Informed Search & Heuristics

Dr. Fayyaz ul Amir Afsar Minhas
PIEAS Biomedical Informatics Research Lab
Department of Computer and Information Sciences
Pakistan Institute of Engineering & Applied Sciences
PO Nilore, Islamabad, Pakistan
http://faculty.pieas.edu.pk/fayyaz/

Review of Tree Search Based Methods

- Search Strategies
  - Determines Order of Node Expansion

```
function TREE-SEARCH(problem, strategy) return a solution or failure
    Initialize search tree to the initial state of the problem
    do
        if no candidates for expansion then return failure
        choose leaf node for expansion according to strategy
        if node contains goal state then return solution
        else expand the node and add resulting nodes to the search tree
    enddo
```

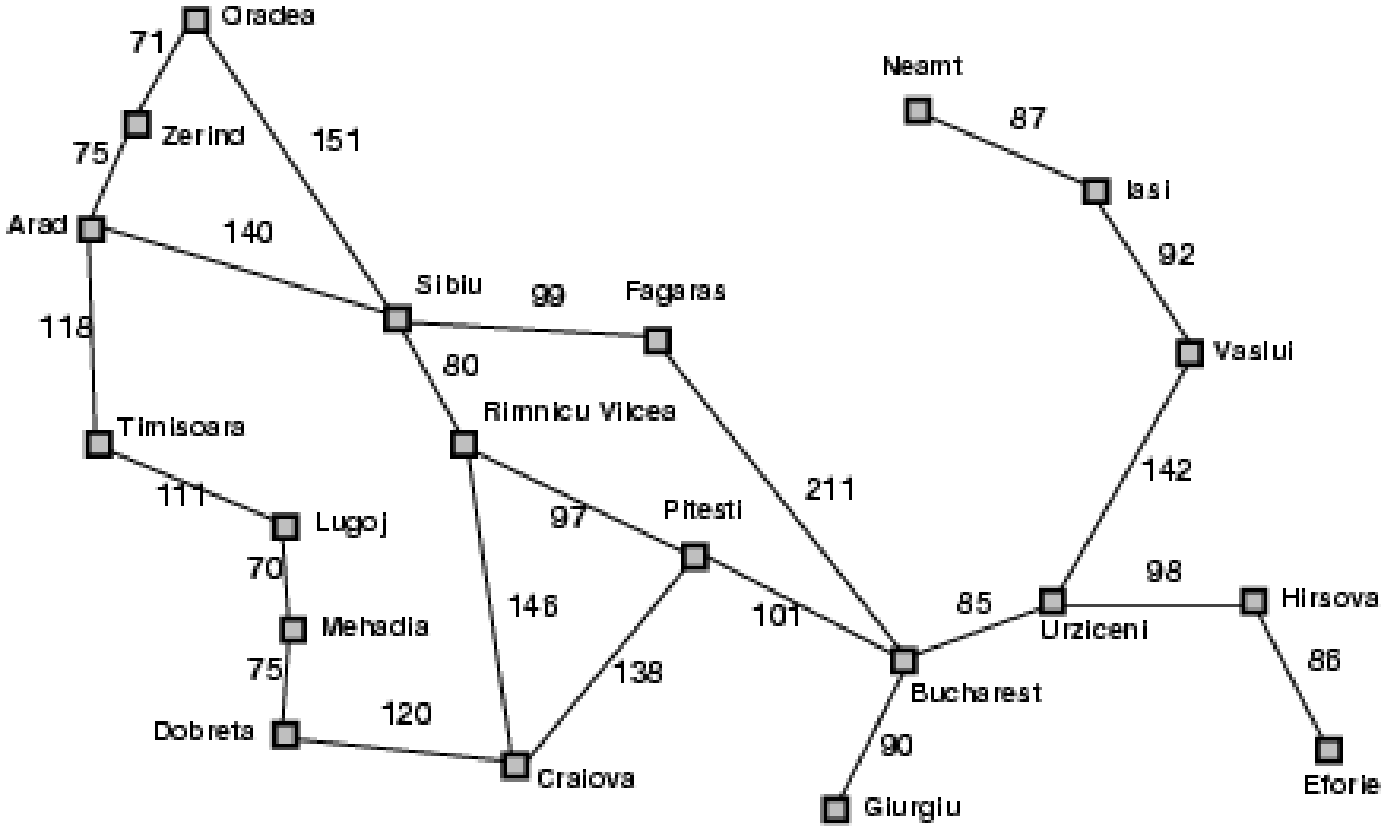
- Uninformed Search Strategies
  - Did not use problem specific knowledge beyond the definition of the problem itself

Informed Search Strategies

- Use information beyond the problem definition for selecting a particular node for expansion
- Order of node expansion is determined by an evaluation function f(n)
  - Estimate of "Desirability" of a node. Smaller f(n) means more "Desirability"
  - The node with the lowest value of the evaluation function is chosen for expansion
- Implemented by using a priority LIFO or FIFO queue
- Called "Best First Search"

Heuristics

- Definition
  - *A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood*
- Mathematically,
  - h(n) = estimated cost of cheapest path from node n to goal node such that h(n) = 0 when n is a goal node (some other constraints too)
- Example
  - Straight line distance from Bucharest, h_{SLD}

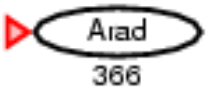


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

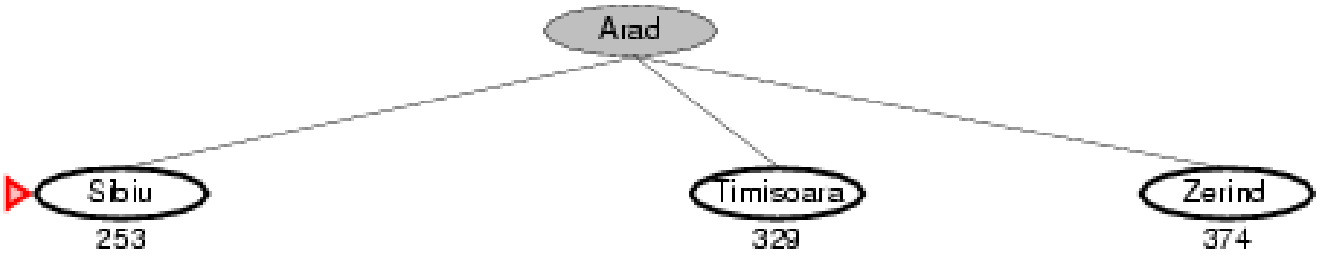
Greedy Best First Search

- Expands the node that is closest to the goal on the grounds that this likely to lead to a solution quickly
- The evaluation function is the same as the heuristic function
  - f(n) = h(n)
- Example:
  - Reaching Bucharest using $h_{SLD}(n)$ = straight-line distance from *n* to Bucharest
- Greedy
  - Because it tries to get as close to the goal node as possible

