Informed Search

Introduction to informed search
The A* search algorithm
Designing good admissible heuristics

(AIMA Chapter 3.5.1, 3.5.2, 3.6)

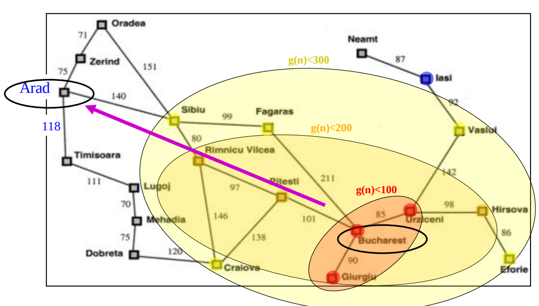
Outline – Informed Search

PART I - Today
- **Informed = use problem-specific knowledge**
- **Best-first search and its variants**
- **A* - Optimal Search using Knowledge**

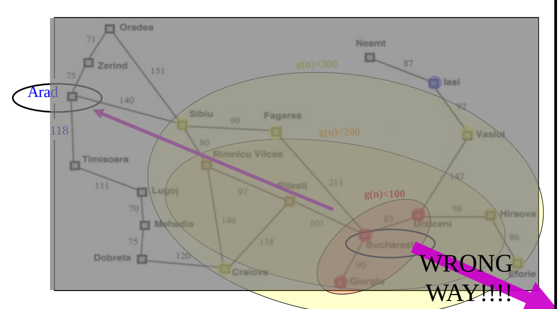
- **Proof of Optimality of A***
- **A* for maneuvering AI agents in games**
- **Heuristic functions?**
- **How to invent them**

PART II
- **Local search and optimization**
  - Hill climbing, local beam search, genetic algorithms,…
- **Local search in continuous spaces**
- **Online search agents**

Is Uniform Cost Search the best we can do?
Consider finding a route from Bucharest to Arad..



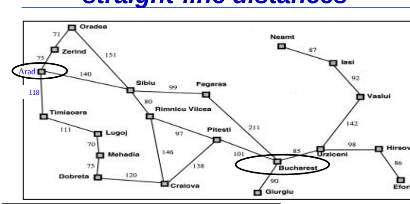
Is Uniform Cost Search the best we can do?
Consider finding a route from Bucharest to Arad..



A Better Idea…

- **Node expansion based on *an* *estimate* which *includes distance to the goal***
- **General approach of informed search:**
  - *Best-first search*: node selected for expansion based on an *evaluation function* ***f(n)***
    - —***f(n)*** includes *estimate* of distance to goal ***(new idea!)***
- **Implementation: Sort frontier queue by this new *f(n)*.**
  - Special cases: greedy search, ***A* search***

Simple, useful estimate *heuristic*: *straight-line distances*



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Heuristic (estimate) functions

Heureka! ---Archimedes

[dictionary]*"A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood."*

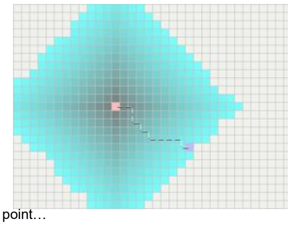
Heuristic knowledge **is useful, but not necessarily correct.**

Heuristic algorithms **use heuristic knowledge to solve a problem.**

A *heuristic function h(n)* takes a state *n* and returns an *estimate* of the distance from *n* to the goal.

(graphic: http://hyperbolegames.com/2014/10/20/eureka-moments/)

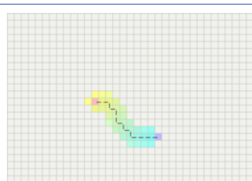
Breadth First for Games, Robots, …



- **Pink: Starting Point**
- **Blue: Goal**
- **Teal: Scanned squares**
  - Darker: Closer to starting point…

Graphics from
http://theory.stanford.edu/~amitp/GameProgramming/
(A great site for practical AI & game Programming

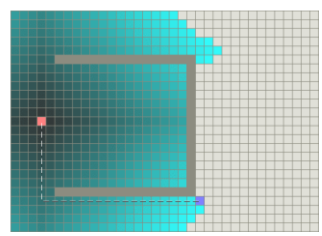
An optimal *informed search* algorithm (A*)



