

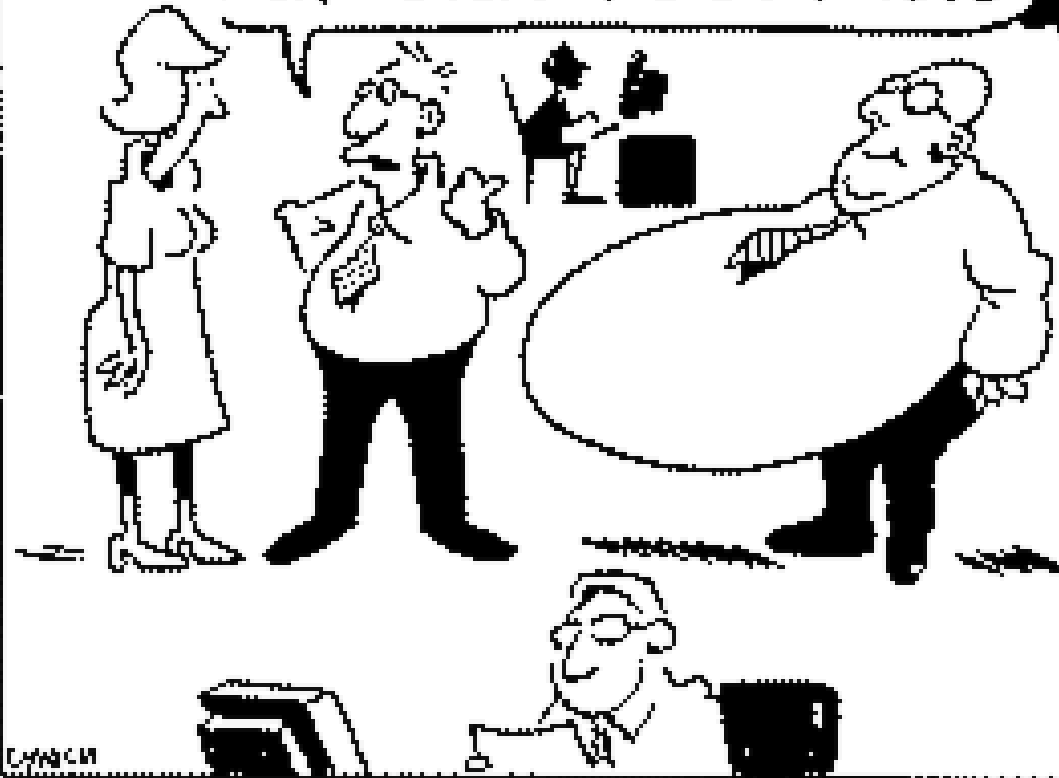
# Informed search algorithms

## Chapter 3

(Based on Slides by Stuart Russell,  
Dan Klein, Richard Korf, Carl Kingsford, Subbarao  
Kambhampati, Eric Ringger, and UW-AI faculty)

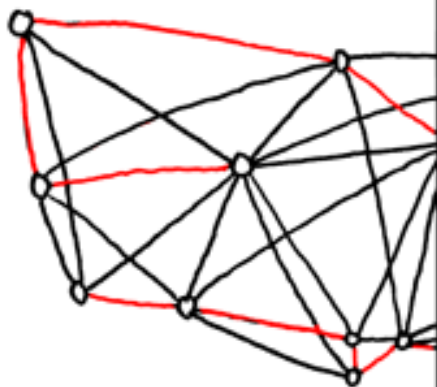
*“Intuition, like the rays of the sun, acts only in an inflexibly straight line; it can guess right only on condition of never diverting its gaze; the freaks of chance disturb it.”*

AND WARREN HERE  
IS IN CHARGE OF  
OUR GUT FEELINGS



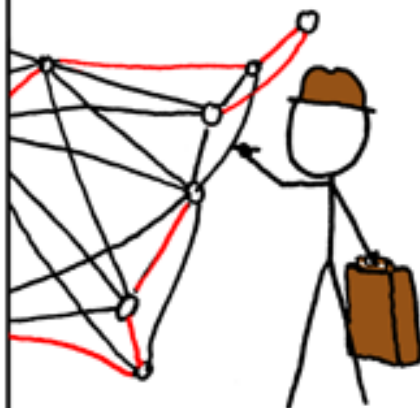
BRUTE-FORCE  
SOLUTION:

$$O(n!)$$



DYNAMIC  
PROGRAMMING  
ALGORITHMS:

$$O(n^2 2^n)$$



SELLING ON EBAY:  
 $O(1)$

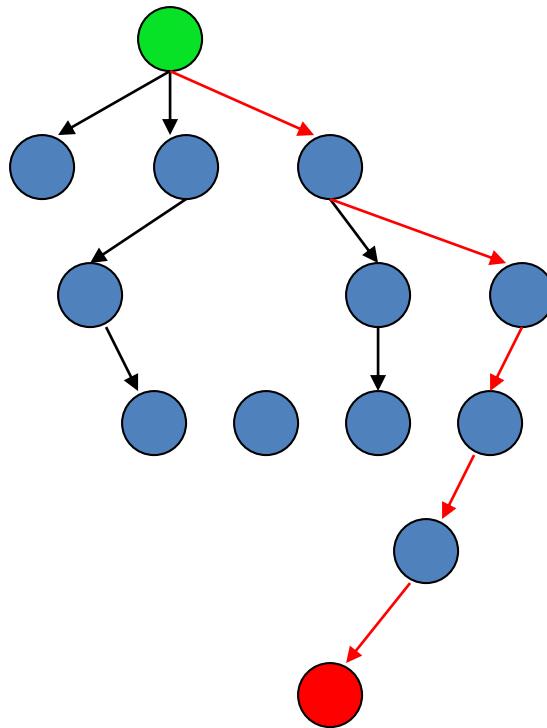
STILL WORKING  
ON YOUR ROUTE?

SHUT THE  
HELL UP.



# 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
```

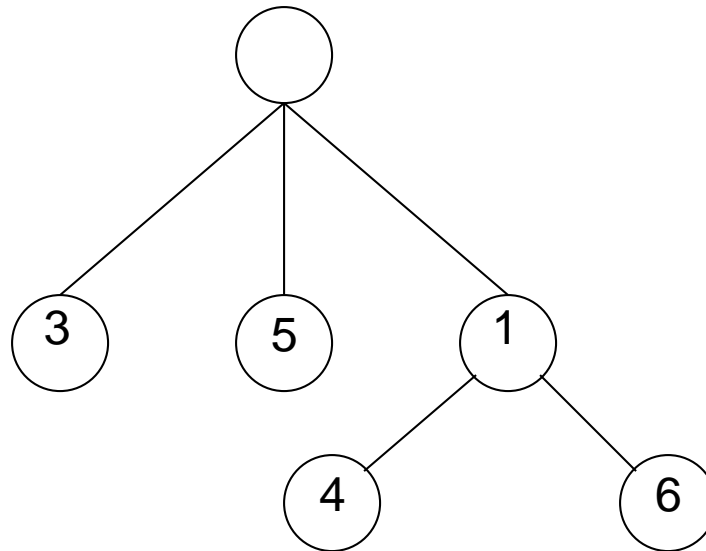
# General Graph Search Paradigm

```
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

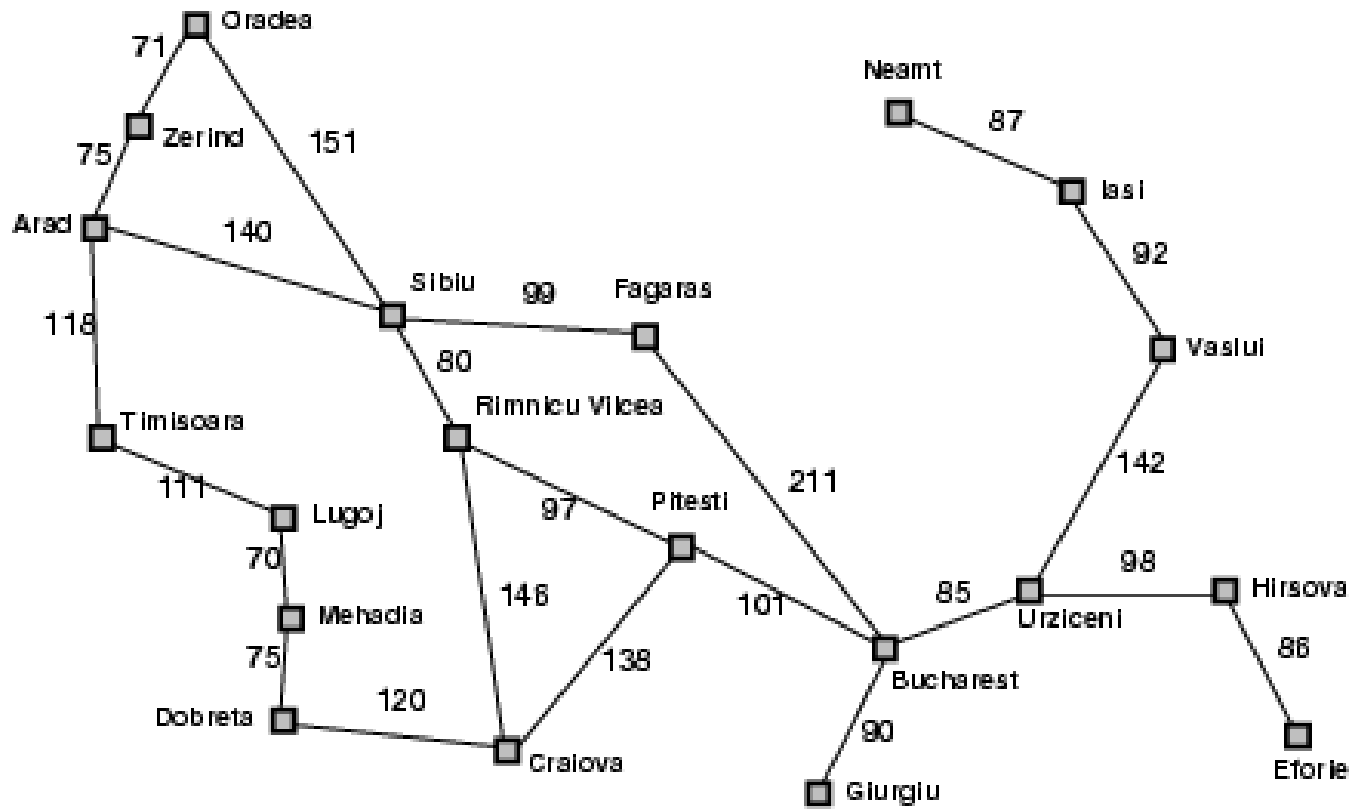
- Use an **evaluation function  $f(n)$**  for node  $n$ .
- Always choose the node from fringe that has the **lowest**  $f$  value.



