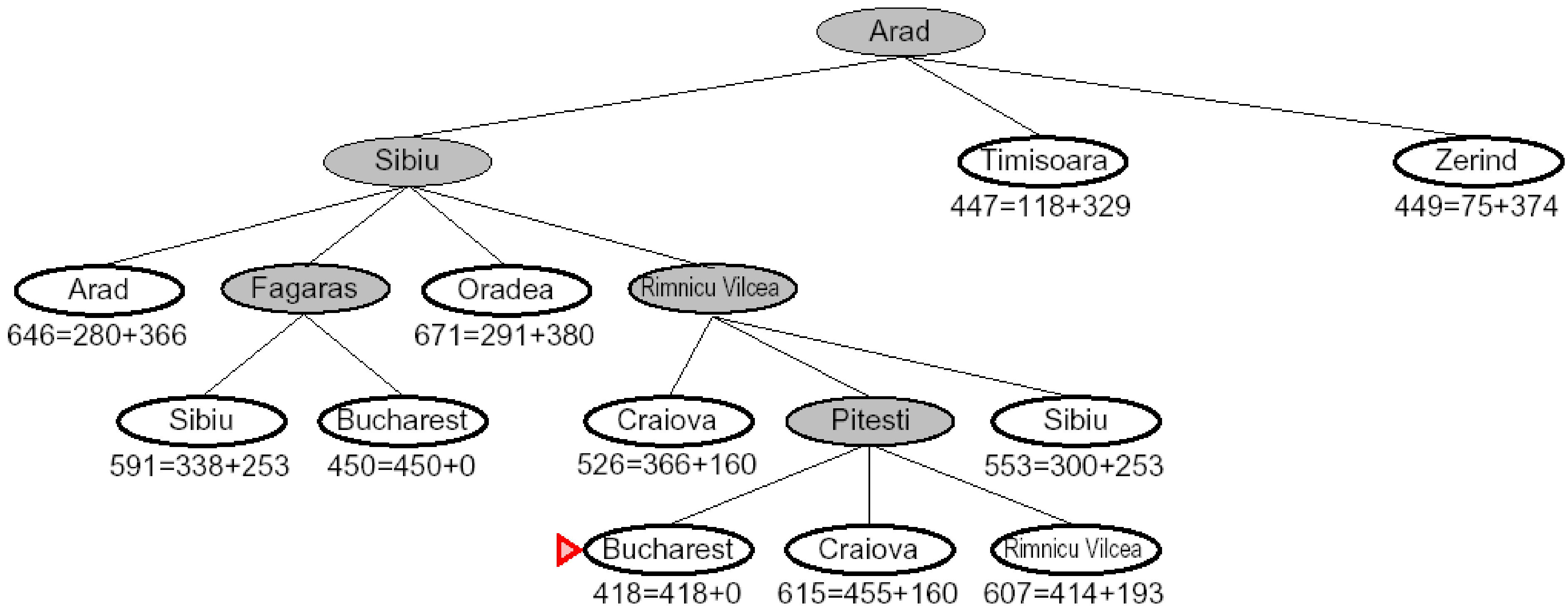

Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253



Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

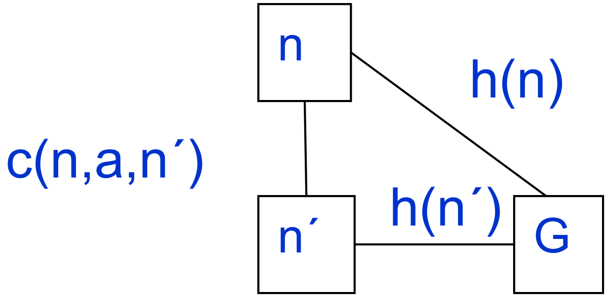

Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

Admissible heuristics

- A heuristic function $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If $h(n)$ is admissible, A* using `TREE-SEARCH` is optimal

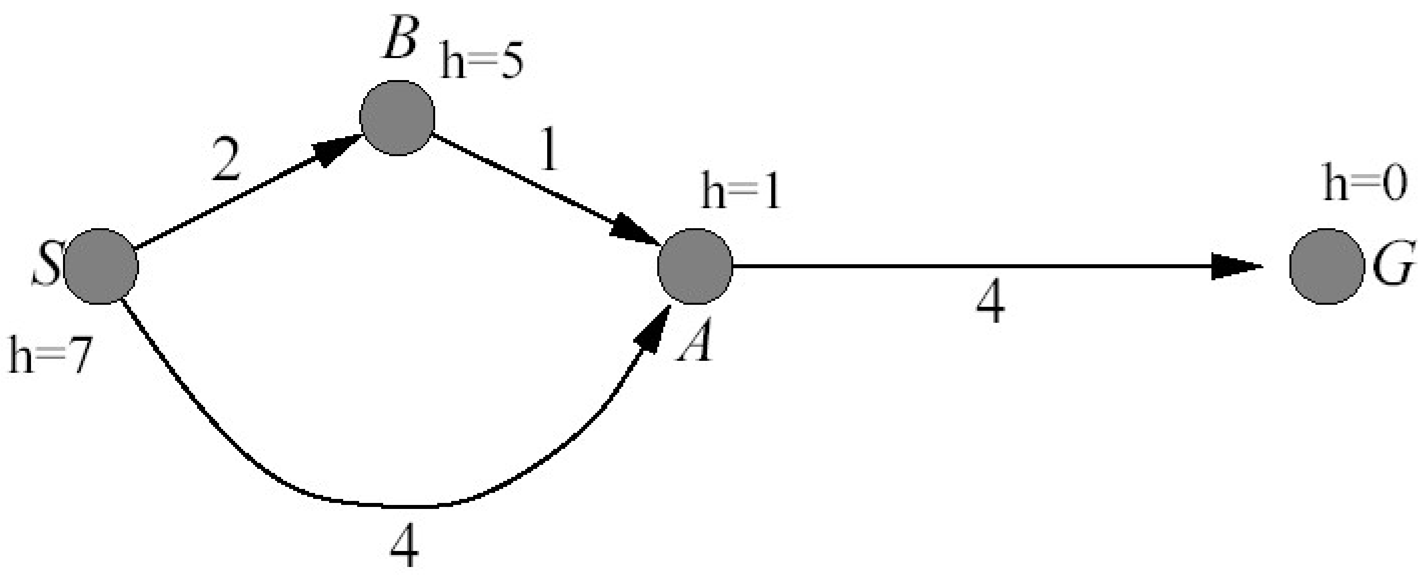
Consistent Heuristics

- h(n) is consistent if
  - for every node n
  - for every successor n´ due to legal action a
  - h(n) <= c(n,a,n´) + h(n´)



- Every consistent heuristic is also admissible.
- Theorem: If *h(n)* is consistent, A* using GRAPH–SEARCH is optimal

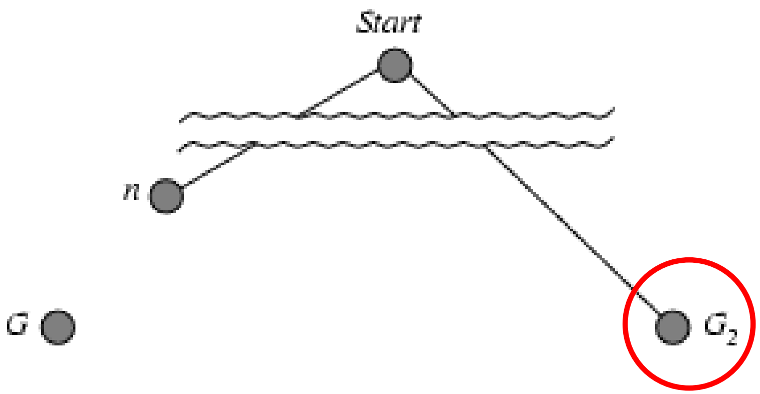
Example



Source: http://stackoverflow.com/questions/25823391/suboptimal-solution-given-by-a-search

Proof of Optimality of (Tree) A*

- Assume *h*() is admissible.
  Say some sub-optimal goal state G_2 has been generated and is on the frontier.
  Let *n* be an unexpanded state such that *n* is on an optimal path to the optimal goal *G*.



