Artificial Intelligence:
Search Methods for Problem Solving

Finding Optimal Paths

A First Course in Artificial Intelligence: Chapter 5

Deepak Khemani

Department of Computer Science & Engineering

IIT Madras

Optimal Solutions

- So far we have focused on finding the solution quickly
- We saw the use of a heuristic function to guide search
- Now we turn our attention to the quality of the solution
- We look at algorithms to find optimal solutions/paths
- Our focus will be on algorithms that guarantee optimal solutions
  - unlike the stochastic methods we have seen so far.
- We will then move on to finding optimal solutions quickly!

Finding Optimal Paths

- Breadth First Search finds a solution with the smallest *number* of moves.
- But if *cost* of all moves is not the same then an *optimal solution* may not be the one with the smallest number of moves.
- For example, changing two buses to reach the railway station may be counted as four moves
  - Get into bus 1
  - Get off bus 1
  - Get into bus 2
  - Get off bus 2
- This may be *cheaper* than
  a smaller length solution of two moves.
  - Get into autorickshaw
  - Get off autorickshaw.

Brute Force

The following algorithm is guaranteed to return the optimal solution

British Museum Procedure
    Explore the entire search space.
    Return the optimal solution
End.

P. H. Winston : the only way to locate something in the British Museum is to explore the entire museum!

- The algorithm is conceptually simple.
- It simply searches the entire search tree.
- Computationally it is mindlessly expensive.

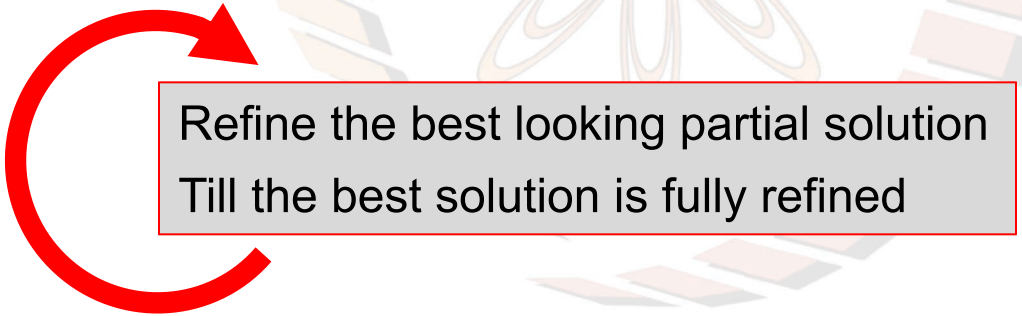
Goal : Search as little of the space as possible, while guaranteeing the optimal solution.

Heuristic Refinement

A *Best First* approach to solving problems

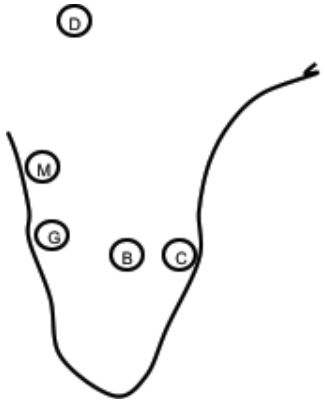
Given a set of candidates a search algorithm has to choose from.

Each candidate is tagged with an estimated cost of the complete solution.



The manner in which these estimates are computed gives rise to different algorithms.

A Traveling Salesman Problem (TSP)



What is the shortest round trip covering the five cities and back?

Distance Matrix

	Chennai	Goa	Mumbai	Delhi	Bangalore
Chennai	0	800	1280	2190	360
Goa	800	0	590	2080	570
Mumbai	1280	590	0	1540	1210
Delhi	2190	2080	1540	0	2434
Bangalore	360	570	1210	2434	0

Branch & Bound in a refinement space

- Initial solution set contains *all* solutions.
- In each step we partition the set into a smaller sets.
- Until we can pick a solution that is fully specified.

- Each solution set needs to have an *estimated cost.*
- B&B will refine the solution that has the least estimated cost.
- The process will continue until
  - we have a complete solution at hand, and
  - when no other candidate solution has a better estimated cost.

An optimal solution can be guaranteed by ensuring that the estimated cost is a lower bound on the actual cost.

Higher the estimate the better

- A solution will never be cheaper than it is estimated to be.
- Thus if a fully refined solution is cheaper than a partially refined one, the latter need not be explored – PRUNING.
- Estimate cost = 0 is the simplest estimate, BUT
- Zero estimate cost will not prune any solutions!
- The higher the estimate, the better the pruning.

Branch & Bound for TSP

- Let the candidate solutions be permutations of the list of city names.
- The initial solution includes all permutations.
- Refine the cheapest solution set by specifying a specific segment.

Heuristic for choosing segment:

– The one with the minimum cost

Cost Estimation Considerations:

– For *tighter* estimates, edges that cause three segment cycles are avoided.

– For *expediting* cost estimation, the edge with the least cost, that does not cause three segment cycles is considered, even if it means nodes with degree more than 2 are considered for estimation

A Lower Bounding Estimated Cost for TSP

	Chennai	Goa	Mumbai	Delhi	Bangalore
Chennai	0	800	1280	2190	360
Goa	800	0	590	2080	570
Mumbai	1280	590	0	1540	1210
Delhi	2190	2080	1540	0	2434
Bangalore	360	570	1210	2434	0

A Lower Bounding Estimated Cost for TSP

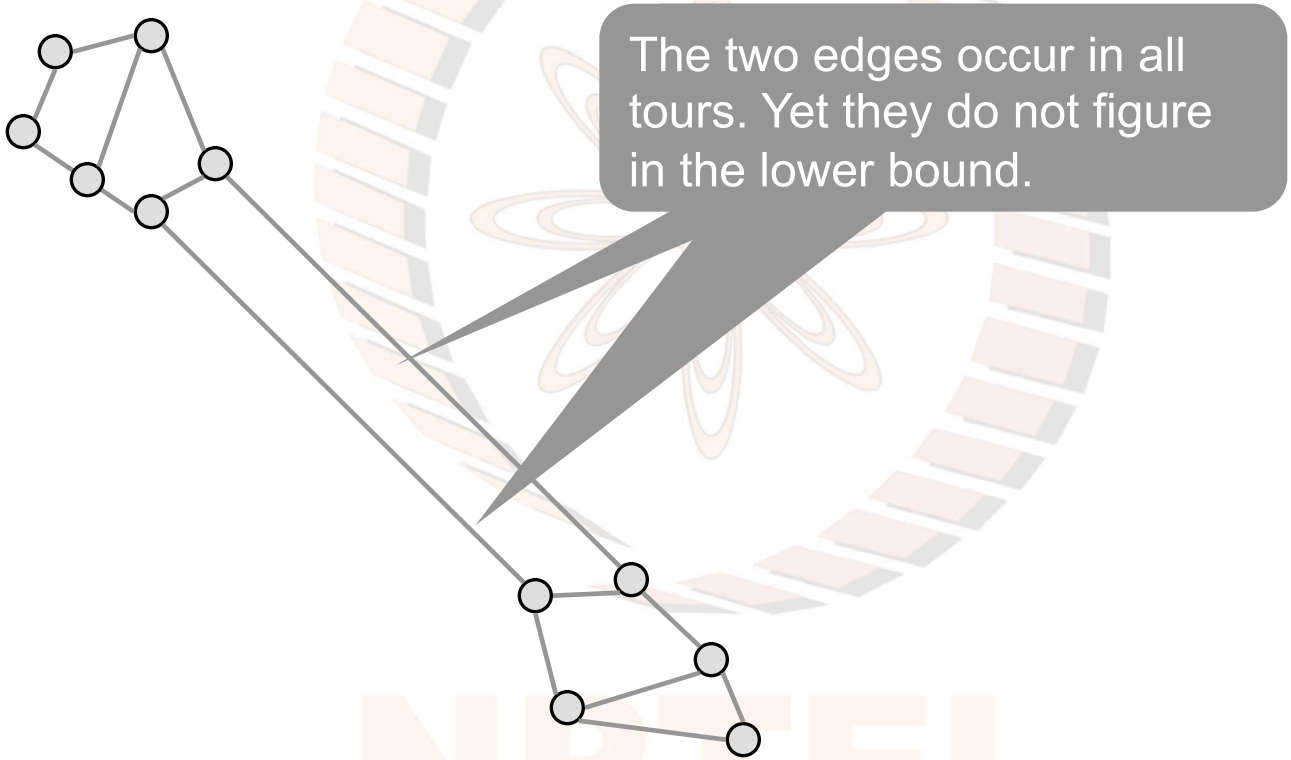
	Chennai	Goa	Mumbai	Delhi	Bangalore
Chennai	0	800	1280	2190	360
Goa	800	0	590	2080	570
Mumbai	1280	590	0	1540	1210
Delhi	2190	2080	1540	0	2434
Bangalore	360	570	1210	2434	0

- Consider first the absolute lower bound for any tour.
- For each row add up the smallest two positive entries.
  - For example, for row 1 select 360 and 800.
  - The contribution of the Chennai segment will be smallest when it lies between Goa and Bangalore in the tour.
- In this manner pick the smallest two costs for each city, sum them up and divide by two.

$$LB = \frac{(360 + 800) + (570 + 590) + (590 + 1210) + (1540 + 2080) + (360 + 570)}{2}$$

$$= \frac{8460}{2} = 4335$$

Lower Bound may not be feasible



Refinement

- As segments get fixed in the tour *their known costs* are included in the estimate.
- As this happens the estimate becomes more accurate.
- Consider now the refinement search space.
  - The root consists of a node representing all solutions.
  - We can partition this set in two, one including a particular arc and the other excluding it.
  - These two sets can then be further refined recursively till each node describes one tour precisely.
- We apply the heuristic of choosing the smallest segment.
  In our example this is
  Chennai – Bangalore with a cost 360 km.