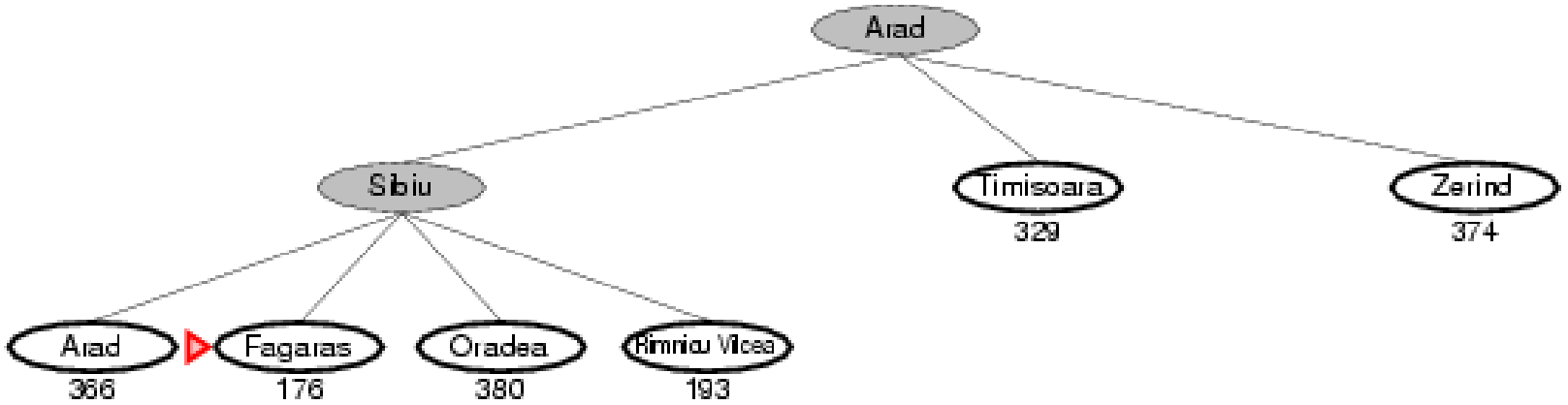
GBFS at Work



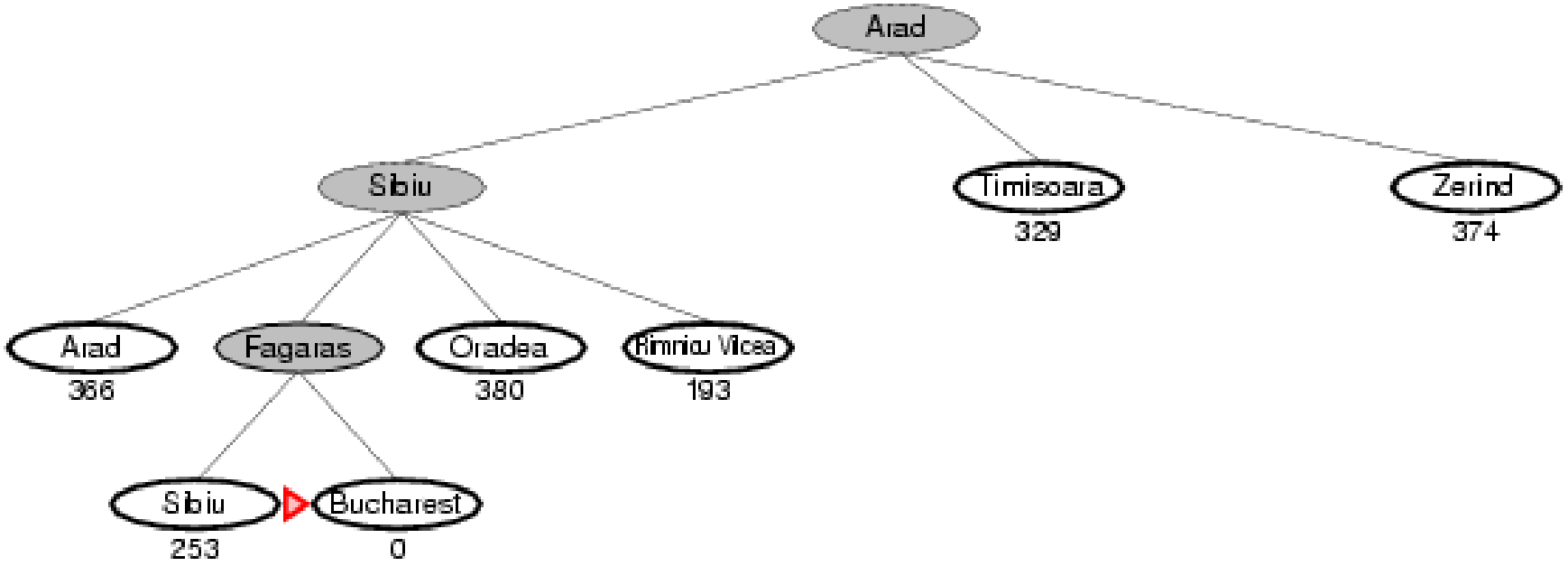
GBFS at Work…



GBFS at Work…

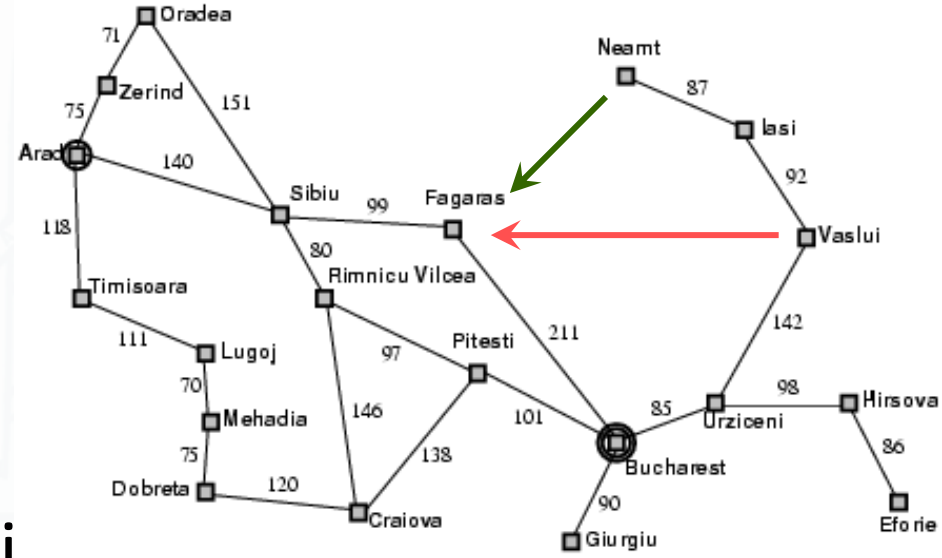

Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
176
Oradea
380
Rimnicu Vilcea
193

GBFS at Work…


Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0

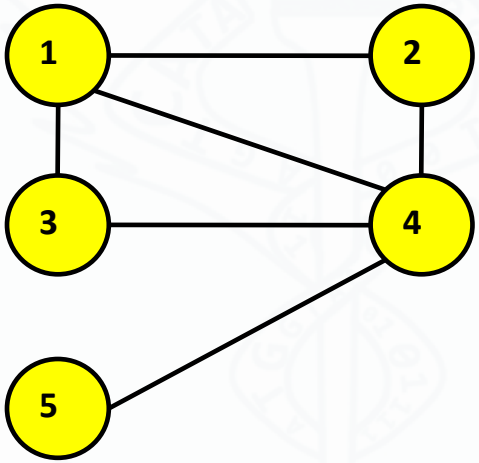
Properties of GBFS

- GBFS is similar to DFS in the way it prefers to follow a single path all the way to the goal but will back up when it hits a dead end
- Completeness
  - No
- Optimality
  - No
  - Example
    - Reach Fagaras from Iasi
- Time & Space Complexity
  - $O(b^m)$, can be reduced with a good h(n)



Greed!

- Reach from (1) to (5)
