

Global Path Planning via Optimal Search and Shortest Paths

Brian C. Williams
16.410 / 13
Session 6

Slides draw upon material from:
6.034 Tomas Lozano Perez and Winston,
Russell and Norvig AIMA
Cormen, Leiserson and Rivest ItA

Brian Williams, Fall 05

1

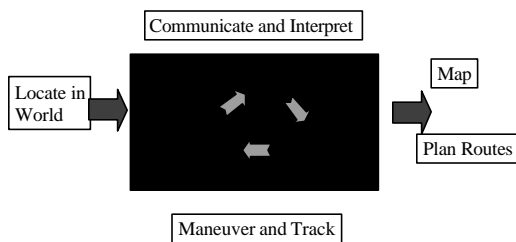
Assignment

- Reading:
 - Lecture notes plus...
 - Informed search and exploration:
AIMA Ch. 4.1-2
 - Computing Shortest Paths:
Cormen, Leiserson & Rivest, (optional)
“Introduction to Algorithms” Ch. 25.1-2
- Problem Sets:
 - PS #2 due today,
 - PS#3 out today. Due Wednesday, October 5th.

Brian Williams, Fall 05

2

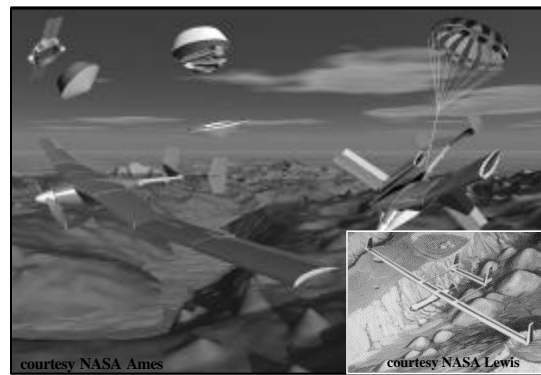
Agent Architecture and Building Blocks



Brian Williams, Fall 05

3

How Do We Maneuver?



Questions About Maneuvering

- What is the most effective route?
 - ⇒ Today
- How do we quickly compensate for error?
 - ⇒ Today (time permitting)
- How do we anticipate adversaries?
 - ⇒ Monday
- How do we anticipate the effects of error?
 - ⇒ Monday

Brian Williams, Fall 05

6

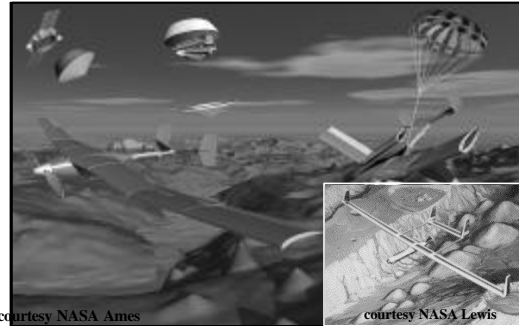
L6 and L7: Global Maneuvering using Roadmaps



Brian Williams, Fall 05

7

L8 and L9: Fine-grained Maneuvering using Linear Programs



courtesy NASA Ames

courtesy NASA Lewis

Answering Questions Using Roadmaps

- What is the most effective route?
⇒ Find the shortest path from S to G
- How do we quickly compensate for error?
⇒ Precompute action policies (shortest path trees)
- How do we anticipate adversaries?
⇒ Perform adversarial game tree search
- How do we anticipate the effects of error?
⇒ Perform stochastic game tree search

Brian Williams, Fall 05

9

Outline

Today:

- Encoding roadmaps as weighted graphs
- Finding a shortest path
 - Best-first search
 - Branch and bound
 - Approximate methods (appendix)
- Compensating for error using policies
 - Computing shortest path trees (optional)



Next Lecture:

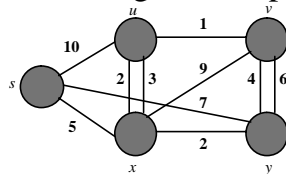
- Anticipating adversaries
- Anticipating environmental error
- Generating road maps



Brian Williams, Fall 05

Courtesy of NASA.