Estimated cost of partial solutions

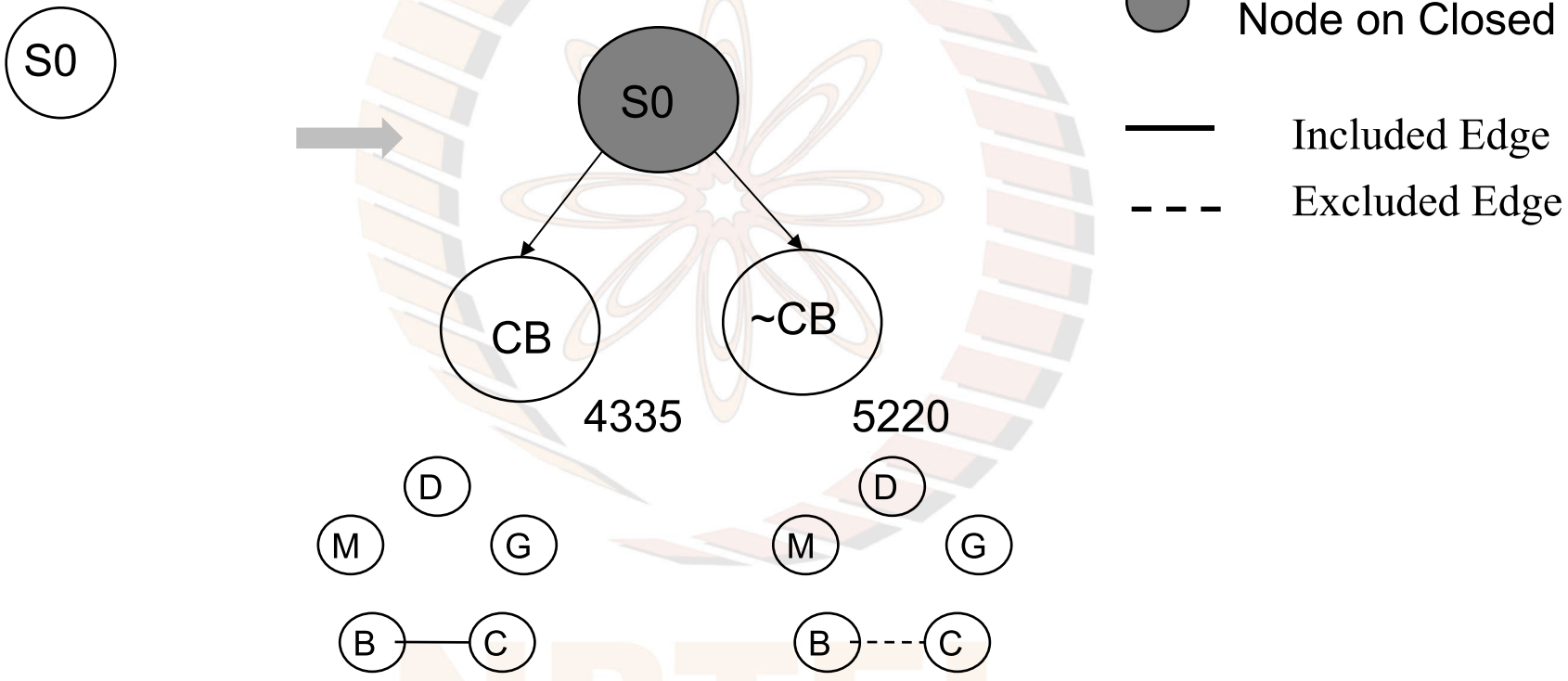
- For example the estimated cost of a tour including the Chennai-Mumbai (CM) segment is
- $LB = \frac{(360 + 1280) + (570 + 590) + (590 + 1280) + (1540 + 2080) + (360 + 570)}{2}$
- $= \frac{9010}{2} = 4505$
- Suppose in addition to the above we also want to include the Mumbai Bangalore (MB) segment. The cost of this segment, 1210, needs to be included.
- $LB = \frac{(360 + 1280) + (570 + 590) + (1210 + 1280) + (1540 + 2080) + (360 + 1210)}{2}$
- $= \frac{10480}{2} = 5240$
- BUT to avoid the Chennai-Mumbai-Bangalore cycle we have to REMOVE the Chennai Bangalore segment from the estimate!

Chennai-Mumbai-Bangalore tour

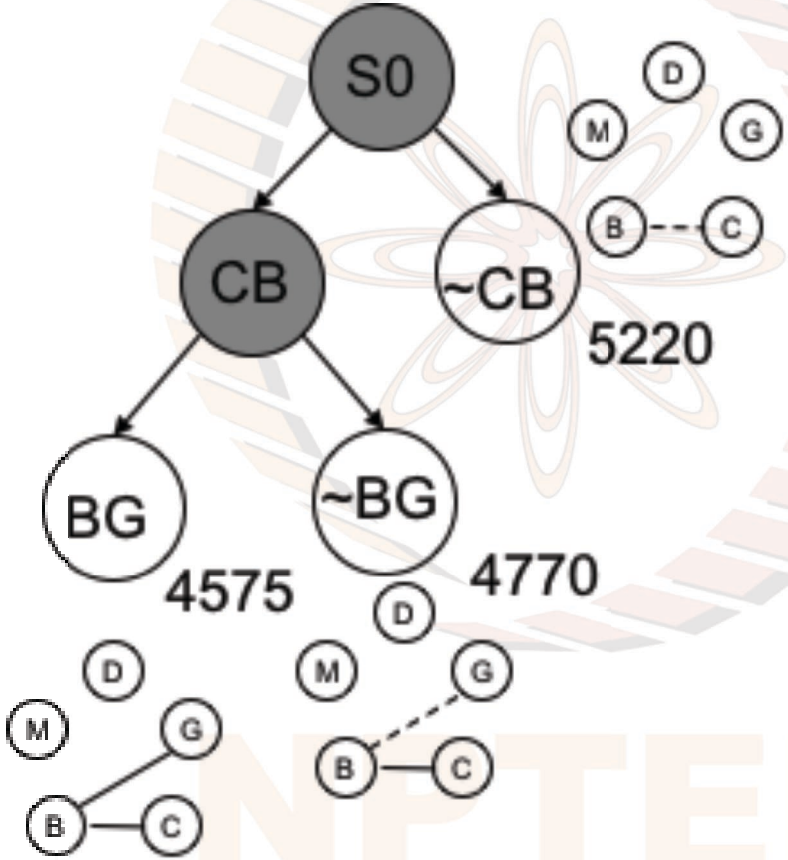
- The BC segment costing 360 contributes twice in the above estimate.
- One must replace them with the next better costs, 800 and 570.

- $LB = \frac{(800 + 1280) + (570 + 590) + (1210 + 1280) + (1540 + 2080) + (570 + 1210)}{2}$
- $= \frac{11130}{2} = 5565$
- Tighter estimates should mean more pruning.
- We look at B&B on the example problem.

