Informed Search

Bùi Tiến Lên

2022

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Contents

Informed search strategies

Informed (**heuristic**) search strategies

- Use problem-specific knowledge beyond the definition of the problem itself
- Can find solutions more efficiently than can an uninformed strategy

What is heuristic

- **Additional knowledge** of the problem is imparted to the search algorithm.
- Heuristic **estimates** how close a state is to a goal.

Best-first search

Best-first search

- An instance of the general **tree/graph search** algorithm
- A node is selected for expansion based on an **evaluation function**, $f(n)$, which is construed as a cost estimate.
  - The node with the *lowest* evaluation $f(n)$ is expanded first
  - The implementation of best-first graph search is identical to that for uniform-cost search, except for the use of f instead of g to order the priority queue
  - The choice of f determines the search strategy

Best-first search (cont.)

- Most best-first algorithms include as a component of f a **heuristic function**, denoted $h(n)$:

  $h(n)$ estimated cost of the cheapest path from the state at node n to a goal state

- $h(n)$ depends only on the state at node n
- Assumption of h
  - Arbitrary, nonnegative, problem-specific functions
  - Constraint: if n is a goal node, then $h(n) = 0$

Greedy best-first search

Greedy best-first search

- Try to expand the node that is **closest** to the goal, on the grounds that this is likely to lead to a solution quickly.
- Thus, it evaluates nodes by using just the heuristic function

$$f(n) = h(n)$$

Illustration: Romania

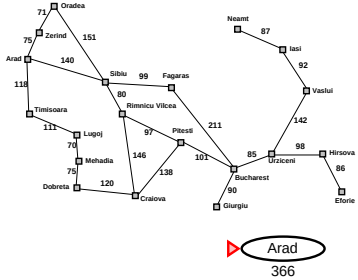
- Use the **straight-line distance** heuristic, which we will call h_{SLD}

Table 1: Values of h_{SLD} – straight-line distances to Bucharest.

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	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

Illustration: Romania (cont.)

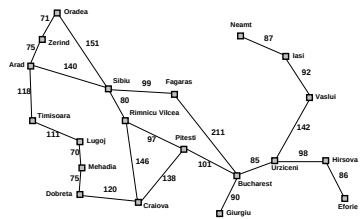
- The initial state



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

Illustration: Romania (cont.)

- After expanding Arad



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

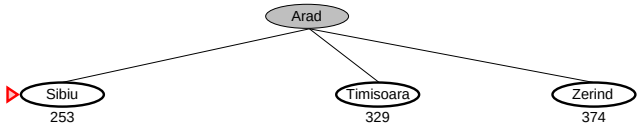
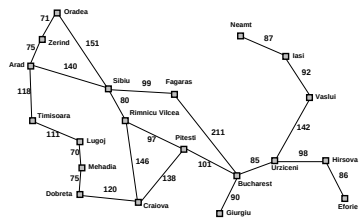


Illustration: Romania (cont.)

- After expanding Sibu



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

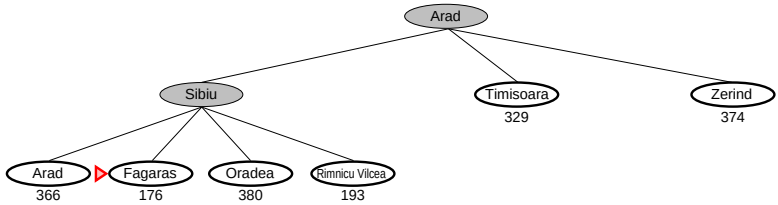
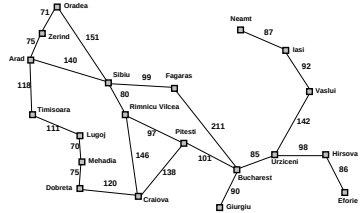


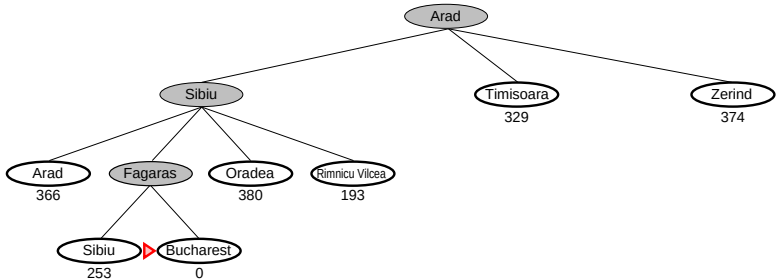
Illustration: Romania (cont.)