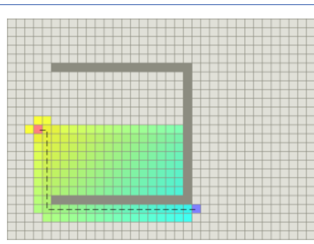
- **We add a *heuristic estimate* of distance to the goal**

- **Yellow: examined nodes with *high estimated* distance**
- **Blue: examined nodes with *low estimated* distance**

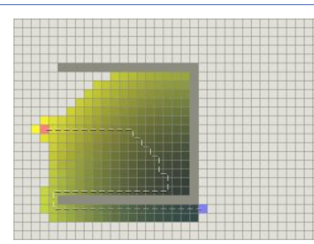
Breadth first in a world with obstacles



Informed search (A*) in a world with obstacles



Greedy best-first search in a world with obstacles



Review: Best-first search

Basic idea:

- ***select node for expansion*** **with minimal** ***evaluation function f(n)***
  - where $f(n)$ is some function that includes ***estimate heuristic*** $h(n)$ of the remaining distance to goal

- **Implement using priority queue**
- **Exactly UCS with $f(n)$ replacing $g(n)$**

Greedy best-first search: $f(n) = h(n)$

- **Expands the node that *is estimated* to be closest to goal**
- **Completely ignores $g(n)$: the cost to get to n**
- **Here, $h(n) = h_{SLD}(n)$ = straight-line distance from ` to Bucharest**

Greedy best-first search example

Frontier queue:
Arad 366

- **Initial State = Arad**
- **Goal State = Bucharest**

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

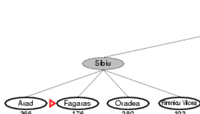
Greedy best-first search example

Frontier queue:
Sibiu 253
Timisoara 329
Zerind 374

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Greedy best-first search example

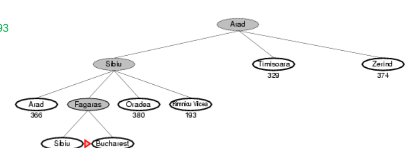
Frontier queue:
Fagaras 176
Rimnicu Vilcea 193
Timisoara 329
Arad 366
Zerind 374
Oradea 380



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Greedy best-first search example

Frontier queue:
Bucharest 0
Rimnicu Vilcea 193
Sibiu 253
Timisoara 329
Arad 366
Zerind 374
Oradea 380

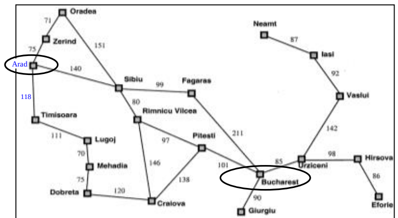


Goal reached !!

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

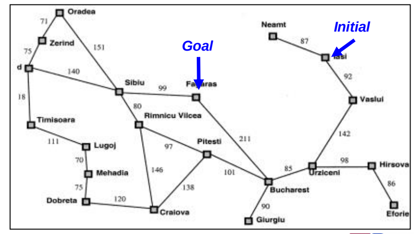
Properties of greedy best-first search

- ***Optimal?***
  - No!
    - —Found: *Arad → Sibiu → Fagaras → Bucharest (450km)*
    - —Shorter: *Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest (418km)*



Properties of greedy best-first search

- ***Complete?***
  - No – can get stuck in loops,
  - e.g., Iasi → Neamt → Iasi → Neamt →…



Properties of greedy best-first search

- ***Complete?*** **No – can get stuck in loops,**
  - e.g., Iasi → Neamt → Iasi → Neamt → …
- ***Time?*** ***O(b^m) – worst case*** **(like Depth First Search)**
  - But a good heuristic can give dramatic improvement of *average cost*
- ***Space?*** ***O(b^m)*** **– priority queue, so worst case: keeps all (unexpanded) nodes in memory**
- ***Optimal?*** **No**

