COMP30030: Introduction to Artificial Intelligence

Neil Hurley

School of Computer Science
University College Dublin
neil.hurley@ucd.ie

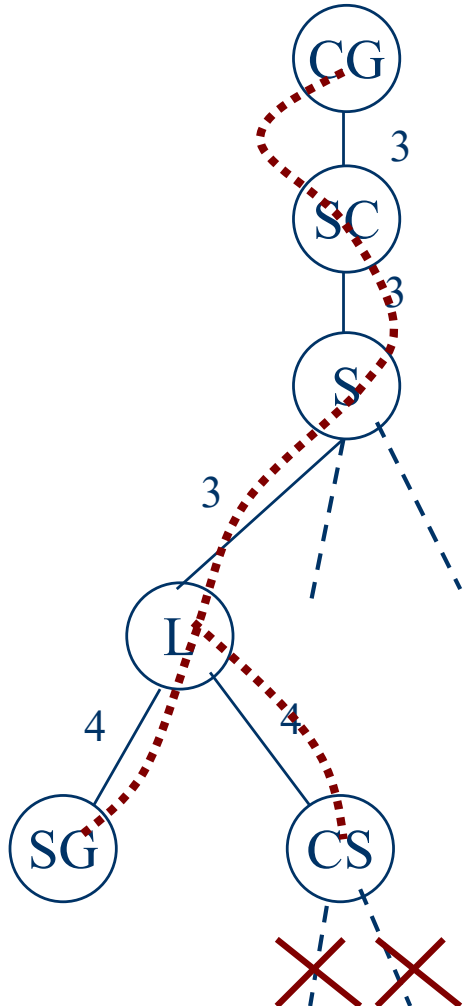
September 27, 2018

Lowest-cost First Search

- Greedy best search relies on a heuristic estimate of the distance to the goal.
- Another approach is to carry out best-first search but instead prioritise for low g(n) — the (known) distance that node *n* is from the initial state, along the current path.
- Dijkstra's shortest path algorithm applies this strategy: at each iteration, the path that is chosen to expand is the one whose distance from the initial state is a minimum.

Branch & Bound Search

- Suppose you want to find the shortest path from CG to SG and you have searched part of the UCD search tree.
  - You have found one solution of length 13 (CG-SG-S-L-SG).
  - Not sure whether this is the shortest one available you continue your search and develop a path CG-SG-S-L-CS which also has length 13.
  - But now immediately you can stop following this path because if it indeed reaches SG then without doubt it will be longer than the previous solution. So you are free to go on and extend another shorter partial path.



Branch and Bound Search

- Keep track of a lower bound and upper bound on solution cost at each node
  - lower bound: $LB(n) = g(n)$ or possibly $LB(n) = g(n) + h(n)$, when $h(n)$ is admissible)
  - upper bound: UB = g(n'), where n' is the best solution found so far.
    - if no solution has been found yet, set the upper bound to ∞.
  - When a node n is selected for expansion:
    - if $LB(n) \geq UB$, remove n from open list without expanding it – this is called "pruning the search tree"
    - else expand n, adding all of its neighbours to the open list
- Often a LIFO queue is used, so that the search is carried out in depth-first manner – "depth-first branch and bound".

A* Search

- A* search uses both the path cost, $g(n)$, and heuristic evaluation $h(n)$
  - $g(n)$ is the exact cost of the current path up to the current node n.
  - $h(n)$ estimates of the cost from node n to the goal along the current path.
- A* operates a priority queue with priority given by $f(n) = g(n) + h(n)$, where nodes are processed in increasing order of $f(n)$.

$f(n)$ estimates the *total* path cost of going from a start node to a goal via n.

