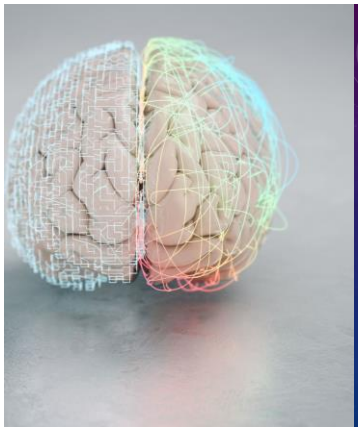


ARTIFICIAL INTELLIGENCE

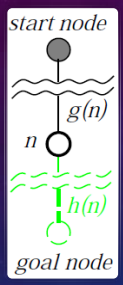
•Lecture 5 – Informed Search

1

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:
  - Uniform Cost Search (uninformed)
  - Greedy (best-first) Search (informed)
  - A* Search (informed)

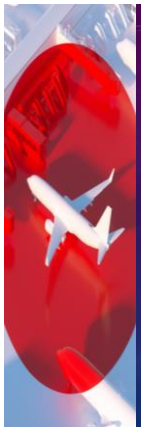
2



EVALUATION FUNCTION

- Evaluation function f(n) = g(n) + h(n)
  - g(n) = exact cost so far to reach n
  - h(n) = estimated cost to goal from n
  - f(n) = estimated total cost of cheapest path through n to goal
- Special cases:
  - Uniform Cost Search: f(n) = g(n)
  - Greedy (best-first) Search: f(n) = h(n)
  - A* Search: f(n) = g(n) + h(n)

3

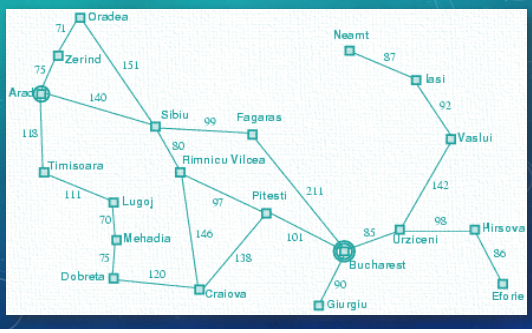


EXAMPLE: TRAVELING IN ROMANIA

- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow from Bucharest
- Formulate goal: Be in Bucharest
- Formulate problem:
  - States: various cities
  - Actions: drive between cities
- Find solution:
  - Sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest.

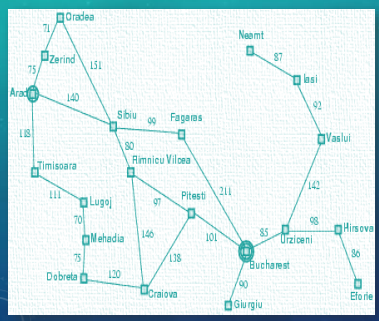
4

EXAMPLE: TRAVELING IN ROMANIA



5

ROMANIA - 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

6

GREEDY BEST-FIRST SEARCH

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

24 July 2024 7

7

GREEDY BEST-FIRST SEARCH EXAMPLE

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

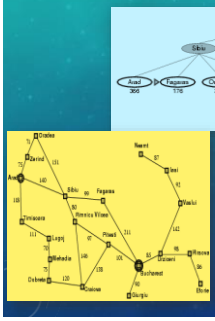
8

GREEDY BEST-FIRST SEARCH EXAMPLE

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

9

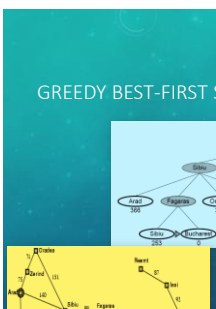
GREEDY BEST-FIRST SEARCH EXAMPLE



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

10

GREEDY BEST-FIRST SEARCH EXAMPLE



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

11

PROPERTIES OF GREEDY BEST-FIRST SEARCH

Complete? No – can get stuck in loops, e.g., with Oradea as goal and start from Iasi:

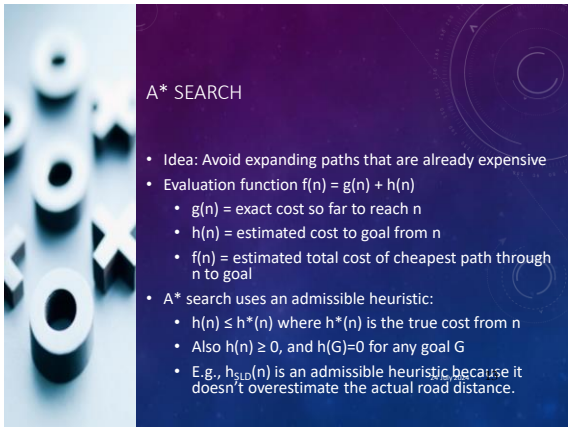
- Iasi → Neamt → Iasi → Neamt →
  Complete in finite space with repeated state checking

Time? $O(b^m)$, but a good heuristic can give dramatic improvement

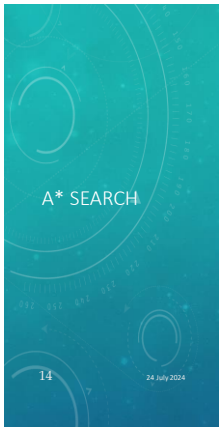
Space? $O(b^m)$ -- keeps all nodes in memory

Optimal? No.

12



13

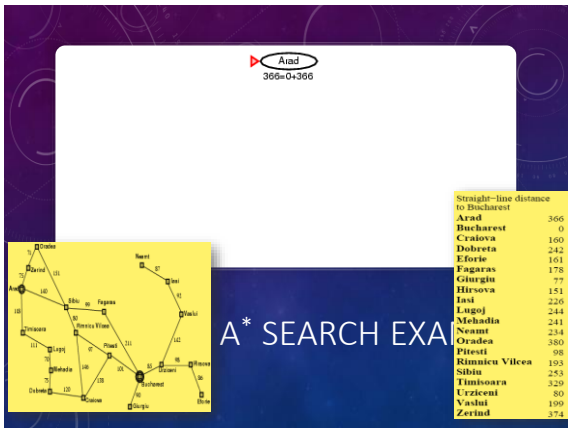


If we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of g(n) + h(n)

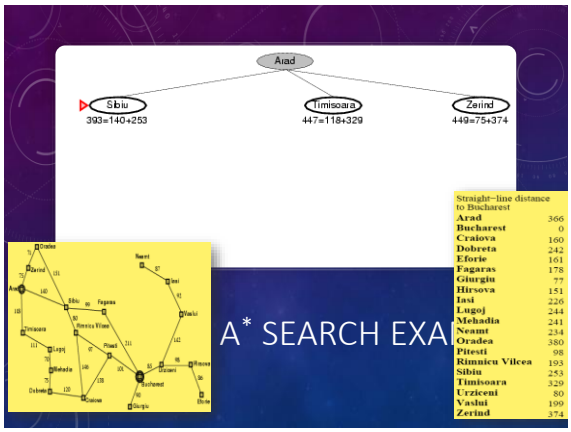
This strategy is more than just reasonable

Provided that h(n) satisfies certain conditions, A* using TREE search is both complete and optimal.