# 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



Straight-line distance  
to Bucharest

<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	176
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	10
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

# Old Friends

- Breadth First =
  - Best First
  - with  $f(n) = \text{depth}(n)$
- Uniform cost search =
  - Best First
  - with  $f(n) = \text{the sum of edge costs from start to } n$

# Greedy best-first search

- Evaluation function  $f(n) = h(n)$  (**h**euristic)  
= 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

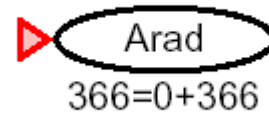
# Properties of greedy best-first search

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

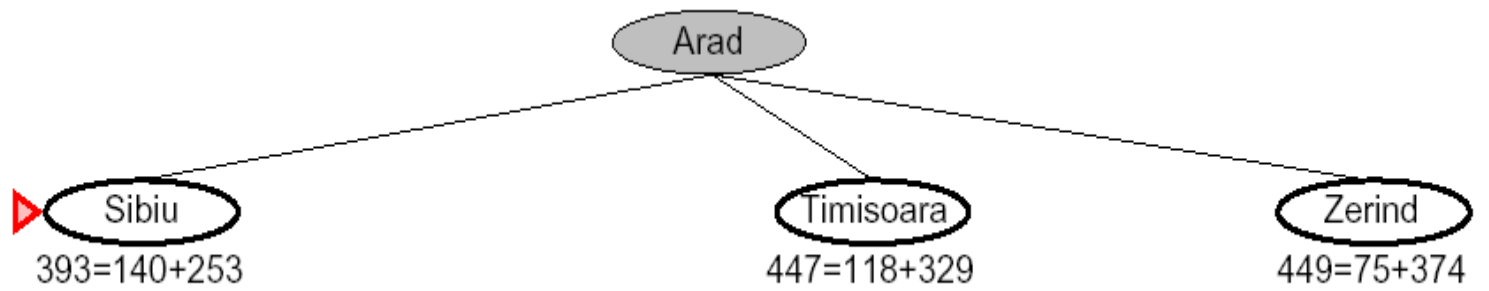
# A\* search

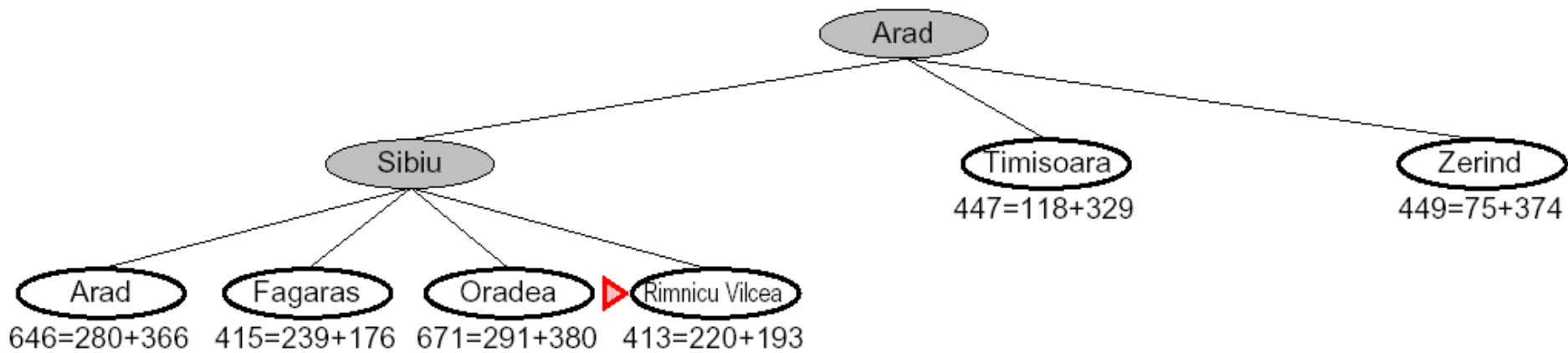
- Idea: avoid expanding paths that are already expensive
- Evaluation function  $f(n) = g(n) + h(n)$
- $g(n)$  = cost so far to reach  $n$
- $h(n)$  = estimated cost from  $n$  to goal
- $f(n)$  = estimated total cost of path through  $n$  to goal

