IS-ZC444: ARTIFICIAL INTELLIGENCE

Lecture-06: Problem Solving by Search (contd..)

Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

Sept 26, 2020 FLIPPED (WILP @ BITS-Pilani Jul-Nov 2020)

A* Search (A star)

How to choose next node for expansion?

Use both, path-cost and heuristic $f(n) = g(n) + h(n)$

- Expand nodes in order of increasing f value [1]
- A* is both optimal[2] and complete
- Consistency or monotonicity: if n' is successor of n then

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

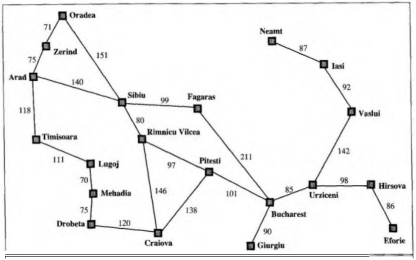
- No other optimal algorithm is guaranteed to expand fewer nodes than A*

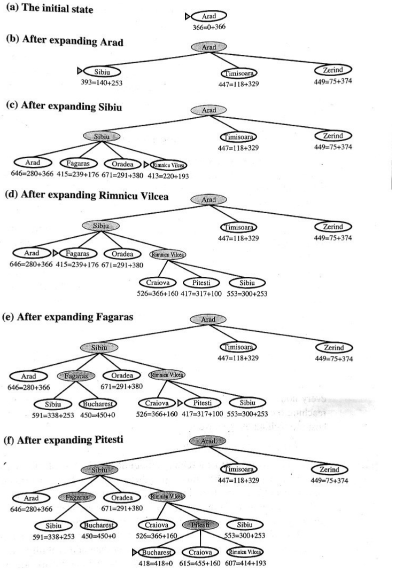
[1] Since $g(n)$ gives the path cost from start node to node n, and $h(n)$ is the estimated cost of the cheapest path from node n to goal, $f(n)$ represents **estimated cost of cheapest solution through** n

[2] Provided h is **admissible** (never overestimates) and **consistent**

A* In Action

Map (Romania)





Heuristic

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

Goal to Travel

Arad → Bucharest

Optimality of A*

- For a consistent h

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

where n' is successor of n

Optimality of A*

- For a consistent h

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

where n' is successor of n

1. Value of f is non-decreasing along the path

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

Optimality of A*

- For a consistent h

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

where n' is successor of n

1. Value of f is non-decreasing along the path

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

Optimality of A*

- For a consistent h

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

where n' is successor of n

1. Value of f is non-decreasing along the path

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

Optimality of A*

- For a consistent h

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

where n' is successor of n

1. Value of f is non-decreasing along the path

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

Optimality of A*

- For a consistent h

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

where n' is successor of n

1. Value of f is non-decreasing along the path

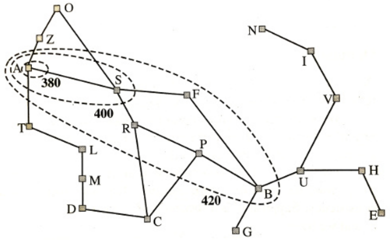
$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

2. A* expands nodes in order of optimal path

If not, then there would be another suitable node n' in frontier (we have chose n instead of n'). Since f is non decreasing function n' have smaller f value then n. Can we neglected n' (a contradiction)

Completeness for A*

Fact that f is nondecreasing, lead to **contours**



- Progress through contour crossing
- Would lead to a contour containing goal state

This provides a basis for completeness of A*

Issue: Expected to have infinitely many nodes in each contour

Error: Absolute $\Delta = (h^* - h)$ Relative $\epsilon = (h^* - h)/h^*$ (Note[3])

[3] Time complexity of A* is $O(b^{\epsilon d})$ where d is solution depth. It is very high and makes A* impractical.

Memory bound A*

IDA*

IDA*: Memory requirements for A* could be reduced using iterative-deepening. Cutoff used is f cost

Recursive best-first search (RBFS)

Use heuristics to choose the best node to explore until *f_limit* reaches[a].

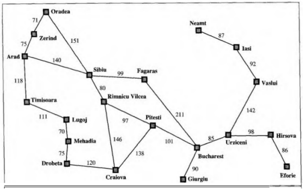
[a] *f_limit* represents f-value of best alternate path from any ancestor

```
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 ← the 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
```

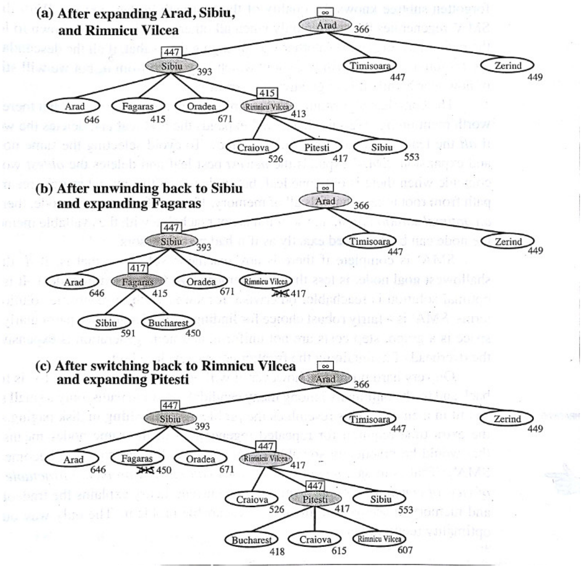
RBFS in Action

Map (Romania)



Heuristic

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



Simplified Memory-bound A* (SMA*)

IDA* and RBFS uses **too little memory** even if more is available

SMA* proceeds as A* until

the memory is full. Then it drops worst leaf node (with highest *f*-value). Value of forgotten node is backed up to its parent.

