CS1504 Artificial Intelligence

A^* Search

C. Aravindan
<AravindanC@ssn.edu.in>

Professor of Computing
SSN College of Engineering

August 12, 2024

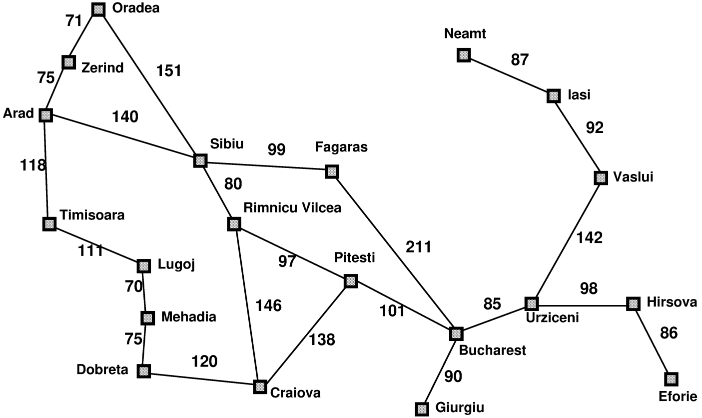
Evaluating a state

- Sometimes, it may be possible to design an evaluation function $f(s)$ that evaluates the "badness" (to be minimized) or "goodness" (to be maximized) of a state s
- In such cases, the most desirable state may be chosen from the working set
- Working set is maintained as a priority queue based on the evaluation function f
- Obviously, the quality of search depends on the evaluation function f

Heuristics

- Usually, such an evaluation function $f(s)$ is designed based on some heuristics $h(s)$ — estimation of cost of reaching a goal state from state s
- For example, can you think of a heuristics for the route finding problem in a map? — Straight line distance (SLD) from the current city to the destination city
- Heuristics should be an easy function to compute!
- $h(s^*)$ should be 0 for any goal state s^*

Example: Route finding problem



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

Find route from Arad to Bucharest

Example: Sliding puzzle

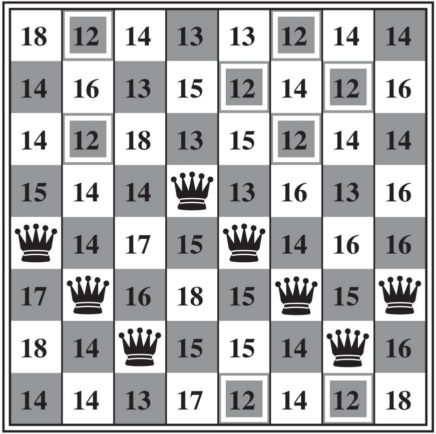
- Consider the sliding puzzle, such as

3	2	7
5	8	
1	4	6

- What may be a good heuristics for this state space?
- $h_1(s)$: Number of misplaced tiles — for the above state $h_1(s) = 7$
- $h_2(s)$: Sum of Manhattan distances of tiles from their goal positions — for the above state $h_2(s) = 2 + 0 + 2 + 2 + 1 + 1 + 4 + 1 = 13$
- Which heuristics is better? — an estimate which is closer to the actual is always better!
- We say that h_2 dominates h_1
- An admissible heuristics is one which does not overestimate — in our example, both $h_1(x)$ and $h_2(x)$ are admissible

Example: *n*-queens problem

- What may a good heuristics for *n*-queens problem?
- Cost estimate: Number of pairs of queens that are attacking each other, either directly or indirectly



Best-first greedy search

- As a simple strategy, we may let the evaluation function f to be the same as the heuristics function h
- Nodes in the working set (priority queue) are organized based on estimated cost and the one with the least cost is given preference
- This is a generalization of greedy design strategy, that you have learnt in the previous semester

Best-First Greedy: Complexities

- Is Greedy strategy complete? — No! — not in general
- Is it optimal? — No!
- Time complexity? — $O(b^m)$
- Space complexity? — $O(b^m)$

Questions?