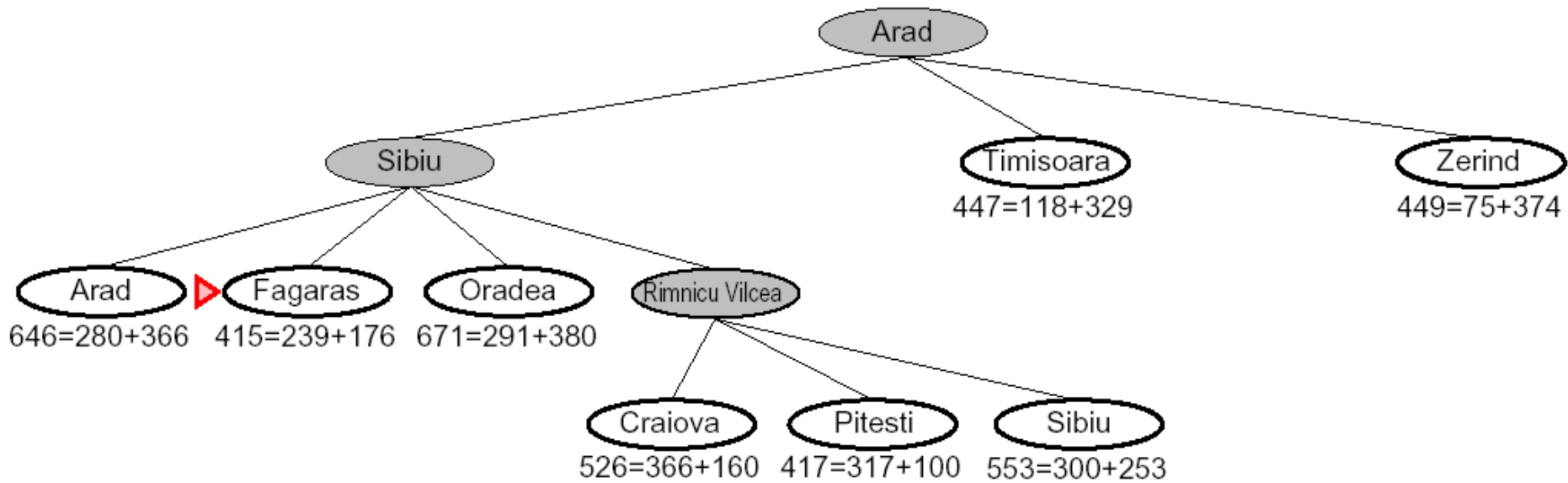
# A\* for Romanian Shortest Path

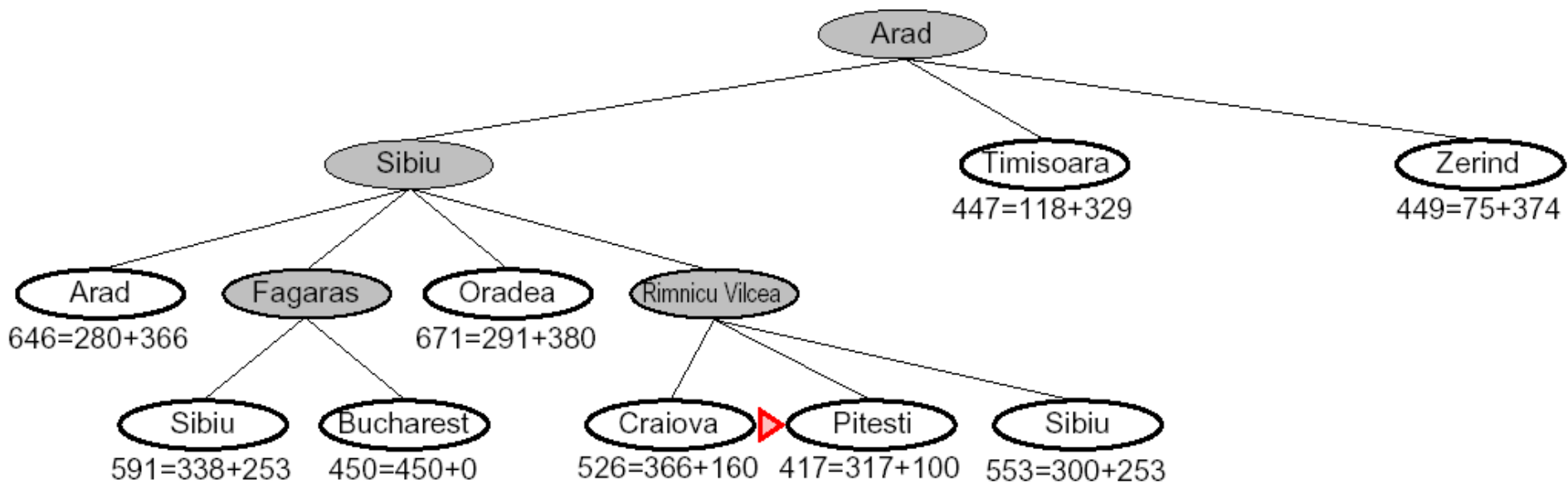


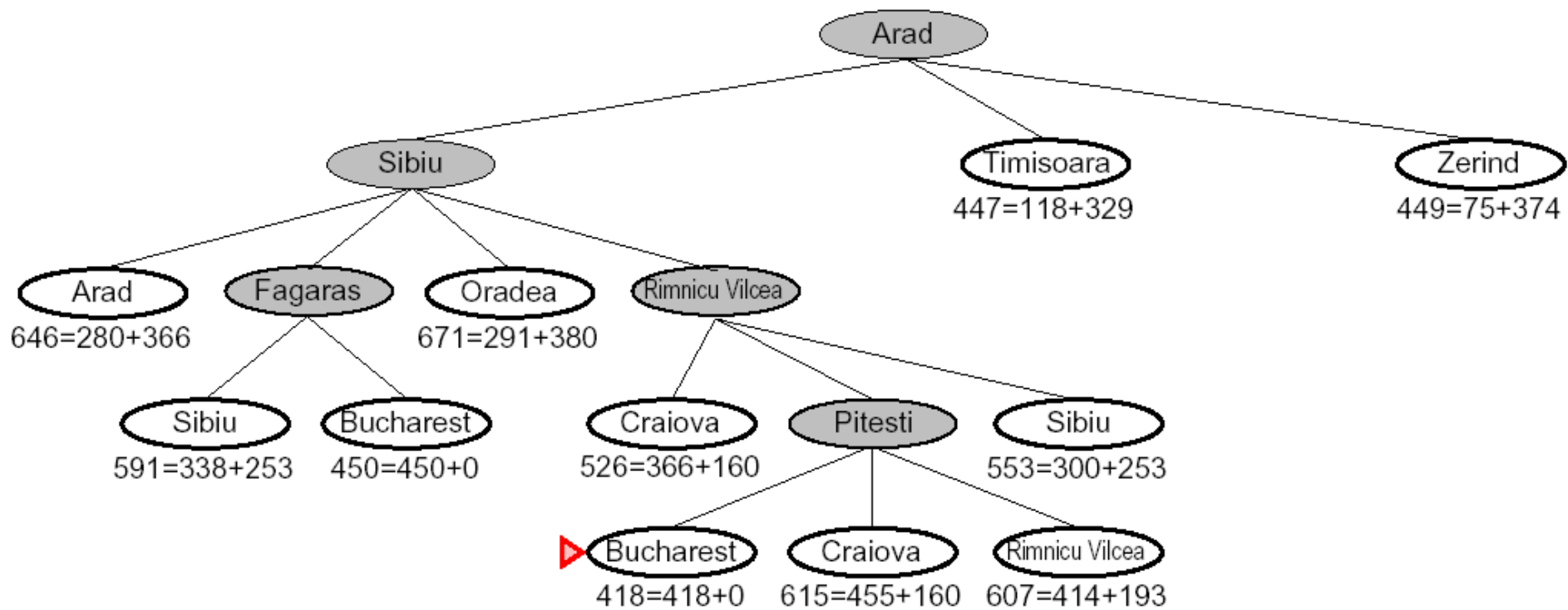










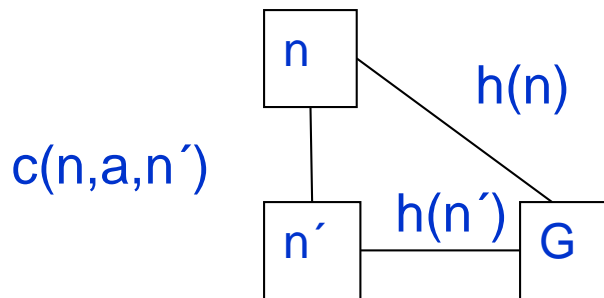


# Admissible heuristics

- A heuristic  $h(n)$  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

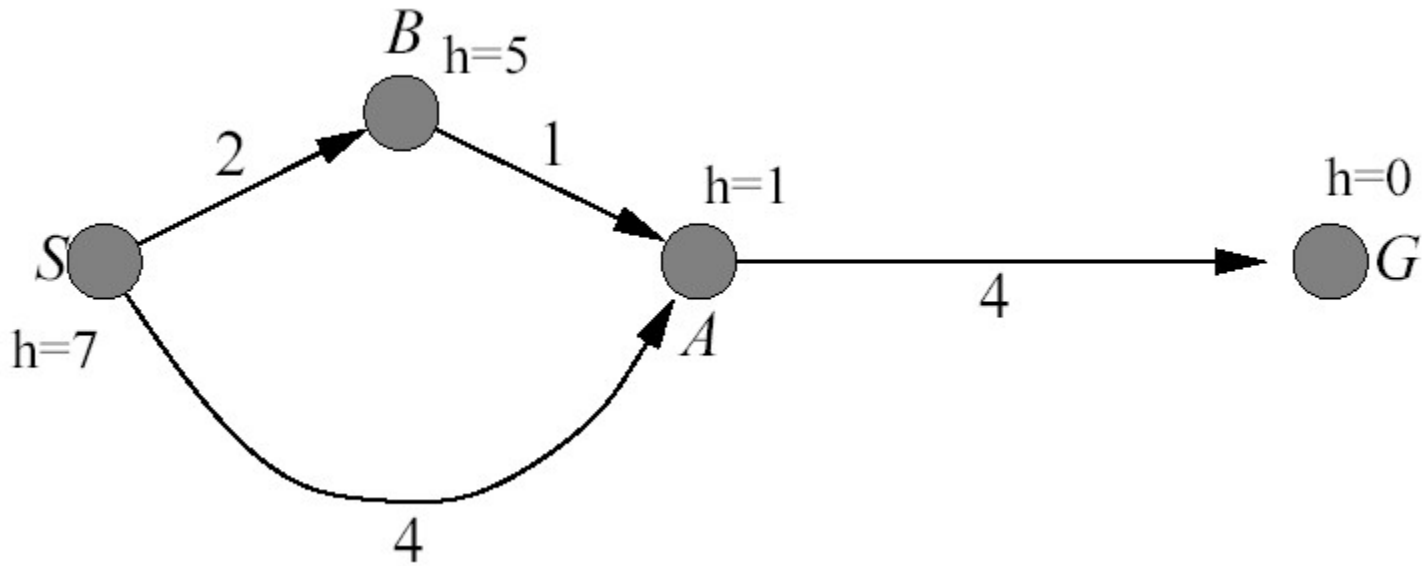
# Consistent Heuristics

- $h(n)$  is consistent if
  - for every node  $n$
  - for every successor  $n'$  due to legal action  $a$
  - $h(n) \leq c(n,a,n') + h(n')$



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

# Example

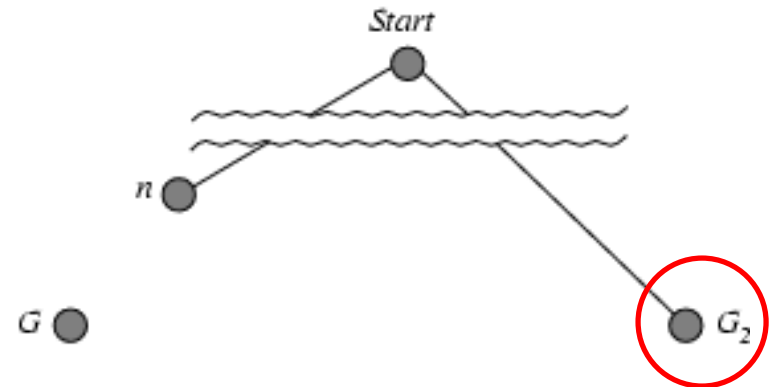


Source: <http://stackoverflow.com/questions/25823391/suboptimal-solution-given-by-a-search>



# Proof of Optimality of (Tree)A\*

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



## Focus on $G_2$ :

$$f(G_2) = g(G_2)$$

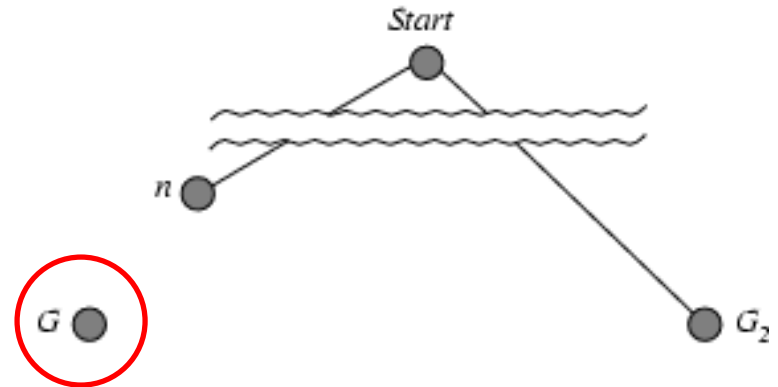
$$\text{since } h(G_2) = 0$$

$$g(G_2) > g(G)$$

$$\text{since } G_2 \text{ is suboptimal}$$

# Proof of Optimality of (Tree)A\*

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



$$f(G_2) = g(G_2)$$

$$\text{since } h(G_2) = 0$$

$$g(G_2) > g(G)$$

$$\text{since } G_2 \text{ is suboptimal}$$

**Focus on G:**

$$f(G) = g(G)$$

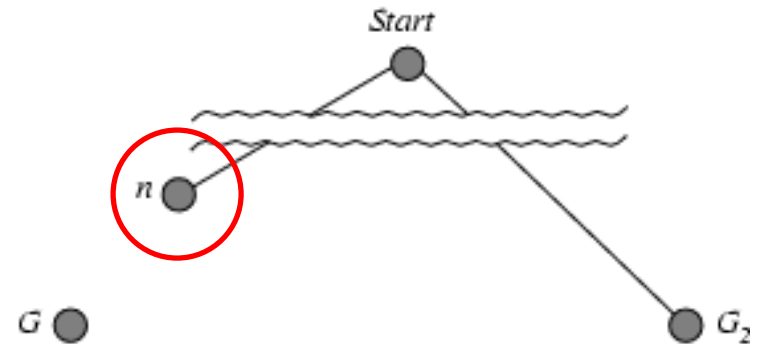
$$\text{since } h(G) = 0$$

$$f(G_2) > f(G)$$

$$\text{substitution}$$

# Proof of Optimality of (Tree) $A^*$

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



$$f(G_2) = g(G_2) \quad \text{since } h(G_2) = 0$$

$$g(G_2) > g(G) \quad \text{since } G_2 \text{ is suboptimal}$$

$$f(G) = g(G) \quad \text{since } h(G) = 0$$

$$f(G_2) > f(G) \quad \text{substitution}$$

**Now focus on n:**

$$h(n) \leq h^*(n) \quad \text{since } h \text{ is admissible}$$

$$g(n) + h(n) \leq g(n) + h^*(n) \quad \text{algebra}$$

$$f(n) = g(n) + h(n) \quad \text{definition}$$

$$f(G) = g(n) + h^*(n) \quad \text{by assumption}$$

$$f(n) \leq f(G) \quad \text{substitution}$$

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  $f \leq f(G)$  )

- Time? Exponential

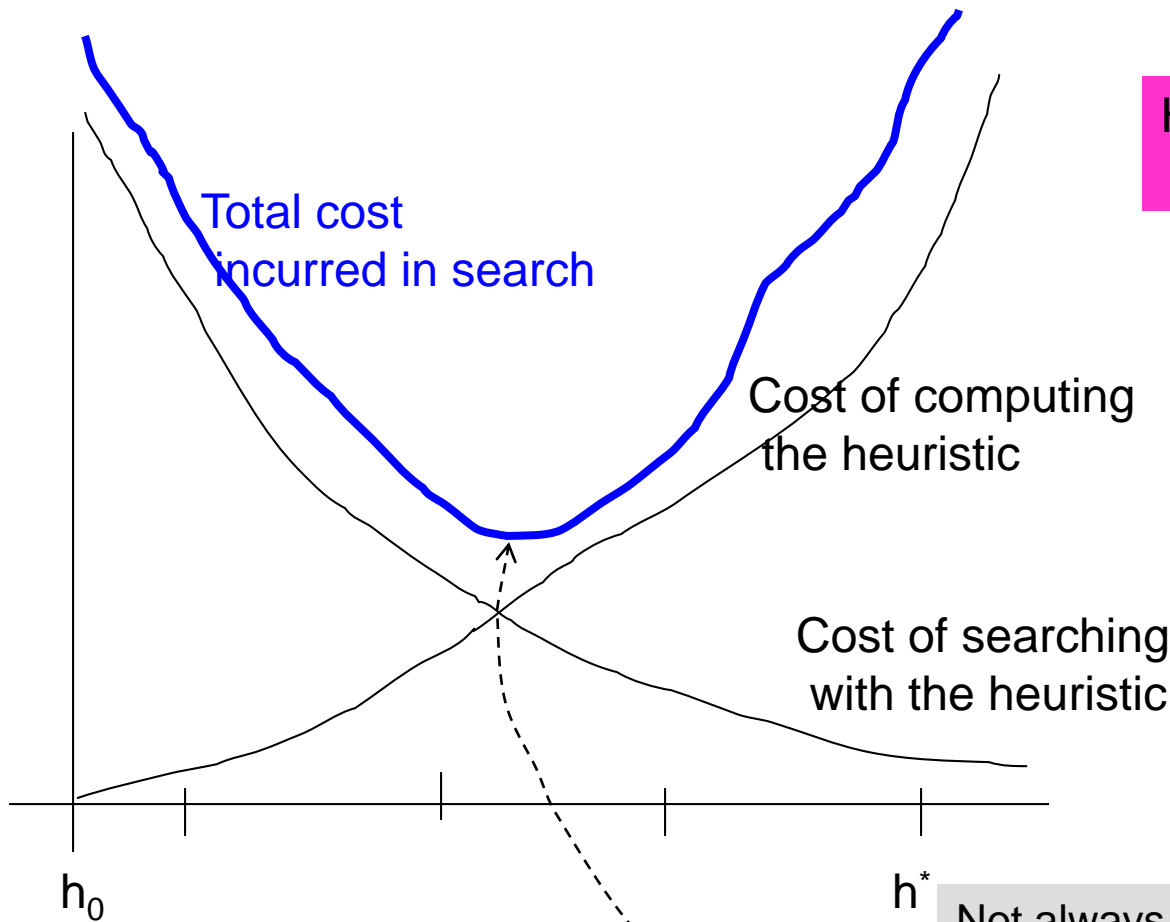
- Space? Keeps all nodes in memory

- Optimal?