- To avoid selecting same node for deletion and expansion, SMA* expands newest best leaf and deletes oldest worst leaf.
- Returns best reachable solution
- Complete if depth of shallowest goal is less them memory size
- May faces **thrashing** like scenario so memory limitation can make a problem intractable

Learning to Search Better

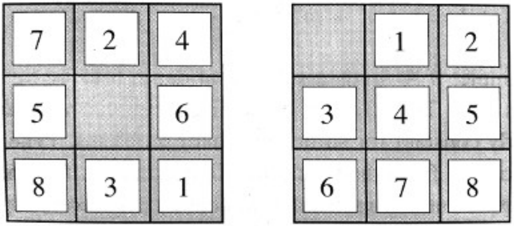
could an agent learn how to search better?

- Yes
- by using **metalevel state space**[4]
- Each state in metalevel state space is a computation step
- Consider learning in this space to avoid exploring unpromising subtrees through experience
- The goal of such learning tasks would be to minimize the total cost of problem solving

[4] each state in metalevel state space captures the internal (computational) state of the program.

Heuristic Function

Consider 8-puzzle problem



Start State Goal State

- Average solution cost is 22 steps
- Branching factor is $\sim$3, so $3^{22} = 3.1 \times 10^{10}$ states in tree
- However, graph search would have $9!/2 = 181,440$ states[5]
- To find shortest solution using A*, we need to find appropriate heuristic functions

[5] 15-puzzle has 10^{13} states

Heuristic Function for 8-puzzle

$\mathbf{h_1}$**:** number of misplaced tiles

$\mathbf{h_2}$**:** sum of the distances of the tiles from their goal position[a]

[a] Since blocks cannot move along diagonal, distance would involve horizontal and vertical moves (city-block or Manhattan distance)

- For our example $h_1 = 8$, $h_2 = 18$
- **Effective branching factor (b^*):** if the algorithm generates N number of nodes and the solution is found at depth d, then

$$N + 1 = 1 + (b^*) + (b^*)^2 + (b^*)^3 + ... + (b^*)^d$$

you get N and d when you run the algorithm. Substitute N and d in above equation to get b^*

Effective branching factor can characterize the quality of heuristic

Heuristic Function for 8-puzzle

Experiment: A* algorithm is run on 1200 random problems with solution length 2 to 24 (100 for each even number).

	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

$h_2(n) \geq h_1(n)$ so h_2 dominates h_1.

From the table it is clear that **h_2 is better than h_1**

It is better to use heuristic function with larger values

Admissible Heuristics from relaxed problem

Consider following relaxations (relaxed 8-puzzle problem)

1. Tile could move anywhere (not only one square towards adjacent empty block). h_1 would give exact number of steps
2. Tile could move one square in any direction, even into an occupied square. h_2 would give exact number of steps

State space graph of relaxed problem is a supergraph of original state space because of removal of restrictions

- Solution in original problem space would work for relaxed setting
- However there could be a better solution in relaxed setting (having lower cost)

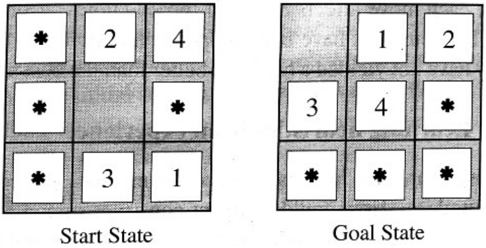
lower cost? means no overestimate on actual cost

A solution in relaxed problem space could be used as heuristic

$$h(n) = max\{h_1(n), h_2(n), .., h_m(n)\}$$

Admissible Heuristics from subproblem

Following is a subproblem (only arrange 1-2-3-4)



- **Pattern database** stores each subproblem and solution cost
- Solution cost could be used as a heuristic
- we could also make database for 5-6-7-8
- But we cannot add the costs of 1-2-3-4 and 5-6-7-8
- **Disjoint pattern database** solution cost is taken as moves involving only 1,2,3,4 in the solution path. Now we can add cost of 1-2-3-4 and 5-6-7-8 database

Learning Heuristics from Experience

- Experience means solving lots of 8-puzzle problem
- Each optimal solution provides examples from which $h(n)$ could be learned
- Using decision tree, neural nets or something else..
- Instead of state n we have to use **feature** say $x_1(n)$ is number of misplaced tiles
- $x_2(n)$ could be number of pairs of adjacent tiles that are not adjacent in the goal state

Heuristic would be obtained by combining $x_1(n)$, $x_2(n)$, ...

One such approach could be to use a liner combination

$$h(n) = c_1 x_1(n) + c_2 x_2(n) + \ldots$$

Thank You!

Thank you very much for your attention!

Queries ?

(Reference[6])

[6] Book - *AIMA*, ch-03, Russell and Norvig.