Solving problems by search II

We now look at how an agent might achieve its goals using more sophisticated search techniques.

Aims:

- to introduce the concept of a *heuristic* in the context of search problems;
- to introduce some further algorithms for conducting the necessary search for a sequence of actions, which are able to make use of a heuristic.

Reading: Russell and Norvig, chapter 4.

Problem solving by informed search

Basic search methods make limited use of any *problem-specific knowledge* we might have.

- Use of the available knowledge is limited to the *formulation* of the problem as a search problem.
- We have already seen the concept of *path cost* $g(n)$

  $g(n) =$ cost of any path (sequence of actions) in a state space

- We can now introduce an *evaluation function*. This is a function that attempts to measure the *desirability of each node*.

The evaluation function will clearly not be perfect. (If it is, there is no need to search!)

Best-first search and greedy search

Best-first search simply expands nodes using the ordering given by the evaluation function.

- We could just use path cost, but this is misguided as path cost is not in general *directed* in any sense *toward the goal*.
- A *heuristic function*, usually denoted $h(n)$ is one that *estimates* the cost of the best path from any node n to a goal.
- If n is a goal then $h(n) = 0$.

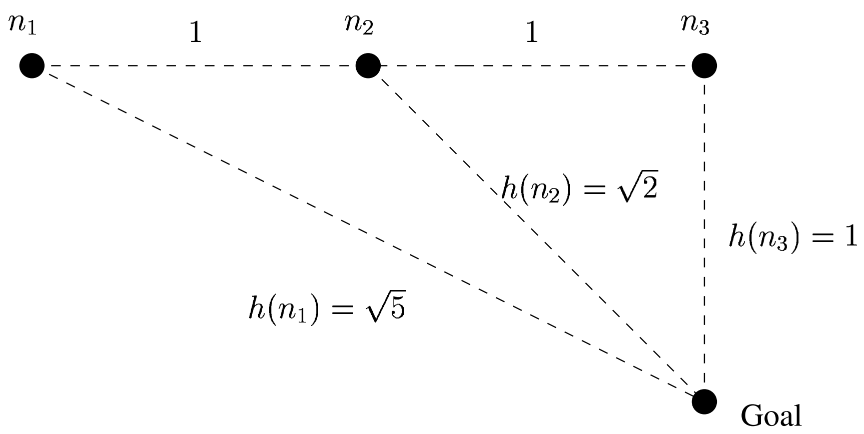
Using a heuristic function along with best-first search gives us the *greedy search* algorithm.

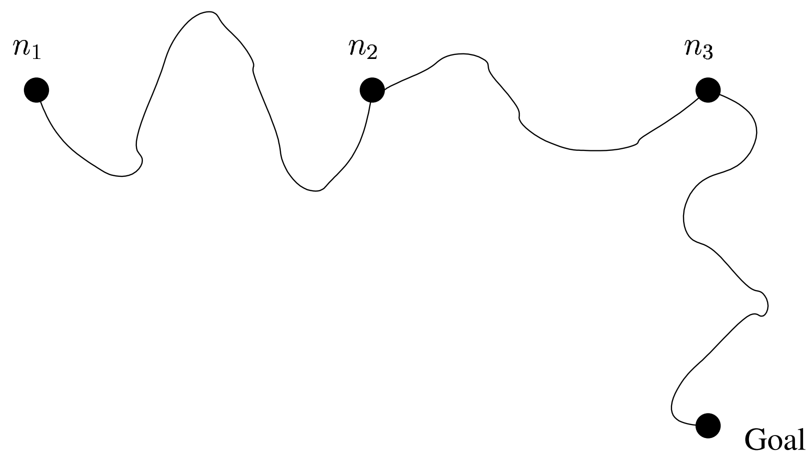
Example: route-finding

A reasonable heuristic function here is

$$h(n) = \text{straight line distance from } n \text{ to the nearest goal}$$

Example:





Example: route-finding

Greedy search suffers from some problems:

- its time complexity is $O(\text{branching}^{\text{depth}})$;
- it is not optimal or complete;
- its space-complexity is $O(\text{branching}^{\text{depth}})$.

BUT: greedy search is often very effective, provided we have a good $h(n)$.

$A^\star$ search

$A^\star$ *search* combines the good points of:

- greedy search—by making use of $h(n)$;
- uniform-cost search—by being optimal and complete.

It does this in a very simple manner: it uses path cost $g(n)$ and also the heuristic function $h(n)$ by forming

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

where

$$g(n) = \text{cost of path } \textit{to } n$$

and

$$h(n) = \text{estimated cost of best path } \textit{from } n$$

So: $f(n)$ is the estimated cost of a path *through* n.

$A^\star$ search

$A^\star$ search:

- a best-first search using $f(n)$;
- it is both complete and optimal...
- ...provided that h is an *admissible heuristic*.

Definition: an admissible heuristic $h(n)$ is one that *never overestimates* the cost of the best path from n to a goal.