Yes (depending upon search algo and heuristic property)

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

How informed should the heuristic be?



Reduced level of abstraction  
(i.e. more and more concrete)

- 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

# Memory Problem?

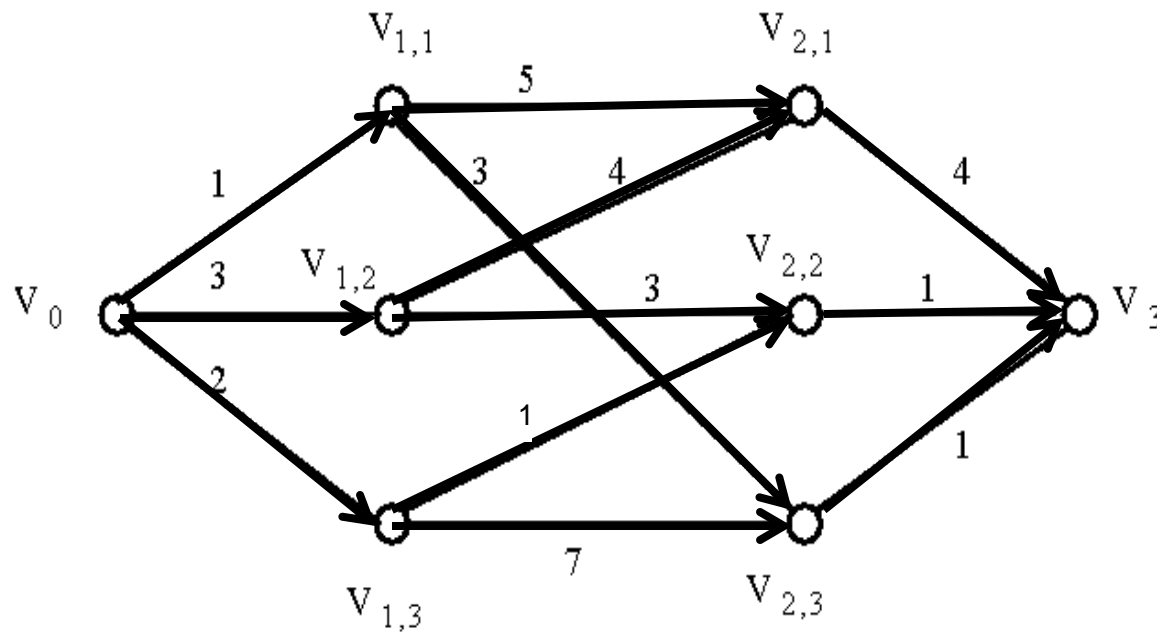
- Iterative deepening A\*
  - Similar to ID search
  - While (solution not found)
    - Do DFS but prune when cost (f) > current bound
    - Increase bound

# 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

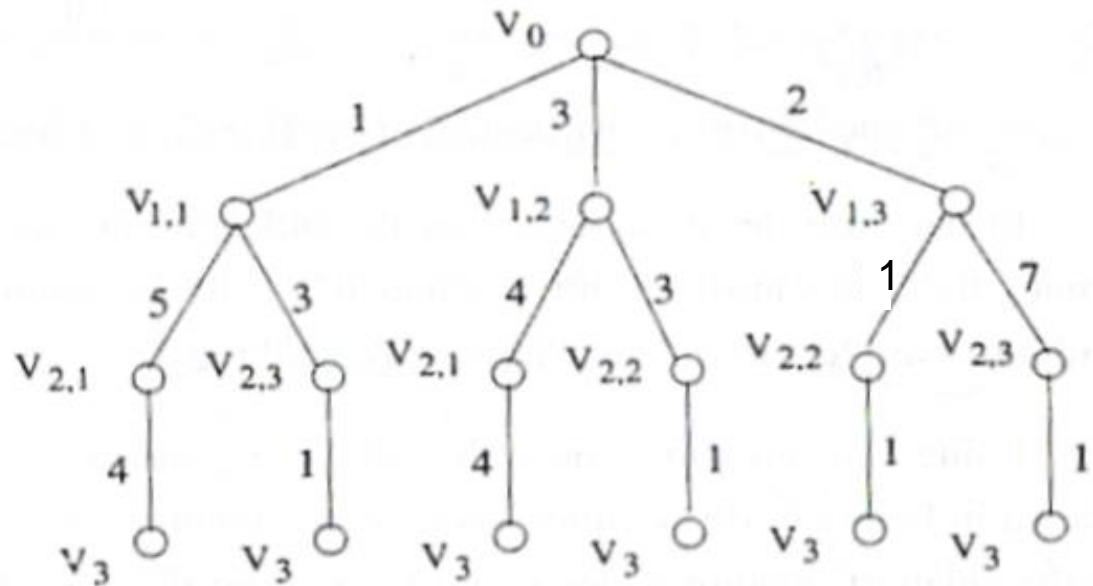
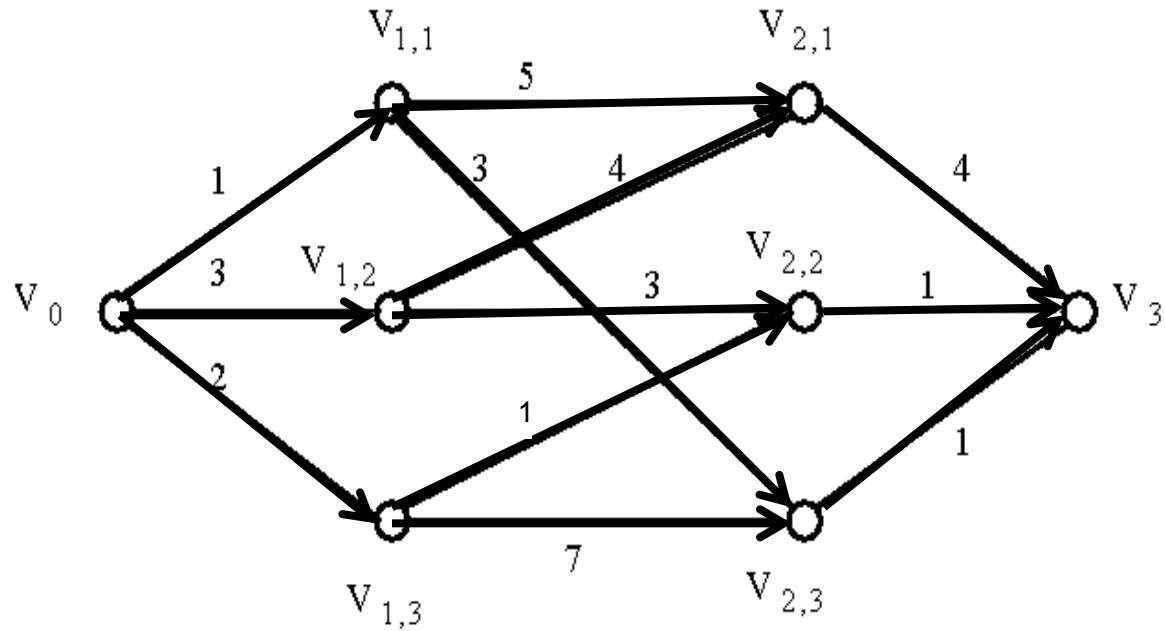
# A Multi-Stage Graph Searching Problem.

Find the shortest path from  $V_0$  to  $V_3$

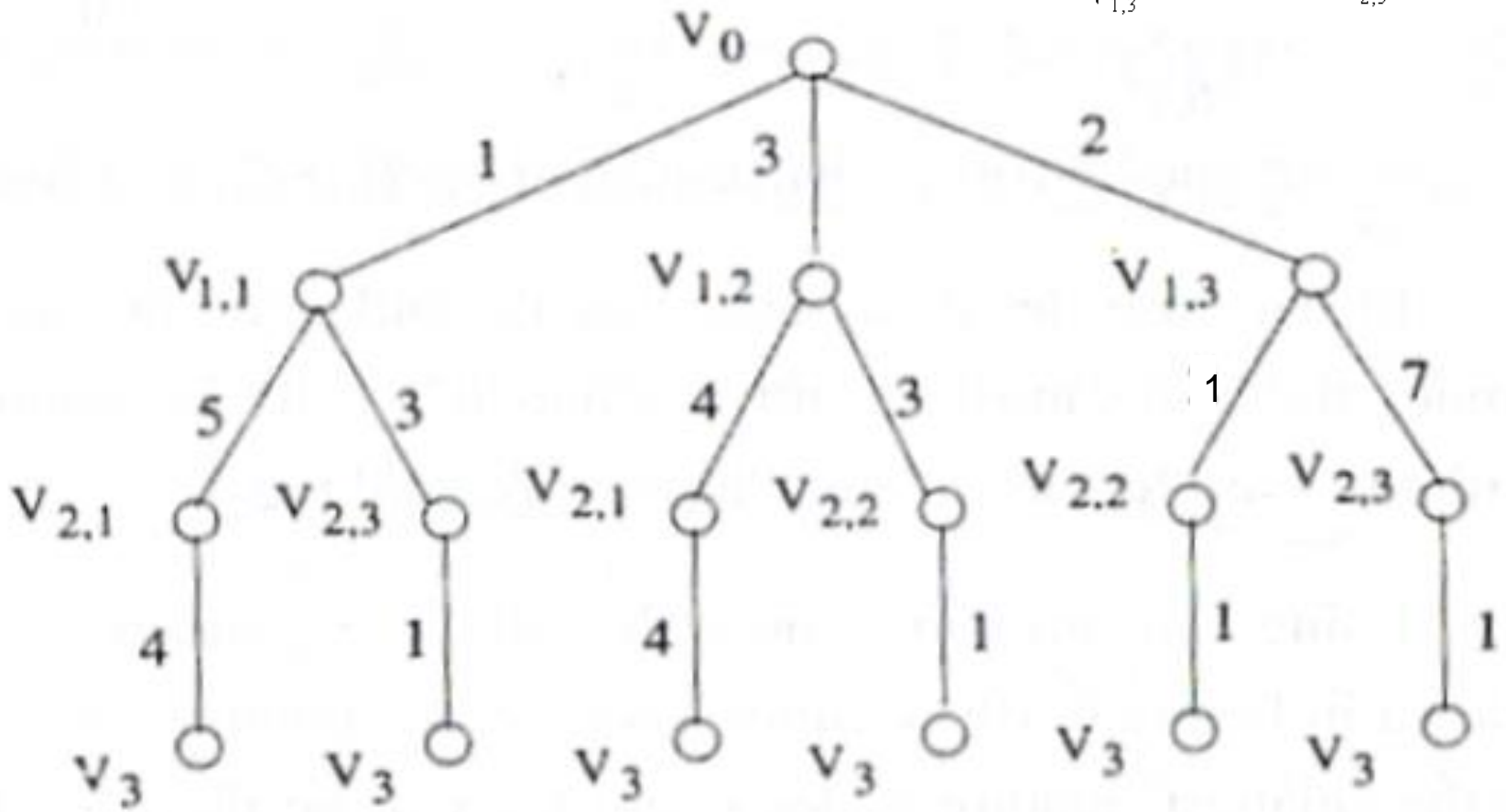
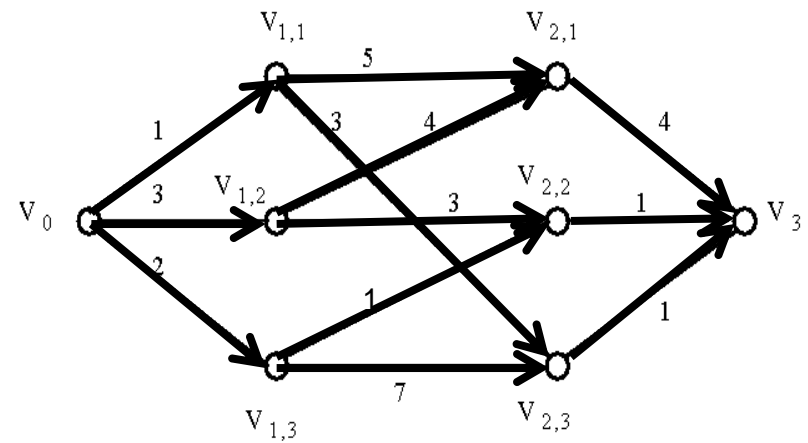