- After expanding Fagaras



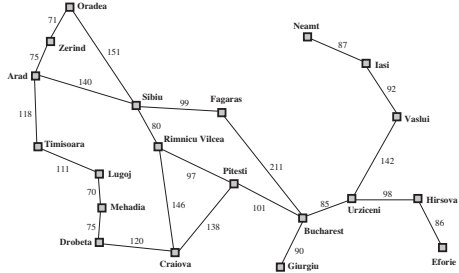
Straight-line distance to Bucharest

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



Properties

- **Time**: $O(b^m)$, but a good heuristic can give dramatic improvement
- **Space**: $O(b^m)$, keeps all nodes in memory
- **Complete**: No, can get stuck in loops, e.g., find a path from Iasi to Fagaras, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$ ...
  - Yes in finite space with repeated-state checking
- **Optimal**: No



A* search

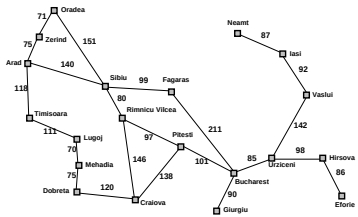
A* search

- A* search (pronounced A-star search) is the most widely known form of best-first search
- Ideas
  - Use heuristic to guide search, but not only
  - 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 of the cheapest path to goal from n
  - $f(n)$: estimated cost of the cheapest solution through n
- The **algorithm** is identical to UNIFORM-COST-SEARCH except that A* uses $g + h$ instead of g

Illustration: Romania



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

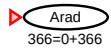
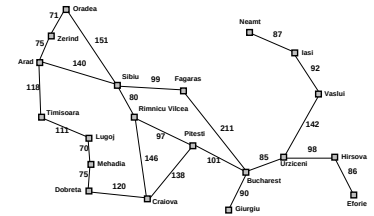


Illustration: Romania (cont.)



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

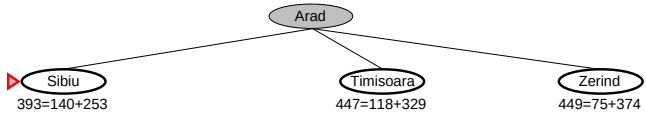
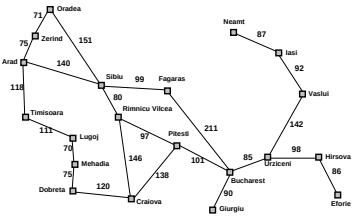


Illustration: Romania (cont.)



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

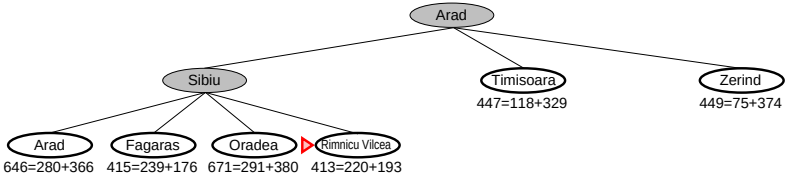
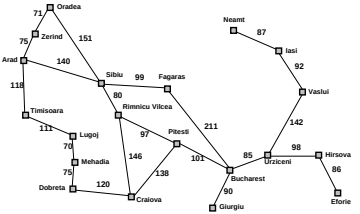


Illustration: Romania (cont.)



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

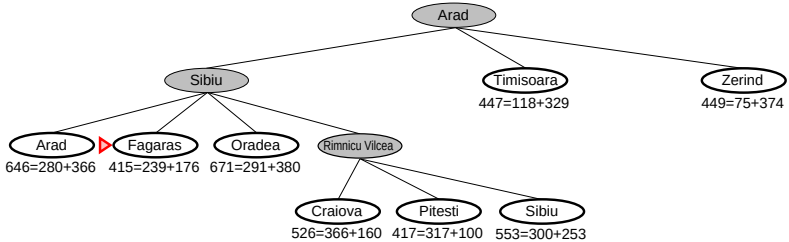
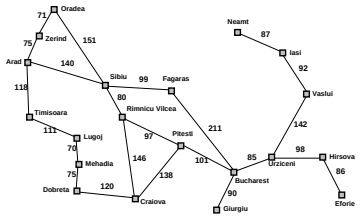


Illustration: Romania (cont.)



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

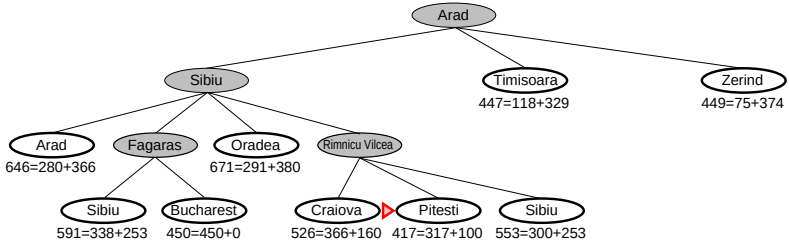
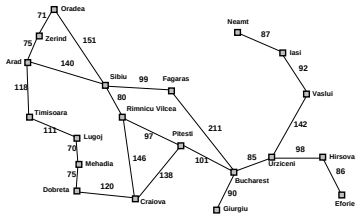
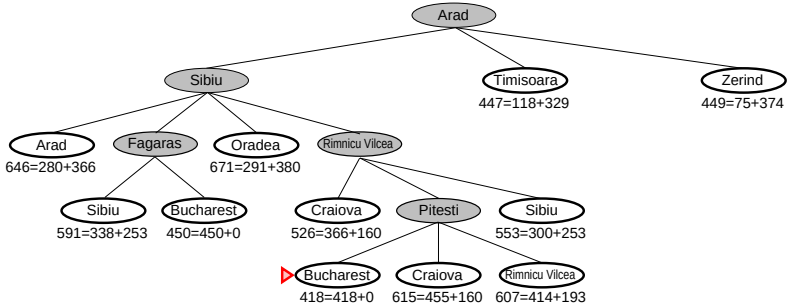


