Informed Search

Tian-Li Yu

Taiwan Evolutionary Intelligence Laboratory (TEIL)
Department of Electrical Engineering
National Taiwan University
tianliyu@ntu.edu.tw

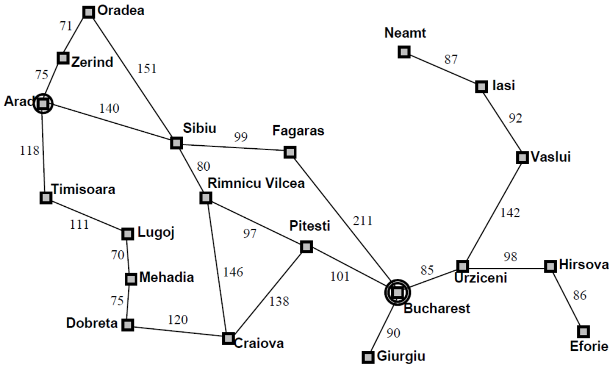
Readings: AIMA Sections 3.5∼3.6

Outline

Best-First Search

- Informed search, a.k.a. heuristic search.
- Idea: Use an evaluation function for each node to estimate the desirability.
  - Expand the most desirable unexpanded node.
- The evaluation function is called heuristic , denoted as $h(n)$.
  - It estimates of cost from node n to the closest goal.
- Special cases:
  - Greedy search, $f(n) = h(n)$.
  - A* search, $f(n) = g(n) + h(n)$.

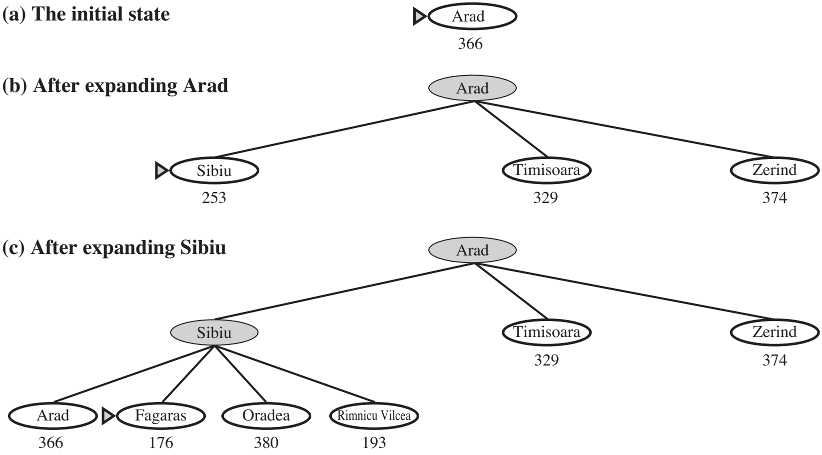
Greedy Search



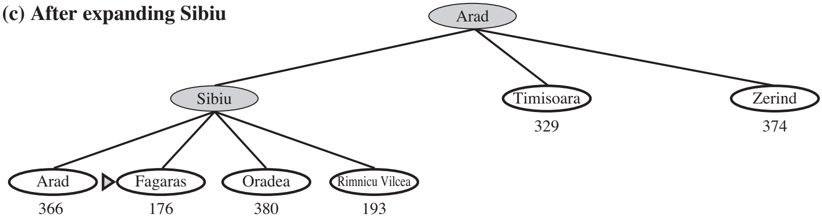
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

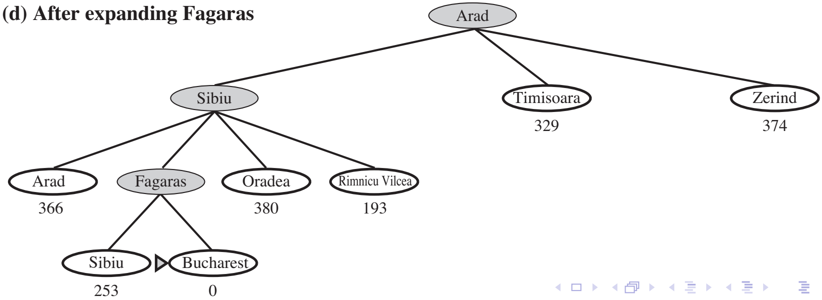
- $h_{SLD}(n)$ = straight-line distance from n to Bucharest.
- Greedy search expands the node that appears to be closest to goal.

