Title: Informed Search Methods

Required reading: AIMA, Chapter 3 (Sections 3.5, 3.6)

LWH: Chapters 6, 10, 13 and 14.

Introduction to Artificial Intelligence

CSCE 476-876, Fall 2019

URL: `www.cse.unl.edu/~choueiry/F19-476-876`

Berthe Y. Choueiry (Shu-we-ri)

(402)472-5444

Outline

- Categorization of search techniques

- Ordered search (search with an evaluation function)

- Best-first search:
  (1) Greedy search   (2) A*

- Admissible heuristic functions:
  how to compare them?
  how to generate them?
  how to combine them?

Types of Search (I)

1- Uninformed vs. informed

2- Systematic/constructive vs. iterative improvement

Uninformed :

    use only information available in problem definition,

    no idea about distance to goal

    $\rightarrow$ can be incredibly ineffective in practice

Heuristic :

    exploits some knowledge of the domain

    also useful for solving optimization problems

Types of Search (II)

Systematic, exhaustive, constructive search:
   a <u>partial solution</u> is incrementally extended into <u>global solution</u>

Partial solution =
   sequence of transitions between states

Global solution =
   Solution from the initial state to the goal state

Examples: $\begin{cases} \text{Uninformed} \\ \\ \text{Informed (heuristic): Greedy search, A}^* \end{cases}$

$\rightarrow$ Returns the path; solution = path

Types of Search (III)

Iterative improvement:

A state is gradually modified and evaluated until
reaching an (acceptable) optimum

$\rightarrow$ We don't care about the path, we care about 'quality' of state

$\rightarrow$ Returns a state; a solution = good quality state

$\rightarrow$ Necessarily an informed search

Examples (informed):
$\begin{cases}
\text{Hill climbing} \\
\text{Simulated Annealing (physics), Taboo search} \\
\text{Genetic algorithms (biology)}
\end{cases}$

Ordered search

• Strategies for systematic search are generated by choosing which node from the fringe to expand first

• The node to expand is chosen by an **evaluation function**, expressing 'desirability' $\longrightarrow$ **ordered search**

• When nodes in queue are sorted according to their decreasing values by the evaluation function $\longrightarrow$ **best-first search**

• Warning: 'best' is actually 'seemingly-best' given the evaluation function. Not always best (otherwise, we could march directly to the goal!)

Search using an evaluation function

• Example: uniform-cost search!

   What is the evaluation function?

   Evaluates cost from .... to ....?

• How about the cost **to** the goal?

$$h(n) = \underline{\text{estimated}} \text{ cost of the cheapest}$$
path from the state at node n to a goal state

   $h(n)$ would help focusing search

Cost to the goal

This information is <u>not</u> part of the problem description

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

Best-first search

1. <u>Greedy search</u> chooses the node n closest to the goal such as $h(n)$ is minimal

2. <u>A* search</u> chooses the least-cost solution

   solution cost $f(n)$ $\begin{cases} g(n)\text{: cost from root to a given node } n \\ + \\ h(n)\text{: cost from the node } n \text{ to the goal node} \end{cases}$

   such as $f(n) = g(n) + h(n)$ is minimal

Greedy search

$\rightarrow$ First expand the node whose state is 'closest' to the goal!