  - All path costs 1



"Thinking to get at once all the gold that the goose could give, he killed it, and opened it only to find—nothing."

Aesop (lived 6th century BC), Greek writer.
Aesop's Fables "The Goose with the Golden Eggs" (6th century BC).

A* Search

- Most widely-known form of Best First Search
- Its evaluation function, f(n), combines the cost [g(n)] of reaching to node n and the estimated cost [h(n)] of reaching to the goal from node n
  - f(n) = g(n) + h(n)
- Avoids expanding paths that are already expensive.

A* search

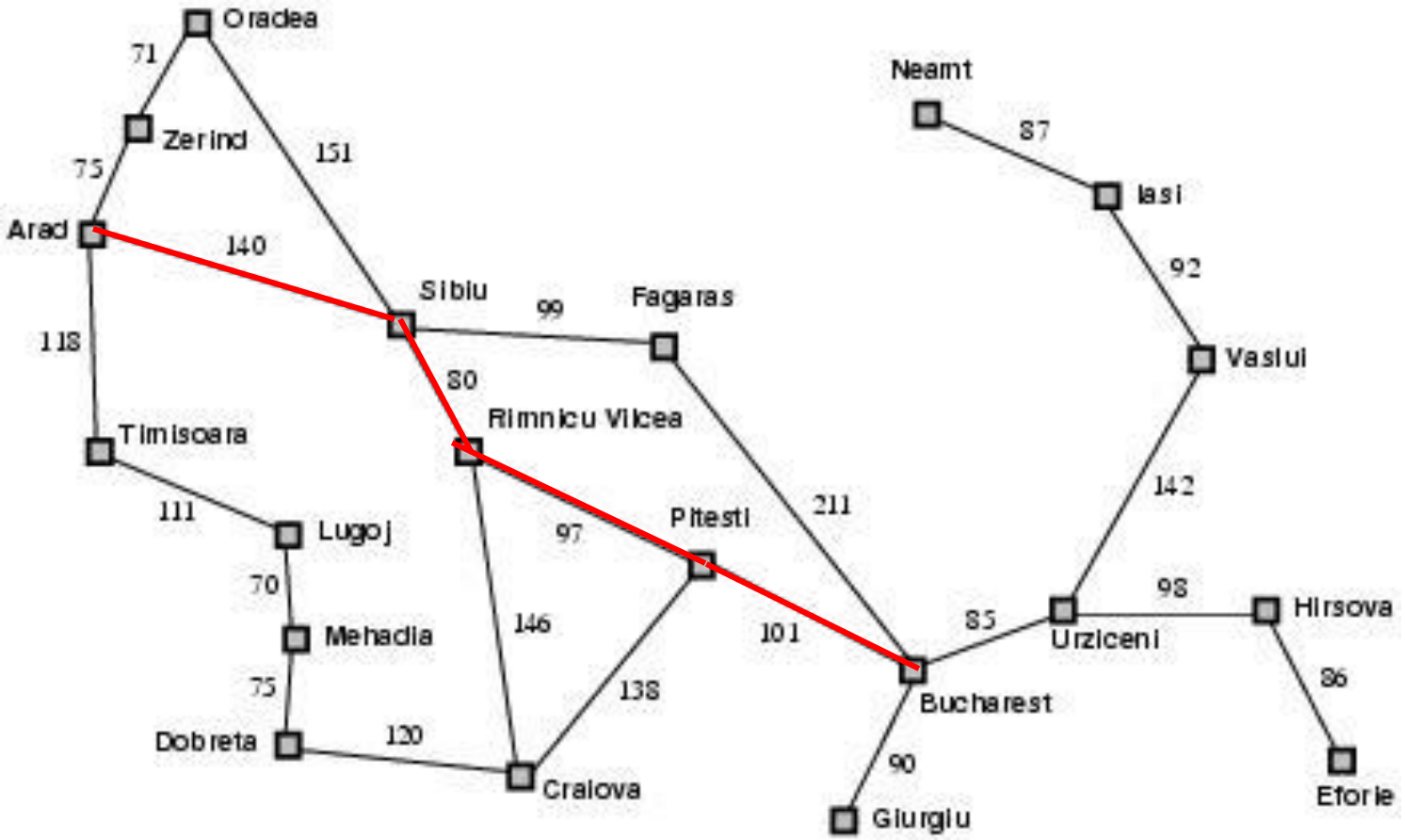
- A* search uses an admissible heuristic
  - A heuristic is admissible if it never overestimates the cost to reach the goal
  - Are optimistic

