COMP30030: Introduction to Artificial Intelligence

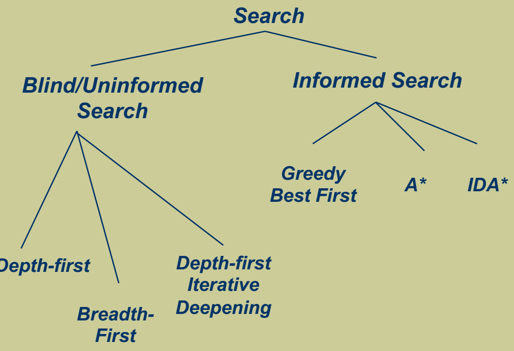
Neil Hurley

School of Computer Science
University College Dublin
neil.hurley@ucd.ie

September 18, 2018

Search

- So far we have seen how to define a problem in terms of states (initial and goal) & operators, and recognize a solution as a sequence of operator applications.



Object: find best path to a goal state

- The next step - **finding the solution**, is achieved by searching for an appropriate sequence of operators. The basic idea in nearly all of the search techniques that we will look at is to maintain and extend a set of partial solution sequences.

Search Costs

- There are two important costs to consider with respect to search-based problem solving:
  - **- Search Cost** – The cost of finding a solution
  - **- Solution Costs** – The cost of using this solution

- Different search techniques offer different guarantees with regards to both of these costs.

- In general the longer one spends searching, the better resulting solution; that is, high search costs usually mean low solution costs.

Search Costs

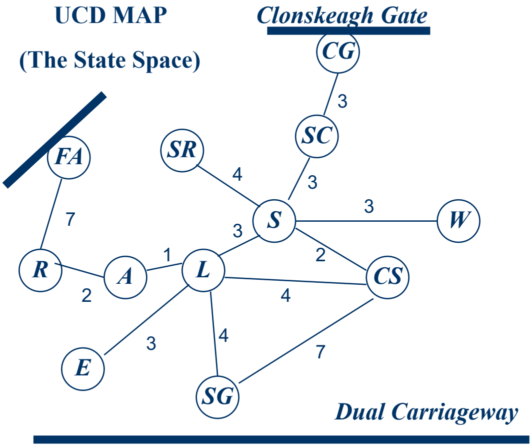
- Depending on the type of problem and the way in which the solution will be used search may or may not be a good idea, and more importantly one particular type of search may be preferred over another.

- Very briefly, if the solution found is to be applied or used on a regular basis then it is important that this solution be as optimal as possible, even if it means sacrificing search time. In other words solution cost should be minimal at the expense of a high search cost.

- On the other hand, if the problem is a one-off then optimizing the solution cost may no longer be a priority. Instead optimizing the search cost may be the number one concern.

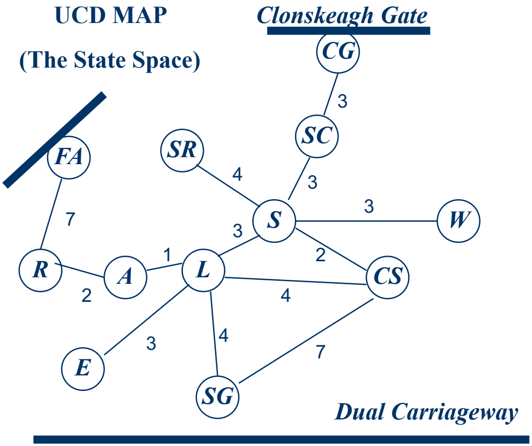
Exploring Search Spaces

- In order to explain many techniques we will look at the problem of route planning, in particular planning a route from Clonskeagh Gate to Stillorgan Gate in the following map of UCD.



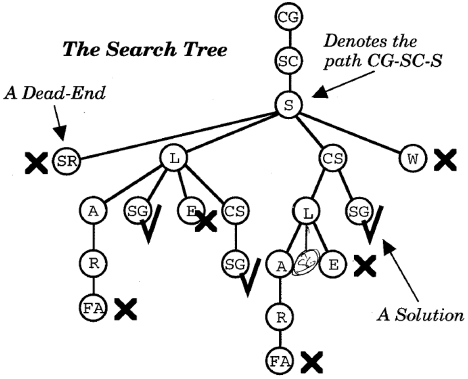
Search Spaces

- In this problem the states are the locations on the map; the map is the state space.
- The initial state is CG and the goal state is the SG.
- There is one operator, **Move**, that allows one to get from state x to state y so long as there is a edge on the map connecting x and y.



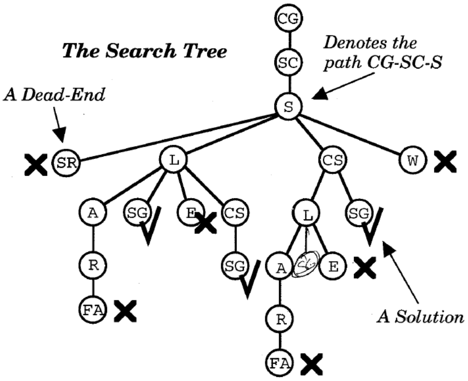
Search Trees

- If we try to find all possible paths from CG then we will certainly find a path to SG, because one does exist.
- We are in effect transforming the map (which is of course a graph) into a **search tree**. Each node in this tree represents a **path** (not a state) even though locations are a more convenient node label – cycles are ignored.
- Search methods start out with no knowledge about the size of these trees. It is crucial to use a search method that is likely to develop the smallest number of paths.