Analysis of A*

Assume that costs are strictly positive:

- Completeness: yes
- Time complexity: $O(b^d)$ where d is the depth at which the goal is found
  - note that the heuristic could be completely non-informative and thus the edge costs could all be the same, meaning that A* would do the same thing as BFS
- Space complexity: $O(b^d)$
  - Like BFS, A* maintains a fringe that grows with the size of the tree.
- Optimality: yes

Optimality of A* I

If A* returns a solution, that solution is guaranteed to be optimal, as long as

- the branching factor is finite
- edge costs are strictly positive : $d(n, n') > \epsilon > 0$ for all nodes n and their children n'.
- h(n) is admissible – an underestimate of the length of the shortest path from n to a goal node.

Optimality of A* II

Theorem

If A selects a path p, p is the shortest (i.e., lowest-cost) path.*

- Assume for contradiction that some other path p' is actually the shortest path to a goal
- Consider the moment just before p is chosen from the frontier. Some part of path p' will also be on the frontier; let's call this partial path p''.
- Because p was expanded before p'', $f(p) \leq f(p'')$.
- Because p is a goal, $h(p) = 0$. Thus
- $g(p) \leq g(p'') + h(p'')$.
- Because h is admissible, $g(p'') + h(p'') \leq g(p')$ for any path p' to a goal that extends p''.
- Thus $g(p) \leq g(p')$ for any other path p' to a goal. This contradicts our assumption that p' is the shortest path.

Optimal Efficiency of A* I

In fact, we can prove something even stronger about A*. Given the particular heuristic $h()$, no search algorithm can do better.

- Optimal Efficiency: Among all *optimal* algorithms that start from the same start node and use the same heuristic $h()$

  A* expands the minimal number of nodes.

- To be more precise, in case A* is unlucky in how it breaks ties: define optimal efficiency as expanding the minimal number of nodes n for which $f(n) \neq f^*$, where f^* is the cost of the shortest path.

Optimal Efficiency of A* II

Theorem

A is optimally efficient.*

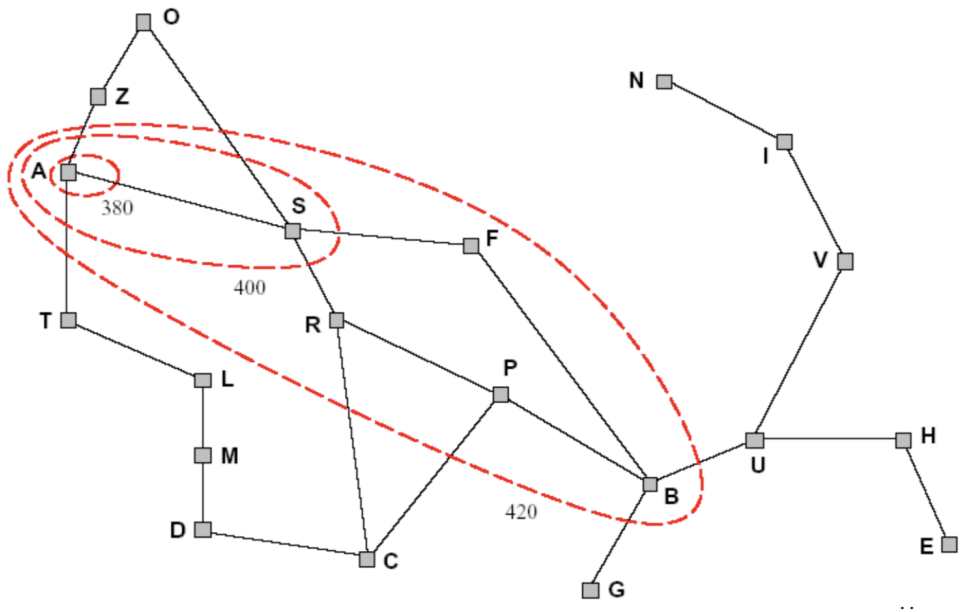
- Let f^* be the cost of the shortest path to a goal. Consider any algorithm A' which has the same start node as A*, uses the same heuristic and fails to expand some node n' expanded by A* for which $g(n') + h(n') < f^*$.
- Assume that A' is optimal and seek a contradiction.
- Consider a different search problem which is identical to the original and on which h returns the same estimate for each node, except that n' has a child node n'' which is a goal node, and the true cost of the path to n'' is $f(n')$.
- That is, the edge from n' to n'' has a cost of $h(n')$: the heuristic is exactly right about the cost of getting from n' to a goal.

