Informed Search

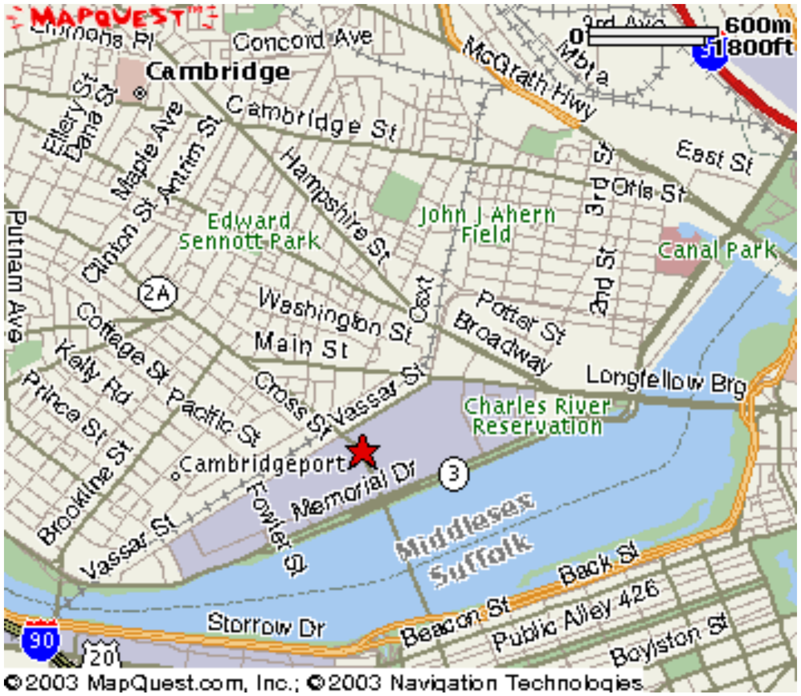
Brian C. Williams
16.410-13
September 23rd, 2015

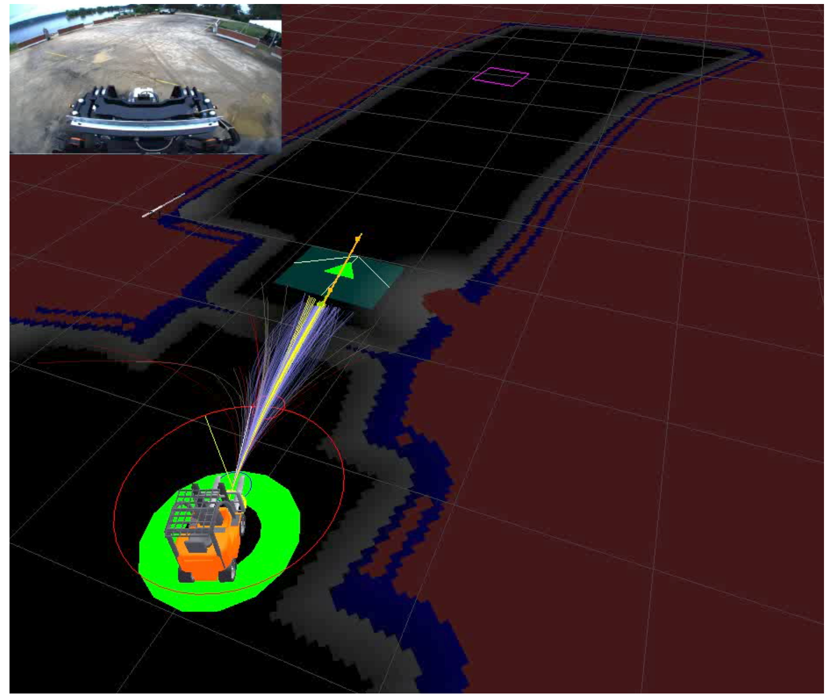
Slides adapted from:
6.034 Tomas Lozano Perez, Winston,
David Hsu, and
Russell and Norvig AIMA

Assignment

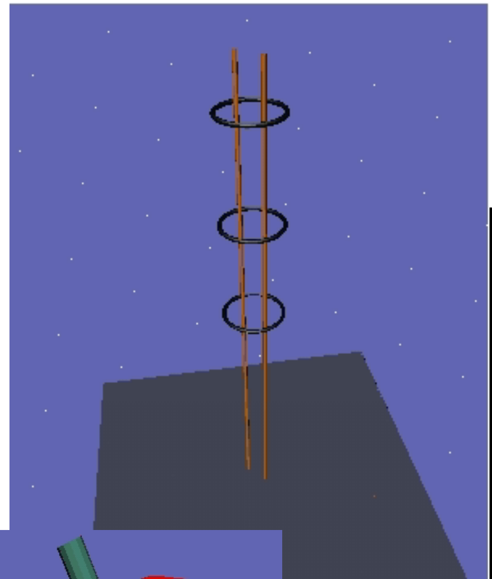
- Remember:
  - PS #2, due Today at midnight, Wednesday, September 23rd, 2015.
  - Problem Set #3, out Today, due Wednesday, September 30th, 2015.
  -
- Reading:
  - Today: Informed search and exploration: AIMA Ch. 4.1-2, Ch. 25.4.
    Computing Shortest Paths: Cormen, Leiserson & Rivest, (opt.)
    "Introduction to Algorithms" Ch. 25.1-.2.
  - Wed: Activity Planning: [AIMA] Ch.10 & 11.

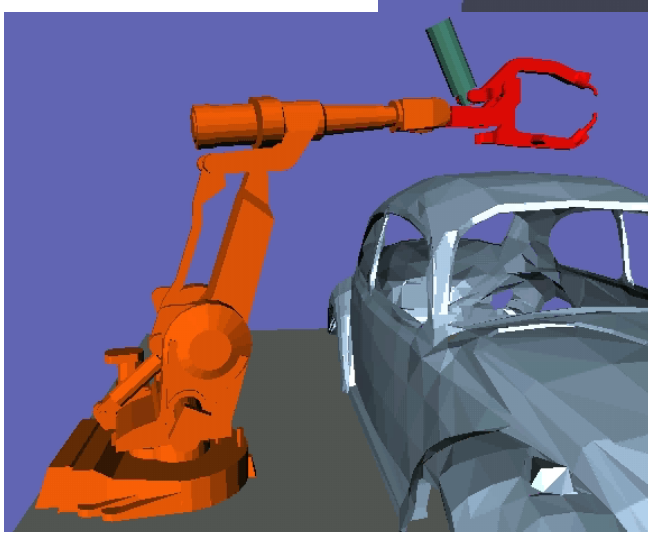
Motion planning



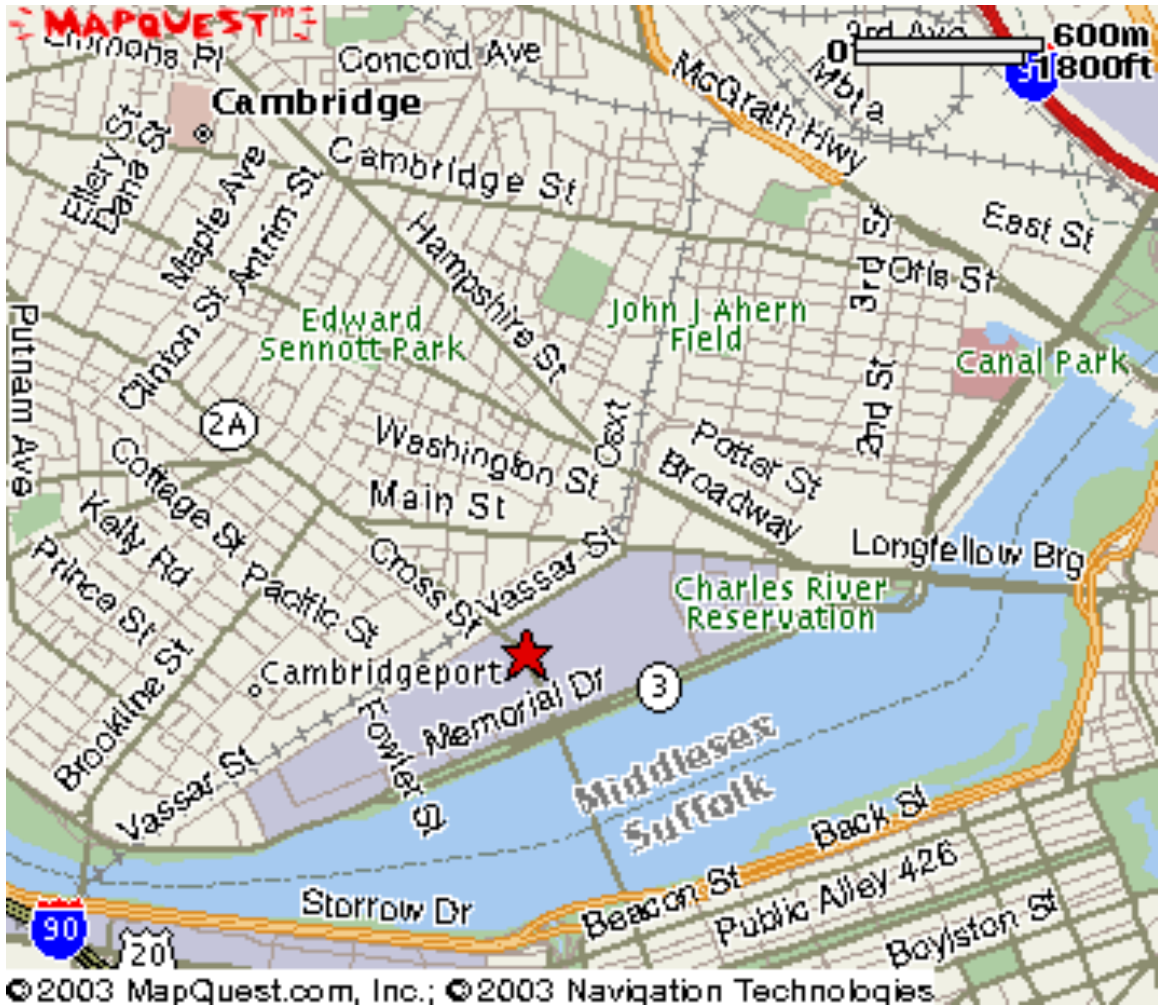


Motion planning



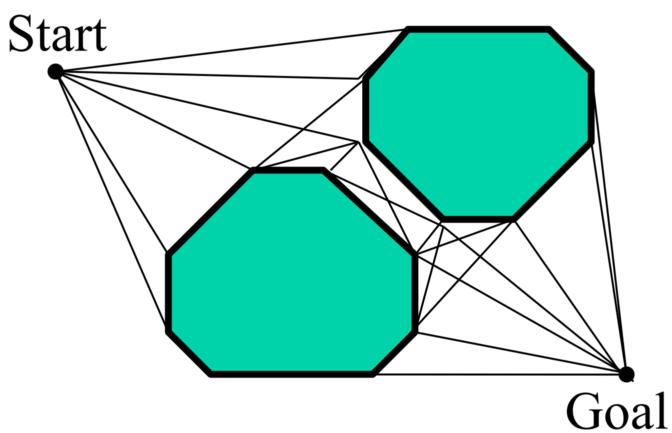


Review: Roadmaps are an effective state space abstraction

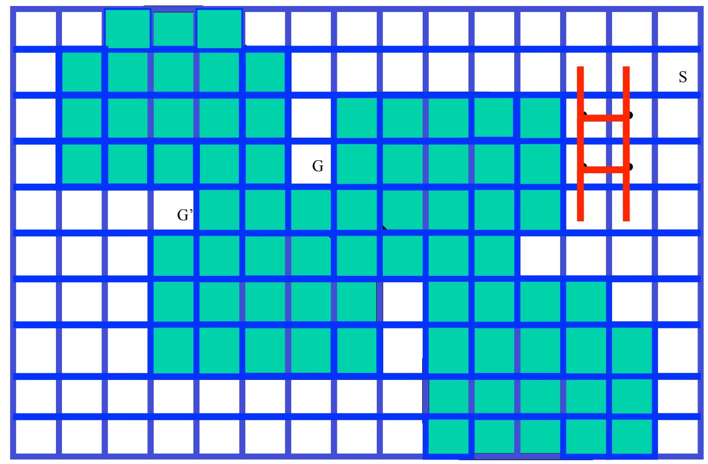


Constructing Road Maps

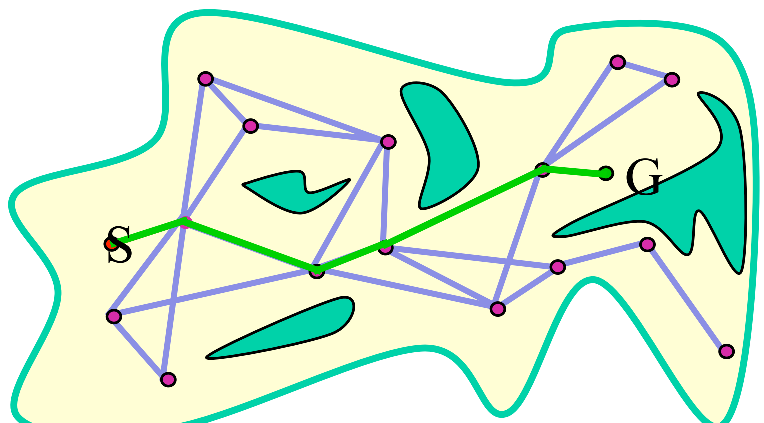
Configuration Spaces
And Visibility Graphs



Cell Decompositions



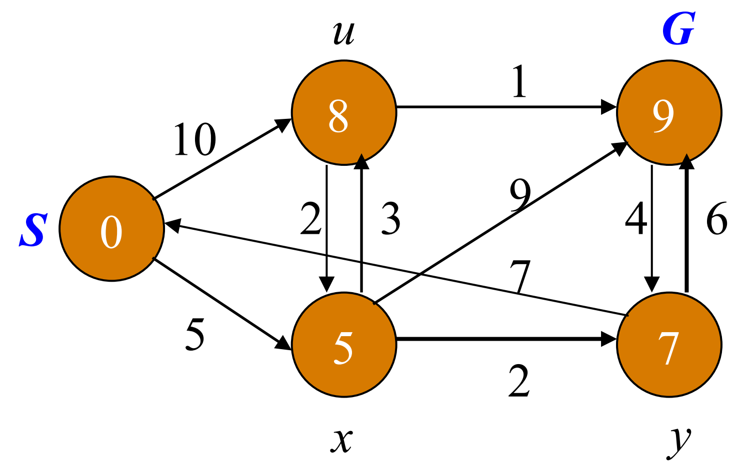
Probabilistic Road Maps



Finding A Shortest Path

Input: <gr, w, S, G>, where

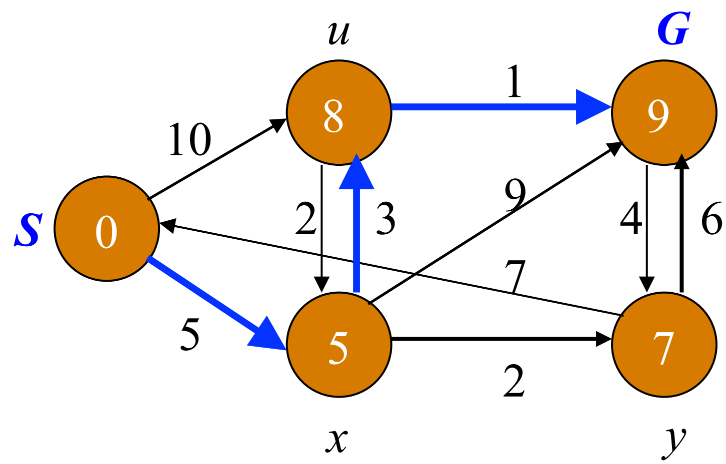
- gr is a (directed) graph <V,E> with
- weight function *w: VxV* ➔ *R*,
- S ∈ V is the Start and G ∈ V is the Goal.



Finding A Shortest Path

Input: <gr, w, S, G>, where

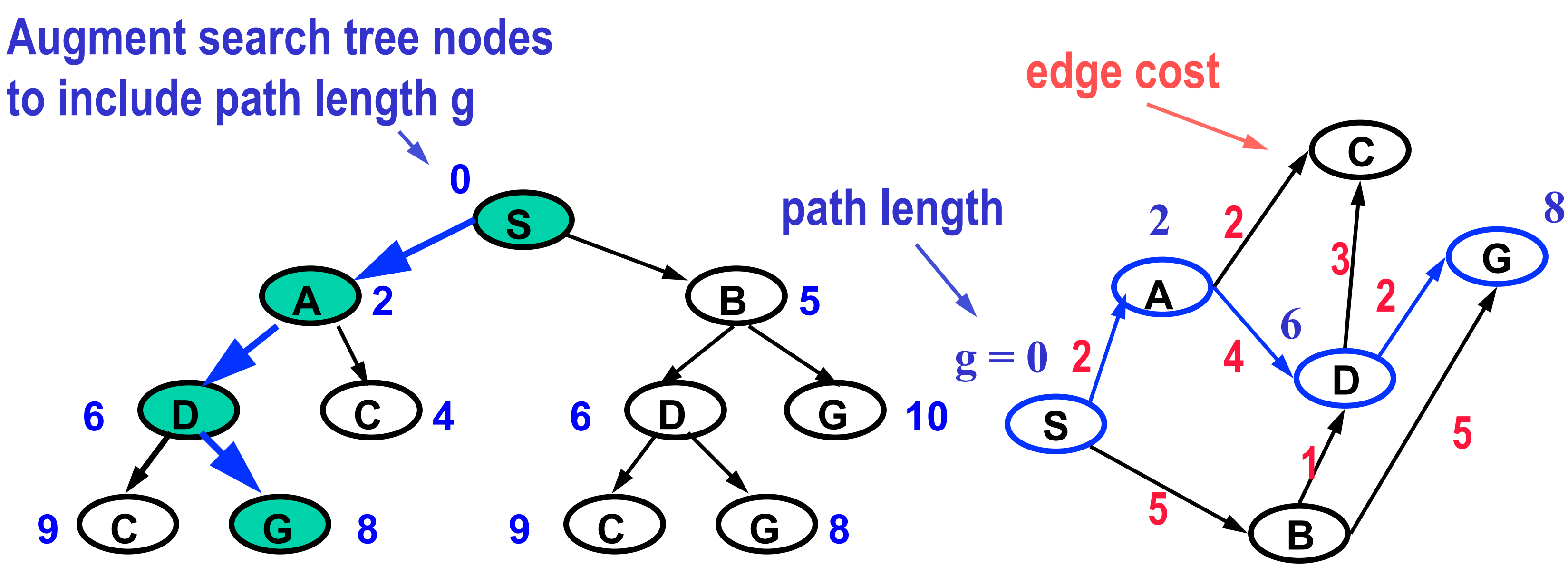
- gr is a (directed) graph <V,E> with
- weight function *w: VxV* ➔ *R*,
- S ∈ V is the Start and G ∈ V is the Goal.