Illustration: Romania (cont.)



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



Properties

- **Time** Exponential in [relative error in $h \times$ length of solution.]
- **Space** Keeps all nodes in memory
- **Complete** Yes, unless there are infinitely many nodes with $f \leq f(G)$
- **Optimal** Yes, cannot expand f_{i+1} until f_i is finished
  - A* expands all nodes with $f(n) < C^*$
  - A* expands some nodes with $f(n) = C^*$
  - A* expands no nodes with $f(n) > C^*$

Admissibility and Consistency

Concept 1

$h(n)$ is an **admissible** heuristic

- if it *never overestimates* the cost to reach the goal
- or if for every node n, $h(n) \leq h^*(n)$ ($h^*(n)$ is a the true optimal cost to reach the goal state from n)

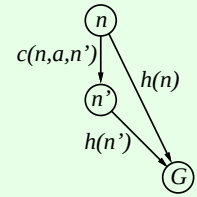
- An example of an admissible heuristic is the straight-line distance h_{SLD} that we used in getting to Bucharest

Admissibility and Consistency (cont.)

Concept 2

$h(n)$ is an **consistent** heuristic if for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n'

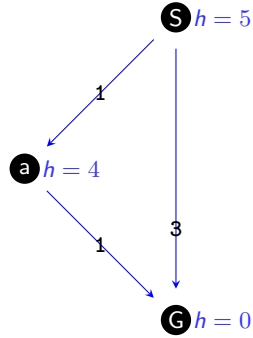
$$h(n) \leq c(n, a, n') + h(n')$$



- Consistent heuristic is also admissible

Optimal for A*

- Using **tree search** algorithm to find a path from S to G



Optimal for A* (cont.)

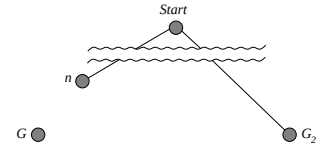
Theorem 1

*The **tree-search version** of A* is optimal if $h(n)$ is admissible*

Proof

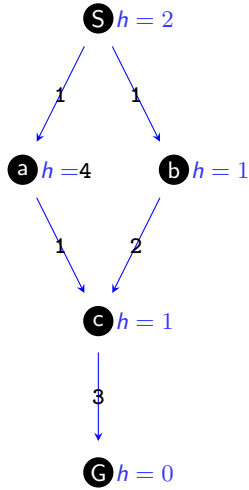
Suppose some suboptimal goal G_2 has been generated and is in the frontier. Let n be an unexpanded node on a shortest path to an optimal goal G (graph separation)

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



Optimal for A* (cont.)

- Using **graph search** algorithm to find a path from S to G



Optimal for A* (cont.)

Theorem 2

*The **graph-search version** of A* is optimal if $h(n)$ is consistent*

Proof

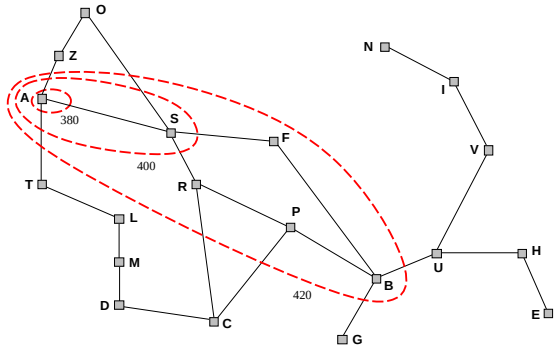
- if $h(n)$ is consistent, then the values of $f(n)$ along any path are nondecreasing. Suppose n' is a successor of n; then $g(n') = g(n) + c(n, a, n')$ for some action a, and we have

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

- Whenever A* selects a node n for expansion, the optimal path to that node has been found. Were this not the case, there would have to be another frontier node n' on the optimal path from the start node to n; $f(n') < f(n)$ $\Rightarrow$ n' would have been selected first.

Contours

- Search algorithm expands nodes in order of increasing f value
- We can draw **contours** in the state space, just like the contours in a topographic map
  - Gradually adds "f-contours" of nodes, contour i has all nodes with $f < f_i$, where $f_i < f_{i+1}$



Contours (cont.)

