Muhamad Hesham's T-Blog

Problem Solving Techniques part1

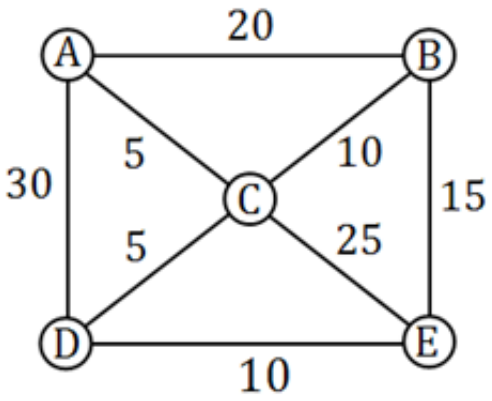
April 8, 2010 — MHesham

We know that the **simple reflex agen**t is one of the simplest agents in design and implementation, it is based on a kind of tables that map a certain action to a corresponding state the environment will be in, this kind of mapping could not be applicable in large and complex environments in which storing the mapping and learning it could consume too much (e.i: Automated taxi driver environment), Such environments may be a better place for **goal-based agents** to arise, that is because such agents consider future actions and their expected outcome.

This article is about giving a brief about a kind of **goal-based agent** called a **problem-solving agent.**

We start here by defining precisely the elements of a problem and its solution, we also have some examples that illustrate more how to formulate a problem, next article will be about a general purpose algorithms that can be used to solve these problems, this algorithms are search algorithms categorized as the following:

- Uniformed search (Blind search): when all we know about a problem is its definition.
- Informed search (Heuristic search): beside the problem definition, we know that a certain action will make us more close to our goal than other action.



In this section we will use a map as an example, if you take fast look you can deduce that each node represents a city, and the cost to travel from a city to another is denoted by the number over the edge connecting the nodes of those 2 cities.

In order for an agent to solve a problem it should pass by 2 phases of formulation:

- **Goal Formulation:**
  - Problem solving is about having a goal we want to reach, (e.i: I want to travel from 'A' to 'E').
  - Goals have the advantage of limiting the objectives the agent is trying to achieve.
  - We can say that goal is a set of environment states in which our goal is satisfied.
- **Problem Formulation:**
  - A problem formulation is about deciding what actions and states to consider, we will come to this point it shortly.
  - We will describe our states as "in(*CITYNAME*)" where *CITYNAME* is the name of the city in which we are currently in.

Now suppose that our agent will consider actions of the from "Travel from city A to City B". and is standing in city 'A' and wants to travel to city 'E', which means that our current state is in(A) and we want to reach the state in(E).

There are 3 roads out of A, one toward B, one toward C and one toward D, none of these achieves the goal and will bring our agent to state in(E), given that our agent is not familiar with the geography of our alien map then it doesn`t know which road is the best to take, so our agent will pick any road in random.

Now suppose that our agent is updated with the above map in its memory, the point of a map that our agent now knows what action bring it to what city, so our agent will start to study the map and consider a hypothetical journey through the map until it reaches E from A.

Once our agent has found the sequence of cities it should pass by to reach its goal it should start following this sequence.

The process of finding such sequence is called **search**, a search algorithm is like a black box which takes **problem** as input returns a **solution**, and once the solution is found the sequence of actions it recommends is carried out and this is what is called the **execution phase.**

We now have a simple (formulate, search, execute) design for our problem solving agent, so lets find out precisely how to formulate a problem.

