Informed (Heuristic) Search

Outline

- Informed: use problem-specific knowledge
- General search strategy: Best-first search
- A* search
- Heuristic functions
  - How to invent them

2

Tree / Graph search

```
function GRAPH-SEARCH(problem) return a solution or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(problem.INITIAL-STATE))
    loop do
        if EMPTY?(fringe) then return failure
        node ← POP(fringe)
        if problem.GOAL-TEST(node.STATE)
            then return SOLUTION(node)
        add node.STATE to closed
        for each action in problem.ACTIONS(node.STATE) do
            child ← Child-Node(problem, node, action)
            if child.STATE is not in closed or fringe then
                fringe ← INSERT(child, fringe)
```

A strategy is defined by picking *the order of node expansion*

Best-first search

- General strategy of informed search:
  - *Best-first search*: node is selected for expansion based on an ***evaluation function f(n)***
- Idea: *f(n)* is a cost estimate
  - Choose node with lowest evaluation
- Implementation: graph search is identical to uniform-cost
  - *fringe* is a priority queue sorted in *f(n)*

Best-first Graph search

```
function Best-First-SEARCH(problem) return a solution or failure
    closed ← an empty set
    fringe ← a priority queue ordered by f(n) with MAKE-NODE(problem.INITIAL-
    STATE) as the only element
    loop do
        if EMPTY?(fringe) then return failure
        node ← POP(fringe)
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        add node.STATE to closed
        for each action in problem.ACTIONS(node.STATE) do
            child ←  Child-Node(problem, node, action)
            if child.STATE is not in closed or fringe then
                fringe ← INSERT(child, fringe)
            else if child.STATE is in fringe with higher f(n) then
                replace that fringe node with child
```

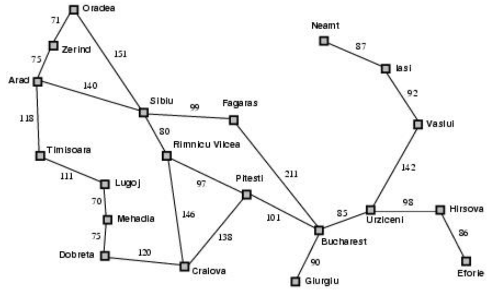
5

A heuristic function

- Knowledge of the problem is represented in the form of a **heuristic function**
- $h(n)$ = estimated cost of the cheapest path from the state at node *n* to a goal node.
- $h(n)$ nonnegative. If *n* is goal then $h(n)=0$

6

Romania with step costs



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

- h_{SLD}=straight-line distance heuristic.
- h_{SLD} can **NOT** be computed from the problem description itself

7

Greedy best-first search

- $f(n)=h(n)$
  - Expand node that appears to be closest to goal

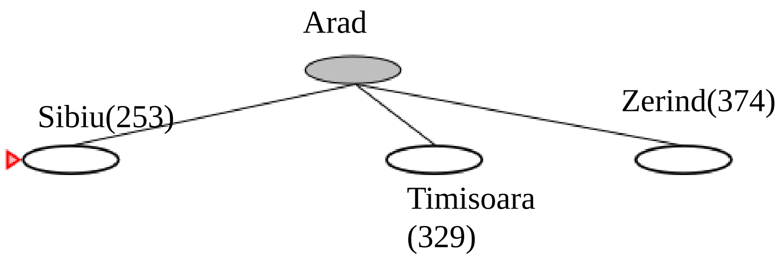
8

Greedy search example

- Assume that we want to use greedy search to solve the problem of traveling from Arad to Bucharest.
- The initial state=Arad

9

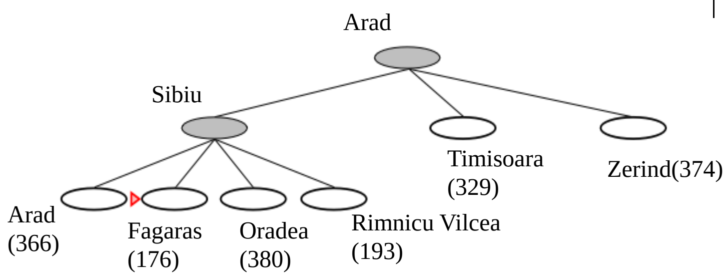
Greedy search example



- The first expansion step produces:
  - Sibiu, Timisoara and Zerind
- Greedy best-first will select Sibiu.