Roadmaps are Formalized as Weighted Graphs



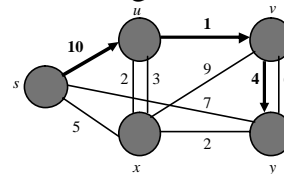
Graph $G = \langle V, E \rangle$

Weight function $w: E \rightarrow \mathbb{R}$

Brian Williams, Fall 05

11

Weighted Graphs Have Weighted Paths



Path

Path weight

$p = \langle v_0, v_1, \dots, v_k \rangle$

$w(p) = \sum w(v_{i-1}, v_i)$

Example:

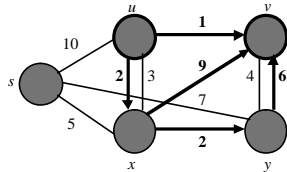
$p = \langle s, u, v, y \rangle$

$w(p) = 10 + 1 + 4$

Brian Williams, Fall 05

12

Weighted Graphs Have Shortest Paths



Shortest path weight $d(u,v) = \min \{ w(p) : u \stackrel{?}{\sim} p \vee \}$ else 8

Example: $d(u,v) = \min \{ w(\langle u,v \rangle), w(\langle u, x, v \rangle), w(\langle u,x,y,v \rangle) \}$
 $= \min \{ 1, 2+9, 2+2+6 \}$
 $= 1$

Brian Williams, Fall 05

13

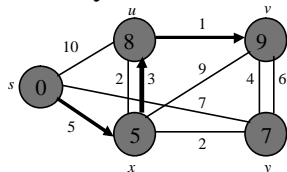
Notational Note

- x (lower case) An element
- S (upper case) A set
- $x \text{ in } S$ "x is an element of set S"
- $\{x : p(x)\}$ "The set of all elements x such that p(x) holds."
- $u \stackrel{?}{\sim} p \vee$ "A path p from u to v."

Brian Williams, Fall 05

14

Shortest Paths Contain Only Shortest Paths



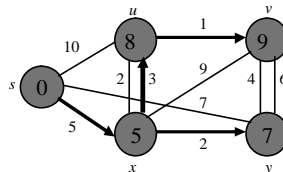
• Subpaths of shortest paths are shortest paths.

- $s \stackrel{?}{\sim} p \vee$ = $\langle s, x, u, v \rangle$ Shortest
- $s \stackrel{?}{\sim} p u$ = $\langle s, x, u \rangle$ Shortest
- $s \stackrel{?}{\sim} p x$ = $\langle s, x \rangle$ Shortest
- $x \stackrel{?}{\sim} p v$ = $\langle x, u, v \rangle$ Shortest
- $x \stackrel{?}{\sim} p v$ = $\langle x, u \rangle$ Shortest
- $u \stackrel{?}{\sim} p v$ = $\langle u, v \rangle$ Shortest

Brian Williams, Fall 05

15

Corollary: Shortest Paths Can Be Grown From Shortest Paths



The length of shortest path $s \stackrel{?}{\sim} p u \stackrel{?}{\sim} v$

is $d(s,v) = d(s,u) + w(u,v)$.

- " $\langle u,v \rangle \in E$ $d(s,v) = d(s,u) + w(u,v)$

Brian Williams, Fall 05

16

Finding A Shortest Path

Input: $\langle Gr, w, S, G \rangle$, where

- Gr is a (directed) graph $\langle V, E \rangle$ with weight function w ,
- $S \in V$ is the Start and $G \in V$ is the Goal.

Output:

A simple path $P = \langle v_1, v_2 \dots v_n \rangle$ from S to G, with the shortest path weight $d(S,G)$, and its corresponding weight.

Brian Williams, Fall 05

17

Finding All Shortest Paths

Input: $\langle Gr, w, S \rangle$, where

- Gr is a (directed) graph $\langle V, E \rangle$ with weight function w ,
- $S \in V$ is the Start.

Output for Single Source Shortest Path:



A weighted tree containing, for every $G_i \in V$, a simple path $P = \langle v_1, v_2 \dots v_n \rangle$ from S to G_i , that has the shortest path weight $d(S,G)$.

