COMP261 Lecture 7

A* Search

A* search

- Can we do better than Dijkstra's algorithm?
- Yes!
  - want to explore more promising paths, not just shortest so far.
  - ⇒ need to change the priority on the fringe:
    - choose node on the fringe that is the most promising:

Total path length will be
cost from start to this node (we know this cost)
+ cost from this node to goal (we can only estimate this cost)

A* uses a heuristic estimate of total path length:
costSoFar + heuristic estimate
heuristic estimate must be guaranteed to be no more than the real cost.

A* algorithm (fast version)

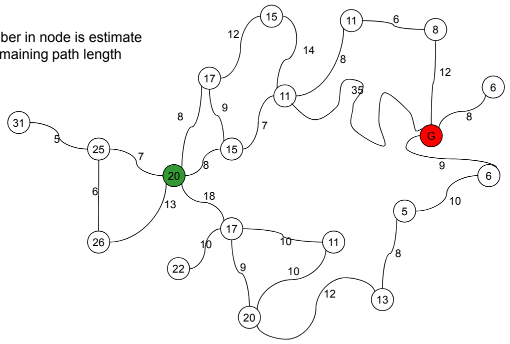
Minor change on Dijkstra's algorithm:

Initialise: for all nodes *visited* ← false, *pathFrom* ← null
enqueue(⟨*start, null,* 0, estimate(*start, goal*) ⟩, *fringe*),
Repeat until *fringe* is empty:
  ⟨*node, from, costToHere, totalCostToGoal* ⟩ ← dequeue(*fringe*)
  If not *node.visited* **then**
    node.visited ←true, *node.pathFrom*←*from*, *node.cost*←*costToHere*
    If *node* = *goal* **then** exit
    for each *edge* to *neigh* out of *node*
      if not *neigh.visited* **then**
        costToNeigh ← *costToHere* + *edge.weight*
        estTotal ← *costToNeigh* + estimate(*neighbour, goal*)
        fringe.enqueue(⟨*neighbour, node, costToNeigh, estTotal*⟩)

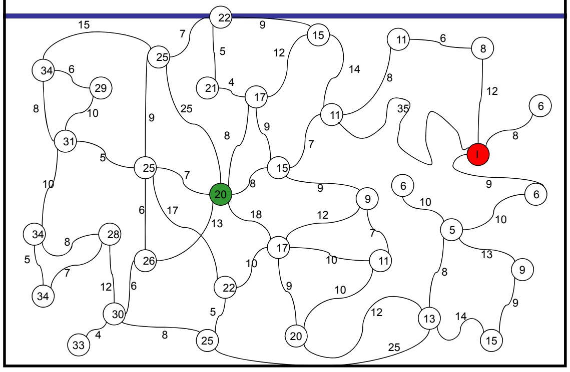
- fringe = priority queue, ordered by total cost to Goal
- estimate(node, goal) must be admissible and consistent.

A* example.

Number in node is estimate of remaining path length

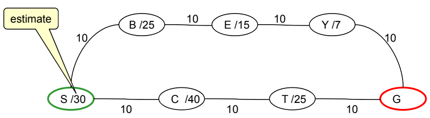


An example.



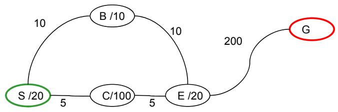
A* heuristic

- A heuristic estimate of the remaining path is admissible if it always underestimates the remaining cost
- If it is not admissible, it may not find the shortest path:



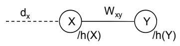
A* heuristic: monotonic/consistent

- Admissible is not enough for the fast version of A*:
  When visit a node, must be the best path to a node



- To be able to commit to visited nodes:
  - heuristic must get more accurate as you go along a path

$d_x + h(X) \;<=\; d_x + W_{xy} + h(Y)$



- Consistent heuristic: $h(X) - h(Y) <= W_{xy}$

NB! See excellent discussion on this in Wikipedia article on A* search. What matters is the *priority* on the queue which is the *total path length*. Its essential that once you "greedily" commit to the node, you can never later find a node with a smaller priority (total path length) value – this what makes fast version of A* work.

A* heuristic

- Consistent heuristics can be hard to find
  (Euclidean distance to goal *is* consistent)

- If the estimate is admissible, but is not consistent,
  then:
  ⇒ cannot commit to a node when we take it off the queue
  ⇒ may need to revisit nodes
  ⇒ no point in the visited set

- Do we have to keep searching all possible paths?
- When is it not worth putting a neighbour on the fringe?
  If this path to the neighbour is worse than the best to neighbour so far.
  If this path to the neighbour is worse than the best path to the goal so far

A* algorithm (slow version)

No *<u>visited set</u>!* But must store length of best path to each node

```
Initialise: bestToGoal ← ∞ , for all nodes node.pathLength ← ∞
fringe.enqueue(⟨start, null, 0, estimate(start, goal ) ⟩)
Repeat until fringe is empty:
    ⟨node, from, costToHere, totalCostToGoal ⟩ ⟩ ← fringe.dequeue
    If costToHere < node.cost then                  // shorter route to node
        node.pathFrom← from,   node.cost ← costToHere
        If node = goal  then exit                   // found shortest route to goal
        for each edge from node to neighbour
            toNeigh ← costToHere + edge.weight

            if toNeigh < neigh.cost  then
                estTotal ← toNeigh + estimate(neighbour, goal )
                if estTotal < bestToGoal then
                fringe.enqueue(⟨neighbour, node, toNeigh, estTotal ⟩)
                if neighbour = goal  then bestToGoal ← toNeigh
```

More on A*

- Very general search strategy
  - not just paths: eg: search for optimal loading of a truck
  - any optimisation problem where build up a solution as a series of steps.
  - Works with implicit graphs (AI search problems)
    (but need to store nodes you have seen)
  - need a lower bound estimate of full cost [admissible]
  - need a consistent estimate to use the fast version.
  - If cannot guarantee consistency, may be exponential

- Heuristics
  - Issue: how to find a good, admissible, consistent heuristic