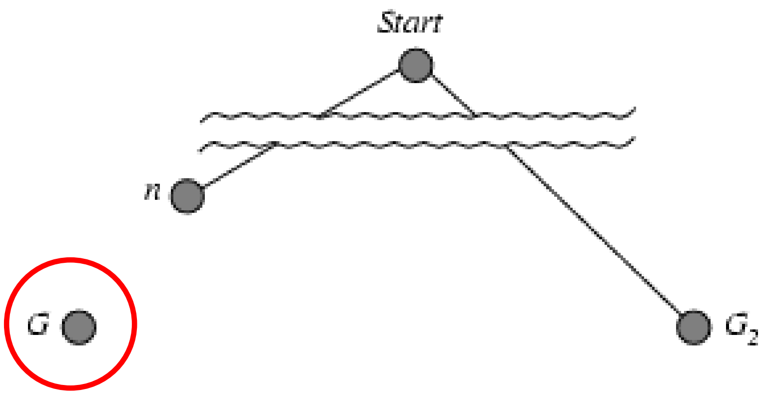
Focus on G_2:

$f(G_2) = g(G_2)$ since $h(G_2) = 0$

$g(G_2) > g(G)$ since G_2 is suboptimal

Proof of Optimality of (Tree) A*

- Assume *h*() is admissible.
  Say some sub-optimal goal state G_2 has been generated and is on the frontier. Let *n* be an unexpanded state such that *n* is on an optimal path to the optimal goal *G*.



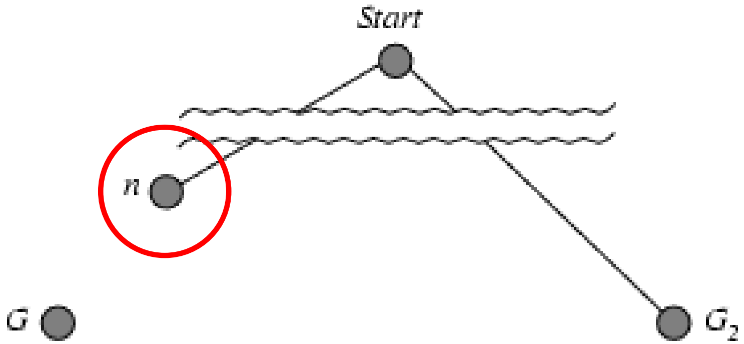
$f(G_2) = g(G_2)$	since $h(G_2) = 0$
$g(G_2) > g(G)$	since G_2 is suboptimal

Focus on G:

$f(G) = g(G)$	since $h(G) = 0$
$f(G_2) > f(G)$	substitution

Proof of Optimality of (Tree)A^*

- Assume *h*() is admissible.
  Say some sub-optimal goal state G_2 has been generated and is on the frontier. Let *n* be an unexpanded state such that *n* is on an optimal path to the optimal goal *G*.



$f(G_2) = g(G_2)$ since $h(G_2) = 0$
$g(G_2) > g(G)$ since G_2 is suboptimal

$f(G) = g(G)$ since $h(G) = 0$
$f(G_2) > f(G)$ substitution

Now focus on n:

$h(n) \le h^*(n)$ since h is admissible
$g(n) + h(n) \le g(n) + h^*(n)$ algebra
$f(n) = g(n) + h(n)$ definition
$f(G) = g(n) + h^*(n)$ by assumption
$f(n) \le f(G)$ substitution

Hence $f(G_2) > f(n)$, and A^* will never select G_2 for expansion.

Properties of A*

- Complete?

  Yes (unless there are infinitely many nodes with f ≤ *f(G)*)

- Time? Exponential (worst case all nodes are added)

- Space? Keeps all nodes in memory

- Optimal?

  Yes (depending upon search algo and heuristic property)

A*

http://www.youtube.com/watch?v=huJEgJ82360

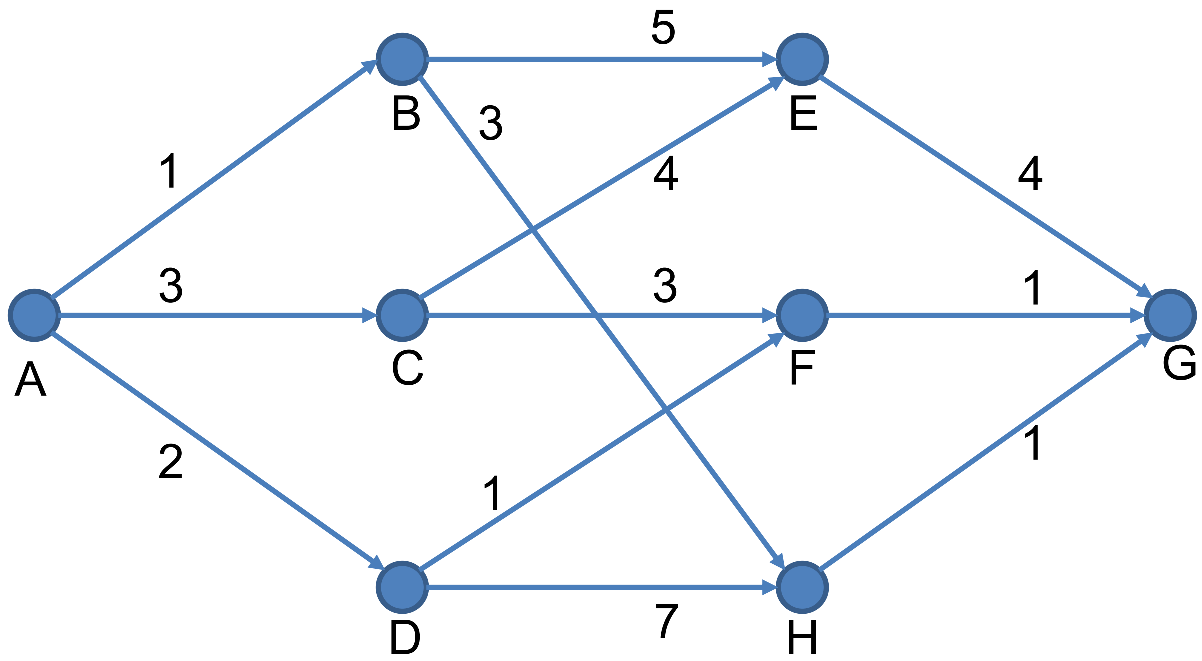
Memory Problem?

- Iterative deepening A*
  - Similar to ID search
  - While (solution not found)
    - Do DFS but prune when cost (f) > current bound
    - Increase bound

Depth First Branch and Bound

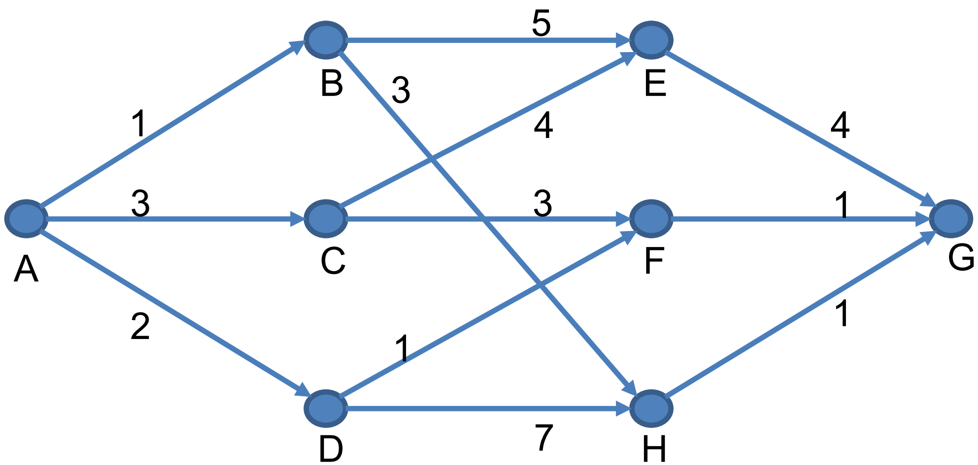
- 2 mechanisms:
  - BRANCH: A mechanism to generate branches when searching the solution space
    - Heuristic strategy for picking which one to try first.
  - BOUND: A mechanism to generate a bound so that many branches can be terminated