$\rightarrow$ Minimize $h(n)$

> **function** BEST-FIRST-SEARCH(*problem*, EVAL-FN) **returns** a solution sequence
>   **inputs**: *problem*, a problem
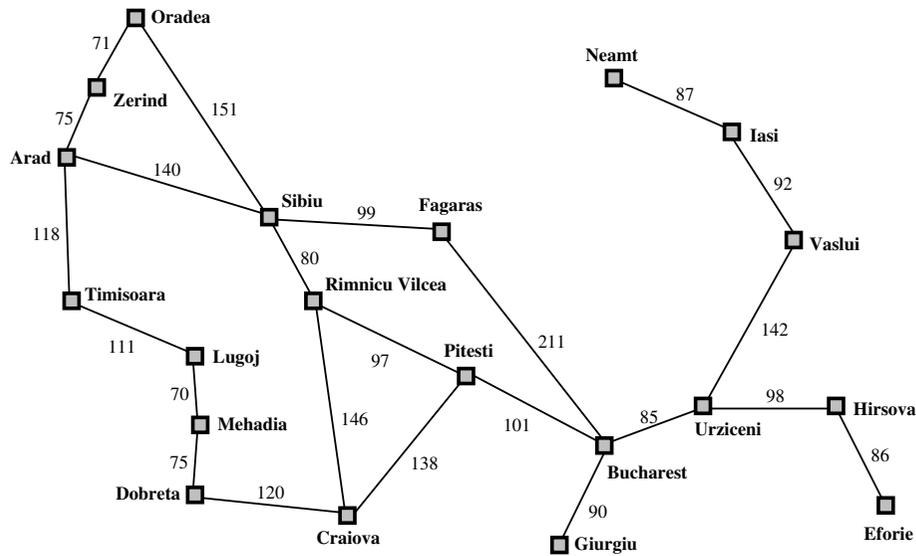>   *Eval-Fn*, an evaluation function
>
>   *Queueing-Fn* $\leftarrow$ a function that orders nodes by EVAL-FN
>   **return** GENERAL-SEARCH(*problem*, *Queueing-Fn*)

$\rightarrow$ Usually, cost of reaching a goal may be <u>estimated</u>,
    not determined exactly

$\rightarrow$ If state at n is goal, $h(n)=$?

$\rightarrow$ How to choose $h(n)$?   Problem specific! Heuristic!

Greedy search: Romania

$$h_{\text{SLD}}(n) = \text{straight-line distance between } n \text{ and goal location}$$

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 search: Trip from Arad to Bucharest

(a) The initial state

Arad
366

(b) After expanding Arad

Arad

Sibiu 253   Timisoara 329   Zerind 374

(c) After expanding Sibiu

Arad

Sibiu   Timisoara 329   Zerind 374

Arad 366   Fagaras 176   Oradea 380   Rimnicu Vilcea 193

(d) After expanding Fagaras

Arad

Sibiu   Timisoara 329   Zerind 374

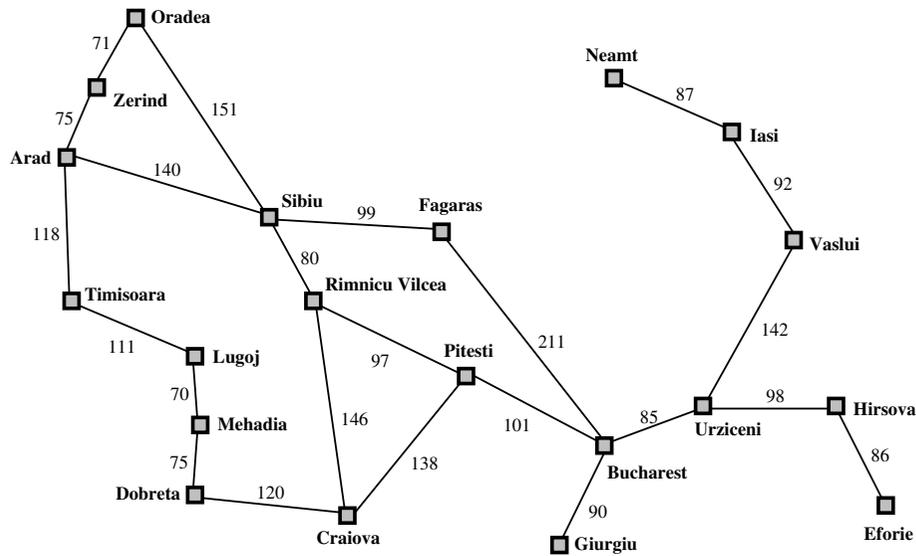Arad 366   Fagaras   Oradea 380   Rimnicu Vilcea 193