Brian Williams, Fall 05

18

Outline

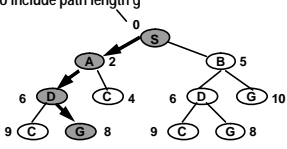
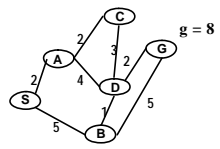
- Encoding roadmaps as weighted graphs
- Finding a shortest path
 - Best-first search
 - Uniform cost search
 - Greedy search
 - A* search
 - Branch and bound
 - Approximate methods (Appendix)
- Compensating for error using policies

Brian Williams, Fall 05

Optimal Search

Extend search tree nodes to include path length g

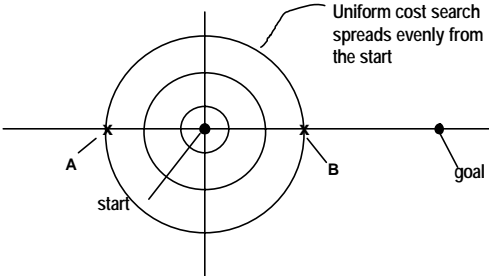
Problem: Find the path to the goal G with the shortest path length g.

Brian Williams, Fall 05

Classes of Search

Blind (uninformed)	Depth-First	Systematic exploration of whole tree
	Breadth-First	until the goal is found.
	Iterative-Deepening	
Best-first (informed)	Uniform cost	Using path "length" as a measure,
	Greedy	finds "shortest" path.
	A*	

Brian Williams, Fall 05




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

Brian Williams, Fall 05

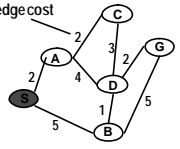
Uniform Cost

path length

0



edge cost




Enumerates partial paths in order of increasing path length g.

Brian Williams, Fall 05

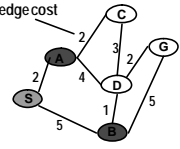
Uniform Cost

path length

0



edge cost



Enumerates partial paths in order of increasing path length g.

Brian Williams, Fall 05

Uniform Cost

path length

edge cost

Enumerates partial paths in order of increasing path length g.

Brian Williams, Fall 05 25

Uniform Cost

path length

edge cost

Enumerates partial paths in order of increasing path length g.

Brian Williams, Fall 05 26

Uniform Cost

path length

edge cost

Enumerates partial paths in order of increasing path length g.

Brian Williams, Fall 05 27

Uniform Cost

path length

edge cost

Better path visited later
Enumerates partial paths in order of increasing path length g.

Brian Williams, Fall 05 28

Uniform Cost

path length

edge cost

Better path visited later Expands nodes already visited
Enumerates partial paths in order of increasing path length g.
May expand vertex more than once.

Brian Williams, Fall 05 29

Uniform Cost

path length

edge cost

Better path visited later Expands nodes already visited
Enumerates partial paths in order of increasing path length g.
May expand vertex more than once.

Brian Williams, Fall 05 30

Uniform Cost

path length

Best path **expanded** first

edge cost

Expands nodes already visited

Enumerates partial paths in order of increasing path length g .

May expand vertex more than once.

Brian Williams, Fall 05 31

Why Expand a Vertex More Than Once?

path length

edge cost

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

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

Brian Williams, Fall 05 32

Why Expand a Vertex More Than Once?

path length

edge cost

- 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 expanded only the first path that visits each vertex X?

Brian Williams, Fall 05 33

Why Expand a Vertex More Than Once?

path length

edge cost

- 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 expanded only the first path to visit each vertex X?

Brian Williams, Fall 05 34

Why Expand a Vertex More Than Once?

path length

edge cost

- 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 to visit each vertex X?
 ⇒ **Eliminate the Visited List**

Brian Williams, Fall 05 35

Uniform Cost Search Algorithm

Let GR be a Graph Let Q be a list of simple partial paths in GR
 Let S be the start vertex in GR and Let G be a Goal vertex in GR.
 Let g be the path weight from S to N.

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 05 36

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

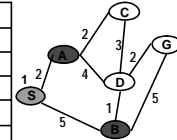
Brian Williams, Fall 05

37

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.

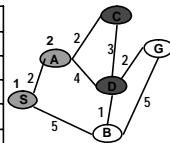
Brian Williams, Fall 05

38

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.

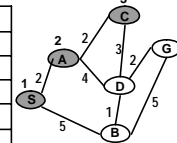
Brian Williams, Fall 05

39

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.

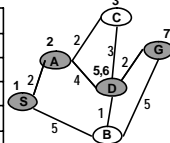
Brian Williams, Fall 05

40

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)



Brian Williams, Fall 05

41

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 beenqueued.
- 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.