B&B TSP Example



B&B TSP Example



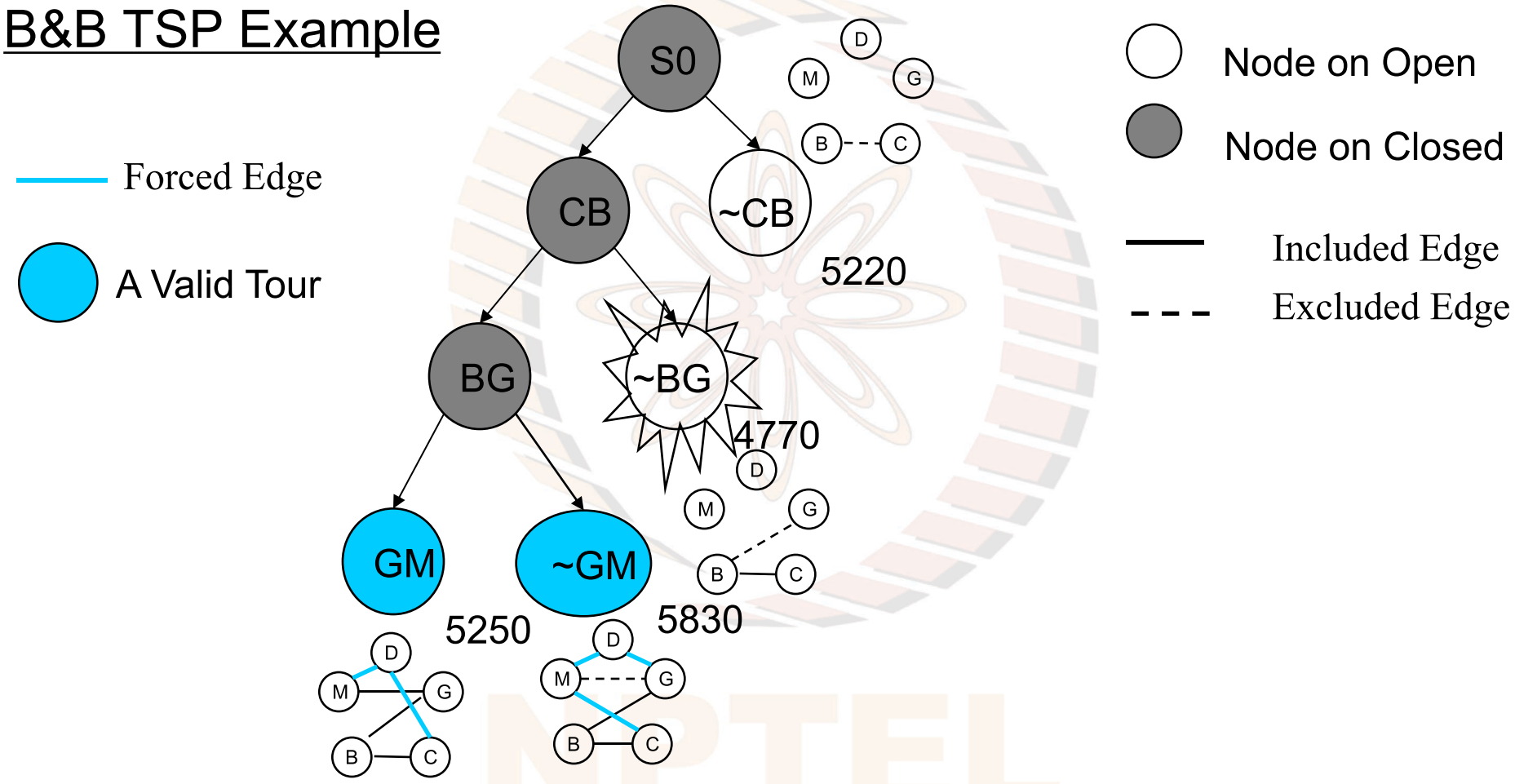
Node on Open

Node on Closed

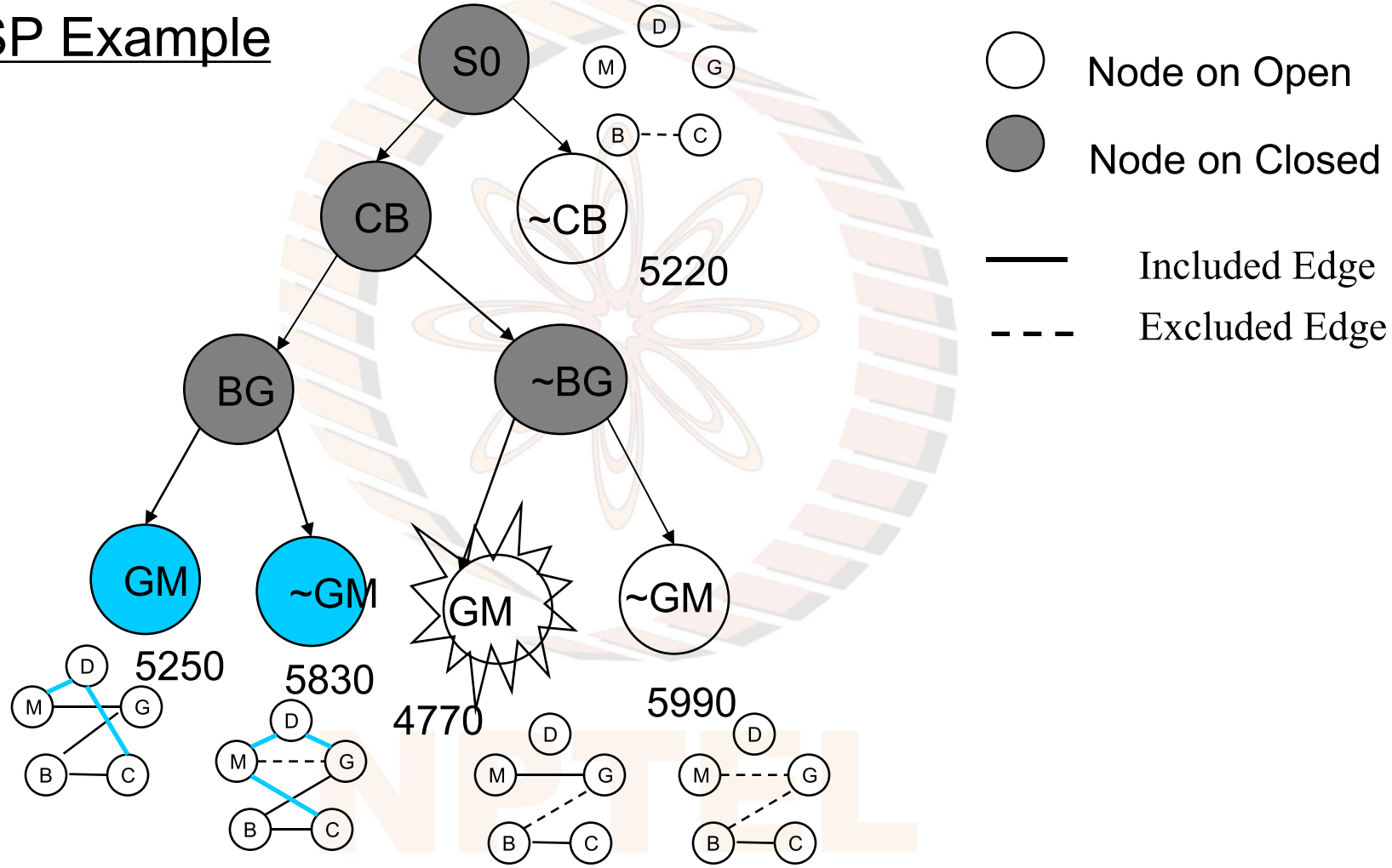
Included Edge

Excluded Edge

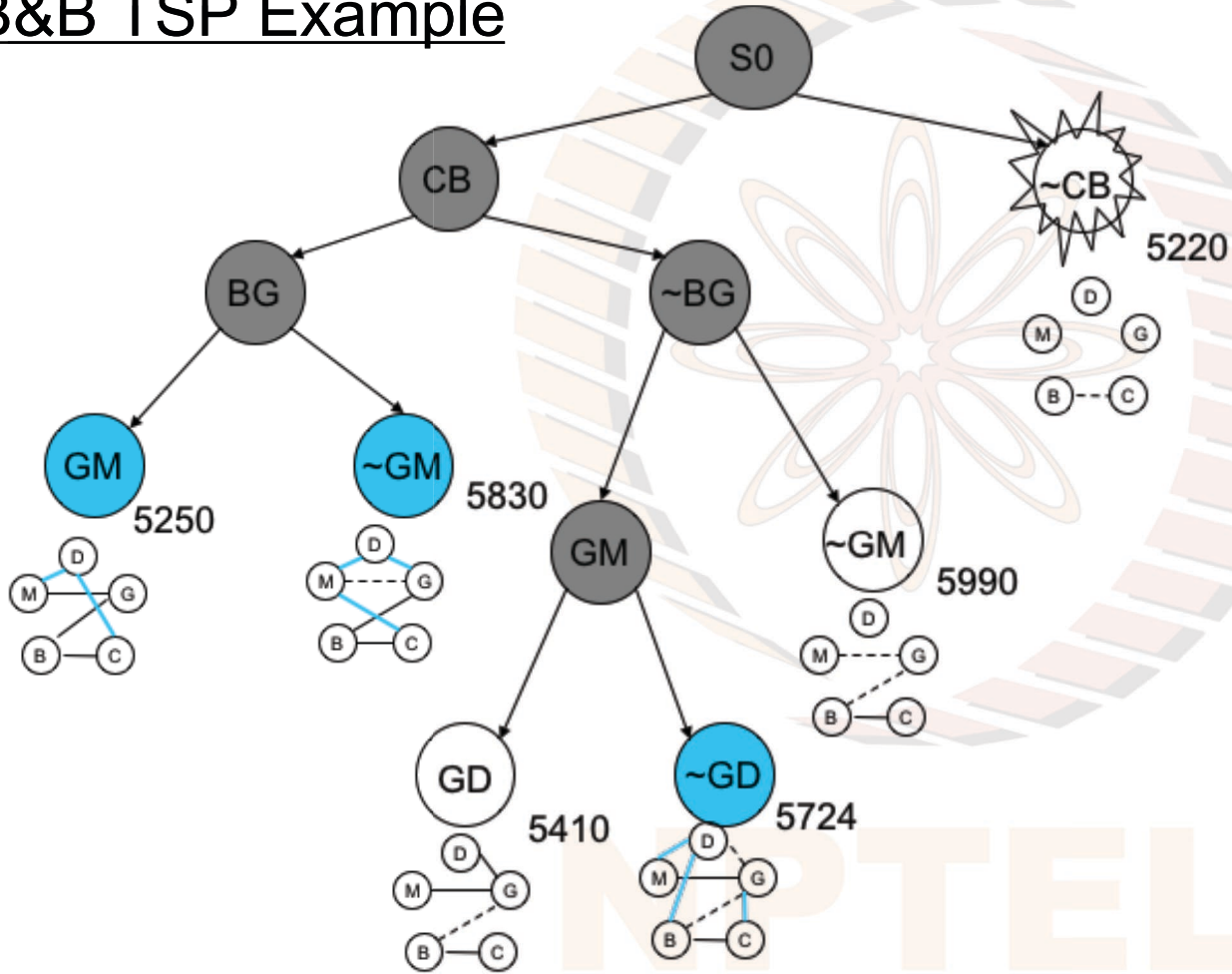
B&B TSP Example



B&B TSP Example



B&B TSP Example



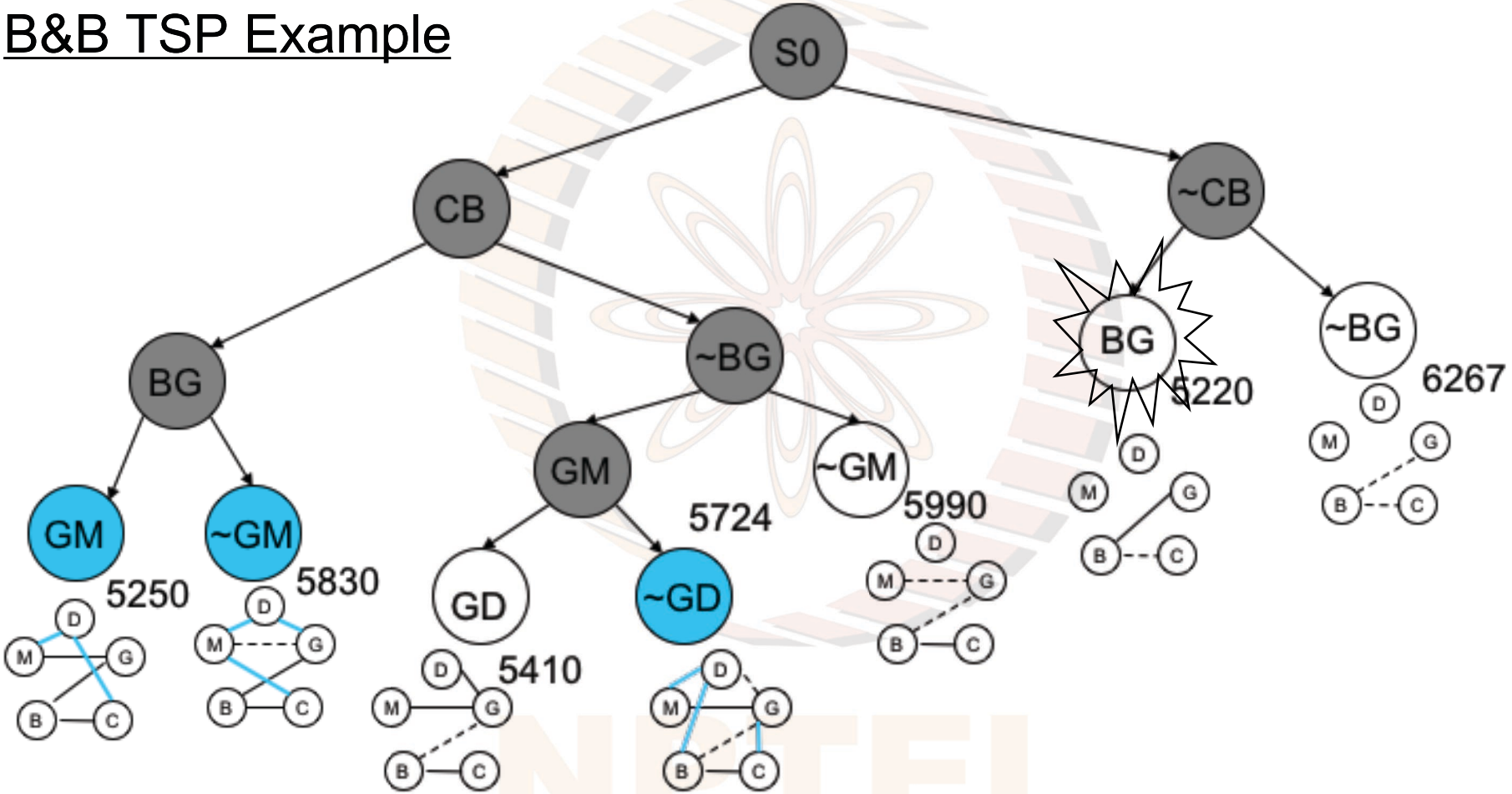
Node on Open

Node on Closed

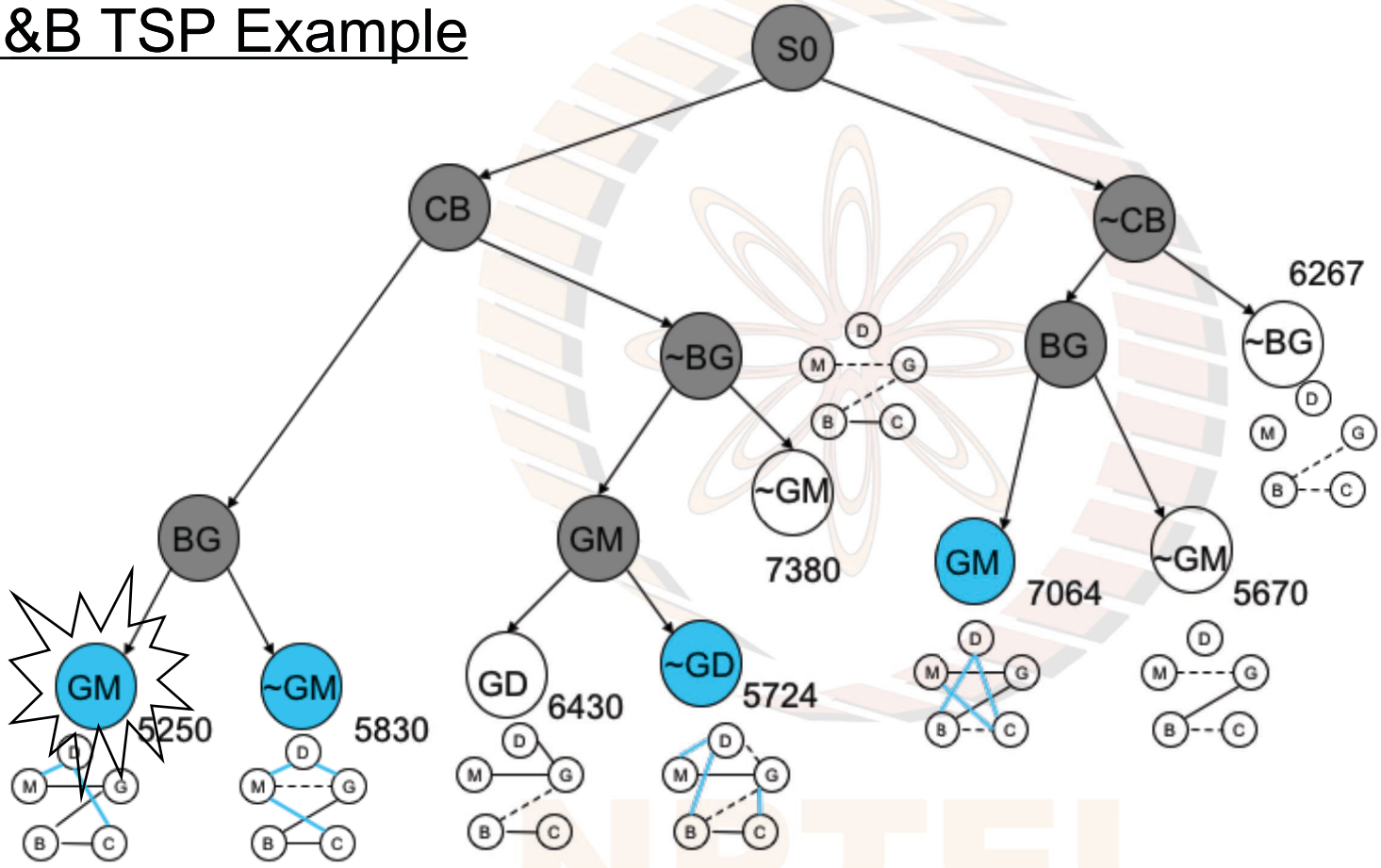
Included Edge

Excluded Edge

B&B TSP Example

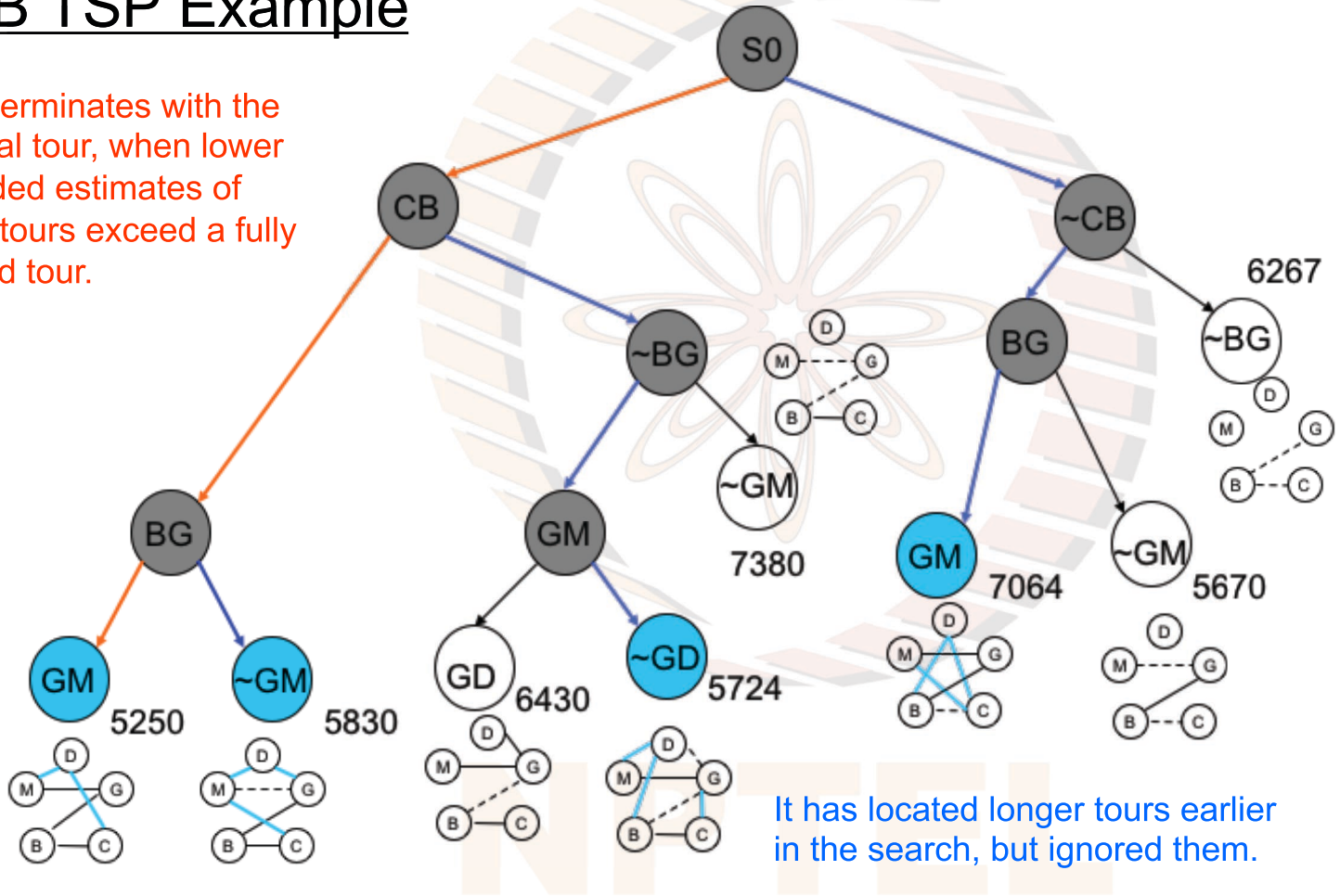


B&B TSP Example



B&B TSP Example

B&B terminates with the optimal tour, when lower bounded estimates of other tours exceed a fully refined tour.



It has located longer tours earlier in the search, but ignored them.

State Space Search
Branch & Bound

State Space (with no edge costs): Blind Search

Depth First Search

Deepest candidates are best

Dives headlong into the search space optimistically

Breadth First Search

Shallowest candidates are best

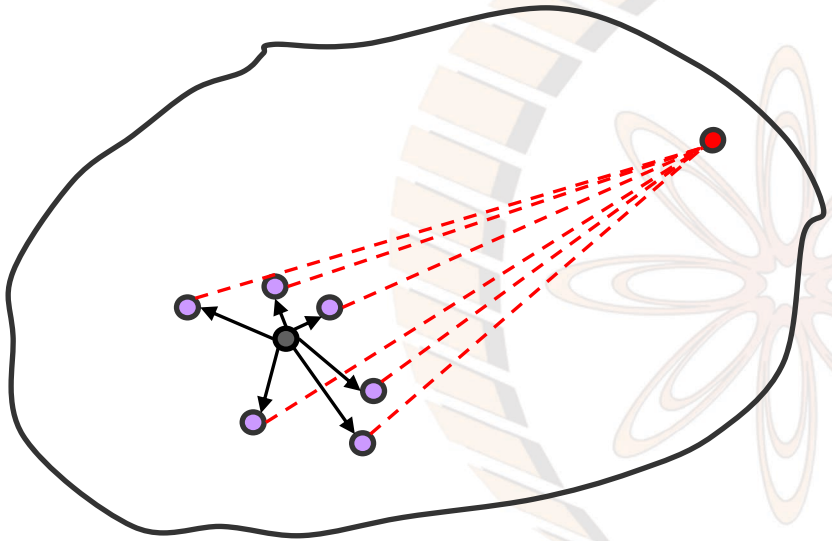
Stays as close to Start as possible

Wades into the search space and finds a solution with the *shortest number of hops*

Depth First Iterative Deepening

Depth First masquerading as Breadth First

State Space: Heuristic functions (recap)



The heuristic function **estimates** the distance to the goal.

This estimate, h(n), can be used to decide **which** node to pick from OPEN

Best First Search

Candidates that appear to be *closest to the goal* are best

Chooses the candidate with the *lowest h-value node* in the hope of finding a solution *sooner*.

Branch and Bound

- Organize a search space that does not preclude any solution.
  - Necessary for finding the optimal solution

- Search space could be the state space, in which a partial sequence of moves is extended.
- Search space could also be the solution space in which an abstract solution is refined.
  - As seen with the TSP example

- Continue looking for a solution (extend/refine) until
  - A complete solution (with known cost) is available, and
  - no other possible solution with a smaller cost exists.

The basic idea behind B&B is to prune those parts of a search space which *cannot* contain a better solution.

