Informed Search

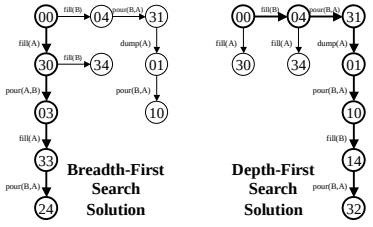
Burr H. Settles
CS-540, UW-Madison
www.cs.wisc.edu/~cs540-1
Summer 2003

Announcements

- If you are not enrolled in the class, and still would like to be, come see me later
  - There's space for up to 35 students in the class
  - That means 6 more seats, from the waiting list
- Don't be scared by Homework #1

Last Time

- Several of you were confused about why we only find 12 of the 20 possible states in yesterday's water jug problem:



Last Time

- Someone asked about why we might want to choose the BFS strategy over IDS
  - They are both complete, optimal, and complexities of $O(b^d)$ for time, but IDS has $O(bd)$ for space
  - It turns out, IDS has $O(2 \times b^d)$ time complexity
  - Recall that in "big-O" notation, we ignore linear factors, so *in the limit* IDS is $O(2 \times b^d) \rightarrow O(b^d)$
- Someone else asked why DFS has $O(bm)$ space complexity, when it could expand more than $b \times m$ states
  - This is an average case
  - Also has to do with open/closed list management

Last Time

- Yet someone else asked what search algorithms have to do with AI… my answer: *AI is all search!!*
- Even *someone else* pointed out that UCS appears to be Dijkstra's Algorithm, a famous example of dynamic programming (DP)… *and it is!!*
  - DP is a family of algorithms that are guaranteed to find optimal solutions for problems
  - They work by breaking the problem up into sub-problems and solving the simplest sub-problems first
  - Other examples of DP are the Viterbi Algorithm, which is used in speech recognition software, and the CYK Algorithm, for finding the most probable "parse tree" for natural language sentences

Last Time

- We talked about building goal-based agents and utility-based agents using search strategies
- But the strategies we discussed were uninformed because the agent is given nothing more than the problem definition
- Today we'll present informed search strategies

Searching in Large Problems

- The search space of a problem is generally described in terms of the number of *possible* states in the search:

Water Jug	12	states
Tic-Tac-Toe	3^9	states
Rubik's Cube	10^{19}	states
100-variable SAT	10^{30}	states
Chess	10^{120}	states

7

Searching in Large Problems

- Some problems' search spaces are too large to search efficiently using uninformed methods
- Sometimes we have additional domain knowledge about the problem that we can use to inform the agent that's searching
- To do this, we use heuristics (informed guesses)
  - Heuristic means "serving to aid discovery"

8

Heuristic Searching

- We define a heuristic function *h(n)*:
  - Is computed from the state at node *n*
  - Uses domain-specific information in some way
- Heuristics can estimate the "goodness" of a particular node (or state) *n*:
  - How close is *n* to a goal node?
  - What might be the minimal cost path from *n* to a goal node?

9

Heuristic Searching

We will formalize a heuristic *h(n)* as follows:

$h(n) \geq 0$ for all nodes *n*

$h(n) = 0$ implies that *n* is a goal node

$h(n) = \infty$ implies that *n* is a "dead end" from which a goal cannot be reached

10

Best-First Search

- Best-first search is a generic informed search strategy that uses an evaluation function *f(n)*, incorporating some kind of domain knowledge
- *f(n)* is used to sort states in the open list using a *priority queue* (like UCS)

11

Greedy Search

- Greedy search is the best-first search strategy, with a simple evaluation function of $f(n) = h(n)$
- It relies only on the heuristic to select what is *currently believed* to be closest to the goal state
- Last time someone asked if UCS was the same as Greedy search… are they?

12

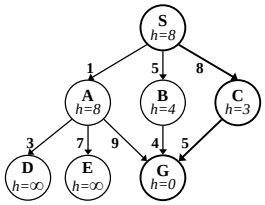
Greedy Search Example

f(n) = h(n)

# of states tested: 3, expanded: 2

State	Open	Closed
--	S:8	--
S	C:3, B:4, A:8	S
C	G:0, B:4, A:8	SC
G	**Goal!!**	

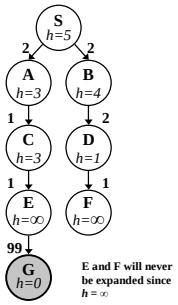
Path: S-C-G
Cost: 13



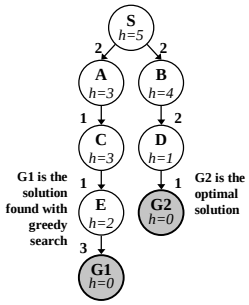
Greedy Search Issues

- Greedy Search is generally *faster* than the uninformed methods
  - It has more knowledge about the problem domain!
- Resembles DFS in that it tends to follow a path that is initially good, and thus:
  - *Not complete* (could chase an infinite path or get caught in cycles if no open/closed lists)
  - *Not optimal* (a better solution could exist through an expensive node)

Greedy Search Issues



E and F will never be expanded since *h* = ∞



G1 is the solution found with greedy search

G2 is the optimal solution

Algorithm A Search

- To try and solve the problems of greedy search, we can conduct an A search by defining our evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ is the minimal cost path from the initial node to the current node
- This adds a UCS-like component to the search:
  - $g(n)$ is the cost to reach n
  - $h(n)$ is the estimated cost from n to a goal