Brian Williams, Fall 05

42

Implementing the Search Strategies

Depth-first:	Pick first element of Q Add path extensions to front of Q	Uses visited list
Breadth-first:	Pick first element of Q Add path extensions to end of Q	Uses visited list
Uniform-cost:	Pick first element of Q Add path extensions to Q in increasing order of path weight g.	No visited list
Best-first: (generalizes uniform-cost)	Pick first element of Q Add path extensions in increasing order of any cost function f	No visited list

Brian Williams, Fall 05

43

Best-first Search Algorithm

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 be a cost function on N.

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.

Brian Williams, Fall 05

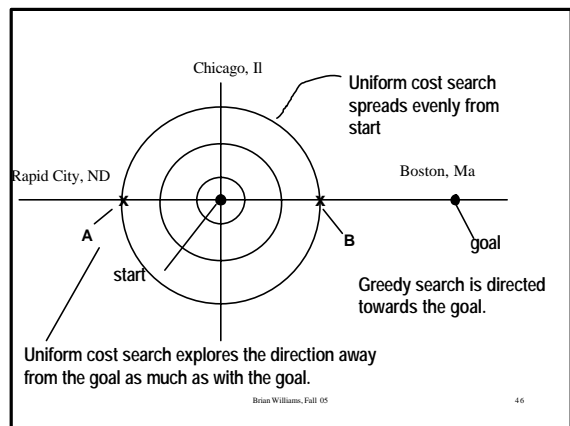
44

Classes of Search

Blind (uninformed)	Depth-First	Systematic exploration of whole tree until the goal is found.
	Breadth-First	
	Iterative-Deepening	
Best-first	Uniform-cost	Using path "length" as a measure, finds "shortest" path.
	Greedy	
	A*	

Brian Williams, Fall 05

45



Brian Williams, Fall 05

46

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 paths 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.

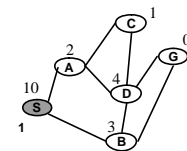
Brian Williams, Fall 05

47

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. Order of nodes in blue.

Brian Williams, Fall 05

48

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		

Heuristic values in red
Order of nodes in blue.

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

Brian Williams, Fall 05 49

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		

Heuristic values in red
Order of nodes in blue.

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

Brian Williams, Fall 05 50

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		

Heuristic values in red
Order of nodes in blue.

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

Brian Williams, Fall 05 51

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.

Brian Williams, Fall 05 52

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.

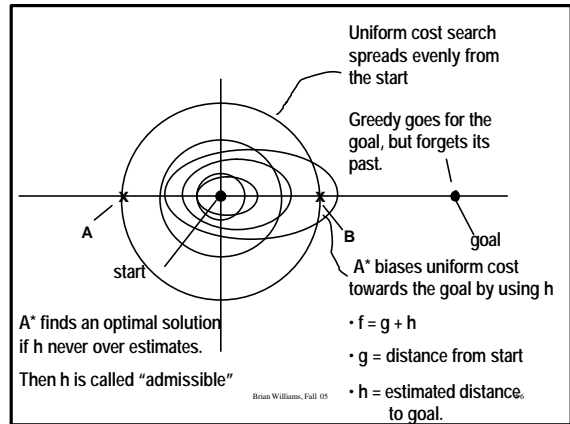
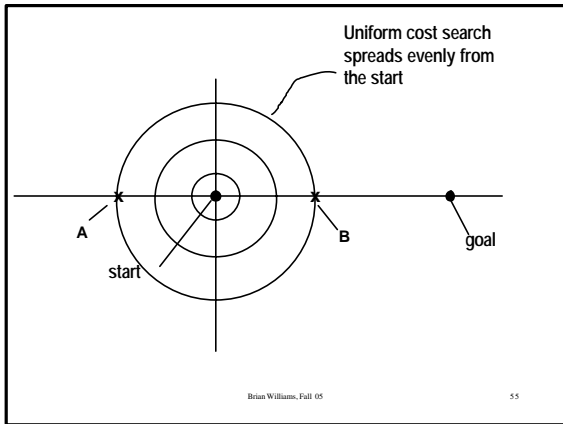
Did Greedy search produce the shortest path?