- The bands of uniform-cost search will be “circular” around the start state.
- The bands of A* with more accurate heuristics will stretch toward the goal state and become more narrowly focused around the optimal path

The 8-puzzle

- The average solution cost for a randomly generated 8-puzzle instance is about 22 steps
- The branching factor b is about 3
- 8-puzzle:
  - $3^{22} \approx 3.1 \times 10^{10}$ states (using an exhaustive tree search)
  - $9!/2 = 181,440$ reachable states (using graph search)
- 15-puzzle: around 10^{13} states

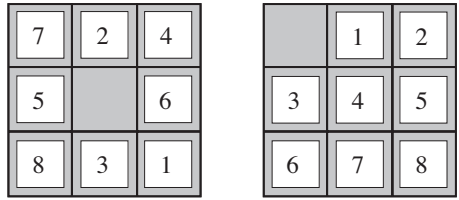
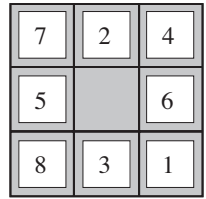


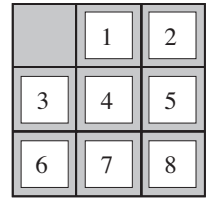
Figure 1: A typical instance of the 8-puzzle. The optimal solution is 26 steps long.

Admissible heuristics for 8-puzzle

- h_1 = the number of misplaced tiles. h_1 is an admissible heuristic because it is clear that any tile that is out of place must be moved at least once
- h_2 = the sum of the distances of the tiles from their goal positions. h_2 is also admissible because all any move can do is move one tile one step closer to the goal



Start State

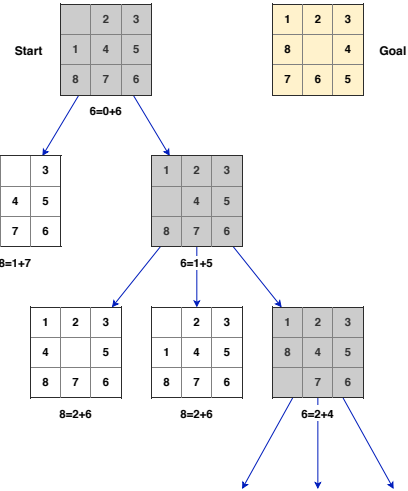


Goal State

- $h_1 = 1+1+1+1+1+1+1+1 = 8$ and
  $h_2 = 3+1+2+2+2+3+3+2 = 18$

Solve 8 puzzle

- Using h_2



Memory-bounded heuristic search

Memory-bound heuristic search

- In practice, A* usually runs out of space long before it runs out of time
- **Idea**: try something like DFS, but not forget everything about the branches we have partially explored

Iterative-deepening A* (IDA*)

- The main difference with IDS
  - Cut-off use the f-value ($g + h$) rather than the depth
  - At each iteration, the cutoff value is the smallest f-value of any node that exceeded the cutoff on the previous iteration
- Avoid the substantial overhead associated with keeping a sorted queue of nodes.
- Practical for many problems with unit step costs, yet difficult with real valued costs

Algorithm

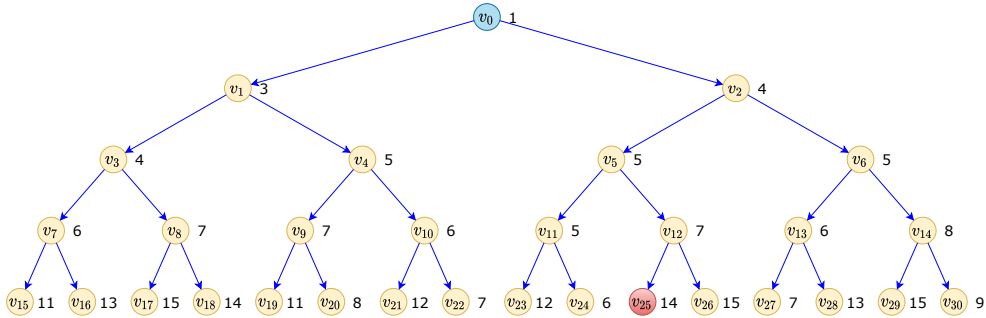
```
function IDA*(problem) returns a solution, or failure
local variables:
  f_limit: the current f-COST limit
  root: a node
  root ← MAKE-NODE(problem.INITIAL-STATE)
  f_limit ← root.f
  loop do
     solution, f_limit ← DFS-CONTOUR(problem, root, f_limit)
     if solution is non-null  then return solution
     if f_limit = ∞ then return failure
```

Algorithm (cont.)

```
function DFS-Contour(problem, node, f_limit)
returns a solution, or failure and a new f-cost limit
  if node.f > f_limit then return null, node.f
  if problem.Goal-Test(node.State) then return Solution(node)
  successors ← ∅
  for each action in problem.Actions(node.State) do
    add Child-Node(problem, node, action) into successors
  f_next ← ∞
  for each s in successors do
    solution, f_new ← DFS-Contour(problem, s, f_limit)
    if solution is non-null then return solution, f_limit
    f_next ← Min(f_next, f_new)
  return null, f_next
```

Illustration

- IDA* search for the shortest route from v_0 (start node) to v_{25} (goal node); each node has a f value.