Optimal Efficiency of A* III

- A' would behave identically on this new problem: the only difference between the new problem and the original problem is beyond node n' which A' does not expand.
- The cost of the path to n'' is lower than cost of the path found by A'.
- This violates our assumption that A' is optimal.

A*: search in increasing $f()$ value

- A* expands nodes in order of increasing $f()$ value.
- It gradually adds "f-contours" – compare with BFS – where contour i has all nodes with $f = f_i$ and $f_{i+1} > f_i$.



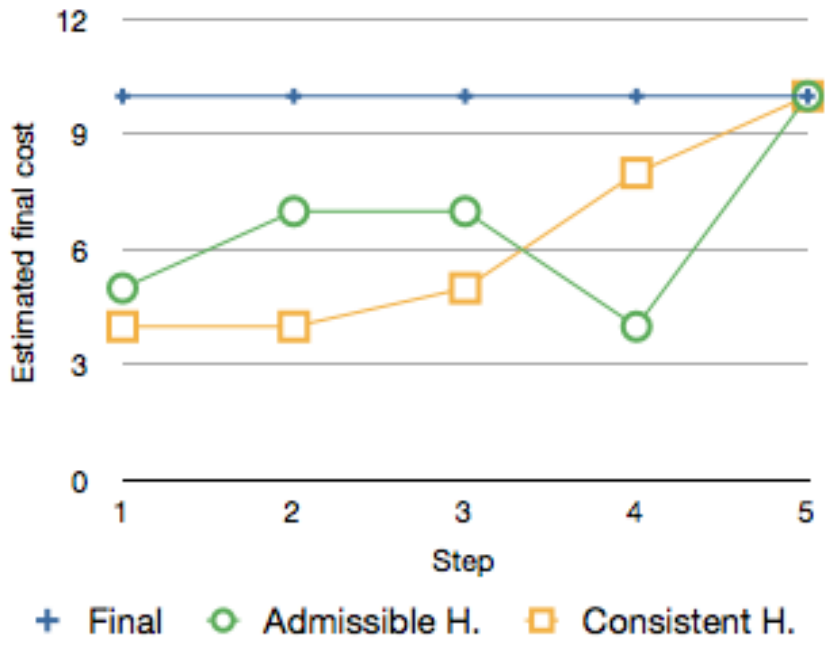
A* with Graph-search I

- Recall that for a graph-search as opposed to a tree-search, we maintain a closed-list of expanded nodes, so that re-expansion of nodes can be avoided.
- If A* is to remain optimal for graph-search, we need to ensure that each time a node is expanded, the path cost by which it was reached is the lowest possible.
- Otherwise, the node would need to be re-expanded
- The requires a stronger condition on the heuristic function – that it is consistent or monotone. This means that for any node n with child n', linked by an edge of cost $d(n, n')$, the heuristic should satisfy

$$. \ h(n) \leq d(n, n') + h(n')$$

A* with Graph-search II

- The estimated distance to the goal at node n is no greater than the estimated distance at a child node n' plus the cost of moving to the child.
- In other words the estimated cost to reach the goal, $f(p)$, is monotonically non-decreasing along any path.