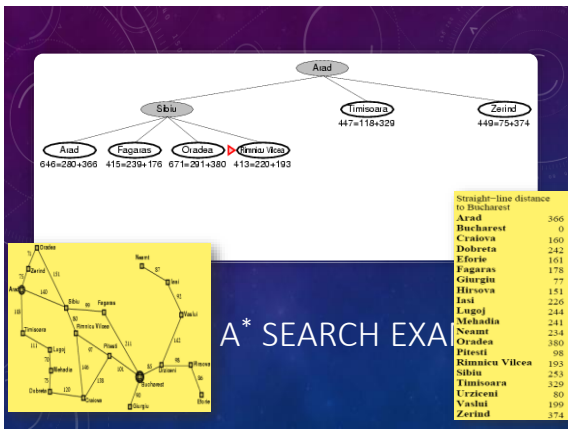
14



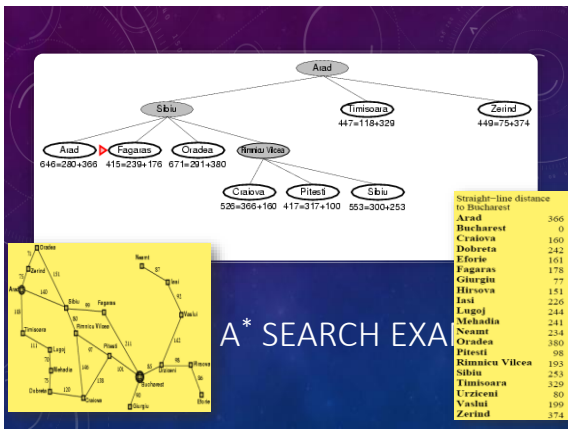
15



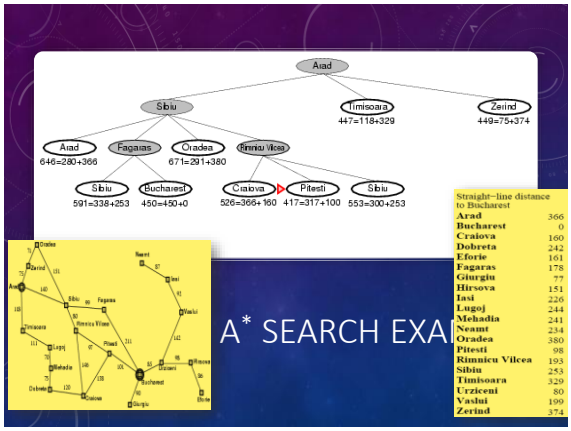
16



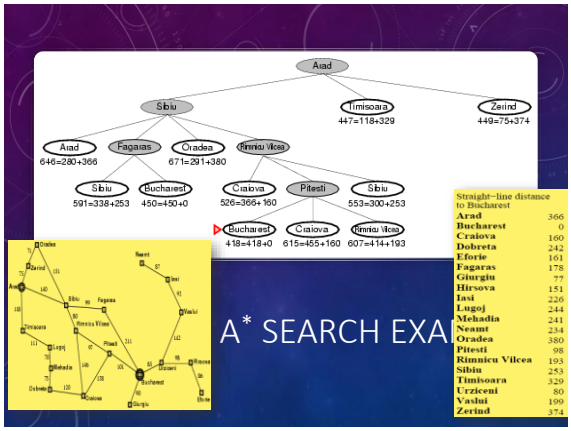
17



18



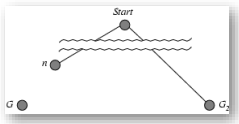
19



20

OPTIMALITY OF A* (PROOF)

Suppose some suboptimal goal 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.

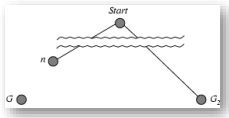


$f(G_2) = g(G_2)$ since $h(G_2) = 0$
$g(G_2) > g(G)$ since G_2 is non-optimal
$f(G) = g(G)$ since $h(G) = 0$
$f(G_2) > f(G)$ from above

21

OPTIMALITY OF A* (PROOF)

Suppose some suboptimal goal 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.