A* search

- **Best-known form of best-first search.**
- **Key Idea: avoid expanding paths that are already expensive, but expand most promising first.**
- ***Simple idea*: *f(n)=g(n) + h(n)***
  - ***g(n)*** the cost (so far) to *reach* the node
  - ***h(n)*** estimated cost to *get from the node to the goal*
  - ***f(n)*** estimated *total cost* of path through *n* to goal
- **Implementation: Frontier queue as priority queue by increasing *f(n)* *(as expected…)***

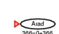
Admissible heuristics

- **A heuristic *h(n)* is *admissible* if it *never overestimates* the cost to reach the goal; i.e. it is *optimistic***
  - Formally: $\forall n$, *n* a node:
    1. $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from *n*
    2. $h(n) \geq 0$ so $h(G)=0$ for any goal *G*.
- ***Example:* $h_{SLD}(n)$ never overestimates the actual road distance**

Theorem: If *h(n)* is *admissible*, A* using Tree Search is *optimal*

A* search example

Frontier queue:
Arad 366



A* search example

Frontier queue:

Sibiu 393

Timisoara 447

Zerind 449

We add the three nodes we found to the Frontier queue.

We sort them according to the g()+h() calculation.

A* search example

Frontier queue:

Rimricu Vicea 413

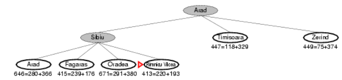
Fagaras 415

Timisoara 447

Zerind 449

Arad 646

Oradea 671



When we expand Sibiu, we run into Arad again. Note that we've already expanded this node once; but we still add it to the Frontier queue again.

A* search example

Frontier queue:

Fagaras 415

Pitesti 417

Timisoara 447

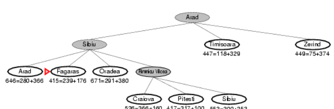
Zerind 449

Craiova 526

Sibiu 553

Arad 646

Oradea 671



We expand Rimricu Vicea.

A* search example

Frontier queue:

Pitesti 417

Timisoara 447

Zerind 449

Bucharest 450

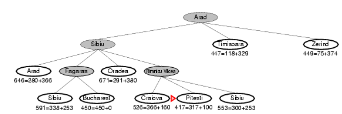
Craiova 526

Sibiu 553

Sibiu 591

Arad 646

Oradea 671



When we expand Fagaras, we find Bucharest, but we're not done. The algorithm doesn't end until we "expand" the goal node – it has to be at the top of the Frontier queue.

A* search example

Frontier queue:

Bucharest 418

Timisoara 447

Zerind 449

Bucharest 450

Craiova 526

Sibiu 553

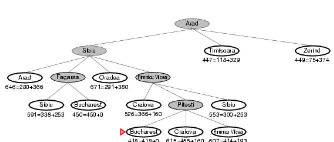
Sibiu 591

Rimricu Vicea 607

Craiova 615

Arad 646

Oradea 671



Note that we just found a better value for Bucharest!

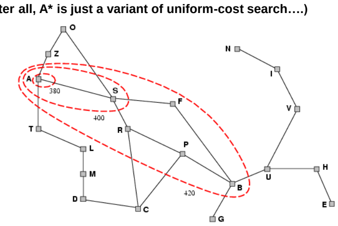
Now we expand this better value for Bucharest since it's at the top of the queue.

We're done and we know the value found is optimal!

Optimality of A* (intuitive)

- **Lemma: A* expands nodes on frontier in order of increasing *f* value**

- **Gradually adds "*f*-contours" of nodes**
- **Contour *i* has all nodes with $f=f_i$, where $f_i < f_{i+1}$**
- **(After all, A* is just a variant of uniform-cost search….)**



Optimality of A* using Tree-Search (proof idea)

- **Lemma: A* expands nodes on frontier in order of increasing f value**
- **Suppose some suboptimal goal G_2 *(i.e a goal on a suboptimal path)* has been generated and is in the frontier along with an optimal goal G.**

Must prove: $f(G_2) > f(G)$

(Why? Because if $f(G_2) > f(n)$, then G_2 will never get to the front of the priority queue.)