Brian Williams, Fall 05 53

Classes of Search

Blind (uninformed)	Depth-First	Systematic exploration of whole tree
	Breadth-First	until the goal is found.
	Iterative-Deepening	
Best-first	Uniform-cost	Using path "length" as a measure,
	Greedy	finds "shortest" path.
	A*	

Brian Williams, Fall 05 54



Simple Optimal Search Algorithm

BFS + Admissible Heuristic

Let GR be a Graph Let Q be a list of simple partial paths in GR
 Let S be the start vertex in GR and 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.

Brian Williams, Fall 05

57

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.

Then h is called "admissible"

Brian Williams, Fall 05

58

Admissible heuristics for 8 puzzle?

S

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)

Brian Williams, Fall 05

59

Can We Prune Search?

Recall: Shortest Paths Can Be Grown From Shortest Paths

Dynamic Programming Principle: Exploiting solutions to shared subproblems can produce exceptional speedup.

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

- When BFS dequeues the first partial path with head node U, this path is 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.

Brian Williams, Fall 05

60

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 and 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.

Brian Williams, Fall 05

61

A* Optimal Search Algorithm

BFS + Dyn Prog + Admissible Heuristic

Let GR be a Graph Let Q be a list of simple partial paths in GR
 Let S be the start vertex in GR and 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 the one-step extensions of N to each child.
7. Add to Q all the extended paths.
8. Go to step 2.

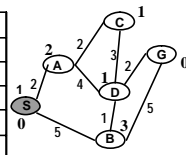
Brian Williams, Fall 05

62

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.

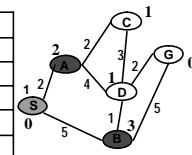
Brian Williams, Fall 05

63

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

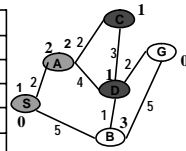
Brian Williams, Fall 05

64

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

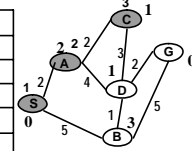
Brian Williams, Fall 05

65

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

Brian Williams, Fall 05

66

A* (BFS + DynProg + Admissible Heuristic)

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

	Q	Expanded
1	(S)	
2	(A S) (8 BS)	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

Brian Williams, Fall 05 67

A* (BFS + DynProg + Admissible Heuristic)

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

	Q	Expanded
1	(S)	
2	(A S) (8 BS)	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 BS)	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

Brian Williams, Fall 05 68

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	$b \cdot m$	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

Brian Williams, Fall 05 69

Classes of Search

Blind (uninformed)	Depth-First	Systematic exploration of whole tree until the goal is found.
	Breadth-First	
	Iterative-Deepening	
Best-first	Uniform-cost	Uses path "length" measure. Finds "shortest" path.
	Greedy	
	A*	
Bounding	Branch and Bound	Prunes suboptimal branches
	Alpha/Beta (L7)	Prunes options the adversary rules out

Brian Williams, Fall 05 70

Branch and Bound

- Maintain the best solution found thus far (incumbent).
- 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

Brian Williams, Fall 05 71

Branch and Bound

- Maintain the best solution found thus far (incumbent).
- 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

Brian Williams, Fall 05 72

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 and 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 $U = \infty, P = ()$;
2. If Q is empty, return Incumbent U and P ,
Else, remove a partial path N from Q ;
3. If $f(N) \geq U$, Go to step 2.
4. If $\text{head}(N) = G$, then $U = f(N)$ and $P = N$ (a better path to the goal)
5. (Else) Find all children of $\text{head}(N)$ (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.

Brian Williams, Fall 05

73

Classes of Search

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

Brian Williams, Fall 05

74

Outline

Today:

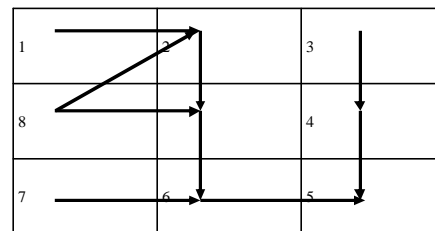
- Encoding roadmaps as weighted graphs
- Finding a shortest path
- Compensating for error using policies
 - Policies as shortest path trees
 - Relaxing a bound
 - Dijkstra's algorithm



Brian Williams, Fall 05

Courtesy of NASA.

Compensating for Error Using Policies



- Policy, $p(v) ? e$, dictates how to act in all states.
- Policy p corresponds to a shortest path tree from all vertices to the destination.