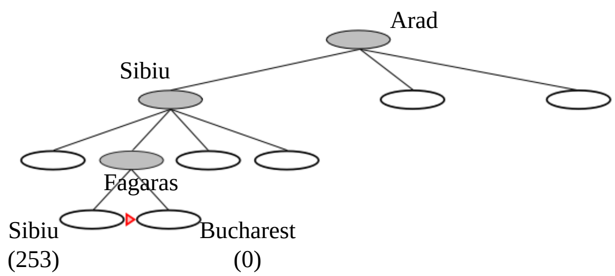
10

Greedy search example



- If Sibiu is expanded we get:
  - Arad, Fagaras, Oradea and Rimnicu Vilcea
- Greedy best-first search will select: Fagaras

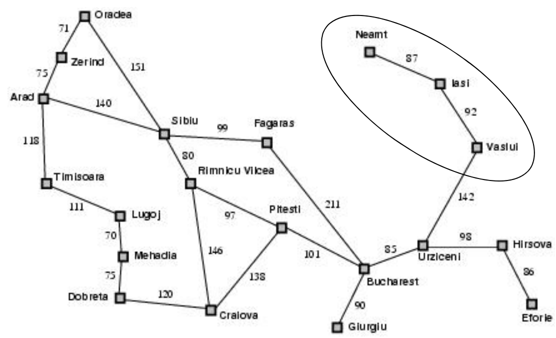
Greedy search example



- If Fagaras is expanded we get:
  - Sibiu and Bucharest
- Goal reached !!
  - Fast, yet not optimal (see Arad, Sibiu, Rimnicu Vilcea, Pitesti)

Greedy search, evaluation

- Completeness: NO (cf. DF-search)
  - Complete in finite space with graph search
  - Minimizing h(n) can result in false starts, e.g. Iasi to goal Fagaras.



13

Greedy search, evaluation

- Completeness: NO
- Time complexity? $O(b^m)$
  - Cf. Worst-case DF-search

  (m is maximum depth of search space)
  - Good heuristic can give dramatic improvement.
- Space complexity: $O(b^m)$
  - Keeps all nodes in memory
  - might jump to other branches without finishing current branch
- Optimality? NO

14

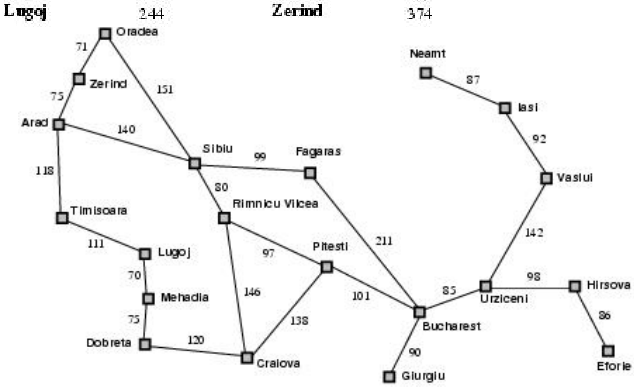
A* search

- Idea: avoid expanding paths that are already expensive.
- Evaluation function $f(n)=g(n) + h(n)$
  - $g(n)$ the cost (so far) to reach the node.
  - $h(n)$ estimated cost of the cheapest path to get from the node to goal.
  - $f(n)$ estimated total cost of the cheapest path through n to goal.

