

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

Planning as Graph Search :.....	2
State Spaces.....	3
Categories of Graph Search	4
Algorithmic Complexity	5
Nilsson's GRAPHSEARCH Algorithm	6
Cost and Optimality of GRAPHSEARCH.....	7

Planning as Graph Search :

A problem is defined by

a universe, $\{U\}$,

an initial state, i

A set of Goal states, $\{G\}$.

Planning is the generation of a sequence of actions to transform i to a state $g \in \{G\}$

The "paradigm" for planning is "Generate and Test".

Given a current state, s

1) Generate all neighbor states $\{N\}$ reachable via 1 action.

2) For each $n \in \{N\}$ test if $n \in \{G\}$. If yes, exit

3) Select a next state, $s \in \{N\}$ and loop.

Planning requires search over a graph for a path.

A taxonomy of graph search algorithms includes the following

1) Depth first search

2) Breadth first search

3) Heuristic Search

4) Hierarchical Search

The first three are unified within the GRAPHSEARCH algorithm of Nilsson.

Graph searching has exponential algorithm complexity.

"knowledge" can be used to reduce the complexity.

State Spaces

In the absence of domain knowledge, an agent needs to explore the entire state space to discover the shortest path from a state "s" to a goal state $g \in \{G\}$.

Domain knowledge can be used to guide this search.

To illustrate this, consider the problem of path planning for a mobile robot.

Navigation requires a "map". The classic map for path planning is a "network of places".

A place is defined as

- 1) A name
- 2) An inclusion test (a predicate $At(x)$)
- 3) A list of "adjacent" places that can be reached by a single action.

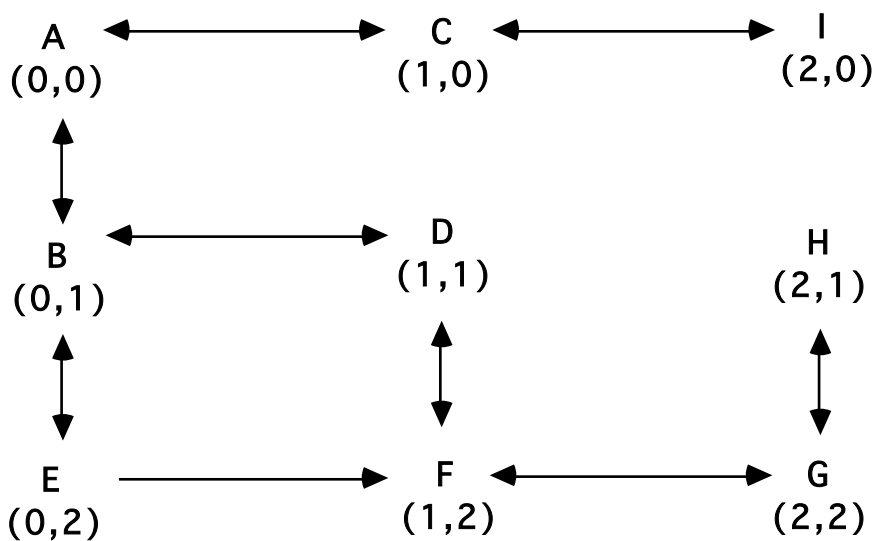
The set of places compose a network. ("A network of places").

Navigation planning requires finding a sequence of places that lead from i to $\{G\}$. The search for a path generates a tree of possible paths.

There are 3 forms of search algorithms that can be used to generate this tree: Depth first, breadth first, and heuristic.

Let us use the following graph to illustrate these three algorithms.

Note that the places are labeled with coordinates (x,y) on a Cartesian map. This allows us to define a metric to evaluate the "cost" of alternative paths.



Given that the robot is at place "E" and we require it to go to H.

Terminology and notation:

- 1) A graph is composed of nodes and arcs.
- 2) Arcs are directed: $n_i \rightarrow n_j$
- 2) Paths through the graph are represented by a tree.
- 3) Within the tree:
 - n_j is the successor (or daughter) of n_i ,
 - n_i is the predecessor (or father) of n_j .

Categories of Graph Search

Breadth first, depth first and heuristic search are all variations on the same 4 step algorithm. The algorithm requires maintaining a list of "previously visited" nodes $\{C\}$ (Closed list) and a list of available nodes to explore $\{O\}$ (Open list).

$n \in \{G\}$

Given a node s :

- 1) Create the list of successors of s : $\{N\} = \text{Neighbors}(s)$
FOR ALL $n \in \{N\}$
- 2) Eliminate any previously visited nodes from $\{N\}$
IF $n \in \{C\}$ THEN $\{N\} := \{N\} - n$
ELSE
- 3) If Not, then Add n to the closed list
 $\{C\} \leftarrow \{C\} + n$
- 4) Test if n is a goal node:
If $n \in \{G\}$ then EXIT with Success
- 5) Add n to the open list
 $\{O\} \leftarrow \{O\} + n$
END
- 6) Choose a node, s , from $\{O\}$. Remove s from $\{O\}$. Go to 1.

Steps 5 (and 6) determine 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

We estimate algorithm complexity with the Order operator $O()$.
Algorithm complexity order is equivalent for all linear functions.

$$O(AN+B) = O(N)$$

The algorithm complexity of graph search depends on

b: The branching factor; The average number of neighboring states $\{N\}$
 $b = E\{\text{card}(\{N\})\}$ ($E\{\}$ 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.

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

Cost of a path through a node n , $f(n)$, includes cost from Start to current node ($g(n)$), plus estimated cost from current node to goal $h(n)$.

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

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

Nilsson's GRAPHSEARCH 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).

Breadth first, depth first and heuristic search are all variations on the same 3 step GRAPHSEARCH algorithm, depending on whether the Open list is a stack, queue or sorted.

Symbols :

- T : Search Tree
- $\{G\}$: Set of Goal States
- S : Departure State
- $\{N\}$: List of Neighbor States
- $\{O\}$: List of Open States
- $\{C\}$: List of Closed States
- n, e : Nodes representing states

GRAPHSEARCH Algorithm: Given a start state S, list of Goal states $\{G\}$.

- 1) Create T, $\{O\}$ and $\{C\}$, (initially empty).
- 2) Place S in $\{O\}$, and as root to T.
- 3) LOOP: if $\{O\} = \{\}$ (empty set) then EXIT with failure
- 4) Extract n from $\{O\}$, add to $\{C\}$: $\{O\} \leftarrow \{O\} - n$, $\{C\} \leftarrow \{C\} + n$
- 5) If $n \in \{G\}$, then
 - Construct a Solution stack with states from N to S.
 - Unstack solution stack. This is best path.
 - EXIT with Success.
- 6) $\{N\}$ gets neighbors of n : $N \leftarrow \text{Neighbors}(n)$
- 7) For each $e \in \{N\}$
 - IF e is in $\{C\}$, then Remove e from $\{N\}$: $\{N\} \leftarrow \{N\} - e$.
 - else
 - a) For A and A* :Calculate cost f(e) of path from S to G through e
 $f(e) = g(e) + h(e)$
 - b) add (e, f(e)) to $\{O\}$.
 If $\{O\}$ is stack (LIFO) then search is depth-first
 If $\{O\}$ is queue (FIFO) then search is breadth-First),
 If $\{O\}$ is sorted on f(e) then search is heuristic (A or A*)
 - c) add e to T as successor to n
- 8) Go to step 3.

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 $h(n)$ than 0!