Output:

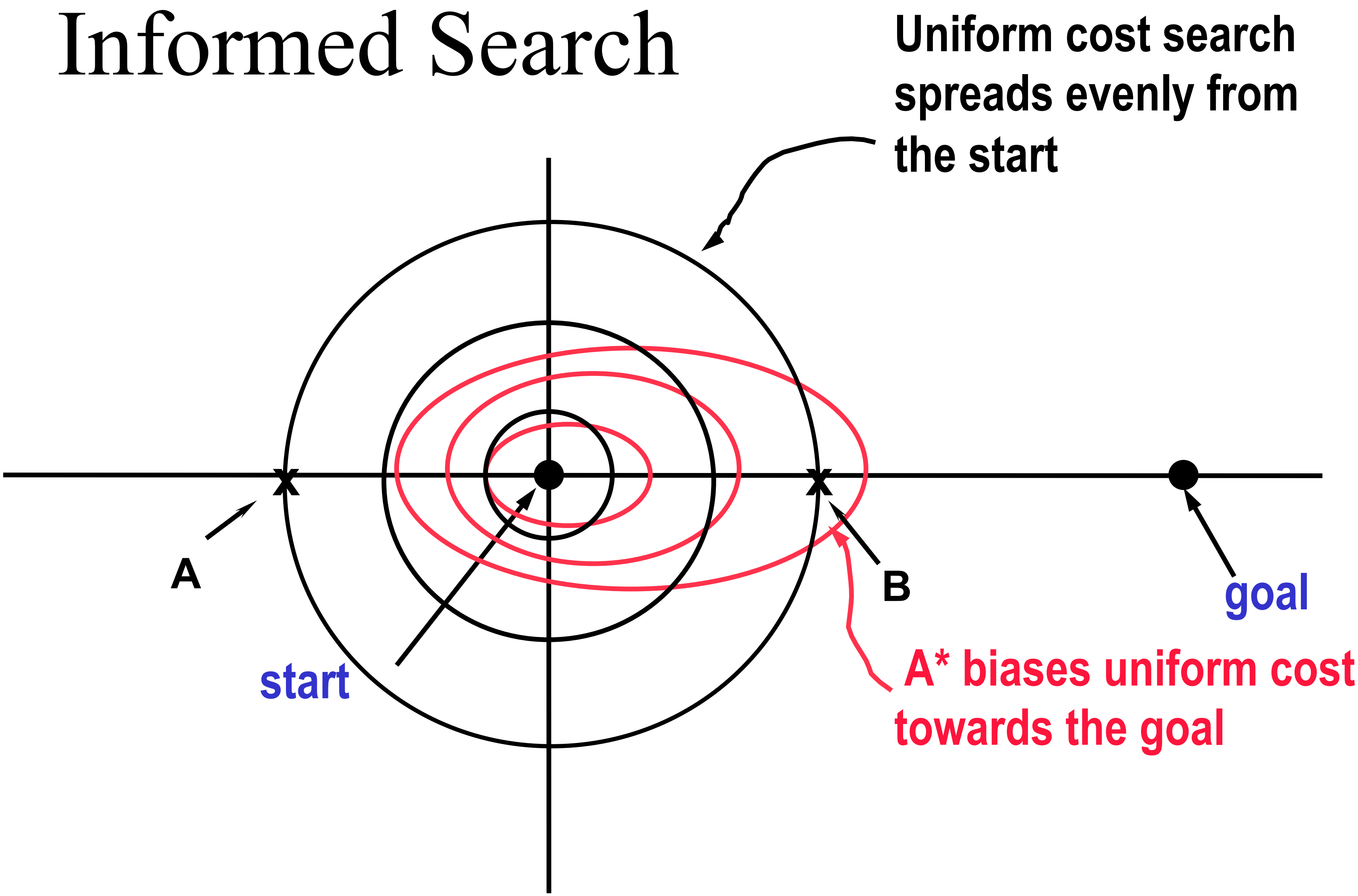
A simple path P = <v_1, v_2 …v_n> from S to G, with the shortest path weight g = δ(S,G), and its corresponding weight.

Optimal Search



Problem: Find the path to the goal G with the shortest path length g.

Informed Search



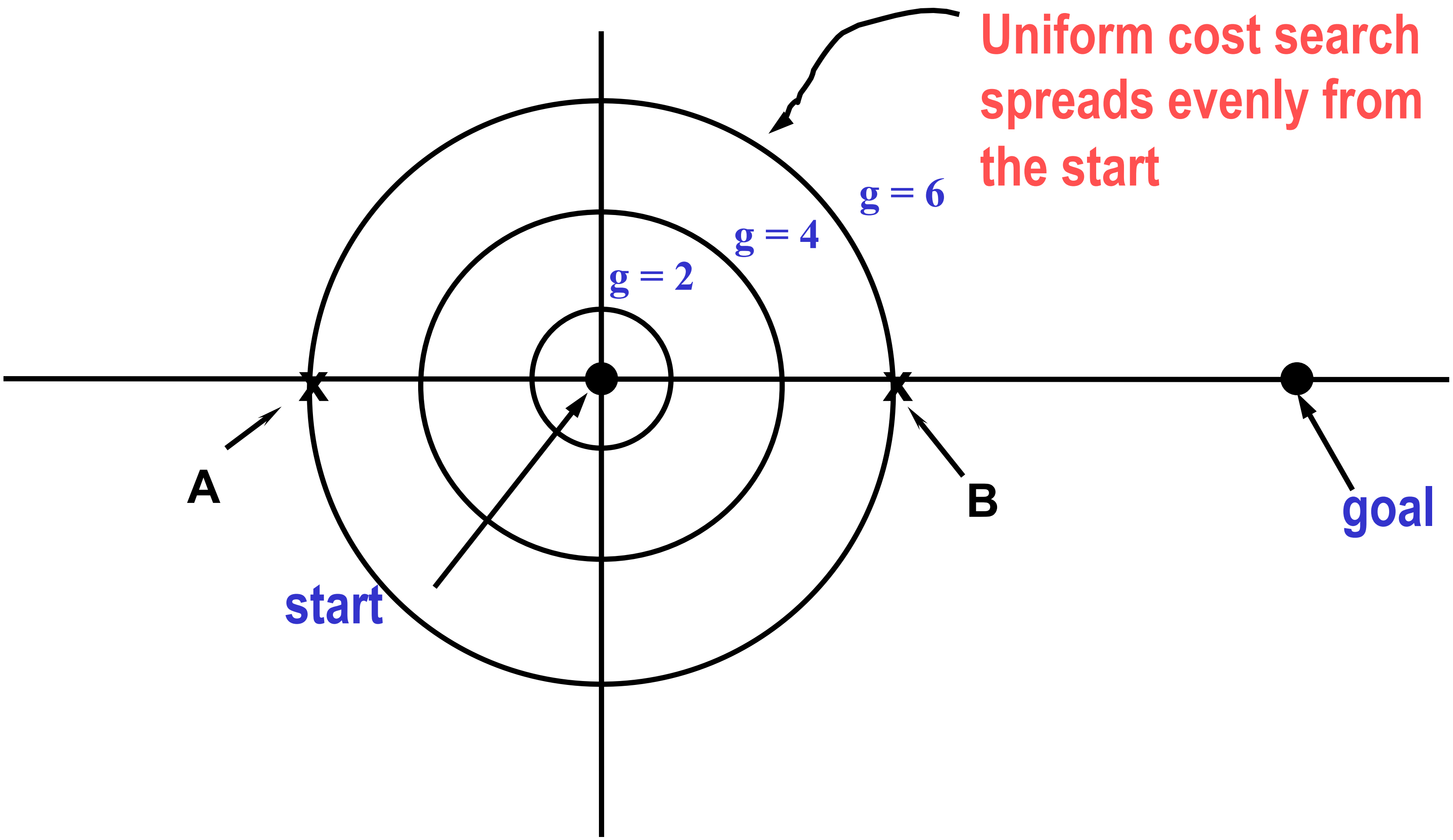
Classes of Search

Blind (uninformed)	**Depth-First** **Breadth-First** **Iterative-Deepening**	Systematic exploration of whole tree until the goal is found.
Best-first (informed)	**Uniform-cost** **Greedy** **A***	**Uses path "length" measure to find "shortest" path.**
Bounding	**Branch and Bound**	Prunes suboptimal branches.
	Alpha/Beta	Prunes options that the adversary rules out.
Variants	**Hill-Climbing (w backup)** **Beam** **IDA***	

Classes of Search

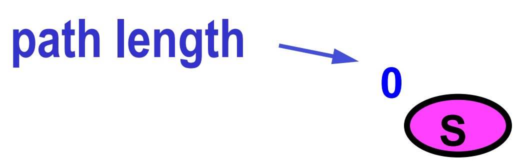
Blind	**Depth-First**	Systematic exploration of whole tree
(uninformed)	**Breadth-First**	until the goal is found.
	Iterative-Deepening	

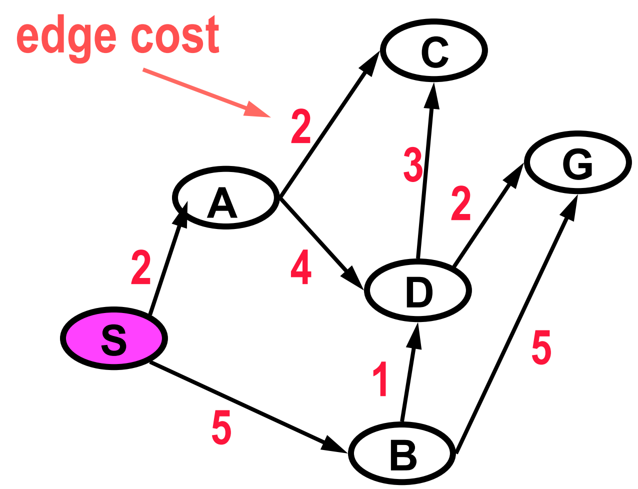
Best-first	**Uniform-cost**	**Uses path "length" measure to**
	Greedy	**find "shortest" path.**
	A*	



Does uniform cost search find the shortest path? **Yes, Optimal**

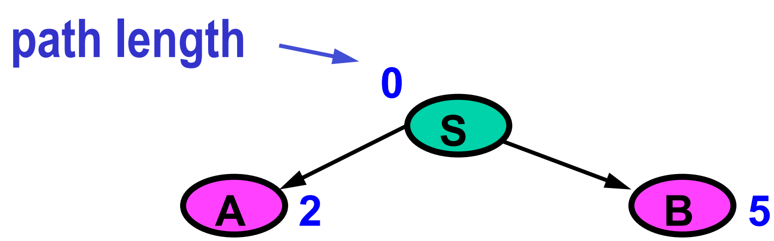
Uniform Cost

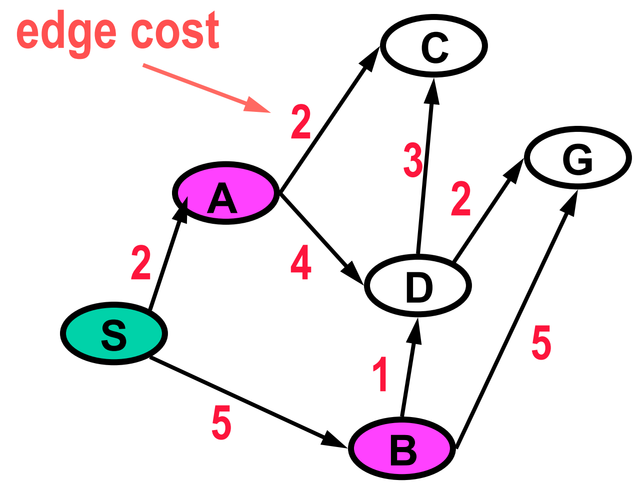




Enumerates partial paths in order of increasing path length g.

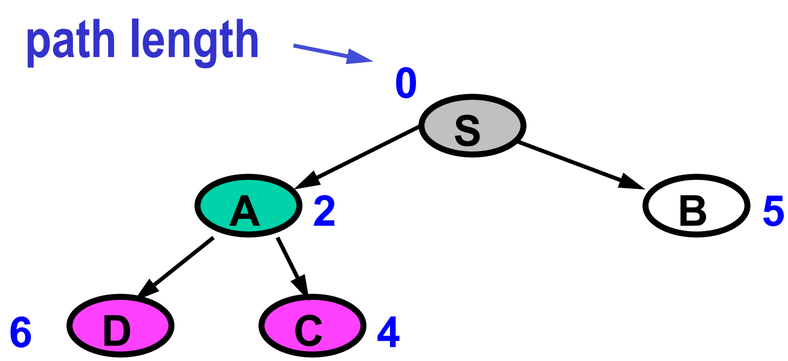
Uniform Cost

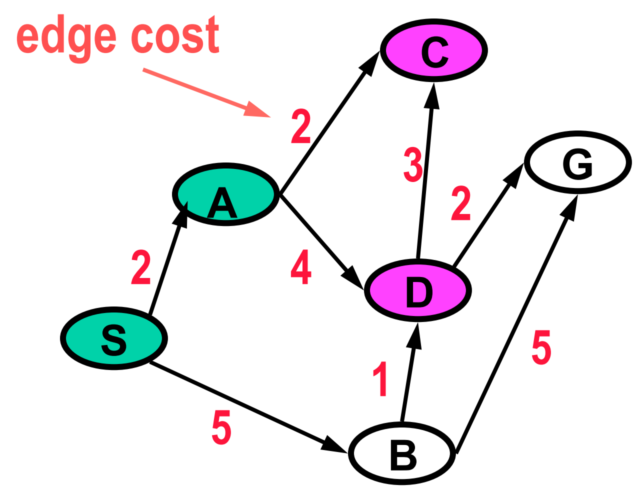




Enumerates partial paths in order of increasing path length g.

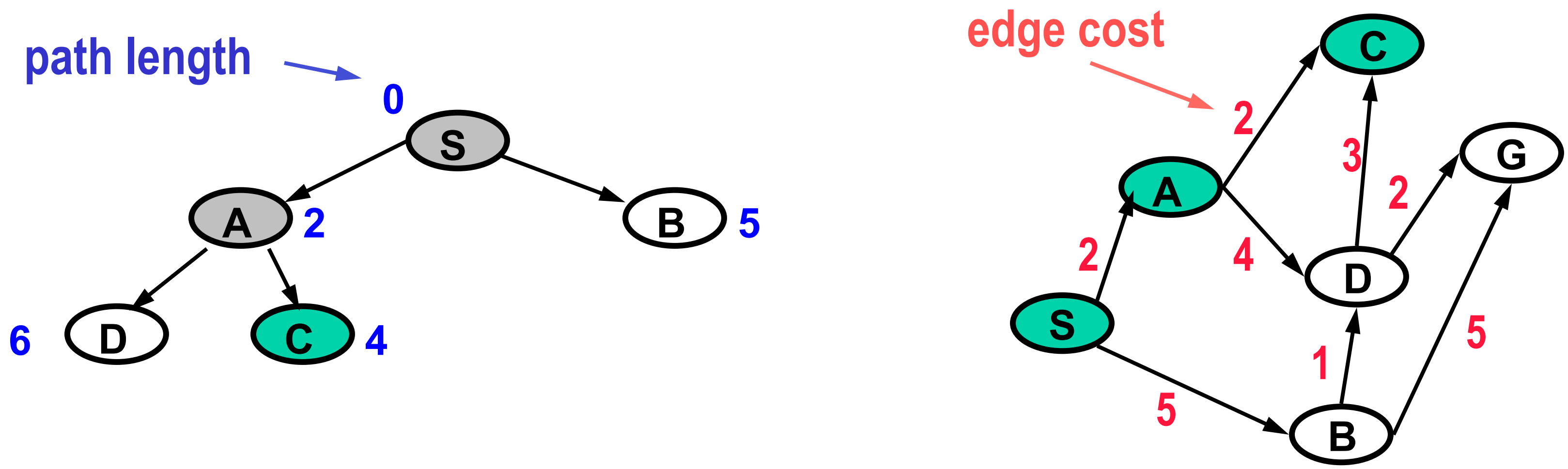
Uniform Cost





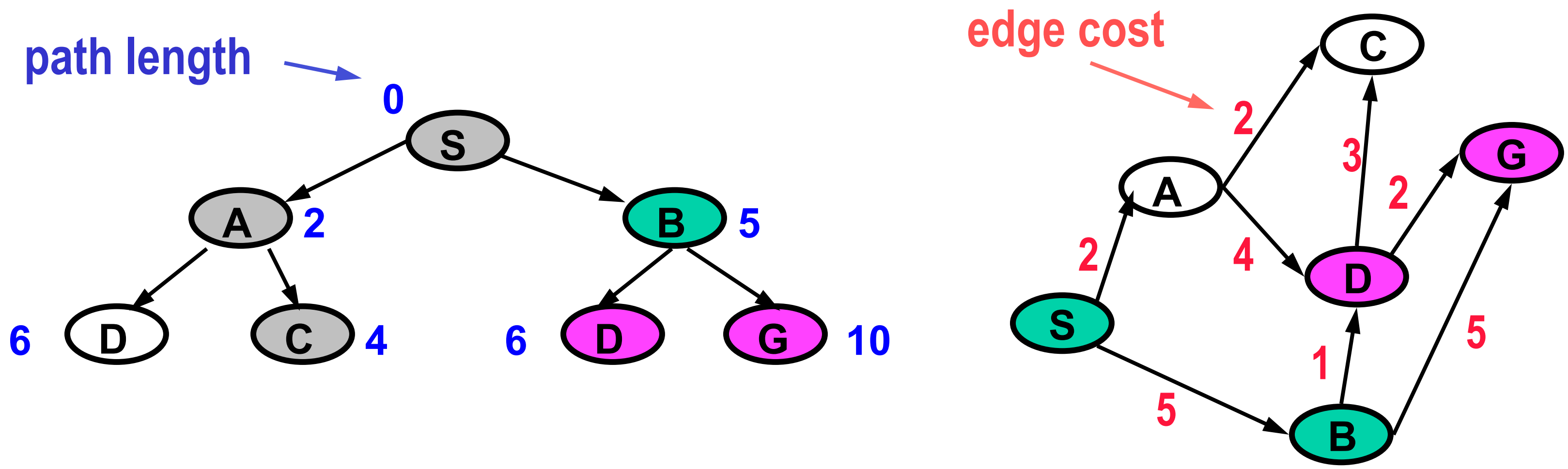
Enumerates partial paths in order of increasing path length g.

Uniform Cost



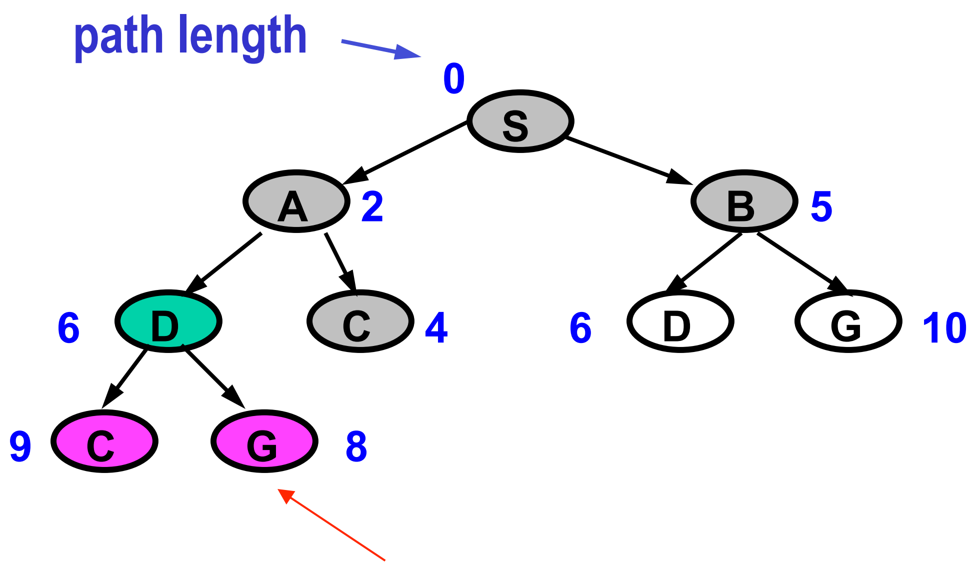
Enumerates partial paths in order of increasing path length g.

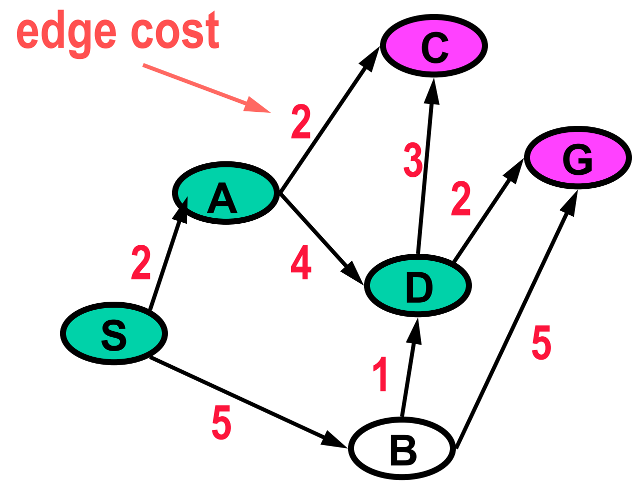
Uniform Cost



Enumerates partial paths in order of increasing path length g.