15

Romania example

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



16

A* search example

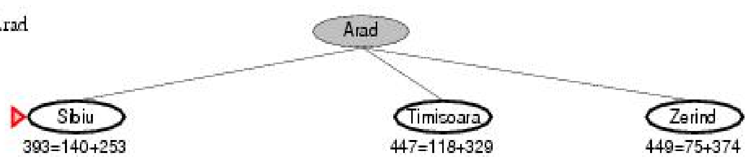
(a) The initial state

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

17

A* search example

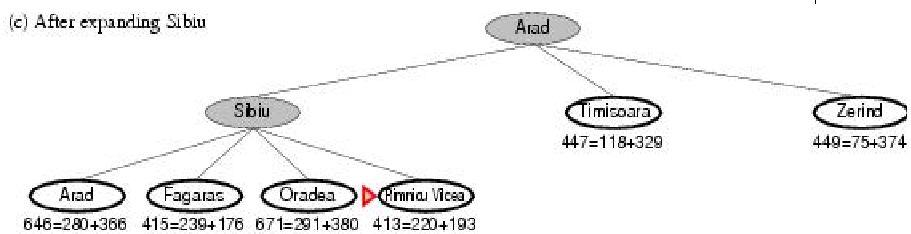
After expanding Arad



- 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

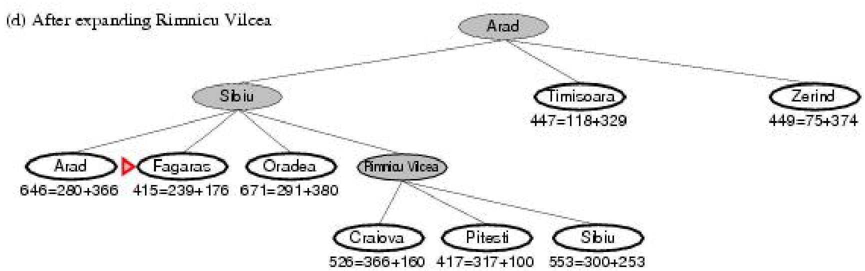
18

A* search example



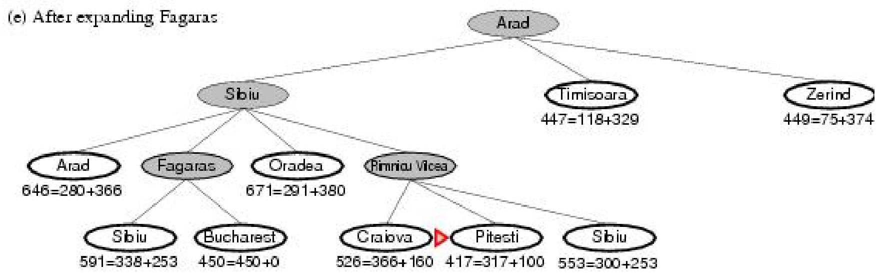
- Expand Sibiu and determine *f(n)* for each node
- Best choice is Rimnicu Vilcea

A* search example



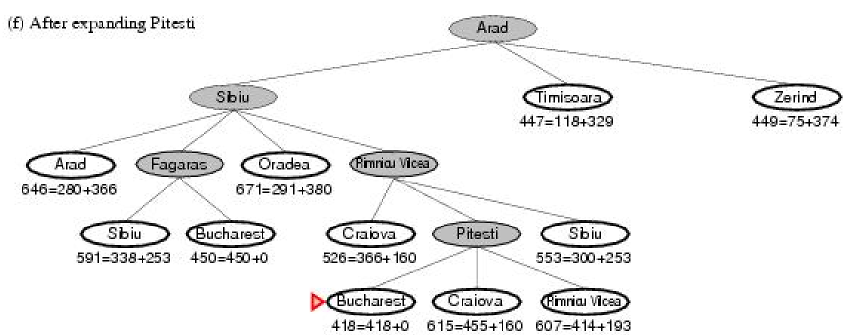
- Expand Rimnicu Vilcea and determine *f(n)* for each node
- Best choice is Fagaras

A* search example



- Expand Fagaras and determine *f(n)* for each node
- Best choice is Pitesti !!!

A* search example



- Expand Pitesti and determine *f(n)* for each node
- Best choice is Bucharest !!!
  - Optimal solution

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.

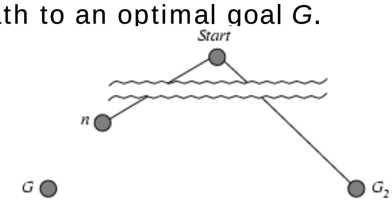
    e.g. $h_{SLD}(n)$ never overestimates the actual road distance

- Theorem: If $h(n)$ is admissible, A* using TREE-SEARCH is optimal

23

Optimality of A* (proof)

- Suppose some suboptimal goal node G_2 has been generated and is in the fringe.
- Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G.