Admissible search

- In the greedy strategy, we have used only the “future” cost estimation to choose the next best state

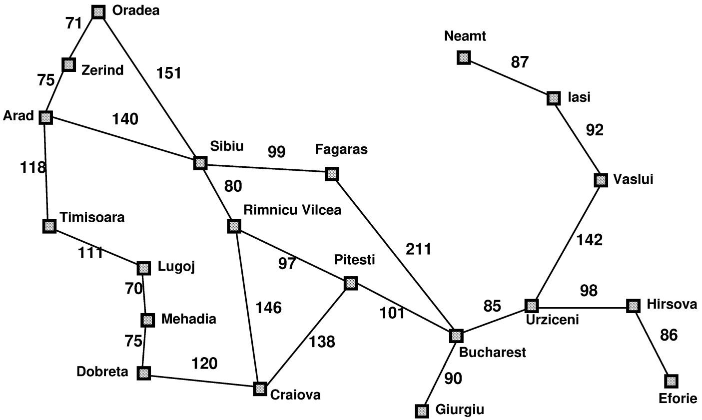
Admissible search

- In the greedy strategy, we have used only the “future” cost estimation to choose the next best state
- It may be prudent to consider evaluating a state s by the sum of cost of reaching that state s from the start state and the estimated cost of reaching a goal state from s

Admissible search

- In the greedy strategy, we have used only the "future" cost estimation to choose the next best state
- It may be prudent to consider evaluating a state s by the sum of cost of reaching that state s from the start state and the estimated cost of reaching a goal state from s
- In other words $f(s) = g(s) + h(s)$

Best-first A^* search

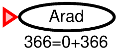


Straight−line distance to Bucharest

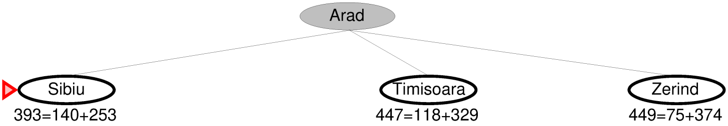
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Find the best route from Arad to Bucharest

A^* search

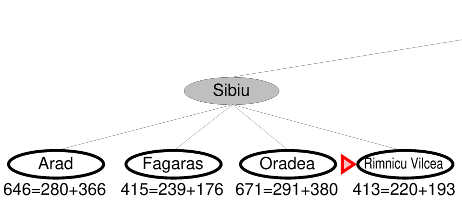


A^* search



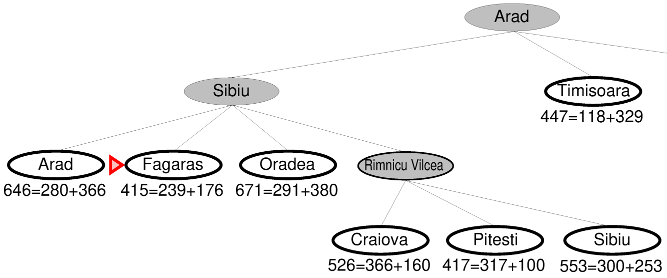
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374

A^* search



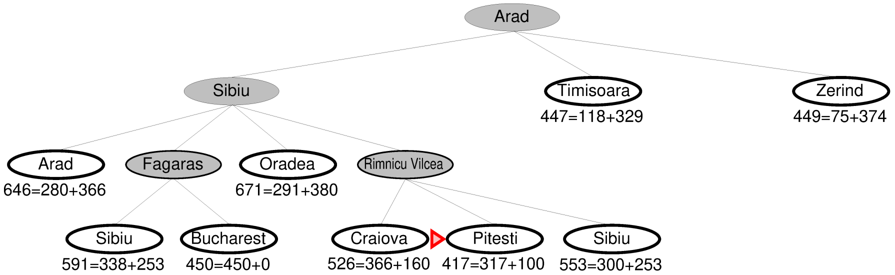
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193

