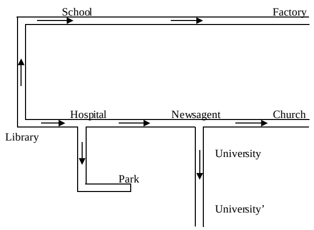
Best First Search

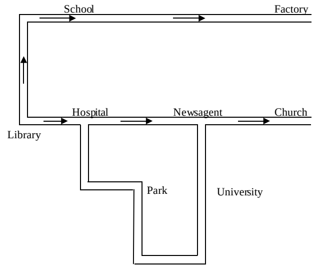
- Greedy Search
  - expand the “best” successor node
  - minimise cost estimate to nearest goal: *h(n)*
    - heuristic function - problem specific
    - prefers biggest local bite regardless of long term effect - hence the name!

1. Start with *agenda* = [initial state]
2. While *agenda* not empty:
   (a) remove the best node N from *agenda*
   (b) if it is the goal then return success
   else find its successors
   (c) assign successor nodes a score using evaluation
   function and add scored nodes to *agenda*

Greedy Search cont’t

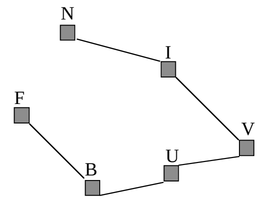


Greedy Search cont’d



- Evaluation
  - Non-optimal finds high cost paths
  - Incomplete - gets stuck oscillating
  - Complexity - time and space exponential => b^m

Greedy Search Again



Heuristic Functions:
h(N) = 100, *h*(I) = 150, *h*(V) = 200
Expanding Iasi gives:
(N:100, V:200) <- priority queue
Expanding Neamt (N:100) gives:
(I:150, V:200)
Expanding Iasi again (I:100) gives:
(N:100, V:200, V:200)
etc.

Beam Search

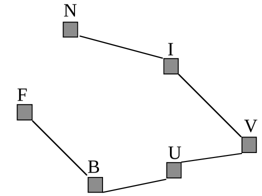
- Based on Breadth first
  - Expands best *w* nodes at each level (others ignored)
  - Minimises cost estimate *h(n)*
  - Only *wb* nodes explored at any depth
  - Only *wd* nodes stored.

1. Start with *queue* = [initial state] and *found* = FALSE
2. While *queue* not empty and not *found* do:
   (a) loop *w* times
      (a1) remove first node N from *queue*
      (a2) if N is goal state then *found* = TRUE
      (a3) find all successor nodes of N and add them to the *queue*
   (b) assign successor nodes a score and prioritise accordingly.
   (c) remove last *(l - w)* nodes from *queue*

Beam Search (2)

- Evaluation
  - Non-optimal
    - May still find high cost paths
      - But less likely to
  - Incomplete -
    - May still get stuck oscillating
      - But less likely to
  - Complexity
    - time and space polynomial
      - wb and wd

Beam Search Again



Heuristic Functions:

h(N) = 100, *h*(I) = 150, *h*(V) = 200

Expanding Iasi gives:

(N:100, V:200) <- priority queue

Expanding Neamt and Vaslui gives:

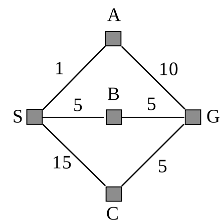
(~~I:150~~, U:190, I:150)

Expanding again gives:

(N:100, ~~V:200~~, B:178, ~~V:200~~)

etc.

Uniform Cost Search



Expanding the start gives:

(A:1 B:5 C:15)

Expanding A:1 gives:

(B:5 G:11 C:15)

Expanding B:5 gives:

(G:10 G:11 C:15)

A* Search

- Synthesis
  - Uniform Search minimizes *g(n)*
    - optimal, complete, but inefficient
  - Greedy search minimizes *h(n)*
    - non-optimal and incomplete
  - Combine and minimize *f(n)*
    - ***f(n) = g(n) + h(n)***
- Restrictions:
  - *h(n)* must never overestimate the cost to reach a goal
    - Admissable heuristic - optimistic
  - *f(n)* must be monotonic

