# Informed Search

Dr. Richard J. Povinelli

# Objectives

- You should
  - be able to explain and contrast uniformed and informed searches.
  - be able to compare, contrast, classify, and implement various search algorithms including best-first, greedy, A*, RBFS, SMA*, hill-climbing, simulated annealing, and genetic algorithm searches.

# Review: General search

- Basic idea: offline, simulated exploration of state space by generating successors of already-explored states (a.k.a. expanding states)
- A strategy is defined by picking the order of node expansion

```
function Tree-Search( problem, fringe) returns a solution, or failure
    fringe ← Insert(Make-Node(Initial-State[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← Remove-Front( fringe )
        if Goal-Test[problem] applied to State(node) succeeds
            then return node
        fringe ← InsertAll (Expand(node, problem), fringe)
```



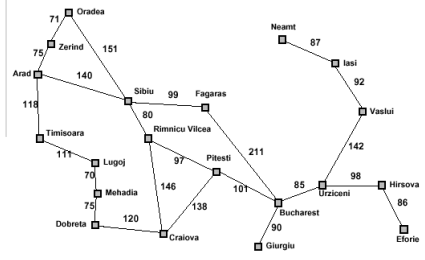
# Best-first search

- 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 search
    - A* search



# Romania with step costs in km




| Straight-line distance to Bucharest | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 178 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 98 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |



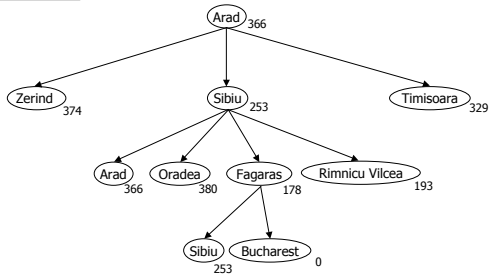
# Greedy search

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



# Greedy search example






# Properties of greedy search

- Complete??
- Time??
- Space??
- Optimal??



# Properties of greedy search

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

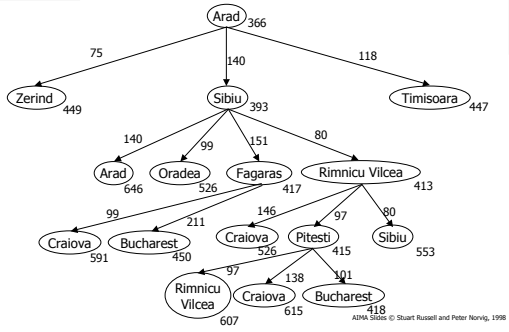


# A* search

- 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 to goal from $n$
  - $f(n)$ = estimated total cost of path through $n$ to goal
- A* search uses an *admissible* heuristic
  - $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from n.
  - e.g., $h_{SLD}(n)$ never overestimates the actual road distance
- Theorem: A* search is optimal



# A* search example



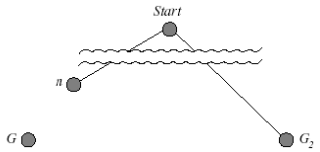



# CAT – A*

- With a partner discuss
  - Is A* better than greedy? Why?
  - What is the difference between A* and greedy?
  - Is the difference in search results on the map between A* and greedy search just a fluke?
- Work with a partner for 5 minutes.

# Optimality of A* (standard proof)

- Suppose some suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.



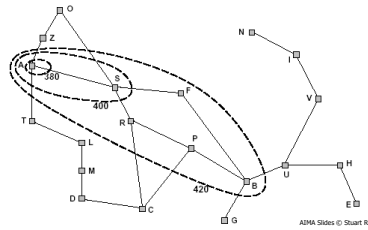

- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $> g(G_1)$ since $G_2$ is suboptimal
- $\geq f(n)$ since h is admissible
- Since $f(G_2) > f(n)$, **A* will never select $G_2$ for expansion**





# Optimality of A* (more useful)

- Lemma: A* expands nodes in order of increasing $f$ value
- Gradually adds "f -contours" of nodes (cf. breadth-first adds layers)
- Contour $i$ has all nodes with $f = f_i$, where $f_i < f_{i+1}$

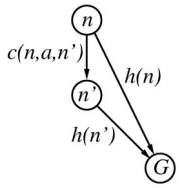







# Proof of lemma: Consistency

- A heuristic is consistent if
  $H(n) \leq c(n,a,n') + h(n')$
- If $h$ is consistent, we have
  $f(n') = g(n') + h(n')$
  $= g(n) + c(n,a,n') + h(n')$
  $\geq g(n) + h(n)$
  $= f(n)$
- i.e., $f(n)$ is nondecreasing along any path.