A^* search

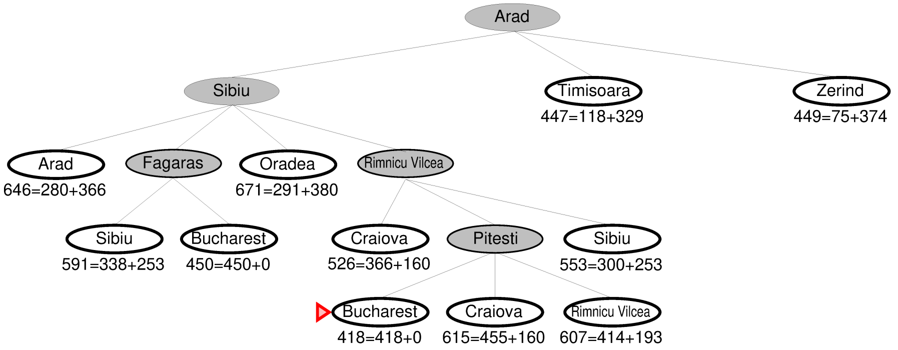


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A^* search

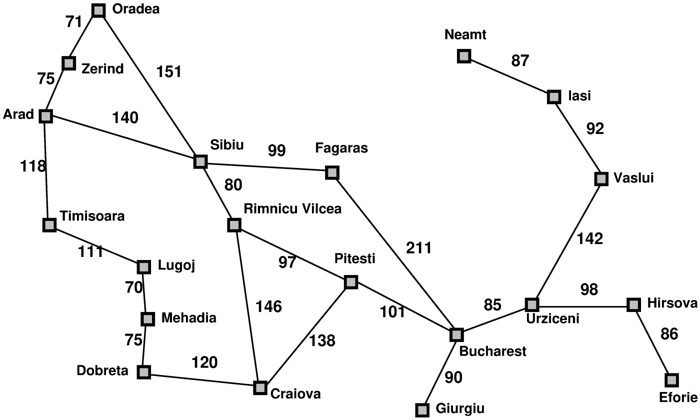

Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A^* search



Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

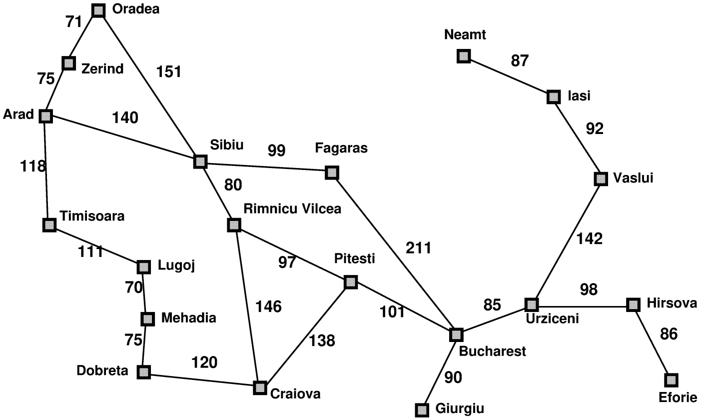
A^* search



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

A^* search

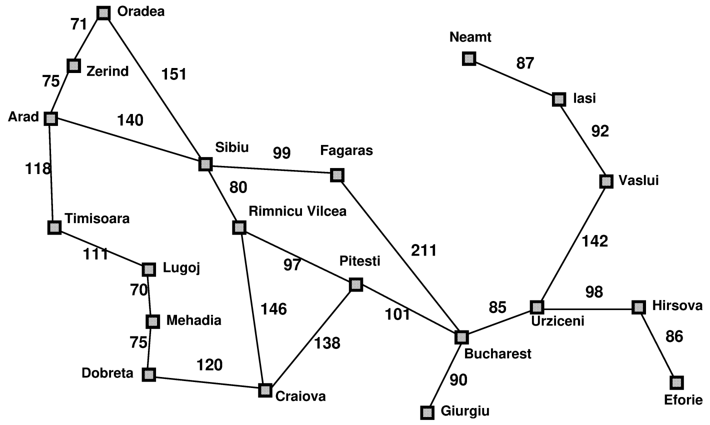


Straight−line distance to Bucharest

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

Solution found: Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest

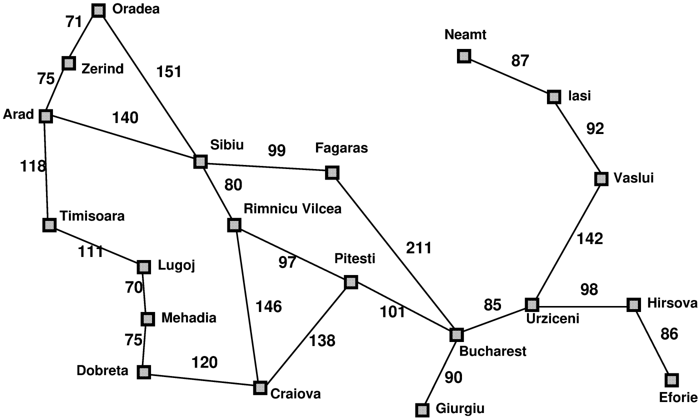
A^* search



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

Solution found: Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest with total cost 418