    Formally:

    1. h(n) <= h*(n) where h*(n) is the true cost from n
    2. h(n) >= 0 so h(G)=0 for any goal G.

    e.g. $h_{SLD}(n)$ never overestimates the actual road distance

Romania example



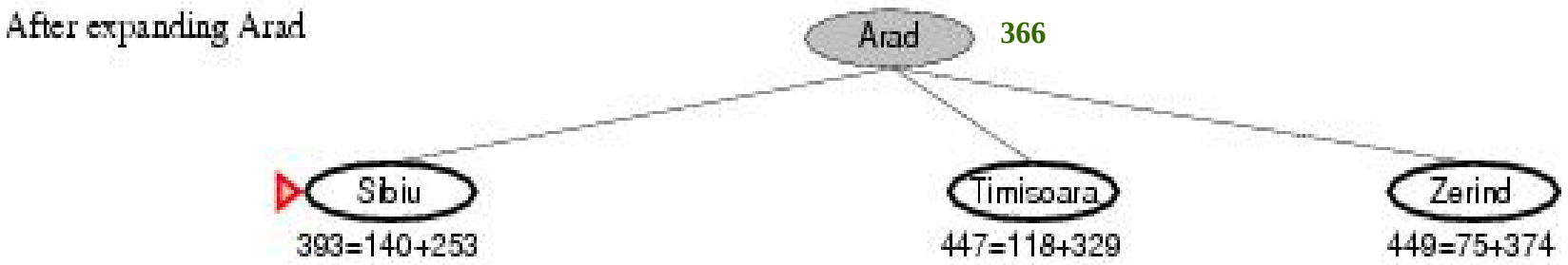
C* = 418

A* search example

(a) The initial state

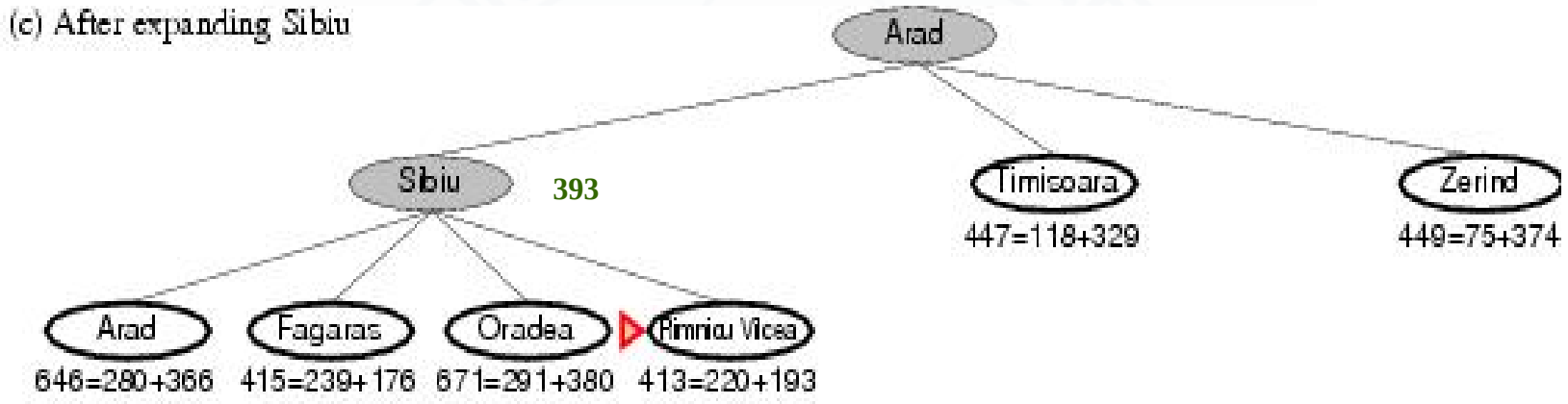
- Find Bucharest starting at Arad
  - f(Arad) = c(??,Arad)+h(Arad)=0+366=366

A* search example



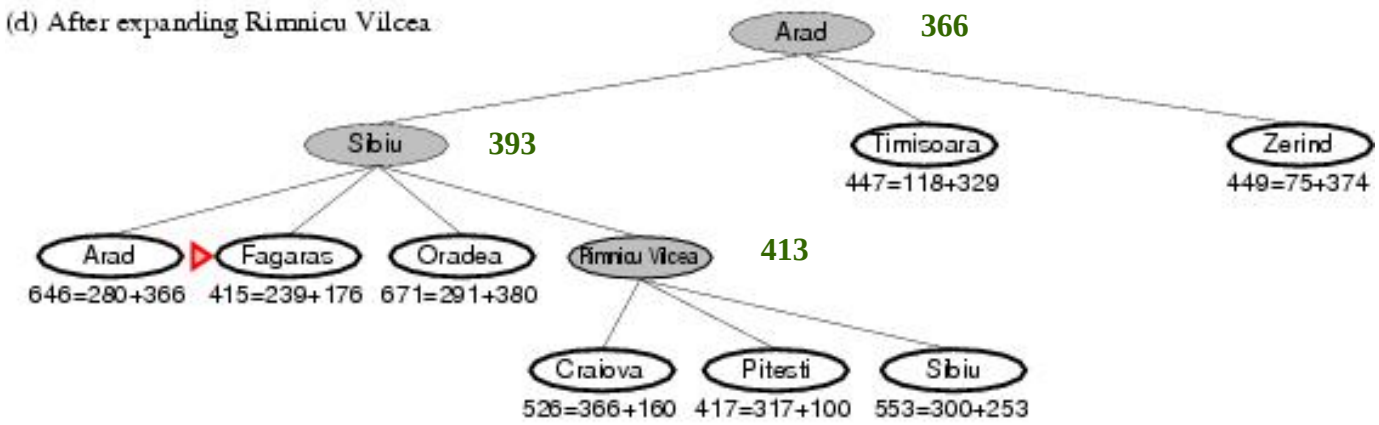
- Expand Arrad and determine f(n) for each node
  - f(Sibiu)=c(Arad,Sibiu)+h(Sibiu)=140+253=393
  - f(Timisoara)=c(Arad,Timisoara)+h(Timisoara)=118+329=447
  - f(Zerind)=c(Arad,Zerind)+h(Zerind)=75+374=449
- Best choice is Sibiu

