Intelligent Systems: Reasoning and Recognition

James L. Crowley

ENSIMAG 2 and MoSIG M1 Winter Semester 2010

Lecture 4 12 February 2010

Problem Solving as Planning
Heuristic Search with GRAPHSEARCH

Nilsson's GRAPHSEARCH Algorithm (A*)

Breadth first, depth first and heuristic search are all variations on the same 3 step algorithm. The algorithm requires maintaining a list of "previously visited" states {C} (Closed list) and a list of available states to explore {O} (Open list).

It constructs a search tree of T, whose root is s.

GRAPHSEARCH Algorithm

Given a node s:
1) Create the list of successors of s: {N}
2) For each n in {N}, if n is in {C} , exit with success, else
    If n is a goal node exit with success,
    else add n to Open List and add n to search tree as child of s.
3) Choose a node, s, from {O}.
    Remove s from {O}.
    Add s to the closed list
    Go to 1

Given node s
{N} = Neighbors(s)
FOR ALL n ∈ {N}
    IF n ∈ {C} THEN {N} := {N} – n
    ELSE
        If n ∈ {G} then EXIT with Success
    ELSE
        {O} <- {O}+n
END
    s := ChooseNode {O}
    {O} := {O} – S
    {C} <- {C} + s
Go to 1.

Step 3 determines the nature of the search:

Breadth First search: {O} is a queue (FIFO)
Depth First search: {O} is a stack (LIFO).
Heuristic search: {O} is sorted based on a cost, f.

Algorithmic Complexity

The algorithm complexity of graph search depends on

b: The branching factor; The average number of neighboring states {N}
b = E{card({N})} (E{} is expectation)
d: Depth. The minimum number of nodes from i to {G}.

Breadth First search: {O} is a queue (FIFO)

For breadth first search, finding the optimal path requires exhaustive search.
Computation Cost $O(b^d)$, memory $O(b^d)$.

Depth First search: {O} is a stack (LIFO).

For depth first search, finding the optimal path requires exhaustive search, however
Computation Cost $O(b^d)$, memory $O(d)$.
However, depth first requires setting a maximum depth d_{max}.

Heuristic search: {O} is sorted based on a cost, f.

For Heuristic search, we reduce the order by reducing the branching factor:
This give Computation and memory of $O(c^d)$ where $c \leq b$.

Heuristic Search is NOT exhaustive. We avoid unnecessary branches.

Nilssons GRAPHSEARCH provides Heuristic search in two forms.

Algorithm A : uses an arbitrary cost estimate.
Algorithme A* : uses an "optimal" cost estimate to produce "optimal" search.

A^* requires that the cost function and cost estimate meet the "optimality conditions".

Cost and Optimality of GRAPHSEARCH

Notation :

S : departure state
B : goal
$k(n_i, n_j)$: minimal theoretical cost between states n_i and n_j
$g^*(n) = k(S, n)$: The cost of the shortest path from S to n.
$h^*(n) = k(n, B)$: The cost of the shortest path from n to B.

$f^*(n) = g^*(n) + h^*(n)$ The cost of the shortest path from S to B passing by n.

Problem: If we do not know the shortest path, how can we know $g^*(n)$ or $h^*(n)$?
Solution estimate the costs.
Define:

$h(n)$: estimated cost from n to B
$g(n)$: estimated cost from S et n.
$f(n) = g(n) + h(n)$

Nilsson showed that whenever $f(n) \leq f^*(n)$, the first path that is found from n_1 to n_2 will always be the shortest.

This requires two conditions:

Condition 1: that the heuristic UNDER-ESTIMATES the cost.

$h(n) \leq h^*(n)$

Condition 2: that h(n) is "monotonic". That is :

$h(n_i) - h(n_j) \leq k(n_i, n_j)$

This is almost always true whenever $h(n) \leq h^*(n)$!!

From this he showed that because first path from S to n is shortest,

$g(n) = g^*(n)$!

Thus as long as $h(n) \leq h^*(n)$ then $f(n) \leq f^*(n)$ and the search is "optimal".

Nilsson called this the A* condition.

A* is "optimal" because the first path found is the shortest path.

Whenever the cost metric is the length of the path, then Euclidean distance to the goal provides an "optimal" heuristic!

This is also true for scalar multiples of distance, for example, time traveled or risk.

(assuming constant speed, distance = speed/time.)

Note that whenever $h(n) = 0$, then $h(n) \leq h^*(n)$!!
This is dijkstra's algorithm, used for network routing.

We can speed up search by using a better h(n) that 0!

Nilsson's GRAPHSEARCH Algorithm

Symbols :

T : Search Treee
G : Set of Goal States
S : Departure State
M : List of Neighbor States
Open : List of Open States
Closed : List of Closed States
n, e : Nodes representing states

Algorithm :

1) Create T, Open and Closed, (initially empty).
2) Place S in Open, and as root to T.
3) LOOP: if OPEN then EXIT with failure
4) Extract n from Open Open <- Open - n, Closed <- Closed + n

5) If n is an element of G, then
Construct a Solution stack with states from N to S.
Unstack solution stack. This is best path.
EXIT with Success.

6) M gets neighbors of n M <- Neighbors(n)

7) For each e in M
IF e is in CLOSED, then Remove M <- M - e.
else
a) For A and A* :Calculate cost f(e) of path from s to G through e
$f(e) = g(e) + h(e) = K(s,e) + h(e)$
b) add e to Open as LIFO (depth first), FIFO (breadth First), sorted list (A*
c) add 2 to T as successor to n

8) Go to step 3.