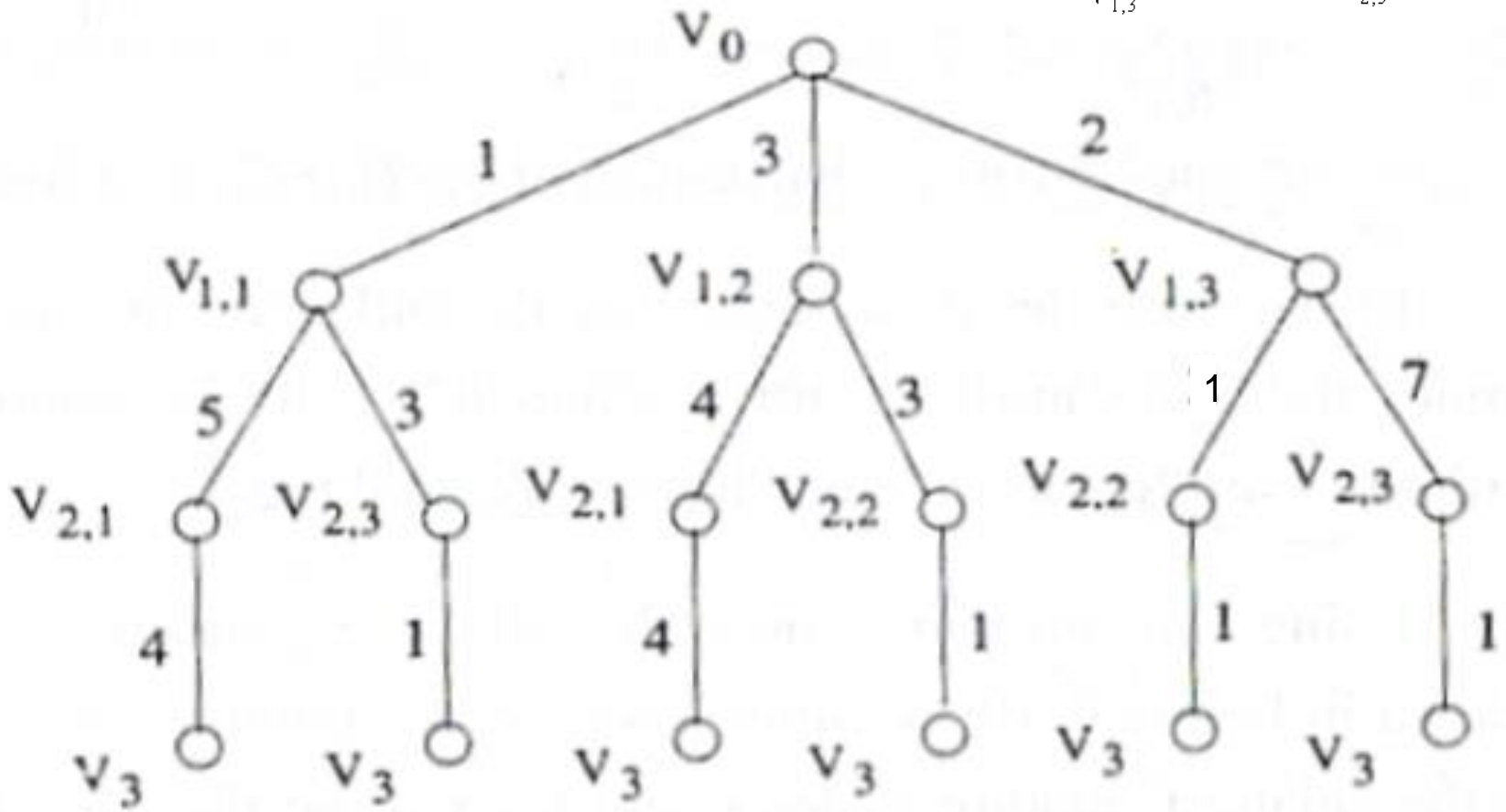
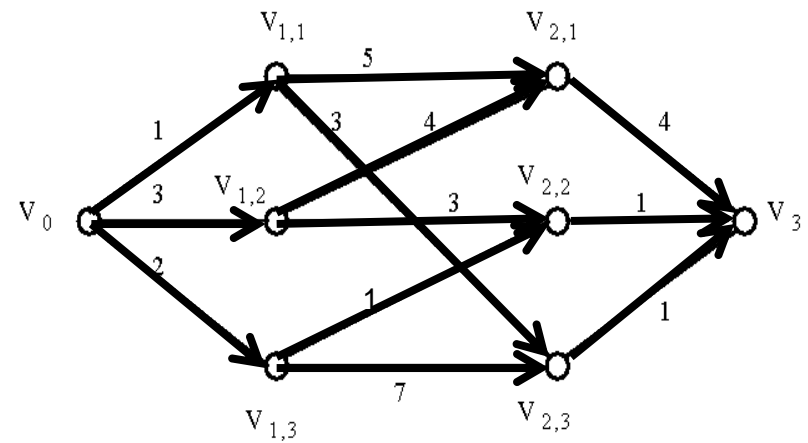
# E.G.:A Multi-Stage Graph Searching Problem



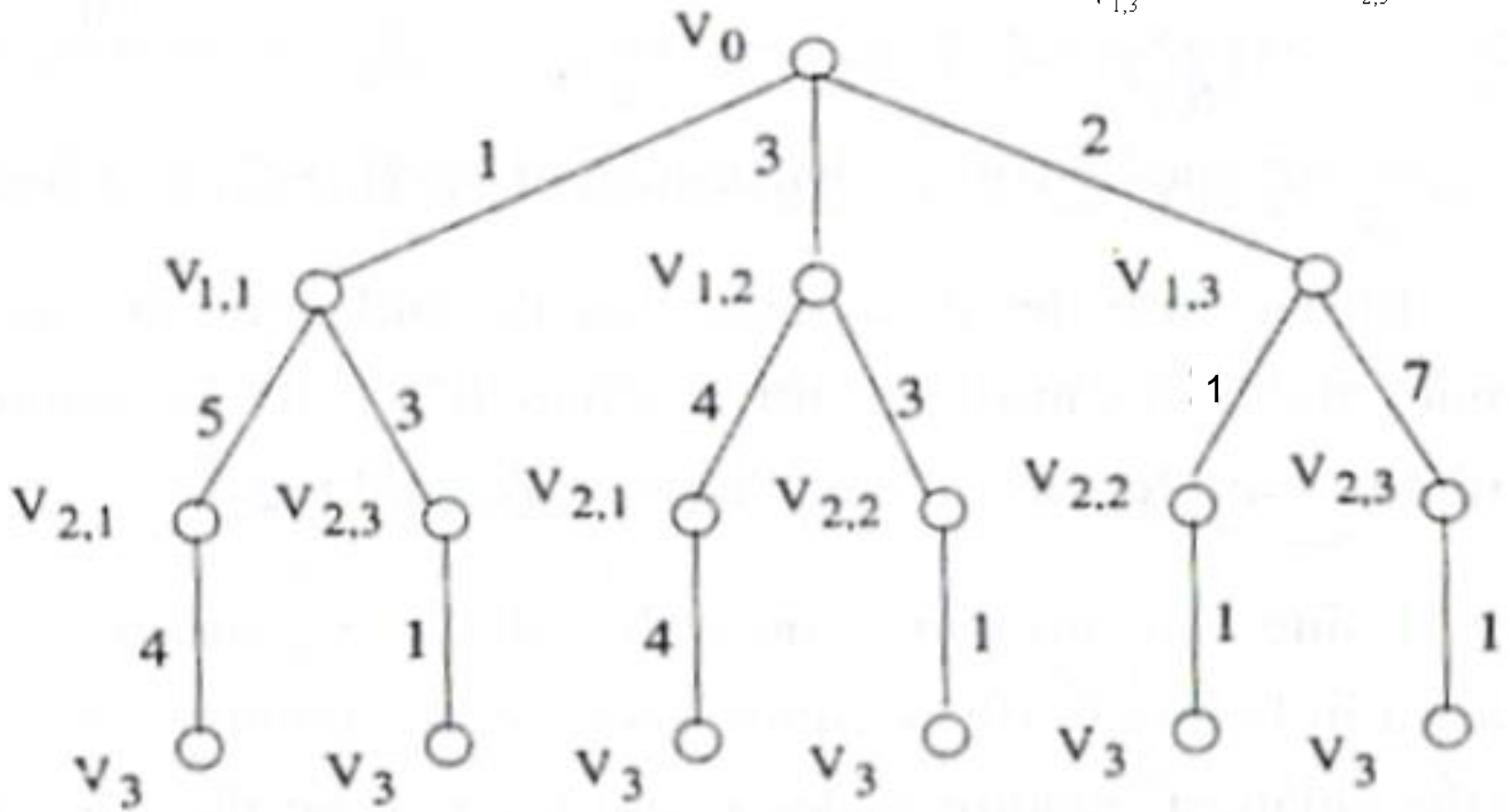
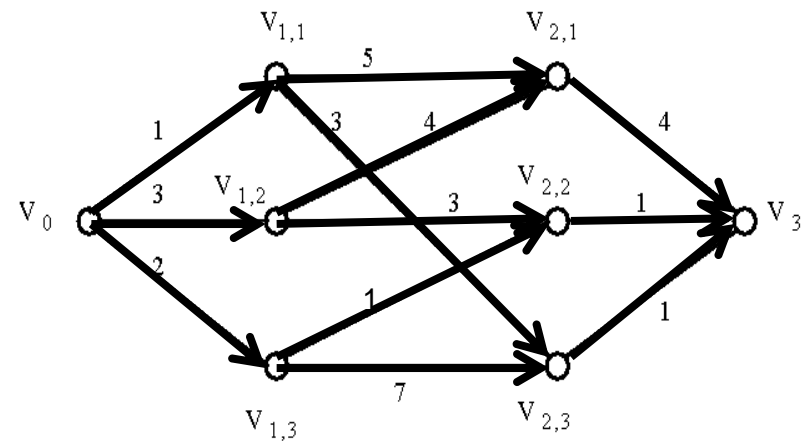
# Dfs-B&B



