Informed Search

- Blind search - no notion concept of the "right direction"
  - can only recognize goal once it's achieved
- Heuristic search – we have rough idea of how good various states are, and use this knowledge to guide our search
- Can find solutions more efficient than uninformed
- General approach is *best-first-search*
- A node is selected based on an *evaluation function f(n)*
- A node that **seems** to be best is picked and it may not be the actual best

Best First Search

- The Idea:
  - use an ***evaluation function*** for each node... estimate of ``desirability''
  - Expand most desirable unexpanded node

Implementation

- Fringe: is a queue sorted in decreasing order of desirability

Special cases

Greedy

A*

Cost function *f(n)*

- A function f is maintained for each node

$f(n) = g(n) + h(n)$, n is the node in the open list

- "Node chosen" for expansion is the one with least f value
- $g(n)$ is the cost from root S to node n
- $h(n)$ is the estimated cost from node n to a goal
- For BFS: $f = 0$,
- For DFS: $f = 0$,
- For greedy g =0

Hill climbing

This is a ***greedy*** algorithm

Expands a node it sees closest to a goal

f(n) =h(n)

The algorithm

select a heuristic function;

set C, the current node, to the highest-valued initial node;

Loop until success or no more children(fail)

select N, the highest-value child of C;

return C if its value is better than the value of N;

Hill climbing

Complete:

No, Can get stuck in loop. Complete if loops are avoided.

Time complexity?

O(b^m), but with some good heuristic, it could give better results

Space complexity?

O(b^m), keeps all nodes in memory

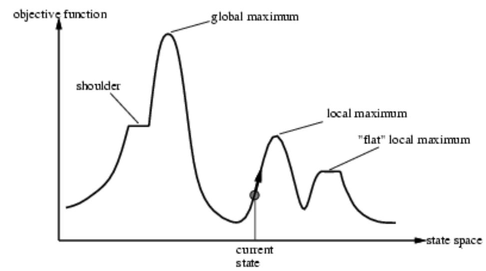
Optimality?

No

e.g. Arad→Sibiu→Rimnicu Virea→Pitesti→Bucharest is shorter!

Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima,…etc



Problems with hill climbing

1. Local maximum problem: there is a peak, but it is lower than the highest peak in the whole space.
2. The plateau problem: all local moves are equally unpromising, and all peaks seem far away.
3. The ridge problem: almost every move takes us down.

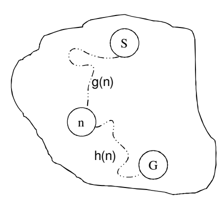
Solution:

Random-restart hill climbing is a series of hill-climbing searches with a randomly selected start node whenever the current search gets stuck.

Algorithm A*

- One of the most important advances in AI search algs.
- Idea: avoid expanding paths that are already expensive

  f(n) = g(n) + h(n)
- *g(n)* = least cost path to n from S found so far
- *h(n)* = estimated cost to goal from n
- *f(n)* = estimated total cost of path through n to goal



The A* procedure

Hill-climbing (and its improved versions) may miss an optimal solution. Here is a search method that ensures **optimality** of the solution.

The algorithm

keep a list of partial paths (initially root to root, length 0);
repeat
succeed if the first path P reaches the goal node;
otherwise remove path P from the list;
extend P in all possible ways, add new paths to the list;
sort the list by **the sum of two values:** the real cost of P till now, and an estimate of the remaining distance;
prune the list by leaving only the shortest path for each node reached so far;
until
success or the list of paths becomes empty;

The A* procedure

A heuristic that never overestimates is also called **optimistic** or **admissible**.

We consider three functions with values ≥ 0:

- g(n) is the actual cost of reaching node n,
- h*(n) is the actual ***unknown*** remaining cost,
- h(n) is the optimistic estimate of h(n).

Admissible heuristics

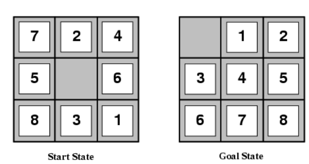
- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Theorem: If $h(n)$ is admissible, A* using is optimal

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)



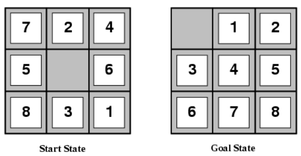
- $h_1(S)$ = ?
- $h_2(S)$ = ?