Formulating problems

A problem can be defined formally by 4 components:

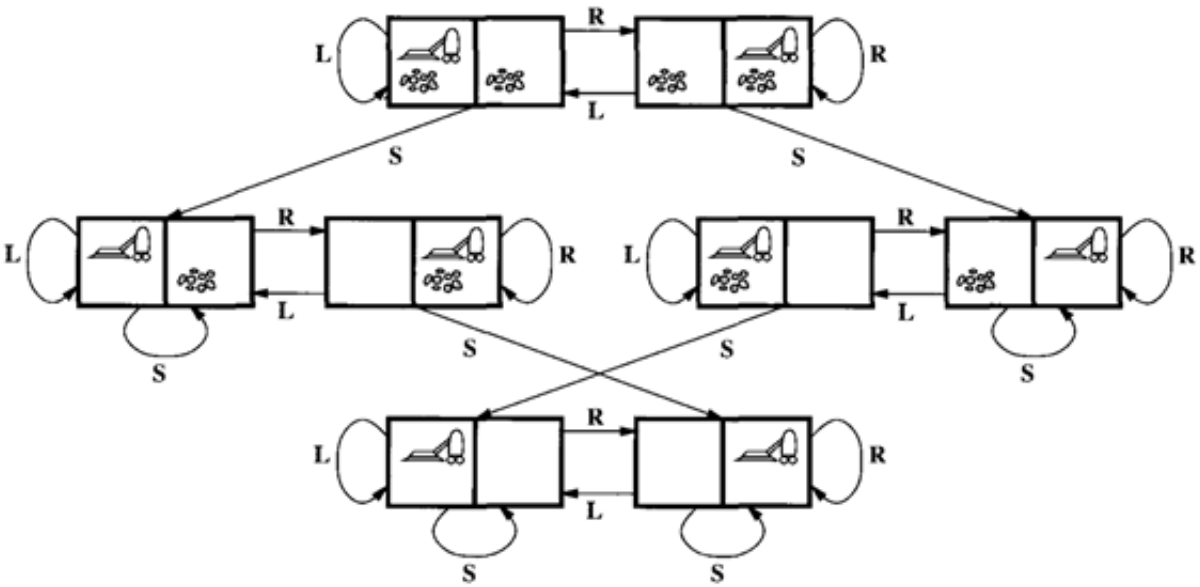
1. Initial State:
   - it is the state from which our agents start solving the problem {e.i: in(A)}.
2. State Description:
   - a description of the possible **actions** available to the agent, it is common to describe it by means of a **successor function**, given state *x* then *SUCCESSOR-FN(x)* returns a set of ordered pairs <*action*, *successor*> where *action* is a legal action from state *x* and successor is the state in which we can be by applying *action*.

- The initial state and the successor function together defined what is called **state space** which is the set of all possible states reachable from the initial state {e.i: in(A), in(B), in(C), in(D), in(E)}.

3. Goal Test:
   - we should be able to decide whether the current state is a *goal state* {e.i: is the current state is in(E)?}.
4. Path cost:
   - a function that assigns a numeric value to each path, each step we take in solving the problem should be somehow weighted, so If I travel from A to E our agent will pass by many cities, the cost to travel between two consecutive cities should have some cost measure, {e.i: Traveling from 'A' to 'B' costs 20 km or it can be typed as c(A, 20, B)}.

A solution to a problem is path from the initial state to a goal state, and **solution quality** is measured by the path cost, and the **optimal solution** has the lowest path cost among all possible solutions.

Example Problems

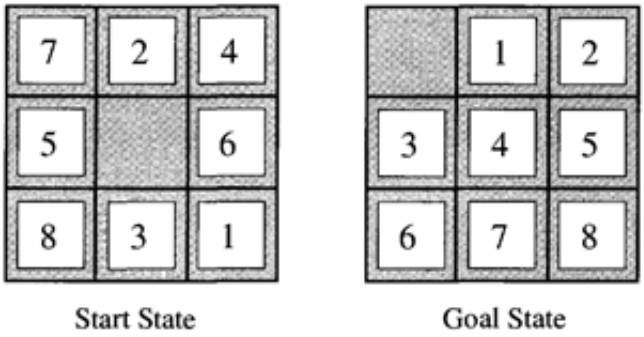


Vacuum world

1. Initial state:
   - Our vacuum can be in any state of the 8 states shown in the picture.
2. State description:
   - Successor function generates legal states resulting from applying the three actions {Left, Right, and Suck}.
   - The states space is shown in the picture, there are 8 world states.
3. Goal test:

- Checks whether all squares are clean.

