CSE 473: Artificial Intelligence
Spring 2012

Search: Cost & Heuristics

Dan Weld

With slides from
Dan Klein, Stuart Russell, Andrew Moore, Luke Zettlemoyer

Announcements

Project 1: Search
Due Wed 4/11

Start!

Search thru a Problem Space / State Space

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state [test]
- Output:
  - Path: start ⇒ a state satisfying goal test
  - [May require shortest path]
  - [Sometimes just need state passing test]

Graduation?

- Getting a BS in CSE as a search problem?
  (don't think too hard)
- Space of States
- Operators
- Initial State
- Goal State

4

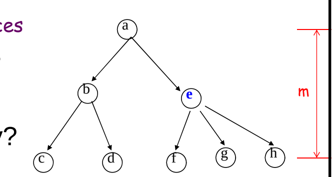
Concept Learning

Labeled Training Examples
<p1,blond,32,mc,ok>
<p2,red,47,visa,ok>
<p3,blond,23,cash,ter>
<p4,…
Output: f: <pn…> → {ok, ter}

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output:

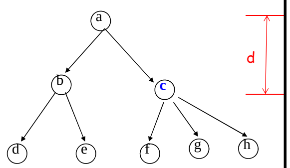
Depth First Search

- Maintain stack of nodes to visit
  - Check path to root to prune duplicates
- Evaluation
  - Complete?
    Not for infinite spaces
  - Time Complexity?
    $O(b^m)$
  - Space Complexity?
    $O(bm)$



Breadth First Search

- Maintain queue of nodes to visit
- Evaluation
  - Complete?
    Yes
  - Time Complexity?
    $O(b^d)$
  - Space Complexity?
    $O(b^d)$



Memory a Limitation?

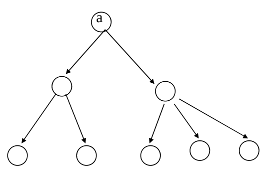
- Suppose:
  - 2 GHz CPU
  - 4 GB main memory
  - 100 instructions / expansion
  - 5 bytes / node
- 200,000 expansions / sec
  - Memory filled in 400 sec … < 7 min

© Daniel S. Weld

8

Iterative Deepening Search

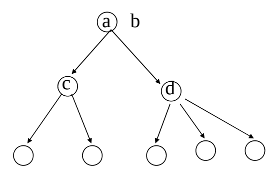
- DFS with limit; incrementally grow limit
- Evaluation
  - Complete?
  - Time Complexity?
  - Space Complexity?



9

Iterative Deepening Search

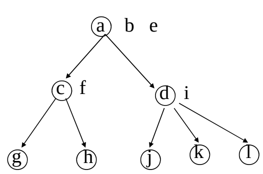
- DFS with limit; incrementally grow limit
- Evaluation
  - Complete?
  - Time Complexity?
  - Space Complexity?



10

Iterative Deepening Search

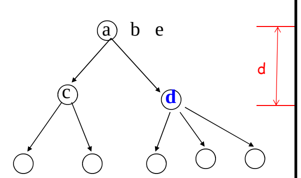
- DFS with limit; incrementally grow limit
- Evaluation
  - Complete?
  - Time Complexity?
  - Space Complexity?



11

Iterative Deepening Search

- DFS with limit; incrementally grow limit
- Evaluation
  - Complete?
    Yes
  - Time Complexity?
    $O(b^d)$
  - Space Complexity?
    $O(b^d)$



12

Cost of Iterative Deepening

b	ratio ID to DFS
2	3
3	2
5	1.5
10	1.2
25	1.08
100	1.02

Speed

Assuming 10M nodes/sec & sufficient memory

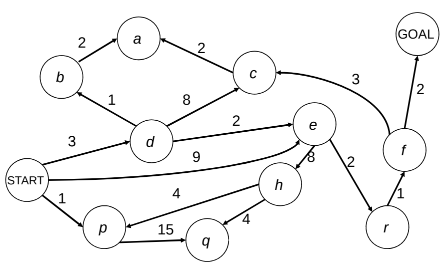
	BFS Nodes	BFS Time		Iter. Deep. Nodes	Iter. Deep. Time
8 Puzzle	10^5	.01 sec		10^5	.01 sec
2x2x2 Rubik's	10^6	.2 sec		10^6	.2 sec
15 Puzzle	10^{13}	6 days	1Mx	10^{17}	20k yrs
3x3x3 Rubik's	10^{19}	68k yrs	8x	10^{20}	574k yrs
24 Puzzle	10^{25}	12B yrs		10^{37}	10^{23} yrs

Why the difference?
Rubik has higher branch factor
15 puzzle has greater depth

\# of duplicates

Slide adapted from Richard Korf presentation

Costs on Actions



What does BFS do?

… finds the shortest path in terms of number of transitions.
It does ***not*** find the least-cost path.