Greedy Search on Romania Map



Greedy Search on Romania Map





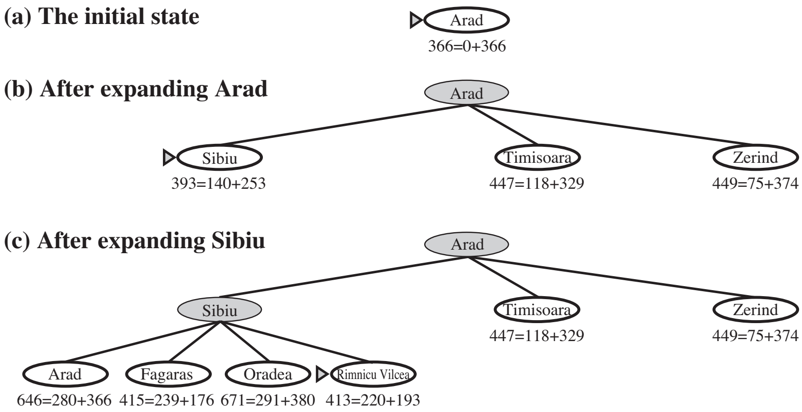
Properties of Greedy Search

- Completeness: No.
  - TREE-SEARCH may get stuck in loops and never reach any goal even in finite state spaces.
    For example, from Iasi to Fagaras, Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow \cdots$
  - GRAPH-SEARCH is complete in finite spaces, but not complete in infinite ones.
- Optimality: No.
- Time complexity: $O(b^m)$, but a good heuristic can give dramatic improvement.
- Space complexity: $O(b^m)$, since it keeps all nodes in memory.

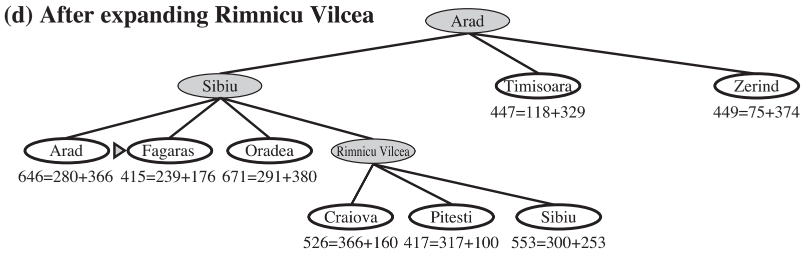
A^* Search

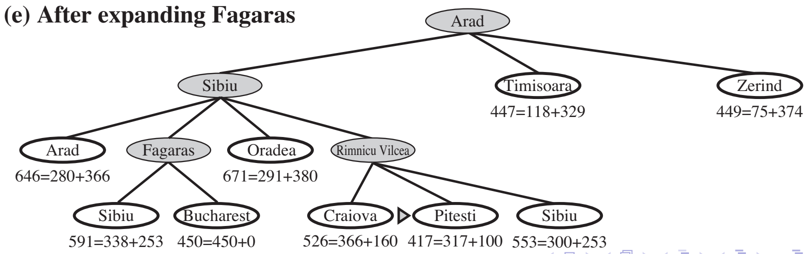
- Idea: Avoid expanding paths that are already expensive.
- Evaluation function: $f(n) = g(n) + h(n)$
  - $g(n)$: cost so far to reach n.
  - $h(n)$: estimated cost to goal from n.
  - $f(n)$: estimated total cost from the starting node to goal through n.
- A^* search combines the advantages of the uniform-cost search and the greedy search.