(From Johannes Simon https://commons.wikimedia.org/w/index.php?curid=12198271)

A* with Graph-search III

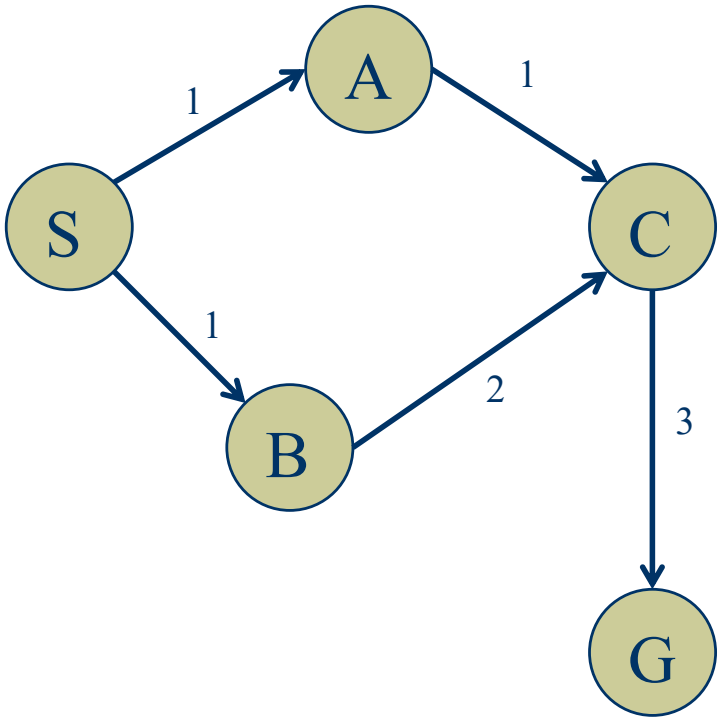
- Consistency implies admissibility:
  - Consider a path $n_1 \rightarrow n_2 \rightarrow \cdots \rightarrow n_m = G$. Then for a consistent $h()$:

$$
\begin{aligned}
h(n_1) &\leq d(n_1, n_2) + h(n_2) \\
&\leq d(n_1, n_2) + d(n_2, n_3) + h(n_3) \\
&\leq d(n_1, n_2) + d(n_2, n_3) + \cdots + d(n_{m-1}, n_m) + h(n_m) \\
&= .g(n_m) + h(n_m) = g(n_m), \text{ since } h(n_m) = 0 \text{ for goal node}
\end{aligned}
$$

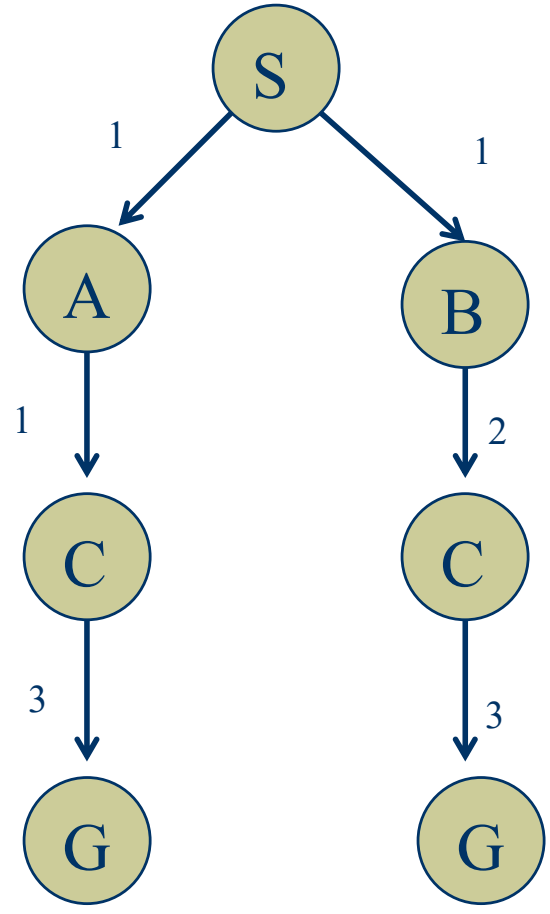
Hence the estimate to the goal at a node n_1 is a lower bound on the length of the path to the goal — $h()$ is admissible.

A* Graph Search Gone Wrong?