Sibiu 253   Bucharest 0
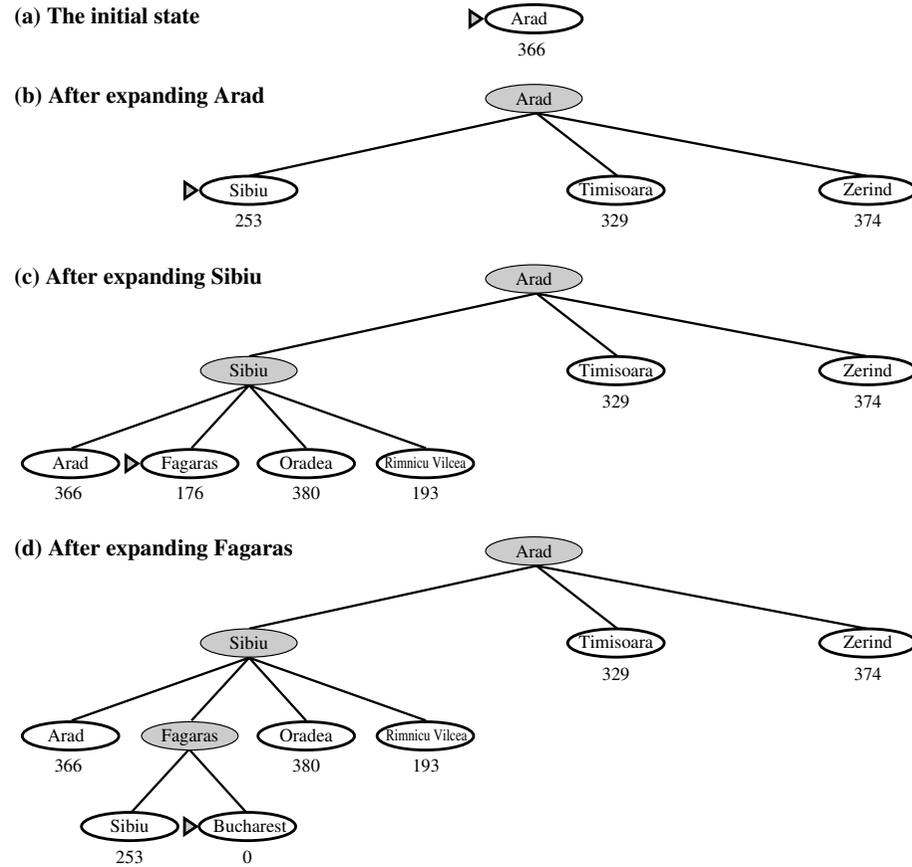
... Greedy search!   quick, but not optimal!

Greedy search: Problems

From Iasi to Fagaras?
$\begin{cases} \text{False starts: Neamt is a dead-end} \\ \text{Looping} \end{cases}$

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 search: Properties

$\rightarrow$ Like depth-first, tends to follow a single path to the goal

$\rightarrow$ Like depth-first $\begin{cases} \text{Not complete} \\ \text{Not optimal} \end{cases}$

$\rightarrow$ Time complexity: $O(b^m)$, m maximum depth

$\rightarrow$ Space complexity: $O(b^m)$ retains all nodes in memory

$\rightarrow$ Good h function (considerably) reduces space and time
but h functions are problem dependent :—(

Hmm...

Greedy search minimizes estimated cost to goal $h(n)$

$\rightarrow$ cuts <u>search cost</u> considerably

$\rightarrow$ but not optimal, not complete

Uniform-cost search minimizes cost of the path so far $g(n)$

$\rightarrow$ is optimal and complete

$\rightarrow$ but can be wasteful of resources

New-Best-First search minimizes $f(n) = g(n) + h(n)$

$\rightarrow$ combines greedy and uniform-cost searches

$f(n) =$ estimated cost of cheapest solution via n

$\rightarrow$ Provably: complete and optimal, <u>if $h(n)$ is admissible</u>

A^* Search

- **A^* search**
  Best-first search expanding the node in the fringe with minimal
  $f(n) = g(n) + h(n)$