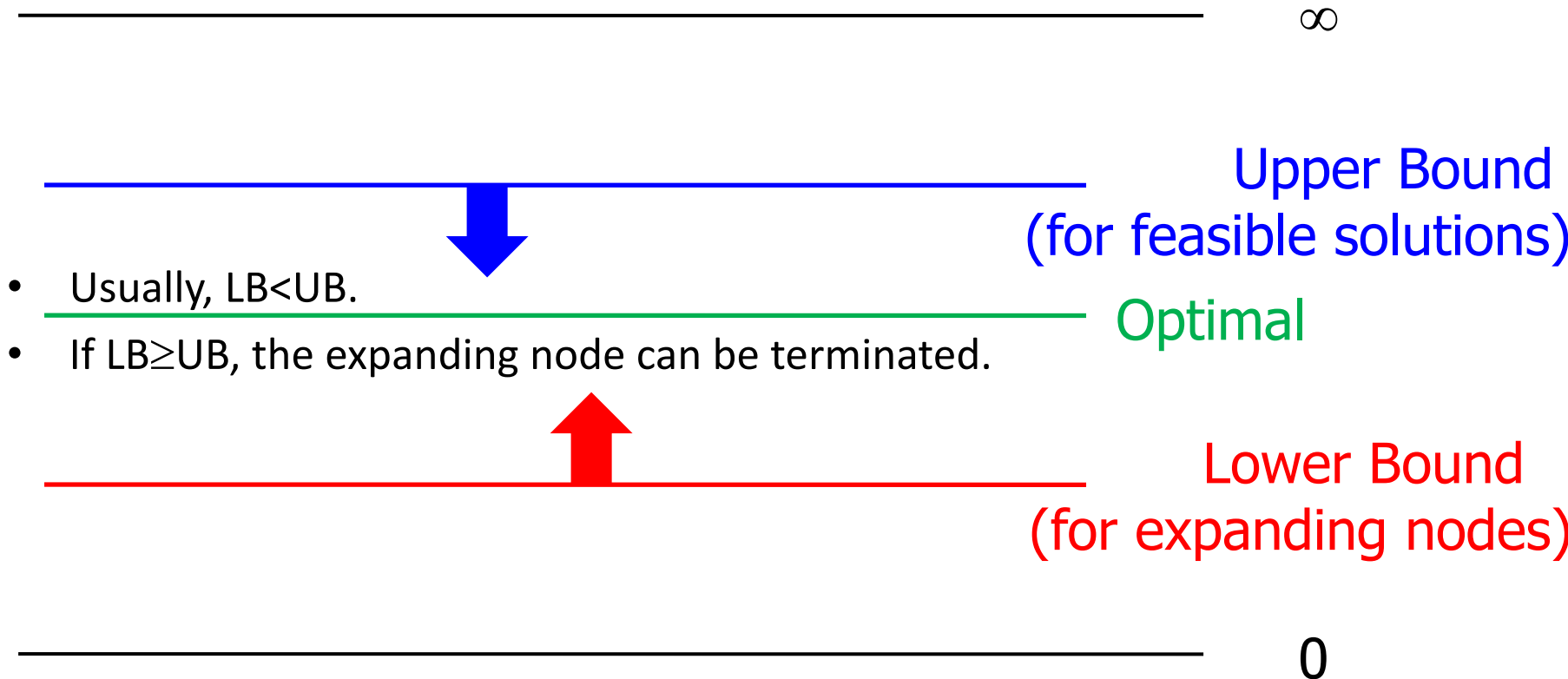
# Dfs-B&B



# Dfs-B&B



# For Minimization Problems



# DFS B&B vs. IDA\*

- Both optimal
- IDA\* never expands a node with  $f > \text{optimal cost}$ 
  - 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\*
  - $f(n) = g(n) + w \cdot h(n)$  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)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

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)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = ?$  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 all nodes exactly once

What is a good admissible heuristic for  $(a_1 \rightarrow a_2 \rightarrow \dots \rightarrow a_k)$

- length of the cheapest edge leaving  $a_k$  +  
length of cheapest edge entering  $a_1$
- length of shortest path from  $a_k$  to  $a_1$
- length of minimum spanning tree of rest of the nodes

# 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

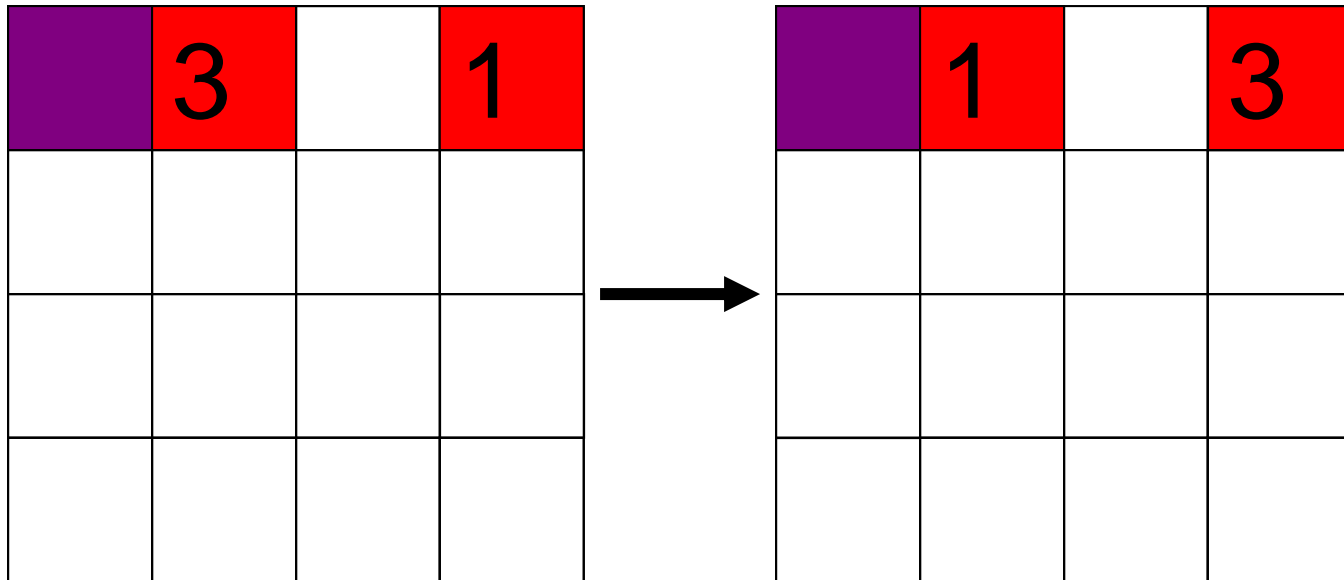
# Performance of IDA\* on 15 Puzzle

- 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.