Brian Williams, Fall 05

76

Finding All Shortest Paths

Input: $\langle Gr, w, S \rangle$, where

- Gr is a (directed) graph $\langle V, E \rangle$
with weight function w ,
- $S \in V$ is the Start.

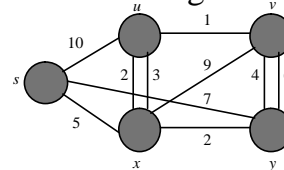
Output for Single Source Shortest Path:

A weighted tree containing, for every $G_i \in V$,
a simple path $P = \langle v_1, v_2 \dots v_n \rangle$ from S to G_i ,
that has the shortest path weight $d(S, G_i)$.

Brian Williams, Fall 05

77

Shortest Path Tree From a Single Source



Problem: Compute shortest path to all vertices from source s

Brian Williams, Fall 05

78

Shortest Path Tree From a Single Source

Problem: Compute shortest path to all vertices from source s

- estimate $d[v]$ estimated shortest path length from s to v

Brian Williams, Fall 05 79

Shortest Path Tree From a Single Source

Algorithm: Single-Source Shortest Path

Problem: Compute shortest path to all vertices from source s



- estimate $d[v]$ estimated shortest path length from s to v
- predecessor $p[v]$ final edge of shortest path to v
- induces *shortest path tree*

Brian Williams, Fall 05 80

Outline

Today:

- Encoding roadmaps as weighted graphs
- Finding a shortest path
- Compensating for error using policies
 - Policies as shortest path trees
 - **Relaxing a bound**
 - Dijkstra's algorithm

Brian Williams, Fall 05

Idea: Start With Upper Bound

Initialize-Single-Source(G, s)

1. for each vertex $v \in V[G]$
2. do $d[v] ? 8$
3. p[v] ? NIL
4. $d[s] ? 0$

$O(v)$

Brian Williams, Fall 05 82

Relax Bounds to Shortest Path

Relax (u, v, w)

1. if $d[u] + w(u,v) < d[v]$
2. do $d[v] ? d[u] + w(u,v)$
3. p[v] ? u

Brian Williams, Fall 05 83

Properties of Relaxing Bounds

After calling Relax(u, v, w)

- $d[u] + w(u,v) = d[v]$

d remains a shortest path upperbound after repeated calls to Relax.



Once $d[v]$ is the shortest path value the value persists

Brian Williams, Fall 05 84

Outline

Today:

- Encoding roadmaps as weighted graphs
- Finding a shortest path
- Compensating for error using policies
 - Policies as shortest path trees
 - Relaxing a bound
 - **Dijkstra's algorithm**

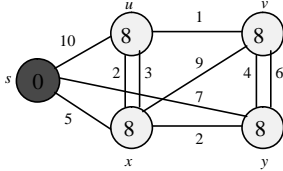



Brian Williams, Fall 05

Courtesy of NASA.

Dijkstra's Algorithm

Assume all edges are non-negative.
Idea: Greedily relax all out arcs of each minimum cost, unrelaxed node

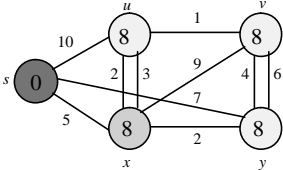


$Q = \{s, u, v, x, y\}$ Vertices to relax
 $S = \{s\}$ Vertices with shortest path value

Brian Williams, Fall 05 86

Dijkstra's Algorithm

Assume all edges are non-negative.
Idea: Greedily relax all out arcs of each minimum cost, unrelaxed node

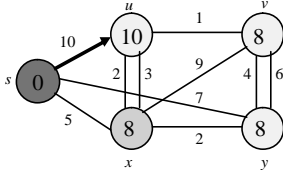


$Q = \{x, y, u, v\}$ Vertices to relax
 $S = \{s\}$ Vertices with shortest path value

Brian Williams, Fall 05 87

Dijkstra's Algorithm

Assume all edges are non-negative.
Idea: Greedily relax all out arcs of each minimum cost, unrelaxed node

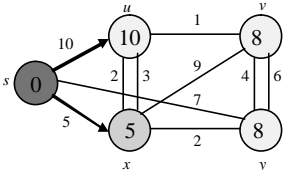


$Q = \{x, y, u, v\}$ Vertices to relax
 $S = \{s\}$ Vertices with shortest path value
Shortest path edge ? [v] = bold arrows