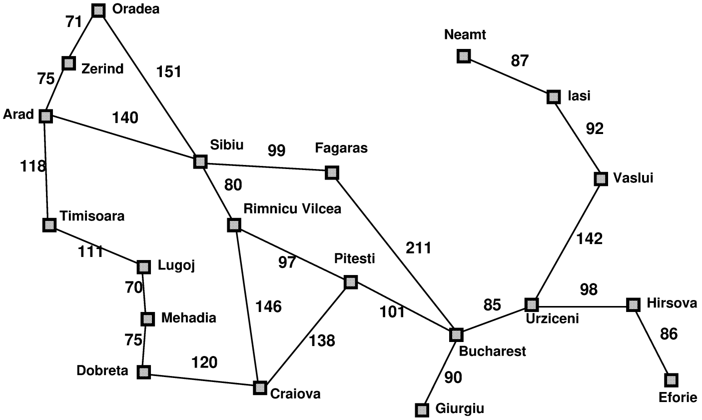


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Solution found: Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest with total cost 418

Does A^* work?

A^* search



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

Solution found: Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest with total cost 418

Does A^* work? — Yes!

Admissible Heuristics

- A heuristic h is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the **true** cost to reach the goal state from n.

Admissible Heuristics

- A heuristic h is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the **true** cost to reach the goal state from n.
- In other words, $f(n) = g(n) + h(n) \leq C^*$, where C^* is the optimal path cost

Admissible Heuristics

- A heuristic h is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the **true** cost to reach the goal state from n.
- In other words, $f(n) = g(n) + h(n) \leq C^*$, where C^* is the optimal path cost
- An admissible heuristic **never overestimates** the cost to reach the goal, i.e., it is always **optimistic**

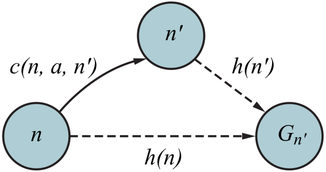
Admissible Heuristics

- A heuristic h is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the **true** cost to reach the goal state from n.
- In other words, $f(n) = g(n) + h(n) \leq C^*$, where C^* is the optimal path cost
- An admissible heuristic **never overestimates** the cost to reach the goal, i.e., it is always **optimistic**
- The SLD heuristics and the two heuristics for sliding puzzle problem are examples of admissible heuristics

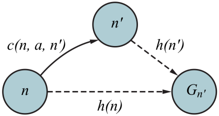
Consistent Heuristics

- A heuristics is consistent if for every node n, $h(n) \leq c(n, a, n') + h(n')$, where n' is a successor of n generated by some action a

- A heuristics is consistent if for every node n, $h(n) \leq c(n, a, n') + h(n')$, where n' is a successor of n generated by some action a



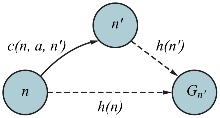
Consistent Heuristics



- When h is consistent, we can infer the following

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

Consistent Heuristics

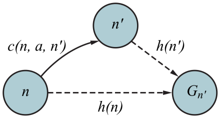


- When h is consistent, we can infer the following

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

- That means, evaluation function f is monotonic — it is non-decreasing along any path

Consistent Heuristics



- When h is consistent, we can infer the following

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

- That means, evaluation function f is monotonic — it is non-decreasing along any path
- Every consistent heuristics is also admissible

A^* is optimal

Theorem

When h is an admissible heuristics, A^* using TREE-SEARCH is optimal

A^* is optimal

Theorem

When h is an admissible heuristics, A^* using TREE-SEARCH is optimal

- Suppose a sub-optimal goal G_2 is generated and is in the frontier

A^* is optimal

Theorem

When h is an admissible heuristics, A^* using TREE-SEARCH is optimal

- Suppose a sub-optimal goal G_2 is generated and is in the frontier
- Let G be an optimal goal. There must be some node n in the frontier that leads to G

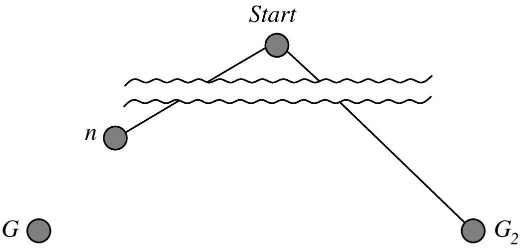
A^* is optimal

Theorem

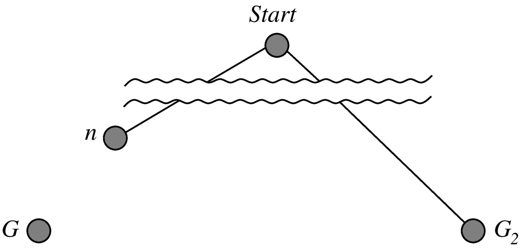
When h is an admissible heuristics, A^* using TREE-SEARCH is optimal