Uniform Cost

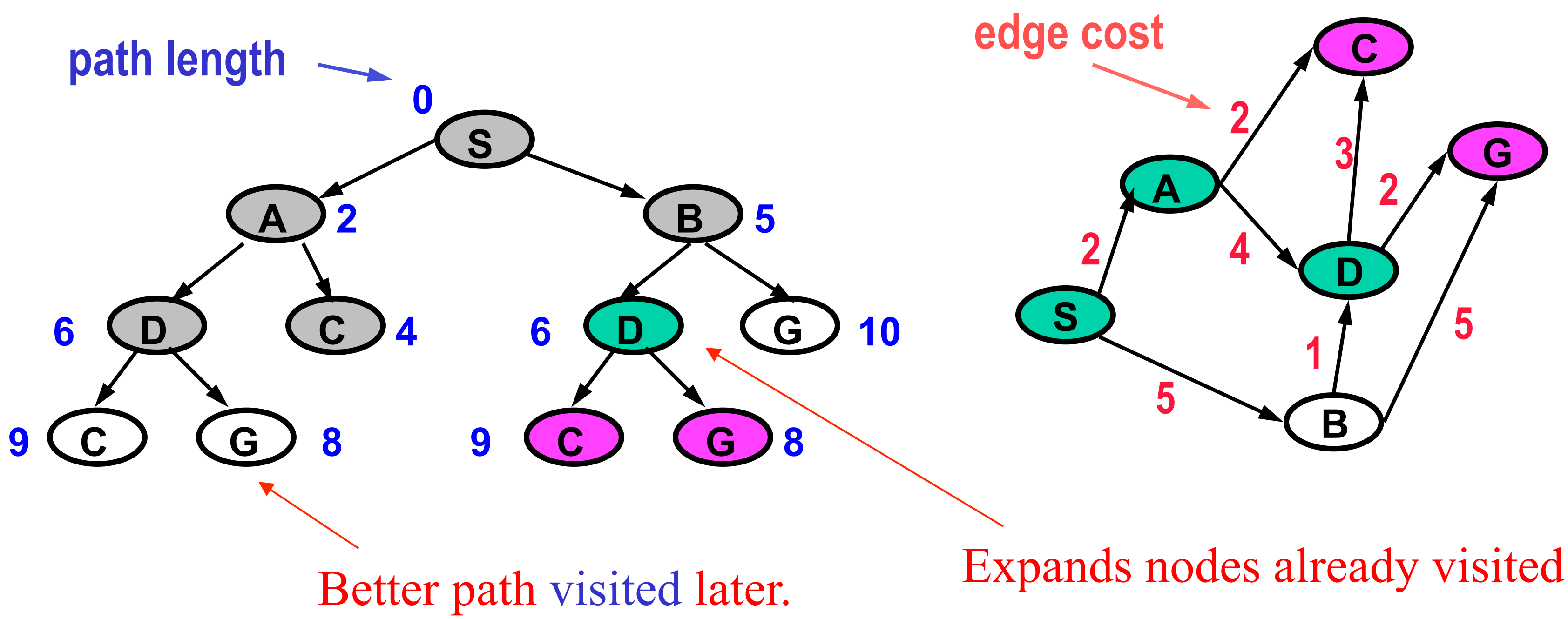




Better path visited later.

Enumerates partial paths in order of increasing path length g.

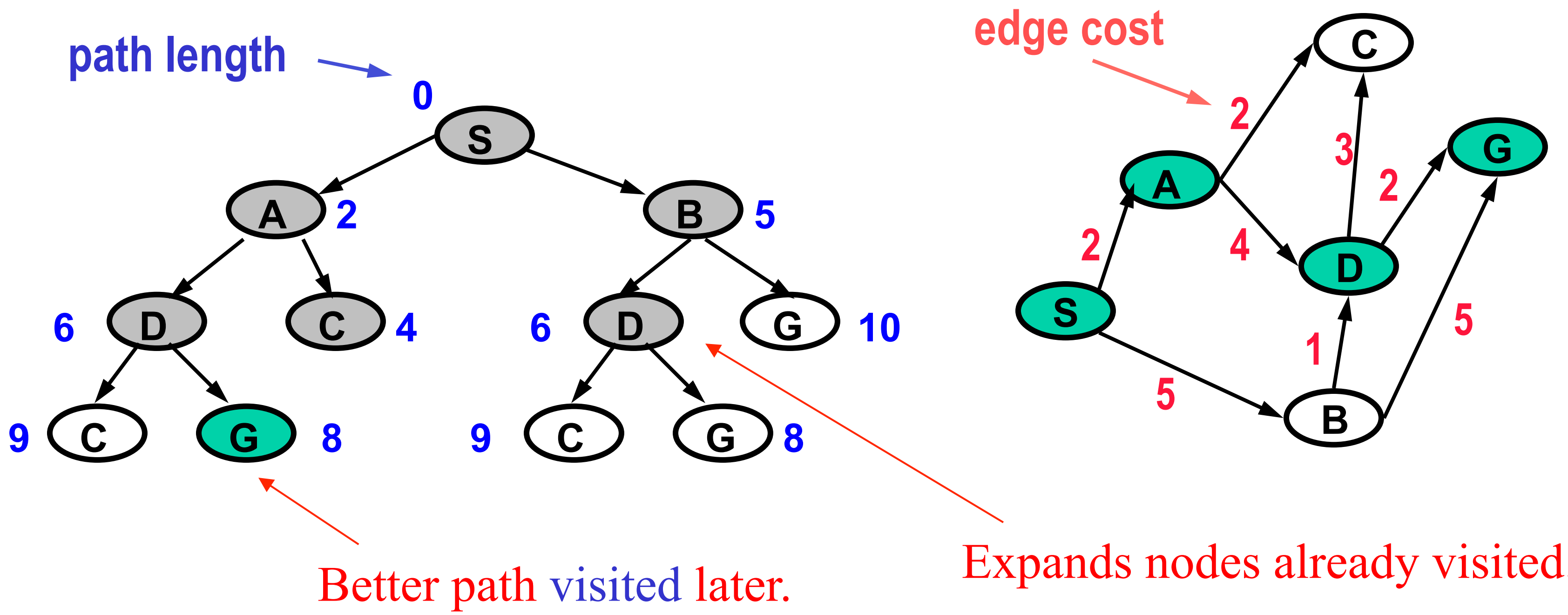
Uniform Cost



Enumerates partial paths in order of increasing path length g.

May expand vertex more than once.

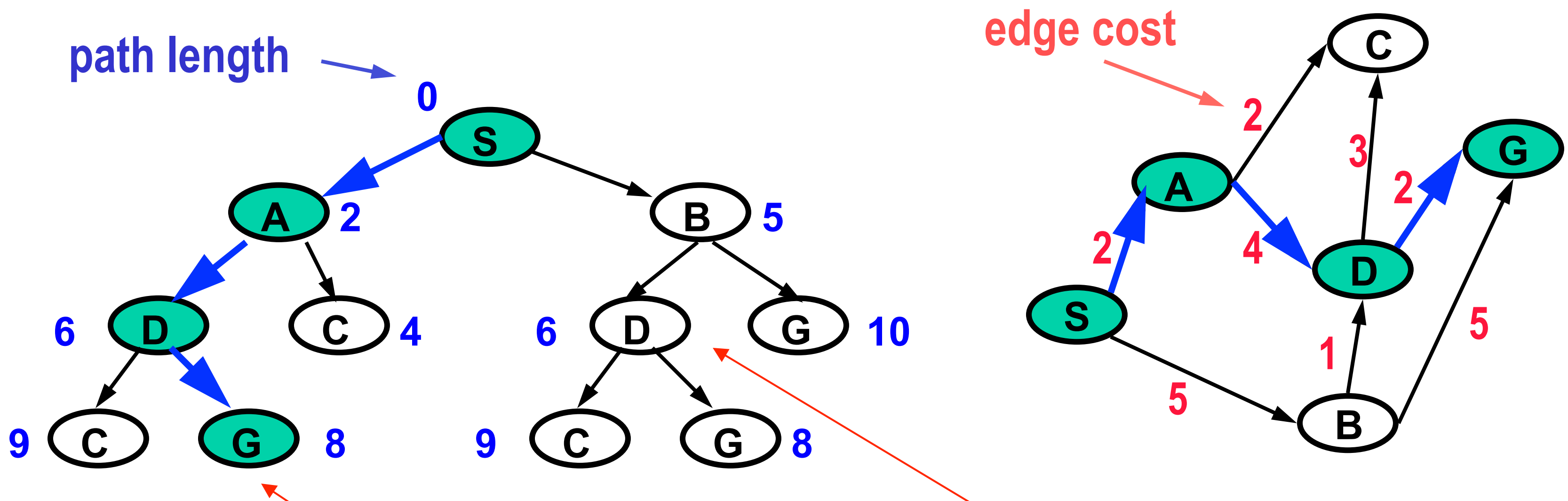
Uniform Cost



Enumerates partial paths in order of increasing path length g.

May expand vertex more than once.

Uniform Cost



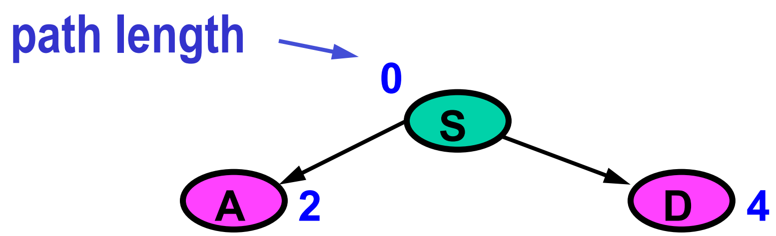
Best path **expanded** first.

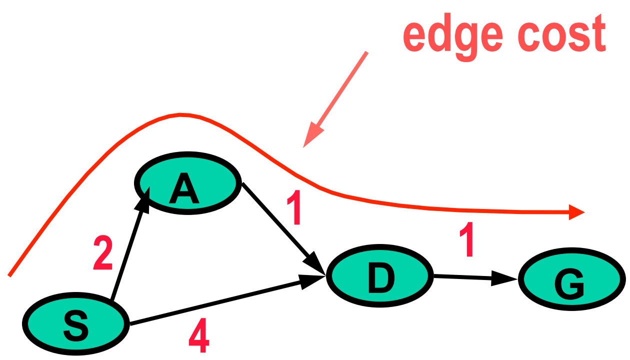
Expands nodes already visited.

Enumerates partial paths in order of increasing path length g.

May expand vertex more than once.

Why Expand a Vertex More Than Once?

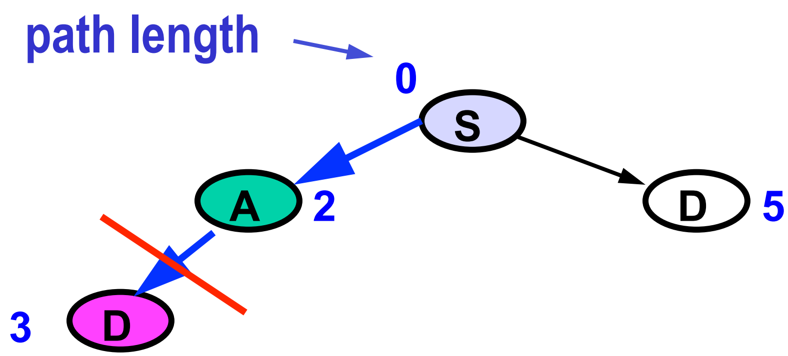


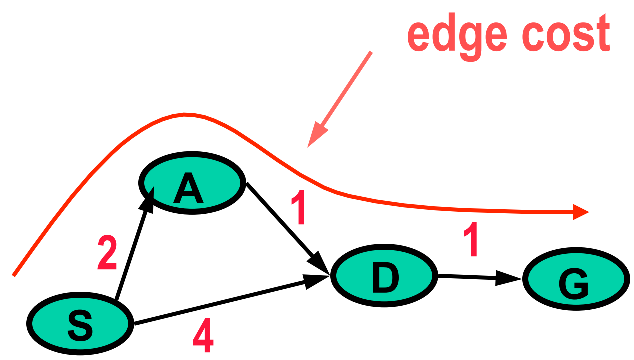


- The shortest path from S to G is (G D A S).
- D is reached first using path (D S).

Suppose we expand only the first path that visits each vertex X?

Why Expand a Vertex More Than Once?

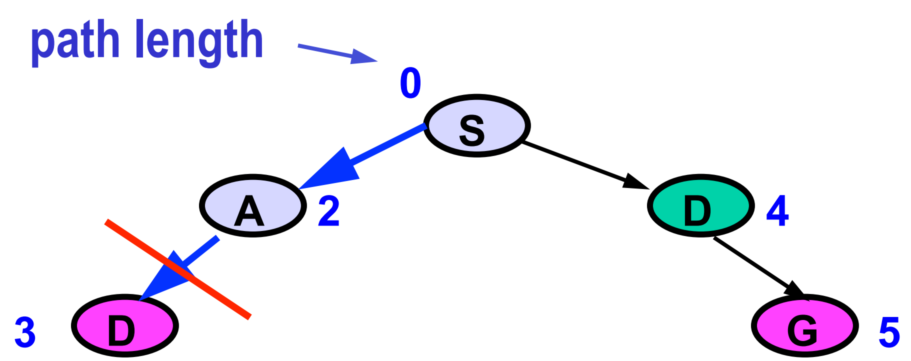


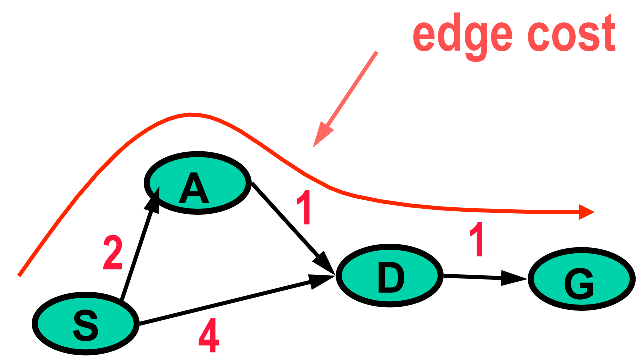


- The shortest path from S to G is (G D A S).
- D is reached first using path (D S).
- This prevents path (D A S) from being expanded.

Suppose we expand only the first path that visits each vertex X?

Why Expand a Vertex More Than Once?

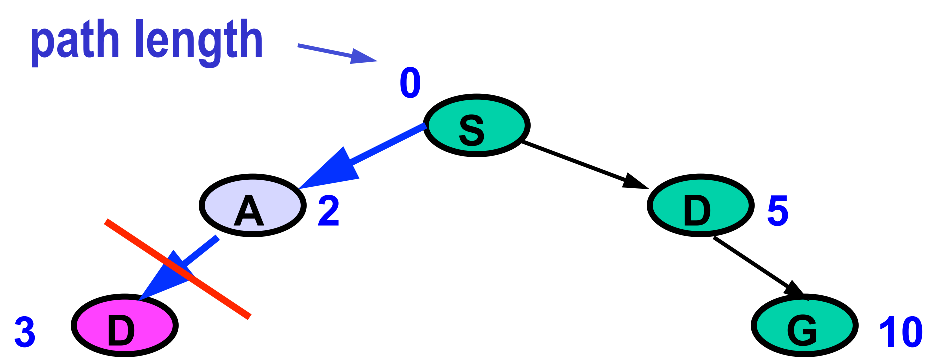


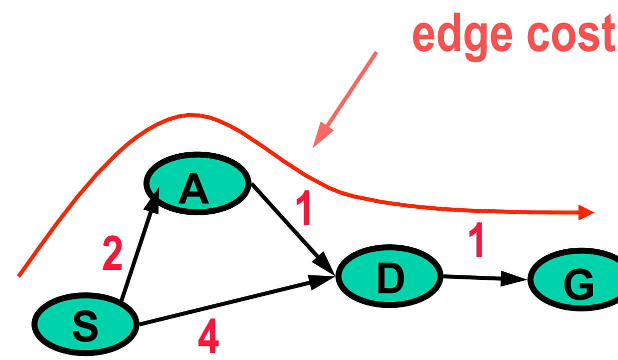


- The shortest path from S to G is (G D A S).
- D is reached first using path (D S).
- This prevents path (D A S) from being expanded.

Suppose we expand only the first path that visits each vertex X?

Why Expand a Vertex More Than Once?





- The shortest path from S to G is (G D A S).
- D is reached first using path (D S).
- This prevents path (D A S) from being expanded.
- The suboptimal path (G D S) is returned.

Suppose we expanded only the first path that visits each vertex X?

⇨ **Solution: Eliminate the Visited List.**

Generic Search Algorithm

Let gr be a Graph.
Let S be the start vertex in gr.

Let Q be a list of simple partial paths in gr.
Let G be a Goal vertex in gr.

1. **Initialize Q with partial path (S) as only entry; set Visited = ( );**
2. **If Q is empty, fail; Else, pick partial path N from Q;**
3. **If head(N) = G, return N ; (we've reached the goal!)**
4. **(Otherwise) Remove N from Q;**
5. **Find all children of head(N) (its neighbors in gr) not in Visited and create all the one-step extensions of N to each child;**
6. **Add to Q all the extended paths;**
7. **Add children of head(N) to Visited;**
8. **Go to Step 2.**

Uniform Cost Search Algorithm

Let gr be a weighted Graph.

Let S be the start vertex in gr.

Let g be the path weight from S to N.

Let Q be a list of simple partial paths in gr.

Let G be a Goal vertex in gr.

1. **Initialize Q with partial path (S) as only entry; ~~set Visited = ( );~~**
2. **If Q is empty, fail; Else, pick partial path N from Q with best g;**
3. **If head(N) = G, return N ; (we' ve reached the goal!)**
4. **(Otherwise) Remove N from Q;**
5. **Find all children of head(N) (its neighbors in Gr) ~~not in Visited~~ and create all the one-step extensions of N to each child;**
6. **Add to Q all the extended paths;**
7. ~~**Add children of head(N) to Visited;**~~
8. **Go to Step 2.**

Brian Williams, Fall 15

Implementing the Search Strategies

Depth-first:

Pick first element of Q — Uses visited list

Add path extensions to front of Q

Breadth-first:

Pick first element of Q — Uses visited list

Add path extensions to end of Q

Uniform-cost:

Pick first element of Q — **No visited list**

Add path extensions to Q in order of increasing path weight g.

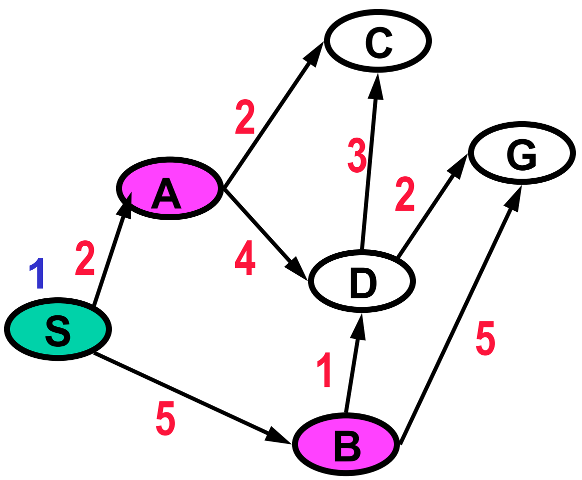
Implement priority queue with a heap. For graph with n nodes:

- Keeping a queue sorted takes time $O(n^2)$.
- Heap implementation takes time $O(n \lg n)$.

Best First with Uniform Cost

Pick first element of Q; Insert path extensions, sorted by g.

	Q
1	(0 S)
2	(2 A S) (5 B S)
3	
4	
5	
6	
7	



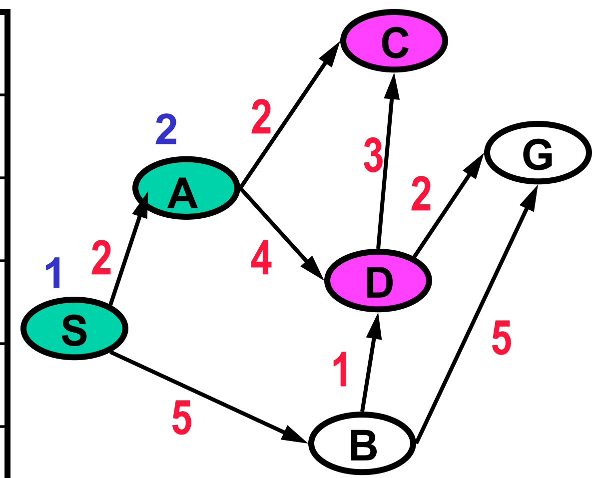
Here we:

- Insert on queue in order of g.
- Remove first element of queue.

