COMP3308/COMP3608, Lecture 4a
ARTIFICIAL INTELLIGENCE

More on A*

Russell and Norvig, ch. 4

Review

- **A search strategy defines the order of node expansion from the fringe**
  - **Fringe list (open list): nodes reached but not yet expanded**
  - **Expanded list (closed list): nodes that have been expanded**
- **A* selects the node with the smallest f-cost for expansion**
  f(n)=g(n)+h(n)
  g(n)* = cost so far to reach *n
  ***h(n)* = estimated cost from n to goal**
  ***f(n)* = estimated total cost of path through *n* to goal**

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do  // nodes in fringe ordered in increasing order based on eval. function (f for A*)
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)// 1st select node for expansion, then check if it is a goal
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    fringe ← INSERTALL(EXPAND(node, problem), fringe)
```


A Closer Look at `Tree-Search`

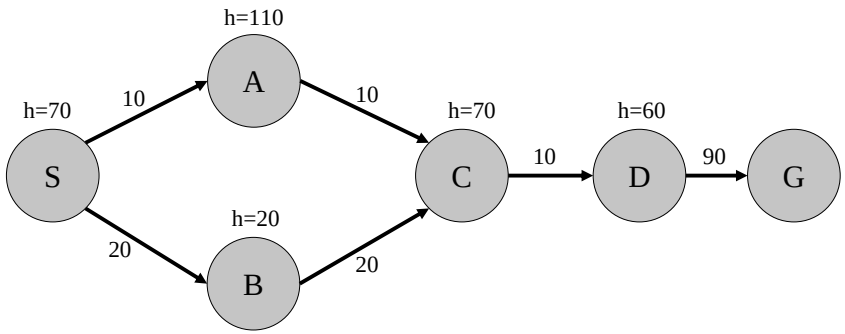
```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do  // nodes in fringe ordered in increasing order based on eval. function (f for A*)
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)// 1st select node for expansion, then check if it is a goal
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

1) Select node *n* from fringe
2) If (n=goal) stop
else expand(n)

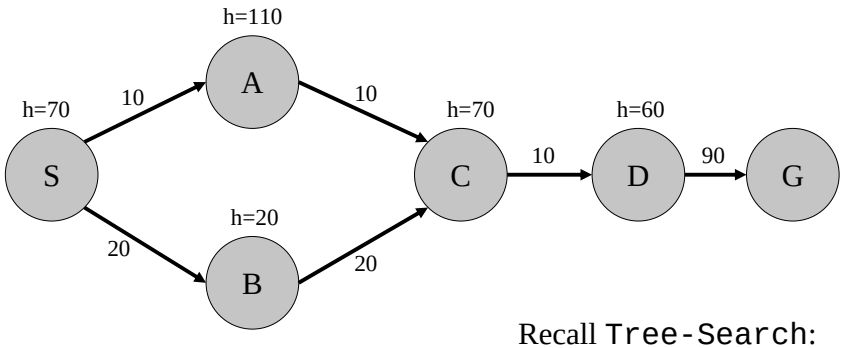
- **Does `Tree-Search` discard repeated nodes, i.e. revisited states?**

Searching Graphs

- **Searching graphs is more complex that searching trees**
  - **trees – 1 path to each node**
  - **graphs – there may be more than 1 path to a node**
- **Consider this example, let's run A* with `Tree-Search`**



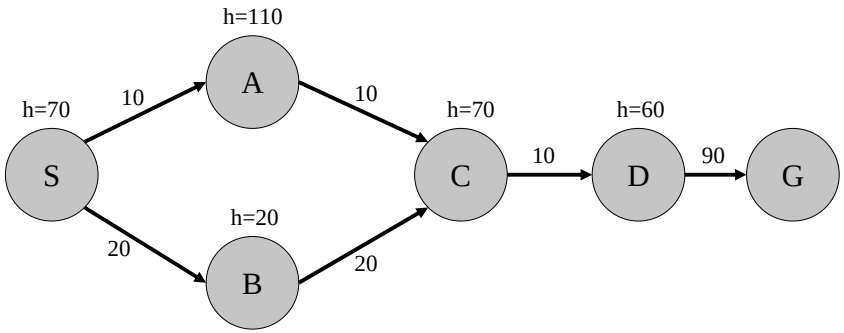
A* Example



Recall `Tree-Search`:
1) Select *n* from fringe
2)If (n=goal) stop
else expand(n)

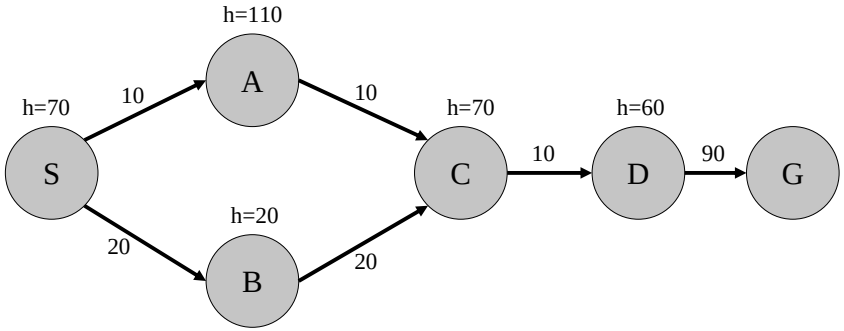
Fringe (OPEN): (S,70)
Expanded (CLOSED):

A* Example



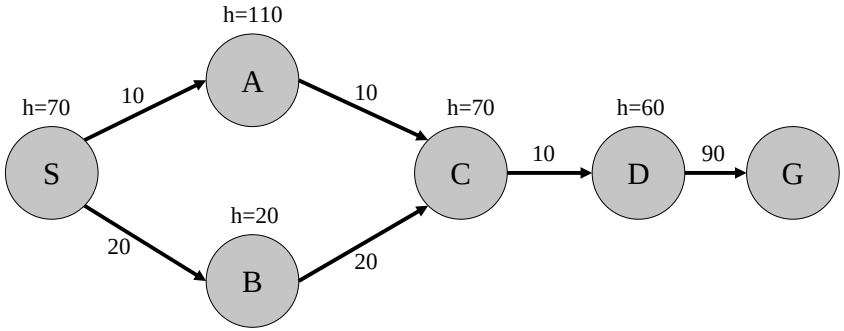
Fringe (OPEN): (A,120), (B,40)
Expanded (CLOSED): (S,70)