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

Admissible heuristics

- If $h_2(n) >= h_1(n)$ for all n, both are admissible
- Then h_2 dominates h_1 and is usually better for search

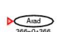
Typical Costs

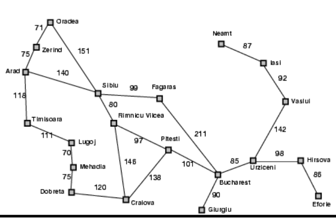
- d = 14 IDS = 3,473,941 nodes
  - A*(h_1) = 539 nodes
  - A*(h_2) = 113 nodes
- d = 24 IDS ~ 54,000,000,000 nodes
  - A(h_1) = 39,135 nodes
  - A(h_2) = 1,641 nodes

Remark: Given h_1 and h_2 any two admissible functions

then $h(n) = \max \{h_1(n), h_2(n)\}$ is also admissible

A* search example

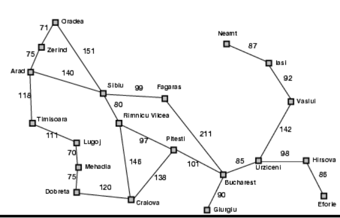




Straight–line distance to Bucharest

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

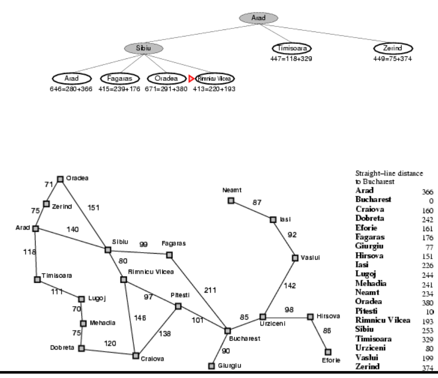
A* search example



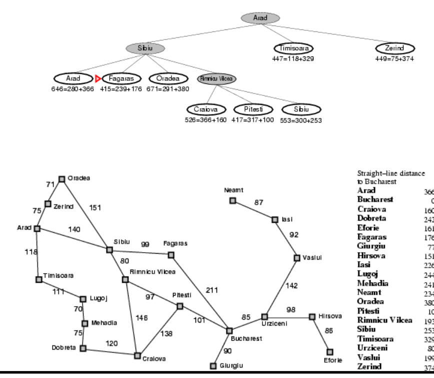
Straight–line distance to Bucharest

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

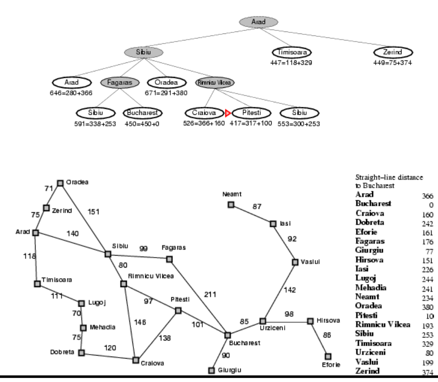
A* search example



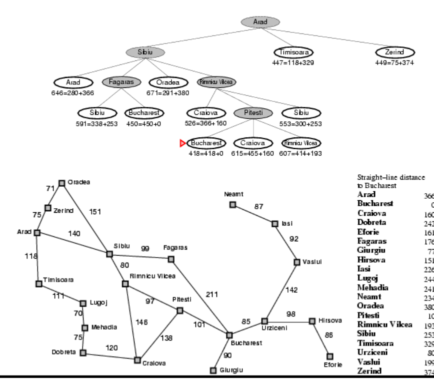
A* search example



A* search example



A* search example



Properties of A*

- Complete?

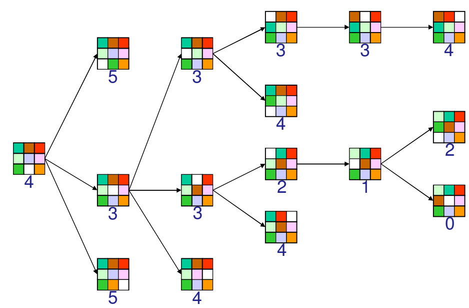
Yes (unless there are infinitely many)

- Time/Space?
- Exponential mostly
- Optimal?