Example

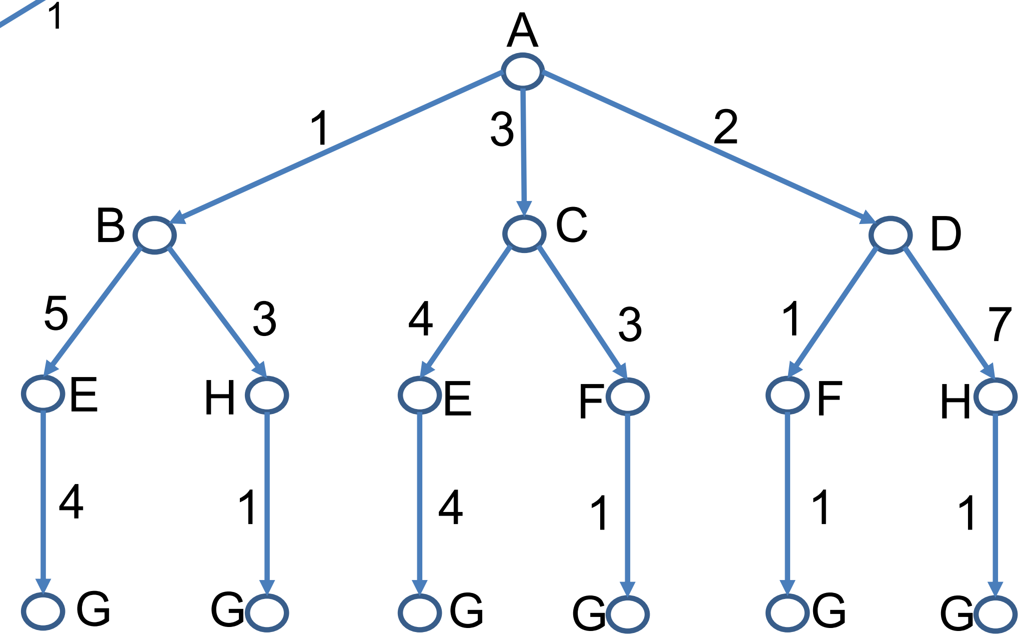


Find optimal path from A to G

Search Tree



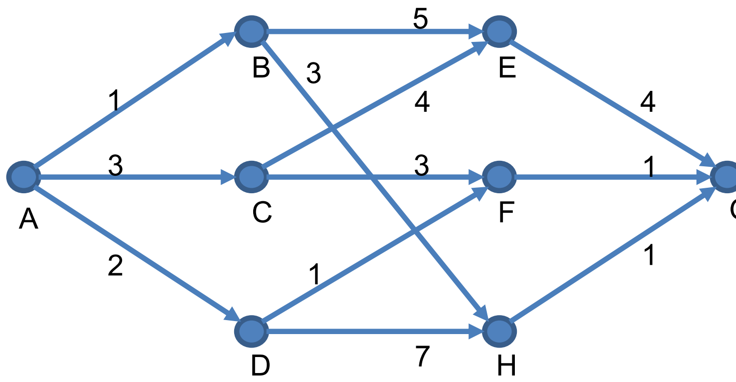
A
B
C
D
E
F
H
G
1
3
2
5
3
4
3
1
7
4
1
1

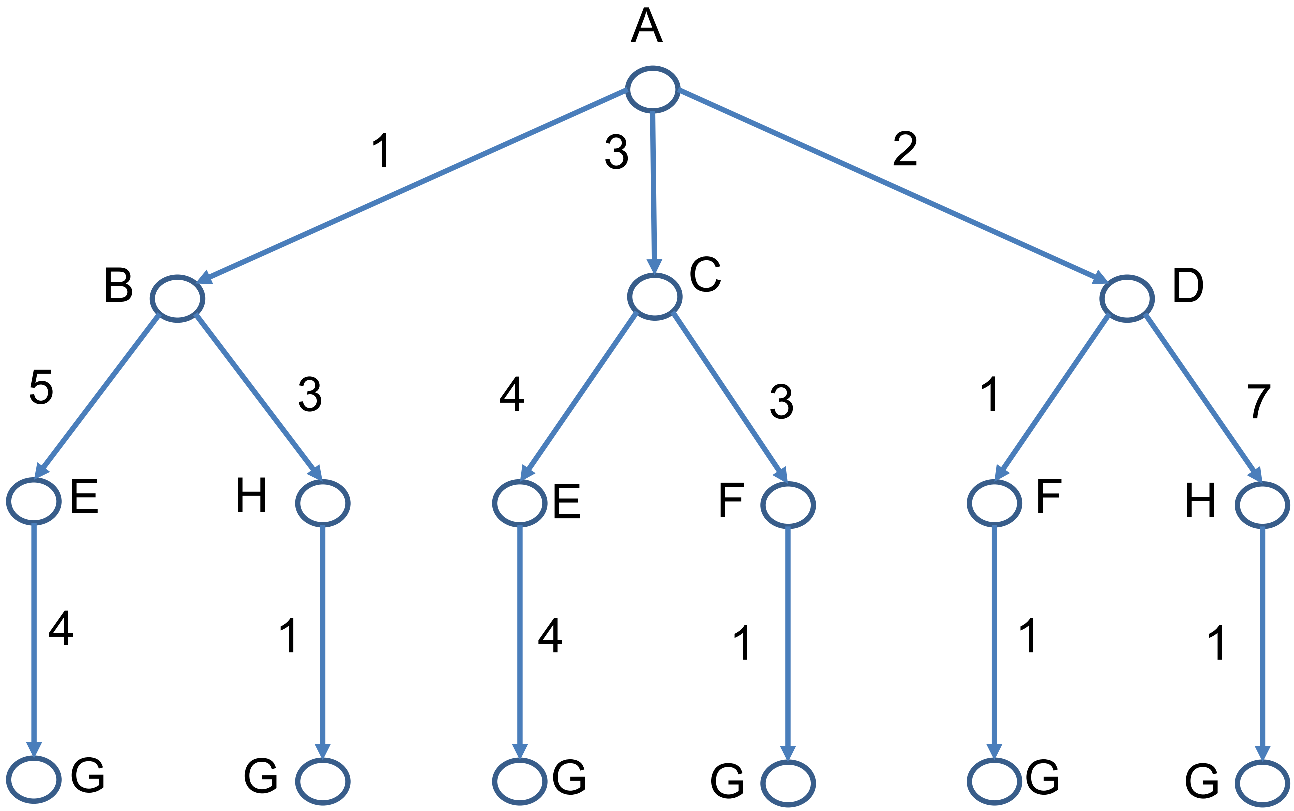


A
B
C
D
1
3
2
E
H
E
F
F
H
5
3
4
3
1
7
4
1
4
1
1
1
G
G
G
G
G
G

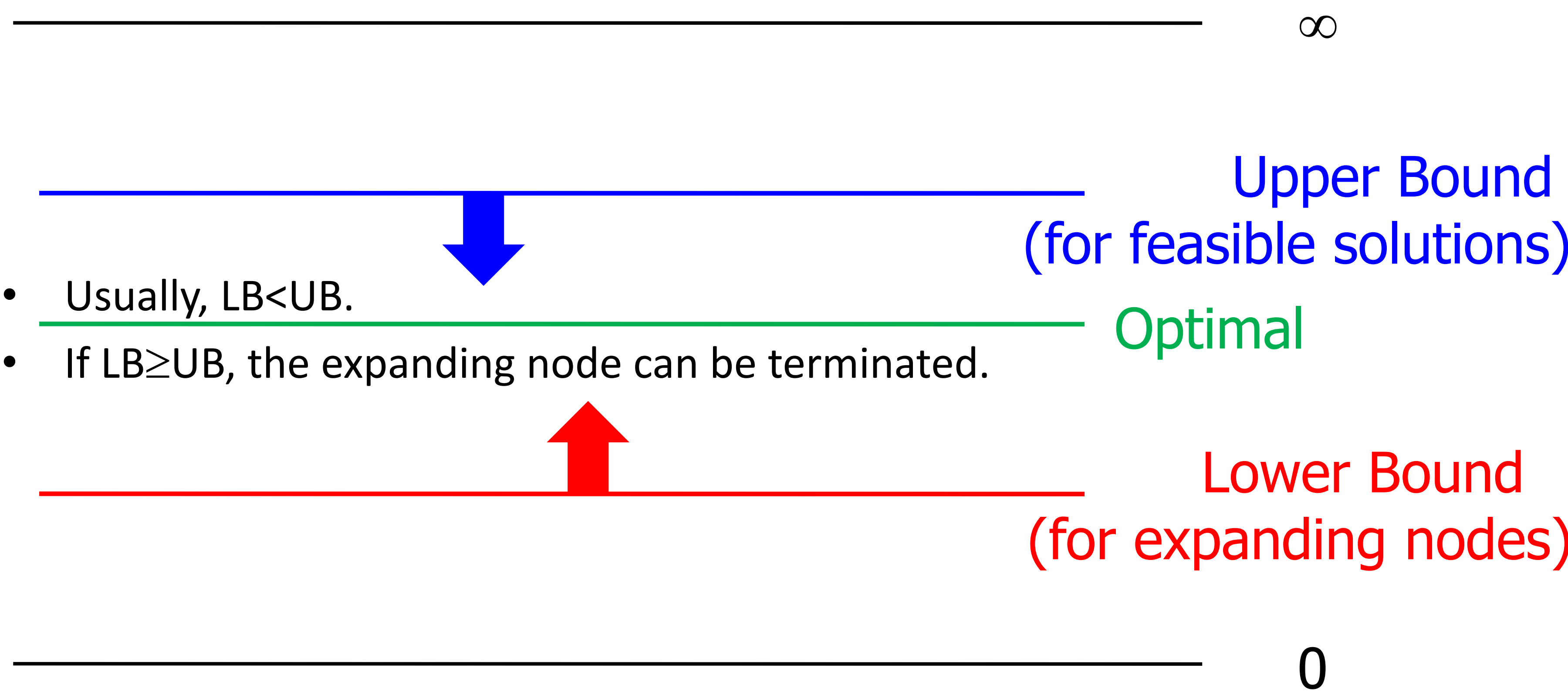
DFS B&B

E.g., Branch policy: take lowest cost edge first





For Minimization Problems



- Usually, LB<UB.
- If LB≥UB, the expanding node can be terminated.