Search Trees

- **It is important to distinguish between the state space and the search tree.**
  - These are only 12 nodes in the UCD state space, one for each location. But there can be an infinite number of search tree nodes (if we permit cycles) and even in the non-cycling search tree above, there are 20 nodes, representing 20 non-cycling paths.



Finding a Path

- The UCD map is a graph with nodes representing locations, and edges representing paths between nodes (with distances attached).
- Finding a path on a graph is not generally the same as the true cost of the real-world trip – it is an abstraction
    - Inessential details are removed.
    - An abstraction is valid if the abstraction solution can be expanded to a solution in the more detailed real-world

Elements of a Search Problem

- An initial state
- Actions are available for each state, often represented as successor(state), which returns a set of (action, state) pairs.
- The goal, which may be a single state or a set of states.
- Each action has an associated cost. The total path cost is the sum of the costs of each chosen action along the path from the initial state to a reached state.

Generating a Search Tree

- Initial state at the root, and successively visited nodes connected by branches.
- The tree starts with only the root. It grows by expanding a chosen node N:
  - Add to the tree the nodes *reachable* from N, with associated paths from N as branches.

Nodes of the Search Tree

- A node is a data structure containing (possibly all of)
    1. STATE: the state in the state space to which the node corresponds
    2. PARENT NODE: the node in the search tree that generated this node;
    3. ACTION: the action that was applied to the parent to generate the node;
    4. PATH COST: the cost, traditionally denoted by $g(n)$, of the path from the initial state to the node, as indicated by the parent pointers;
    5. DEPTH: the number of steps along the path from the initial state.

Fringe Nodes and Open/Closed Lists

- When we expand a node we generate its children and add them onto the tree, for later visit.
- The set of nodes generated but not expanded is called the fringe.
- The search tree process is generally managed through a set of lists:
    1. The closed list contains the set of nodes that have been expanded.
    2. The open list contains the set of fringe nodes.
    3. Whether or not the closed list is used depends on whether we are carrying out *tree search* or *graph search*.

General Search Strategy

```
function STATE-SPACE-SEARCH(strategy)
    Initialise search tree using the initial state.
    while nodes remaining to be expanded do
        choose a node for expansion according to strategy
        if the node contains a goal state then
            return the corresponding solution
        else
            expand the node
        end if
    end while
    return failure
end function
```

General Search Strategy

- Typically the open and closed lists of nodes are stored in a data structure such as a queue
- The "strategy" is then incorporated into the precise nature of this queue – the order in which it nodes are popped from the queue e.g.
  - First-in First-out (FIFO)
  - Last-in First out (LIFO)
  - Priority queue (with some appropriate priority function)

General Tree Search Strategy

```
1: function TREE-SEARCH
2:     fringe ← INSERT(MAKE-NODE(initial_state, fringe))
3:     while not EMPTY(fringe) do
4:         node ← POP(fringe) % strategy encoded in how queue is accessed
5:         if GOALTEST(state) is true then
6:             return SOLUTION(node)
7:         else
8:             fringe ← INSERT-ALL(EXPAND(node),fringe)
9:         end if
10:    end while
11:    return failure
12: end function
```

General Graph Search Strategy

```
function GRAPH-SEARCH
    fringe ← INSERT(MAKE-NODE(initial_state, fringe))
    while not EMPTY(open-list) do
        node ← POP(open-list)
        if GOALTEST(state) is true then
            return SOLUTION(node)
        else
            closed-list ← APPEND(node)
            successors = EXPAND(node) $\setminus$ closed-set
            fringe ← INSERT-ALL(successors,fringe)
        end if
    end while
    return failure
end function
```

- Note that checking the closed list may be costly – hence graph search may not always be feasible.

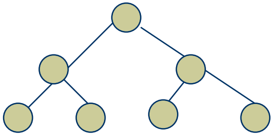
Evaluation Concerns

- **Search methods can be usefully compared in terms of the following characteristics:**
  - **Time complexity**: How long will it take to find a solution? (This is related to search cost)
  - **Space complexity**: How much memory will be used?
  - **Completeness**: When success is possible, is it guaranteed?
  - **Optimality**: Will the best (min solution cost) solution be found? (This will be related to solution cost).

Evaluation Concerns

- With regards to the complexity characteristics, time and space complexity are measured in terms of the number of nodes that need to be examined (expanded) and held in memory respectively.

- These values are computed using information such as the branching factor of the tree (commonly referred to as “b”) and the maximum depth of the tree (usually represented by “d”); if a tree has a branching factor of b then every non-leaf (interior) node has b children.



$b = 2$
$d = 2$

Classes of Search Strategies

- **The search strategy is responsible for deciding which node to expand next.**

- **Uninformed Search**
  - No problem-specific knowledge that would allow to prefer to expend one node over another.
  - Often referred to as **blind** search strategies.

- **Informed Search**
  - Use additional problem-specific knowledge to influence the search.
  - Often referred to as **directed** or **heuristic** strategies.