Informed Search Strategies

Artificial Intelligence Programming in Prolog

Lecturer: Tim Smith

Lecture 9

21/10/04

Blind Search

- Depth-first search and breadth-first search are examples of *blind* (or uninformed) search strategies.
- Breadth-first search produces an optimal solution (eventually, and if one exists), but it still searches blindly through the state-space.
- Neither uses any knowledge about the specific domain in question to search through the state-space in a more directed manner.
- If the search space is big, blind search can simply take too long to be practical, or can significantly limit how deep we're able to look into the space.

Informed Search

- A search strategy which searches the most promising branches of the state-space first can:
  - find a solution more quickly,
  - find solutions even when there is limited time available,
  - often find a *better* solution, since more profitable parts of the state-space can be examined, while ignoring the unprofitable parts.
- A search strategy which is better than another at identifying the most promising branches of a search-space is said to be more *informed*.

Best-first search

- To implement an *informed* search strategy, we need to slightly modify the skeleton for agenda-based search that we've already seen.
- Again, the crucial part of the skeleton is where we update the agenda.
- Rather than simply adding the new agenda items to the beginning (depth-first) or end (breadth-first) of the existing agenda, *we add them to the existing agenda in order* according to some measure of how promising we think a state is, with the most promising ones first. This gives us *best-first search*.

```
update_agenda(OldAgenda, NewStates, NewAgenda) :-
    ~~append(NewStates, OldAgenda, NewAgenda).~~
     sort_agenda(NewStates, OldAgenda, NewAgenda).
```

Best-first search (2)

```
sort_agenda([], NewAgenda, NewAgenda).
sort_agenda([State|NewStates], OldAgenda, SortedAgenda):-
     insert(State, OldAgenda, NewAgenda),
     sort_agenda(NewStates, NewAgenda, SortedAgenda).

insert(New, [], [New]).
insert(New, [Old|Agenda], [New,Old|Agenda]):-
     h(New,H), h(Old,H2), H =< H2.
insert(New, [Old|Agenda], [Old|Rest]):-
     insert(New, Agenda, Rest).
```

← Compares heuristic evaluation for each state.

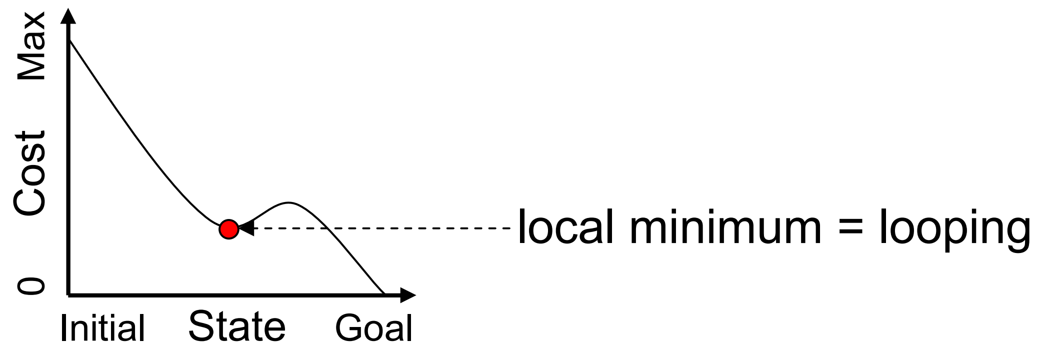
- This is a very general skeleton. By implementing **sort_agenda/3**, according to whatever domain we're looking at, we can make the search strategy *informed* by our knowledge of the domain.
- Best-first search isn't so much a search strategy, as a mechanism for implementing many *different* types of informed search.

Uniform-cost search

- One simple way to sort the agenda is by the *cost-so-far*. This might be the number of moves we've made so far in a game, or the distance we've travelled so far looking for a route between towns.
- If we sort the agenda so that the states with the *lowest* costs come first, then we'll always expand these first, and that means that we're sure we'll always find an *optimal* solution first.
- This is *uniform-cost search*. It looks a lot like breadth-first search, except that it will find an optimal solution even if the steps between states have different costs (e.g. the distance between towns is irregular).
- However, uniform-cost search doesn't really direct us towards the goal we're looking for, so it isn't very informed.