A Tiny Search Problem with Edge Costs

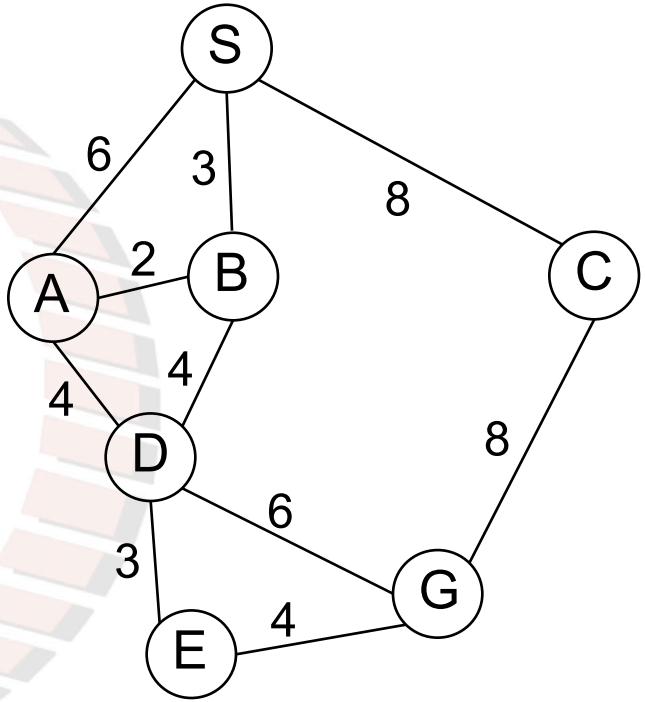
The *MoveGen* function

S → (A,B,C)
A → (S,B,D)
B → (S,A,D)
C → (S,G)
D → (A,B,G,E)
E → (D,G)
G → (C,D,E)

The State Space

Note: The placement of nodes in the graph may not reflect the edge costs accurately.

Cost of a path is the sum of the cost of edges in the path.

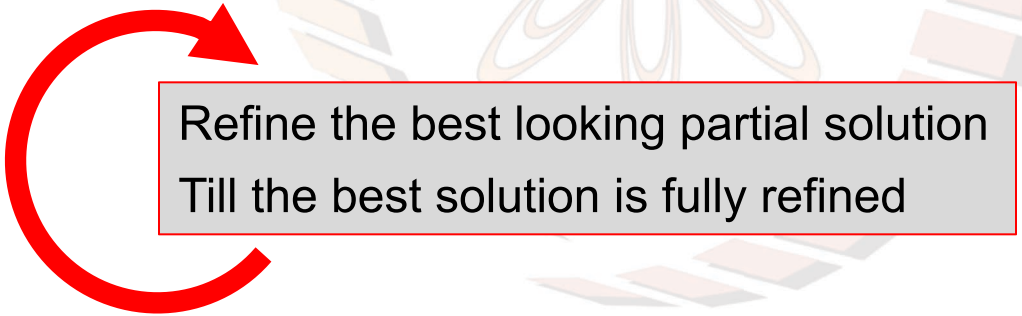


What is the *shortest path* from S to G?

Branch & Bound

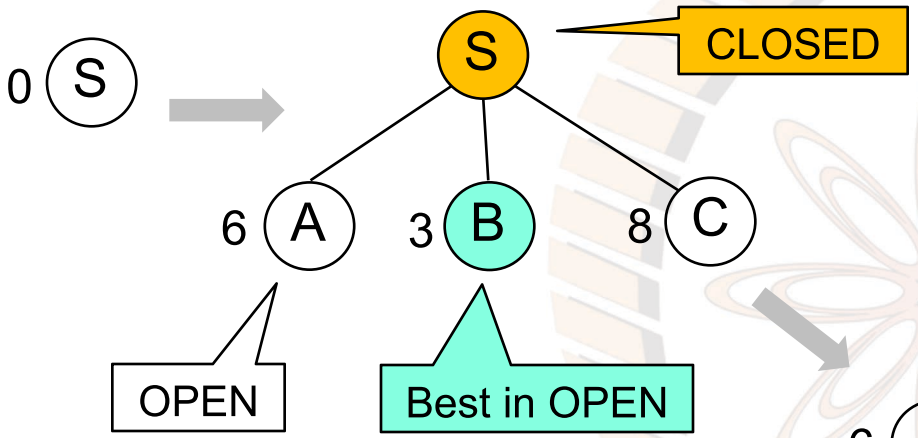
Each candidate is tagged with an *estimated cost of the complete solution*.

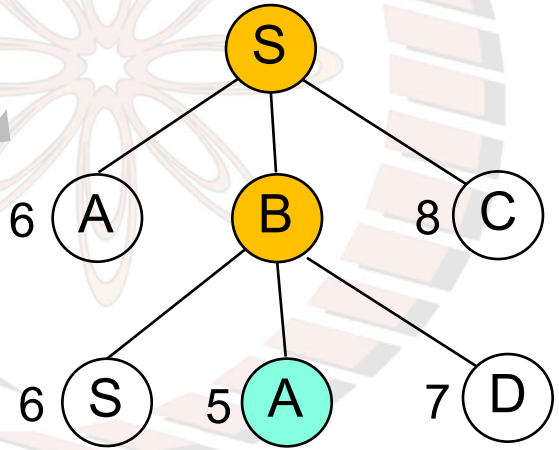
Estimated cost of (full) solution = actual cost of partial solution

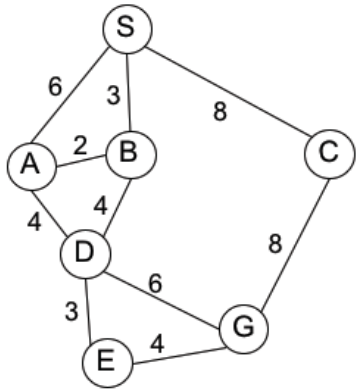


Branch & Bound extends the cheapest partial path

Branch & Bound



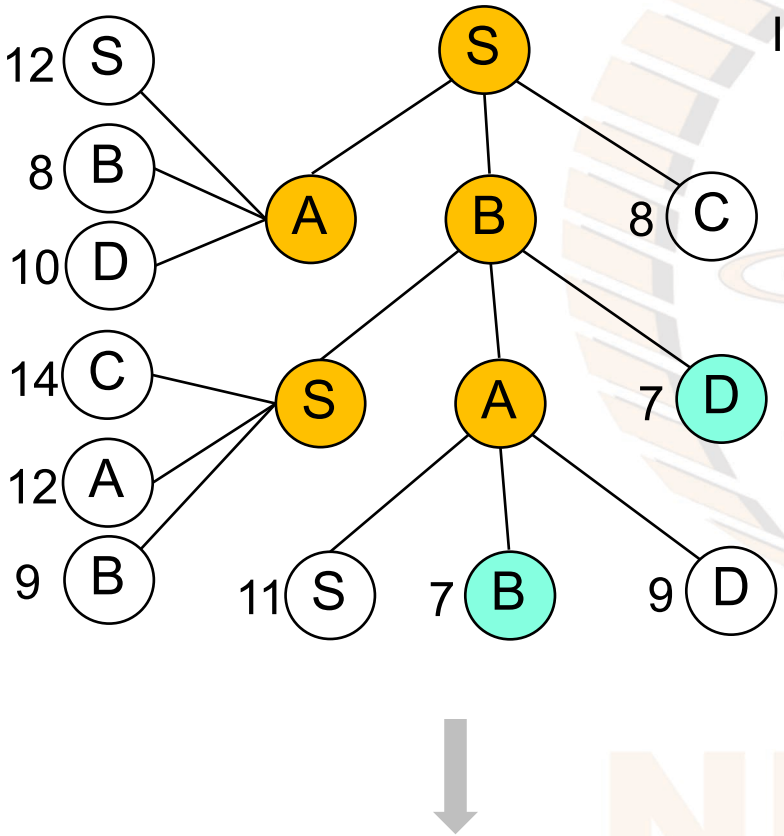




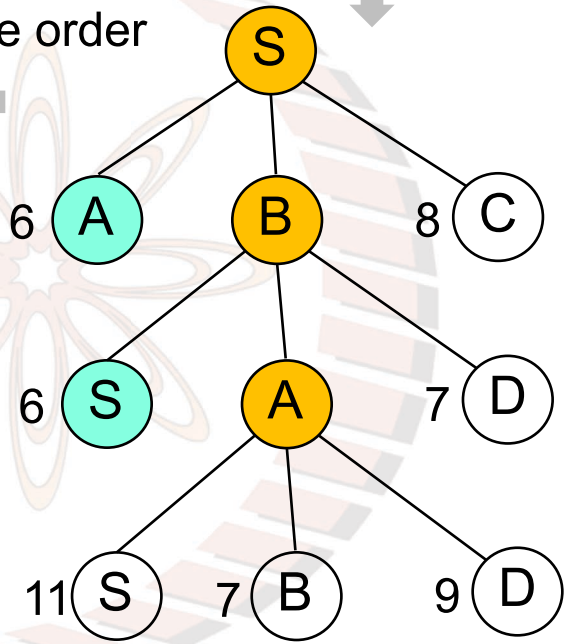
Each node in the Search Space represents a path from S to the node.

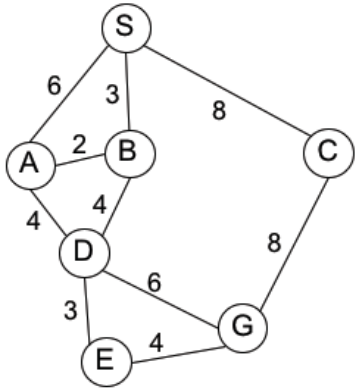
Branch & Bound extends the cheapest partial path