- Suppose a sub-optimal goal G_2 is generated and is in the frontier
- Let G be an optimal goal. There must be some node n in the frontier that leads to G
- We need to prove that A^* selects n ahead of G_2 from the frontier

A^* is optimal

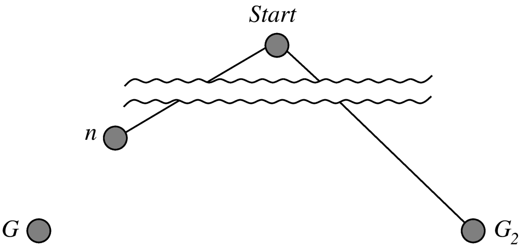


A^* is optimal



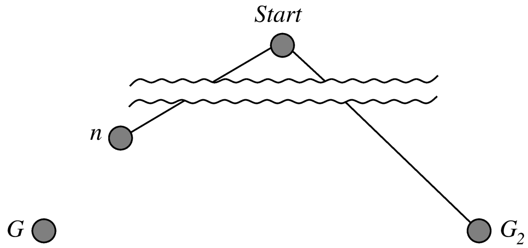
- $f(G_2) = g(G_2)$

A^* is optimal



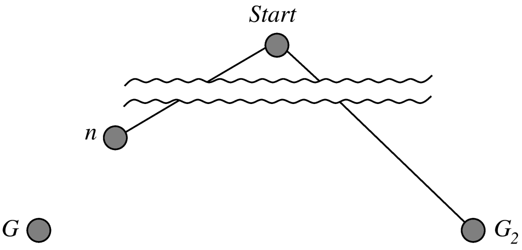
- $f(G_2) = g(G_2)$
- $f(G) = g(G)$

A^* is optimal



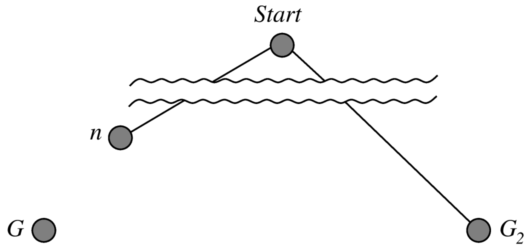
- $f(G_2) = g(G_2)$
- $f(G) = g(G)$
- $g(G_2) > g(G)$

A^* is optimal



- $f(G_2) = g(G_2)$
- $f(G) = g(G)$
- $g(G_2) > g(G)$
- $f(G_2) > f(G)$

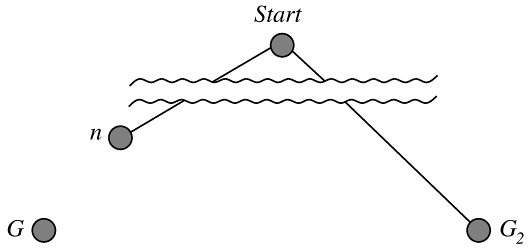
A^* is optimal



- $f(G_2) = g(G_2)$
- $f(G) = g(G)$
- $g(G_2) > g(G)$
- $f(G_2) > f(G)$

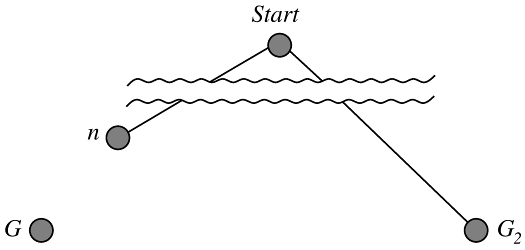
- $g(n) + h(n) \leq C^*$

A^* is optimal



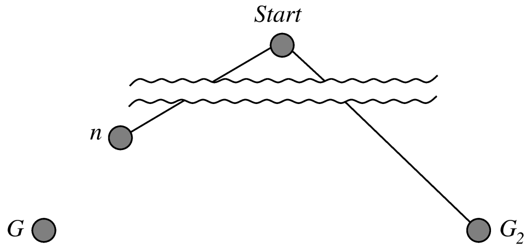
- $f(G_2) = g(G_2)$
- $f(G) = g(G)$
- $g(G_2) > g(G)$
- $f(G_2) > f(G)$
- $g(n) + h(n) \leq C^*$
- $f(n) \leq g(G)$