A* search example



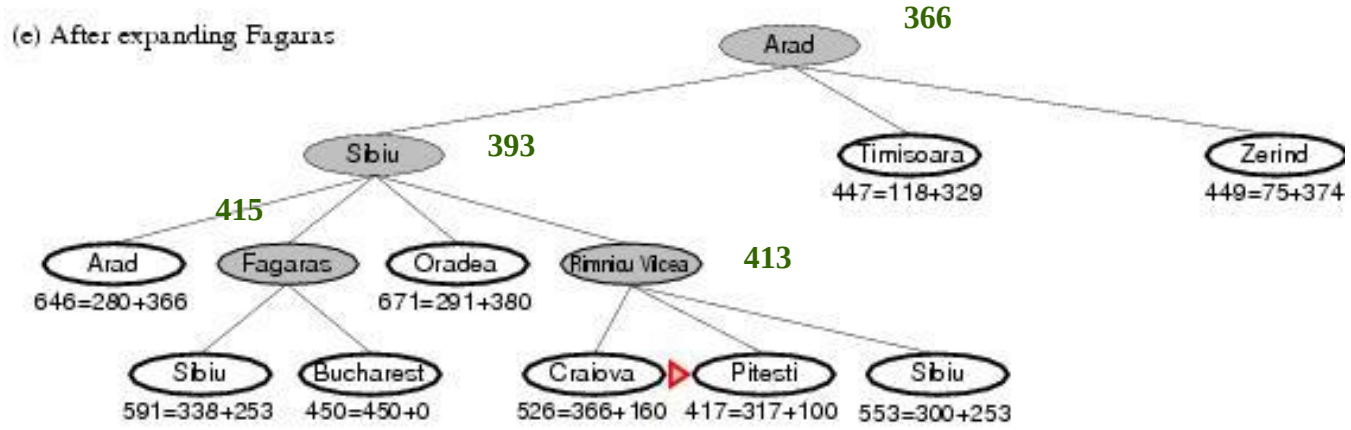
- Expand Sibiu and determine f(n) for each node
  - f(Arad)=c(Sibiu,Arad)+h(Arad)=280+366=646
  - f(Fagaras)=c(Sibiu,Fagaras)+h(Fagaras)=239+179=415
  - f(Oradea)=c(Sibiu,Oradea)+h(Oradea)=291+380=671
  - f(Rimnicu Vilcea)=c(Sibiu,Rimnicu Vilcea)+ h(Rimnicu Vilcea)=220+192=413
- Best choice is Rimnicu Vilcea

A* search example



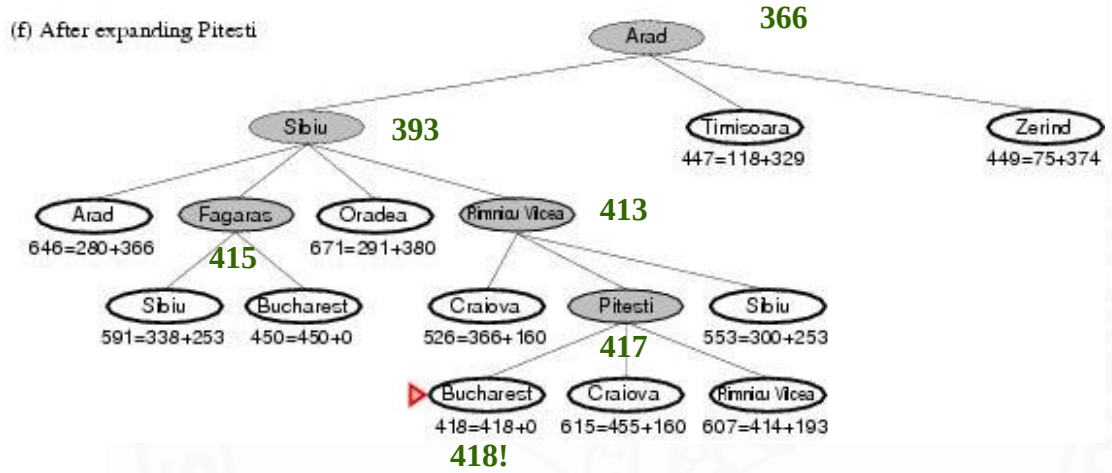
- Expand Rimnicu Vilcea and determine f(n) for each node
  - f(Craiova)=c(Rimnicu Vilcea, Craiova)+h(Craiova)=360+160=526
  - f(Pitesti)=c(Rimnicu Vilcea, Pitesti)+h(Pitesti)=317+100=417
  - f(Sibiu)=c(Rimnicu Vilcea,Sibiu)+h(Sibiu)=300+253=553
- Best choice is Fagaras

A* search example



- Expand Fagaras and determine f(n) for each node
  - f(Sibiu)=c(Fagaras, Sibiu)+h(Sibiu)=338+253=591
  - f(Bucharest)=c(Fagaras,Bucharest)+h(Bucharest)=450+0=450
- Best choice is Pitesti !!!