- **A^* search with admissible $h(n)$**
  Provably complete, optimal, and optimally efficient using
  TREE-SEARCH

- **A^* search with consistent $h(n)$**
  Remains optimal even using GRAPH-SEARCH

(See TREE-SEARCH versus GRAPH-SEARCH page 77)

Admissible heuristic

An admissible heuristic is a heuristic that <u>never overestimates</u> the cost to reach the goal

    $\rightarrow$ is optimistic

    $\rightarrow$ thinks the cost of solving is less than it actually is
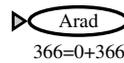
Example:
$$\begin{cases} \text{travel: straight line distance} \\ \text{I need 3 years to finish college (at least!)} \\ \text{We are 3 years away from the first flight to Mars (at least!)} \end{cases}$$

If h is admissible,
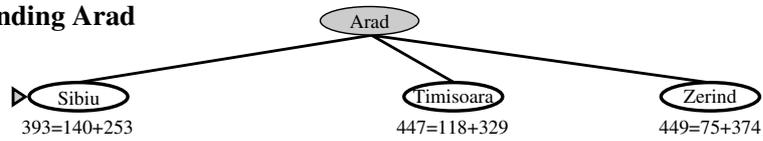
      $f(n)$ never overestimates the actual cost of the best solution through n.
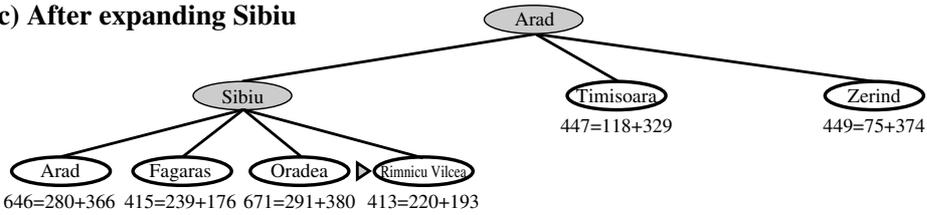
A* Search From Arad to Bucharest
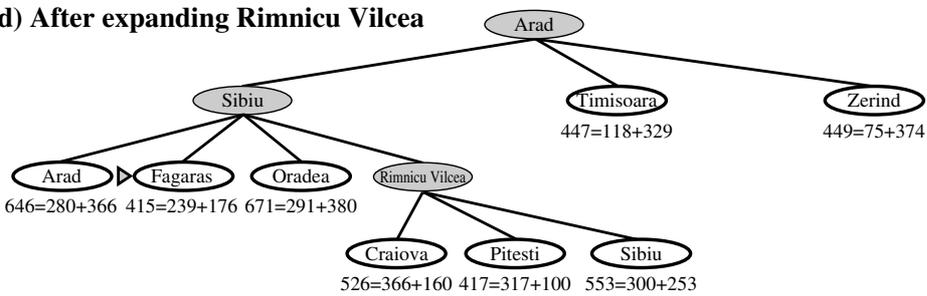
(a) The initial state

Arad
366=0+366

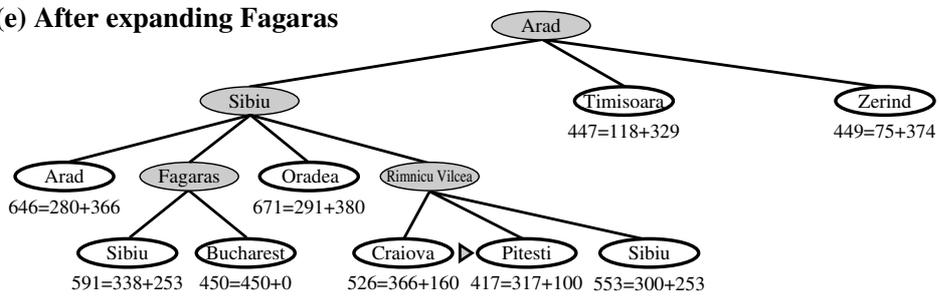**(b) After expanding Arad**

Arad

- Sibiu — 393=140+253
- Timisoara — 447=118+329
- Zerind — 449=75+374

(c) After expanding Sibiu

Arad

- Sibiu
  - Arad — 646=280+366
  - Fagaras — 415=239+176
  - Oradea — 671=291+380
  - Rimnicu Vilcea — 413=220+193
- Timisoara — 447=118+329
- Zerind — 449=75+374