A* Example



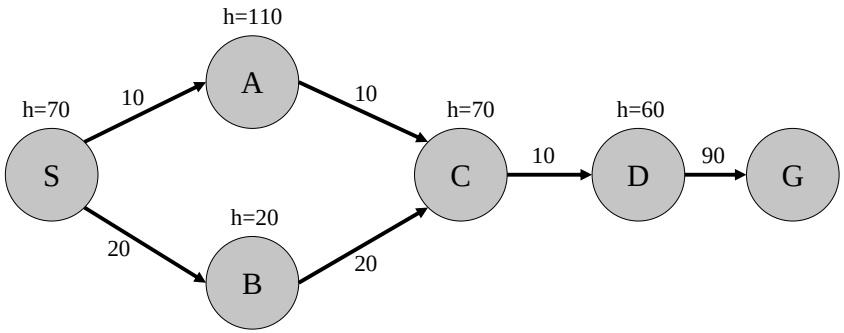
Fringe (OPEN): (A,120), (C,110)
Expanded (CLOSED): (S,70), (B,40)

A* Example



Fringe (OPEN): (A,120), (D,110)
Expanded (CLOSED):(S,70), (B,40), (C,110)

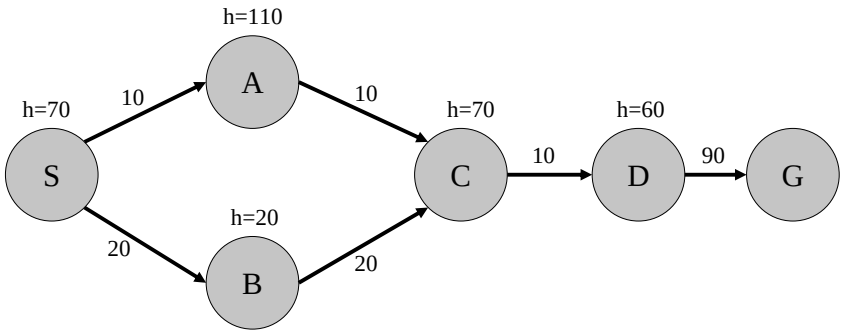
A* Example



Fringe (OPEN): (A,120), (G,140)

Expanded (CLOSED):(S,70), (B,40), (C,110), (D,110)

A* Example

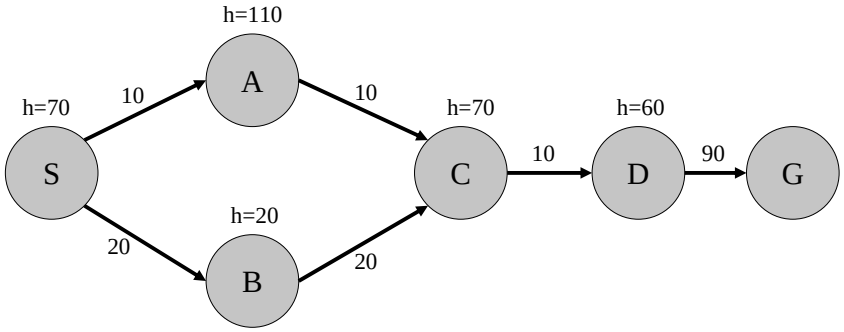


Fringe (OPEN): (G,140), (C,90)

Node C will be selected for expansion, i.e. state C will be revisited

Expanded (CLOSED):(S,70), (B,40), (C,110), (D,110), (A,120)

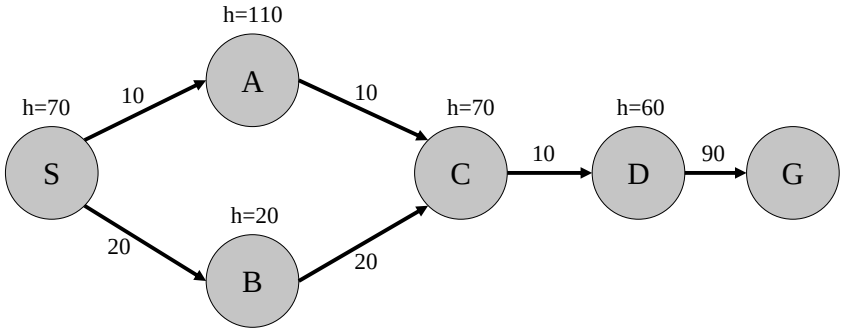
A* Example



Fringe (OPEN): (G,140), (D, 90)

Expanded (CLOSED):(S,70), (B,40), (C,110), (D,110), (A,120), (C,90)

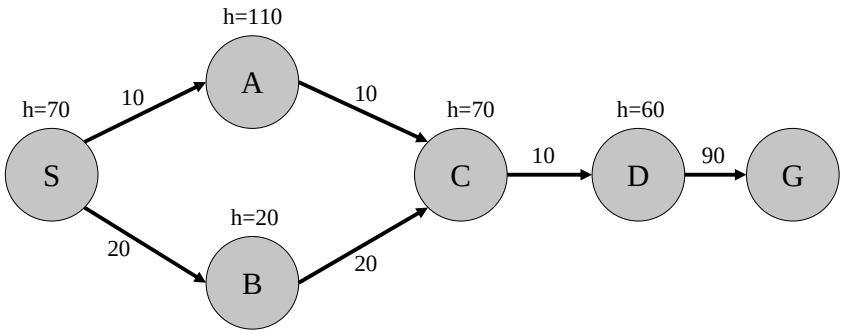
A* Example



Fringe (OPEN): (G,140), (G,120)

Expanded (CLOSED):(S,70), (B,40), (C,110), (D,110), (A,120), (C,90), (D,90)