A^* is optimal



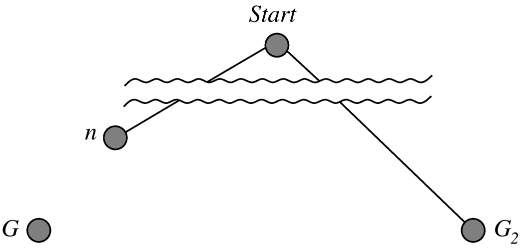
- $f(G_2) = g(G_2)$
- $f(G) = g(G)$
- $g(G_2) > g(G)$
- $f(G_2) > f(G)$
- $g(n) + h(n) \leq C^*$
- $f(n) \leq g(G)$
- $f(n) \leq f(G)$

A^* is optimal



- $f(G_2) = g(G_2)$
- $f(G) = g(G)$
- $g(G_2) > g(G)$
- $f(G_2) > f(G)$
- $g(n) + h(n) \leq C^*$
- $f(n) \leq g(G)$
- $f(n) \leq f(G)$
- $f(n) < f(G_2)$

A^* is optimal



- $f(G_2) = g(G_2)$
- $f(G) = g(G)$
- $g(G_2) > g(G)$
- $f(G_2) > f(G)$

Hence A^* selects n ahead of G_2

- $g(n) + h(n) \leq C^*$
- $f(n) \leq g(G)$
- $f(n) \leq f(G)$
- $f(n) < f(G_2)$

A^* is optimal

Theorem

When h is a consistent heuristics, A^* using GRAPH-SEARCH is optimal

A^* is optimal

Theorem

When h is a consistent heuristics, A^* using GRAPH-SEARCH is optimal

- Suppose n' is a successor of n. From consistency property, we know that $f(n') \geq f(n)$

A^* is optimal

Theorem

When h is a consistent heuristics, A^* using GRAPH-SEARCH is optimal

- Suppose n' is a successor of n. From consistency property, we know that $f(n') \geq f(n)$
- Suppose A^* selects n for expansion, then optimal path to n has been found

A^* is optimal

Theorem

When h is a consistent heuristics, A^* using GRAPH-SEARCH is optimal

- Suppose n' is a successor of n. From consistency property, we know that $f(n') \geq f(n)$
- Suppose A^* selects n for expansion, then optimal path to n has been found — otherwise, there must be another node m in the frontier that leads to n, and $f(m) < f(n)$

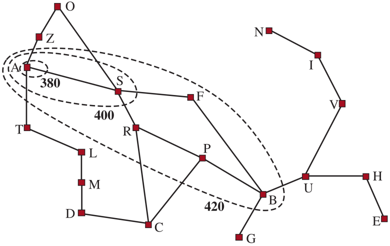
A^* is optimal

Theorem

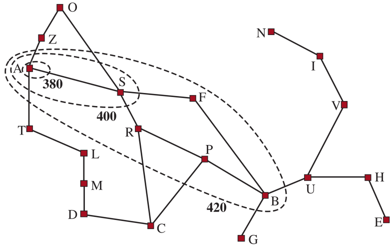
When h is a consistent heuristics, A^* using GRAPH-SEARCH is optimal

- Suppose n' is a successor of n. From consistency property, we know that $f(n') \geq f(n)$
- Suppose A^* selects n for expansion, then optimal path to n has been found — otherwise, there must be another node m in the frontier that leads to n, and $f(m) < f(n)$
- It follows from these two observations that the first goal state found by A^* must be optimal

Properties of A^* search

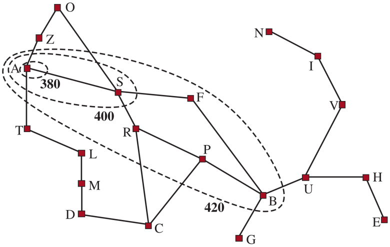


Properties of A^* search



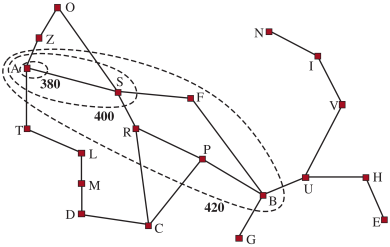
- If C^* is the optimal cost, then
  - A^* expands all nodes with $f(n) \leq C^*$
  - A^* might expand some nodes where $f(n) = C^*$
  - A^* expands no nodes with $f(n) > C^*$