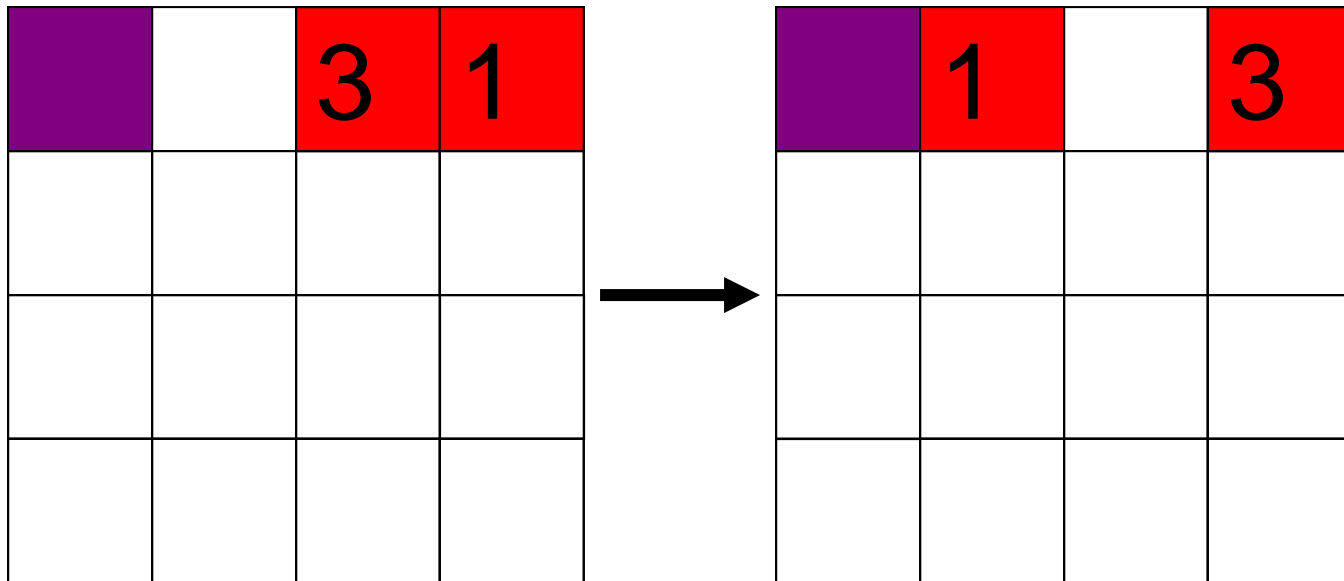
# Example: Linear Conflict



Manhattan distance is  $2+2=4$  moves

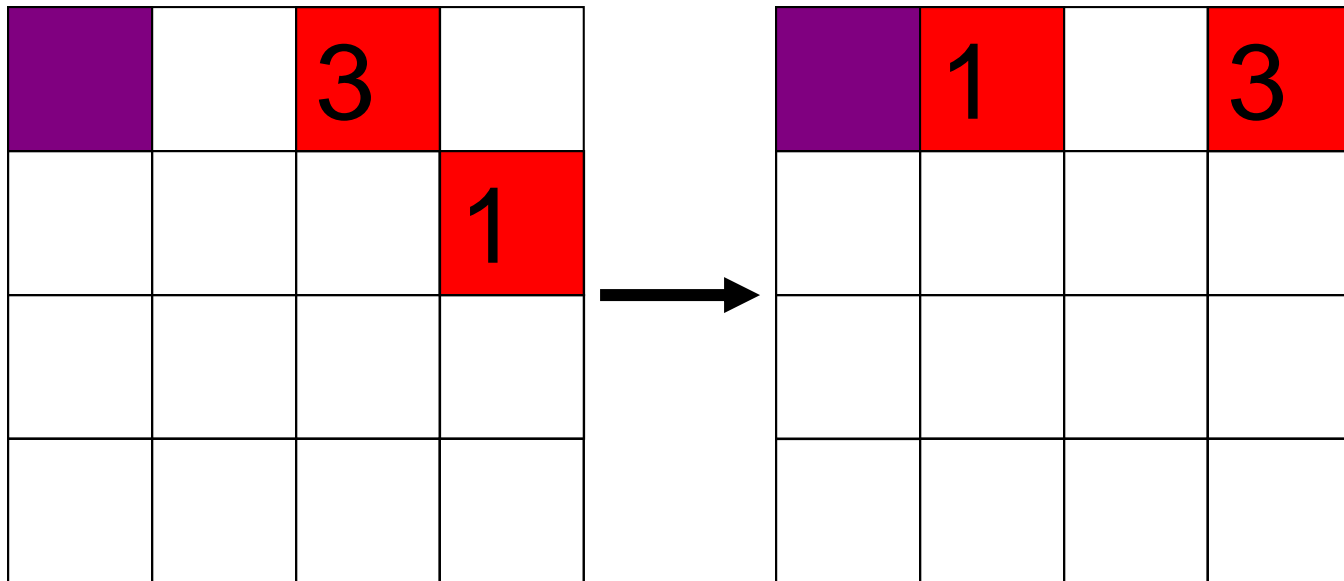


# Example: Linear Conflict



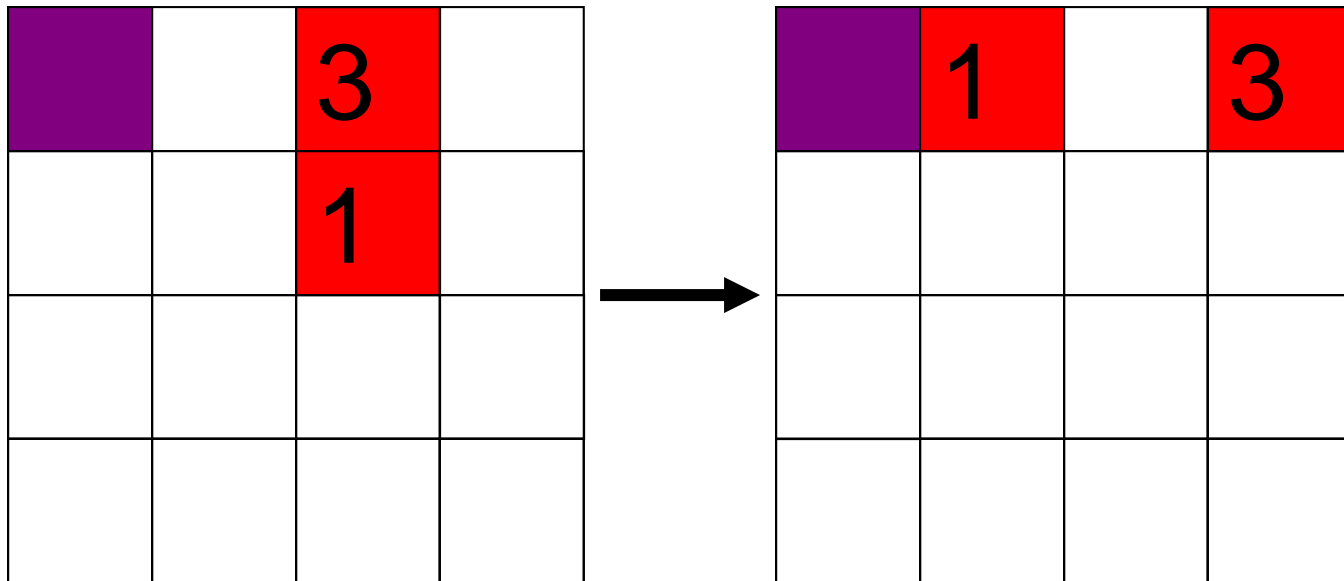
Manhattan distance is  $2+2=4$  moves

# Example: Linear Conflict



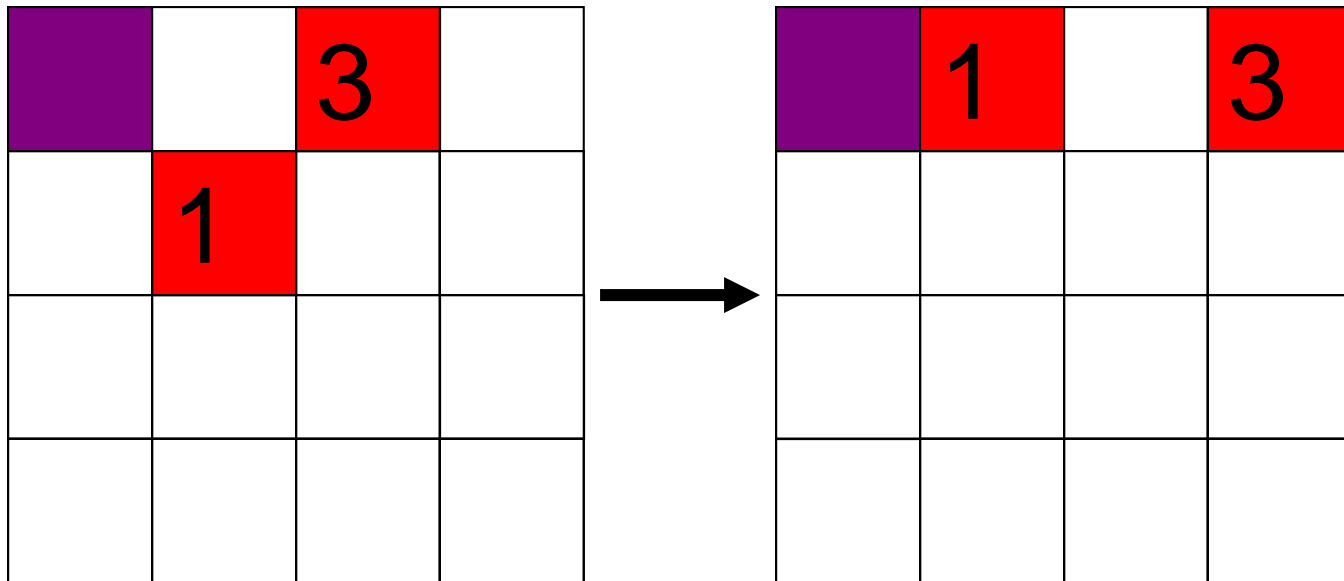
Manhattan distance is  $2+2=4$  moves

# Example: Linear Conflict



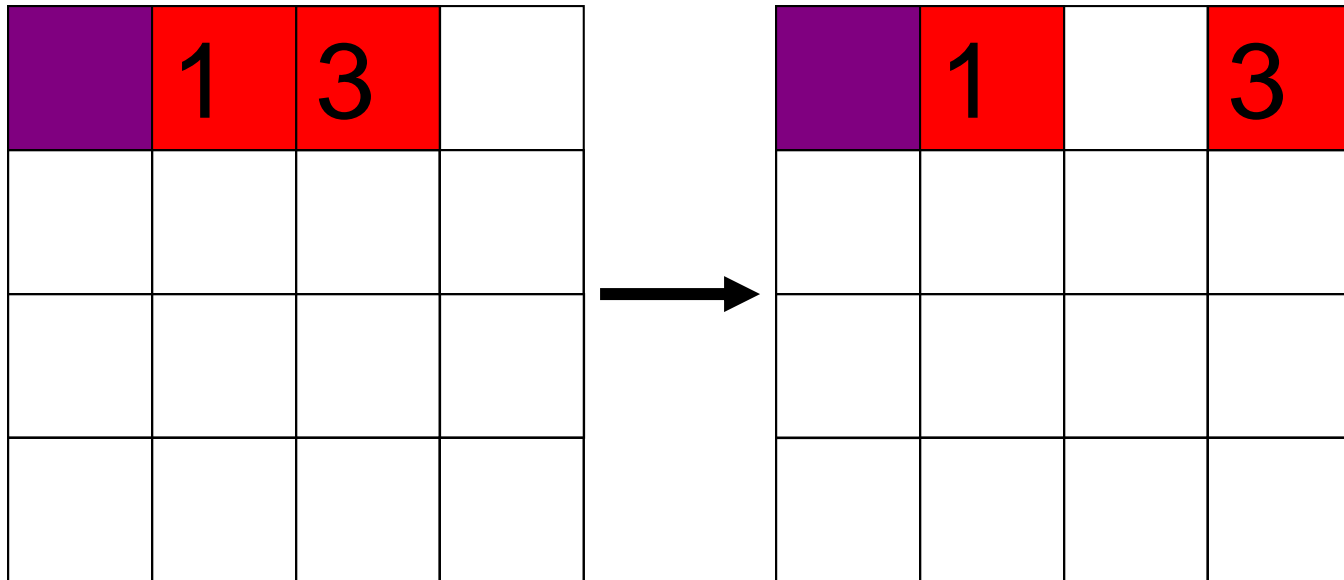
Manhattan distance is  $2+2=4$  moves

# Example: Linear Conflict



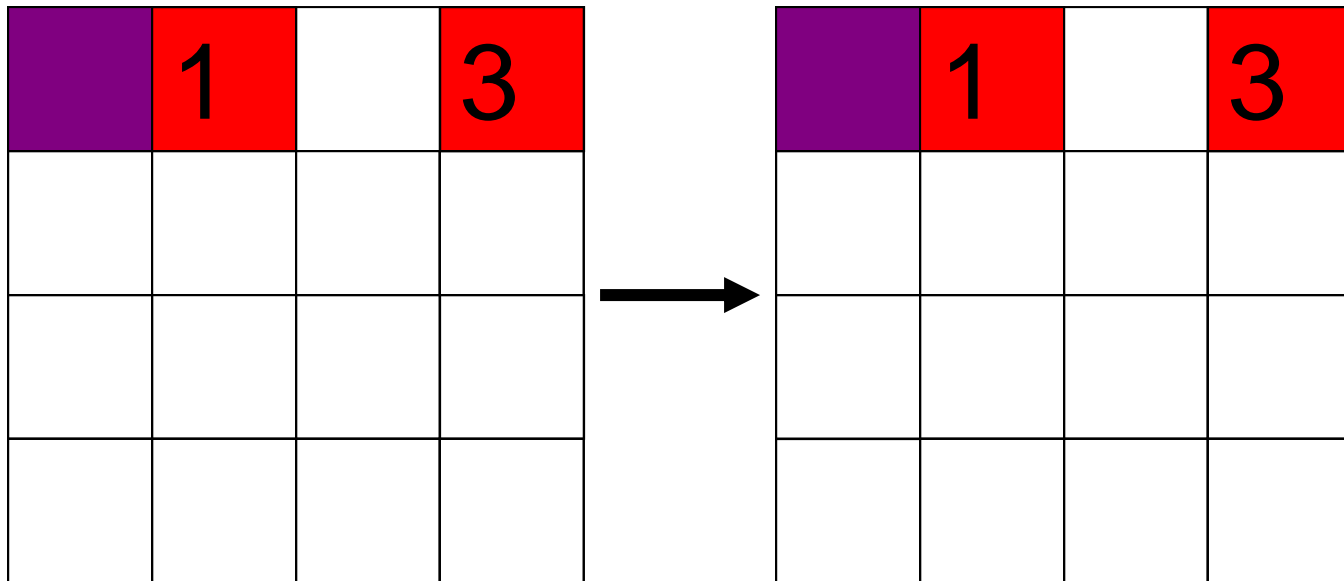
Manhattan distance is  $2+2=4$  moves

# Example: Linear Conflict



Manhattan distance is  $2+2=4$  moves

# Example: Linear Conflict

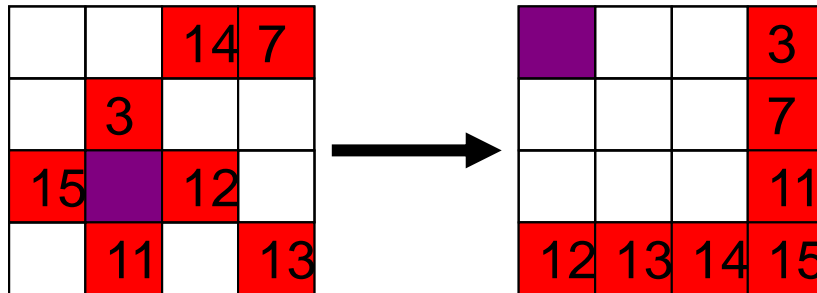


Manhattan distance is  $2+2=4$  moves, but linear conflict adds 2 additional moves.

