Heuristic Search

- **Idea:** don’t ignore the goal when selecting paths.
- Often there is extra knowledge that can be used to guide the search: **heuristics.**
- \( h(n) \) is an estimate of the cost of the shortest path from node \( n \) to a goal node.
- \( h(n) \) uses only readily obtainable information (that is easy to compute) about a node.
- \( h \) can be extended to paths: \( h(\langle n_0, \ldots, n_k \rangle) = h(n_k) \).
- \( h(n) \) is an underestimate if there is no path from \( n \) to a goal that has path length less than \( h(n) \).
Example Heuristic Functions

- If the nodes are points on a Euclidean plane and the cost is the distance, we can use the straight-line distance from \( n \) to the closest goal as the value of \( h(n) \).

- If the graph is one of queries for a derivation from a KB, one heuristic function is the number of atoms in the query.

- If the nodes are locations and cost is time, we can use the distance to a goal divided by the maximum speed.
Best-first Search

Idea:
select the path whose end is closest to a goal
according to the heuristic function.

Best-first search selects a path on the frontier with
minimal $h$-value.

It treats the frontier as a priority queue ordered by $h$. 
Illustrative Graph — Best-first Search

© David Poole, Alan Mackworth, Randy Goebel, and Oxford University Press 1998-2002
Complexity of Best-first Search

- It uses space exponential in path length.
- It isn’t guaranteed to find a solution, even of one exists.
- It doesn’t always find the shortest path.
Heuristic Depth-first Search

- It’s a way to use heuristic knowledge in depth-first search.

- **Idea:** order the neighbors of a node (by $h$) before adding them to the front of the frontier.

- It locally selects which subtree to develop, but still does depth-first search. It explores all paths from the node at the head of the frontier before exploring paths from the next node.

- Space is linear in path length. It isn’t guaranteed to find a solution. It can get led up the garden path.
**A* Search**

- A* search uses both path cost and heuristic values.

- $\text{cost}(p)$ is the cost of the path $p$.

- $h(p)$ estimates of the cost from the end of $p$ to a goal.

- Let $f(p) = \text{cost}(p) + h(p)$. $f(p)$ estimates of the total path cost of going from a start node to a goal via $p$. 

$$
\begin{align*}
\text{start} & \rightarrow_p n \rightarrow \text{goal} \\
\text{cost}(p) & \quad h(n) \\
\hline
f(p) & 
\end{align*}
$$
\underline{A* Search Algorithm}

- A* is a mix of lowest-cost-first and best-first search.
- It treats the frontier as a priority queue ordered by $f(n)$.
- It always selects the node on the frontier with the lowest estimated distance from the start to a goal node constrained to go via that node.
Admissibility of $A^*$

If there is a solution, $A^*$ always finds an optimal solution — the first path to a goal selected — if

- the branching factor is finite
- arc costs are bounded above zero (there is some $\epsilon > 0$ such that all of the arc costs are greater than $\epsilon$), and
- $h(n)$ is an underestimate of the length of the shortest path from $n$ to a goal node.
Why is $A^*$ admissible?

- If a path $p$ to a goal is selected from a frontier, can there be a shorter path to a goal?

- Suppose path $p'$ is on the frontier. Because $p$ was chosen before $p'$, and $h(p) = 0$:

  
  \[
  \text{cost}(p) \leq \text{cost}(p') + h(p').
  \]

- Because $h$ is an underestimate

  \[
  \text{cost}(p') + h(p') \leq \text{cost}(p'')
  \]

  for any path $p''$ to a goal that extends $p'$

- So $\text{cost}(p) \leq \text{cost}(p'')$ for any other path $p''$ to a goal.
Why is A* admissible?

- There is always an element of an optimal solution path on the frontier before a goal has been selected. This is because, in the abstract search algorithm, there is the initial part of every path to a goal.

- A* halts, as the minimum g-value on the frontier keeps increasing, and will eventually exceed any finite number.