Proof:

1. **$g(G_2) > g(G)$ — since G_2 is suboptimal**
2. **$f(G_2) = g(G_2)$ — since $f(G_2)=g(G_2)+h(G_2)$ & $h(G_2) = 0$, since G_2 is a goal**
3. **$f(G) = g(G)$ — similarly**
4. **$f(G_2) > f(G)$ — from 1,2,3**

Also must show that G is added to the frontier before G_2 is expanded – see AIMA for argument in the case of Graph Search

A* search, evaluation

- **Completeness: YES**
  - Since bands of increasing *f* are added
  - As long as *b* is finite
    - —(guaranteeing that there aren't infinitely many nodes n with $f(n) < f(G)$)

A* search, evaluation

- **Completeness: YES**
- **Time complexity:**
  - Number of nodes expanded is still exponential in the length of the solution.

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

Proof of Lemma: Consistency

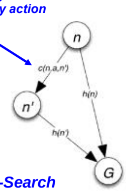
- **A heuristic is *consistent* if**

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

- **Lemma: If h is consistent,**

$$f(n') = g(n') + h(n') = g(n) + c(n,a,n') + h(n') \ge g(n) + h(n) = f(n)$$

Cost of getting from n to n' by any action a



i.e. f(n) is *nondecreasing* along any path.

Theorem: if h(n) is consistent, *A* using Graph-Search is optimal*

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 $f(n) < f(G)$
  - A* expands one node with $f(n)=f(G)$
  - A* expands no nodes with $f(n)>f(G)$

Also *optimally efficient* (not including ties)

Creating Good Heuristic Functions

AIMA 3.6

Heuristic functions

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- 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

Admissible heuristics

E.g., for the 8-puzzle:

- *$h_{oop}(n)$* = number of out of place tiles
- *$h_{md}(n)$* = total Manhattan distance (i.e., # of moves 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

- $h_{oop}(S)$ = ?
- $h_{md}(S)$ = ?

Admissible heuristics

E.g., for the 8-puzzle:

- *$h_{oop}(n)$* = number of out of place tiles
- *$h_{md}(n)$* = total Manhattan distance (i.e., # of moves 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

- $h_{oop}(S)$ = ? 8
- $h_{md}(S)$ = ? 3+1+2+2+2+3+3+2 = 18

Relaxed problems

- A problem with fewer restrictions on the actions than the original 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 *$h_{oop}(n)$* gives the shortest solution
- If the rules are relaxed so that a tile can move to ***any adjacent square,*** then *$h_{md}(n)$* gives the shortest solution

Defining Heuristics: *h(n)*

- Cost of an exact solution to a ***relaxed*** problem (fewer restrictions on operator)
- Constraints on ***Full*** Problem:

  A tile can move from square A to square B *if* A is adjacent to B *and* B is blank.
  - Constraints on *relaxed* problems:
    - —A tile can move from square A to square B *if* A is adjacent to B. (h_{md})
    - —A tile can move from square A to square B *if* B is blank.
    - —A tile can move from square A to square B. (h_{oop})

Dominance

- **If $h_2(n) \geq h_1(n)$ for all n (both admissible)**
  - then h_2 *dominates* h_1
- **So h_2 is optimistic, but more accurate than h_1**
  - h_2 is therefore better for search
  - Notice: h_{md} dominates h_{oop}
- **Typical search costs (average number of nodes expanded):**
  - *d=12* Iterative Deepening Search = 3,644,035 nodes
    $A^*(h_{oop})$ = 227 nodes
    $A^*(h_{md})$ = 73 nodes
  - *d=24* IDS = too many nodes
    $A^*(h_{oop})$ = 39,135 nodes
    $A^*(h_{md})$ = 1,641 nodes

Iterative Deepening *A** and beyond

Beyond our scope:

- **Iterative Deepening A***
- **Recursive best first search (incorporates A* idea, despite name)**
- **Memory Bounded A***
- **Simplified Memory Bounded A* - R&N say the best algorithm to use in practice, but not described here at all.**
  - (If interested, follow reference to Russell article on Wikipedia article for SMA*)

(see 3.5.3 if you're interested in these topics)