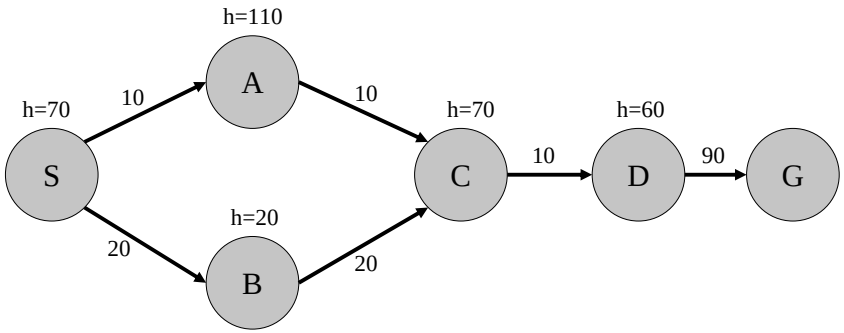
A* Example



Fringe (OPEN): (G,140)

Expanded (CLOSED):(S,70), (B,40), (C,110), (D,110), (A,120), (C,90), (D,90), (G,120) stop – goal node selected for expansion

A* Example



Path found: SACDG, cost: 120

Admissible heuristic

- **A heuristic *h(n)* is admissible if for every node *n*:**
  - ***h(n)* ≤*h*(n)* where *h*(n)* is the <u>true</u> cost to reach a goal from *n***
  - **=> if *G* is a goal node *h(G)=0***
- **Admissible heuristics are *optimistic***

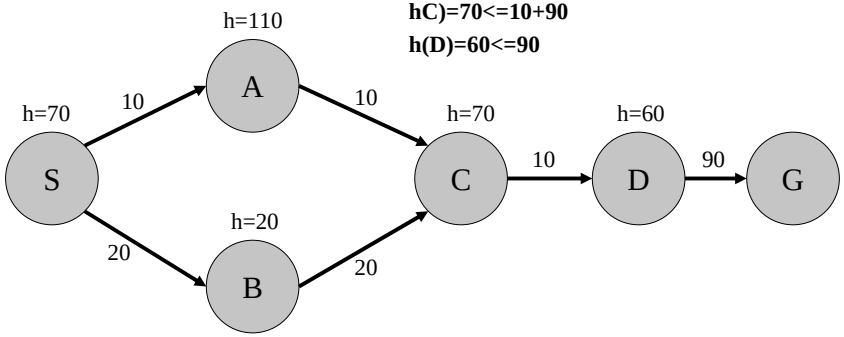
  - **think that the cost of solving the problem is less than it actually is!**
- **Examples**
  - **$h_{SLD}(n)$ never overestimates the actual road distance from n to a goal**
  - **h=number of misplaced tiles is admissible for the sliding-tile puzzle**

- **Theorem: If *h* is an *admissible heuristic* than A* is complete and optimal (with `Tree-search`)**
  - **Complete: finds a solution**
  - **Optimal: finds the least cost solution**

Our Example - Admissible Heuristic?

- **Is *h* admissible for our example?**

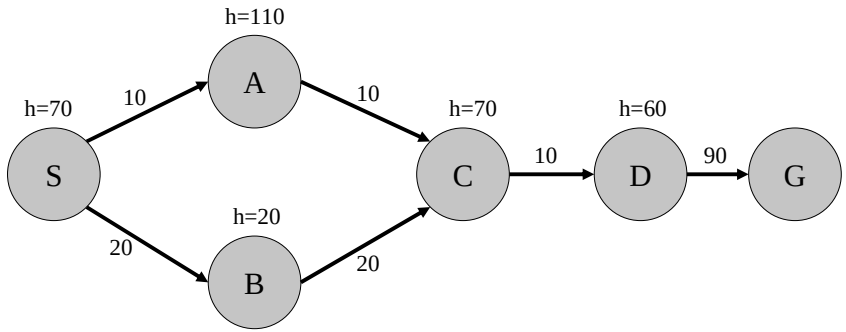
h(S)=70<=10+10+10+90 (the shorter path)
h(A)=110<=10+10+90
h(B)=20<=20+10+90
hC)=70<=10+90
h(D)=60<=90



- **Check that A* with `Tree-search` has found the optimal solution**
- **Theorem: If h is admissible => first path to a goal off Fringe (Open) is optimal**
  - **i.e. when A* expands <u>a goal</u> node it has found the optimal solution to it**

Re-opening Closed Nodes

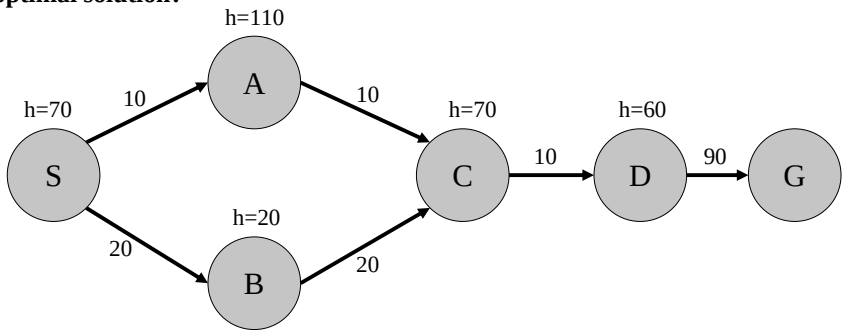
- ***h* is admissible and A* has found the optimal path**



- **Did we re-open closed nodes (i.e. add nodes to fringe that were already expanded)? – Yes, node C.**
  - **i.e. we did <u>not</u> discard repeated states**

Not Re-opening Closed Nodes

- **Did we re-open closed nodes (i.e. add nodes to fringe that were already expanded) – Yes, node C.**
- **If we do not re-open closed nodes (i.e. discard them), will we find the optimal solution?**