A* Search on Romania Map

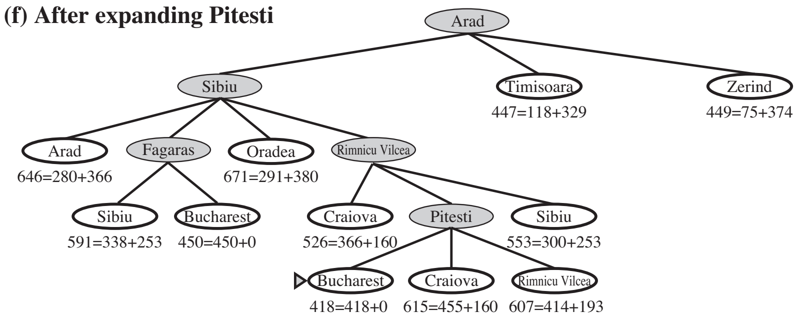


A* Search on Romania Map





A* Search on Romania Map



Properties of A^* Search

- Completeness: Yes. Unless infinite nodes with $f \leq f(goal)$.
- Optimality: Depends on whether h is

Admissible

- Never overestimates the actual cost.
- $\forall n, h(n) \leq h^*(n)$, where h^* is the actual cost.
- e.g., $h_{SLD}(n) \leq h^*(n)$.

Consistent

- A.k.a. monotonicity.
- $\forall$ successor n' of any n generated by any action a, $h(n) \leq c(n, a, n') + h(n')$, where c is the step cost.

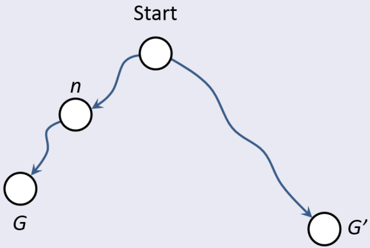
- Time complexity: $O(b^{\epsilon d})$ for constant step costs, where $\epsilon = (h^* - h)/h^*$ (relative error) and d is the solution depth. Effective branch factor is b^ϵ.
- Space complexity: $O(b^d)$, since it keeps all nodes in memory.

Optimality of A*

1. A* is optimal on trees if h is admissible.
2. A* is optimal on graphs if h is admissible and consistent.

Proof of A*'s optimality on trees.

Suppose some suboptimal goal G' has been generated and is in the queue. Let n be an unexpanded node on a shortest path to an optimal goal G.

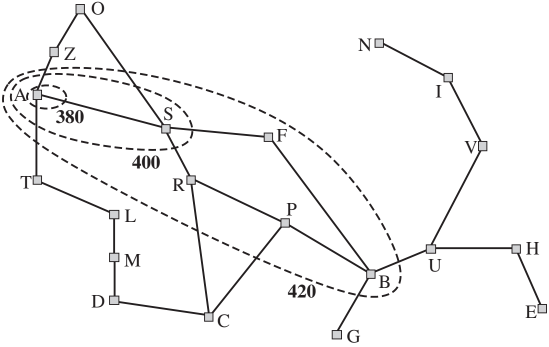


$$
\begin{aligned}
f(G') &= g(G') + h(G') \\
&= g(G') \\
&> g(G) \\
&= g(n) + h^*(n) \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

$\square$

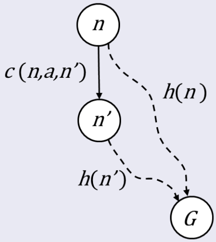
Optimality of A* on Graphs

- Lemma: If $h(n)$ is consistent, the values of f along any path in A* are nondecreasing.
- Gradually adds f-contours of nodes.
- Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$.



Optimality of A* on Graphs

Lemma: if $h(n)$ is consistent, the values of f along any path are nondecreasing.



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

Therefore,

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

$\square$

- Now we see that consistency is actually triangle inequality.

Iterative Deepening A* (IDA*)

- Time complexity is not A*'s biggest drawback.
- A* usually runs out of memory before it reaches goals.
- Iterative deepening A* (IDA*):
  - Use f $(g + h)$ as cutoff instead of the depth.
  - Initial cutoff: $f(s_0) = h(s_0)$
  - Perform DFS on nodes where $f(n) <$ cutoff.
  - Reset cutoff to smallest f of non-expanded nodes.