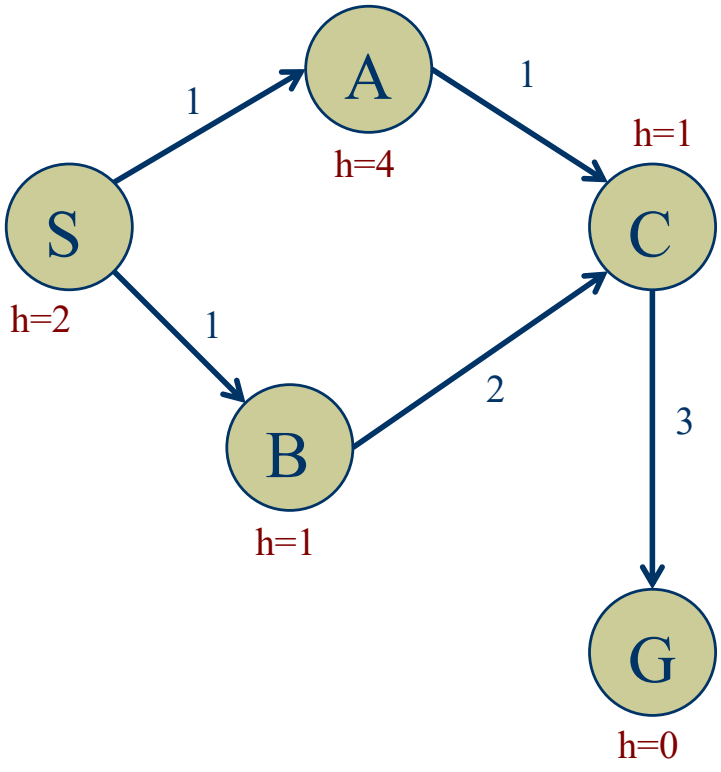
Search Graph:

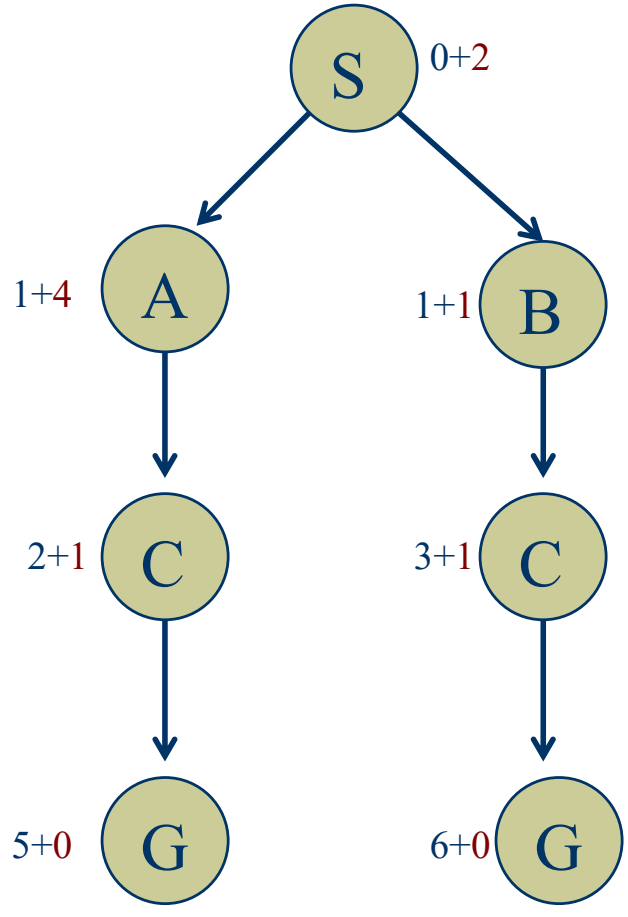


Search Tree:



A* Graph Search Gone Wrong?