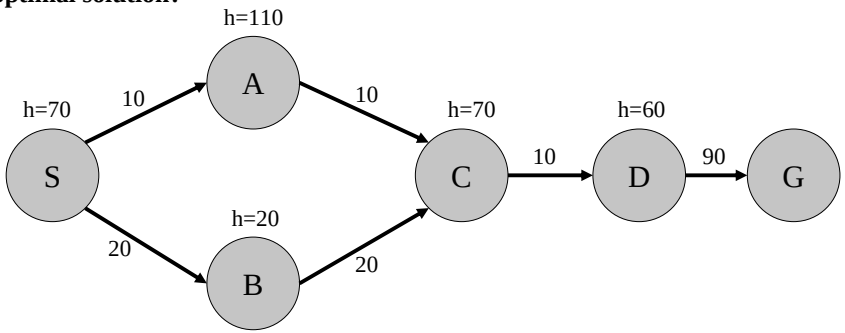


Fringe (OPEN): (S,70)

Expanded (CLOSED):

Not Re-opening Closed Nodes

- **Did we re-open closed nodes (i.e. add nodes to fringe that were already expanded) – Yes, node C.**
- **If we do not re-open closed nodes (i.e. discard them), will we find the optimal solution?**

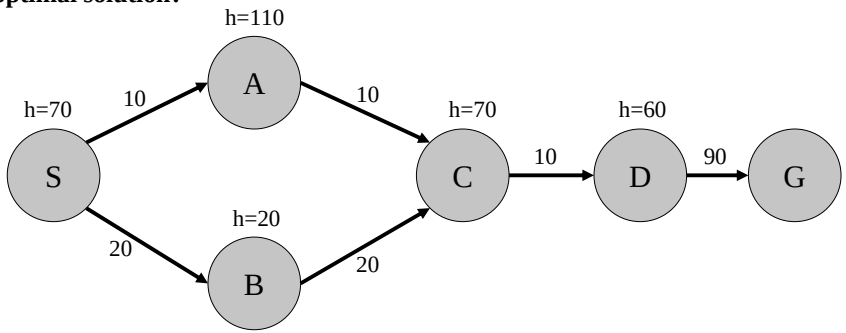


Fringe (OPEN): (A,120), (B,40)

Expanded (CLOSED): (S,70)

Not Re-opening Closed Nodes

- **Did we re-open closed nodes (i.e. add nodes to fringe that were already expanded) – Yes, node C.**
- **If we do not re-open closed nodes (i.e. discard them), will we find the optimal solution?**

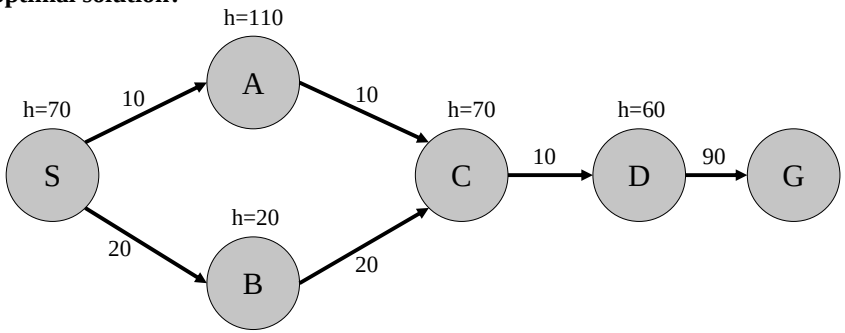


Fringe (OPEN): (A,120), (C,110)

Expanded (CLOSED): (S,70), (B,40)

Not Re-opening Closed Nodes

- **Did we re-open closed nodes (i.e. add nodes to fringe that were already expanded) – Yes, node C.**
- **If we do not re-open closed nodes (i.e. discard them), will we find the optimal solution?**

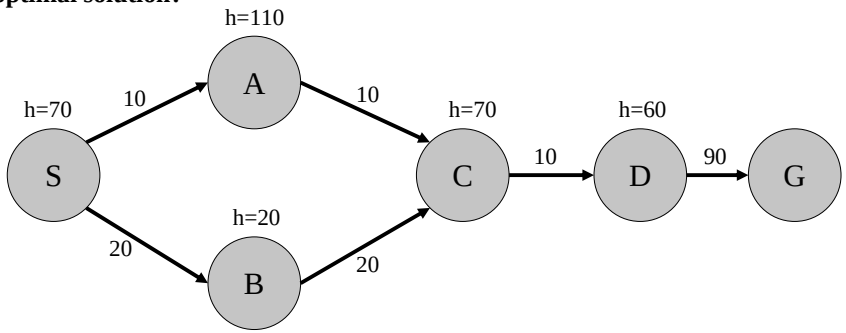


Fringe (OPEN): (A,120), (D,90)

Expanded (CLOSED): (S,70), (B,40), (C,110)

Not Re-opening Closed Nodes

- **Did we re-open closed nodes (i.e. add nodes to fringe that were already expanded) – Yes, node C.**
- **If we do not re-open closed nodes (i.e. discard them), will we find the optimal solution?**

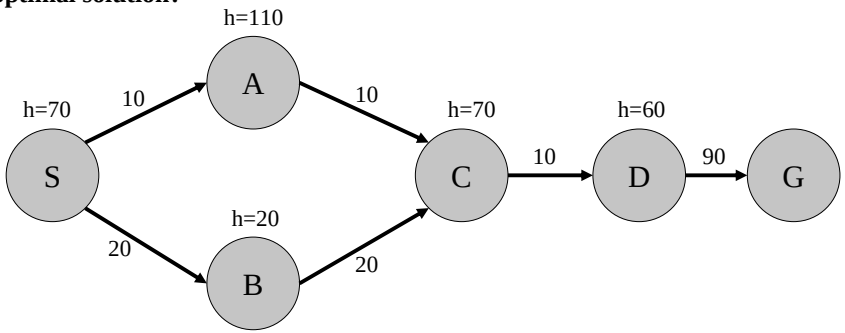


Fringe (OPEN): (A,120), (G,140)

Expanded (CLOSED): (S,70), (B,40), (C,110), (D,90)