Best First with Uniform Cost

Pick first element of Q; Insert path extensions, sorted by g.

	Q
1	(0 S)
2	(2 A S) (5 B S)
3	(4 C A S) (5 B S) (6 D A S)
4	
5	
6	
7	



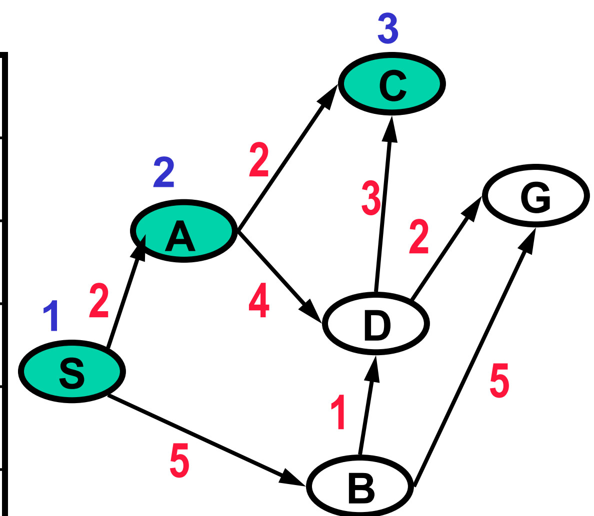
Here we:

- Insert on queue in order of g.
- Remove first element of queue.

Best First with Uniform Cost

Pick first element of Q; Insert path extensions, sorted by g.

	Q
1	~~(0 S)~~
2	~~(2 A S)~~ (5 B S)
3	~~(4 C A S)~~ (5 B S) (6 D A S)
4	(5 B S) (6 D A S)
5	
6	
7	



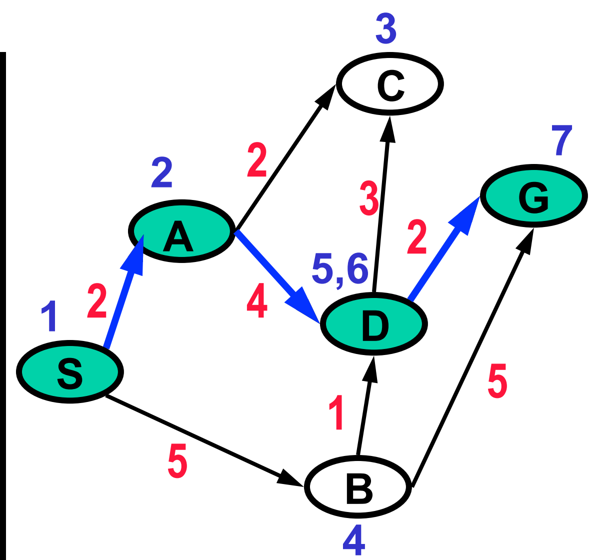
Here we:

- Insert on queue in order of g.
- Remove first element of queue.

Best First with Uniform Cost

Pick first element of Q; Insert path extensions, sorted by g.

	Q
1	~~(0 S)~~
2	~~(2 A S)~~ (5 B S)
3	~~(4 C A S)~~ (5 B S) (6 D A S)
4	~~(5 B S)~~ (6 D A S)
5	~~(6 D B S)~~ (6 D A S) (10 G B S)
6	~~(6 D A S)~~ (8 G D B S) (9 C D B S) (10 G B S)
7	~~(8 G D A S)~~ (8 G D B S) (9 C D A S) (9 C D B S) (10 G B S)



Can we stop as soon as the goal is enqueued ("visited")?

	Q
1	(0 S)
2	(2 A S) (5 B S)
3	(4 C A S) (5 B S) (6 D A S)
4	(5 B S) (6 D A S)
5	(6 D B S) (6 D A S) (10 G B S)
6	(6 D A S) (8 G D B S) (9 C D B S) (10 G B S)
7	(8 G D A S) (8 G D B S) (9 C D A S) (9 C D B S) (10 G B S)

- Other paths to the goal that are shorter may not yet be enqueued.
- Only when a path is pulled off the Q are we guaranteed that no shorter path will be added.
- This assumes all edges are positive.

Implementing the Search Strategies

Depth-first:

Pick first element of Q — Uses visited list

Add path extensions to front of Q

Breadth-first:

Pick first element of Q — Uses visited list

Add path extensions to end of Q

Uniform-cost:

Pick first element of Q — No visited list

Add path extensions to Q in increasing order of path weight g.

Best-first: (generalizes uniform-cost)

Pick first element of Q — **No visited list**

Add path extensions in increasing order of any cost function f.

Best-first Search Algorithm

Let gr be a Graph

Let S be the start vertex in gr.

Let f be a cost function on N.

Let Q be a list of simple partial paths in gr.

Let G be a Goal vertex in gr.

1. **Initialize Q with partial path (S) as only entry;**
2. **If Q is empty, fail. Else, pick partial path N from Q with best f;**
3. **If head(N) = G, return N; (we' ve reached the goal!)**
4. **(Otherwise) Remove N from Q;**
5. **Find all children of head(N) (its neighbors in gr) and create all the one-step extensions of N to each child;**
6. **Add to Q all the extended paths;**
7. **Go to Step 2.**

Cost and Performance

Searching a tree with branching factor b, solution depth d, and max depth m

Search Method	Worst Time	Worst Space	Guaranteed to find a path?	Optimal?
Depth-First	b^m	$b*m$	Yes	No
Breadth-First	b^{d+1}	b^{d+1}	Yes	Yes for unit edge cost
Best-First	b^{d+1}	b^{d+1}	Yes	Yes if uniform cost or A* w admissible heuristic
Beam (beam width = k)				
Hill-Climbing (no backup)				
Hill-Climbing (backup)				

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

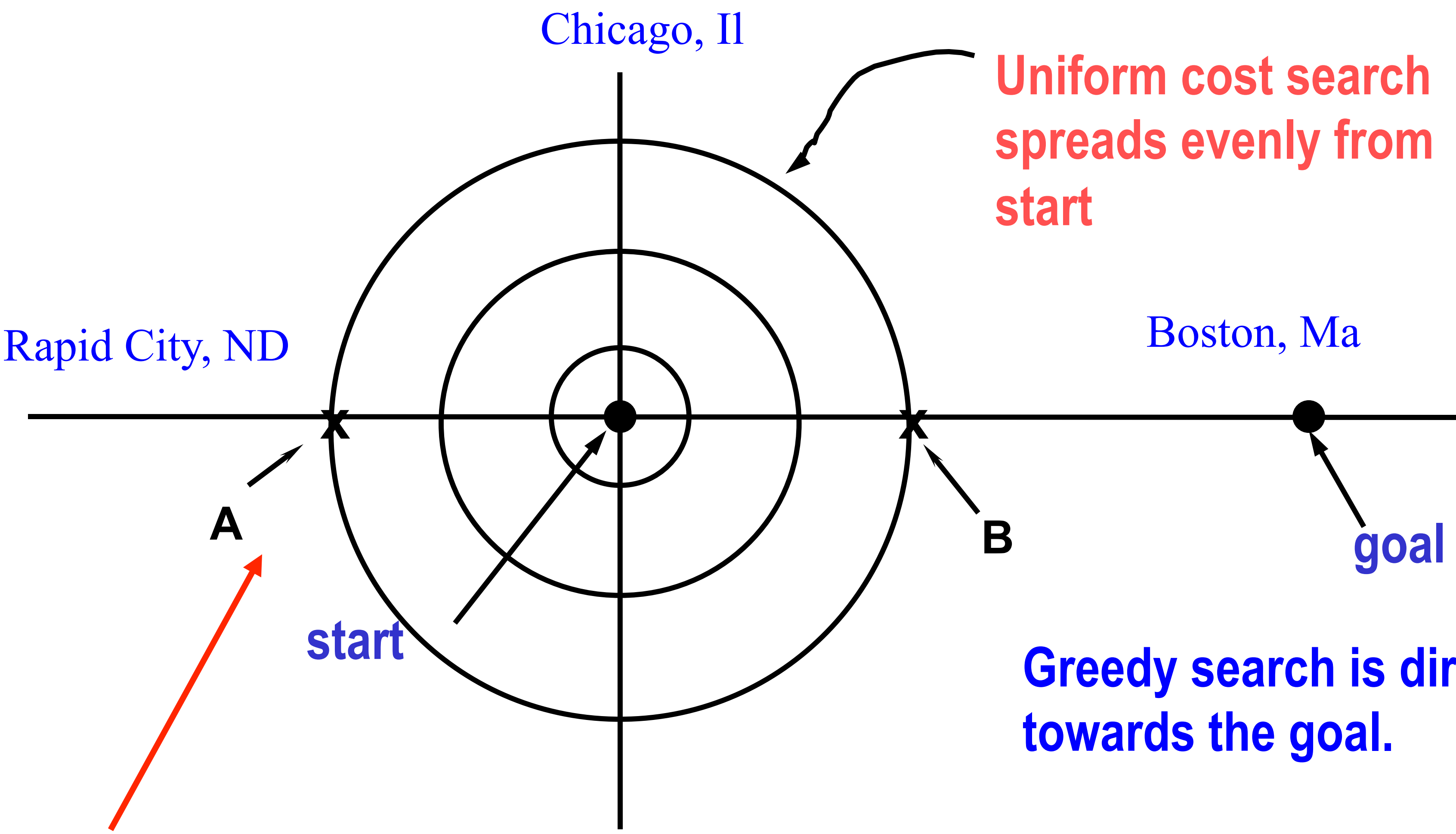
Remarks

- UCS is a straightforward instance of BFS.
- UCS is complete and optimal.
- However, like BFS (or DFS),
  UCS does not consider the goal node
  during search and could be slow.

Classes of Search

Blind	**Depth-First**	Systematic exploration of whole tree
(uninformed)	**Breadth-First**	until the goal is found.
	Iterative-Deepening	

Best-first	**Uniform-cost**	**Uses path "length" measure. Finds**
	Greedy	**"shortest" path.**
	A*	



Uniform cost search spreads evenly from start

Greedy search is directed towards the goal.

Uniform cost search explores the direction away from the goal as much as with the goal.

Greedy Search

Search in an order imposed by a heuristic function, measuring cost to go.

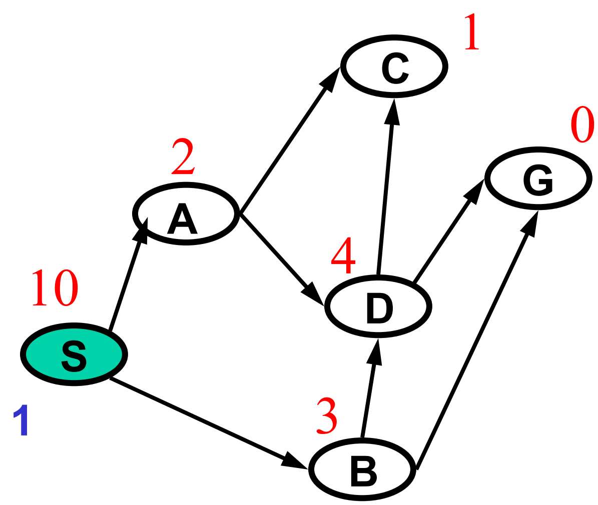
Heuristic function h – is a function of the current node n, not the partial path s to n.

- **Estimated distance to goal – h (n,G)**
  - **Example: straight-line distance in a road network.**
- **"Goodness" of a node – h (n)**
  - **Example: elevation.**
    - **Foothills, plateaus and ridges are problematic.**

Greedy

Pick first element of Q; Insert path extensions, **sorted by h.**

	Q	
1	(10 S)	
2		
3		
4		
5		



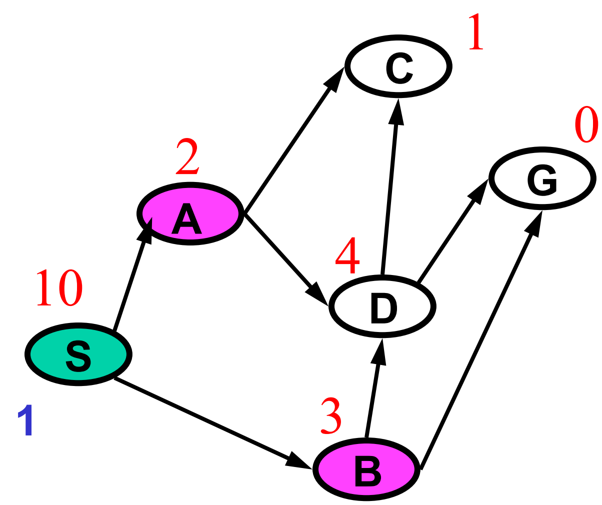
Added paths in blue; heuristic value of head is in front.

Heuristic values in red
Order of nodes in blue.

Greedy

Pick first element of Q; Insert path extensions, **sorted by h.**

	Q	
1	~~(10 S)~~	
2	(2 A S) (3 B S)	
3		
4		
5		



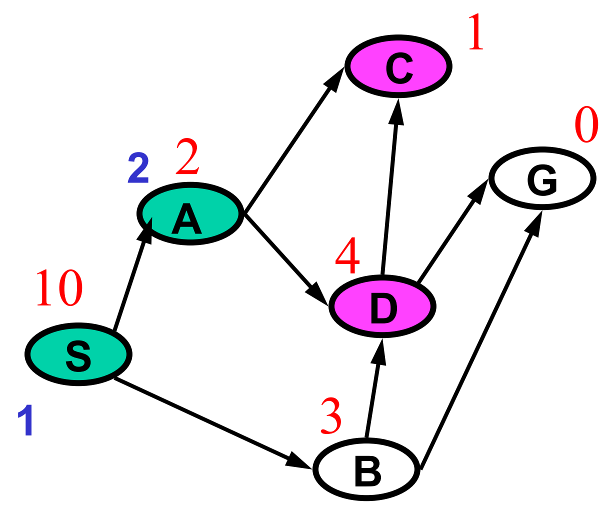
Added paths in blue; heuristic value of head is in front.

Heuristic values in red
Order of nodes in blue.

Greedy

Pick first element of Q; Insert path extensions, **sorted by h.**

	Q	
1	~~(10 S)~~	
2	~~(2 A S)~~ (3 B S)	
3	(1 C A S) (3 B S) (4 D A S)	
4		
5		



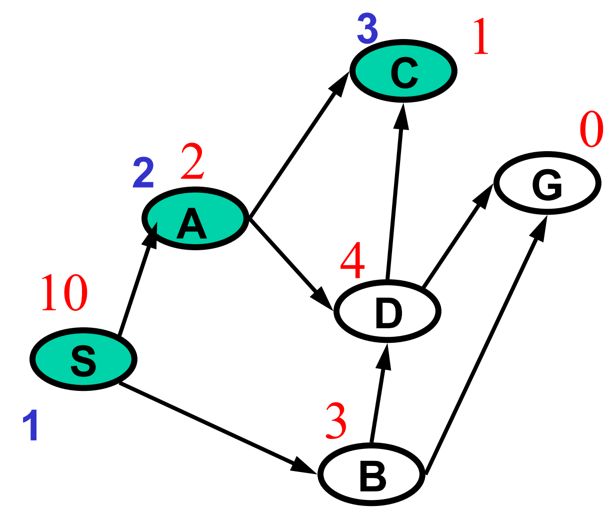
Added paths in blue; heuristic value of head is in front.

Heuristic values in red
Order of nodes in blue.

Greedy

Pick first element of Q; Insert path extensions, **sorted by h.**

	Q	
1	~~(10 S)~~	
2	~~(2 A S)~~ (3 B S)	
3	~~(1 C A S)~~ (3 B S) (4 D A S)	
4	(3 B S) (4 D A S)	
5		



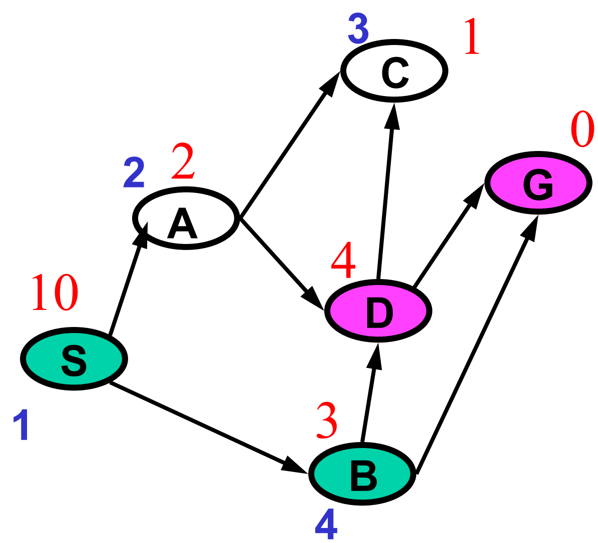
Added paths in blue; heuristic value of head is in front.

Heuristic values in red
Order of nodes in blue.

Greedy

Pick first element of Q; Insert path extensions, **sorted by h.**

	Q	
1	~~(10 S)~~	
2	~~(2 A S)~~ (3 B S)	
3	~~(1 C A S)~~ (3 B S) (4 D A S)	
4	~~(3 B S)~~ (4 D A S)	
5	(0 G B S) (4 D A S) (4 D B S)	



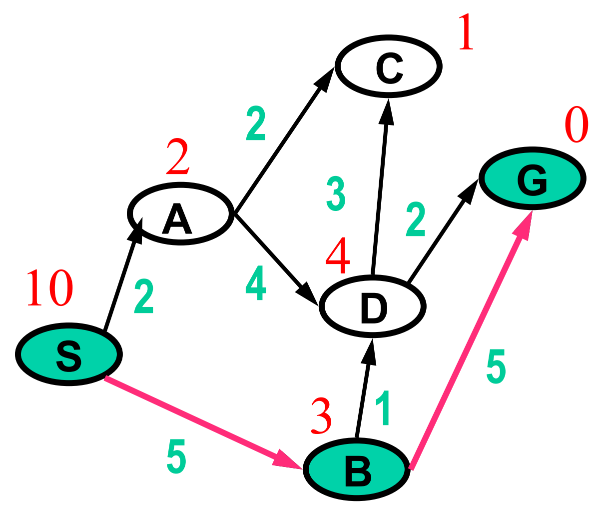
Heuristic values in red
Order of nodes in blue.

Added paths in blue; heuristic value of head is in front.

Greedy

Pick first element of Q; Insert path extensions, **sorted by h.**

	Q	
1	~~(10 S)~~	
2	~~(2 A S)~~ (3 B S)	
3	~~(1 C A S)~~ (3 B S) (4 D A S)	
4	~~(3 B S)~~ (4 D A S)	
5	(0 G B S) (4 D A S)) (4 D B S)	



Heuristic values in red
Edge cost in green.

Added paths in blue; heuristic value of head is in front.

Did Greedy search produce the shortest path?