Properties of A^* search



- If C^* is the optimal cost, then
  - A^* expands all nodes with $f(n) \leq C^*$
  - A^* might expand some nodes where $f(n) = C^*$
  - A^* expands no nodes with $f(n) > C^*$
- This implies that A^* is complete if there are only finitely many nodes with f-cost less than C^*

Properties of A^* search



- If C^* is the optimal cost, then
  - A^* expands all nodes with $f(n) \leq C^*$
  - A^* might expand some nodes where $f(n) = C^*$
  - A^* expands no nodes with $f(n) > C^*$
- This implies that A^* is complete if there are only finitely many nodes with f-cost less than C^*
- Since no nodes with $f(n) > C^*$ need to be expanded, pruning is implicit

Properties of A^* search

- A^* is optimal with admissible heuristics

Properties of A^* search

- A^* is optimal with admissible heuristics
- A^* is, in general, complete — that is, it finds a solution, if exists

Properties of A^* search

- A^* is optimal with admissible heuristics
- A^* is, in general, complete — that is, it finds a solution, if exists
- A^* is optimally efficient — no other optimal algorithm is guaranteed to expand fewer nodes than A^*

Properties of A^* search

- A^* is optimal with admissible heuristics
- A^* is, in general, complete — that is, it finds a solution, if exists
- A^* is optimally efficient — no other optimal algorithm is guaranteed to expand fewer nodes than A^*
- Time complexity?

Properties of A^* search

- A^* is optimal with admissible heuristics
- A^* is, in general, complete — that is, it finds a solution, if exists
- A^* is optimally efficient — no other optimal algorithm is guaranteed to expand fewer nodes than A^*
- Time complexity? — $O(b^{\Delta})$ in general, where the absolute error $\Delta = h^* - h$ (where h^* is the actual cost) —

Properties of A^* search

- A^* is optimal with admissible heuristics
- A^* is, in general, complete — that is, it finds a solution, if exists
- A^* is optimally efficient — no other optimal algorithm is guaranteed to expand fewer nodes than A^*
- Time complexity? — $O(b^{\Delta})$ in general, where the absolute error $\Delta = h^* - h$ (where h^* is the actual cost) — it may also be expressed in terms of relative error, $\epsilon = (h^* - h)/h^*$, as $O(b^{\epsilon d})$ —

Properties of A^* search

- A^* is optimal with admissible heuristics
- A^* is, in general, complete — that is, it finds a solution, if exists
- A^* is optimally efficient — no other optimal algorithm is guaranteed to expand fewer nodes than A^*
- Time complexity? — $O(b^\Delta)$ in general, where the absolute error $\Delta = h^* - h$ (where h^* is the actual cost) — it may also be expressed in terms of relative error, $\epsilon = (h^* - h)/h^*$, as $O(b^{\epsilon d})$ — however, it is fast, in practice, when good heuristics are used

Properties of A^* search

- A^* is optimal with admissible heuristics
- A^* is, in general, complete — that is, it finds a solution, if exists
- A^* is optimally efficient — no other optimal algorithm is guaranteed to expand fewer nodes than A^*
- Time complexity? — $O(b^{\Delta})$ in general, where the absolute error $\Delta = h^* - h$ (where h^* is the actual cost) — it may also be expressed in terms of relative error, $\epsilon = (h^* - h)/h^*$, as $O(b^{\epsilon d})$ — however, it is fast, in practice, when good heuristics are used
- Space complexity?

Properties of A^* search

- A^* is optimal with admissible heuristics
- A^* is, in general, complete — that is, it finds a solution, if exists
- A^* is optimally efficient — no other optimal algorithm is guaranteed to expand fewer nodes than A^*
- Time complexity? — $O(b^{\Delta})$ in general, where the absolute error $\Delta = h^* - h$ (where h^* is the actual cost) — it may also be expressed in terms of relative error, $\epsilon = (h^* - h)/h^*$, as $O(b^{\epsilon d})$ — however, it is fast, in practice, when good heuristics are used
- Space complexity? — same as that of time complexity (every generated node needs to be kept in memory) — worst-case space requirement is similar to that of breadth-first search