Not Re-opening Closed Nodes

- **Did we re-open closed nodes (i.e. add nodes to fringe that were already expanded) – Yes, node C.**
- **If we do not re-open closed nodes (i.e. discard them), will we find the optimal solution?**

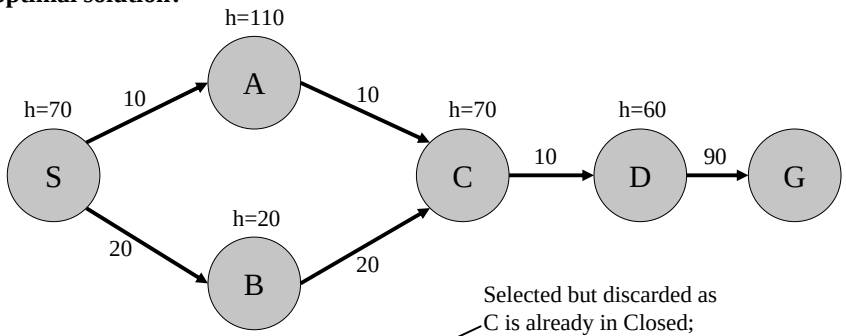


Fringe (OPEN): (G,140), **(C,90)**

Expanded (CLOSED): (S,70), (B,40), **(C,110),** (D,90), (A,120)

Not Re-opening Closed Nodes

- **Did we re-open closed nodes (i.e. add nodes to fringe that were already expanded) – Yes, node C.**
- **If we do not re-open closed nodes (i.e. discard them), will we find the optimal solution?**



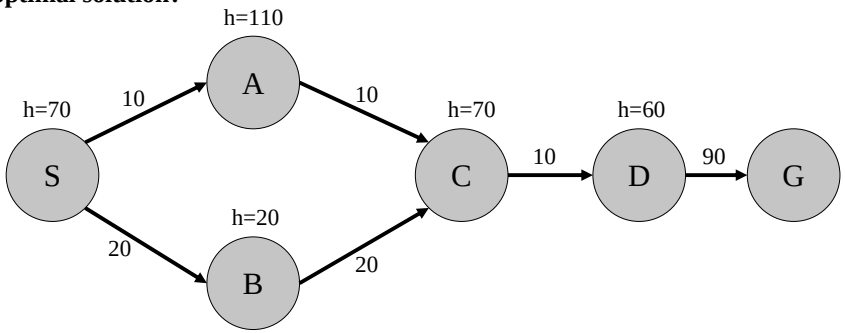
Selected but discarded as C is already in Closed; Then G is selected

Fringe (OPEN): (G,140), **(C,90)**

Expanded (CLOSED): (S,70), (B,40), **(C,110),** (D,90), (A,120)

Not Re-opening Closed Nodes

- **Did we re-open closed nodes (i.e. add nodes to fringe that were already expanded) – Yes, node C.**
- **If we do not re-open closed nodes (i.e. discard them), will we find the optimal solution?**

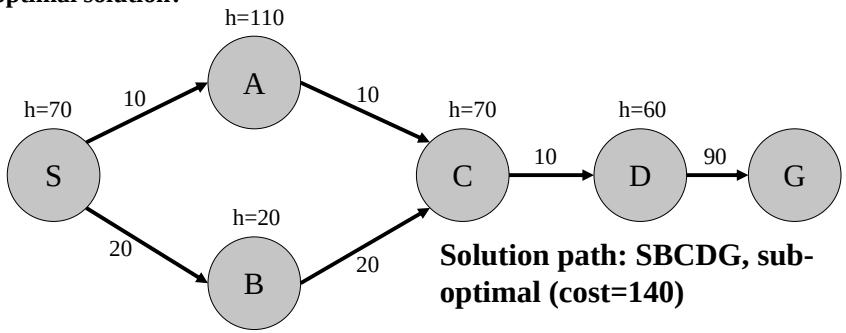


Fringe (OPEN):

Expanded (CLOSED): (S,70), (B,40), **(C,110),** (D,90), (A,120), (G,140) goal-stop

Not Re-opening Closed Nodes

- **Did we re-open closed nodes (i.e. add nodes to fringe that were already expanded) – Yes, node C.**
- **If we do not re-open closed nodes (i.e. discard them), will we find the optimal solution?**



Solution path: SBCDG, sub-optimal (cost=140)

Fringe (OPEN):

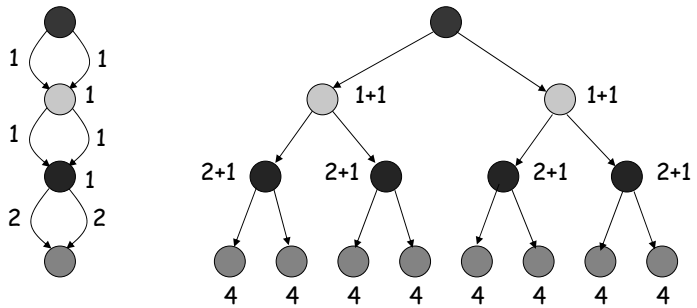
Expanded (CLOSED): (S,70), (B,40), **(C,110),** (D,90), (A,120), (G,140) goal-stop

Conclusions so far

- **Graph search, A* with admissible heuristic:**
  - **Closed nodes are re-opened, i.e. repeated states are not disregarded: complete and optimal (solution found and it is least cost)**
  - **Closed nodes are not re-opened, i.e. repeated states are disregarded: complete but not optimal (solution found but it is not least cost)**

However…

- **If we do not discard nodes already in closed (i.e. the repeated states), the search can be exponential in the number of visited states and can take a very long time**
- **We want to prune some or all of the repeated states**



Problem Formulation and Repeated States

- **Sometimes it is possible to avoid repeated states using a clever problem formulation – ex. 8-queens problem:**
- **Representation 1:**
- ***States:*** **Any arrangement of 0 to 8 queens**
- ***Operators:*** **Add a queen to any square**
- ***State space:*** **3 x 10^{14} states (there are repeated states)**
- **Searching *graph* – n! paths to a state with n queens**