Greedy Search

- Alternatively, we might sort the agenda by the *cost of getting to the goal from that state*. This is known as greedy search.
- An obvious problem with greedy search is that it doesn't take account of the cost so far, so it isn't optimal, and can wander into dead-ends, like depth-first search.



- In most domains, we also don't *know* the cost of getting to the goal from a state. So we have to guess, using a *heuristic evaluation function.*
  - If we knew how far we were from the goal state we wouldn't need to search for it!

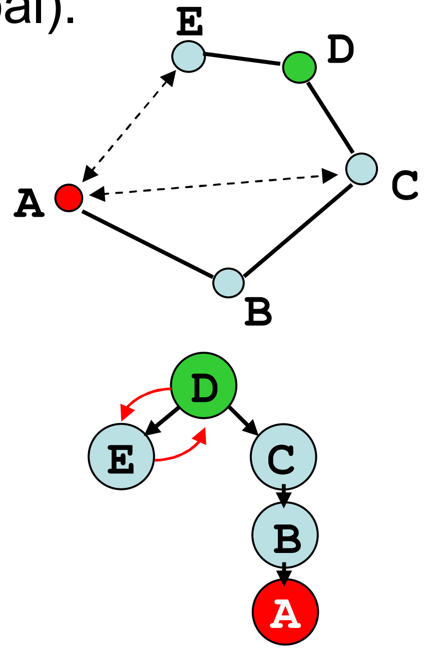
Heuristic evaluation functions

- A *heuristic evaluation* function, **`h(n)`**, is the estimated cost of the cheapest path from the state at node **`n`**, to a goal state.
- Heuristic evaluation functions are very much dependent on the domain used. **`h(n)`** might be the estimated number of moves needed to complete a puzzle, or the estimated straight-line distance to some town in a route finder.
- Choosing an appropriate function greatly affects the effectiveness of the state-space search, since it tells us which parts of the state-space to search next.
- A heuristic evaluation function which accurately represents the *actual cost* of getting to a goal state, tells us very clearly which nodes in the state-space to expand next, and leads us quickly to the goal state.

Example Heuristics

Straight-line distance

- The distance between two locations on a map can be known without knowing how they are linked by roads (i.e. the absolute path to the goal).

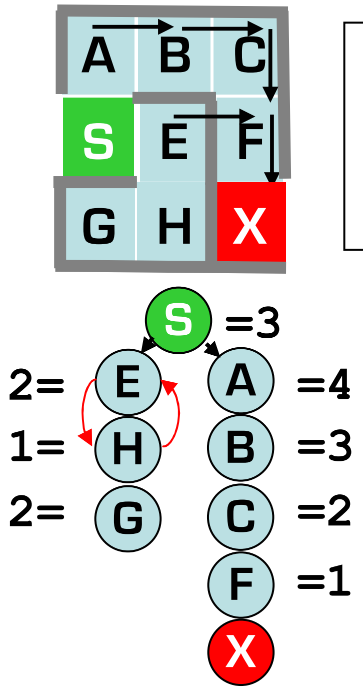


Problem Space

Search Tree

Manhattan Distance

- The smallest number of vertical and horizontal moves needed to get to the goal (ignoring obstacles).



Manhattan Distance
A = 4
E = 2

Combining cost-so-far and heuristic function

- We can combine the strengths of uniform-cost search and greedy search.
- Since what we're really looking for is the optimal path between the initial state, and some goal state, a better measure of how promising a state is, is the *sum* of the cost-so-far, and our best estimate of the cost from there to the nearest goal state.
- For a state **`n`**, with a cost-so-far **`g(n)`**, and a heuristic estimate of the cost to goal of **`h(n)`**, what we want is:

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

- This proves to be a very effective strategy for controlling state-space search. When used with best-first search, as a way of sorting the agenda---where the agenda is sorted so that the states with the *lowest* values of **`f(n)`** come first, and are therefore expanded first---this is known as *Algorithm A*.