IDA*(*problem*)

```
currentCutoff = f(s0)
repeat
    result = f-LIMITED-SEARCH(problem, currentCutoff)
    if result ≠ cutoff
        return result
    currentCutoff = smallest f-value of non-expanded nodes.
```

IDA* Traversal on Romania Map

- 1st iteration: $currentCutoff = 366$ (Arad)

  Arad → Sibiu → Timisoara → Zerind
- 2nd iteration: $currentCutoff = 393$ (Sibiu)

  Arad → Sibiu → Arad → Fagaras → Oradea → Rimnicu Vilcea → Timisoara → Zerind
- 3rd iteration: $currentCutoff = 413$ (Rimnicu Vilcea)

  Arad → Sibiu → Arad → Fagaras → Oradea → Rimnicu Vilcea → Craiora → Pitesti → Sibiu → Timisoara → Zerind
- 4th iteration: $currentCutoff = 415$ (Fagaras)

  Arad → Sibiu → Arad → Fagaras → Sibiu → Bucharest → Oradea → Rimnicu Vilcea → Craiora → Pitesti → Sibiu → Timisoara → Zerind
- 5th iteration: $currentCutoff = 417$ (Pitesti)

  . . .

Properties of IDA*

- Completeness and Optimality same as A*.
- Time complexity: $O(b^{\epsilon d})$.
- Space complexity: $O(bd)$.
- Practical for problems with unit step costs.
- What happens if all f-values are different (real-values)?

  The number of iterations can equal the number of nodes whose f-value is less than the cost of an optimal path!

Recursive Best-First Search (RBFS)

- IDA* is problematic when g are real-valued.
- RBFS is a simple recursive algorithm that mimics standard best-first search using only linear space.

$\textsc{Recursive-Best-First-Search}(\textit{problem})$

return $\mathrm{RBFS}(\textit{problem}, \textsc{Make-Node}(\textit{problem.initial\_state}), \infty)$

Recursive Best-First Search (RBFS)

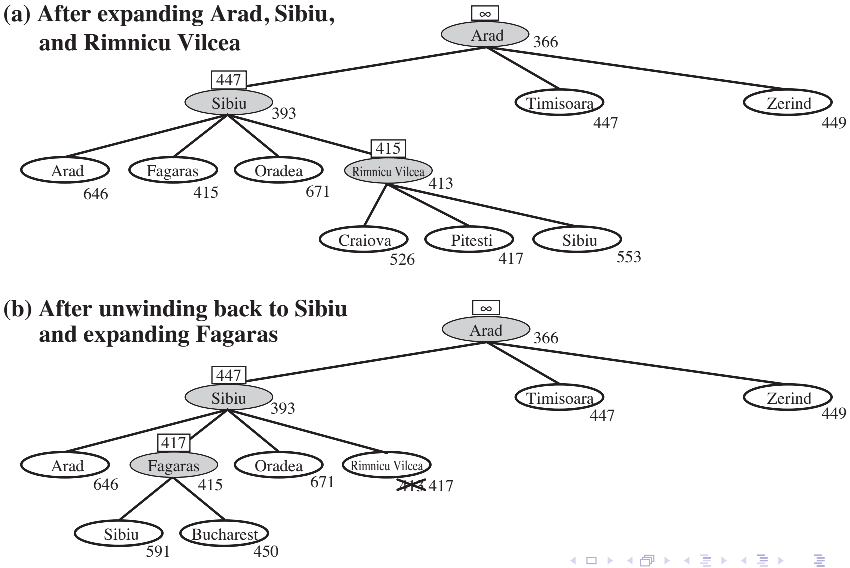
- DFS where each node on the current path remembers the best f-value of any alternative path from its ancestors.
  - Maintains all nodes on current path plus all their siblings ($ancestor(n)$).
- When expanding node n
  - $\forall n' \in children(n)$, compute $f(n')$.
  - if an ancestor n'' has a lower f-value than all n's children, then
    - Assign f-value of the cheapest child to n.
    - Backtrack to n''.
  - Otherwise, proceed as normal.