Branch & Bound

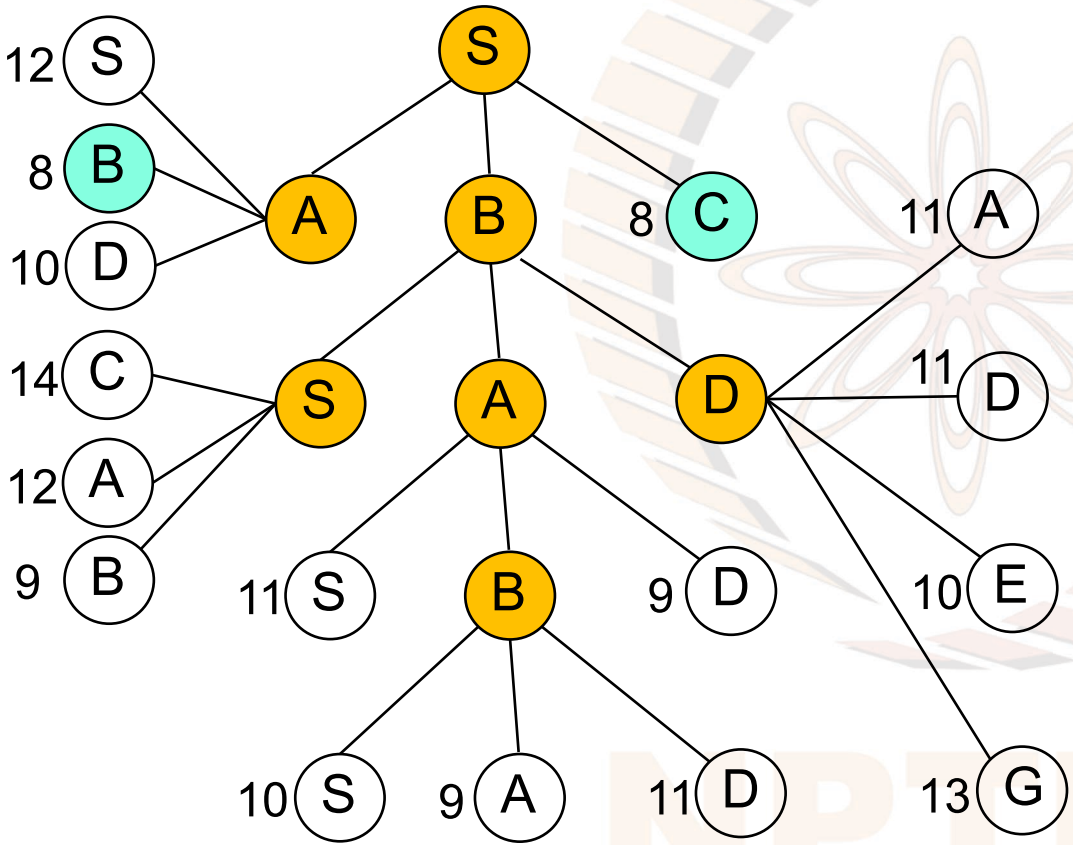


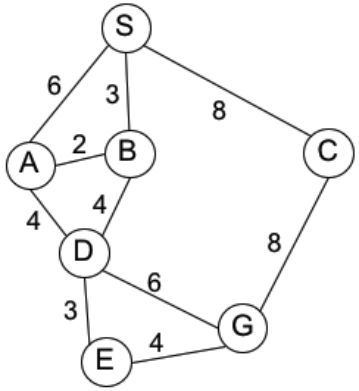
In some order



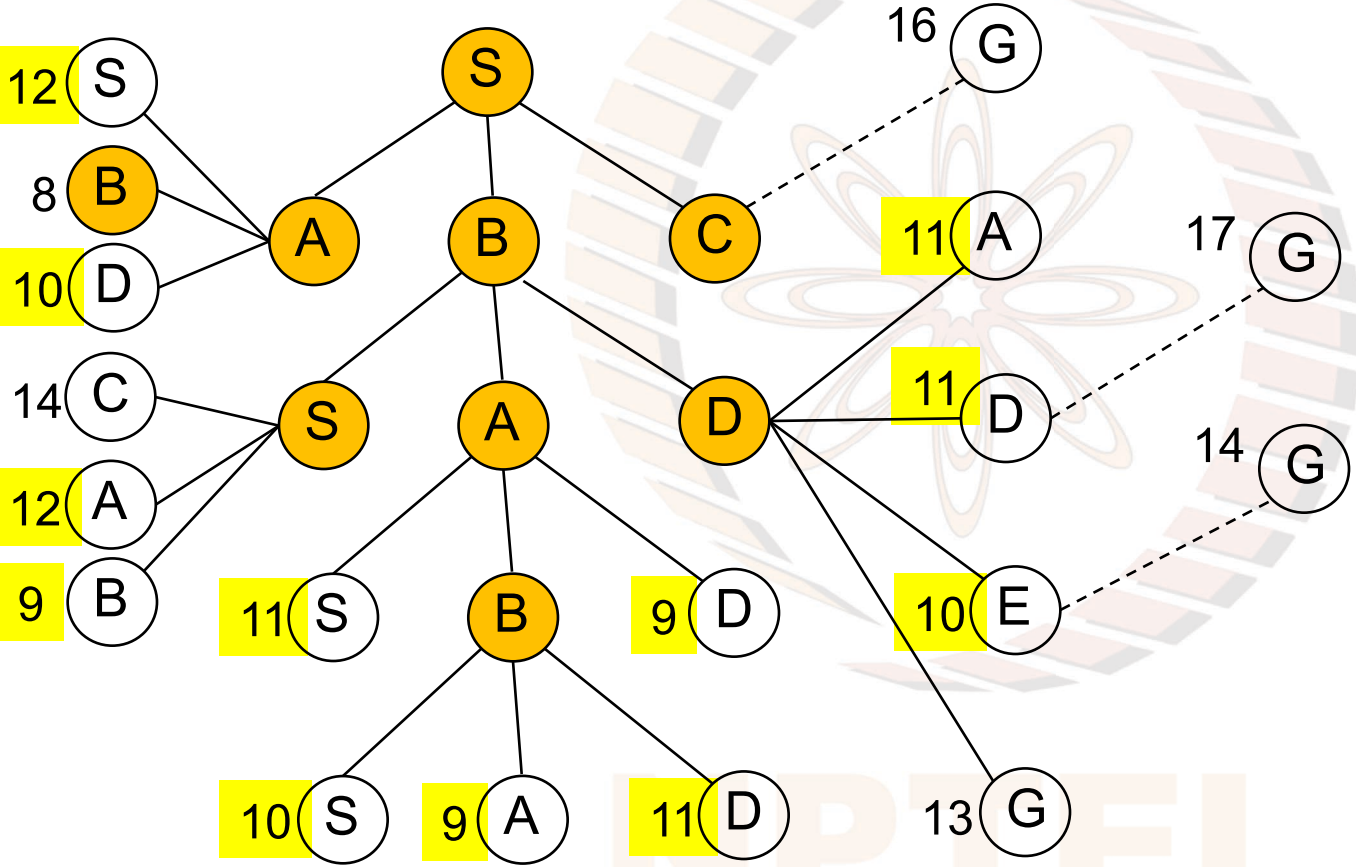


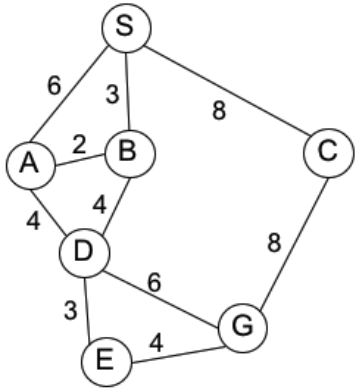
Branch & Bound





Branch & Bound

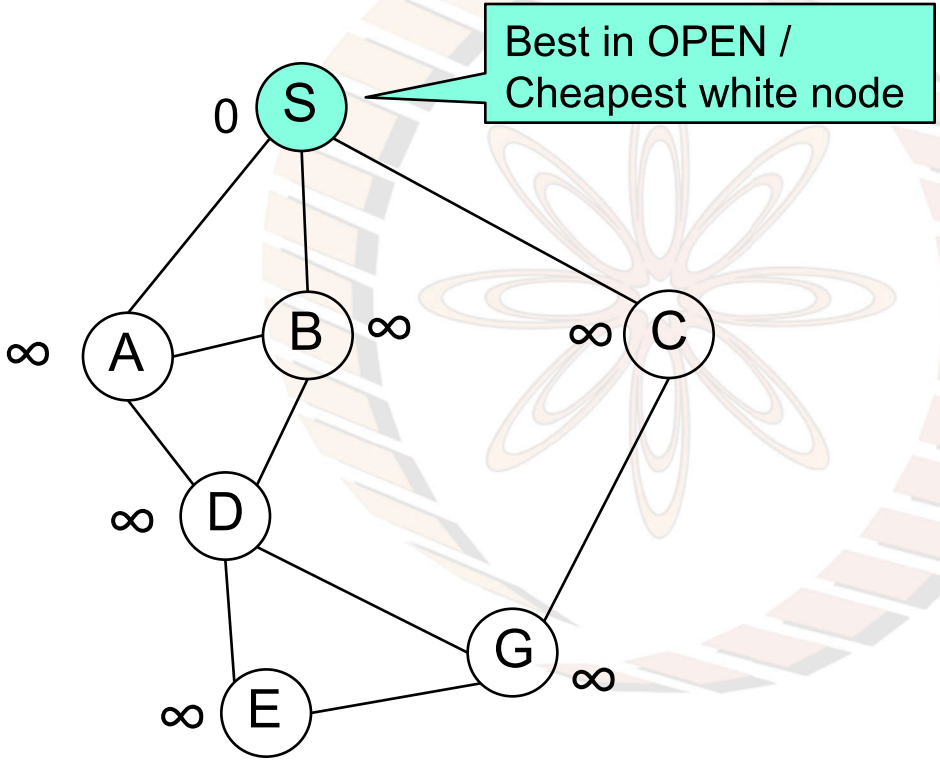


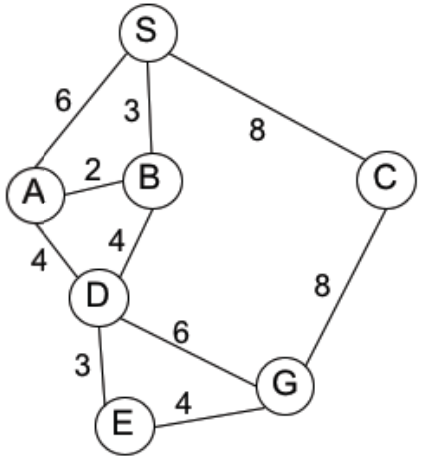


Dijkstra's algorithm

- Dijkstra's algorithm begins by assigning infinite cost estimates to all nodes except the Start node.
- It assigns colour white to all the nodes initially.
- It picks the *cheapest* white node and colours it black.
- Relaxation: Inspect all neighbours of the new black node and check if a cheaper path has been found to them.
- If yes, then update cost of that node, and mark the parent node.