Best-First Search

- Generalization of breadth-first search
- ***Priority*** queue of nodes to be explored
- Cost function f(n) applied to each node

```
Add initial state to priority queue
While queue not empty
    Node = head(queue)
    If goal?(node) then return node
    Add children of node to queue
```

16

Priority Queue Refresher

- A priority queue is a data structure in which you can insert and retrieve (key, value) pairs with the following operations:

pq.push(key, value)	inserts *(key, value)* into the queue.
pq.pop()	returns the key with the lowest value, and removes it from the queue.

- You can decrease a key's priority by pushing it again
- Unlike a regular queue, insertions aren't constant time, usually O(log *n*)
- We'll need priority queues for cost-sensitive search methods

Old Friends

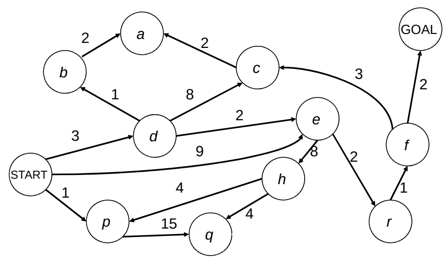
- Breadth First = Best First
  - With f(n) = depth(n)
- Dijkstra's Algorithm (Uniform cost) = Best First
  - With f(n) = the sum of edge costs from start to n

18

Uniform Cost Search

Best first, where

f(n) = "cost from **start** to **n**"

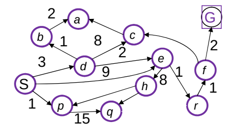


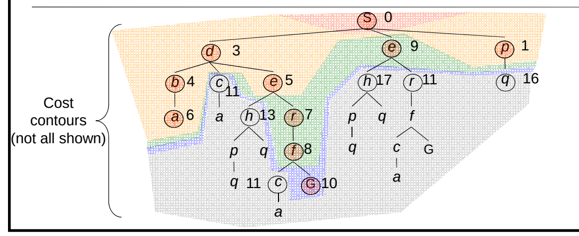
aka "Dijkstra's Algorithm"

Uniform Cost Search

Expansion order:

S, p, d, b, e, a, r, f, e, G

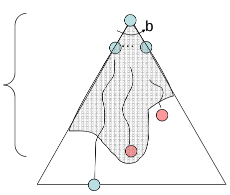




Uniform Cost Search

Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y if finite	N	$O(b^m)$	$O(bm)$
BFS		Y	Y*	$O(b^d)$	$O(b^d)$
UCS		Y*	Y	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$

C^*/ε tiers

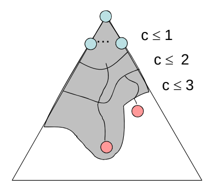


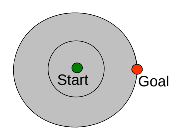
C^* = Optimal cost

ε = Minimum cost of an action

Uniform Cost Issues

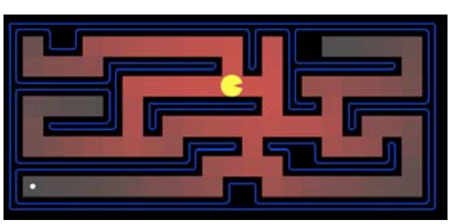
- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location





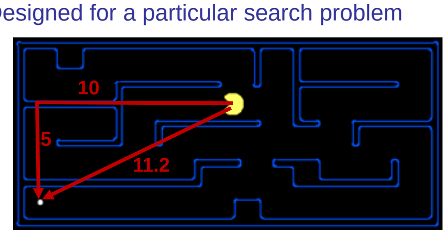
Uniform Cost: Pac-Man

- Cost of 1 for each action
- Explores all of the states, but one



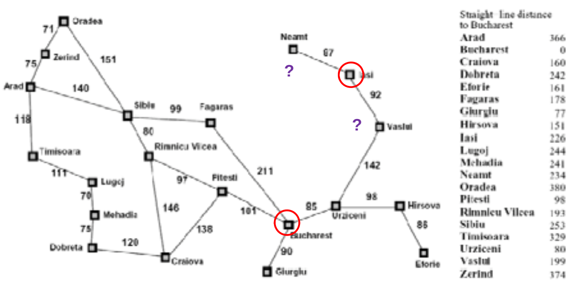
Search Heuristics

- Any ***estimate*** of how close a state is to a goal
- Designed for a particular search problem



- Examples: Manhattan distance, Euclidean distance

Heuristics

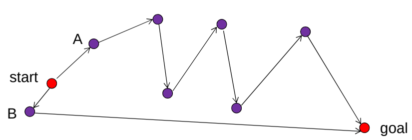


Greedy Search

Best first with f(n) = heuristic estimate of distance to goal

Greedy Search

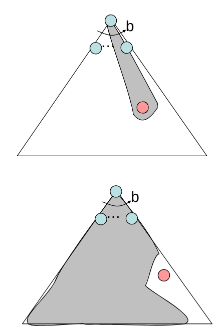
Expand the node that seems closest…



What can go wrong?