A* Search 2

- PATHMAX
  - maintains monotonicity
    - if *n'* is a child of *n* and *f(n') < f(n)* then:

      f(n') = max(f(n), g(n') + f(n'))
- Proof of A* optimality
  - Russel and Norvig page 99
- Proof of A* completeness
  - Russel and Norvig page 100
- A* complexity
  - exponential in the solution length
    - subexponential condition|:

      $|h(n) - h^*(n)| =< O(\log h^*(n))$
    - *memory will run out first!!!*

Heuristic Functions

- The Eight Puzzle

5	4	
6	1	8
7	3	2

1	2	3
8		4
7	6	5

- Possible heuristics
  - No of tiles in wrong position: *h1*
  - Sum of distances from goal: *h2*
    - **City block** or **Manhatten Distance**

Accuracy & Performance

- Effective Branching Factor
  - uniform tree equivalent with N nodes if A* expands N nodes
  - Domination:
    - if one heuristic function has a higher value than another.

It is always better to use heuristic functions with higher values as long as they do not overestimate!

Accuracy & Performance

	Search Cost			Effective Branching Factor		
d	IDS	A*(h1)	A*(h2)	IDS	A*(h1)	A*(h2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	-	1301	211	-	1.45	1.25
18	-	3056	363	-	1.46	1.26
20	-	7276	676	-	1.47	1.27
22	-	18094	1219	-	1.48	1.28
24	-	39135	1641	-	1.48	1.26

Memory Bounded Search

- Memory is usually the first thing to give!
- IDA*
  - extension of ID search to use heuristic information
- SMA*
  - restricts A* agenda size to fit available memory.

Inventing Heuristics

- Relaxed problem
  - less restrictions on operators
  - exact sol'n to RP => good heuristic
- Best heuristic?
  - not always clear
    - $h(n) = max\{h_1(n) \ldots . h_m(n)\}$
- Statistical Information
  - random tests to gather statistics
  - can lose admissibility
- Pick out "Features"
  - relevant info for heuristics from present state
- Machine Learning

Good Heuristic Functions must be efficient as well as accurate

IDA* Search

- Depth First Iterations
  - f-cost replaces depth limit
  - complete search inside f-contour
  - next contour min of successors f-cost
- Space complexity
  - polynomial (aprox. *bd*)
- Time complexity
  - no priority queue saves time
  - proportional to heuristic values
  - ε–Admissible
    - fixed step increase for each iteration
    - sub-optimal by at most ε

SMA* Search

- Problems with IDA*
  - uses too little memory
  - only keeps current f-cost
  - forgets history -> repeats it
- Simplified Memory-bounded A*
  - uses all available memory for search
  - leads to improved search efficiency
- Other Properties
  - Avoids repeated states
    - as far as memory allows
  - Complete
    - if memory stores shallowest sol'n path
  - Optimal
    - if shallowest optimal path can be stored
  - Optimally Efficient
    - if memory can store entire search tree

SMA* Search cont'd

- Forgotten Nodes
  - nodes dropped if high f-cost
    - ancestor retains info of best cost in forgotten sub-tree
    - onle reconstructed if all other paths look worse.

Heuristics for CSP

- Most constrained variable heuristic
  - variable with *fewest* possible values chosen
- Most constraining variable heuristic
  - variable involved in largest no. of constraints on other variables chosen.
- Least constraining value heuristic
  - choose a value that rules out the least no. of values in variables connected to the current variable by constraints.

Iterative Improvement Algorithms

- Hill Climbing (Gradient Descent)
  - Head towards "best" successor node that is better than present one

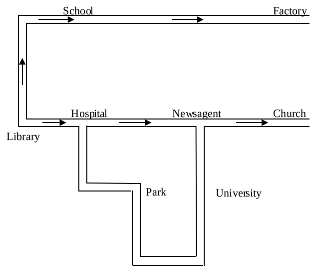
1. Start with *current-state* = initial state.
2. Until *current-state* = goal-state or nor there is no change in *current-state* do:

   (a) Get successors of *current-state* and use evaluation function to assign score to each successor.

   (b) If one of successors has a better score than *current-state* then set new *current-state* to be the successor with the best score

- Problems
  - Local Maxima (Minima)
  - Plateaux
  - Ridges

Hill Climbing cont’d



- Random Restart Hill Climbing
  - improves probability of finding global solution.