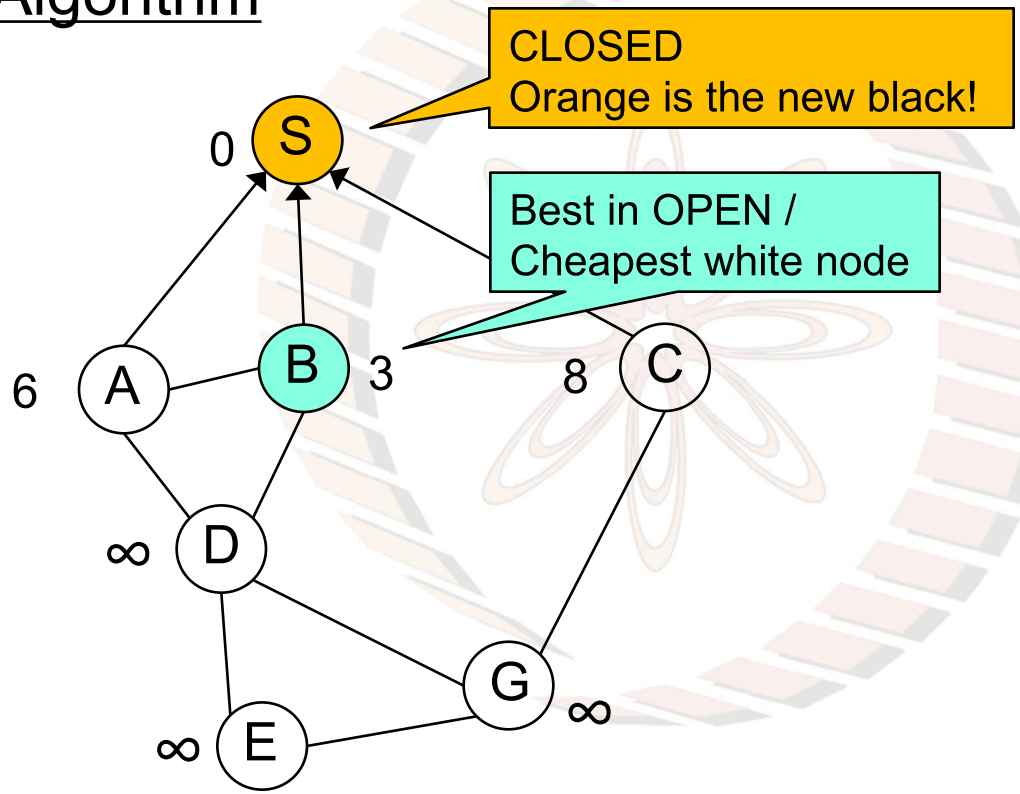
Dijkstra's Algorithm

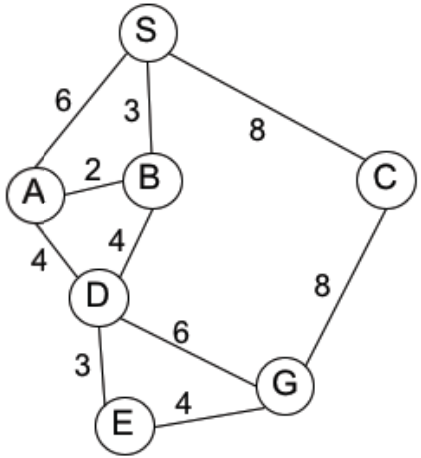




Colour cheapest node and relax the connected edges

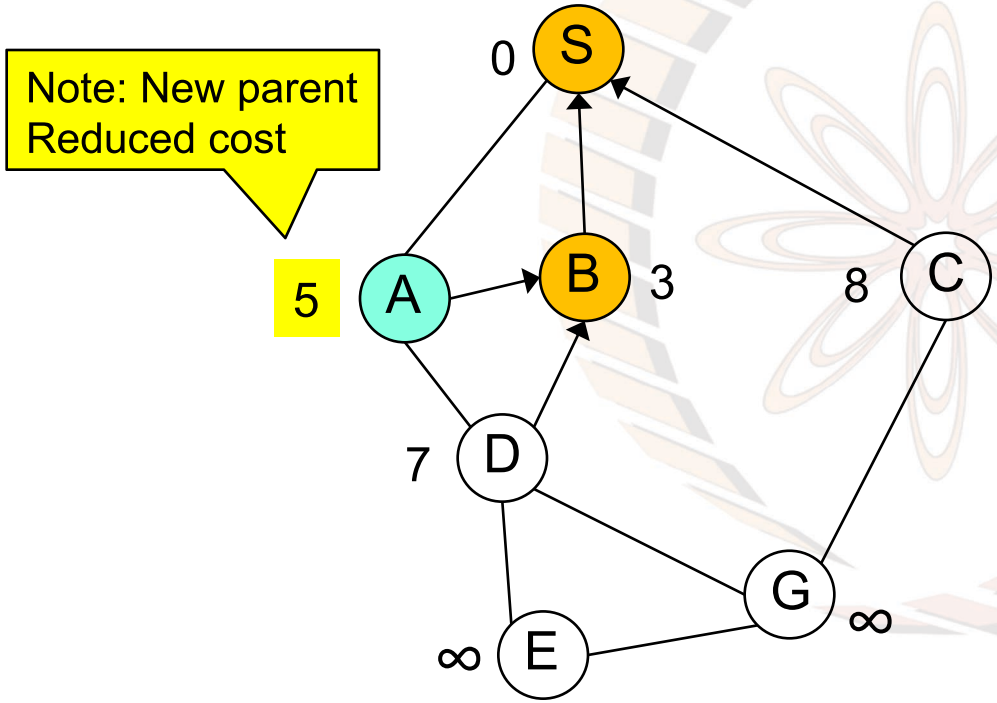
Dijkstra's Algorithm

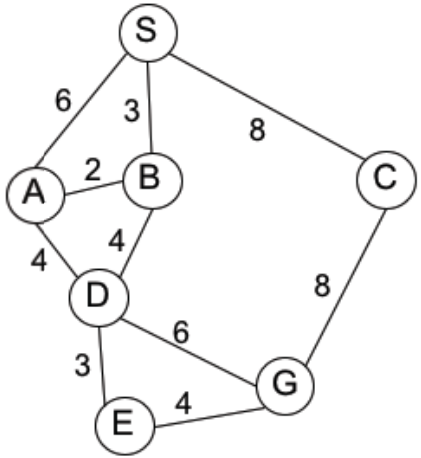




Colour cheapest node and relax the connected edges

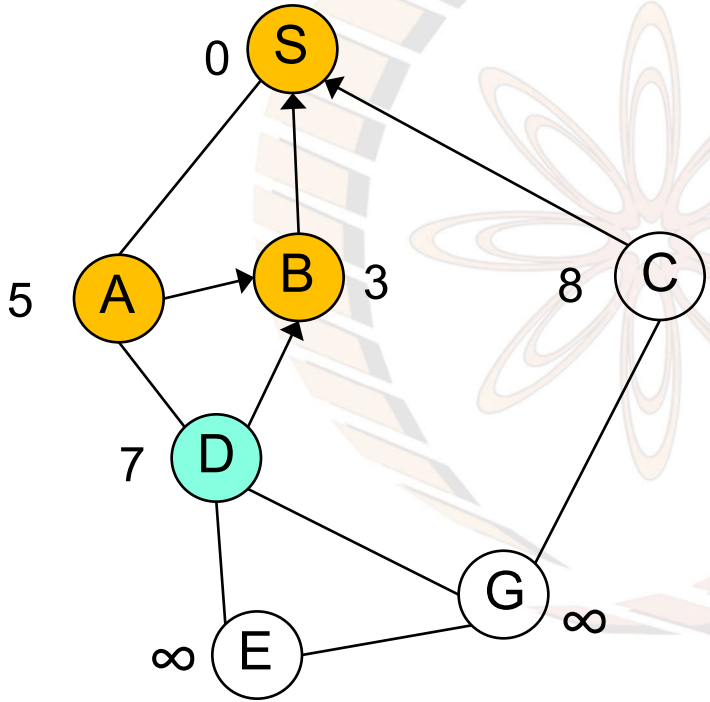
Dijkstra's Algorithm

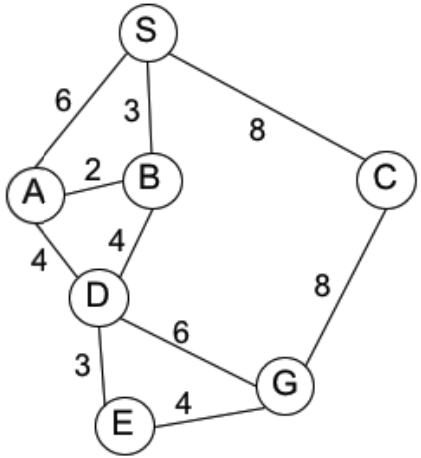




Colour cheapest node and relax the connected edges

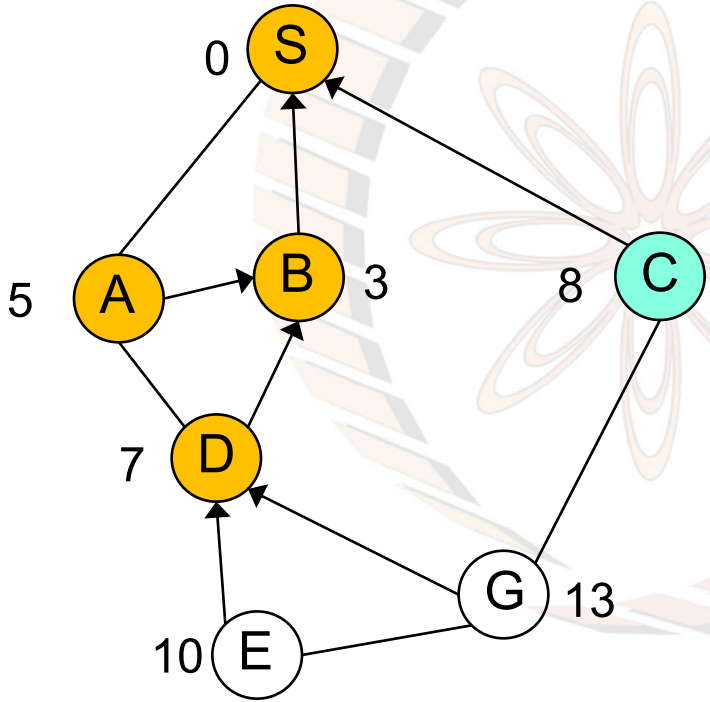
Dijkstra's Algorithm

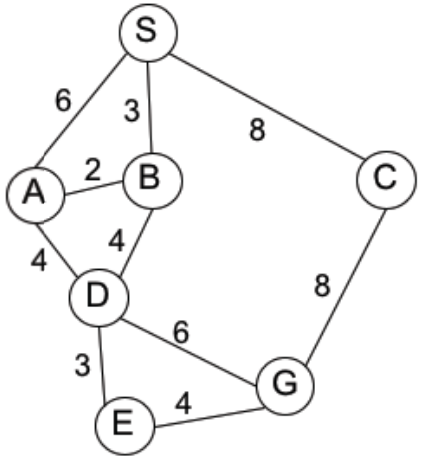




Colour cheapest node and relax the connected edges

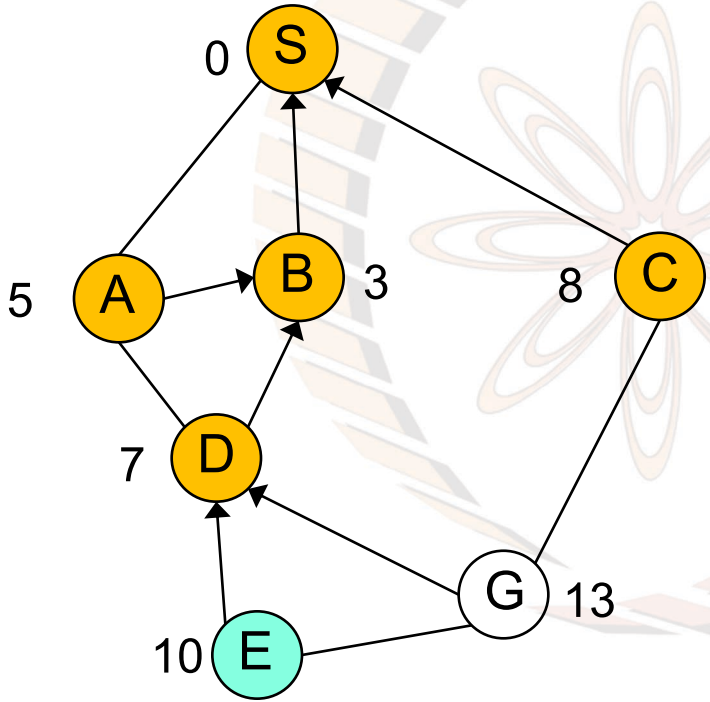
Dijkstra's Algorithm