Properties of A^* search

- A^* is optimal with admissible heuristics
- A^* is, in general, complete — that is, it finds a solution, if exists
- A^* is optimally efficient — no other optimal algorithm is guaranteed to expand fewer nodes than A^*
- Time complexity? — $O(b^{\Delta})$ in general, where the absolute error $\Delta = h^* - h$ (where h^* is the actual cost) — it may also be expressed in terms of relative error, $\epsilon = (h^* - h)/h^*$, as $O(b^{\epsilon d})$ — however, it is fast, in practice, when good heuristics are used
- Space complexity? — same as that of time complexity (every generated node needs to be kept in memory) — worst-case space requirement is similar to that of breadth-first search — memory bounded strategies have been proposed to overcome this

Satisficing Search

- For some problems, performance of A^* may not be OK, as it may expand exponentially many nodes before finding a solution

Satisficing Search

- For some problems, performance of A^* may not be OK, as it may expand exponentially many nodes before finding a solution
- In such cases, satisficing search may be considered, where we may trade-off on the quality of the solution

Satisficing Search

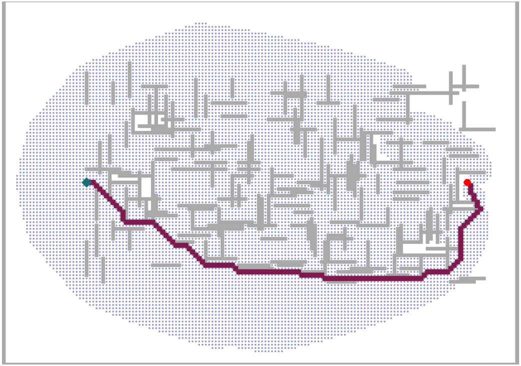
- For some problems, performance of A^* may not be OK, as it may expand exponentially many nodes before finding a solution
- In such cases, satisficing search may be considered, where we may trade-off on the quality of the solution
- Basically, we may permit inadmissible heuristics that may overestimate and take the risk of missing the optimal solution

Satisficing Search

- For some problems, performance of A^* may not be OK, as it may expand exponentially many nodes before finding a solution
- In such cases, satisficing search may be considered, where we may trade-off on the quality of the solution
- Basically, we may permit inadmissible heuristics that may overestimate and take the risk of missing the optimal solution
- For example, we may multiply the straight line distance heuristics by what is known as detour index (engineers normally use a detour index in the range of 1.2 and 1.6)

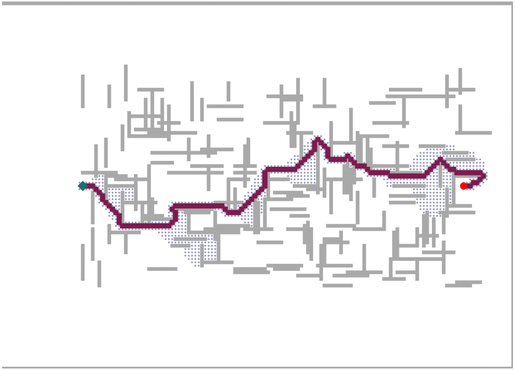
Satisficing Search

- For some problems, performance of A^* may not be OK, as it may expand exponentially many nodes before finding a solution
- In such cases, satisficing search may be considered, where we may trade-off on the quality of the solution
- Basically, we may permit inadmissible heuristics that may overestimate and take the risk of missing the optimal solution
- For example, we may multiply the straight line distance heuristics by what is known as detour index (engineers normally use a detour index in the range of 1.2 and 1.6)
- This idea can be generalized as weighted A^* search, that uses an evaluation function $f(n) = g(n) + W \times h(n)$ for some $W > 1$



- A^* explores several nodes to find an optimal solution in a grid world problem

Satisficing Search — Illustration



- *Weighted* A^*, with $W = 2$, explores comparatively lesser nodes to find a solution that is 5% costlier, in the same instance of grid world problem

Bounded sub-optimal search

- Weighted A^* is an example of a bounded sub-optimal search, where the cost of a solution found is bounded by $W \times C^*$, where C^* is the optimal cost
- Sometimes, it may be required to opt for a bounded cost search, whereby we try to find quickly a solution whose cost does not exceed some prefixed cost C
- If nothing works, sometimes we may for unbounded-cost search whereby we try to quickly find some solution.

Questions?