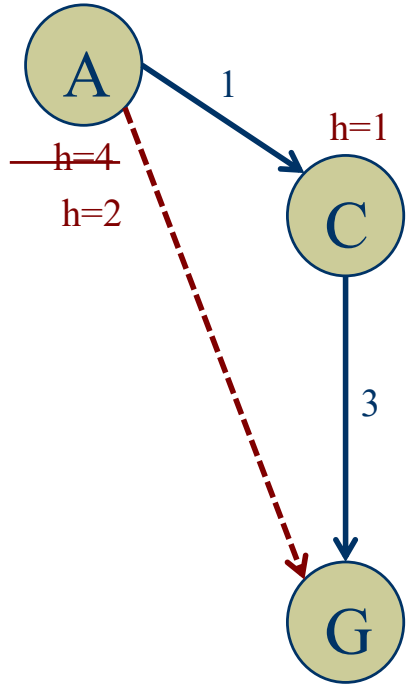


Consistency of Heuristics

- **Main Idea:** estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    h(A) ≤ actual cost from A to G
  - Consistency: heuristic cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)



- **Consequences of consistency:**
  - The f value along a path never decreases

    h(A) ≤ cost(A to C) + h(C)
  - A* graph search is optimal

Inconsistent 8-puzzle heuristic

- **Instead of the sum of the Manhattan distances of all tiles from their goal position , we might decide to just take the distance of tiles {1,2,3} from their goal position as h(n).**
- **This is clearly still admissible, since it is less than the full Manhattan distance.**
- **Then again, we could take the distance of tiles {4,5,6} from their goal position.**
- **Now, suppose, at each node, we random choose between these two sets.**
- **No matter what the choice, we have an admissible heuristic. But now, there is possibly no relation between the distance at the parent at that at the child.**

A*, the ideal search strategy?

- If we have a bad heuristic function, A* is still exponential.
- In fact the condition for A* to find the solution in less than exponential time is that the error on h is: $|h(n) - h^*(n)| \leq O(\log(h^*(n)))$ where h^* is the true distance to the goal.
- The closer h is to h^*, the faster we get there.
- And, since A* behaves like BFS, we have the same exponential memory problem.

IDA*

- Analogously to uninformed search, we have the following dilemma:
  - A* is optimal and will make most efficient use of the heuristic, but has the memory characteristics of BFS
  - Greedy best first and depth-first B&B have better memory characteristics but are not guaranteed to be optimal.
- One solution: iterative deepening.
- Rather than use a depth-bound, we instead use an $f()$ value cutoff – IDA*
- At each iteration, the cutoff is the smallest $f(n)$ that exceeded the cutoff on the previous iteration.
- Can work well if $f()$ values are discrete (e.g. integer values), but what if the cost is a real-valued number? The cutoff could grow by a arbitrarily small ϵ on each iteration, leading to long iterations.

Heuristics in the 8-puzzle problem I

- 181,440 states
- Search without checking repeats is long, as the average branching factor is 3 and average minimal path-to-solution is length 22.
- Checking for repeats is easy (just need an array of 181,440 `bool`) and quick.
- Though the 8-puzzle is simple, when we move to 4×4 (15-puzzle) things already get difficult.
  10^{13} states
  Branching factor 3.5
  Average path to solution 52 moves.

Heuristics in the 8-puzzle problem II

- Compare $h_1() =$ tiles out of place with $h_2() =$ Manhattan distance
- Manhattan is a *tighter* lower bound on the true distance to goal.
- We can empirically evaluate the impact of the tighter bound by examining the average number of nodes N that are expanded by each of these methods.
- If you expand N nodes to find the solution at level d, then the *effective branching factor* is a b such that:

$$N = 1 + b + \cdots + b^d$$

- The effective branching factor b can be used as a measure of the quality of a heuristic.

Heuristics in the 8-puzzle problem III

- For $h_1()$, $b \sim 1.5$, for $h_2() \sim 1.25$
- For solutions of length 22, this corresponds to $h_1()$ expanding around 18,000 nodes, while $h_2()$ expands only around $1,200$ nodes.
- Good heuristics are critical to good performance.
- Recent successes in Machine Learning for playing games such as Go (from Google's Deepmind research team), rely on *learning* good heuristics through game playing.