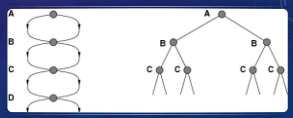
$f(G_2) > f(G)$ from above
$h(n) \le h^*(n)$ since h is admissible
$g(n) + h(n) \le g(n) + h^*(n)$
$f(n) \le f(G)$

Hence $f(G_2) > f(n)$, and A* will never select G_2 for expansion

22

PROBLEM OF REPEATED STATES

- Failure to detect repeated states can turn a linear problem into an exponential one!
  - We don't want to expand a node that has already been expanded



24 July 2024 23

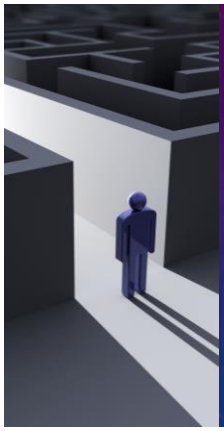
23

GRAPH SEARCH (INSTEAD OF TREE SEARCH)

- Maintain a closed-list containing those nodes that have already been expanded. Then, if a node is encountered that is already in closed-list, it is simply ignored
- This guarantees that no loops are generated, and essentially converts the graph into a tree

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
  closed ← an empty set
  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](STATE[node]) then return SOLUTION(node)
    if STATE[node] is not in closed then
      add STATE[node] to closed
      fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

24



GRAPH-SEARCH PROBLEM

- Its very well to prune the search space by ignoring repeated states
- But the problem is that Graph-search can end up discovering sub-optimal solutions
  - Basically, a loop means that there might be more then one path to a node
  - Once we discover a path to a node, then any other paths to the same node are ignored by graph-search
  - However, it is not necessary that the first path is the optimal one
  - Hence, sub-optimal solutions can be returned.

25

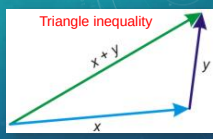
GRAPH-SEARCH PROBLEM

- Solution?
  - We can adopt a uniform-cost approach, in which we keep track of all the paths that have been currently generated
  - Then, we select only that path which has the least-cost.

26

PROBLEM WITH A* PROOF

- This proof can break down with Graph-search
- A* can return sub-optimal solutions, if we don't apply the uniform-cost approach
- However, this is really messy and expensive
- A much better solution is to ensure that the heuristic that you have selected is **consistent**

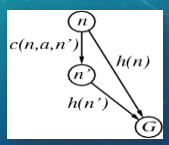


27

CONSISTENT HEURISTICS

- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*,
  $h(n) \leq c(n,a,n') + h(n')$
- If *h* is consistent, we have

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

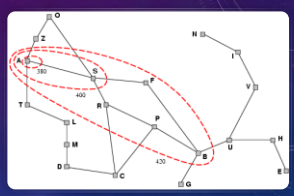


- i.e., *f(n)* is non-decreasing along any path.
- Theorem: If *h(n)* is consistent, A* using GRAPH-SEARCH is optimal

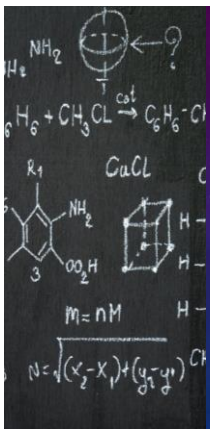
28

OPTIMALITY OF A*

- A* expands nodes in order of increasing *f* value
- Gradually adds "*f*-contours" of nodes
- Contour *i* has all nodes with $f=f_i$, where $f_i < f_{i+1}$



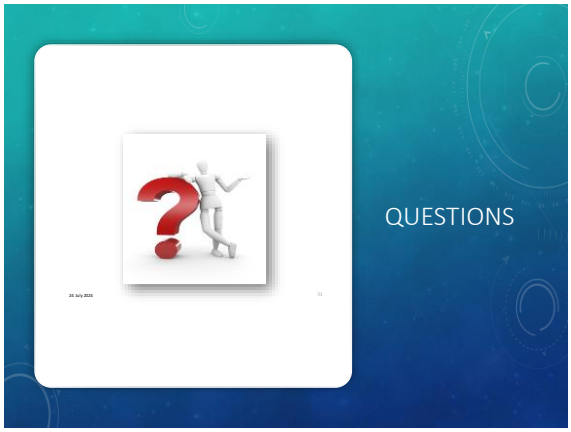
29



PROPERTIES OF A*

- Complete? Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

30



31