Brian Williams, Fall 05 88

Dijkstra's Algorithm

Assume all edges are non-negative.
Idea: Greedily relax all out arcs of each minimum cost, unrelaxed node

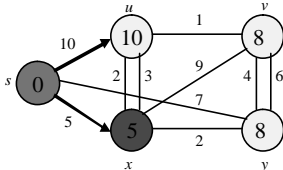


$Q = \{x, y, u, v\}$ Vertices to relax
 $S = \{s\}$ Vertices with shortest path value
Shortest path edge ? [v] = arrows

Brian Williams, Fall 05 89

Dijkstra's Algorithm

Assume all edges are non-negative.
Idea: Greedily relax all out arcs of each minimum cost, unrelaxed node



$Q = \{x, y, u, v\}$ Vertices to relax
 $S = \{s\}$ Vertices with shortest path value
Shortest path edge ? [v] = arrows

Brian Williams, Fall 05 90

Dijkstra's Algorithm

Assume all edges are non-negative.
Idea: Greedily relax all out arcs of each minimum cost, unrelaxed node

$Q = \{y, u, v\}$ Vertices to relax
 $S = \{s, x\}$ Vertices with shortest path value
 Shortest path edge? $[v] =$ arrows

Brian Williams, Fall 05 91

Dijkstra's Algorithm

Assume all edges are non-negative.
Idea: Greedily relax all out arcs of each minimum cost, unrelaxed node

$Q = \{y, u, v\}$ Vertices to relax
 $S = \{s, x\}$ Vertices with shortest path value
 Shortest path edge? $[v] =$ arrows

Brian Williams, Fall 05 92

Dijkstra's Algorithm

Assume all edges are non-negative.
Idea: Greedily relax all out arcs of each minimum cost, unrelaxed node

$Q = \{u, v\}$ Vertices to relax
 $S = \{s, x, y\}$ Vertices with shortest path value
 Shortest path edge? $[v] =$ arrows

Brian Williams, Fall 05 93

Dijkstra's Algorithm

Assume all edges are non-negative.
Idea: Greedily relax all out arcs of each minimum cost, unrelaxed node

$Q = \{v\}$ Vertices to relax
 $S = \{s, x, y, u\}$ Vertices with shortest path value
 Shortest path edge? $[v] =$ arrows

Brian Williams, Fall 05 94

Dijkstra's Algorithm

Assume all edges are non-negative.
Idea: Greedily relax all out arcs of each minimum cost, unrelaxed node

$Q = \{\}$ Vertices to relax
 $S = \{s, x, y, u, v\}$ Vertices with shortest path value
 Shortest path edge? $[v] =$ arrows

Brian Williams, Fall 05 95

Dijkstra's Algorithm

Repeatedly select minimum cost, unrelaxed node and relax out arcs

```

DIJKSTRA( $G, w, s$ )
1. Initialize-Single-Source( $G, s$ )       $O(V)$ 
2.  $S \leftarrow \emptyset$ 
3.  $Q \leftarrow V[G]$ 
4. while  $Q \neq \emptyset$ 
5.   do  $u \leftarrow \text{Extract-Min}(Q)$        $O(V)$       or  $O(\lg V)$ 
6.    $S \leftarrow S \cup \{u\}$       w fib heap
7.   for each vertex  $v \in \text{Adj}[u]$        $O(V) \cdot$ 
8.     do Relax( $u, v, w$ )       $O(E)$ 

```


$= O(V^2 + E)$
 $= O(V \lg V + E)$

Brian Williams, Fall 05

Outline


Today:

- Encoding roadmaps as weighted graphs
- Finding a shortest path
 - Best-first search
 - Branch and bound
 - Approximate methods (Appendix)**
- Compensating for error using policies
 - Computing shortest path trees



Next Lecture:

- Anticipating adversaries
- Anticipating environmental error
- Generating road maps



Brian Williams, Fall 05

Appendix: Approximations to Optimal Search

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

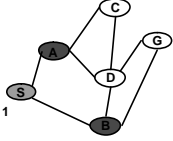
Brian Williams, Fall 05 98

Courtesy of NASA.