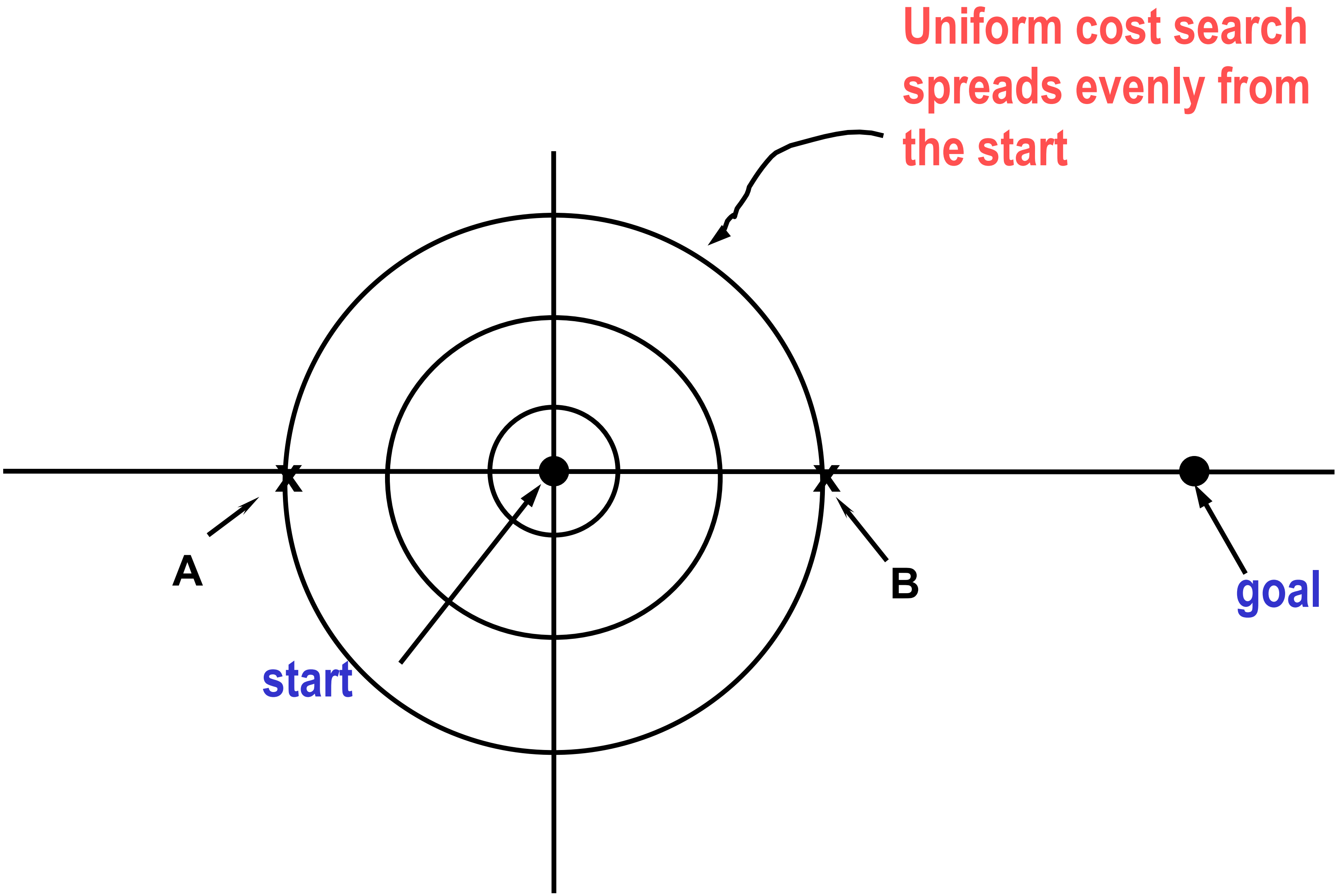
Remarks

- The performance of GS depends strongly on the quality of the heuristic.
  - With a good heuristic,
    GS reaches the goal quickly.
  - With a misleading heuristic,
    GS may "get stuck" and
    perform worse than UCS.
- GS is not optimal.

Classes of Search

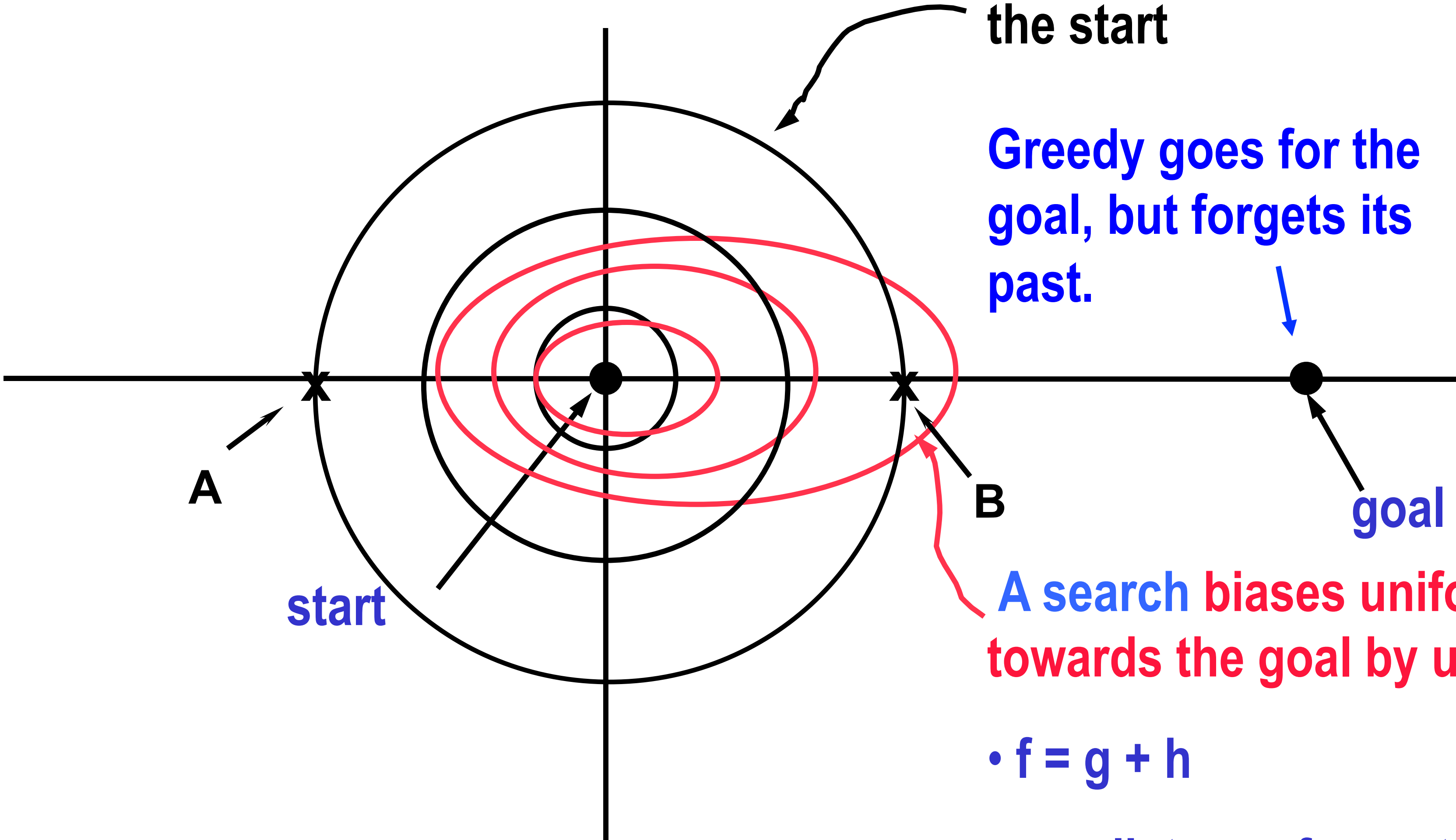
Blind	**Depth-First**	Systematic exploration of whole tree
(uninformed)	**Breadth-First**	until the goal is found.
	Iterative-Deepening	

Best-first	**Uniform-cost**	Uses path "length" measure. Finds
	Greedy	"shortest" path.
	A*	


Uniform cost search spreads evenly from the start
A
start
B
goal

Uniform cost search spreads evenly from the start

Greedy goes for the goal, but forgets its past.



A search biases uniform cost towards the goal by using h:

- f = g + h
- g = distance from start.
- h = estimated distance to goal.

Comparison of UCS and GS

UCS

- Think about the past: order the queue by $g(v)$, the path cost from the start (cost-to-come).

- Optimal.
- Usually not fast.

GS

- Think about the future: order the queue by $h(v)$, the estimated path cost to the goal (cost-to-go).

- Not optimal.
- Maybe fast.

Combining UCS and GS

- What if we put $g(v)$ and $h(v)$ together?
  Order the queue according to

$$f(v) = g(v) + h(v)$$

- $g(v)$: cost-to-come (from the start to v).
- $h(v)$: cost-to-go estimate (from v to the goal).
- $f(v)$: estimated cost of the path (from the start to v and then to the goal).

- Resulting can be both optimal and fast.

Remarks

- A search generalizes both UCS and GS.
  - Setting $h(v)=0$, we get UCS.
  - Ignoring $g(v)$, we get GS.
- A search appears fast, but is not optimal. What is the problem?

A* Search

To make A search optimal,

- $h(v)$ must always underestimate the distance to the goal.
- In other words, the heuristic must be **optimistic** (*admissible*):

$$h(v) \leq h^*(v)$$

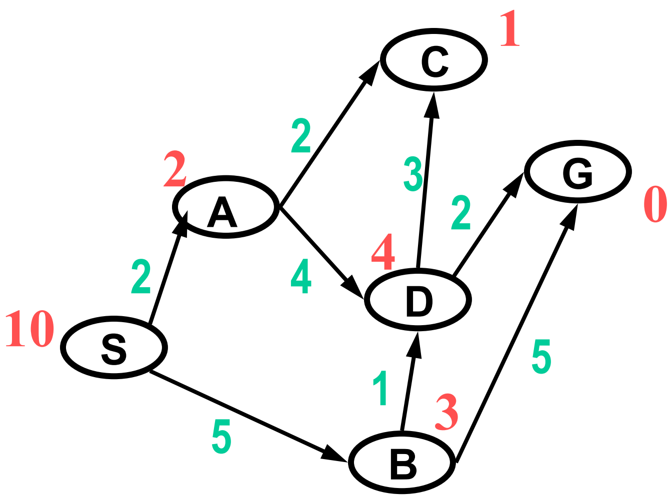
Simple Optimal Search Algorithm
BFS + Admissible Heuristic

Let gr be a Graph
Let S be the start vertex in gr and
Let Q be a list of simple partial paths in gr
Let G be a Goal vertex in gr.
Let f = g + h be an admissible heuristic function.

1. **Initialize Q with partial path (S) as only entry;**
2. **If Q is empty, fail. Else, use f to pick "best" partial path N from Q;**
3. **If head(N) = G, return N;** **(we' ve reached the goal)**
4. **(Otherwise) Remove N from Q;**
5. **Find all the descendants of head(N) (its neighbors in Gr) and create all the one-step extensions of N to each descendant;**
6. **Add to Q all the extended paths;**
7. **Go to Step 2.**

In the example, is h an admissible heuristic?

- A is ok.
- B is ok.
- C is ok.
- D is too big; needs to be ≤ 2.
- S is too big; can always use 0 for start.



Heuristic Values of h in Red.

Edge cost in Green.

A finds an optimal solution
if h never over estimates.

- Search is called A*.
- h is called "admissible."

Admissible heuristics for 8 puzzle?

6	2	8
	3	5
4	7	1

S

1	2	3
8		4
7	6	5

G

What is the heuristic?

- **An underestimate of number of moves to the goal.**

Examples:

1. **Number of misplaced tiles (7)**
2. **Sum of Manhattan distance of each tile to its goal location (17)**

Finding admissible heuristics

- Often domain-specific knowledge is required.
- Examples

- $h(v) = 0$: this always works! However, it is not very useful, and in this case $A^* = UCS$.
- $h(v) = \mathrm{distance}(v, g)$ when the vertices of the graphs are physical locations.
- $h(v) = \|v - g\|_p$, when the vertices of the graph are points in a normed vector space.

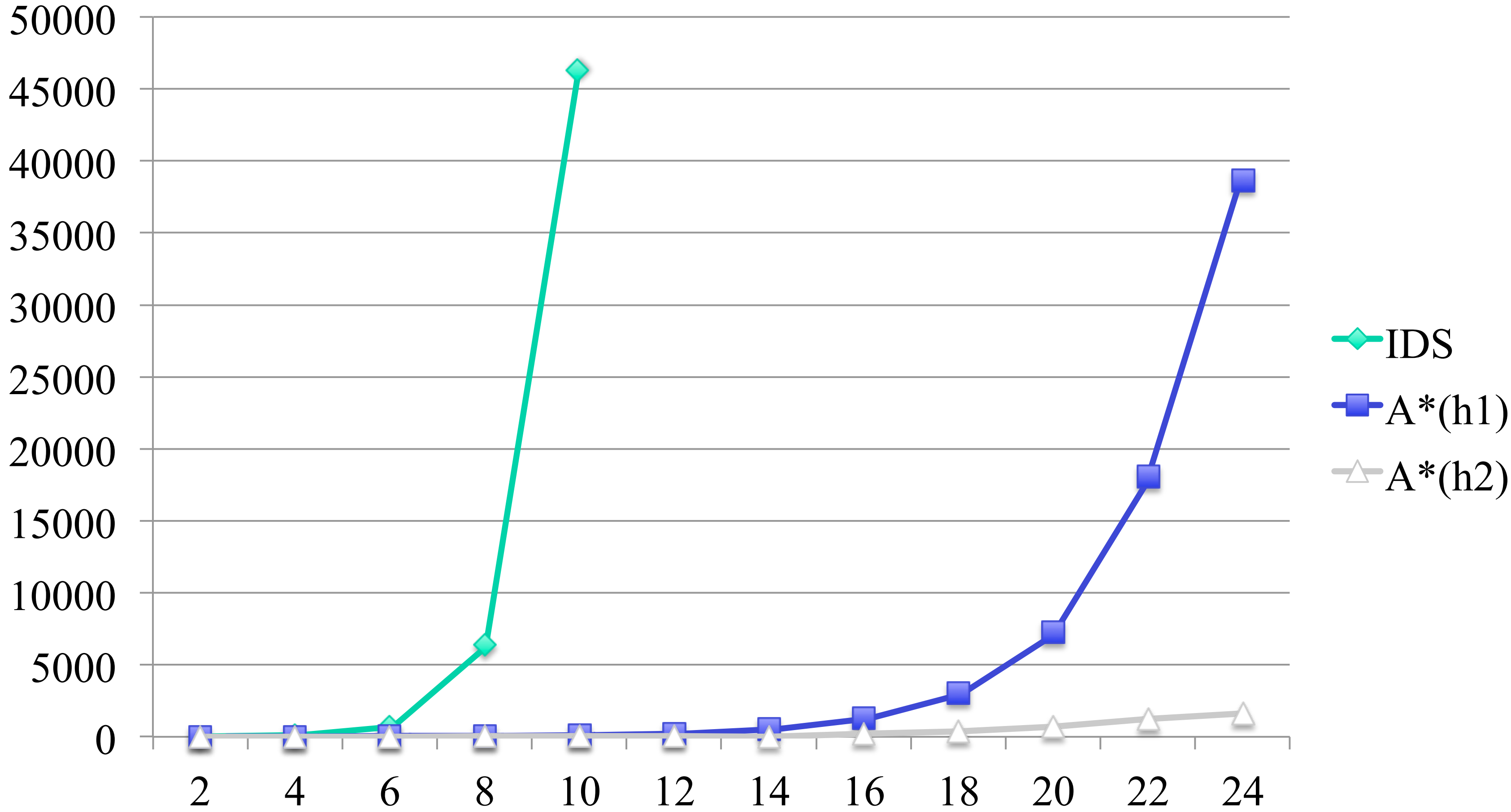
Finding admissible heuristics

- Relaxation
  - Create a relaxed problem by ignoring some constraints in the original problem.
- Consistency
  - A heuristic function h is consistent if

$$h(u) \leq w\left(e = (u, v)\right) + h(v), \quad \forall (u, v) \in E.$$

  - A consistent heuristic function is admissible.

Benefits of heuristics



AIMA, Sect. 3.6, Fig. 3.29

Why the difference?

- $h(v)=0$
- $h(v)=h^*(v)$

A* optimality: intuition

If the heuristic function

- over-estimates the distance to the goal,
  - we eliminate the optimal solution and make a mistake that is irrecoverable.
- under-estimates the distance,
  - the search may be misled.
  - However, as the search continues, the cost of the sub-optimal path rises, and
  - we eventually recover from the mistake.

A* optimality: proof

- Assume that A^* returns P, but $w(P) > w^*$ (w^* is the optimal path weight/cost).
- Find the first unexpanded node on the optimal path P^*, call it n.
- $f(n) > w(P)$, otherwise we would have expanded n.
- $f(n) = g(n) + h(n)$ by definition
- $\quad = g^*(n) + h(n)$ because n is on the optimal path.
- $\quad \leq g^*(n) + h^*(n)$ because h is admissible
- $\quad = f^*(n) = W^*$ because h is admissible
- Hence $W^* \geq f(n) > W$, which is a contradiction.

Can We Prune Search Branches?

Property: Shortest Paths are extensions of Shortest Sub-Paths.

- Suppose path $P = P_1$ o P_2 , from S to G, is shortest.
- Suppose P_2 , from U to G, is not.
- Then there exists P_2' from U to G that is shorter than P_2.
- Hence P' = P_1 o P_2' is shorter than P.
- By contradiction, if P is a shortest, then P_2 is a shortest sub-path.

Can We Prune Search Branches?

Property: Shortest Paths are extensions of Shortest Sub-Paths.

Idea: when *shortest* path S to U is found, ignore other paths S to U.

- When BFS dequeues the first partial path with head node U, this path is *guaranteed to be the shortest path* from S to U.

- Given the first path to U, we don't need to extend other paths to U; delete them (expanded list).

Simple Optimal Search Algorithm
How do we add dynamic programming?

Let gr be a Graph.
Let Q be a list of simple partial paths in gr.
Let S be the start vertex in gr.
Let G be a Goal vertex in gr.
Let f = g + h be an admissible heuristic function.

1. **Initialize Q with partial path (S) as only entry;**
2. **If Q is empty, fail. Else, use f to pick the "best" partial path N from Q;**
3. **If head(N) = G, return N; (we've reached the goal)**
4. **(Else) Remove N from Q;**
5. **Find all children of head(N) (its neighbors in gr) and create all the one-step extensions of N to each child;**
6. **Add to Q all the extended paths;**
7. **Go to Step 2.**

A* Optimal Search Algorithm

BFS + Dyn Prog + Admissible Heuristic

Let gr be a Graph

Let S be the start vertex in gr.

Let Q be a list of simple partial paths in gr.

Let G be a Goal vertex in gr.

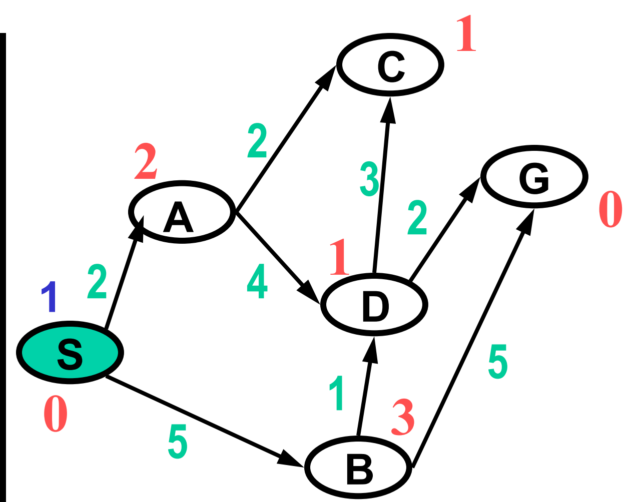
Let f = g + h be an admissible heuristic function.

1. Initialize Q with partial path (S) as only entry; set Expanded = ( );
2. If Q is empty, fail. Else, use f to pick "best" partial path N from Q;
3. If head(N) = G, return N; (we've reached the goal)
4. (Else) Remove N from Q;
5. if head(N) is in Expanded, go to Step 2; otherwise, add head(N) to Expanded;
6. Find all the children of head(N) (its neighbors in gr) not in Expanded, and create all one-step extensions of N to each child;
7. Add to Q all the extended paths;
8. Go to Step 2.

A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q; Insert path extensions, sorted by path length + heuristic.

	Q	Expanded
1	(0 S)	



Heuristic Values of g in Red

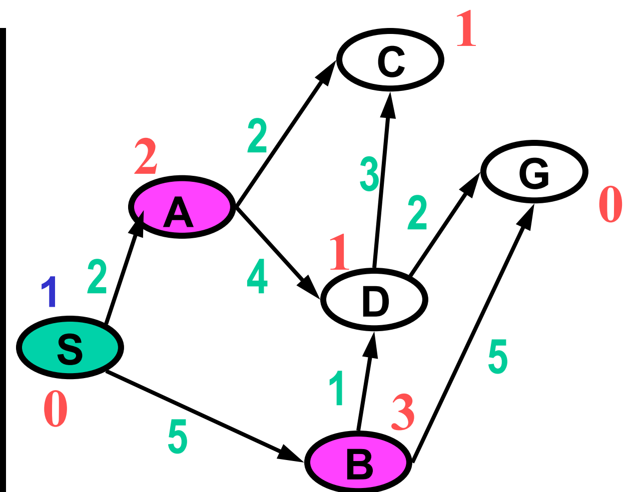
Edge cost in Green

Added paths in blue; cost f at head of each path.