(d) After expanding Rimnicu Vilcea

Arad

- Sibiu
  - Arad — 646=280+366
  - Fagaras — 415=239+176
  - Oradea — 671=291+380
  - Rimnicu Vilcea
    - Craiova — 526=366+160
    - Pitesti — 417=317+100
    - Sibiu — 553=300+253
- Timisoara — 447=118+329
- Zerind — 449=75+374

(e) After expanding Fagaras

Arad

- Sibiu
  - Arad — 646=280+366
  - Fagaras
    - Sibiu — 591=338+253
    - Bucharest — 450=450+0
  - Oradea — 671=291+380
  - Rimnicu Vilcea
    - Craiova — 526=366+160
    - Pitesti — 417=317+100
    - Sibiu — 553=300+253
- Timisoara — 447=118+329
- Zerind — 449=75+374

(f) After expanding Pitesti

Arad

- Sibiu
  - Arad — 646=280+366
  - Fagaras
    - Sibiu — 591=338+253
    - Bucharest — 450=450+0
  - Oradea — 671=291+380
  - Rimnicu Vilcea
    - Craiova — 526=366+160
    - Pitesti
      - Bucharest — 418=418+0
      - Craiova — 615=455+160
      - Rimnicu Vilcea — 607=414+193
    - Sibiu — 553=300+253
- Timisoara — 447=118+329
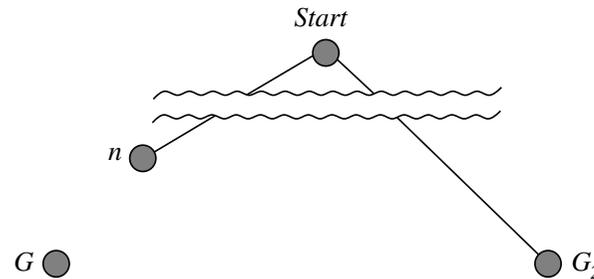- Zerind — 449=75+374
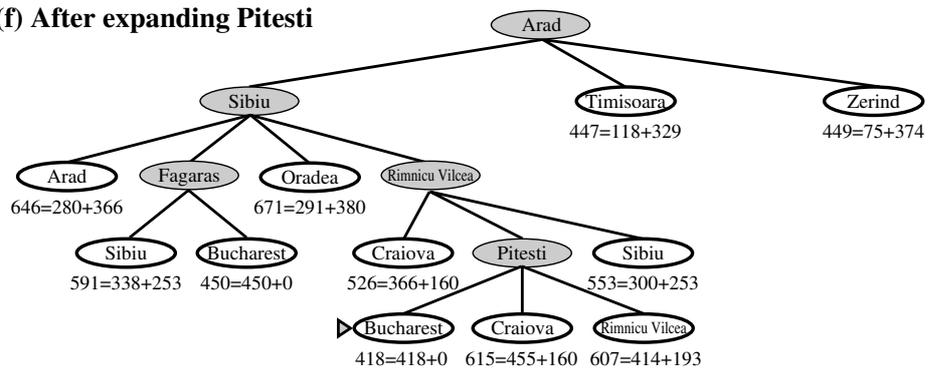
A^* Search is optimal

G, G_2 goal states $\Rightarrow g(G) = f(G)$, $f(G_2) = g(G_2)$ $\qquad$ $h(G) = h(G_2) = 0$