DFS B&B vs. IDA*

- Both optimal
- IDA* never expands a node with f > optimal cost
  - But not systematic
- DFb&b systematic never expands a node twice
  - But expands suboptimal nodes also
- Search tree of bounded depth?
- Easy to find suboptimal solution?
- Infinite search trees?
- Difficult to construct a single solution?

Non-optimal variations

- Use more informative, but inadmissible heuristics

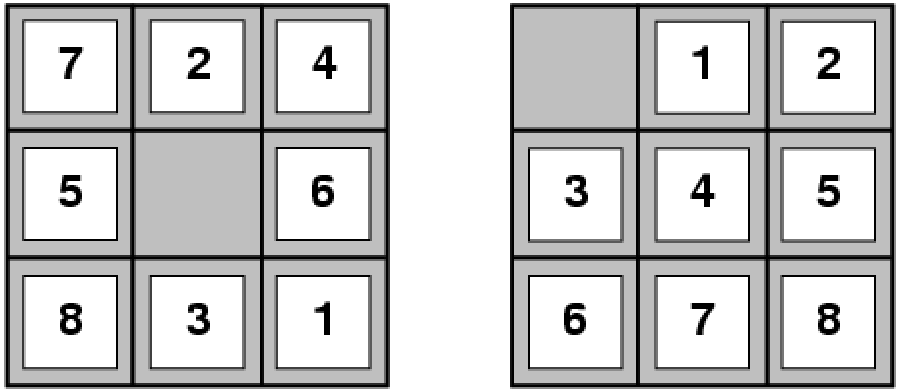
- Weighted A*
  - $f(n) = g(n) + w.h(n)$ where $w>1$
  - Typically $w=5$.
  - Solution quality bounded by w for admissible h

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State Goal State

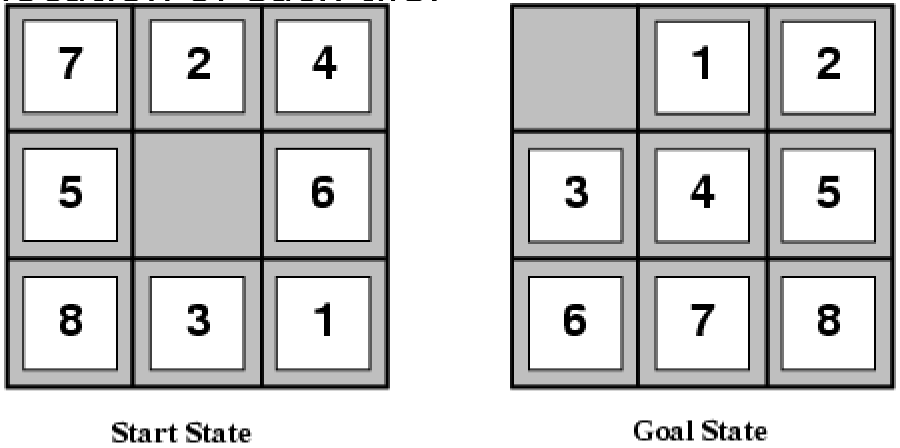
- $h_1(S)$ = ?
- $h_2(S)$ = ?

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



- $h_1(S)$ = ? 8
- $h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 dominates h_1
- h_2 is better for search

- Typical search costs (average number of node expanded):

- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- *d=24* IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

Hamiltonian Cycle Problem

What can be relaxed?

Solution =

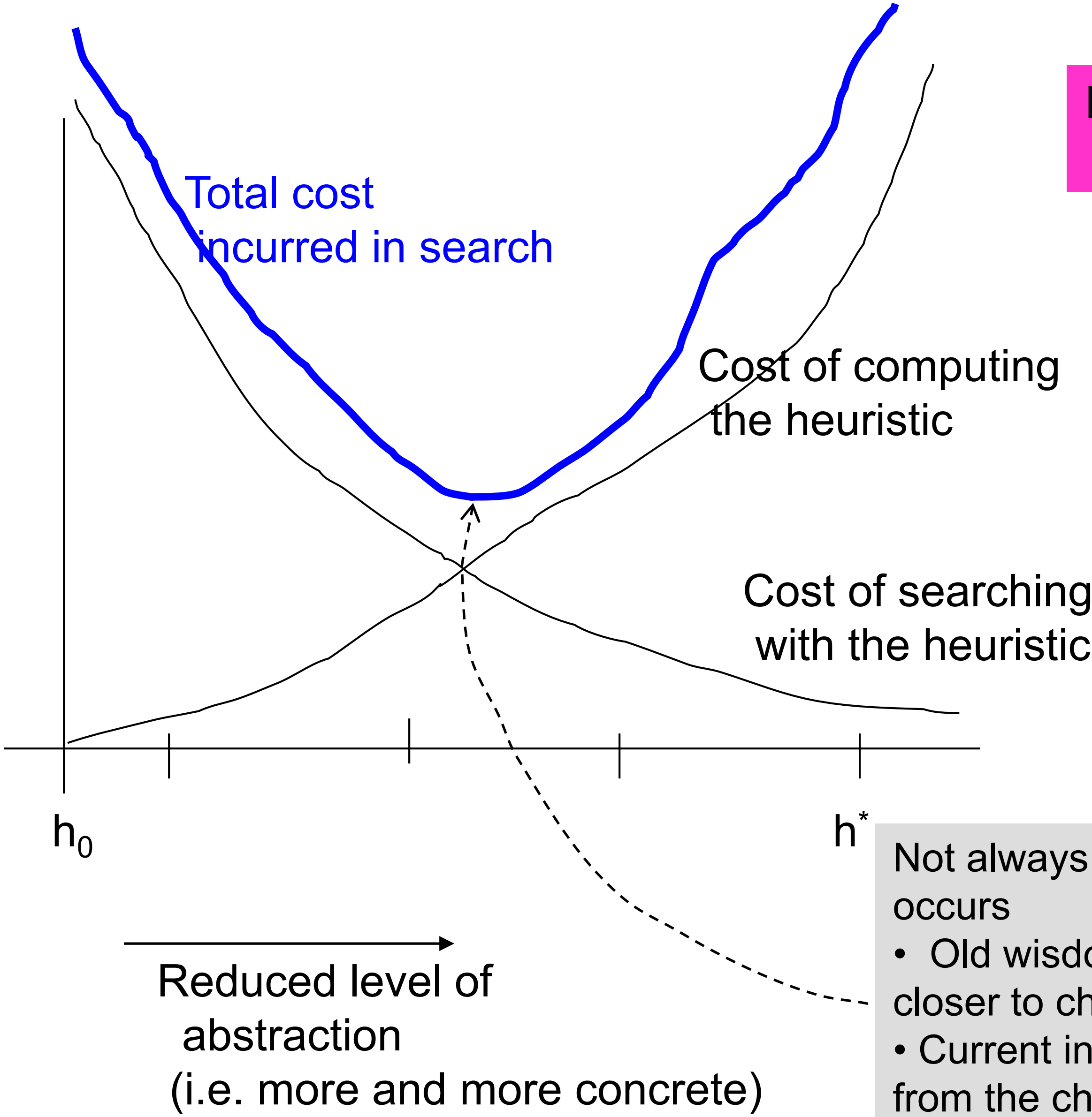
1) Each node degree 2
2) Visit all nodes
3) Visit nodes exactly once

What is a good admissible heuristic for (a1→a2→…→ak)

- length of the cheapest edge leaving ak +
  length of cheapest edge entering a1

- length of shortest path from ak to a1

- length of minimum spanning tree of rest of the nodes



How informed should the heuristic be?