4. Path cost:
   - Each step costs 1, so the path cost is the sum of steps in the path.



Start State Goal State

8-puzzle

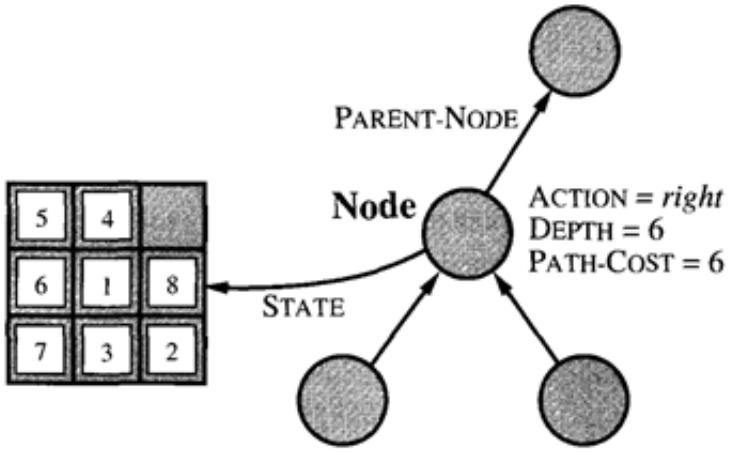
1. Initial state:
   - Our board can be in any state resulting from making it in any configuration.
2. State description:
   - Successor function generates legal states resulting from applying the three actions {move blank Up, Down, Left, or Right}.
   - State description specifies the location of each of the eight titles and the blank.
3. Goal test:
   - Checks whether the states matches the goal configured in the goal state shown in the picture.
4. Path cost:
   - Each step costs 1, so the path cost is the sum of steps in the path.

Searching

After formulating our problem we are ready to solve it, this can be done by searching through the state space for a solution, this search will be applied on a **search tree** or generally a graph that is generated using the initial state and the successor function.

Searching is applied to a search tree which is generated through state expansion, that is applying the successor function to the current state, note that here we mean by state a node in the search tree.

Generally, search is about selecting an option and putting the others aside for later in case the first option does not lead to a solution, The choice of which option to expand first is determined by the **search strategy** used.



The structure of a node in the search tree can be as follows:

1. *State:* the state in the state space to which this state corresponds
2. *Parent-Node:* the node in the search graph that generated this node.
3. *Action:* the action that was applied to the parent to generate this node.
4. *Path-Cost:* the cost of the path from the initial state to this node.
5. *Depth:* the number of steps along the path from the initial state.

It is important to make a distinction between nodes and states, A node in the search tree is a data structure holds a certain state and some info used to represent the search tree, where state corresponds to a world configuration, that is more than one node can hold the same state, this can happened if 2 different paths lead to the same state.

Measuring problem-solving performance

Search as a black box will result in an output that is either **failure** or a **solution**, We will evaluate a search algorithm`s performance in four ways:

1. Completeness: is it guaranteed that our algorithm always finds a solution when there is one ?
2. Optimality: Does our algorithm always find the optimal solution ?
3. Time complexity: How much time our search algorithm takes to find a solution ?
4. Space complexity: How much memory required to run the search algorithm?

Time and Space in complexity analysis are measured with respect to the number of nodes the problem graph has in terms of asymptotic notations.

In AI, complexity is expressed by three factors ***b***, ***d*** and **m:**

1. *b* the **branching factor** is the maximum number of successors of any node.

2. *d* the **depth** of the deepest goal.
3. *m* the maximum **length** of any path in the state space.

The next article will be about uniformed search strategies and will end with a comparison that compares the performance of each search strategy.

References:

- Artificial Intelligence: A Modern Approach (2nd Edition) by Stuart Russell and Peter Norvig
- Wikipedia articles

Posted in Artificial Intelligence. Tags: ai, blind search, goal based agent, measuring problem solving performance, problem formulation, problem solving agent, problem solving techiques, search strategy, search tree, searching, uniformed search. 3 Comments »