Monotonicity

Assume h is admissible. Remember that $f(n) = g(n) + h(n)$ so if n' follows n

$$g(n') \geq g(n)$$

and we expect that

$$h(n') \leq h(n)$$

although this does not have to be the case. The possibility remains that $f(n')$ might be *less* than $f(n)$.

- if it is always the case that $f(n') \geq f(n)$ then $h(n)$ is called *monotonic*;
- $h(n)$ is monotonic if and only if it obeys the triangle inequality.

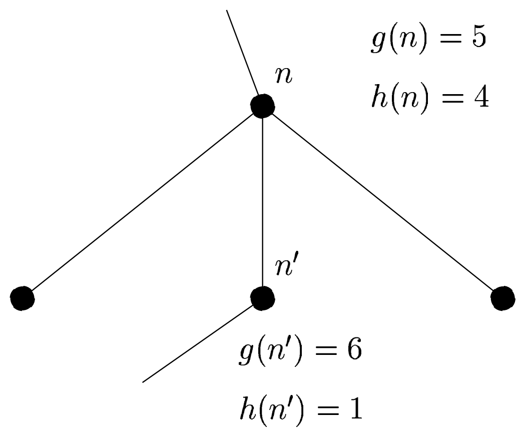
If $h(n)$ is *not* monotonic we can make a simple alteration and use

$$f(n') = \max\{f(n), g(n') + h(n')\}$$

This is called the *pathmax* equation.

The pathmax equation

Why does the pathmax equation make sense?



So here $f(n) = 9$ and $f(n') = 7$.

The fact that $f(n) = 9$ tells us the cost of a path through n is *at least* 9 (because $h(n)$ is admissible).

But n' is *on a path through* n. So to say that $f(n') = 7$ makes no sense.

$A^\star$ search is optimal

To see that $A^\star$ search is optimal we reason as follows.

Let Goal_{opt} be an optimal goal state with

$$f(\text{Goal}_{\text{opt}}) = g(\text{Goal}_{\text{opt}}) = f_{\text{opt}}$$

Let Goal_2 be a suboptimal goal state with

$$f(\text{Goal}_2) = g(\text{Goal}_2) = f_2 > f_{\text{opt}}$$

We need to demonstrate that the search can never select Goal_2.

$A^\star$ search is optimal

Let n be a leaf node on an optimal path to Goal_{opt}. So

$$f_{\text{opt}} \geq f(n)$$

because h is admissible and we're assuming it's also monotonic.

Now say Goal_2 is chosen for expansion *before* n. This means that

$$f(n) \geq f_2$$

so we've established that

$$f_{\text{opt}} \geq f_2 = g(\text{Goal}_2).$$

But this means that Goal_{opt} is not optimal! A contradiction.

$A^\star$ search is complete

$A^\star$ search is complete provided:

1. the graph has finite branching factor;
2. there is a finite, positive constant c such that each operator has cost at least c.

Why is this?

$A^{\star}$ search is complete

The search expands nodes according to increasing $f(n)$. So: the only way it can fail to find a goal is if there are infinitely many nodes with $f(n) < f(\text{Goal})$.

There are two ways this can happen:

1. there is a node with an infinite number of descendants;
2. there is a path with an infinite number of nodes but a finite path cost.

Complexity

- $A^\star$ search has a further desirable property: it is *optimally efficient*.
- This means that no other optimal algorithm that works by constructing paths from the root can guarantee to examine fewer nodes.
- BUT: despite its good properties we're not done yet!
- $A^\star$ search unfortunately still has exponential time complexity in most cases unless $h(n)$ satisfies a very stringent condition that is generally unrealistic:

  $$|h(n) - h'(n)| \leq O(\log h'(n))$$

  where $h'(n)$ denotes the *real* cost from n to the goal.
- As $A^\star$ search also stores all the nodes it generates, once again it is generally memory that becomes a problem before time.

IDA* - iterative deepening $A^\star$ search

Iterative deepening search used depth-first search with a limit on depth that gradually increased.

- IDA* does the same thing *with a limit on f cost*.
- It is complete and optimal under the same conditions as $A^\star$.
- It only requires space proportional to the longest path.
- The time taken depends on the number of values h can take.

If h takes enough values to be problematic we can increase f by a fixed ϵ at each stage, guaranteeing a solution at most ϵ worse than the optimum.

IDA* - iterative deepening $A^\star$ search

```
Action_sequence ida()
{
   float f_limit = f(root);
   Node root = root node for problem;

   while(true)
   {
      (sequence,f_limit) = contour(root,f_limit);
      if (sequence != empty_sequence)
        return sequence;
      if (f_limit == infinity)
        return empty_sequence;
   }
}
```

IDA* - iterative deepening $A^\star$ search

```
(Action_sequence,float) contour(Node node, float f_limit)
{
   float next_f = infinity;
   if (f(node) > f_limit)
      return (empty_sequence,f(node));
   if (goaltest(node))
      return (node,f_limit);
   for (each successor s of node)
   {
      (sequence,new_f) = contour(s,f_limit);
      if (sequence != empty_sequence)
         return (sequence,f_limit);
      next_f = minimum(next_f,new_f);
   }
   return (empty_sequence,next_f);
}
```