A* search and admissibility

- The choice of an appropriate heuristic evaluation function, **`h(n)`**, is still crucial to the behaviour of this algorithm.
- In general, we want to choose a heuristic evaluation function **`h(n)`** which is as close as possible to the *actual* cost of getting to a goal state.
- If we can choose a function **`h(n)`** which never *overestimates* the actual cost of getting to the goal state, then we have a very useful property. Such a **`h(n)`** is said to be *admissible*.
- Best-first search, where the agenda is sorted according to the function **`f(n) = g(n) + h(n)`** and where the function **`h(n)`** is admissible, can be proven to always find an optimal solution. This is known as *Algorithm A*.*

BFS and Admissibility

- Perhaps surprisingly, breadth-first search (where each step has the same cost) is an example of Algorithm A*, since the function it uses to sort the agenda is simply:

```
f(n) = g(n) + 0
```

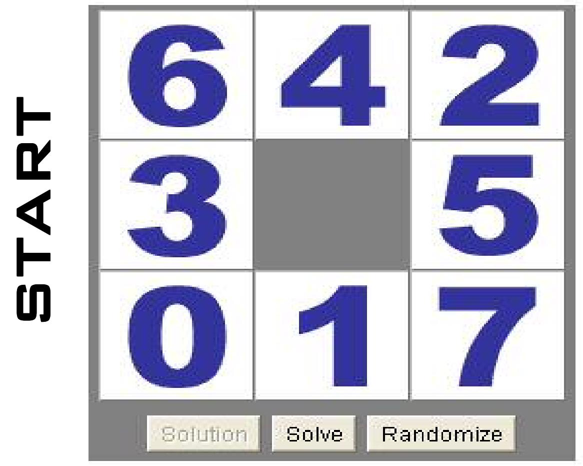
- Breadth-first search takes no account of the distance to the goal, and because a zero estimate cannot possibly be an overestimate of that distance it has to be admissible. This means that BFS can be seen as a basic example of Algorithm A*.

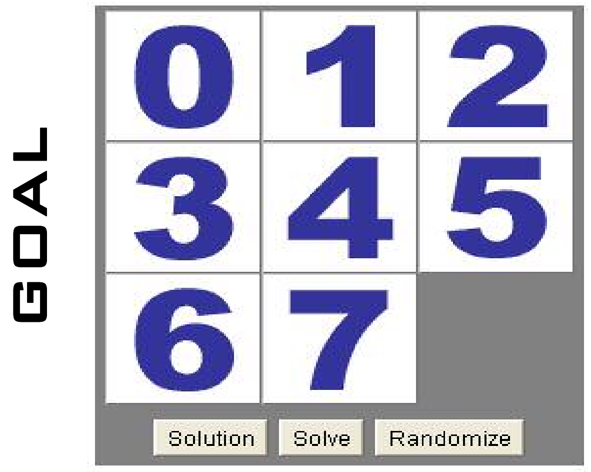
- However, despite being *admissible* breadth-first search isn't a very intelligent search strategy as it doesn't direct the search towards the goal state. The search is still blind.

Informedness

- We say that a search strategy which searches less of the state-space in order to find a goal state is more *informed*. Ideally, we'd like a search strategy which is both admissible (so it will find us an optimal path to the goal state), and informed (so it will find the optimal path *quickly*.)
- Admissibility requires that the heuristic evaluation function, **`h(n)`** doesn't overestimate, but we *do* want a function which is as close as possible to the actual cost of getting to the goal.
- Formally, for two admissible heuristics **`h1`** and **`h2`**, if **`h1(n) <= h2(n)`** for all states **`n`** in the state-space, then heuristic **`h2`** is said to be more *informed* than **`h1`**.

Example: the 8-puzzle





- http://www.permadi.com/java/puzzle8/
- This is a classic Toy Problem (a simple problem used to compare different problem solving techniques).
- The puzzle starts with 8 sliding-tiles out of place and one gap into which the tiles can be slid. The *goal* is to have all of the numbers in order.
- What would be a good, admissible, informed heuristic evaluation function for this domain?

8-puzzle: heuristics

- We could use the number of tiles out of place as our heuristic evaluation function. That would give **`h1(n) = 6`** for this puzzle state.
- We could also use the sum of the distances of the tiles from their goal positions i.e. the Manhattan distance. This would give **`h2(n) = 8`**.
- In fact, it can easily be shown that **`h2`** is both admissible and more informed than **`h1`**.
  - It cannot overestimate, since the number of moves we need to make to get to the goal state must be at least the sum of the distances of the tiles from their goal positions.
  - It always gives a value at least as high as **`h1`**, since if a tile is out of position, by definition it is at least one square away from its goal position, and often more.