A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q; Insert path extensions, sorted by path length + heuristic.

	Q	Expanded
1	~~(0 S)~~	
2		S



Heuristic Values of g in Red

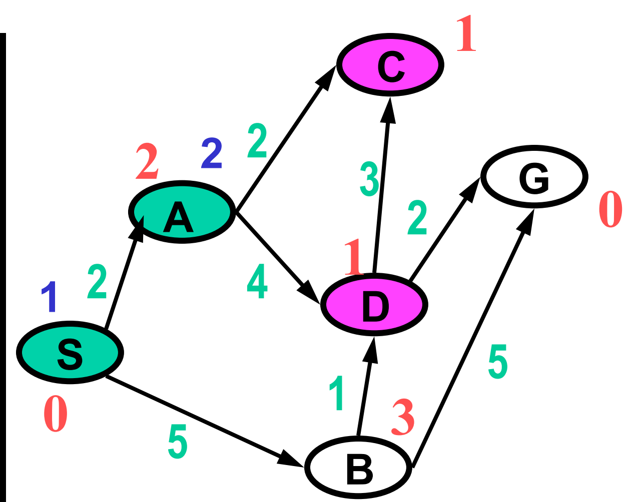
Edge cost in Green

Added paths in blue; cost f at head of each path

A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q; Insert path extensions, sorted by path length + heuristic.

	Q	Expanded
1	~~(0 S)~~	
2	~~(4 A S)~~ (8 B S)	S
3		S A



Heuristic Values of g in Red

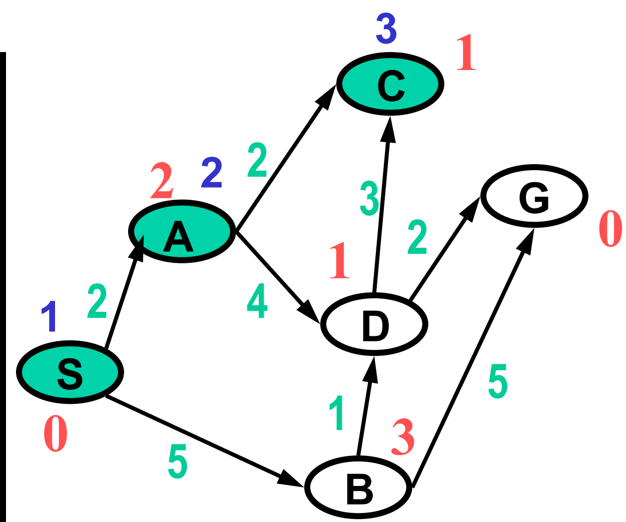
Edge cost in Green

Added paths in blue; cost f at head of each path

A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q; Insert path extensions, sorted by path length + heuristic.

	Q	Expanded
1	~~(0 S)~~	
2	~~(4 A S)~~ (8 B S)	S
3	~~(5 C A S)~~ (7 D A S) (8 B S)	S A
4		S A C



Heuristic Values of g in Red

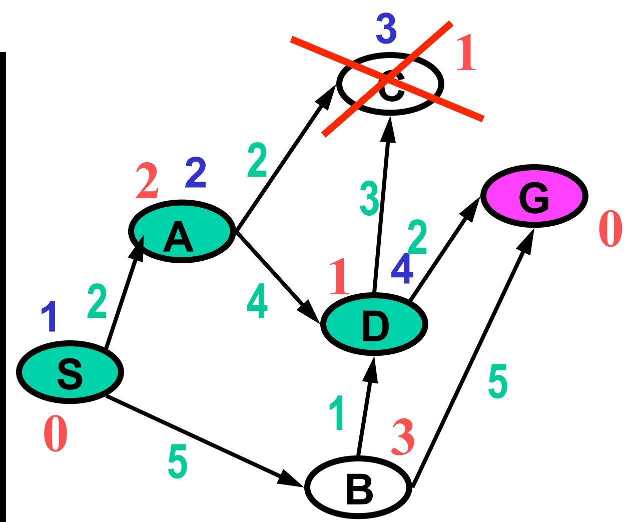
Edge cost in Green

Added paths in blue; cost f at head of each path

A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q; Insert path extensions, sorted by path length + heuristic.

	Q	Expanded
1	(0 S)	
2	(4 A S) (8 B S)	S
3	(5 C A S) (7 D A S) (8 B S)	S A
4	(7 D A S) (8 B S)	S A C
5		S A C D



Heuristic Values of g in Red

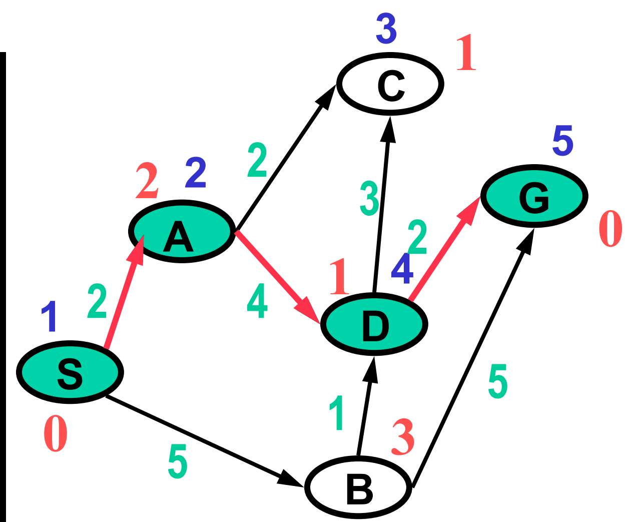
Edge cost in Green

Added paths in blue; cost f at head of each path

A* (BFS + DynProg + Admissible Heuristic)

Pick first element of Q; Insert path extensions, sorted by path length + heuristic.

	Q	Expanded
1	(0 S)	
2	(4 A S) (8 B S)	S
3	(5 C A S) (7 D A S) (8 B S)	S A
4	(7 D A S) (8 B S)	S A C
5	(8 G D A S) (8 B S)	S A C D



Heuristic Values of g in Red

Edge cost in Green

Added paths in blue; cost f at head of each path

Expanded List can offer Exponential Saving

Enumerate all (sub)paths:

- For simple paths of length n through S states, $O(|S|^{2n+1})$.
- For simple paths up to length n, $O(|S|^{2n+2}$.

Enumerate all shortest (sub)paths:

- **Property:** Shortest paths are extensions of Shortest Sub-Paths.
- **Algorithm:** Dynamic Programming:
  - Compute shortest paths of length n from shortest (sub)paths of length n-1.

$$h^*(u) = \min_{(u,v)\in E} \left[w(\ (u,v)\ ) + h^*(v) \right].$$

  - $O(n|S|^2)$ for shortest paths up to length n and |S| states.

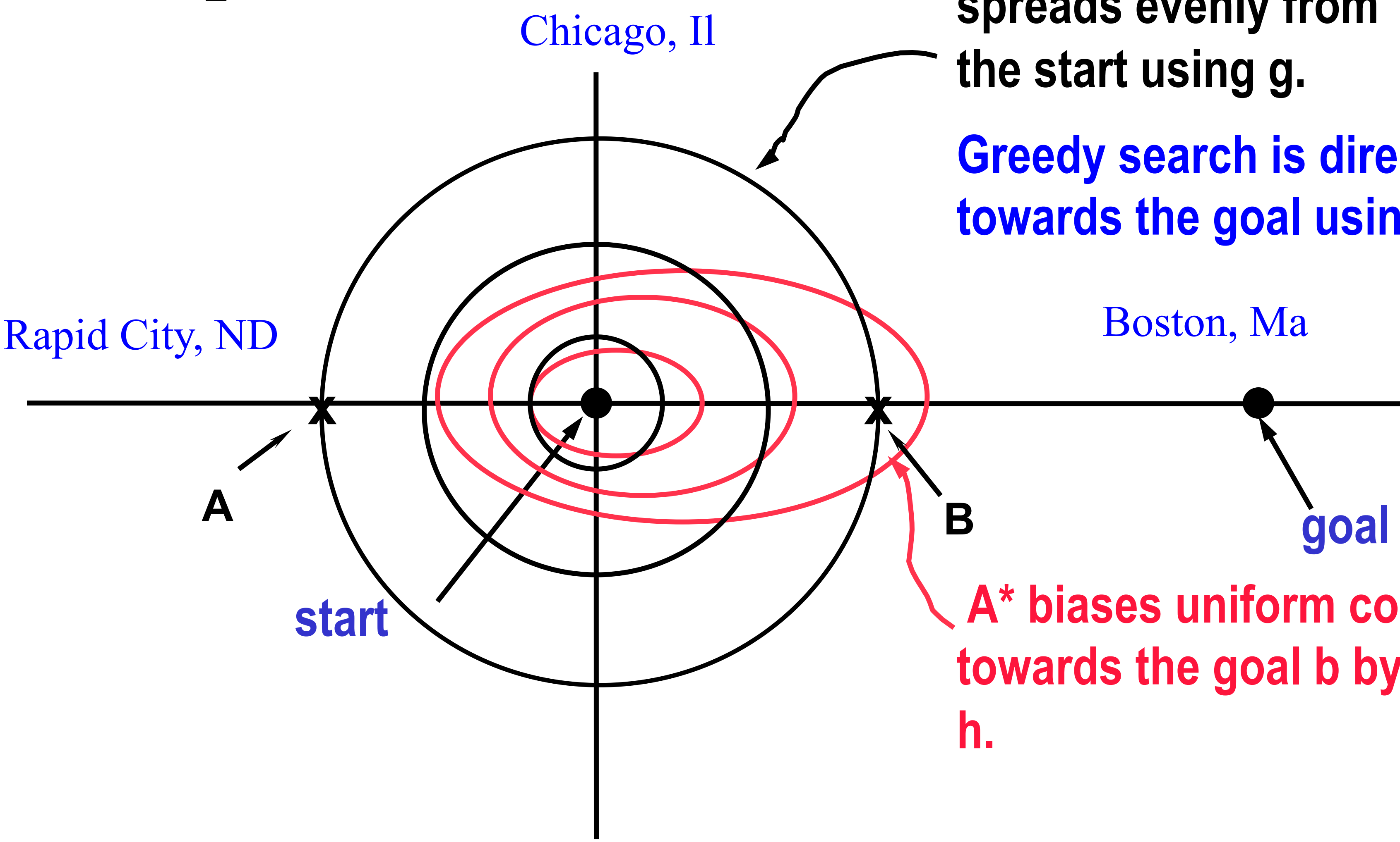
Remarks

- The performance of A* search depends on the quality of the heuristic.
- A* search is optimal.

Recap: Informed Search

Uniform cost search spreads evenly from the start using g.

Greedy search is directed towards the goal using h.



A* biases uniform cost towards the goal b by adding h.

Appendices

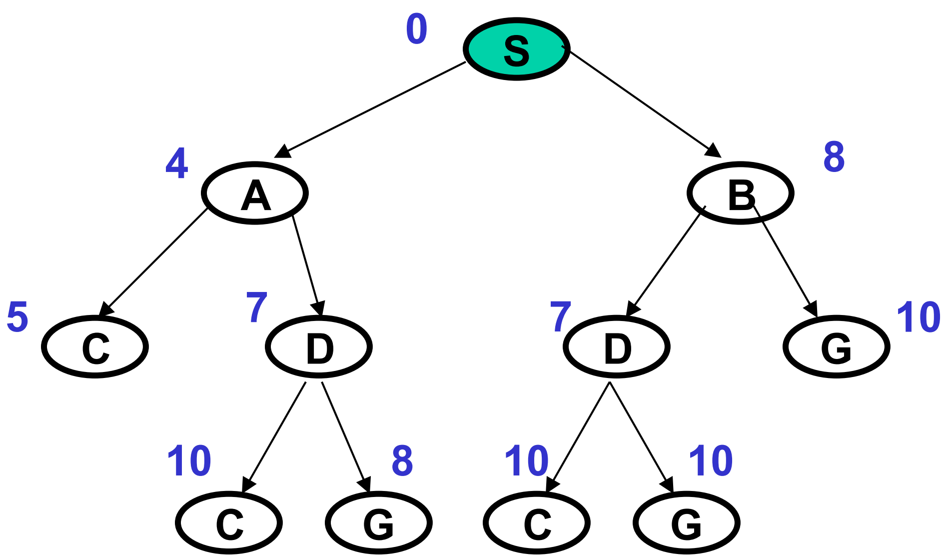
- Bounding.
- Variants.
- More about Informed Search.
- Dynamic Programming.

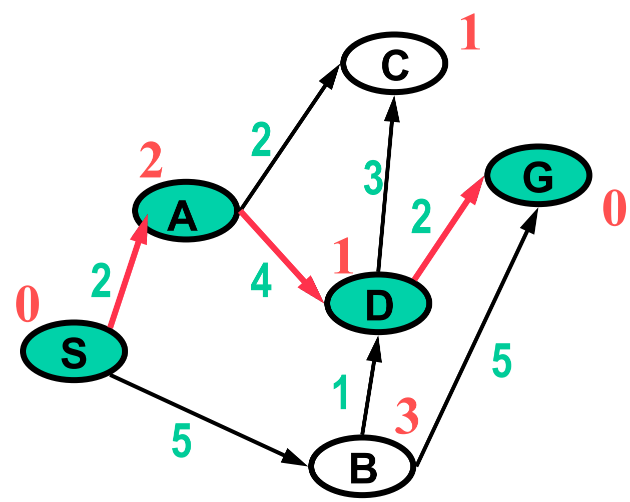
Classes of Search

Blind (uninformed)	**Depth-First** **Breadth-First** **Iterative-Deepening**	Systematic exploration of whole tree until the goal is found.
Best-first	**Uniform-cost** **Greedy** **A***	Uses path "length" measure. Finds "shortest" path.
Bounding	**Branch and Bound**	Prunes suboptimal branches.
	Alpha/Beta (L6)	Prunes options that the adversary rules out.

Branch and Bound

- A* generalizes best-first search.
- How do we generalize depth-first search?



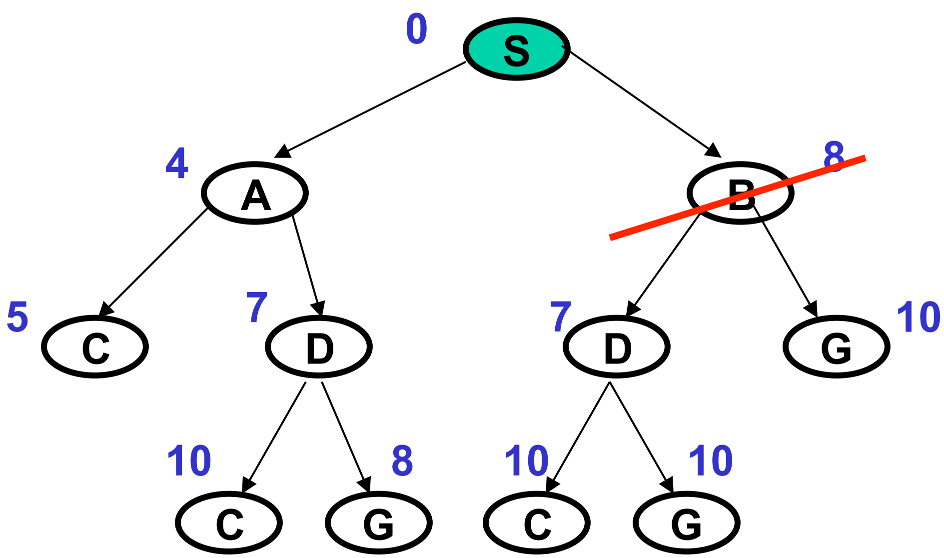


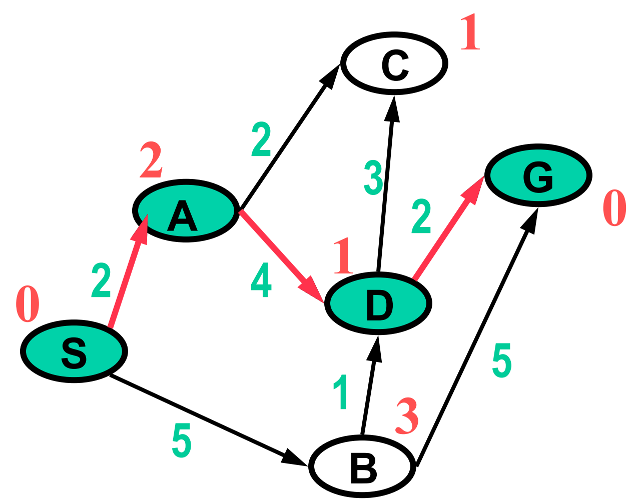
Heuristic Values of g in Red

Edge cost in Green

Branch and Bound

- Idea 1: Maintain the best solution found thus far (incumbent).
- Idea 2: Prune all subtrees worse than the incumbent.





Incumbent:

cost U = ∞, **8**

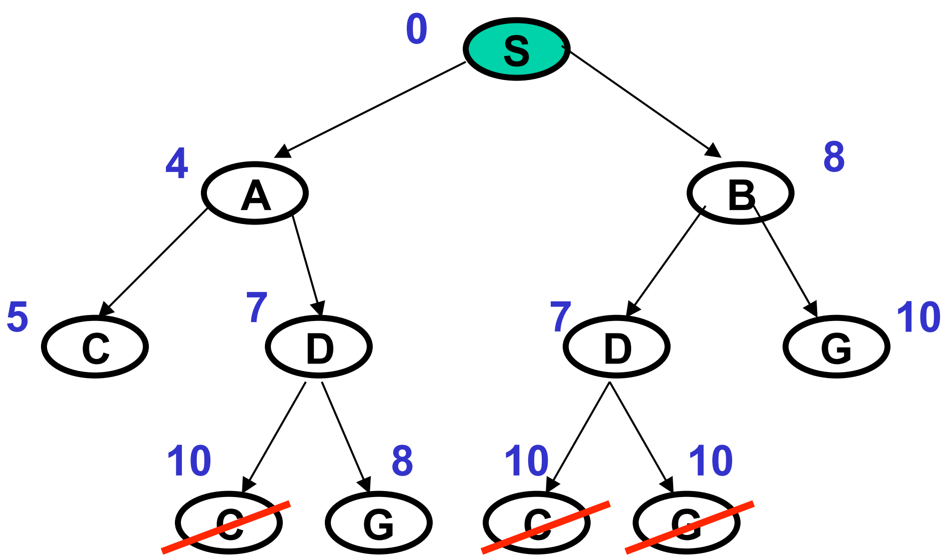
path P = (), **(S A D G)**

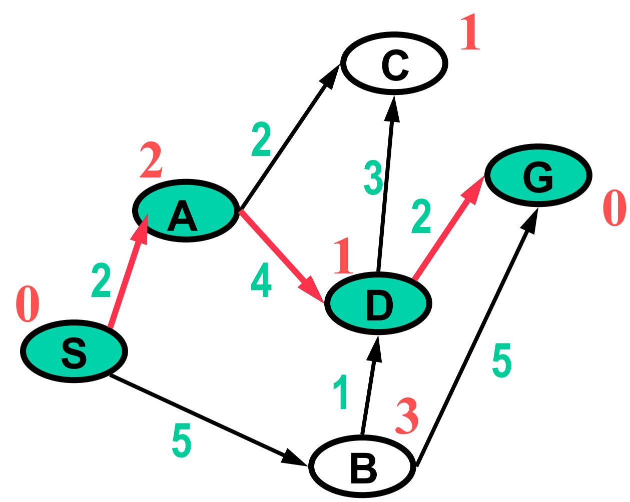
Heuristic Values of g in Red

Edge cost in Green

Branch and Bound

- Idea 1: Maintain the best solution found thus far (incumbent).
- Idea 2: Prune all subtrees worse than the incumbent.
- Any search order allowed (DFS, Reverse-DFS, BFS, Hill w BT…).





Incumbent:

cost U = ∞, **10,** **8**

path P = (), **(S B G)** **(S A D G)**

Heuristic Values of g in Red

Edge cost in Green

Simple Optimal Search Using Branch and Bound

Let gr be a Graph.

Let Q be a list of simple partial paths in gr.

Let S be the start vertex in gr.

Let G be a Goal vertex in gr.