Recursive Best-First Search (RBFS)

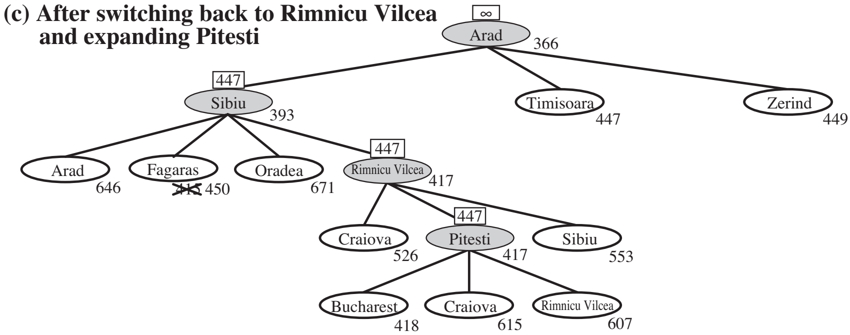
RBFS(*problem*, *node*, *f_limit*)

```
if problem.GOAL-TEST(node.state) return SOLUTION(node)
successors = ϕ
for each action in problem.ACTIONS(node.state) repeat
    add CHILD-NODE(problem, node, action) into successors
if successors is empty return failure, ∞
for each s in successors repeat
    s.f = max(s.g + s.h, node.f)
repeat
    best = the lowest f-value node in successors
    if best.f > f_limit 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 return result
```

RBFS on Romania Map



RBFS on Romania Map



RBFS Traversal on Romania Map

- $f\_limit = \infty$, expanding Arad
  Arad $\rightarrow$ Sibiu $\rightarrow$ Timisoara $\rightarrow$ Zerind
- $f\_limit = 447$ (Timisoara), expanding Sibiu
  Arad $\rightarrow$ Fagaras $\rightarrow$ Oradea $\rightarrow$ Rimnica Vilcea
- $f\_limit = 415$ (Fagaras), expanding Rimnica Vilcea
  Craiova $\rightarrow$ Pitesti $\rightarrow$ Sibiu
- Cutoff occurs. Record $f(RimnicaVilcea)$ as 417. $f\_limit = 417$ (Rimnicu Vilcea), expanding Fagaras
  Sibiu $\rightarrow$ Bucharest
- Cutoff occurs. Record $f(Fagaras)$ as 450. $f\_limit = 447$ (Timisoara), expanding Rimnicu Vilcea (again)
  Craiova $\rightarrow$ Pitesti $\rightarrow$ Sibiu
- $f\_limit = 447$ (Timisoara), expanding Pitesti
  Bucharest $\rightarrow$ Craiova $\rightarrow$ Rimnicu Vilcea
- ...

Properties of RBFS

- Completeness and optimality same as A*.
- Time complexity: Depends on accuracy of h and on how often best path changes.
- Space complexity: $O(bd)$
- Each time RBFS *changes its mind* corresponds to one iteration of IDA*.
- RBFS may need to re-expand forgotten nodes to re-create best-path.

Memory-Bounded Search

- In a sense, both IDA* and RBFS use too little memory.
  - Between iterations, IDA* maintains only one number, the current f-limit ($currentCutoff$).
  - RBFS maintains more, but uses only linear space: if more space were available, it would not benefit from it.
- It seems reasonable to use all the memory available — the more, the better.
- We'd like a memory-bounded version of A*.

Simplified Memory-Bounded A* (SMA*)

- Idea: Run A* as normal until memory is full. Then replace something in memory with newly generated nodes.
- SMA*:
  - When memory is full, drop the worst leaf — node with highest f-value.
  - Like RBFS, SMA* backs up f-value of this forgotten node to it's parent, so we know when to go back to it.
  - If all descends of a node n are forgotten, we don't know which way to go from n, but we know if it's worth re-exploring n.

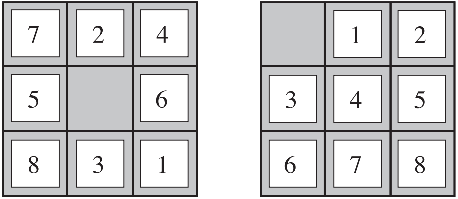
Simplified Memory-Bounded A* (SMA*)

- Problem: What if many nodes have the same f-value?
- Solution: delete the oldest and expand the newest.