# Properties of A*

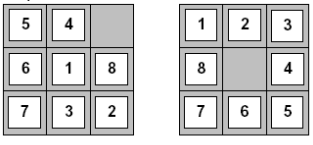
- Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$
- Time?? Exponential in [relative error in $h$ x length of solution.]
- Space?? Keeps all nodes in memory
- Optimal?? Yes — cannot expand $f_{i+1}$ until $f_i$ is finished





# Admissible heuristics

- What are good heuristics? How do we find good heuristics?
- 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



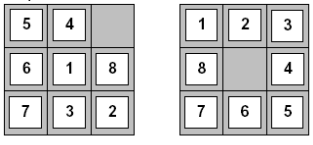

- $h_1(n)$ =??
- $h_2(n)$ =??





# Admissible heuristics

- What are good heuristics? How do find good heuristics?
- 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




- $h_1(n) = 7$
- $h_2(n) = 2+3+3+2+4+2+0+2 = 18$

## Dominance

- If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible) then $h_2$ dominates $h_1$ and is better for search
- Typical search costs:
  - $d$ = 14 IDS = 3,473,941 nodes
  - A*($h_1$) = 539 nodes
  - A*($h_2$) = 113 nodes
- d = 14 IDS = too many nodes
  - A*($h_1$) = 39,135 nodes
  - A*($h_2$) = 1,641 nodes



## Relaxed problems

- Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the 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
- Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem
- For TSP: let path be *any* structure that connects all cities
  - minimum spanning tree heuristic



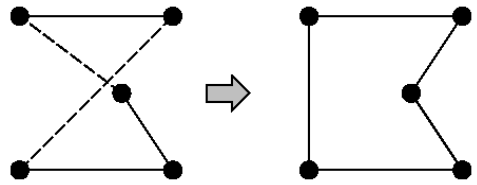
## Iterative improvement algorithms

- In many optimization problems, path is irrelevant; the goal state itself is the solution
- Then state space = set of "complete" configurations;
  - find *optimal* configuration, e.g., TSP
  - or, find configuration satisfying constraints, e.g., n-queens
- In such cases, can use *iterative improvement* algorithms; keep a single "current" state, try to improve it
- Constant space, suitable for online as well as offline search



## Example: Traveling Salesperson Problem

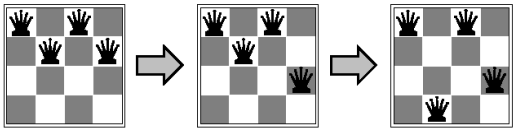
- Find the shortest tour that visits each city exactly once





## Example: n-queens

- Put n queens on an n x n board with no two queens on the same row, column, or diagonal





## CAT – Solve the 4-Queen Problem

- Draw a 4 by 4 grid on a piece of paper.
- Find a solution to the 4-Queen problem.
- Raise your hand when you are done.

# Hill-climbing I

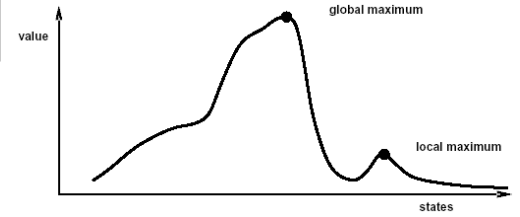
- Also called gradient ascent/descent
- "Like climbing Everest in thick fog with amnesia"

```
function Hill-Climbing (problem) returns a solution state
    inputs:          problem, a problem
    local variables: current, a node
                     next, a node
    current ← Make-Node(Initial-State[problem])
    loop do
        next ← a highest-valued successor of current
        if Value[next] < Value[current] then return current
        current ← next
    end
```



# Hill-climbing II

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






# Simulated annealing

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their size and frequency

```
function Simulated-Annealing( problem, schedule) returns a solution state
    inputs:          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)
```



# Properties of simulated annealing

- At fixed "temperature", $T$, state occupation probability reaches Boltzman distribution
- $p(x) = \alpha e^{E(x)/kT}$
- $T$ decreased slowly enough $\Rightarrow$ always reach best state
- Is this necessarily an interesting guarantee??
- Devised by Metropolis et al., 1953, for physical process modeling
- Widely used in VLSI layout, airline scheduling, etc.



# Evolutionary Algorithms

- Based on ideas taken from evolutionary biology.
  - populations
  - chromosomes
  - crossover
  - mutation
- Many varieties
  - Genetic algorithm
  - Evolution strategies
  - Genetic programming
  - Evolutionary programing



# Genetic Algorithm

```
function Genetic-Algorithm( problem, strategy)
        returns a solution, or failure
    initialize a population (set) of potential solutions to the
        problem encoded as strings
    while convergence criteria not met
        population ← Select (population)
        population ← Crossover (population)
        population ← Mutate (population)
    end
```

# Summary

- Heuristics can improve search performance dramatically
  - A* is optimal and complete, and the most efficient of all optimal search methods