G optimal goal state $\Rightarrow C^* = f(G)$

G_2 suboptimal $\Rightarrow f(G_2) > C^* = f(G)$ $\hfill$ (1)

Suppose n is not chosen for expansion

h admissible $\Rightarrow C^* \geq f(n)$ $\hfill$ (2)

Since n was not chosen for expansion $\Rightarrow f(n) \geq f(G_2)$ $\hfill$ (3)

$(2) + (3) \Rightarrow C^* \geq f(G_2)$ $\hfill$ (4)

(1) and (4) are contradictory $\Rightarrow n$ should be chosen for expansion

Which nodes does A* expand?

GOAL-TEST is applied to STATE(node) when a node is <u>chosen from the fringe</u> for expansion, <u>not</u> when the node is <u>generated</u>

<div align="right">Theorem 3 & 4 in Pearl 84, original results by Nilsson</div>

- *Necessary condition:* Any node expanded by A* cannot have an f value exceeding C^*: For all nodes expanded, $f(n) \leq C^*$

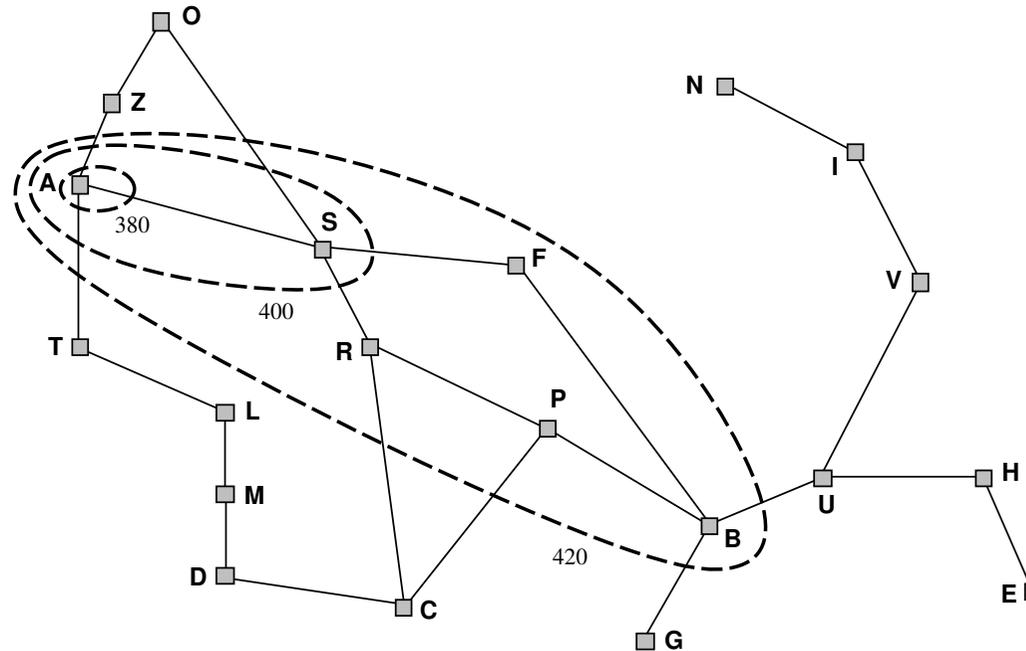- *Sufficient condition:* Every node in the fringe for $f(n) < C^*$ will eventually be expanded by A*

In summary

- A* expands all nodes with $f(n) < C^*$

- A* expands some nodes with $f(n) = C^*$

- A* expands no nodes with $f(n) > C^*$

Expanding contours

A* expands nodes from fringe in increasing f value

We can conceptually draw contours in the search space

The <u>first</u> solution found is necessarily the optimal solution

Careful: a TEST-GOAL is applied at node expansion

A^* **Search** is complete

Since A* search expands all nodes with $f(n) < C^*$, it must eventually reach the goal state unless there are infinitely many

nodes $f(n) < C^* \begin{cases} \text{1. } \exists \text{ a node with infinite branching factor} \\ \text{or} \\ \text{2. } \exists \text{ a path with infinite number of nodes along it} \end{cases}$