- **Representation 2:**
- ***States:*** **Any arrangement of 0 to 8 queens, 1 in each column, with no queen attacking another**
- ***Operators:*** **Place a queen in the left-most empty column such that it is not attacked by any other queen**
- **State space: 2057 states (no repeated states)**
- **Searching *tree* – 1 path to each state**

Problem Formulation and Repeated States

- **Sometimes it is <u>not</u> possible to avoid repeated states**
  - **E.g. problems with reversible actions**
    - **route-finding (Romania)**
    - **sliding-blocks puzzles**
    - **…**
- **The search may take infinite time if repeated states are not avoided**
- **We want to prune some or all of the repeated states**

A* with Discarding of Repeated States

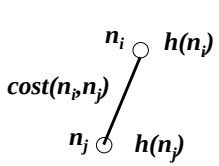
- **We will show that if h is consistent than there is no need to re-open already closed nodes (i.e. A* will be still optimal)**

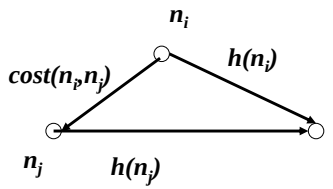
Consistency (Monotonicity) Condition - Review

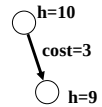
- **h is consistent if it satisfies the triangle inequality for all n**

  $h(ni) \leq cost(ni,nj) + h(ni)$

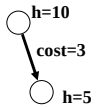
  parent **child**







consistent
10<=9+3



not consistent
10 <= 5+3

Consistency (Monotonicity) Condition –Th 1 & 2

- **Theorem 1: If *h(n)* is consistent, than *f(nj) ≥ f(ni)*, i.e. *f* never decreases along any path** (child, parent)

  Given: h(ni) ≤ c(ni, nj)+h(nj)

  To prove: f(nj) ≥ f(ni)

  Proof: f(nj)=g(nj)+h(hj)=

  =g(ni)+c(ni,nj)+h(nj)=

  ≥ g(ni) + h(ni) =

  =f(ni)

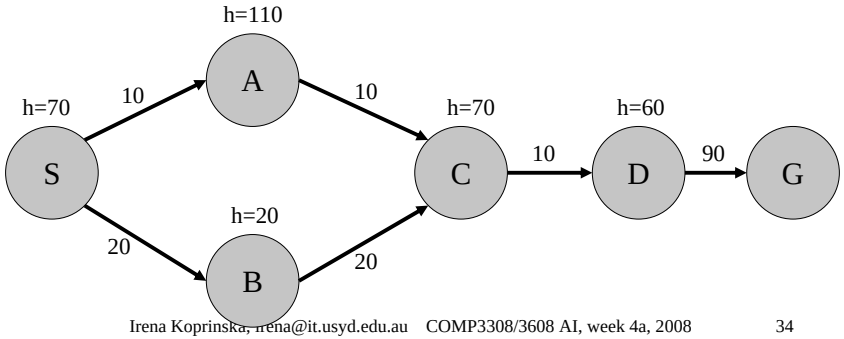
  deff. h(n) consistent

  =>f(nj) ≥ f(ni)
- **Theorem 2: If *f(nj) ≥ f(ni)*, i.e. *f* never decreases along any path, than *h(n)* is consistent.**

Consistency and Admissibility

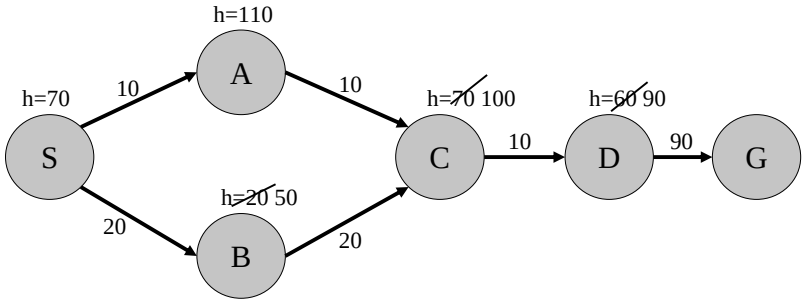
- **Theorem: If *h(n)* is consistent, than *h(n)* is admissible**
- **The opposite is not true, i.e.**
  If *h(n)* is admissible ≠> *h(n)* is consistent

- **Is *h* consistent? h(A)=110 is not <=h(C)+10=80**
  i.e. h(n) is admissible but not consistent



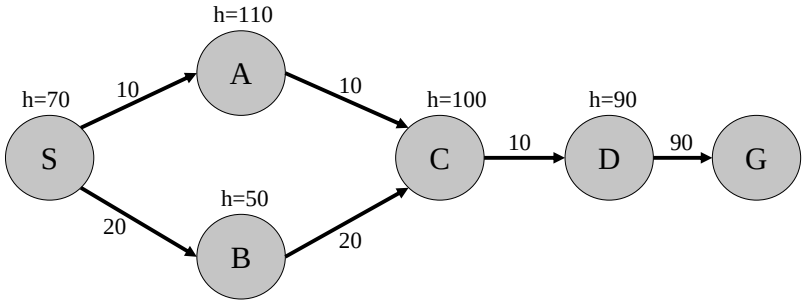
Consistent Heuristics

- **Theorem: If *h(n)* is consistent, than first path to node X off Fringe (Open) is optimal**
  - **node X off Fringe = node X from Fringe has been selected for expansion**
  - **i.e. when A* expands a node it has already found the optimal path to it**
- **Consequence: No need to reopen closed nodes**
  - **i.e. no need to revisit states**
- **Modified example – h is now consistent**