Greedy Search

- A common case:
  - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS in the worst case
  - Can explore everything
  - Can get stuck in loops if no cycle checking
- Like DFS in completeness (if finite # states w/ cycle checking)



A* Search

Hart, Nilsson & Rafael 1968

Best first search with f(n) = g(n) + h(n)

- g(n) = sum of costs from start to n
- h(n) = estimate of lowest cost path n → goal

  h(goal) = 0

If h(n) is admissible and monotonic then A* is optimal

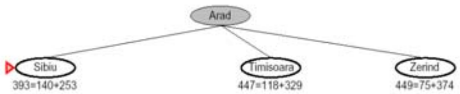
Underestimates cost of reaching goal from node

f values increase from node to descendants (triangle inequality)

A* Example

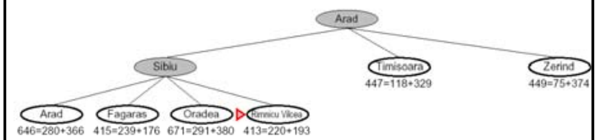
31

A* Example



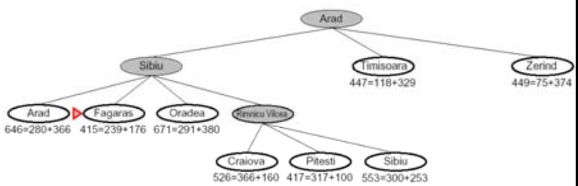
32

A* Example



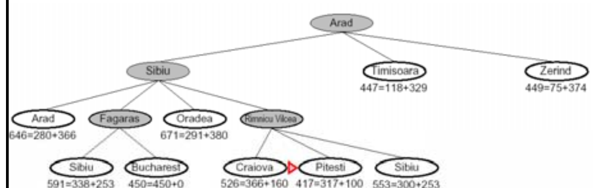
33

A* Example



34

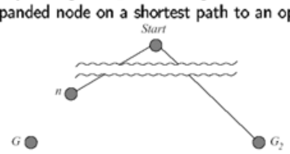
A* Example



35

Optimality of A*

Suppose some suboptimal goal G_2 has been generated and is in the queue.
Let n be an unexpanded node on a shortest path to an optimal goal G_1.



$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible}
\end{aligned}
$$

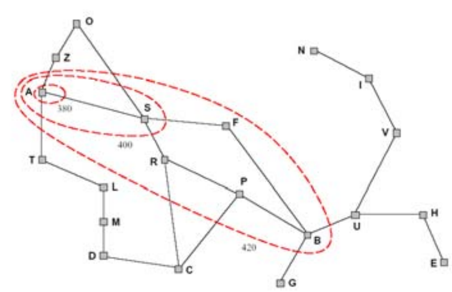
Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

37

Optimality Continued

Lemma: A* expands nodes in order of increasing f value*

Gradually adds "f-contours" of nodes (cf. breadth-first adds layers)
Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$

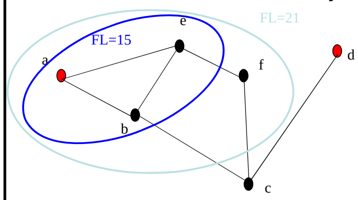


A* Summary

- Pros
- Cons

Iterative-Deepening A*

- Like iterative-deepening depth-first, but...
- Depth bound modified to be an f-limit
  - Start with f-limit = h(start)
  - Prune any node if f(node) > f-limit
  - Next f-limit = min-cost of any node pruned



IDA* Analysis

- Complete & Optimal (ala A*)
- Space usage $\propto$ depth of solution
- Each iteration is DFS - no priority queue!
- # nodes expanded relative to A*
  - Depends on # unique values of heuristic function
  - In 8 puzzle: few values $\Rightarrow$ close to # A* expands
  - In traveling salesman: each f value is unique

    $\Rightarrow$ $1+2+\ldots+n = O(n^2)$ where n=nodes A* expands

    if n is too big for main memory, n^2 is too long to wait!
- Generates duplicate nodes in cyclic graphs

Forgetfulness

- A* used exponential memory
- How much does IDA* use?
  - During a run?
  - In between runs?

© Daniel S. Weld

SMA*

- Use all available memory
- Start like A*
- When memory is full…
  - Erase node with highest f-value
  - First, backup parent with this f-value
  - So… parent knows cost-bound on best child

© Daniel S. Weld

Beam Search

- Idea
  - Best first but only keep N best items on priority queue
- Evaluation
  - Complete?
  - Time Complexity?
  - Space Complexity?

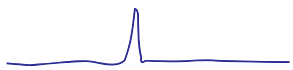
© Daniel S. Weld

Hill Climbing

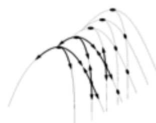
"Gradient ascent"

- Idea
  - Always choose best child; no backtracking
  - Beam search with |queue| = 1
- Problems?
  - Local maxima

  - Plateaus



  - Diagonal ridges



© Daniel S. Weld

Randomizing Hill Climbing

- Randomly disobeying heuristic
- Random restarts

(heavy tailed distributions)

→ Local Search

© Daniel S. Weld

To Do:

- Look at the course website:
  - http://www.cs.washington.edu/cse473/12sp
- Do the readings (Ch 3)
- Start PS1,