Recursive best-first search (RBFS)

- Keep track of the f-value of the best alternative path available from any ancestor of the current node
  - backtrack when the current node exceeds f_limit
- As it backtracks, replace f-value of each along the path with the best value $f(n)$ of its children

Algorithm

```
function RECURSIVE-BEST-FIRST-SEARCH(problem)
returns a solution, or failure
  return RBFS(problem,MAKE-NODE(problem.INITIAL-STATE),∞)
function RBFS(problem,node,f_limit)
returns a solution, or failure and a new f-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← ∅
  for each action in problem.ACTIONS(node.STATE) do
    add CHILD-NODE(problem,node,action) into successors
  if successors is empty then return failure,∞
  for each s in successors do
    # update f with value from previous search, if any
    s.f ← MAX(s.g + s.h, node.f)
  loop do
    best ← lowest f-value node in successors
    if best.f > f_limit then return failure,best.f
    alternative ← the second-lowest f-value among successors
    result,best.f ← RBFS(problem, best, MIN(f_limit, alternative))
    if result ≠ failure then return result
```

Illustration

- Stages in an RBFS search for the shortest route to Bucharest. The f-limit value for each recursive call is shown on top of each current node, and every node is labeled with its f-cost.

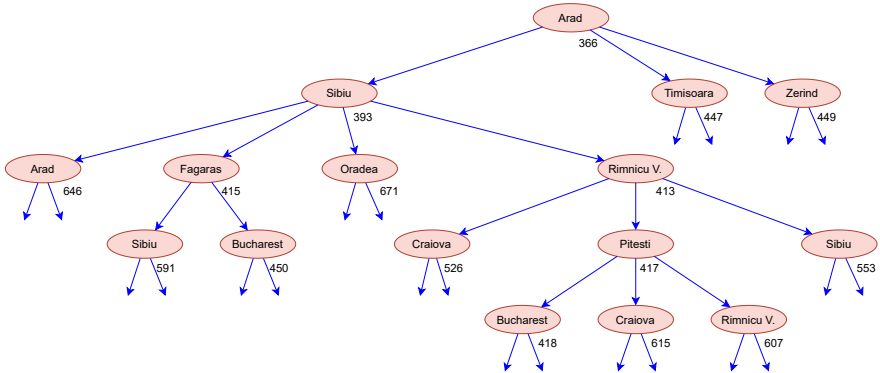
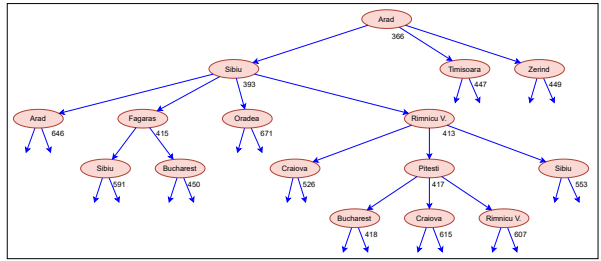


Illustration (cont.)

(a) The path via Rimnicu Vilcea is followed until the current best leaf (Pitesti) has a value that is worse than the best alternative path (Fagaras).



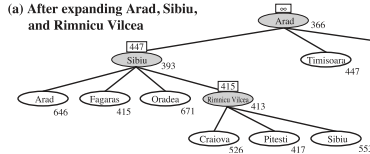


Illustration (cont.)

(b) The recursion unwinds and the best leaf value of the forgotten subtree (417) is backed up to Rimnicu Vilcea; then Fagaras is expanded, revealing a best leaf value of 450.

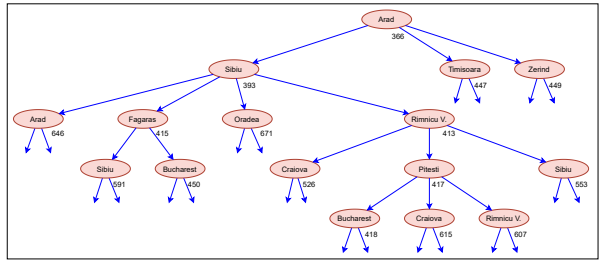
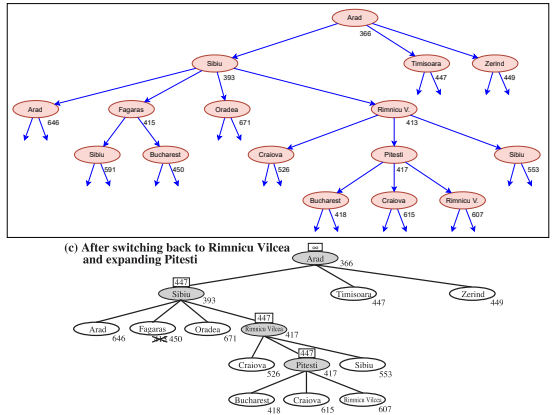


Illustration (cont.)