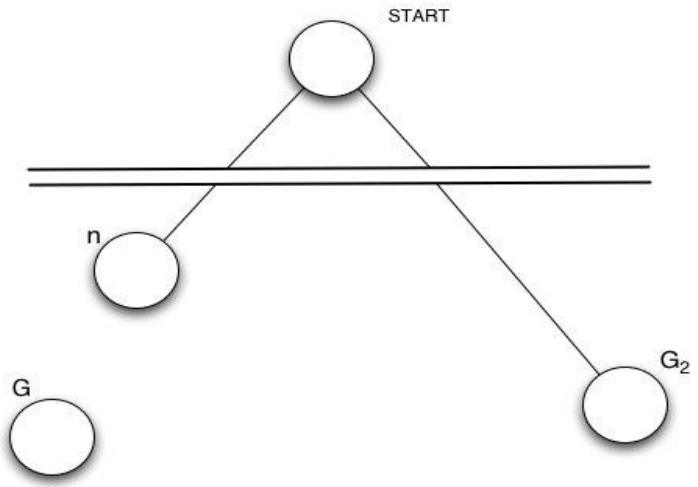
A* search example



- Expand Pitesti and determine f(n) for each node
  - f(Bucharest)=c(Pitesti,Bucharest)+h(Bucharest)=418+0=418

- Best choice is Bucharest !!!
  - Optimal solution (only if h(n) is admissable)
- Note values along optimal path !!

Optimality of A*(standard proof)



- Suppose suboptimal goal G_2 in the queue.
- Let n be an unexpanded node on a shortest path to optimal goal G.

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

$> g(G)$ since G_2 is suboptimal

$f(n) = g(n) + h(n) <= g(G)$ since h is admissible

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

BUT … graph search

- Discards new paths to repeated state (even when they are optimal).
  - Previous proof breaks down
- Solution:
  - Add extra bookkeeping i.e. remove more expensive of two paths.
  - Ensure that optimal path to any repeated state is always first followed.
    - Extra requirement on h(n): consistency (monotonicity)

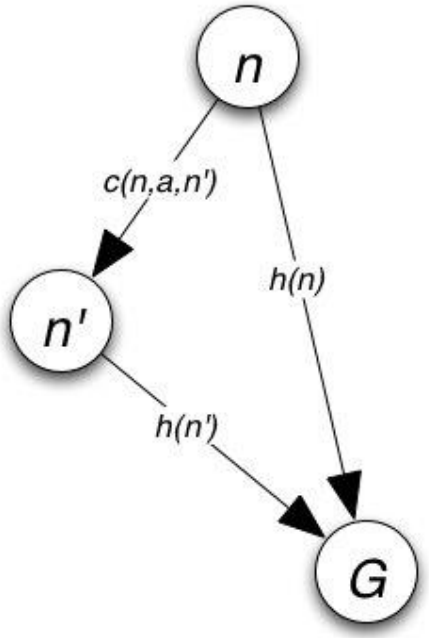
Consistency

- A heuristic is consistent if

$$h(n) \leq c(n, a, n') + h(n')$$

- If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &\geq f(n) \end{aligned}$$



i.e. f(n) is non-decreasing along any path.

Optimality of A*(more usefull)

- A* expands nodes in order of increasing f value
- Contours can be drawn in state space
  - Uniform-cost search adds circles.
  - F-contours are gradually

  Added:

  1) nodes with f(n)<C*

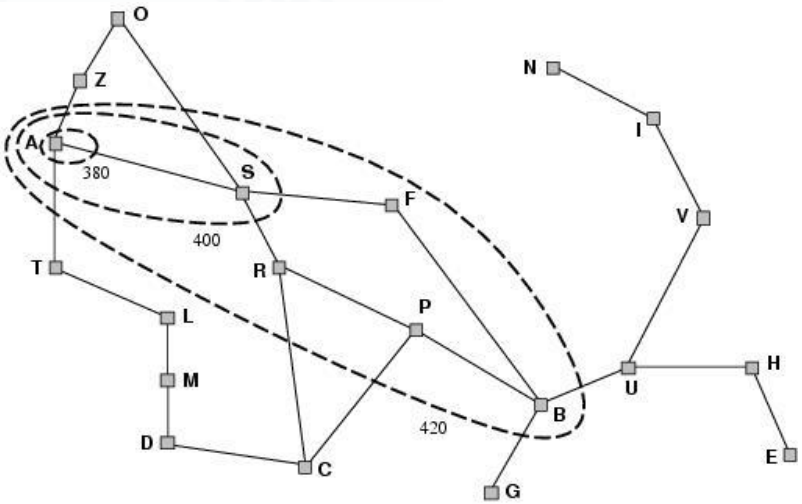
  2) Some nodes on the goal

  Contour (f(n)=C*).

  Contour I has all

  Nodes with $f=f_i$, where

  $f_i < f_i+1$.



A* search, evaluation

- Completeness: YES
  - Since bands of increasing f are added
  - Unless there are infinitly many nodes with f<f(G)

A* search, evaluation

- Completeness: YES
- Time complexity:
  - Number of nodes expanded is still exponential in the length of the solution.
  - It has been shows that unless the following mathematical relation holds for the heuristic function, the growth will be exponential