Comparing Search Costs

- If we compare the search costs for different search strategies used to solve the 8-puzzle.
- We can calculate search costs as the number of nodes in the state-space looked at to reach a solution.

Solution at depth	*Iterative Deepening Search*	A* (`h1)` = Num. tiles out.	A* (`h2)` = Manhat. Dist.
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	364404	227	73
14	3473941	539	113

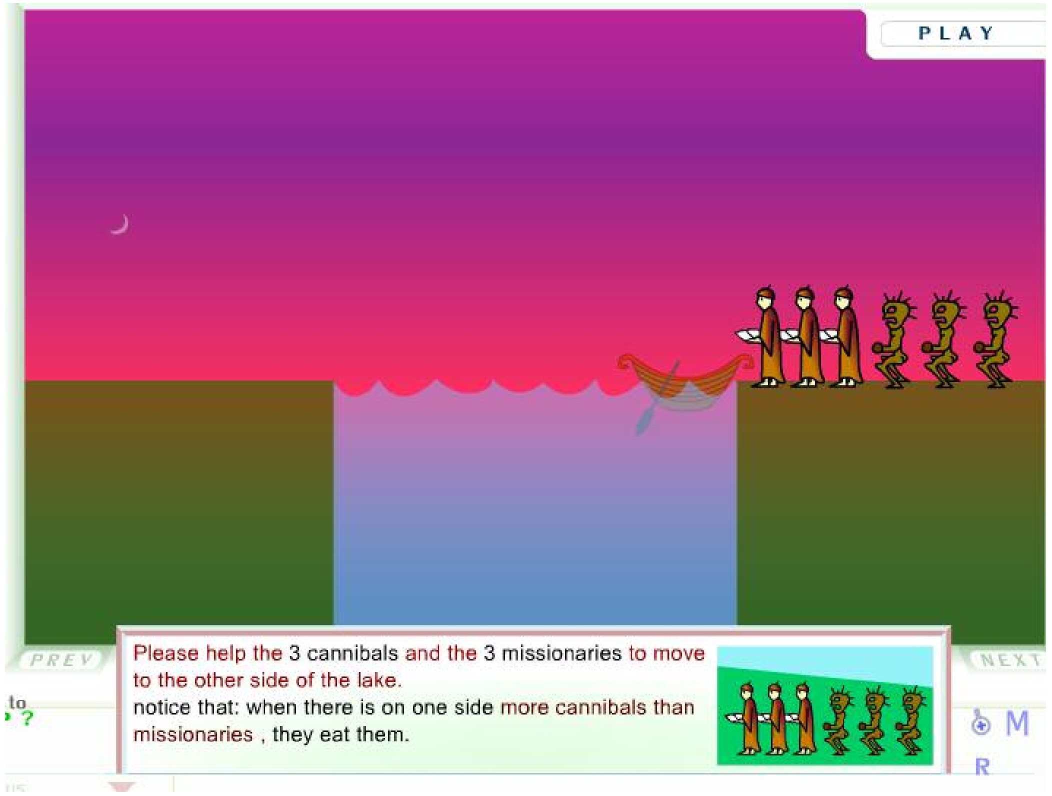
- h2 dominates h1 = for any node, n, h2(n) >= h1(n).
- It is always better to use a heuristic function with higher values, as long as it does not overestimate (i.e. it is admissible).

Summary

- *Blind search*: Depth-First, Breadth-First, IDS
  - Do not use knowledge of problem space to find solution.
- *Informed search*
- *Best-first search*: Order agenda based on some measure of how 'good' each state is.
- *Uniform-cost:* Cost of getting to current state from initial state = **`g(n)`**
- *Greedy search:* Estimated cost of reaching goal from current state
  - *Heuristic evaluation functions,* **`h(n)`**
- *A* search:* **`f(n) = g(n) + h(n)`**
- Admissibility: **`h(n)`** never *overestimates* the actual cost of getting to the goal state.
- Informedness: A search strategy which searches less of the state-space in order to find a goal state is more *informed.*

Missionaries and Cannibals

- http://www.plastelina.net/examples/games/game2.html
- Next weeks practical exercise: complete a agenda-based search program to solve the Missionaries and Cannibals problem.



Please help the 3 cannibals and the 3 missionaries to move to the other side of the lake.
notice that: when there is on one side more cannibals than missionaries , they eat them.