Uninformed Search

18 October 2020 21:18

Objectives:

- Problem solving agents
- Problem types
- Problem formulation
- State space graphs
- Basic tree search and graph search algorithms

An agent could be programmed to act in the world to achieve a fixed goal or set of goals, but then it would not adapt to changing goals, and so would not be intelligent. *An intelligent agent needs to reason about its abilities and its goals* to determine what to do.

A problem-solving agent is a goal-based agent that will determine a sequence of actions that will achieve some goal state

There are 4 steps a problem solving agent must take

- Goal formulation
- Problem formulation
- Search – find the sequence of actions
- Execution

In the simplest scenario, an agent has no uncertainty, and has a state based model of the world, with a goal to achieve. This is either a flat representation or a single level of a hierarchy. The agent is able to determine how to achieve this goal by searching in its representation of the world state space for a way to get from its current state to a state that satisfies its goal.

Given a complete model, it tries to find a sequence of actions that will achieve its goal before it has to act in the world.

This problem is like finding a path from the start node to an end node in a directed graph.

Problem Types

- Deterministic, fully observable – single state problem
- Deterministic, partially observable – multi-state problem
- Stochastic, partially observable – contingency probably
- Unknown state space

Searching

"searching" in this chapter means searching in an internal representation for a path to a goal – from start node to end node.

Search underlies much of artificial intelligence. When an agent is given a problem, it is usually given a description that allows it to recognise a solution, but not an actual algorithm for a solution. The agent has to search for a solution itself.

The difficulty of search and the fact that humans are able to solve some search problems efficiently suggests that computer agents should exploit knowledge about special cases to guide them to a solution. This extra knowledge beyond the search space is called **heuristic knowledge**. This chapter considers one kind of heuristic knowledge in the form of an estimate of the cost from a node to a goal.

State Space

One general formulation of intelligent action is in terms of a state space. A state contains all of the information necessary to compute the effects of an action and to determine whether a states satisfies a goal.

When we do a state space search, we assume:

1. The agent has perfect knowledge of the state space and is planning for the case where it observes what state it is in: there is full observability
2. The agent has a set of actions that have known deterministic effects
3. The agent can determine whether a state satisfies a goal

A solution is a sequence of actions that will get the agent from the current state to the state that satisfies the goal.

A state space problem consists of:

- A set of states
- A distinguished state called the start state
- For each state, a set of actions available to the agent in that state
- An agent function that, given a state and an action, returns a new state
- A goal specified as a Boolean function that is true when state satisfies the goal, in which case we can say that s is a goal state
- A criterion that specifies the quality of an acceptable solution. For example, any sequence of actions that gets the agent to the goal state may be acceptable, or there may be costs associated with actions and the agent may be required to find a sequence that has minimal total cost. A solution that is best according to some criterion is called an optimal solution. We do not always need an optimal solution, for example, we may be satisfied with any solution that is within 10% of optimal.

Graph Searching

In this chapter, the problem of finding a sequence of actions to achieve a goal is abstracted as searching for paths in directed graphs.

To solve a problem, we first define the underlying search space, then apply a search algorithm to that search space. Many problem solving tasks are transformable into the problem of finding a path in a graph.

A directed graph consists of a set of nodes and a set of directed arcs between nodes. The idea is to find a path along these arcs from the start node to a goal node.

In representing a state-space problem, the states are represented as nodes, and the actions as arcs.

The abstraction is necessary because there may be more than one way to represent a problem as a graph. The examples in this chapter are in terms of state-space searching, where nodes represent states and arcs represent actions.

Formal Graph Searching

A directed graph has a set N of nodes and a set A of arcs, where an arc is an ordered pair of nodes.

Nodes are neighbours if there is an arc between them. Note that the relationship is not symmetrical – it doesn't necessarily go both ways.

A path from node s to node g is a sequence of nodes (n0, n1, … nk) such that s = n0 and g = nk, and there is an arc between each ni and ni+1.

A goal is a Boolean function on nodes. If goal(n) is true, we say that node n satisfies the goal, and n is a goal node.

To encode problems as graphs, one node is identifies as a start node. A solution is a path from the start node to a node that satisfies the goal.

Sometimes there is a cost, a non-negative number associated with the arcs. We write the cost of an arc as cost(ni, nj). The costs of arcs induce a cost of paths. Given a path p, cost of path p is the sum of the costs of all the arcs in the path.

An optimal solution is the solution that has the lowest cost.

That is, an optimal solution is a path p from the start node to a goal node such that there is no path p' from the start node to a goal node where the cost is lesser.

A cycle is a non-empty path where the end node is the same as the start node. A directed graph without any cycles is called a directed acyclic graph.

A tree is a DAG where there is one node with no incoming arcs and every other node has exactly one incoming arc. The node with no incoming arcs is the root of the tree. A node with no outgoing arcs is a leaf. In a tree, neighbours are called children.

In many problems, the search graph is not given explicitly, but is constructed as needed. For the search algorithms, all that is required is a way to generate the neighbours of a node, and to determine if the node is a goal node.

The forward branching factor of a node is the number of outgoing arcs. The backward branching factor is the number of incoming arcs to the node. These factors provide measures for the complexity of graph algorithms. We assume the branching factors are bounded, when we discuss the space and time complexity.