Not always clear where the total minimum occurs

- Old wisdom was that the global min was closer to cheaper heuristics
- Current insights are that it may well be far from the cheaper heuristics for many problems
    - E.g. Pattern databases for 8-puzzle
    - Plan graph heuristics for planning

Sizes of Problem Spaces

Problem	Nodes	Brute-Force Search Time (10 million nodes/second)
• 8 Puzzle:	10^5	.01 seconds
• 2^3 Rubik's Cube:	10^6	.2 seconds
• 15 Puzzle:	10^{13}	6 days
• 3^3 Rubik's Cube:	10^{19}	68,000 years
• 24 Puzzle:	10^{25}	12 billion years

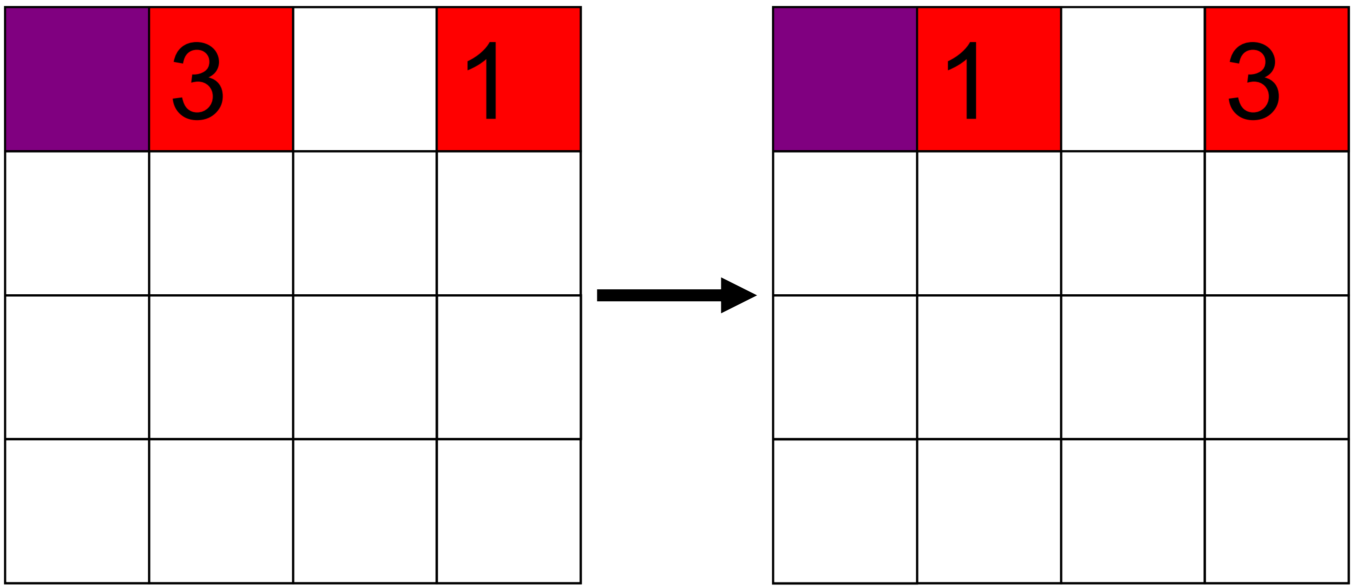
Performance of IDA* on 15 Puzzle

- Random 15 puzzle instances were first solved optimally using IDA* with Manhattan distance heuristic (Korf, 1985).
- Optimal solution lengths average 53 moves.
- 400 million nodes generated on average.
- Average solution time is about 50 seconds on current machines.

Limitation of Manhattan Distance

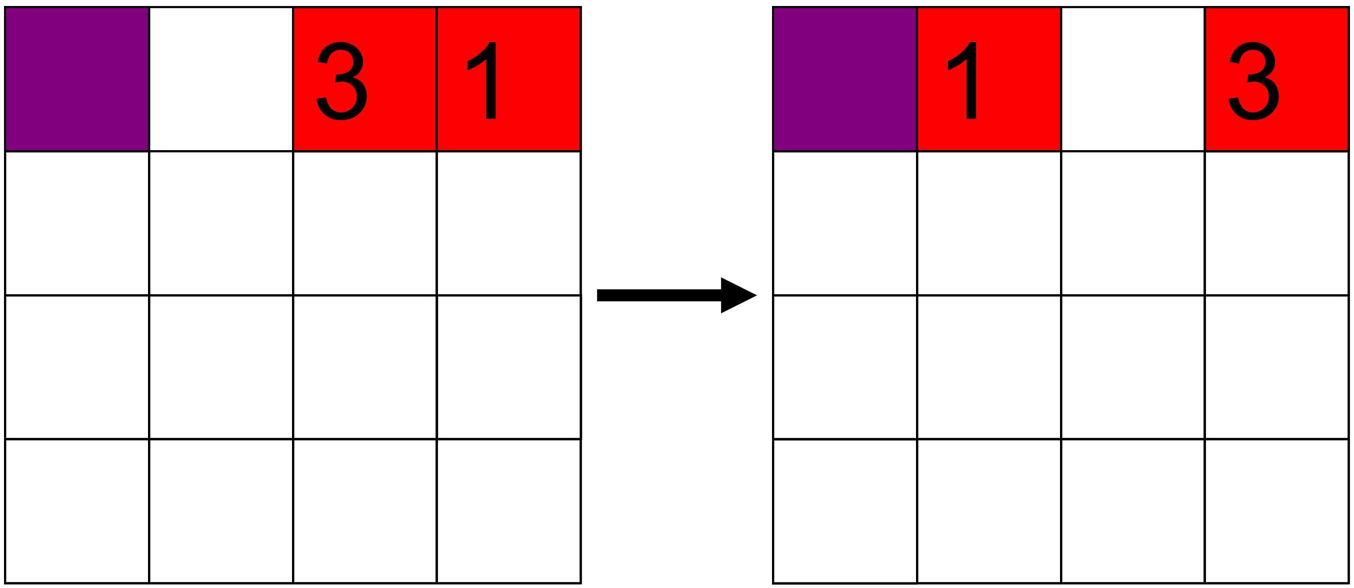
- To solve a 24-Puzzle instance, IDA* with Manhattan distance would take about 65,000 years on average.
- Assumes that each tile moves independently
- In fact, tiles interfere with each other.
- Accounting for these interactions is the key to more accurate heuristic functions.