Let f = g + h be an admissible heuristic function.

U and P are the cost and path of the best solution thus far (Incumbent).

1. **Initialize Q with partial path (S); Incumbent U = ∞, P = ();**
2. **If Q is empty, return Incumbent U and P,**
   Else, remove a partial path N from Q;
3. **If f(N) >= U, Go to Step 2.**
4. **If head(N) = G, then U = f(N) and P = N (a better path to the goal)**
5. **(Else) Find all children of head(N) (its neighbors in gr) and create all the one-step extensions of N to each child.**
6. **Add to Q all the extended paths.**
7. **Go to Step 2.**

Appendices

- Bounding.
- Variants.
- More about Informed Search.
- Dynamic Programming.

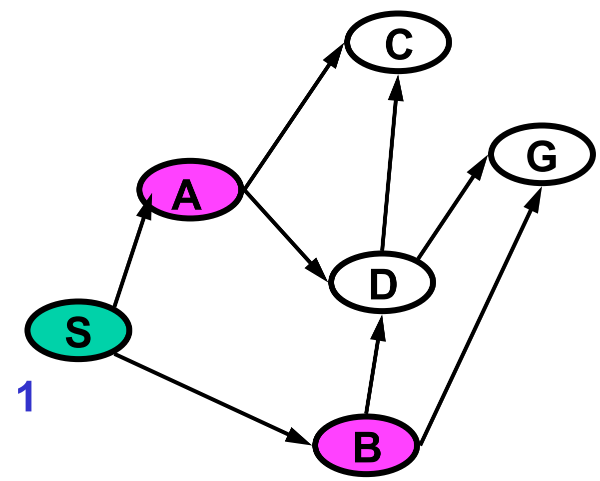
Classes of Search

Blind (uninformed)	**Depth-First** **Breadth-First** **Iterative-Deepening**	Systematic exploration of whole tree until the goal is found.
Best-first	**Uniform-cost** **Greedy** **A***	Uses path "length" measure. Finds "shortest" path.
Bounding	**Branch and Bound**	Prunes suboptimal branches.
	Alpha/Beta	Prunes options that the adversary rules out.
Variants	**Hill-Climbing (w backup)** **Beam** **IDA***	

Hill-Climbing

Pick **first element** of Q; **Replace Q** with **extensions** (sorted by **heuristic value**)

	Q	
1	~~(10 S)~~	
2	(2 A S) (3 B S)	
3		
4		



Heuristic Values

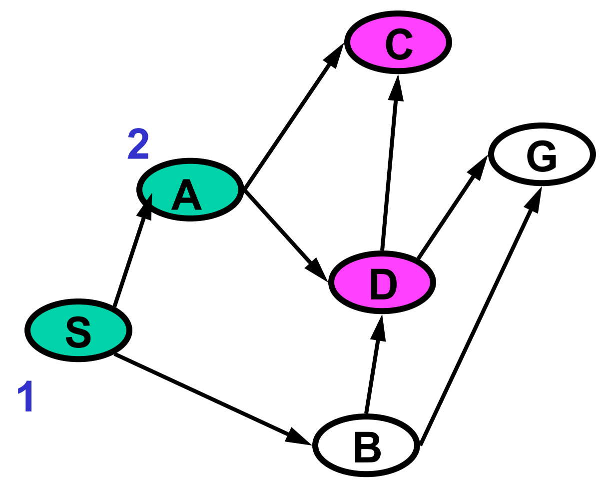
A=2	C=1	S=10
B=3	D=4	G=0

Added paths in blue; heuristic value of head is in front.

Hill-Climbing

Pick **first element** of Q; **Replace Q** with **extensions** (sorted by **heuristic value**)

	Q	
1	~~(10 S)~~	
2	~~(2 A S)~~ (3 B S) Removed	
3	(1 C A S) (4 D A S)	
4		



Heuristic Values

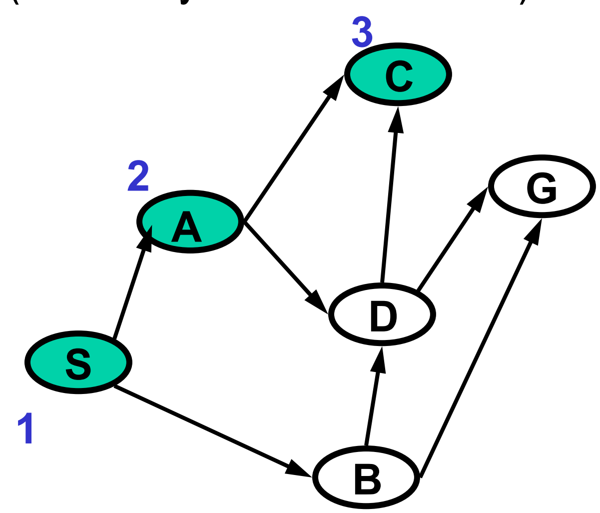
A=2	C=1	S=10
B=3	D=4	G=0

Added paths in blue; heuristic value of head is in front.

Hill-Climbing

Pick **first element** of Q; **Replace Q** with **extensions** (sorted by **heuristic value**)

	Q	
1	~~(10~~ S)	
2	~~(2 A S)~~ (3 B S)	
3	~~(1 C A S)~~ (4 D A S)	
4	( )	



Fails to find a path!

Heuristic Values

A=2	C=1	S=10
B=3	D=4	G=0

Added paths in blue; heuristic value of head is in front.

Cost and Performance

Searching a tree with branching factor b, solution depth d, and max depth m

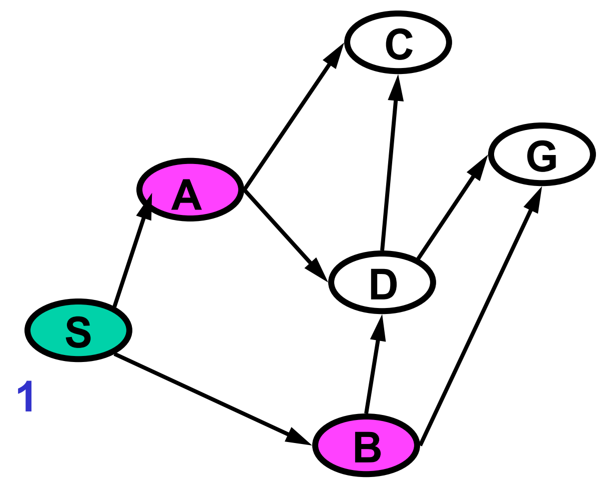
Search Method	Worst Time	Worst Space	Guaranteed to find a path?	Optimal?
Depth-First	b^m	$b*m$	Yes	No
Breadth-First	b^{d+1}	b^{d+1}	Yes	Yes for unit edge cost
Best-First	b^{d+1}	b^{d+1}	Yes	Yes if uniform cost or A* w admissible heuristic.
Beam (beam width = k)				
Hill-Climbing (no backup)	$b*m$	b	No	No
Hill-Climbing (backup)				

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

Hill-Climbing (with backup)

Pick first element of Q; **Add** path extensions (sorted by heuristic value) **to front of Q**

	Q	
1	~~(10 S)~~	
2	(2 A S) (3 B S)	
3		
4		
5		



Heuristic Values

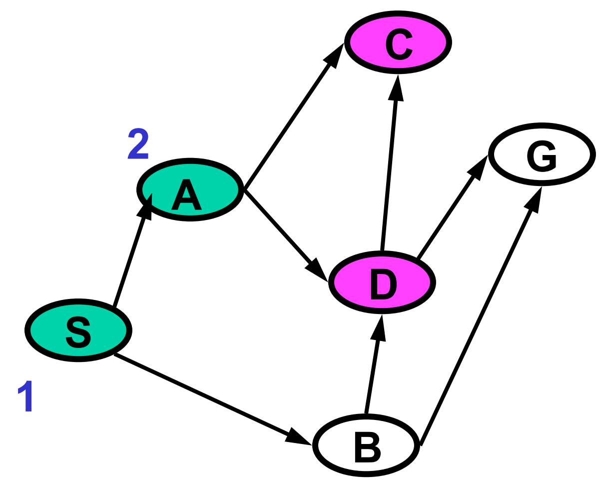
A=2	C=1	S=10
B=3	D=4	G=0

Added paths in blue; heuristic value of head is in front.

Hill-Climbing (with backup)

Pick first element of Q; **Add** path extensions (sorted by heuristic value) **to front of Q**

	Q	
1	~~(10 S)~~	
2	~~(2 A S)~~ (3 B S)	
3	(1 C A S) (4 D A S) (3 B S)	
4	All new nodes before old	
5		



Heuristic Values

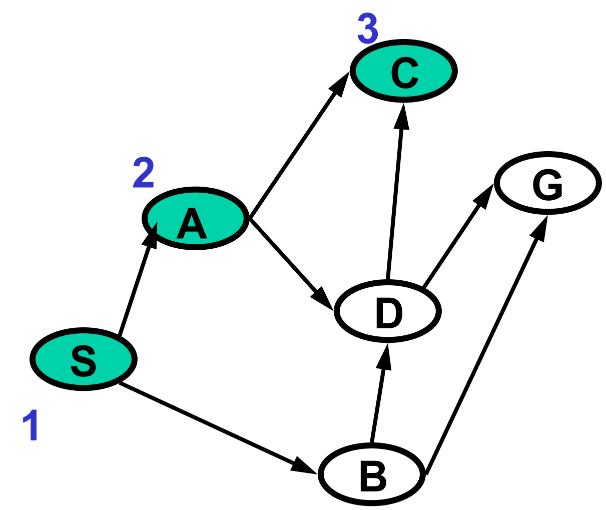
A=2	C=1	S=10
B=3	D=4	G=0

Added paths in blue; heuristic value of head is in front.

Hill-Climbing (with backup)

Pick first element of Q; **Add** path extensions (sorted by heuristic value) **to front of Q**

	Q	
1	~~(10 S)~~	
2	~~(2 A S)~~ (3 B S)	
3	~~(1 C A S)~~ (4 D A S) (3 B S)	
4	(4 D A S) (3 B S)	
5		



Heuristic Values

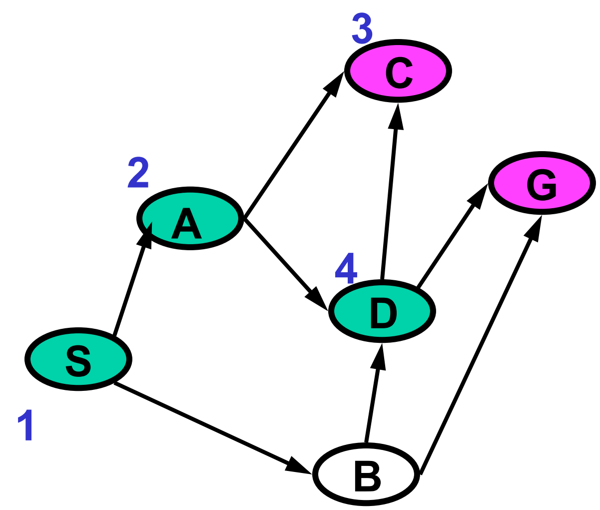
A=2	C=1	S=10
B=3	D=4	G=0

Added paths in blue; heuristic value of head is in front.

Hill-Climbing (with backup)

Pick first element of Q; **Add** path extensions (sorted by heuristic value) **to front of Q**

	Q	
1	~~(10 S)~~	
2	~~(2 A S)~~ (3 B S)	
3	~~(1 C A S)~~ (4 D A S) (3 B S)	
4	~~(4 D A S)~~ (3 B S)	
5	(0 G D A S) (1 C A S) (3 B S)	



Heuristic Values

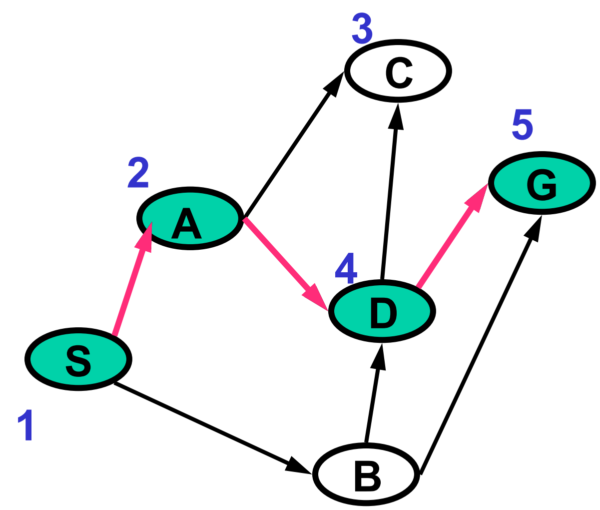
A=2	C=1	S=10
B=3	D=4	G=0

Added paths in blue; heuristic value of head is in front.

Hill-Climbing (with backup)

Pick first element of Q; **Add** path extensions (sorted by heuristic value) **to front of Q**

	Q	
1	(10 S)	
2	(2 A S) (3 B S)	
3	(1 C A S) (4 D A S) (3 B S)	
4	(4 D A S) (3 B S)	
5	(0 G D A S) (1 C A S) (3 B S)	



Heuristic Values

A=2	C=1	S=10
B=3	D=4	G=0

Added paths in blue; heuristic value of head is in front.

Cost and Performance

Searching a tree with branching factor b, solution depth d, and max depth m

Search Method	Worst Time	Worst Space	Guaranteed to find a path?	Optimal?
Depth-First	b^m	$b*m$	Yes	No
Breadth-First	b^{d+1}	b^{d+1}	Yes	Yes for unit edge cost
Best-First	b^{d+1}	b^{d+1}	Yes	Yes if uniform cost or A* w admissible heuristic.
Beam (beam width = k)				
Hill-Climbing (no backup)	$b*m$	b	No	No
Hill-Climbing (backup)				

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

Cost and Performance

Searching a tree with branching factor b, solution depth d, and max depth m

Search Method	Worst Time	Worst Space	Guaranteed to find a path?	Optimal?
Depth-First	b^m	b*m	Yes	No
Breadth-First	b^{d+1}	b^{d+1}	Yes	Yes for unit edge cost
Best-First	b^{d+1}	b^{d+1}	Yes	Yes if uniform cost or A* w admissible heuristic.
Beam (beam width = k)				
Hill-Climbing (no backup)	b*m	b	No	No
Hill-Climbing (backup)	b^m	b*m	Yes	No

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

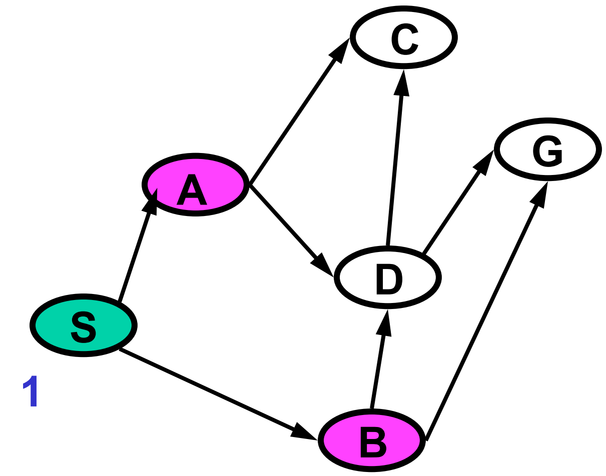
Classes of Search

Blind (uninformed)	**Depth-First** **Breadth-First** **Iterative-Deepening**	Systematic exploration of whole tree until the goal is found.
Best-first	**Uniform-cost** **Greedy** **A***	Uses path "length" measure. Finds "shortest" path.
Bounding	**Branch and Bound**	Prunes suboptimal branches
	Alpha/Beta	Prunes options the adversary rules out
Variants	**Hill-Climbing (w backup)** **Beam** **IDA***	

Beam

Expand all Q elements; Keep the k best extensions (sorted by heuristic value)

	Q	
1	~~(10 S)~~	
2		



Idea: Incrementally expand the k best paths

Heuristic Values

A=2	C=1	S=10
B=3	D=4	G=0

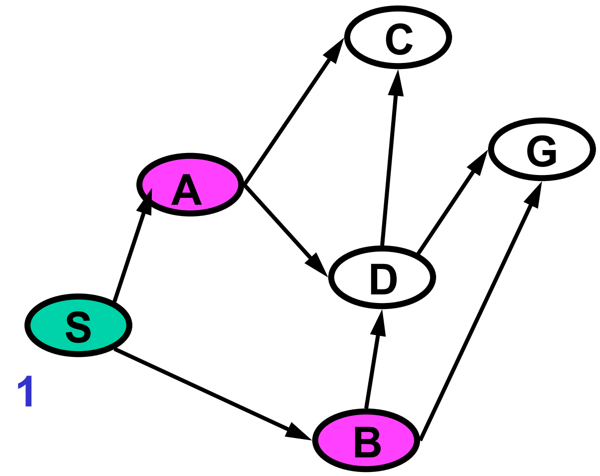
Added paths in blue; heuristic value of head is in front.

Let k = 2

Beam

Expand all Q elements; Keep the **k best extensions** (sorted by **heuristic value**)

	Q	
1	~~(10 S)~~	
2	(2 A S) (3 B S)	



Idea: Incrementally expand the k best paths

Heuristic Values

A=2	C=1	S=10
B=3	D=4	G=0

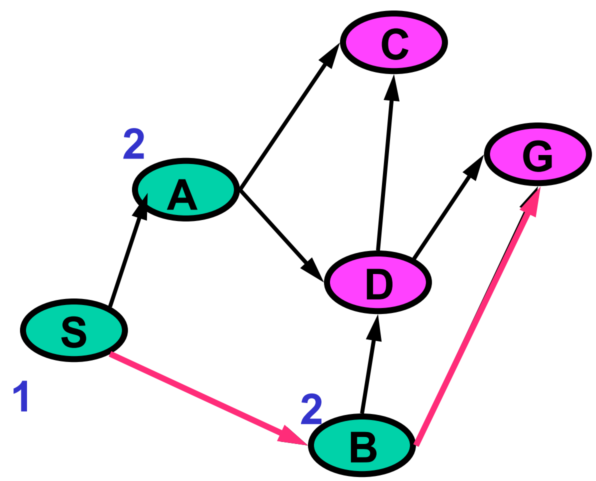
Added paths in blue; heuristic value of head is in front.

Let k = 2

Beam

Expand all Q elements; Keep the **k best extensions** (sorted by **heuristic value**)

	Q	
1	~~(10 S)~~	
2	~~(2 A S)~~ ~~(3 B S)~~	
3	(0 G B S) (1 C A S) ~~(4 D A S) (4 D B S)~~ Keep k best	



Idea: Incrementally expand the k best paths

Heuristic Values

A=2	C=1	S=10
B=3	D=4	G=0

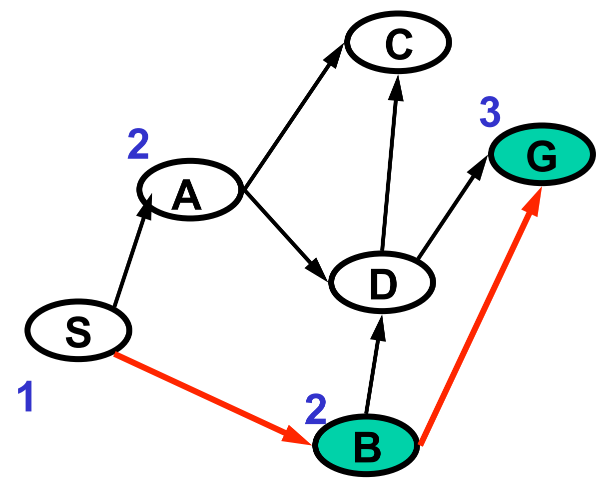
Added paths in blue; heuristic value of head is in front.

Let k = 2

Beam

Expand all Q elements; Keep the **k best extensions** (sorted by **heuristic value**)

	Q	
1	~~(10 S)~~	
2	~~(2 A S)~~ ~~(3 B S)~~	
3	(0 G B S) (1 C A S) ~~(4 D A S) (4 D B S)~~ Keep k best	



Idea: Incrementally expand the k best paths

Heuristic Values

A=2	C=1	S=10
B=3	D=4	G=0

Added paths in blue; heuristic value of head is in front.