(c) The recursion unwinds and the best leaf value of the forgotten subtree (450) is backed up to Fagaras; then Rimnicu Vilcea is expanded. This time, because the best alternative path (through Timisoara) costs at least 447, the expansion continues to Bucharest.



Properties

- **Optimality**
  - Like A*, optimal if $h(n)$ is admissible
- **Time complexity**
  - Difficult to characterize
  - Depends on accuracy of $h(n)$ and how often best path changes
  - Can end up "switching" back and forth
- **Space complexity**
  - Linear time: $O(bd)$
  - Other extreme to A* - uses too little memory even if more memory were available

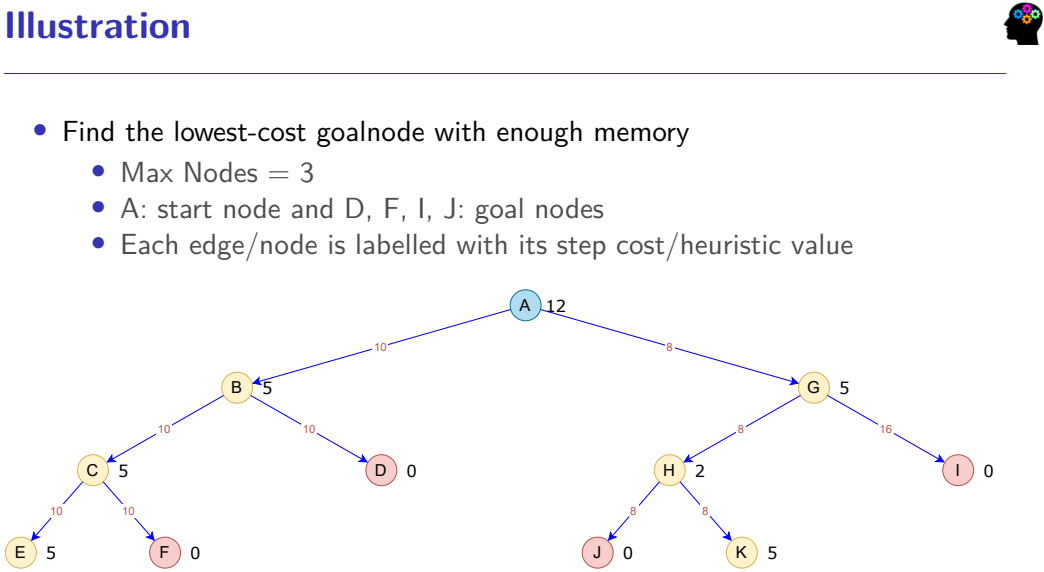
Simplified Memory-bound A* – SMA*

- Similar to A*, but **delete the worst node** (largest f-value) when memory is full
- SMA* expands the (**newest**) best leaf and deletes the (**oldest**) worst leaf
- SMA* also backs up the value of the forgotten node to its parent.
  - If there is a tie (equal f-values), delete the oldest nodes first
- Simplified-MA* finds the **optimal reachable solution** given the memory constraint.
  - The depth of the shallowest goal node, is less than the memory size (expressed in nodes)
- Time can still be exponential.

Algorithm

```
function SMA*(problem) returns a solution, or failure
  frontier ← MAKE-NODE(problem.INITIAL-STATE)
  loop do
    if frontier = ∅ then return failure
    node ← POP(frontier)
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    s ← NEXT-SUCCESSOR(node)
    if s is not a goal and is at maximum depth then
        s.f ← ∞
    else
        s.f ← MAX(s.g + s.h, node.f)
    if all of node's successors have been generated then
      update node's  f-cost and those of its ancestors if necessary
    if SUCCESSORS(node) all in memory then
      remove node from frontier
    if memory is full then
      delete shallowest, highest  f-cost node in frontier
      remove it from its parent's successor list
      insert its parent on frontier if necessary
    frontier ← INSERT(s, frontier)
```