A* with Consistent Heuristic – Our Example

- **Modified example – h is now consistent**

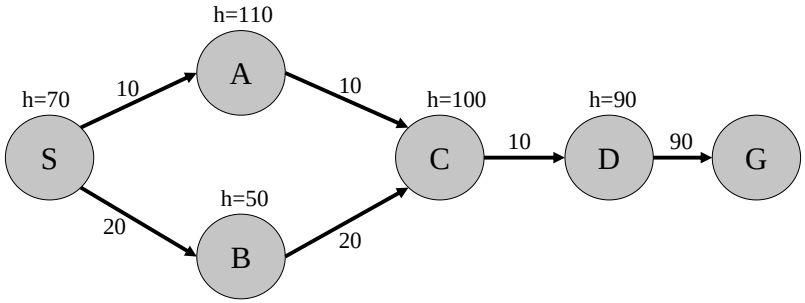


Fringe (OPEN): (S,70)

Expanded (CLOSED):

A* with Consistent Heuristic – Our Example

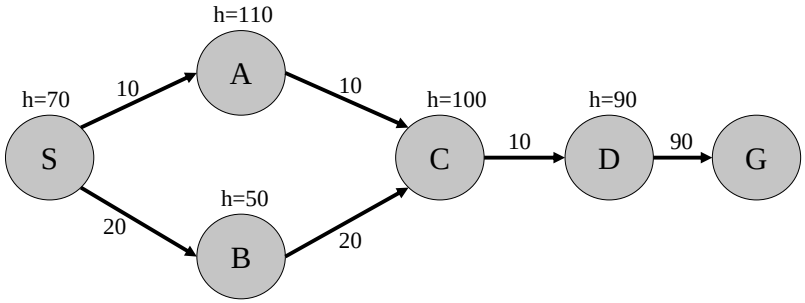
- **Modified example – h is now consistent**



Fringe (OPEN): (A,120), (B,70)
Expanded (CLOSED): (S,70)

A* with Consistent Heuristic – Our Example

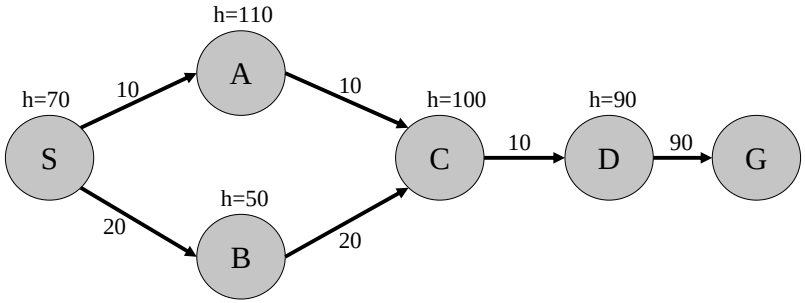
- **Modified example – h is now consistent**



Fringe (OPEN): (A,120), (C,140)
Expanded (CLOSED): (S,70), (B,70)

A* with Consistent Heuristic – Our Example

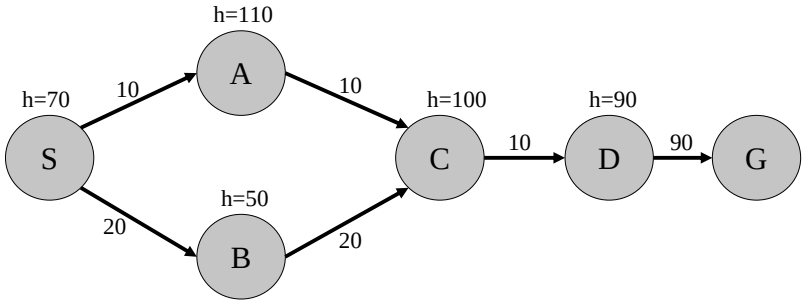
- **Modified example – h is now consistent**



Fringe (OPEN): (C,140), (C,120)
Expanded (CLOSED): (S,70), (B,70), (A,120)

A* with Consistent Heuristic – Our Example

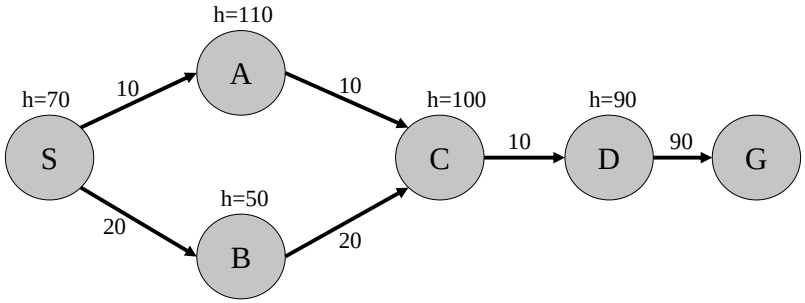
- **Modified example – h is now consistent**



Fringe (OPEN): (C,140), (D, 120)
Optimal path to C found
Expanded (CLOSED): (S,70), (B,70), (A,120), (C,120)

A* with Consistent Heuristic – Our Example

- **Modified example – h is now consistent**

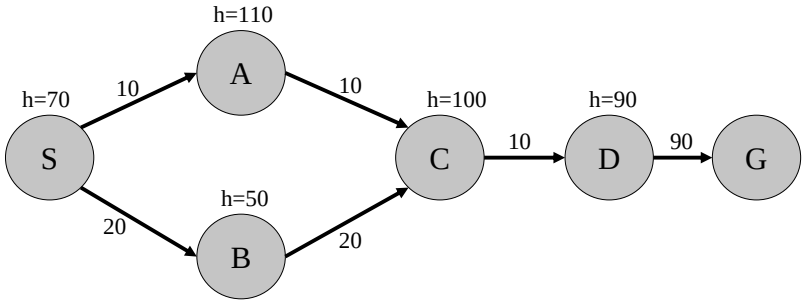


Fringe (OPEN): (C,140), (G,120)

Expanded (CLOSED): (S,70), (B,70), (A,120), (C,120), (D,120)

A* with Consistent Heuristic – Our Example

- **Modified example – h is now consistent**

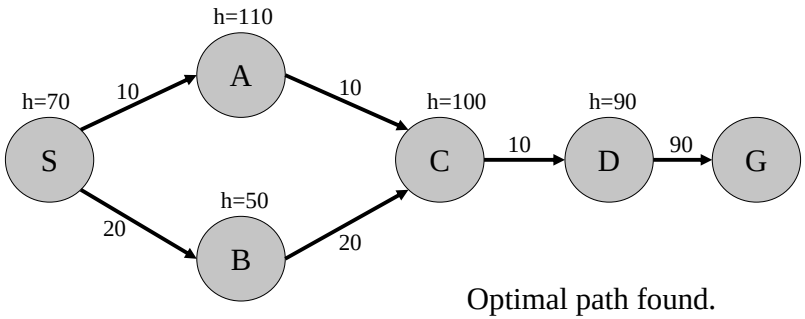


Fringe (OPEN): (C,140)

Expanded (CLOSED): (S,70), (B,70), (A,120), (C,120), (D,120), (G,120) stop

A* with Consistent Heuristic – Our Example

- **Modified example – h is now consistent**



Optimal path found.
No re-opening of closed nodes.