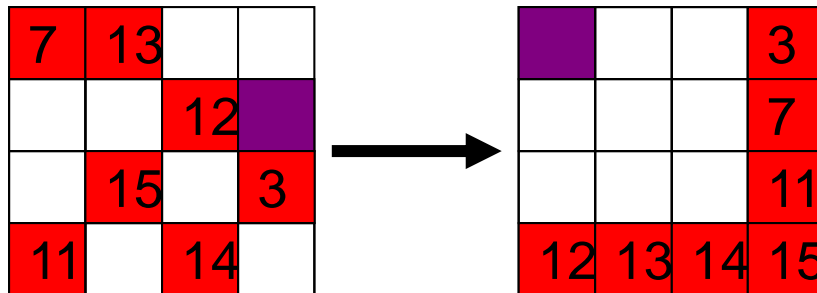
# Linear Conflict Heuristic

- 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.

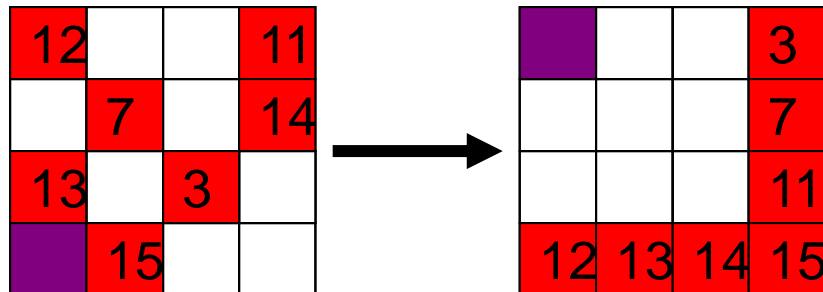
# More Complex Tile Interactions



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



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



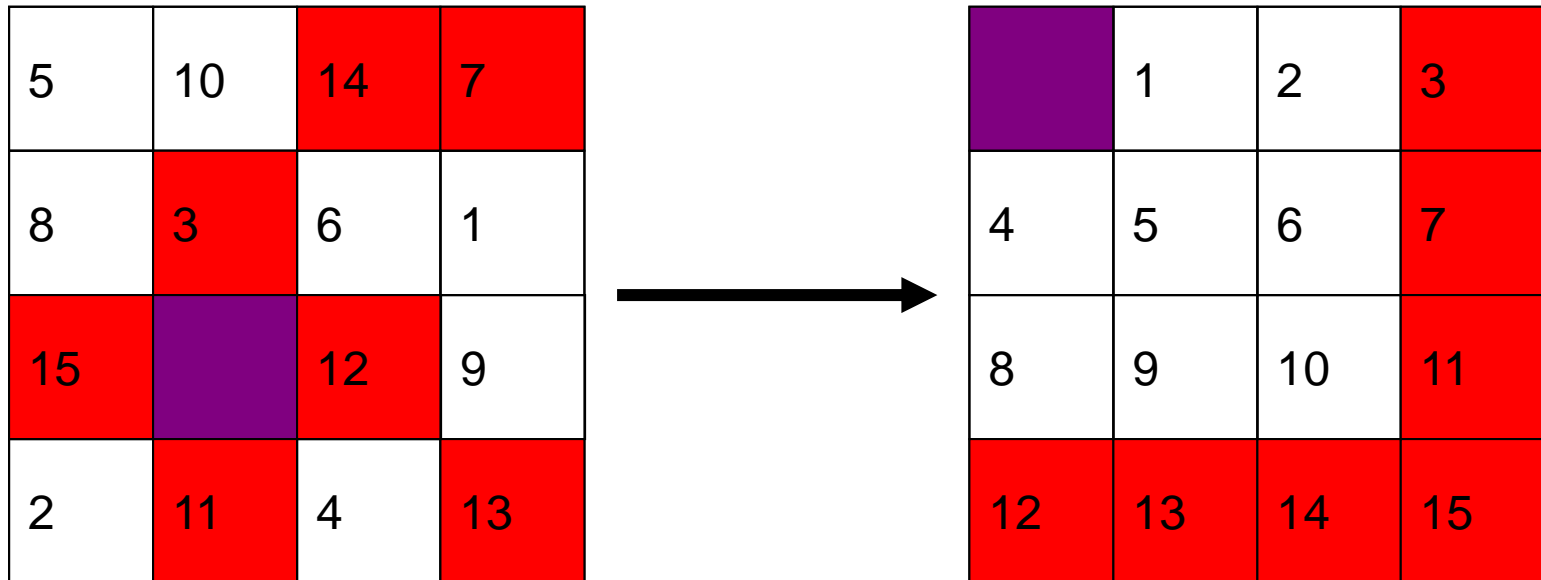
M.d. is 17 moves, but 27 moves are needed



# 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.

# Heuristics from Pattern Databases

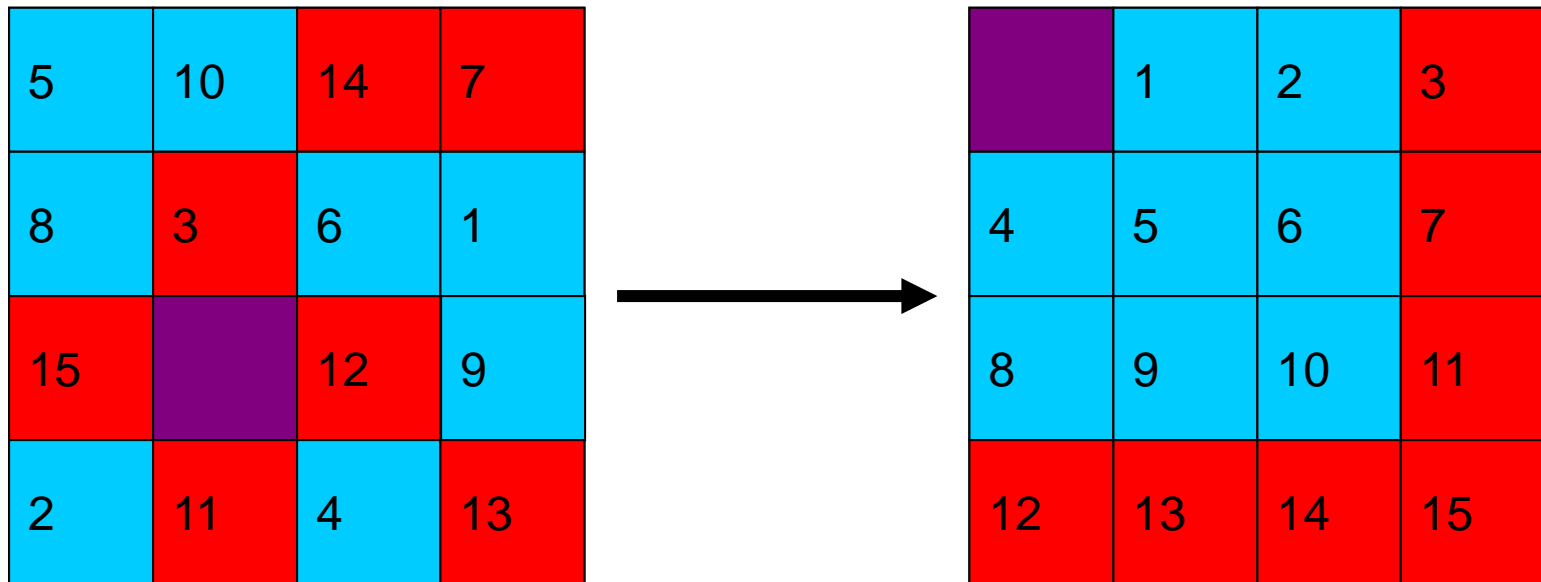


31 moves is a lower bound on the total number of moves needed to solve this particular state.

# 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

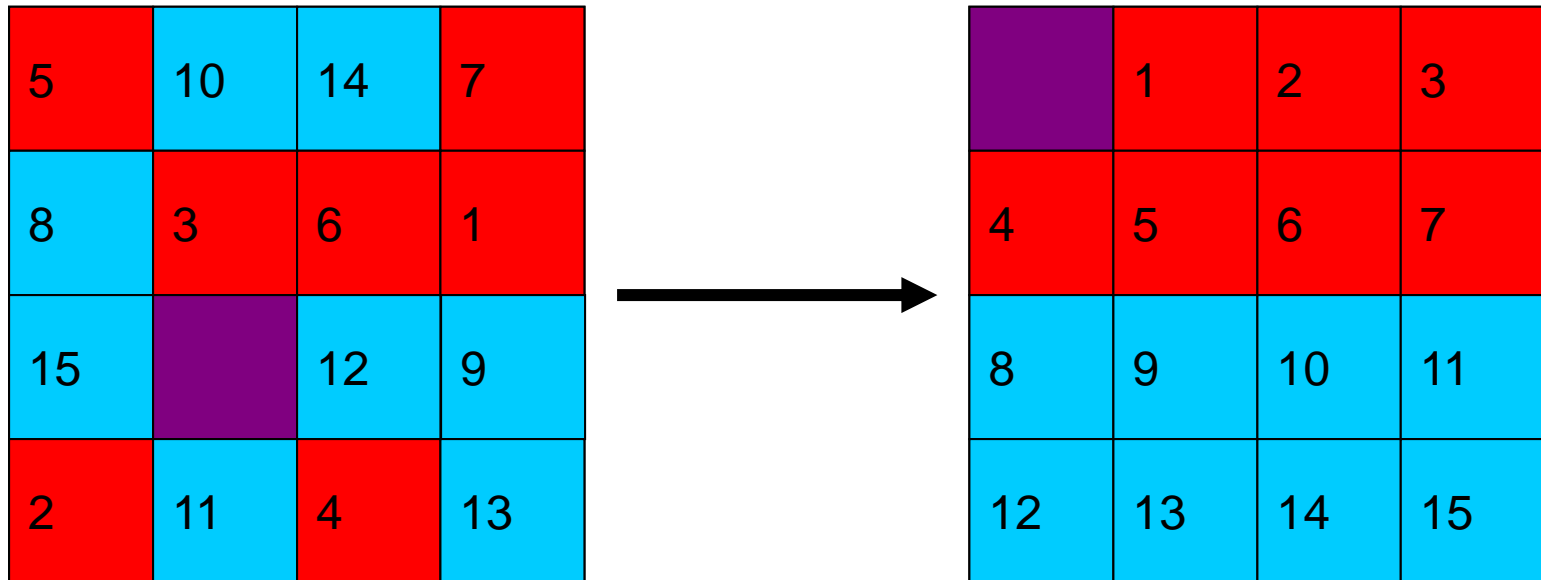
- Culberson and Schaeffer counted all moves needed to correctly position the pattern tiles.
- In contrast, we count only moves of the pattern tiles, ignoring non-pattern moves.
- If no tile belongs to more than one pattern, then we can add their heuristic values.
- Manhattan distance is a special case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

# Example Additive Databases

	1	2	3
4	5	6	7
8	9	10	11
12	13	15	14

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