Let k = 2

Cost and Performance

Searching a tree with branching factor b, solution depth d, and max depth m

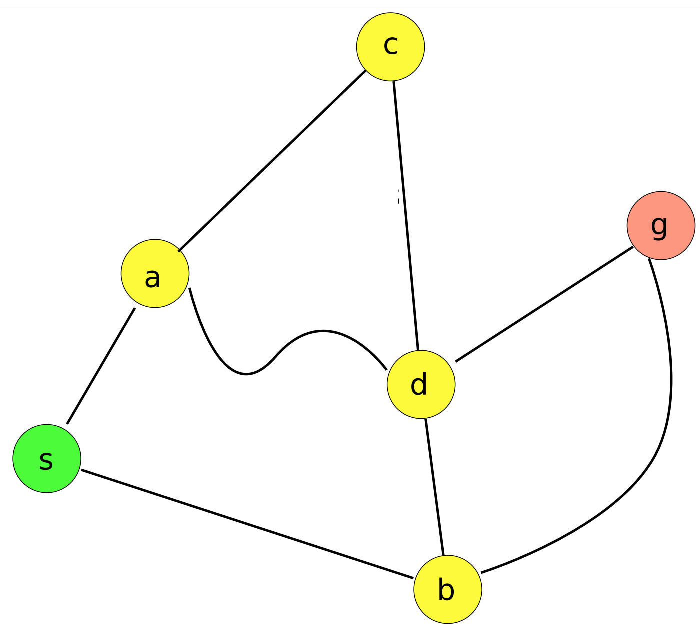
Search Method	Worst Time	Worst Space	Guaranteed to find a path?	Optimal?
Depth-First	b^m	b*m	Yes	No
Breadth-First	b^{d+1}	b^{d+1}	Yes	Yes for unit edge cost
Best-First	b^{d+1}	b^{d+1}	Yes	Yes if uniform cost or A* w admissible heuristic.
Beam (beam width = k)	k*b*m	k*b	No	No
Hill-Climbing (no backup)	b*m	b	No	No
Hill-Climbing (backup)	b^m	b*m	Yes	No

Worst case time is proportional to number of nodes visited
Worst case space is proportional to maximal length of Q

Appendices

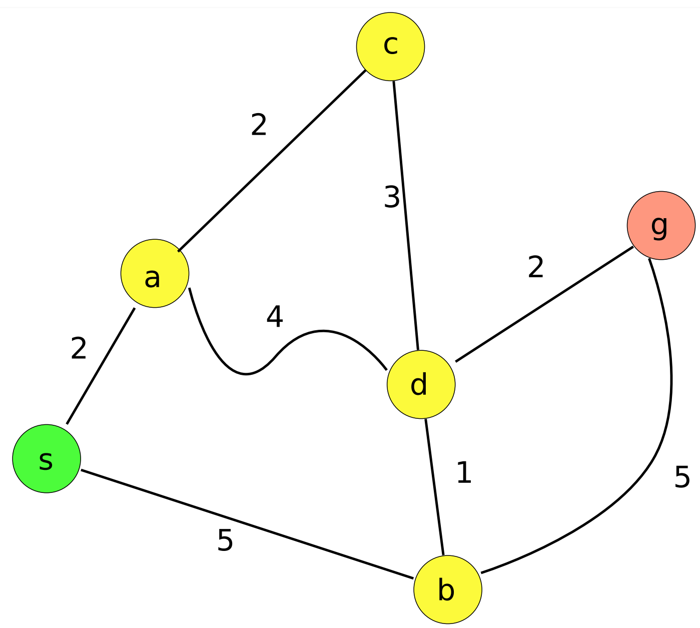
- Bounding
- Variants
- More about Informed Search.
- Dynamic Programming.

Breadth-first search: an example



- Optimal (shortest) path <s,b,g>
- Sub-optimal path <s,a,d,g>, …

Uniform-cost search: an example



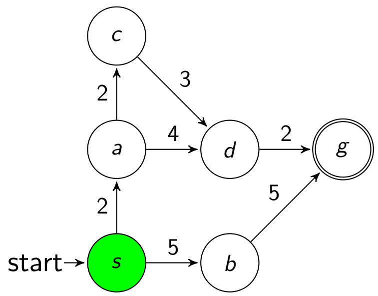
Uniform-cost search

```
Q ← ⟨start⟩ ;          // Initialize the queue with the starting node
while Q is not empty do
    Pick (and remove) the path P with lowest cost g = w(P) from the queue Q ;
    if  head(P) = goal then return P ;                    // Reached the goal
    foreach vertex v such that (head(P), v) ∈ E, do               //for all neighbors
        add ⟨v, P⟩ to the queue Q ;                        // Add expanded paths
return FAILURE ;                                   // Nothing left to consider.
```

A trace of UCS execution

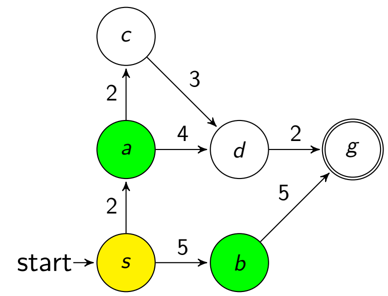
Q:

path	cost
$\langle s \rangle$	0



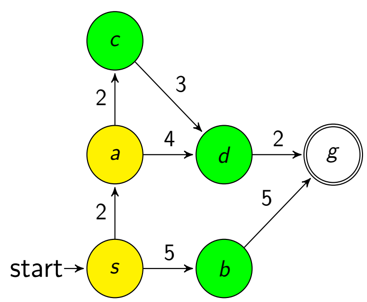
Q:

path	cost
$\langle a, s \rangle$	2
$\langle b, s \rangle$	5



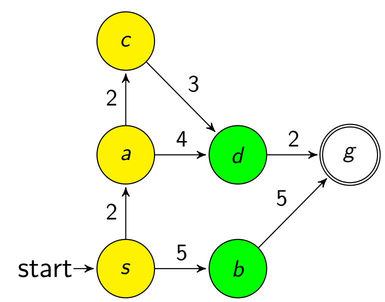
Q:

state	cost
$\langle c, a, s \rangle$	4
$\langle b, s \rangle$	5
$\langle d, a, s \rangle$	6



Q:

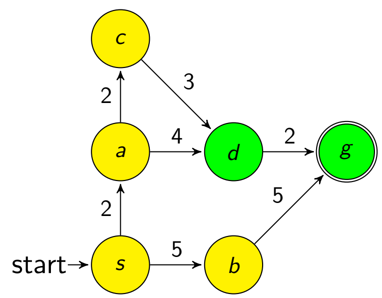
state	cost
$\langle b, s \rangle$	5
$\langle d, a, s \rangle$	6
$\langle d, c, a, s \rangle$	7



A trace of UCS execution

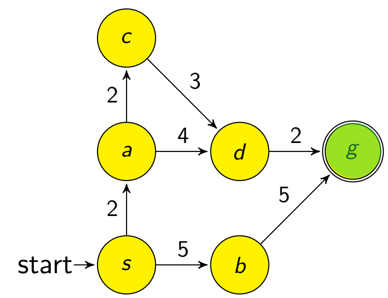
Q:

state	cost
$\langle d, a, s \rangle$	6
$\langle d, c, a, s \rangle$	7
$\langle g, b, s \rangle$	10



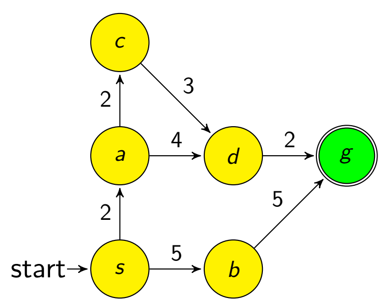
Q:

state	cost
$\langle d, c, a, s \rangle$	7
$\langle g, d, a, s \rangle$	8
$\langle g, b, s \rangle$	10



Q:

state	cost
$\langle g, d, a, s \rangle$	8
$\langle g, d, c, a, s \rangle$	9
$\langle g, b, s \rangle$	10



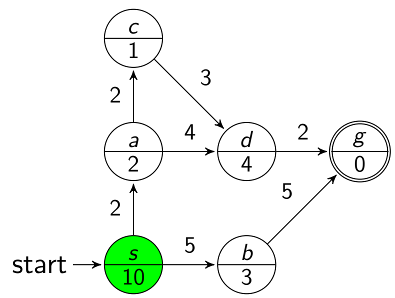
Greedy (best-first) search

```
Q ← ⟨start⟩;            // Initialize the queue with the starting node
while Q is not empty do
    Pick the path P with minimum heuristic cost h(head(P)) from the queue Q;
    if head(P) = goal then return P ;        // We have reached the goal
    foreach vertex v such that (head(P), v) ∈ E, do
        add ⟨v, P⟩ to the queue Q;
return FAILURE ;                          // Nothing left to consider.
```

A trace of GS execution

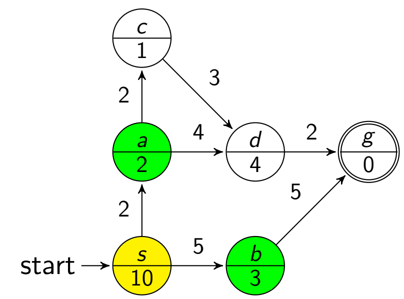
Q:

path	cost	h
$\langle s \rangle$	0	10



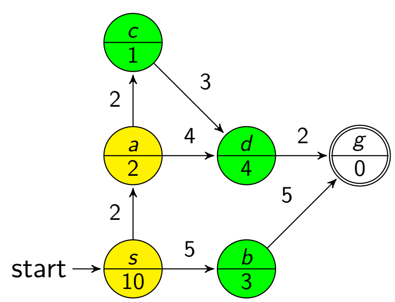
Q:

path	cost	h
$\langle a, s \rangle$	2	2
$\langle b, s \rangle$	5	3



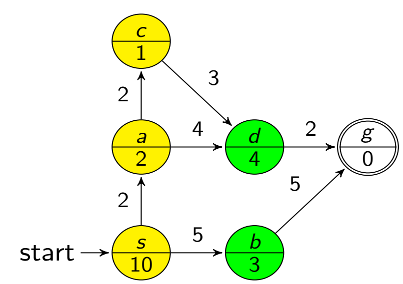
Q:

path	cost	h
$\langle c, a, s \rangle$	4	1
$\langle b, s \rangle$	5	3
$\langle d, a, s \rangle$	6	4



Q:

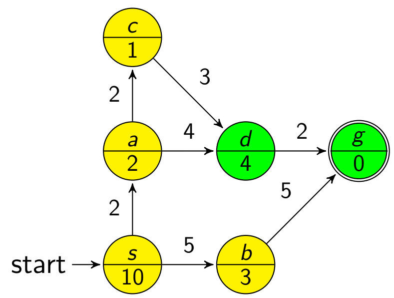
path	cost	h
$\langle b, s \rangle$	5	3
$\langle d, a, s \rangle$	6	4
$\langle d, c, a, s \rangle$	7	4



A trace of GS execution

Q:

path	cost	h
$\langle g, b, s\rangle$	10	0
$\langle d, a, s\rangle$	6	4
$\langle d, c, a, s\rangle$	7	4



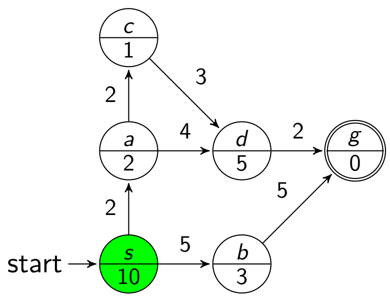
A search

```
Q ← ⟨start⟩;            // Initialize the queue with the starting node
while Q is not empty do
    Pick the path P with minimum estimated cost f(P) = g(P) + h(head(P))
    from the queue Q;
    if head(P) = goal then return P ;        // We have reached the goal
    foreach vertex v such that (head(P), v) ∈ E, do
        add ⟨v, P⟩ to the queue Q;
return FAILURE ;                             // Nothing left to consider.
```

A trace of A search execution

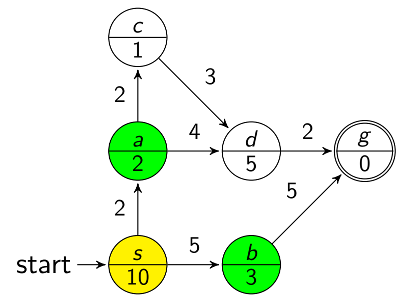
Q:

path	g	h	f
$\langle s \rangle$	0	10	10



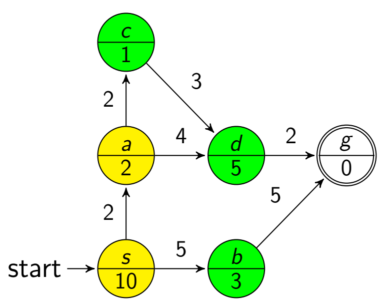
Q:

path	g	h	f
$\langle a, s \rangle$	2	2	4
$\langle b, s \rangle$	5	3	8



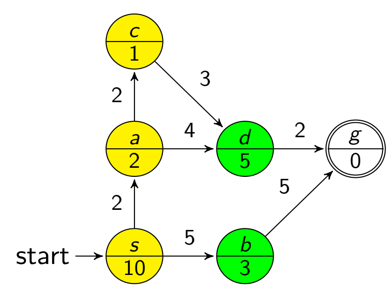
Q:

path	g	h	f
$\langle c, a, s \rangle$	4	1	5
$\langle b, s \rangle$	5	3	8
$\langle d, a, s \rangle$	6	5	11



Q:

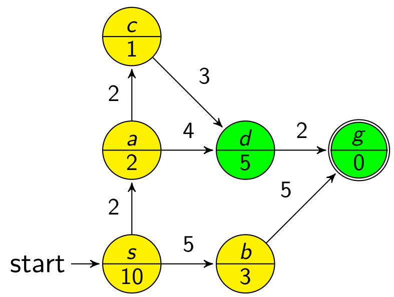
path	g	h	f
$\langle b, s \rangle$	5	3	8
$\langle d, a, s \rangle$	6	5	11
$\langle d, c, a, s \rangle$	7	5	12



A trace of A search execution

Q:

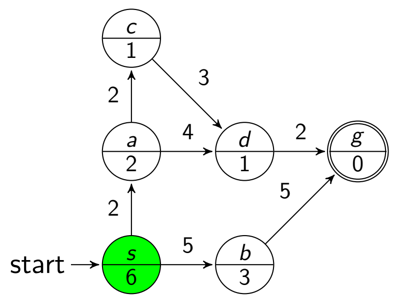
path	g	h	f
$\langle g, b, s \rangle$	10	0	10
$\langle d, a, s \rangle$	6	5	11
$\langle d, c, a, s \rangle$	7	5	12



A trace of A* search execution

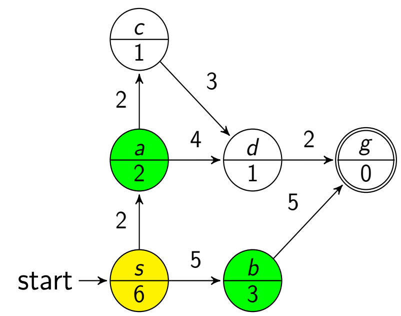
Q:

path	g	h	f
⟨s⟩	0	6	6



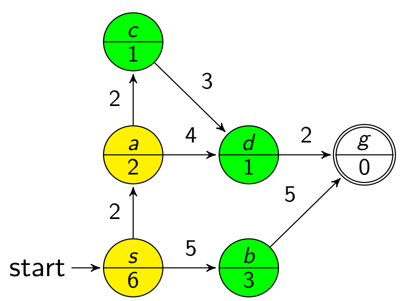
Q:

path	g	h	f
⟨a, s⟩	2	2	4
⟨b, s⟩	5	3	8



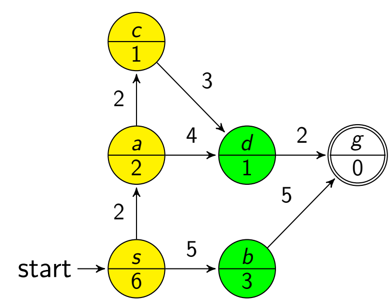
Q:

path	g	h	f
⟨c, a, s⟩	4	1	5
⟨d, a, s⟩	6	1	7
⟨b, s⟩	5	3	8



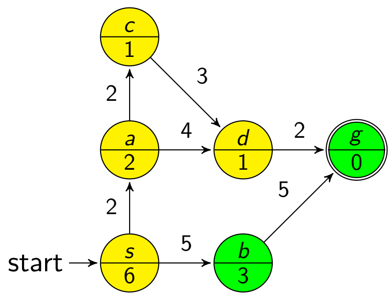
Q:

path	g	h	f
⟨d, a, s⟩	6	1	7
⟨b, s⟩	5	3	8
⟨d, c, a, s⟩	7	1	8



A trace of A* search execution

	path	g	h	f
Q:	$\langle g, d, a, s \rangle$	8	0	8
	$\langle b, s \rangle$	5	3	8
	$\langle d, c, a, s \rangle$	7	1	8



Appendices

- Bounding
- Variants
- More about Informed Search.
- Dynamic Programming.

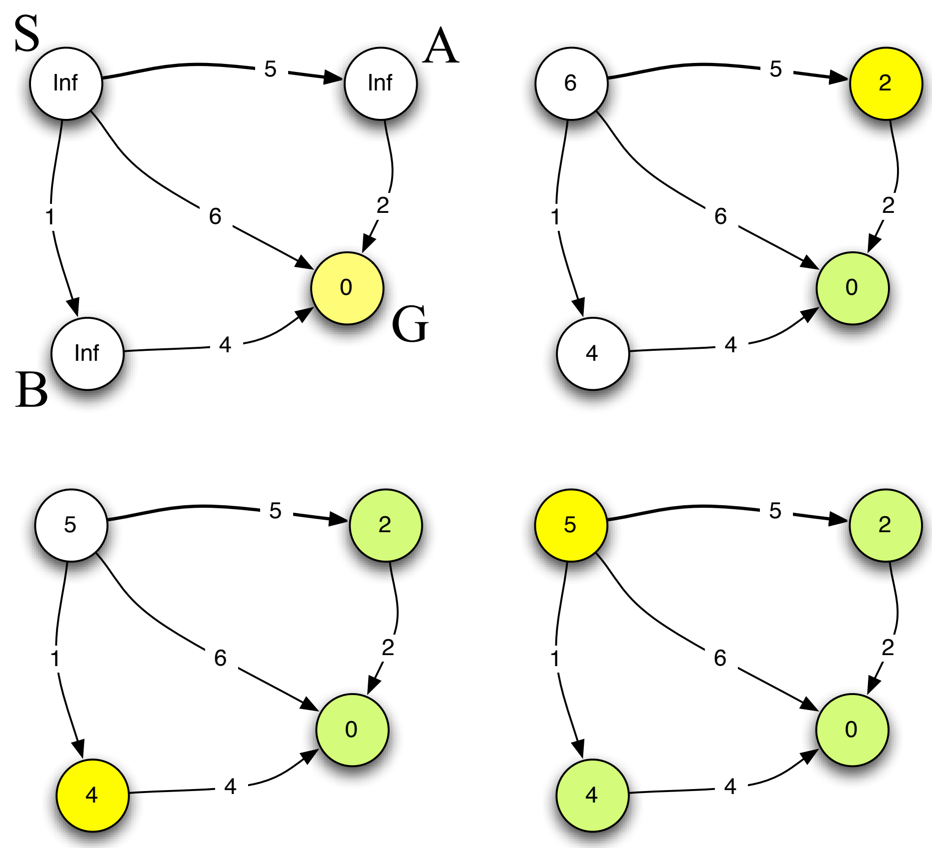
Dynamic programming

- Search algorithms work **towards** the goal.
  Hence the need for the heuristic $h(v)$.
- What if we work **backwards** from the goal?
  $h(G)=0$, and $h(v)$ becomes available when needed.
- Bellman's **dynamic programming** principle:

$$h^*(u) = \min_{(u,v)\in E} \left[w(\ (u,v)\ ) + h^*(v) \right].$$

  – Shortest paths computed from smaller shortest paths.

DP example



Comparison of A* and DP

A*

- Search towards the goal, guided by a heuristic.
- Fast if the heuristic is good.
- Find the optimal path from the start node to the goal node.
- Provide open-loop control.

Dynamic programming

- Work backwards from the goal.
- Slower.
- Find the optimal path from every node to the goal node.
- Provide closed-loop feedback control.