Fringe (OPEN): (C,140)

Expanded (CLOSED): (S,70), (B,70), (A,120), (C,120), (D,120), (G,120) stop

A* with Consistent Heuristic

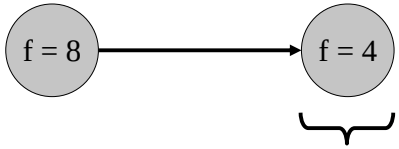
- **Theorem: If h is consistent, A* using `Search-Graph` is optimal**
  - **`Search-graph` does not re-open closed nodes; it discards them**
  - **`Search-tree` re-opens closed nodes**
  - **`Closed` – the list of expanded nodes**

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

The selected node from `fringe` is added to `closed` only if it is not already there

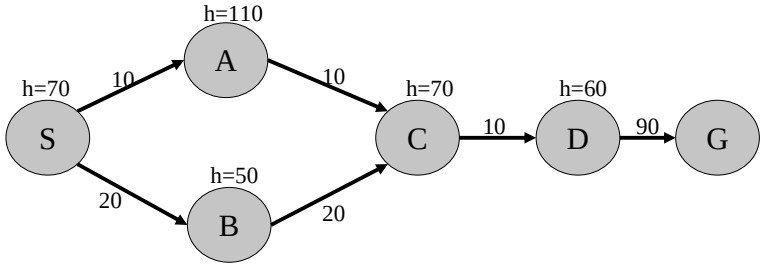
Enforced Monotonicity (Consistency)

- **We can enforce monotonicity along a path by using the parent's f value if it is greater than the child's f-value (*pathmax* equation)**



- **Note that if this is done on-line, i.e. as we search, it may not solve the problem of reopening nodes from closed – see the example on the next slide**

Enforced Monotonicity - Example



….

Fringe (OPEN): (G,140), (C,~~90~~) 120

Expanded (CLOSED):(S,70), (B,70), (C,110), (D,110), (A,120)

- **As we expand A, we notice that h is inconsistent and correct h(C) to 120 but without re-opening the closed node C we will not be able to find the optimal solution**

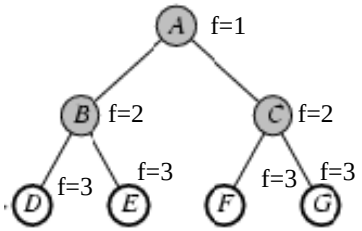
Enforced Monotonicity

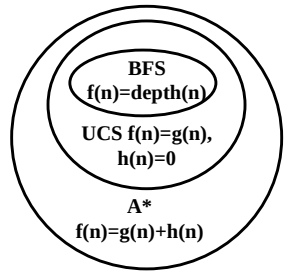
- **If pathmax is applied to an admissible heuristic, will it be still admissible?**

  Yes

Special cases of A*

- **UCS is a special case of A* when h(n)=0 for all n**
  - **A*: f(n)=g(n)+h(n), h(n)=0 => f(n)=g(n), UCS uses g(n)**
- **BFS is a special case of A* when f(n)=depth(n)**





- **=> BFS is a special case of UCS g(n)=depth(n)**

Summary of definitions and results

- **An admissible heuristic never overestimates the truecost to a goal**
- **A consistent (monotonic) heuristic satisfies the triangle equation**
- **h(n) satisfies the triangle equation <=> f(n) does not decrease along any path**

- **Admissible ≠> consistent**
- **Consistent => admissible**

- **Consistent => first path to a node X off the fringe (Open) is optimal for all X**
- **Admissible => first path to a goal node off the fringe (Open) is optimal**
- **Admissible ≠> first path to a node X off the fringe (Open) is optimal for all X**

Summary of Definitions and Results (2)

- **If h(n) is admissible than A* is optimal with `Tree-search` (i.e. nodes revisiting states are not discarded; there are repeated states)**
  - **Whenever A* expands a goal node, it has already found the optimal path to it**
- **If h(n) is consistent than A* is optimal with `Graph-search` (i.e. nodes revisiting states are discarded; there are no repeated states)**
  - **Whenever A* expands a node, it has already found the optimal path to it**

- **If h(n) is consistent, A* s optimally efficient**
  - **No other optimal algorithm will expand fewer nodes than A***

Completeness and Optimality Again

- **A* with consistent heuristic is complete, optimal and optimally efficient, and there is no need to revisit states**
- **However theoretical completeness does not mean practical completeness if it takes too long to get the solution, recall:**

  Time? Exponential in [relative error in h * length of solution]

  Space? Exponential, keeps all nodes in memory

  Both are big problems; typically runs out of space before running out of time
- **So, if we can't design accurate consistent heuristic, it may be better to settle for a non-admissible heuristic that works well in practice or for a local search algorithm even though completeness and optimality are no longer guaranteed.**
- **Also, although dominant (i.e. good) heuristics are better, they may need a lot of time to compute; it may be better to use a simpler heuristic - more nodes will be expanded but overall the search may be faster.**

Acknowledgements

- **www.cs.ualberta.ca/~holte/CMPUT651/admissibility.ppt**

- **http://ai.stanford.edu/~latombe/cs121/2007/slides/D-heuristic-search.ppt**