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

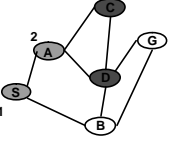
Added paths in blue; heuristic value of head is in front.

Brian Williams, Fall 05 99

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

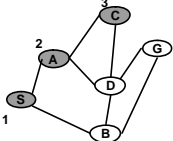
Added paths in blue; heuristic value of head is in front.

Brian Williams, Fall 05 100

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	()



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

Fails to find a path!

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

Brian Williams, Fall 05 101

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^d	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

Brian Williams, Fall 05 102

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			
Hill-Climbing (backup)				

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

Brian Williams, Fall 05

103

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		
Hill-Climbing (backup)				

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

Brian Williams, Fall 05

104

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

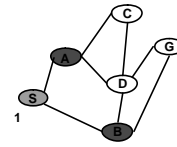
Brian Williams, Fall 05

105

Hill-Climbing (with backup)

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

	Q	
1	(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.

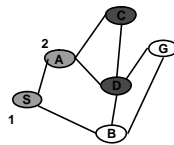
Brian Williams, Fall 05

106

Hill-Climbing (with backup)

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

	Q	
1	(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.

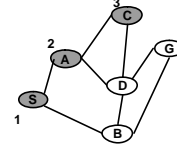
Brian Williams, Fall 05

107

Hill-Climbing (with backup)

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

	Q	
1	(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.

Brian Williams, Fall 05

108

Hill-Climbing (with backup)

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

	Q	
1	(10S)	
2	(2AS) (3BS)	
3	(1CAS) (4DAS) (3BS)	
4	(4BAS) (3BS)	
5	(0GDAS) (1CAS) (3BS)	

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.

Brian Williams, Fall 05 109

Hill-Climbing (with backup)

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

	Q	
1	(10S)	
2	(2AS) (3BS)	
3	(1CAS) (4DAS) (3BS)	
4	(4BAS) (3BS)	
5	(0GDAS) (1CAS) (3BS)	

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.

Brian Williams, Fall 05 110

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

Brian Williams, Fall 05 111

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

Brian Williams, Fall 05 112

Classes of Search

Blind (uninformed)	Depth-First	Systematic exploration of whole tree until the goal is found.
	Breadth-First	
	Iterative-Deepening	
Best-first	Uniform-cost	Uses path "length" measure. Finds "shortest" path.
	Greedy	
	A^*	
Variants	Beam	
	Hill-Climbing (w/ backup)	
	ID A^*	

Brian Williams, Fall 05 113

Beam

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

	Q	
1	(10S)	
2		

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

Idea: Incrementally expand the k best paths

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

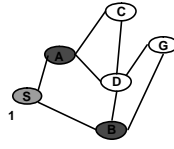
Let $k = 2$

Brian Williams, Fall 05 114

Beam

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

	Q	
1	(10S)	
2	(2AS) (3BS)	



Heuristic Values

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

Idea: Incrementally expand the k best paths

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

Let $k = 2$

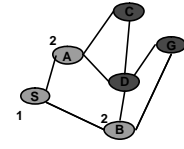
Brian Williams, Fall 05

115

Beam

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

	Q	
1	(10S)	
2	(2AS) (3BS)	
3	(0GBS) (1CAS) Keep (4DAS) (4DBS) k best	



Heuristic Values

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

Idea: Incrementally expand the k best paths

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

Let $k = 2$

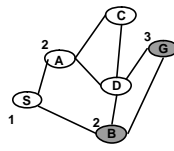
Brian Williams, Fall 05

116

Beam

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

	Q	
1	(10S)	
2	(2AS) (3BS)	
3	(0GBS) (1CAS) Keep (4DAS) (4DBS) k best	



Heuristic Values

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

Idea: Incrementally expand the k best paths

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

Let $k = 2$

Brian Williams, Fall 05

117

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

Brian Williams, Fall 05

118

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^m		
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

Brian Williams, Fall 05

119

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)		$k^m b^m$		
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

Brian Williams, Fall 05

120

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	$b \cdot m$	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)	$k \cdot b \cdot m$	$k \cdot b$	No	No
Hill-Climbing (no backup)	$b \cdot m$	b	No	No
Hill-Climbing (backup)	b^m	$b \cdot m$	Yes	No

Worst case time is proportional to number of nodes visited

Worst case space is proportional to maximal length of Q

Brian Williams, Fall '05

121