- SMA* works as long as there is enough memory for the complete optimal path.
- If not, SMA* needs to switch continuously between candidate paths.
- Causes a similar problem to thrashing in disk paging systems.

Admissible Heuristics for 8-Puzzle

- h_1 = the number of misplaced tiles.
- h_2 = the sum of Manhattan distances of the tiles from their goal positions.



Start State　　Goal State

- $h_1(s_0) = 8$.
- $h_2(s_0) = 3+1+2+2+2+3+3+2 = 18$.

Performance of Heuristic

Definition

For two admissible heuristics h_1 and h_2, h_2 dominates h_1 **iff** $\forall n, h_2(n) \geq h_1(n)$.

Theorem: A* using h_2 never expands more nodes than using h_1.

- Every node with $f(n) < C^*$ is expanded.
- Every node with $h(n) < C^* - g(n)$ is expanded.
- A^* using h_2 expands $n \Rightarrow h_2(n) < C^* - g(n) \Rightarrow h_1(n) \leq h_2(n) < C^* - g(n) \Rightarrow A^*$ using h_1 also expands n.
- $|\{n \mid h_2(n) < C^* - g(n)\}| \leq |\{n \mid h_1(n) < C^* - g(n)\}|$

□

- Given any admissible heuristics h_a and h_b, $h = \max(h_a, h_b)$ is also admissible and dominates h_a and h_b.

Effective Branching Factor

- One way to characterize the quality of a heuristic is effective branching factor b^*.
  - Total number of nodes generated by A*: N
  - Solution depth: d

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

- A well-designed heuristic would have a value of b^* close to 1.

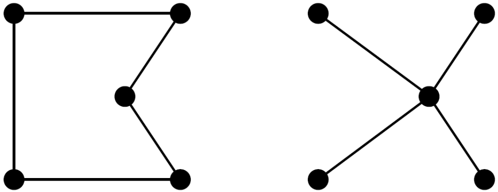
Depth	Nodes generated			Effective branching factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24

Generating Heuristic from Relaxed Problems

- Admissible heuristics can be derived from exact solution cost to a relaxed version of the problem.
- In 8-puzzle, h_1 is derived from that a tile can move to anywhere in one step.
- In 8-puzzle, h_2 is derived from that a tile can move to any adjacent square in one step.
- Key: The optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the original problem.

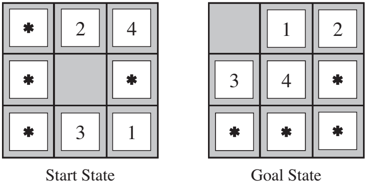
Problem Relaxation Example: TSP

- Traveling salesman problem (TSP).
- Known to be $\mathcal{NP}$-hard.



- Can be relaxed to minimum spanning tree (MST).
- MST cost is never greater than the shortest tour (why? in what condition?).
- Cost can be computed in $O(n^2)$.

Generating Heuristic from Sub-problems



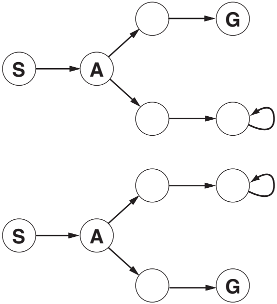
- Admissible heuristic can also be derived from a subproblem.
- Pattern databases store exact solution costs for every possible subproblem instances.
  - For example, every possible position of 1-2-3-4 and the blank.
- Can we use the costs of 1-2-3-4 and 5-6-7-8?
  - Simple addition breaks the admissibility.
  - How about count only those moves involving 1-2-3-4?
  - Then the addition is still admissible.
  - This is the idea behind disjoint pattern databases.