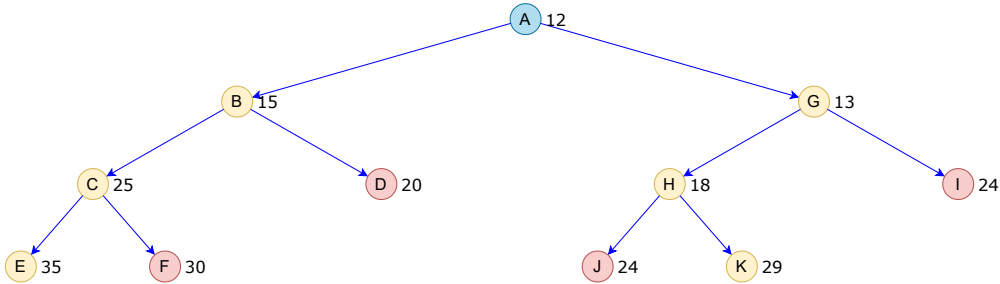
Illustration



- Find the lowest-cost goalnode with enough memory
  - Max Nodes = 3
  - A: start node and D, F, I, J: goal nodes
  - Each edge/node is labelled with its step cost/heuristic value

Illustration (const.)

- Consider only current f-cost



Learning to search better

- Could an agent learn how to search better? Yes
- **Metalevel state space**: in which each state captures the internal (computational) state of a program that is searching in an **object-level state space**.
- A **metalevel learning algorithm** learn from these experiences to avoid exploring unpromising subtrees
- For example, the map of Romania problem
  - The internal state of the A* algorithm is the current search tree.
  - Each action in the metalevel state space is a computation step that alters the internal state; for example, each computation step in A* expands a leaf node and adds its successors to the tree

Heuristic functions

The effect of heuristic accuracy on performance

- **Effective branching factor** b^* characterize the quality of a heuristic.
- If the total number of nodes generated by A* for a particular problem is N and the solution depth is d, then b^* is the branching factor that a uniform tree of depth d would have to have in order to contain $N + 1$ nodes

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

- b^* varies across problem instances, but fairly constant for sufficiently hard problems
- A well-designed heuristic would have a value of b^* close to 1 $\Rightarrow$ fairly large problems solved at reasonable cost

The effect of heuristic accuracy on performance (cont.)

	Search Cost (nodes generated)			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

Table 2: Comparison of the search costs and effective branching factors for the IDS and A* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d.

Dominance and Composite

- If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 **dominates** h_1 and is better for search
  - A* using h_2 will never expand more nodes than A* using h_1
- Hence, it is generally better to use a heuristic function with **higher values**, provided it is consistent and that the computation time for the heuristic is not too long.
- Given a collection of admissible heuristics $\{h_1, ..., h_m\}$ is available for a problem and none of them dominates any of the others.
- The **composite heuristic function** is defined as

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

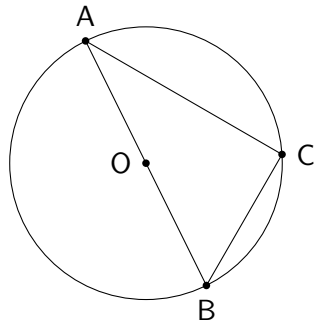
is consistent and dominates all component heuristics

Relaxed problems

Concept 3

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

- Prove that the angle in a semicircle is a right angle.



Generating admissible heuristics from relaxed problems

- The state-space graph of the relaxed problem is a **supergraph** of the original state space because the removal of restrictions creates added edges in the graph.

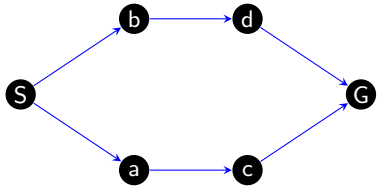


Figure 2: Original problem state-space

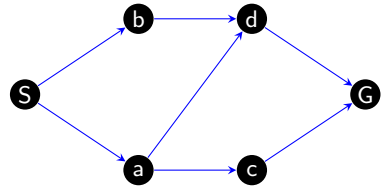


Figure 3: Relaxed problem state-space

Generating admissible heuristics from relaxed problems (cont.)

- Hence, *the cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.*
- Furthermore, because the derived heuristic is an exact cost for the relaxed problem, it must obey the triangle inequality and is therefore **consistent**.

Generating admissible heuristics from relaxed problems (cont.)

Original problem of 8 puzzle

- A tile can move from square A to square B if A is horizontally or vertically adjacent to B and B is blank

We can generate three relaxed problems by removing one or both of the conditions

Relaxed problems

1. A tile can move from square A to square B if A is adjacent to B $\rightarrow h_2$
2. A tile can move from square A to square B if B is blank $\rightarrow$ exercise
3. A tile can move from square A to square B $\rightarrow h_1$

Generating admissible heuristics from subproblems

- Admissible heuristics can also be derived from the solution cost of a **subproblem** of a given problem
  - The cost of the optimal solution of this subproblem $\leq$ the cost of the original problem (lower bound).
  - More accurate than Manhattan distance in some cases