$f(G_2)$	$= g(G_2)$	since $h(G_2)=0$
	$> g(G)=g(n)+h^*(n)$	since G_2 is suboptimal
	$>= f(n)$	since h is admissible

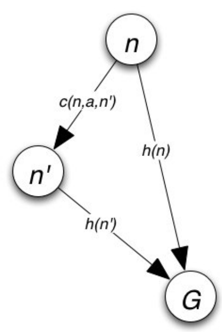
- Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

24

BUT … graph search

- Discards new paths to repeated state.
  - Optimal path to *n* may be discarded because *n* is already in the closed list -> because f(n) value may decrease along a path
- Solution:
  - Add extra bookkeeping i.e. remove more expensive of two paths (could be messy) Or,
  - 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

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

- If h is consistent, we have

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

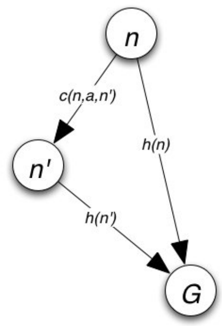
i.e. ***f(n) is nondecreasing along any path.***

The first time a node is selected for expansion, the optimal path to that node is found

- ***A* expands nodes in order of increasing f value***
- Theorem: If *h(n)* is consistent, A* using GRAPH-SEARCH is optimal

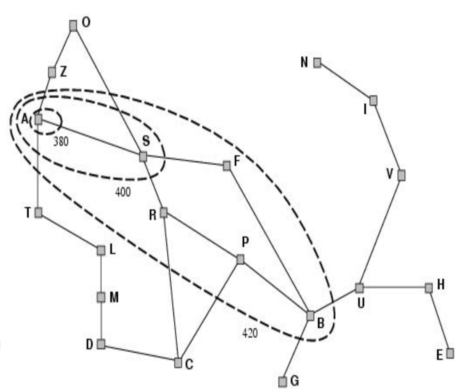
Consistency and Admissibiliy

- Every consistent heuristic is also admissible
- In practice, admissible but inconsistent heuristic functions are rare
  - Heuristics from relaxed problem are always consistent



Optimality of A* graph search with consistent heuristic

- *A* expands nodes in order of increasing f value*
- F-contours: Contours of nodes in state space with equal f values
- Optimality
  - A* gradually expands nodes in bands of increasing f values.
  - A* expands all nodes with $f(n) < C^*$
  - A* expands some nodes with $f(n) = C^*$
  - A* expands no nodes with $f(n) > C^*$



A* graph search, evaluation

- Completeness: YES
  - Since bands of increasing *f* are added
  - Assume step costs > positive constant
- Optimality: YES
  - 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)

- Any algorithm that does not expand all nodes with $f(n) < C^*$ runs the risk of missing the optimal solution

29

A* graph search, evaluation

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

30

A* search, evaluation

- Completeness: YES
- Optimality: YES
- Time complexity: exponential in path length

- Space complexity?

31

A* graph search, evaluation

- Completeness: YES
- Optimality: YES
- Time complexity: exponential in path length

- Space complexity:
  - It keeps all generated nodes in memory
  - Hence *space is the major problem* not time

32

Memory-bounded heuristic search

- Some solutions to A* space problems (maintain completeness and optimality)
  - Iterative-deepening A* (IDA*)
    - DFS but cutoff information is the *f*-cost (*g*+*h*) instead of depth
    - practical for problems with unit step costs
    - May incur substantial overhead with real-valued costs
  - Recursive best-first search(RBFS)
    - Recursive algorithm that attempts to mimic standard best-first search with linear space.
  - (simple) Memory-bounded A* ((S)MA*)
    - Drop the worst-leaf node when memory is full

33

RBFS evaluation

- Like A*, optimal if *h(n)* is admissible
- Space complexity is *O(bd)*.
- RBFS is a bit more efficient than IDA*
  - Still excessive node re-generation (mind changes)
- Time complexity difficult to characterize
  - Depends on accuracy of h(n) and how often best path changes.
  - Cannot check for repeated states (neither IDA*)
- IDA* and RBFS suffer from using ***too little*** memory.