Example: Linear Conflict



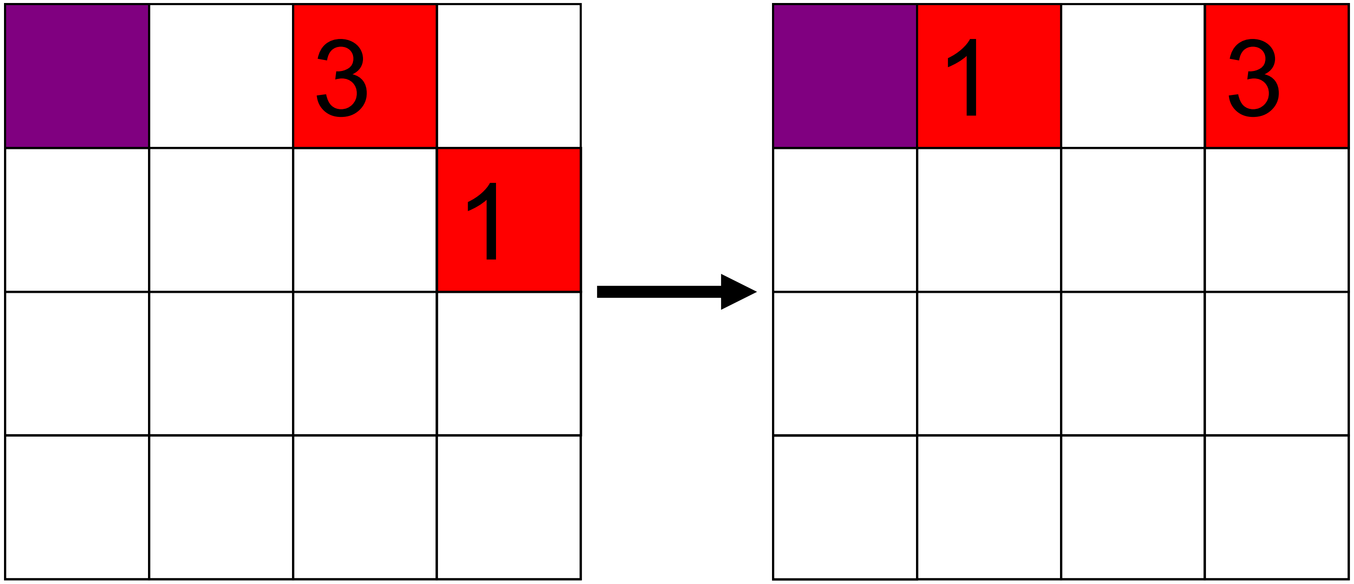
Manhattan distance is 2+2=4 moves

Example: Linear Conflict



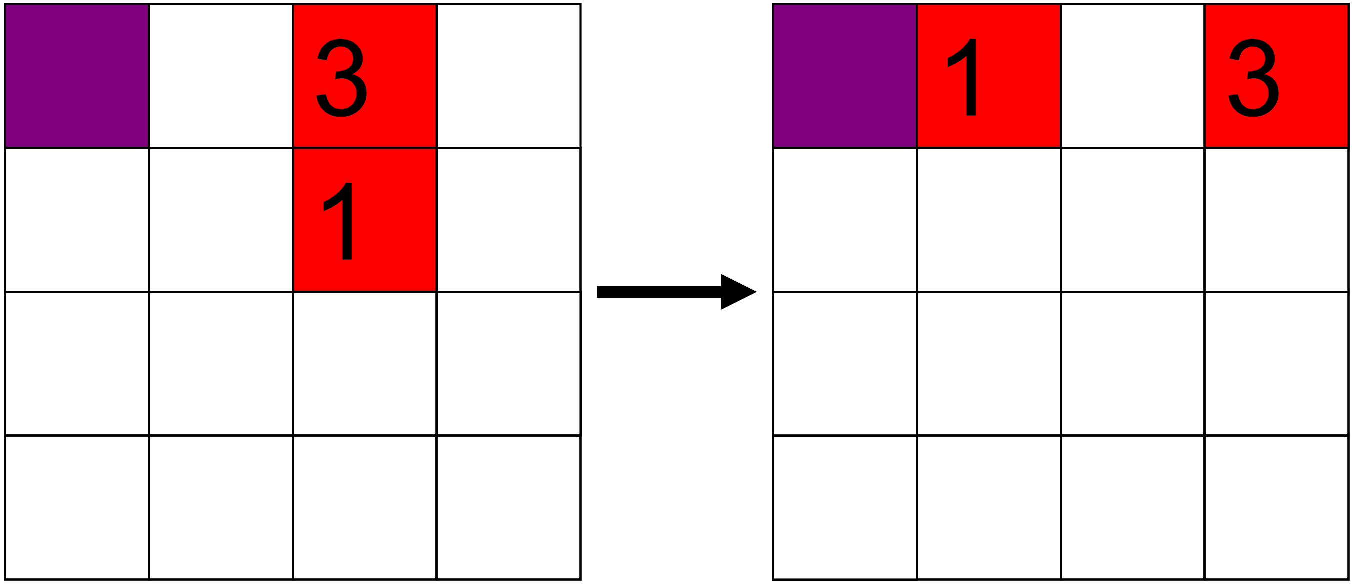
Manhattan distance is 2+2=4 moves

Example: Linear Conflict



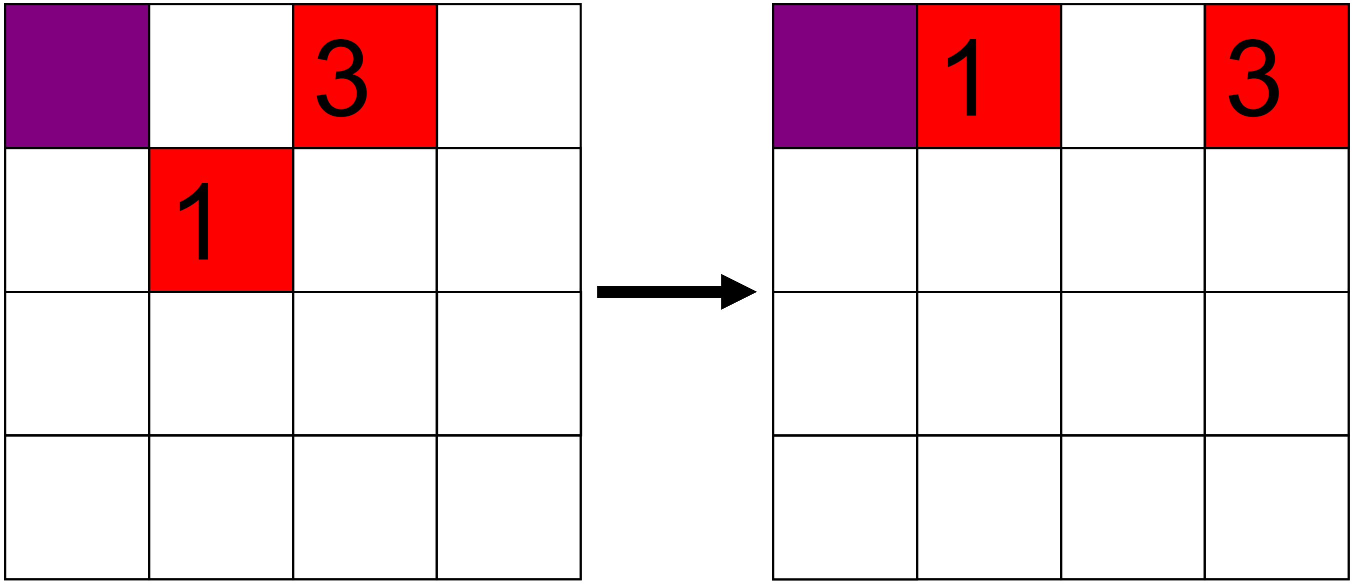
Manhattan distance is 2+2=4 moves

Example: Linear Conflict



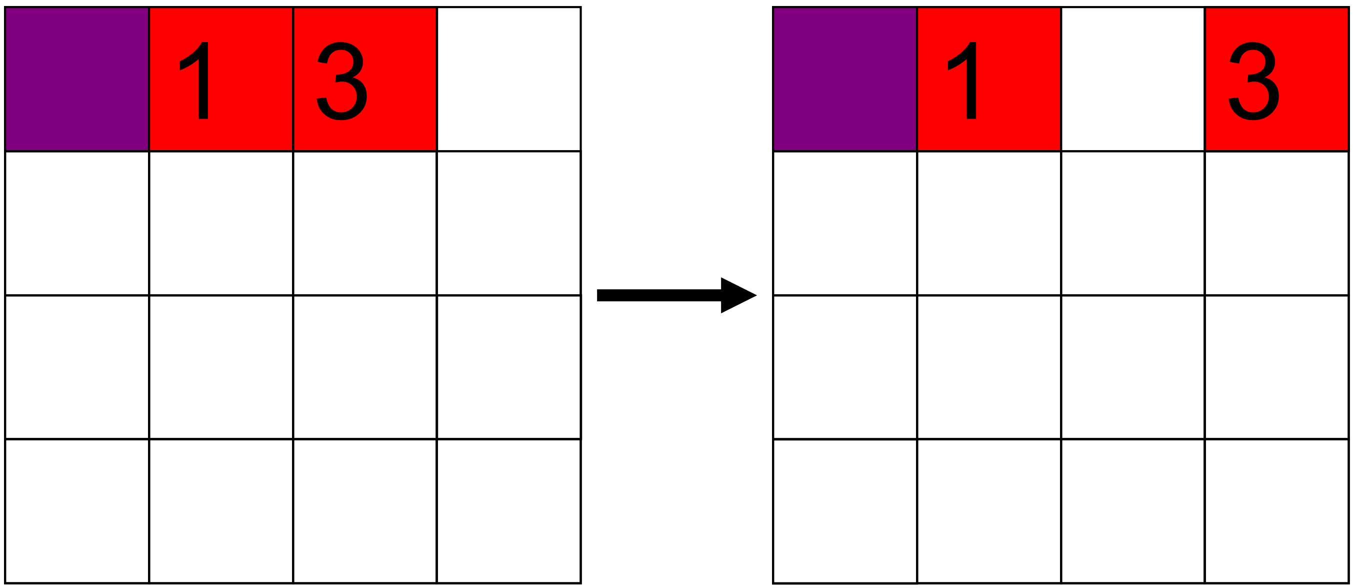
Manhattan distance is 2+2=4 moves

Example: Linear Conflict



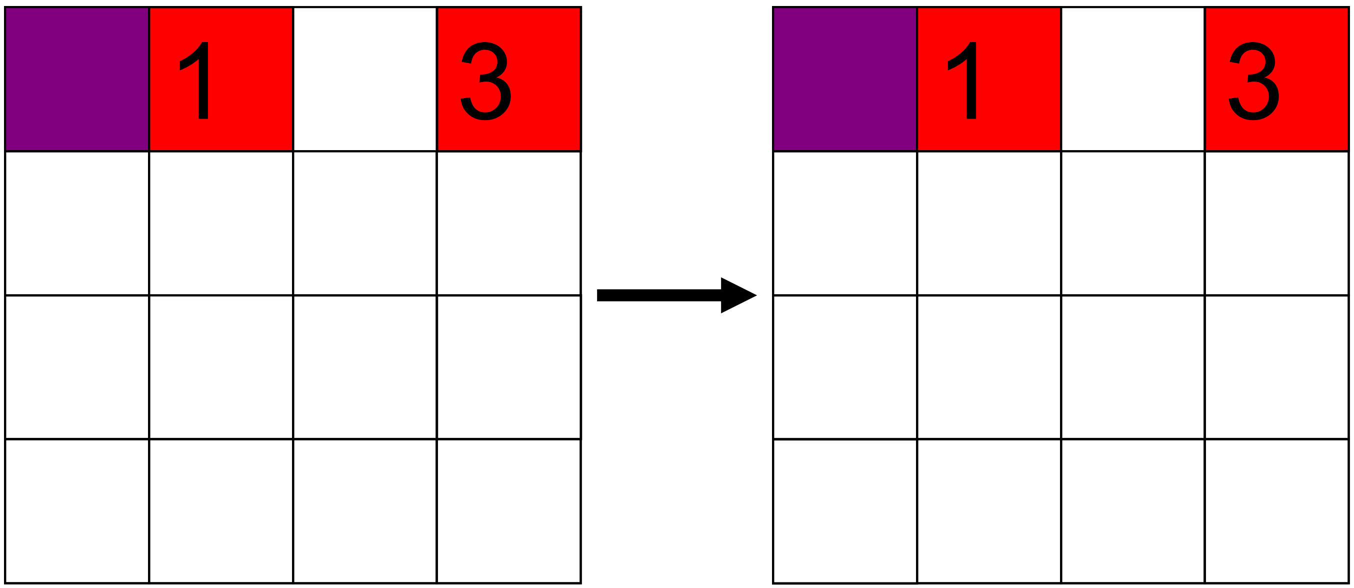
Manhattan distance is 2+2=4 moves

Example: Linear Conflict



Manhattan distance is 2+2=4 moves

Example: Linear Conflict

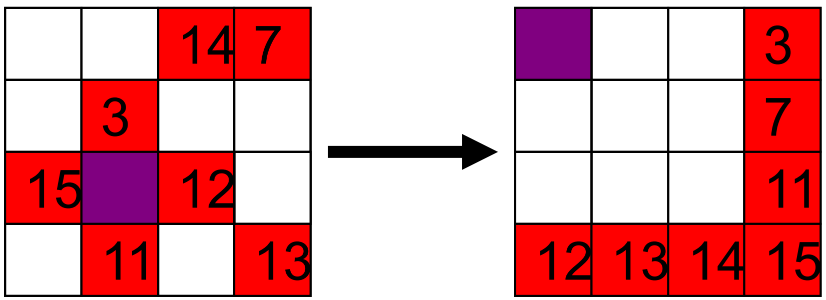


Manhattan distance is 2+2=4 moves, but linear conflict adds 2 additional moves.

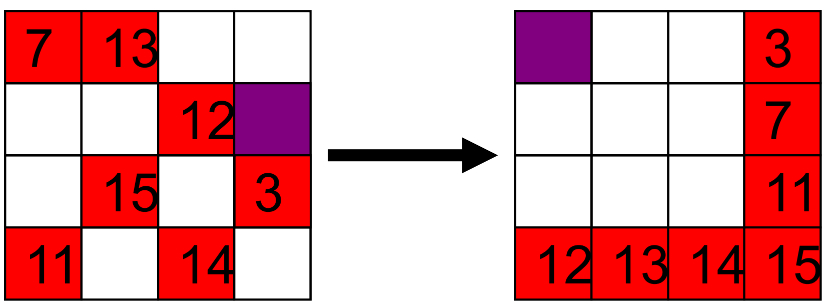