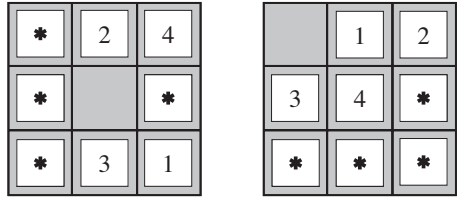
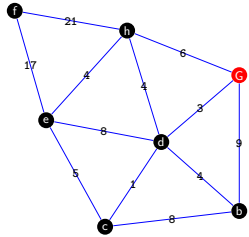


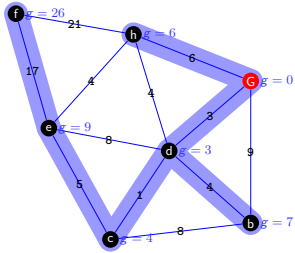
Figure 4: A subproblem of the 8-puzzle instance. The task is to get tiles 1, 2, 3, and 4 into their correct positions, without worrying about what happens to the other tiles.

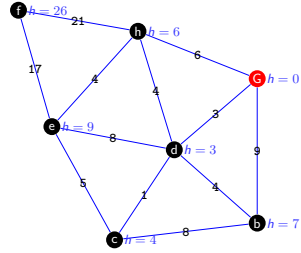
Goal fixed?

- Assume that the goal state G is fixed
- Using UNIFORM-COST-SEARCH to find the shortest path from G to the other states
- The final g values can be considered as optimal h^* values

Goal fixed? (cont.)



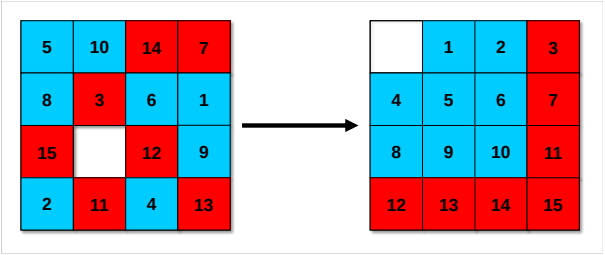




Pattern databases

- **Pattern databases** (PDB) where store the exact solution costs for every possible subproblem instance
  - E.g., every possible configuration of the four tiles and the blank
- The database itself is constructed by searching back from the goal and recording the cost of each new pattern encountered; the expense of this search is amortized over many subsequent problem instances
- The complete heuristic is constructed using the patterns in the databases

Computing the Heuristic

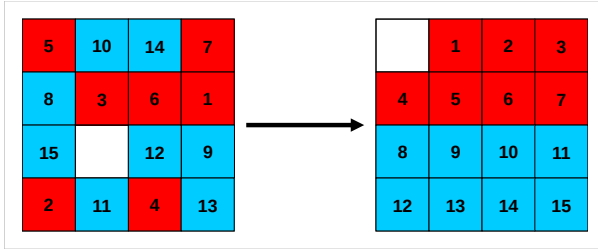


- 31 moves needed to solve red tiles
- 22 moves need to solve blue tiles
- Overall heuristic is maximum of $31 = \max(31, 22)$ moves

Disjoint pattern databases

- Limitation of traditional PDB: Take max $\rightarrow$ diminish returns on additional PDBs
- **Disjoint pattern databases**: *Count only moves of the pattern tiles, ignoring non-pattern moves.*
- If no tile belongs to more than one pattern, **add** their heuristic values.

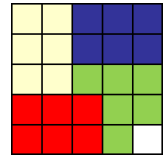
Computing the Heuristic



- 20 moves needed to solve red tiles
- 25 moves needed to solve blue tiles
- Overall heuristic is sum, or $45 = 20 + 25$ moves

Performance

- **15 Puzzle**
  - 1000 $\times$ speedup vs. Manhattan distance
  - IDA* with the two DBs (the 7-tile database contains 58 million entries and the 8-tile database contains 519 million entries) solves 15-puzzles optimally in 30 milliseconds
- **24 Puzzle**
  - 12 million $\times$ speedup vs. Manhattan distance
  - IDA* can solve random instances in 2 days.
  - Requires 4 DBs (each DB has 128 million entries)
  - Without PDBs: 65,000 years



Learning heuristics from experience

- Experience means solving a lot of instances of a problem.
  - E.g., solving lots of 8-puzzles
- Each optimal solution to a problem instance provides examples from which $h(n)$ can be learned
- Learning algorithms
  - Neural nets
  - Decision trees
  - Inductive learning

References

Goodfellow, I., Bengio, Y., and Courville, A. (2016).
Deep learning.
MIT press.

Lê, B. and Tô, V. (2014).
Cở sở trí tuệ nhân tạo.
Nhà xuất bản Khoa học và Kỹ thuật.

Nguyen, T. (2018).
Artificial intelligence slides.
Technical report, HCMC University of Sciences.

Russell, S. and Norvig, P. (2021).
Artificial intelligence: a modern approach.
Pearson Education Limited.