Yes

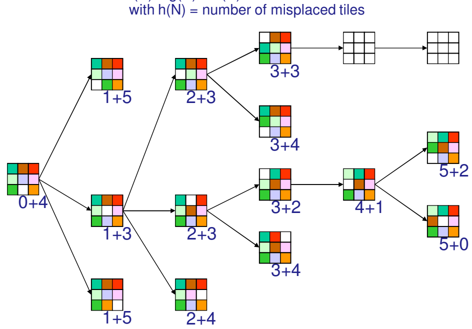
8-Puzzle

f(N) = h(N) = number of misplaced tiles



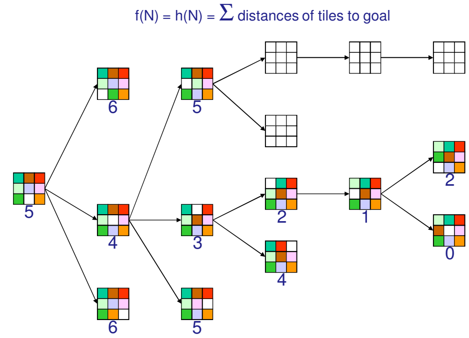
8-Puzzle

f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



8-Puzzle

f(N) = h(N) = $\sum$ distances of tiles to goal



Local Search Algorithms

- In many optimization problems, ***path*** is irrelevant
- the goal state itself is the solution
- Ex: The 8-queen problem, the final configuration of the queens is the important not the order they were put
- Operates using only single current state, rather than multiple paths.
- Find Optimal Configuration (satisfies the constraints)
- Use ***iterative improvement algorithms***
- A **Complete** local search algorithm finds a goal if exists
- An **Optimal** algorithm finds the global minimum or maximum

Local Search Algorithms

- Search algorithms like BFS, DFS or A* explore all the search space systematically by keeping one or more paths in memory and by recording which alternatives have been explored.
- Local search algorithms operate using a single current state (rather than multiple paths)
- move only to neighbors of that state.
- Ignore paths
- Advantages:
  - Use very little memory
  - Can often find reasonable solutions in large or infinite (continuous) state spaces.

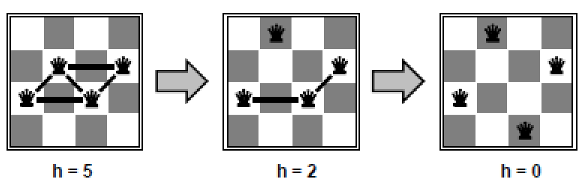
Local Search Algorithms

- Good for Optimization problems
  - find the best state according to some objective function
  - All states have an objective function
  - Goal is to find state with max (or min) objective value
  - Local search can do very well on these problems.

Local Search Algorithms

Example n-queens

- Put n queens on an nxn board with no two queens on the same row, column, or diagonal
- Local search: start with all n, move a queen to reduce conflicts



Local Search Algorithms

- Hill Climbing
- Simulated annealing
- Genetic algorithms
- Local search in continuous spaces

Local beam search

- One Solution to improve hill Climbing.
- Keep track of *k* states instead of one
  - Initially: *k* randomly selected states
  - Next: determine all successors of *k* states
  - If any of successors is goal → finished
  - Else select *k* best from successors and repeat.
- Major difference with random-restart search
  - Information is used for *k* search branches.
- Also improved to stochastic beam search

Simulated Annealing

- Annealing is a process for obtaining low energy states of a solid in a heat bath.
- The process contains two steps:
  - Increase the temperature of the heat bath to a maximum value at which the solid melts.
  - Decrease carefully the temperature of the heat bath until the particles arrange themselves in the ground state of the solid. Ground state is a minimum energy state of the solid.
- The ground state of the solid is obtained only if the maximum temperature is high enough and the cooling is done slowly.

Simulated Annealing

```
function SIMULATED-ANNEALING( problem, schedule) return a solution state
   input: problem, a problem
     schedule, a mapping from time to temperature
   local variables: current, a node.
        next, a node.
        T, a “temperature” controlling the probability of downward steps
     current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
     T ← schedule[t]
     if T = 0 then return current
     next ← a randomly selected successor of current
     ΔE ← VALUE[next] - VALUE[current]
     if ΔE > 0 then current ← next
     else current ← next only with probability e^(ΔE/T)
```

The cost of a solution is equivalent to the “energy” of a state.

Simulated Annealing

- The search is started with a randomized state. loop we will move to neighboring states always accepting the moves that decrease the energy while only accepting bad moves accordingly to a probability distribution dependent on the "temperature" of the system.
- Decrease the temperature slowly, accepting less bad moves at each temperature level until at very low temperatures the algorithm becomes a greedy hill-climbing algorithm.