Linear Conflict Heuristic

- Hansson, Mayer, and Yung, 1991
- Given two tiles in their goal row, but reversed in position, additional vertical moves can be added to Manhattan distance.
- Still not accurate enough to solve 24-Puzzle
- We can generalize this idea further.

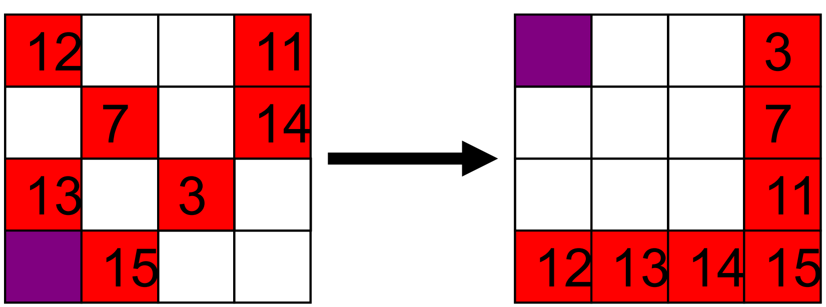
More Complex Tile Interactions



M.d. is 19 moves, but 31 moves are needed.



M.d. is 20 moves, but 28 moves are needed

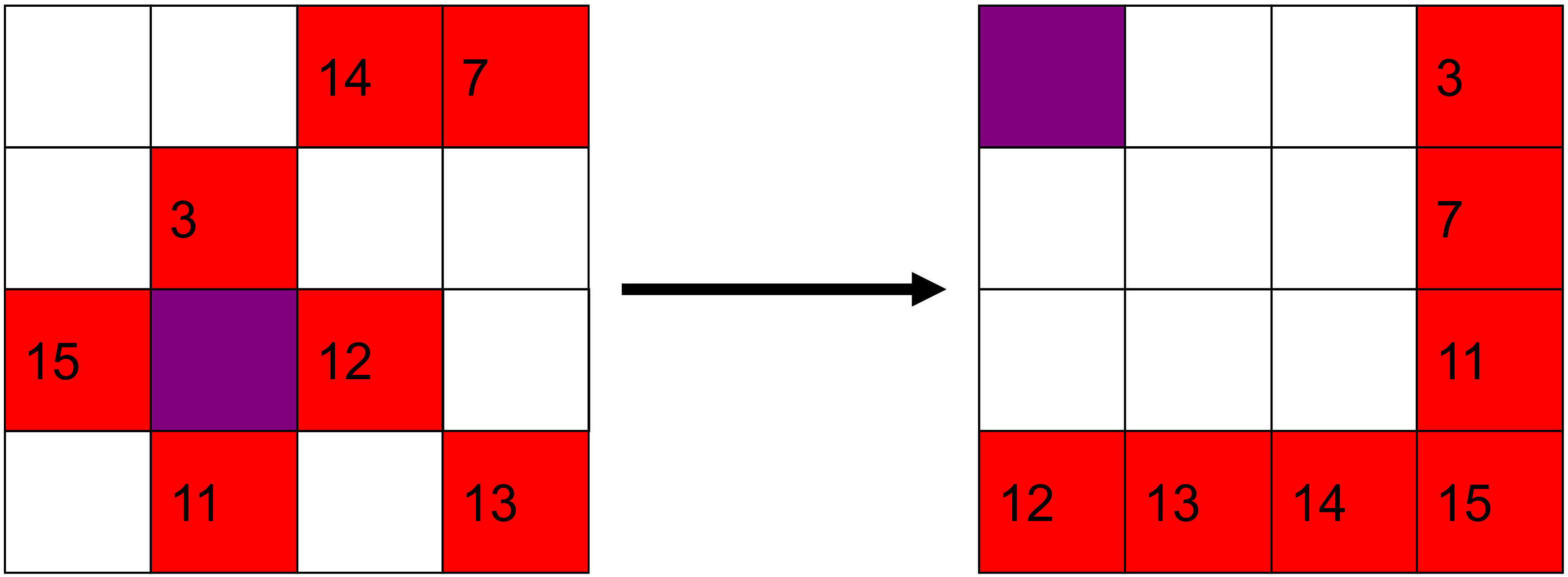


M.d. is 17 moves, but 27 moves are needed

Pattern Database Heuristics

- Culberson and Schaeffer, 1996
- A pattern database is a complete set of such positions, with associated number of moves.
- e.g. a 7-tile pattern database for the Fifteen Puzzle contains 519 million entries.