$$\left|h(n)-h^*(n)\right| \leq O\left(\log h^*(n)\right)$$

A* search, evaluation

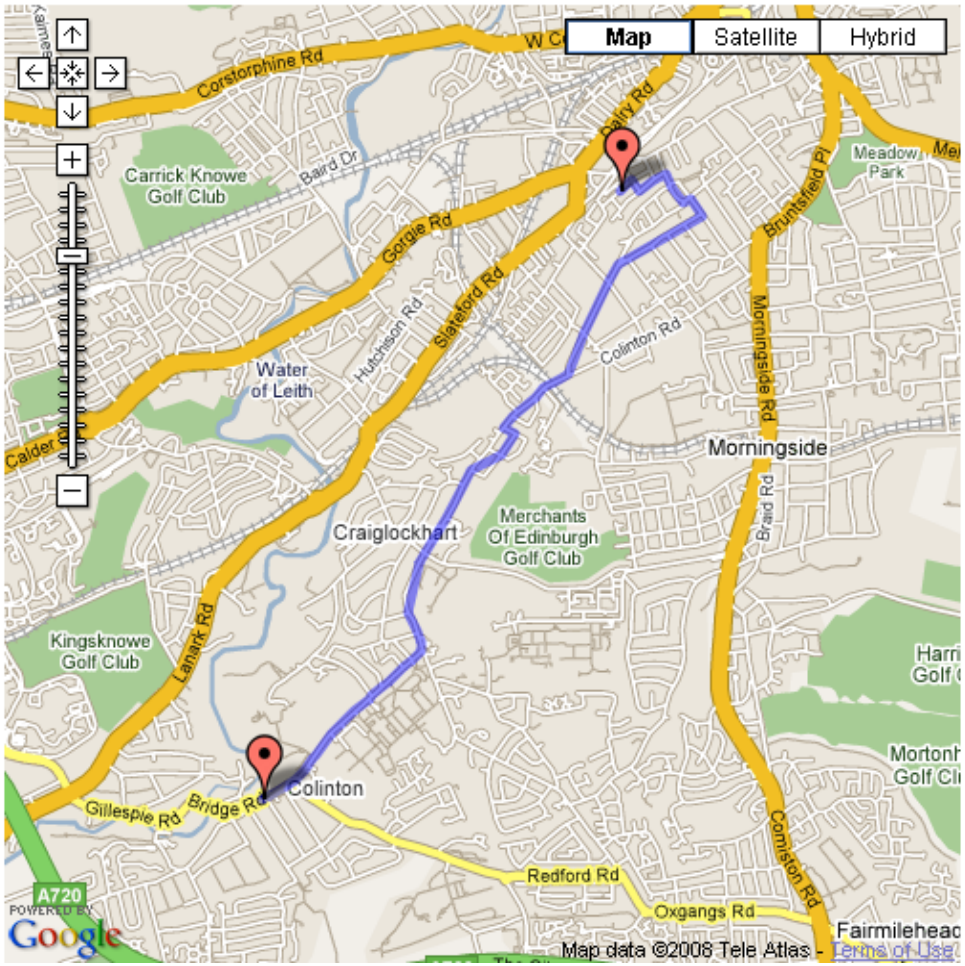
- Completeness: YES
- Time complexity: (exponential with path length)
- Space complexity:
  - It keeps all generated nodes in memory
  - Hence space is the major problem not time

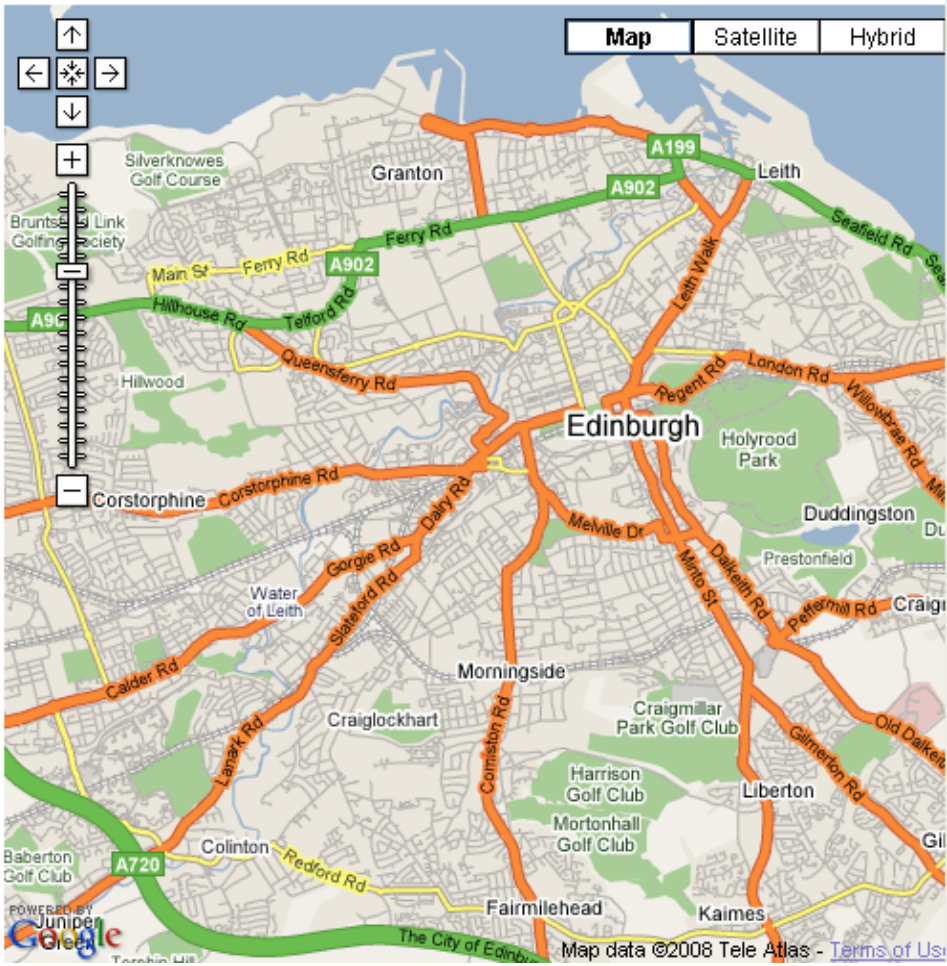
A* search, evaluation

- Completeness: YES
- Time complexity: (exponential with path length)
- Space complexity:(all nodes are stored)
- Optimality: YES
  - Cannot expand f_{i+1} until f_i is finished.
  - A* expands all nodes with f(n)< C*
  - A* expands some nodes with f(n)=C*
  - A* expands no nodes with f(n)>C*

Also optimally efficient (not including ties)

Example Application with A*





http://www.tarikattar.com/napier/osmastermap/

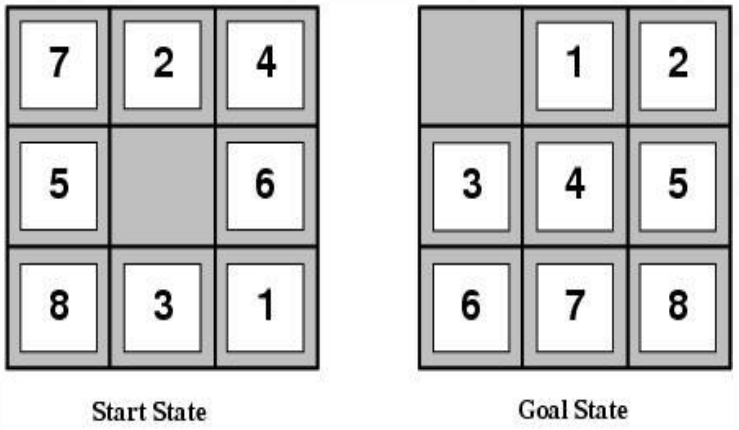
Memory-bounded heuristic search

- Some solutions to A* space problems (maintain completeness and optimality)
  - Iterative-deepening A* (IDA*)
    - Here cutoff information is the f-cost (g+h) instead of depth
  - Recursive best-first search(RBFS)
    - Recursive algorithm that attempts to mimic standard best-first search with linear space.
  - (Simple) Memory-bounded A* (SMA*)
    - Drop the worst-leaf node when memory is full