Algorithm A Search Issues

- A Search is an informed strategy that incorporates both the *real costs* and the *heuristic* functions to find better solutions
- However, if our heuristic makes certain errors (*e.g.* estimating ∞ along the path to a goal):
  - *Still not complete*
  - *Still not optimal*

Admissible Heuristics

- Heuristic functions are good for helping us find good solutions quickly
- But it's hard to design accurate heuristics!
  - They can be expensive to compute
  - They can make errors estimating the costs
- These problems keep informed searches like the A search from being complete and optimal

Admissible Heuristics

- There is hope! We can add a constraint on our heuristic function that, for all nodes *n* in the search space, $h(n) \leq h^*(n)$
  - Where $h^*(n)$ is the true minimal cost from *n* to a goal
- When $h(n) \leq h^*(n)$, we say that $h(n)$ is admissible
- ✱ *Admissible heuristics are inherently optimistic, (i.e. they never overestimate the cost to a goal)*

19

A* Search

- When we conduct an A search using an admissible heuristic, it is called A* search
- A* search is complete if
  - The branching factor is finite
  - Every action has a fixed cost
- A* search is optimal

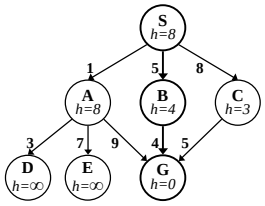
20

A* Search Example

f(n) = g(n) + h(n)

# of states tested: 4, expanded: 3

State	Open	Closed
--	S:8	--
S	A:9, B:9, C:11	S
A	B:9, G:10, C:11, D:∞, E:∞	SA
B	G:9, C:11, D:∞, E:∞	SAB
G	**Goal!!**	



Path: S-B-G
Cost: 9

21

Proof of A* Optimality

- Let:
  - *G1* be the optimal goal
  - *G2* be some other goal
  - *f** be the cost of the optimal path (to *G1*)
  - *n* be some node in the optimal path, but not to *G2*
- Assume that *G2* is found using A* search where $f(n) = g(n) + h(n)$, and $h(n)$ is admissible
  - *i.e.* A* finds a sub-optimal path, which is shouldn't

22

Proof of A* Optimality

- $g(G2) > f^*$
  by definition, *G2* is sub-optimal
- $f(n) \leq f^*$
  by admissibility: since $f(n)$ never overestimates the cost to the goal it must be ≤ the cost of the optimal path
- $f(G2) \leq f(n)$
  G2 must be chosen over *n*, by our assumption
- $f(G2) \leq f^*$
  by transitivity of the ≤ operator

23

Proof of A* Optimality

- $f(G2) \leq f^*$ (from previous slide)
- $g(G2) + h(G2) \leq f^*$
  by substituting the definition of $f(n)$
- $g(G2) \leq f^*$
  since *G2* is a goal node, $h(G2) = 0$
- This *contradicts* the assumption that *G2* is sub-optimal ($g(G2) > f^*$), thus A* is optimal in terms of path cost

✱ *A* never finds a sub-optimal goal*

24

Devising Heuristics

- Often done by relaxing the problem
  - See *AI: A Modern Approach* for more details
- The goal of admissible heuristics is to get as close as possible to the actual cost *without going over*
- Trade-off:
  - A really good *h(n)* might be expensive to compute
  - Could we find the solution faster with a simpler one?

25

Devising Heuristics

- If $h(n) = h^*(n)$ for all n:
  - Only nodes on optimal solution are searched
  - No unnecessary work
  - We know the actual cost from n to goal
- If $h(n) = 0$ for all n:
  - Heuristic is still admissible
  - A* is identical to UCS
- The closer h is to h^*, the fewer nodes will need to be expanded; the more accurate the search will be

26

Devising Heuristics

- If $h1(n) \leq h2(n) \leq h^*(n)$ for each non-goal node n:
  - We say *h2* dominates *h1*
  - *h2* is closer to actual cost, but is still admissible
  - A* using *h1* (*i.e.* A1*) expands at least as many, if not not more nodes than A2*
  - A2* is said to be better informed than A1*

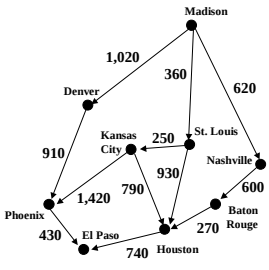
27

Devising Heuristics

Let's revisit our Madison-to-El-Paso example from yesterday, but instead of optimizing dollar cost, we're trying to reduce travel distance

What's a good heuristic to use for this problem?

Is it admissible?



28

Optimal Searching

- Optimality isn't always required (*i.e.* you want *some* solution, not the *best* solution)
  - *h(n)* need not be admissible (necessarily)
  - Greedy search will often suffice
  - This is all problem-dependent, of course
- Can result in fewer nodes being expanded, and a solution can be found faster

29

Partial Searching

- So far we've discussed algorithms that try to find a *path* from the initial state to a goal state
- These are called partial search strategies because they build up partial solutions, which could enumerate the *entire* search space to find solutions
  - This is OK for small "toy world" problems
  - This is not OK for NP-Complete problems or those with exponential search spaces

30

Next Time

- We will discuss complete search strategies, in which each state/node represents a complete, possible solution to the problem
- We will also discuss optimization search, where we try to find such complete solutions that are optimal (the best)

31