A^* is complete if $\begin{cases} \text{on locally finite graphs} \\ \text{and} \\ \exists \delta > 0 \text{ constant, the cost of each operator } > \delta \end{cases}$

A^* **Search** Complexity

Time:

Exponential in (relative error in $h \times$ length of solution path) ... quite bad

Space: must keep all nodes in memory

Number of nodes within goal contour is exponential in length of solution.... unless the error in the heuristic function $|h(n) - h^*(n)|$ grows no faster than the log of the actual path cost: $|h(n) - h^*(n)| \leq O(\log h^*(n))$

In practice, the error is proportional... impractical..

major drawback of A*: runs out of space quickly

$\rightarrow$ Memory Bounded Search IDA*(not addressed here)

A^* Search is optimally efficient

.. for any given evaluation function: no other algorithms that finds the optimal solution is guaranteed to expend fewer nodes than A^*

<u>Interpretation</u> (proof not presented): Any algorithm that does not expand all nodes between root and the goal contour risks missing the optimal solution

Tree-Search vs. Graph-Search

After choosing a node from the fringe and before expanding it, GRAPH-SEARCH checks whether STATE(node) was visited before to avoid loops.

$\rightarrow$ GRAPH-SEARCH may lose optimal solution

Solutions

1. In Graph-Search, discard the more expensive path to a node

2. Ensure that the optimal path to any repeated state is the first one found
   $\rightarrow$ Consistency

Consistency

$h(n)$ is consistent

If $\forall\, n$ and $\forall\, n'$ successor of n along a path, we have

$h(n) \leq k(n, n') + h(n')$, k cost of cheapest path from n to n'

Monotonicity

$h(n)$ is monotone

If $\forall\, n$ and $\forall\, n'$ successor of n generated by action a, we have

$h(n) \leq c(n, a, n') + h(n')$, n' is an <u>immediate</u> successor of n

Triangle inequality ($\langle n, n', \text{goal} \rangle$)

Important: h is consistent $\Leftrightarrow$ h is monotone

Beware: of confusing terminology 'consistent' and 'monotone'

Values of h not necessarily decreasing/nonincreasing

Properties of h: Important results

- h consistent $\Leftrightarrow h$ monotone   (Pearl 84)

- h consistent $\Rightarrow h$ admissible   (AIMA, Exercise 4.7)
  consistency is stricter than admissibility

- h consistent $\Rightarrow f$ is nondecreasing
  $$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

- h consistent $\Rightarrow \text{A}^*$ using GRAPH-SEARCH is optimally efficient
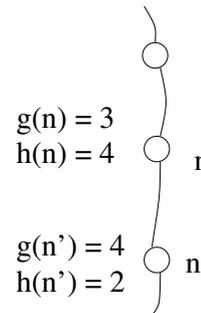
Pathmax equation

You may ignore this slide

Monotonicity of f: values along a path are nondecreasing

When f is not monotonic, use **pathmax** equation

$$f(n') = max(f(n), g(n') + h(n'))$$

A* never decreases along any path out from root

g(n) = 3
h(n) = 4   n

g(n') = 4   n'
h(n') = 2

Pathmax

● guarantees f nondecreasing

● does not guarantee h consistent

● does not guarantee A* + GRAPH-SEARCH is optimally efficient

Summarizing definitions for $\mathbf{A}^*$

- A^* is a best-first search that expands the node in the fringe with minimal $f(n) = g(n) + h(n)$

- An admissible function h never overestimates the distance to the goal.