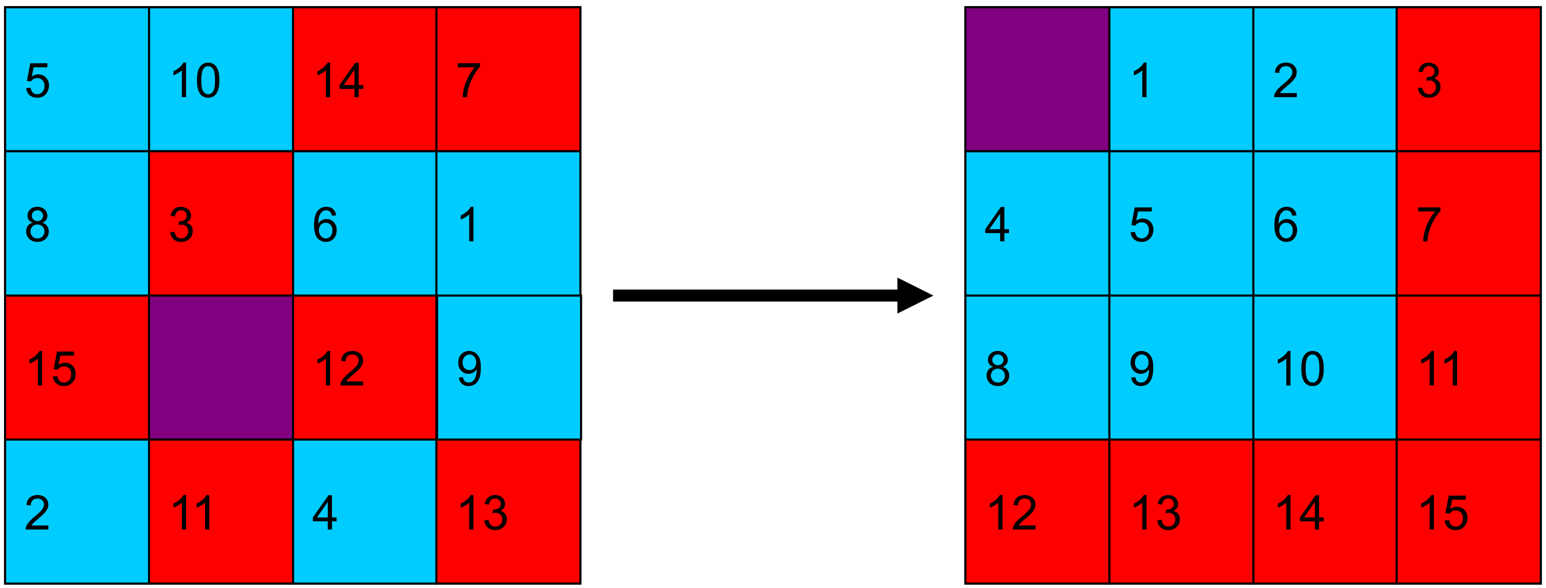
Example 8-tile pattern



Precomputing Pattern Databases

- Entire database is computed with one backward breadth-first search from goal.
- All non-pattern tiles are indistinguishable, but all tile moves are counted.
- The first time each state is encountered, the total number of moves made so far is stored.
- Once computed, the same table is used for all problems with the same goal state.

Combining Multiple Databases



31 moves needed to solve red tiles

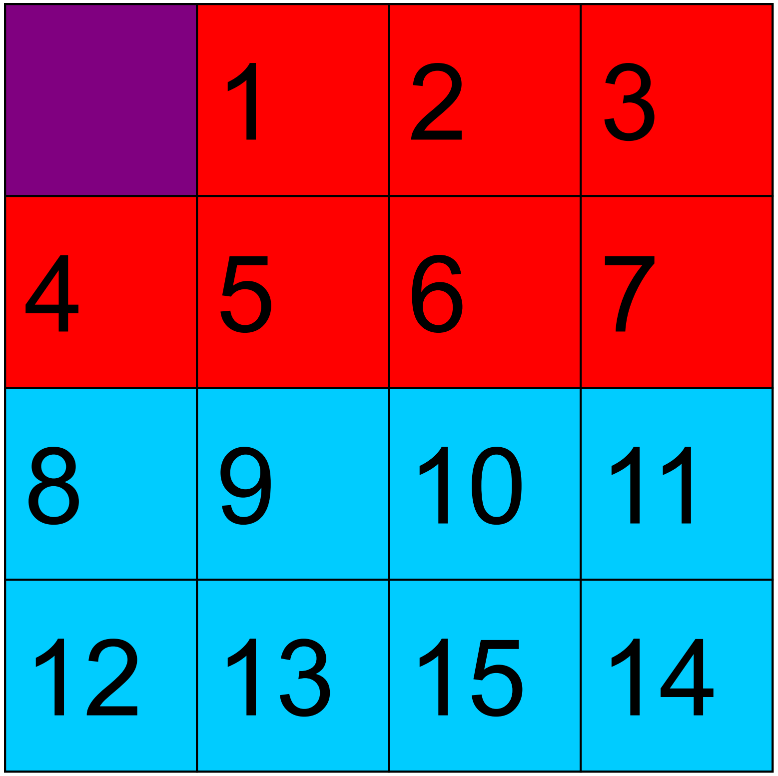
22 moves need to solve blue tiles

Overall heuristic is maximum of 31 moves

Additive Pattern Databases

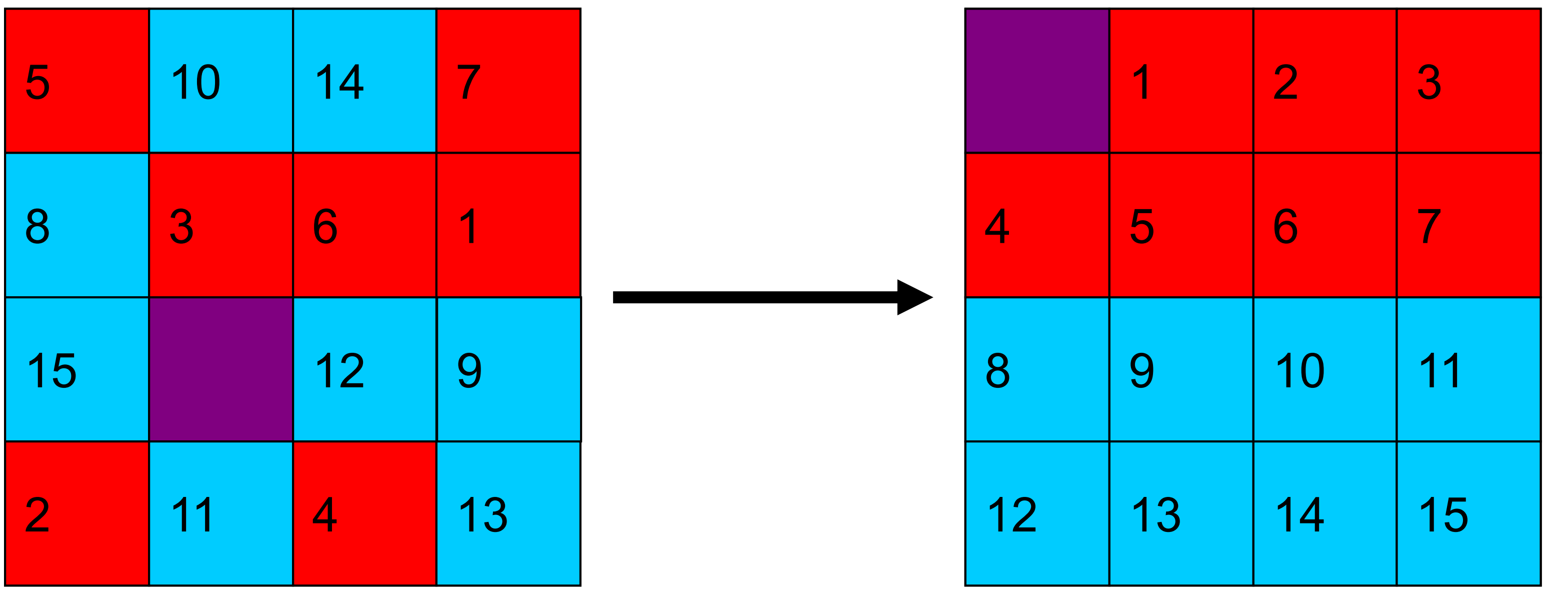
- Culberson and Schaeffer counted all moves needed to correctly position the pattern tiles.
- In contrast, we count only moves of the pattern tiles, ignoring non-pattern moves.
- If no tile belongs to more than one pattern, then we can add their heuristic values.
- Manhattan distance is a special case of this, where each pattern contains a single tile.

Example Additive Databases



The 7-tile database contains 58 million entries. The 8-tile database contains 519 million entries.

Computing the Heuristic



20 moves needed to solve red tiles

25 moves needed to solve blue tiles

Overall heuristic is sum, or 20+25=45 moves

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