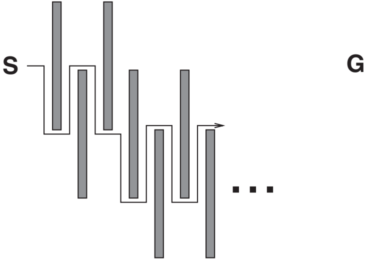
Generating Heuristic from Experience

- Convert a state into the feature domain.
- Feature $f_1(n)$: "number of misplaced tiles".
- Feature $f_2(n)$: "number of pairs of adjacent tiles that are not adjacent in the goal state".
- Both $f_1(goal) = 0$ and $f_2(goal) = 0$.
- $h(n) = c_1 f_1(n) + c_2 f_2(n)$ with $c_1 > 0, c_2 > 0$ (why?).
- We could take randomly generated 8-puzzle and gather statistics to decide constants.
- No guarantee to be admissible or consistent.

Online Search with Unknown Environments

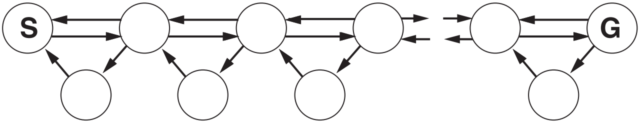
- Competitive ratio $= \dfrac{\text{actual cost}}{\text{minimum cost}}$. We'd like to minimize this.
- If all actions are reversible, online-DFS visits every states exactly twice in the worst case with enough memory.
- If some actions are irreversible, a small (or even finite!) competitive ration can be difficult to achieve.





Search with Limited Memory

- Only one or a few states are stored.
- Single-point hill-climbing gets stuck at a local optimum, causing the competitive ratio to be infinite.
- We may add some random walk (like simulated annealing), but still can be inefficient (exponential in the below example).
- Random walk is complete for finite state spaces.



Learning Real-Time A* (LRTA*)

- $H[s]$: a table of cost estimates indexed by state, initially empty.
- $result[s, a]$: a table indexed by state and action, initially empty.

LRTA*-AGENT(s')

```
if GOAL-TEST(s')
        return stop
if s' is a new state (not in H)
        H[s'] = h(s')
if s ≠ NULL
        result[s, a] = s'
        H[s] = min_{b ∈ ACTIONS(s)} LRTA*-COST(s, b, result[s, b], H)
a = an action b in ACTIONS(s') that minimizes
        LRTA*-COST(s', b, result[s', b], H)
s = s'
return a
```

Learning Real-Time A* (LRTA*)

- LRTA* keeps updating $H[s]$.
- LRTA* always chooses the apparently best action.
- Optimism under uncertainty: If an action has never tried in a state, LRTA* assumes the least possible cost — $h(s)$. This encourages exploration.

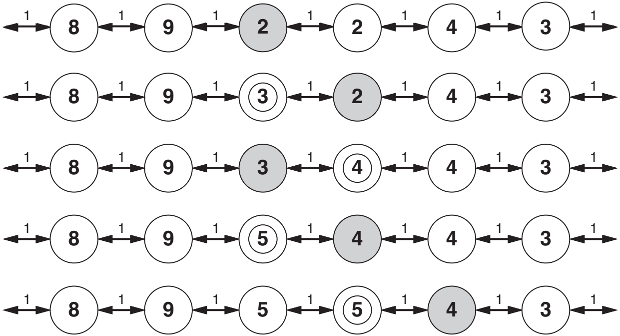
LRTA*-COST(s, a, s', H)

if s' is undefined **return** $h(s)$

else return $c(s, a, s') + H[s']$

Learning Real-Time A* (LRTA*)

- Unlike A*, LRTA* is NOT complete for infinite state spaces.
- With n states, LRTA* guarantees to find optimum within $O(n^2)$ steps, but usually much faster.
- Shaded: agent's location, circle: $H[s]$ updated.



Summary

- Heuristic functions estimate costs of shortest paths.
- Good heuristics can dramatically reduce search cost.
- Greedy best-first search expands lowest h.
  - In general not complete nor optimal.
- A* search expands lowest $g + h$.
  - Optimal when h is admissible (and consistent).
- Memory limitation is an important issue to heuristic search. Search with forgetting and re-expanding are the keys, but still suffers from different conditions.
- A more efficient heuristic can be generated from several admissible heuristics.
- Admissible heuristics can be derived from relaxed problems, subproblems, and experience.
- On-line search with limited memory can easily fail; LRTA* works well if memory is enough.