- h admissible $\Rightarrow \mathrm{A}^*$ is complete, optimal, optimally efficient using TREE-SEARCH

- h consistent $\Leftrightarrow h$ monotone
  h consistent $\Rightarrow h$ admissible
  h consistent $\Rightarrow f$ nondecreasing

- h consistent $\Rightarrow \mathrm{A}^*$ remains optimal using GRAPH-SEARCH

Admissible heuristic functions

Examples

- Route-finding problems: straight-line distance

- 8-puzzle: $\begin{cases} h_1(n) = \text{number of misplaced tiles} \\ h_2(n) = \text{total Manhattan distance} \end{cases}$

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

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

Performance of admissible heuristic functions

Two criteria to compare <u>admissible</u> heuristic functions:

1. Effective branching factor: b^*

2. Dominance: number of nodes expanded

Effective branching factor b^*

– The heuristic expands N nodes in total

– The solution depth is d

$\longrightarrow b^*$ is the branching factor had the tree been uniform

$$N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d = \frac{(b^*)^{d+1} - 1}{b^* - 1}$$

– Example: N=52, d=5 $\rightarrow b^* = 1.92$

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: nodes expanded

Sol. depth	IDS	$\mathbf{A}^*(h_1)$	$\mathbf{A}^*(h_2)$
$d = 12$	3,644,035	227	73
$d = 24$	too many	39,135	1,641

A* expands all nodes $f(n) < C^* \Rightarrow g(n) + h(n) < C^*$

$\Rightarrow h(n) < C^* - g(n)$

If $h_1 \leq h_2$, A* with h_1 will always expand at least as many (if not more) nodes than A* with h_2

$\longrightarrow$ It is always better to use a heuristic function with higher values, as long as it does not overestimate (remains admissible)

How to generate admissible heuristics?

$\rightarrow$ Use *exact* solution cost of a relaxed (easier) problem

Steps:

– Consider problem P

– Take a problem P' easier than P

– Find solution to P'

– Use solution of P' as a heuristic for P

Relaxing the 8-puzzle problem

A tile can move mode square A to square B if

A is (horizontally or vertically) adjacent to B <u>and</u> B is blank

1. A tile can move from square A to square B if A is adjacent to B
   The rules are relaxed so that a tile can move to *any adjacent square*: the shortest solution can be used as a heuristic
   $(\equiv h_2(n))$

2. A tile can move from square A to square B if B is blank
   Gaschnig heuristic (Exercice 3.31, AIMA, page 119)

3. A tile can move from square A to square B
   The rules of the 8-puzzle are relaxed so that a tile can move *anywhere*: the shortest solution can be used as a heuristic
   $(\equiv h_1(n))$

An admissible heuristic for the TSP

Let path be *any* structure that connects all cities

$\implies$ minimum spanning tree heuristic (polynomial)

(Exercice 3.30, AIMA, page 119)

Combining several admissible heuristic functions

We have a set of admissible heuristics $h_1, h_2, h_3, \ldots, h_m$ but no heuristic that dominates all others, what to do?

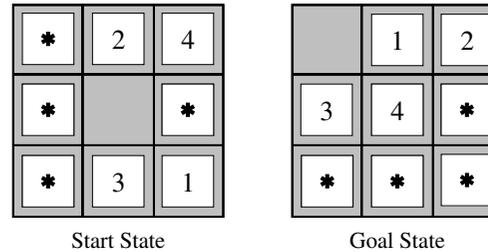$$\longrightarrow h(n) = \texttt{max}(h_1(n), h_2(n), \ldots, h_m(n))$$

h is admissible and dominates all others.

$\rightarrow$ Problem:

Cost of computing the heuristic (vs. cost of expanding nodes)

Using subproblems to derive an admissible heuristic function

Goal: get 1, 2, 3, 4 into their correct positions, ignoring the 'identity' of the other tiles

*	2	4
*		*
*	3	1

Start State

	1	2
3	4	*
*	*	*

Goal State

Cost of optimal solution to subproblem used as a lower bound (and is substantially more accurate than Manhattan distance)

Pattern databases:

- Identify patterns (which represent several possible states)

- Store cost of <u>exact</u> solutions of patterns

- During search, retrieve cost of pattern and use as a (tight) estimate

Cost of building the database is amortized over 'time'