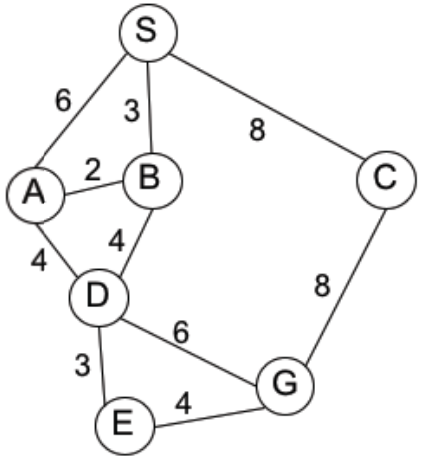




B&B has found a path to G but it is not sure if it is best.

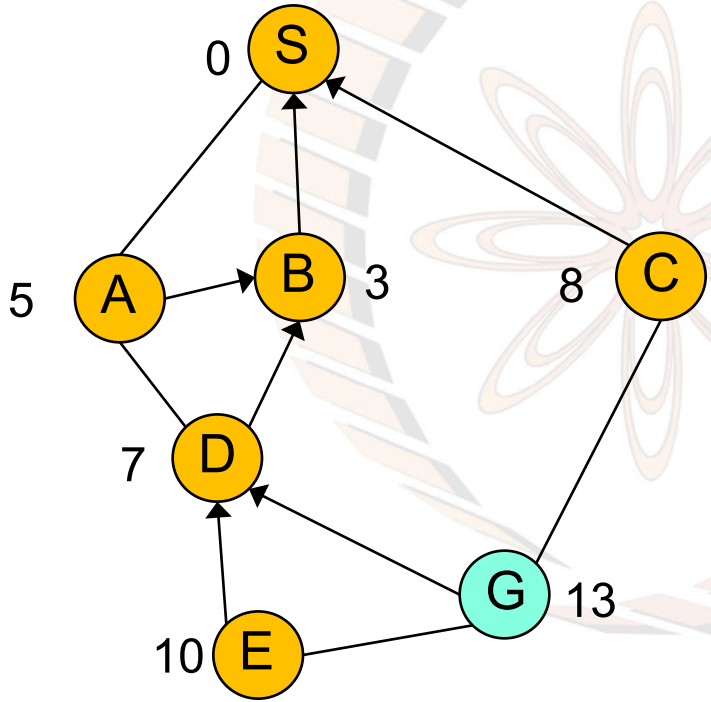
Dijkstra's Algorithm

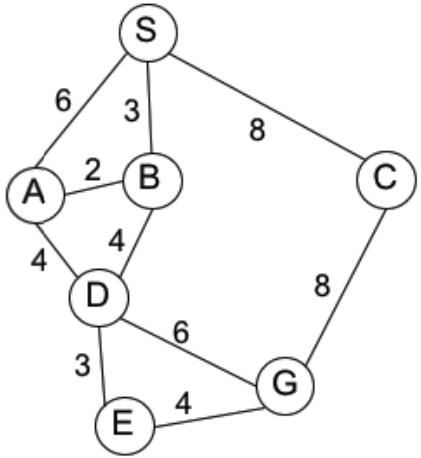




If cost(C,G) were to be less than 5,
then C would become the parent of G

Dijkstra's Algorithm

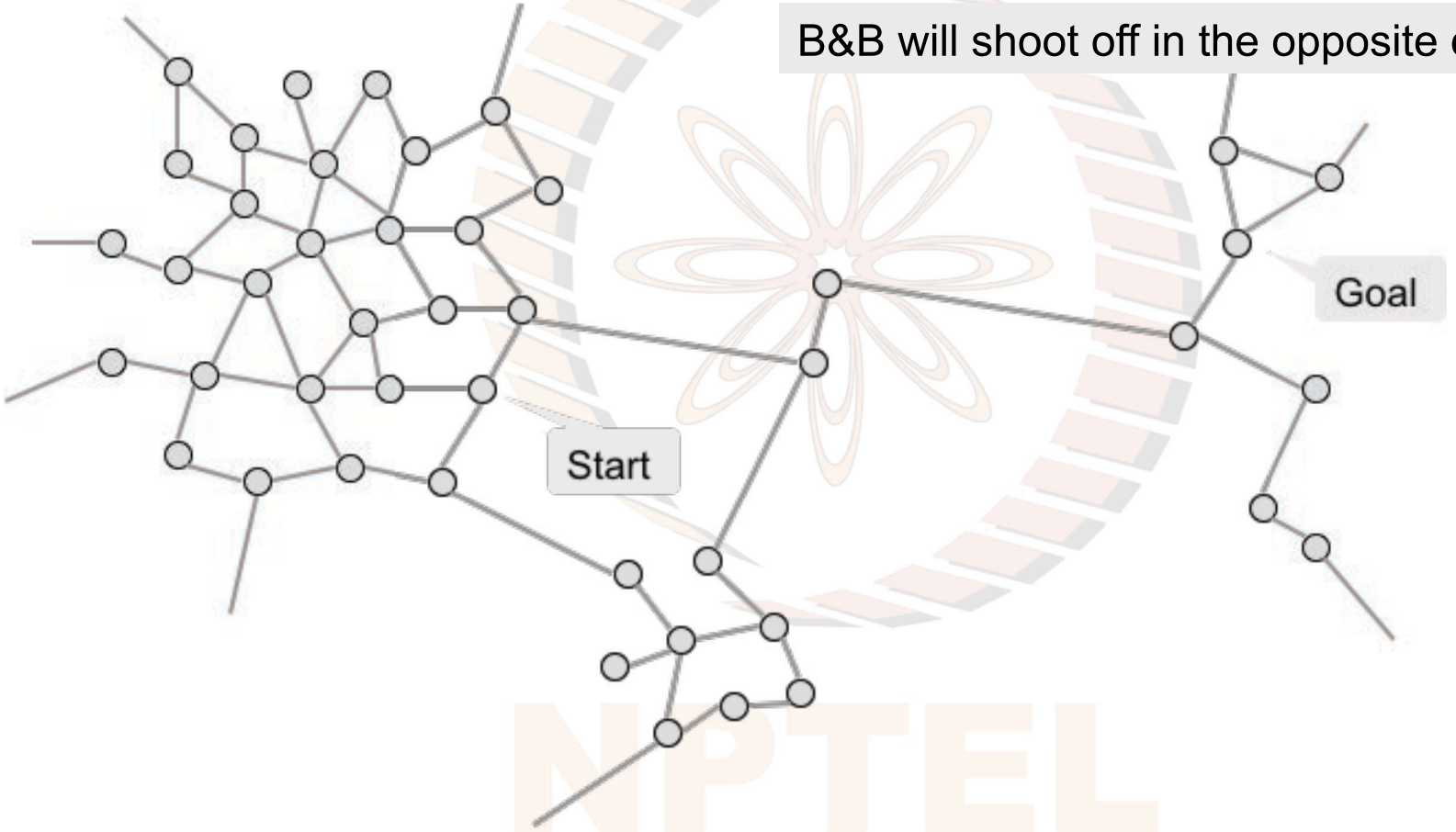




When B&B does pick G it has found the cheapest path to G

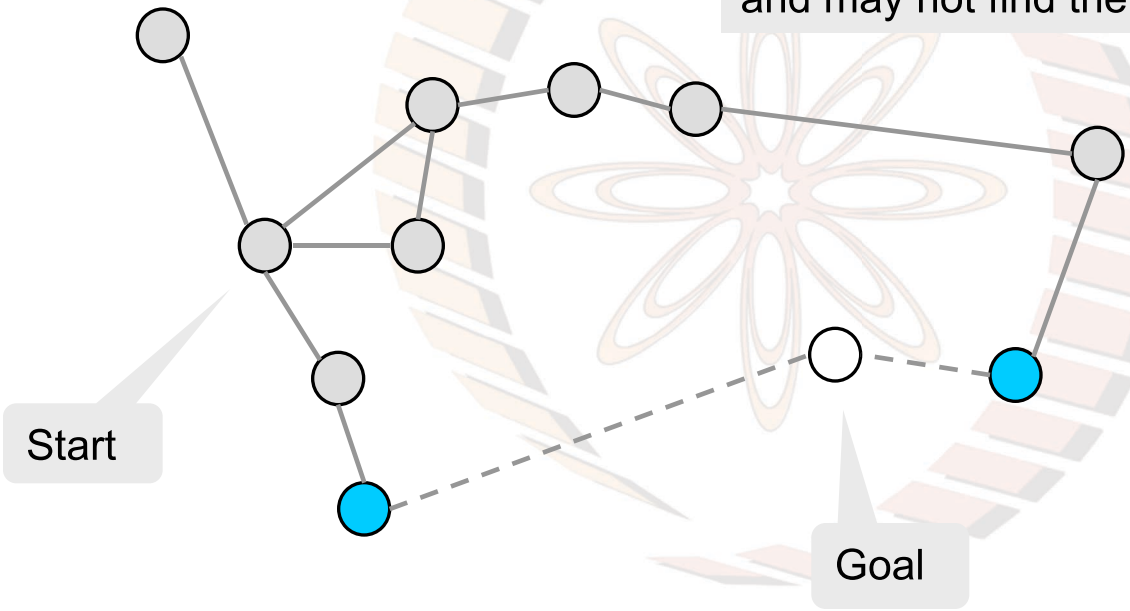
B&B has no sense of direction.....

B&B will shoot off in the opposite direction!



Best First only looks ahead

Best First only looks at the distance to goal, and may not find the shortest path



Algorithm A* combines the best features of both!

Next
Algorithm A*