Learning to search better

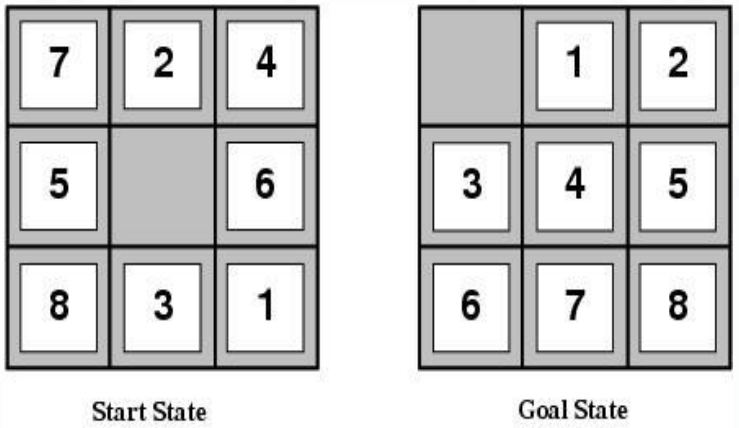
- All previous algorithms use *fixed strategies*.
- Agents can learn to improve their search by exploiting the *meta-level state space*.
  - Each meta-level state is a internal (computational) state of a program that is searching in *the object-level state space*.
  - In A* such a state consists of the current search tree
- A meta-level learning algorithm from experiences at the meta-level.

Heuristic functions



- E.g for the 8-puzzle
  - Avg. solution cost is about 22 steps (branching factor +/- 3)
  - Exhaustive search to depth 22: 3.1 x 10^{10} states.
  - A good heuristic function can reduce the search process.

Heuristic functions



- E.g for the 8-puzzle knows two commonly used heuristics
- h_1 = the number of misplaced tiles
  - $h_1(s)=8$
- h_2 = the sum of the distances of the tiles from their goal positions (manhattan distance).
  - $h_2(s)=3+1+2+2+2+3+3+2=18$

Heuristic quality

- Effective branching factor b*
  - Is the branching factor that a uniform tree of depth *d* would have in order to contain *N+1* nodes.

$$N+1=1+b^*+(b^*)^2+\ldots+(b^*)^d$$

  - Measure is fairly constant for sufficiently hard problems.
    - Can thus provide a good guide to the heuristic's overall usefulness.
    - A good value of b* is 1.

Heuristic quality and dominance

- 1200 random problems with solution lengths from 2 to 24.
- If $h_2(n) >= h_1(n)$ for all n (both admissible)
  then h_2 *dominates* h_1 and is better for search

Inventing admissible heuristics

- Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem:
  - Relaxed 8-puzzle for h_1 : a tile can move anywhere

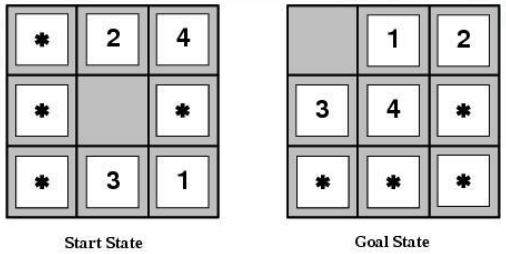
    As a result, $h_1(n)$ gives the shortest solution
  - Relaxed 8-puzzle for h_2 : a tile can move to any adjacent square.

    As a result, $h_2(n)$ gives the shortest solution.
- The optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.

  ABSolver found a usefull heuristic for the rubic cube.

Inventing admissible heuristics

- Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem.
- Given the subproblem, find the cost required to the solution
- This cost is a lower bound on the cost of the real problem.
- Pattern databases store the exact solution to for every possible subproblem instance.
  - The complete heuristic is constructed using the patterns in the DB
  - Disjoint pattern databases (example)
    - Find the number of moves involving tiles 1,2,3,4 for the 1-2-3-4 sub problem
    - Find the number of moves involving tiles 5,6,7,8 for the 5-6-7-8 sub problem
    - Add the two costs as this would be the lower limit on the actual solution



Inventing admissible heuristics

- Another way to find an admissible heuristic is through learning from experience:
  - Experience = solving lots of 8-puzzles
  - An inductive learning algorithm can be used to predict costs for other states that arise during search.
  - A set of features for a state are used in the prediction process
    - For example
      - Number of misplaced tiles
      - Number of pairs of adjacent tiles that are also adjacent in the goal state
  - First collect the statistics (features) about initial states before solving the problem to reach the goal state to give a actual solution cost
  - Use this data for training a predictor (estimator)

Inventing heuristics

- Combining heuristics
  - If a number of heuristics $h_1(n)$, $h_2(n)$,…, $h_m(n)$ are given such that none of them dominates any other then we can combine them to form a heuristic simply by taking the max of the heuristic values for a given node
    - $h(n) = \max \{h_1(n), h_2(n),\ldots, h_m(n) \}$

Examples of applied heuristic generation

- Using pattern databases it is possible to solve random 15-puzzles in a few milliseconds
- Number of nodes generated is reduced by a factor of 10,000 in comparison to using the manhattan distance heuristic
- For 24-puzzles, a speedup of roughly a million can be obtained

- Solve the 8 Queens Puzzle using Simulated Annealing
- Solve the 8 Puzzle using Heuristic Search through A* (Bonus Marks)
- Solve problems 4.3, 4.5-7 in the exercise
- Test Scheduled in next week
- Enjoy Reading the Book Chapter! ;)

End of Lecture

Informed decision-making comes from a long tradition of guessing and then blaming others for inadequate results.

Scott Adams