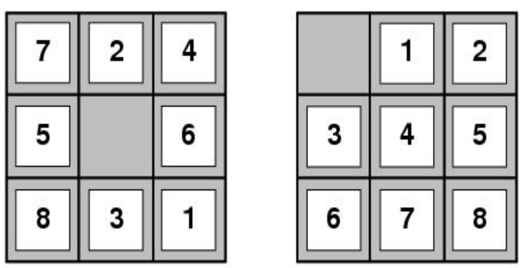
34

Simplified memory-bounded A*

- Use all available memory.
  - I.e. expand best leafs like A* until available memory is full
  - When full, SMA* drops worst leaf node (highest *f*-value)
  - Like RBFS backup value of forgotten node to its parent
- SMA* is complete and optimal with available memory
- Textbook: SMA* might be a good choice as general-purpose algorithm for finding optimal solutions, when the state-space is a graph, step costs are not uniform

35

Heuristic functions

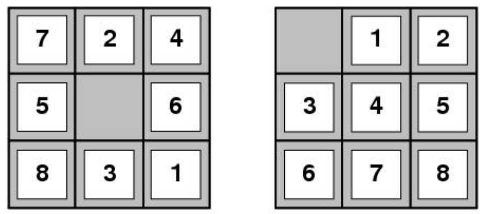


Start State Goal State

- Heuristics for the 8-puzzle?

36

Heuristic functions



Start State   Goal State

- For the 8-puzzle, 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/City block distance).
  - $h_2(s)=3+1+2+2+2+3+3+2=18$
- Admissible?

37

Heuristic quality

- 1200 random 8-puzzle with solution lengths from 2 to 24 (100 each even depth).

	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

38

Heuristic quality

- *Effective branching factor* b* characterizes the quality of a heuristic
  - Is the branching factor that a uniform tree of depth *d* would have in order to contain *N+1* nodes generated by a search algorithm

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

  - Measure is fairly constant for sufficiently hard problems.
    - Experimental measurements of b* on a small set of problems can thus provide a good guide to the heuristic's overall usefulness.
    - A good value of b* is close to 1.

39

Dominance

- If $h_2(n) >= h_1(n)$ for all n (both admissible)
  then h_2 *dominates* h_1 and is always better for search

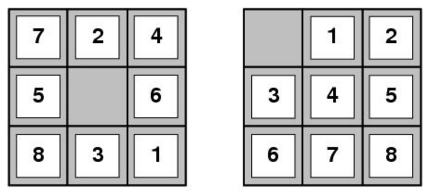
- Given a collection of admissible (consistent) heuristics
  $h(n) = max\ \{h_1(n), ..., h_m(n)\}$
  is also admissible (and consistent) and dominates its components

40

Inventing admissible heuristics

- Admissible heuristics can be derived from the **exact** solution cost of a **relaxed** version of the problem

- A problem with fewer restrictions on the actions is called a relaxed 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.



Start State Goal State

41

Inventing admissible heuristics

- Admissible heuristics can be derived from the **exact** solution cost of a **relaxed** version of the problem
  - The optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.

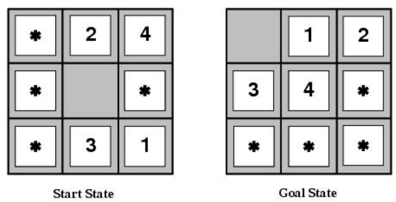
- The cost of an optimal solution to a relaxed problem is a consistent heuristic for the original problem (because it's a true cost)

- Construct relaxed problems automatically
  - If the problem definition can be written down in a formal language
  - ABSolver (1993) found the first useful heuristic for the Rubik's cube

42

Inventing admissible heuristics

- Admissible heuristics can also be derived from the solution cost of a **subproblem** of a given problem.
  - This cost is a lower bound on the cost of the real problem.
- **Pattern databases** store the exact solution for every possible subproblem instance.
  - The heuristic for complete state is obtained by looking up the DB during search
- Can also construct DBs for 5-6-7-8 etc. to combine heuristics



43