COMP30024: Artificial Intelligence

Semester 1, 2025

Contents

3 Week 3: Informed Search Algorithms

3.1 Informed search strategies

Additional information acquired using heuristics to estimate how close the current state is to the goal state

Best-first search

Estimate the desirability of each node using an evaluation function, then expand the most desirable node first (ie. insert successors in decreasing order of desirability)

Greedy search

Evaluation function *h(n)* = estimated cost of n to goal
Expands the node that *appears* to be closest to goal

- incomplete as it may get stuck in loops, complete in finite space with repeated-state checking
- time & space complexity: $O(b^m)$ if the heuristic is bad (will explore all node in worse case)
- not optimal, as the estimated distance may be different from the actual distance

Strongly depends on how good the heuristic function is

A* search

Avoid expanding paths that are already expensive by using evaluation function $f(n) = g(n) + h(n)$
    $g(n)$ = cost so far to reach n (path cost)
    $h(n)$ = estimated cost to goal from n
    $f(n)$ = estimated total cost of path through n to goal
Expands the node that has the lowest value of $f(n)$

Note: A* search uses admissible heuristic, ie $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost

- Complete, unless there are infinitely many nodes
- time: exponential in (relative error of h $\times$ length of solution)
- space: keeps al nodes in memory
- optimal: expands nodes in increasing order of f, gradually adds "f-contours" of nodes and cannot expand f_{i+1} until f_i is finished

There is also no other search strategies that is guaranteed to expand fewer nodes than A* given the same heuristic, but the performance still depends on how good the heuristic is.

3.2 Admissible Heuristics

If $h_2(n) \geq h_1(n)$ (both admissible), then $h_2(n)$ dominates $h_1(n)$ and is a better heuristic

Admissible heuristics can be derived from the *exact* solution cost of a *relaxed* version of the problem

3.3 Iterative Improvement Algorithms

For problems where we only want to reach the goal and the path is irrelevant (e.g. implicit goal test)
Then the state space is the set of complete configurations, and we might want to find the optimal solution (Travelling Salesman) or the configuration that satisfy certain constraints (n-Queens)

Solve by relaxing the problem/constraints, find a 'single' state and improve it
Suitable for both online and offline search
e.g. Travelling Salesman Problem
    - relax the path into any structure that connect all cities: use MST as heuristics
e.g. n-Queens
    - relax constraints: start from only forbidding queens on the same column, then move queens around

Other examples of such problems: scheduling problems, timetabling, electricity load optimisation

5 Week 5: Machine Learning in Game Search

5.1 Some Strategies

Book Learning

Aim: Learn sequence of moves for important positions
e.g. books of opening move → Remember the move taken and the final outcome for every position seen in an opening game
e.g. Learn from mistakes → identify moves that lead to a loss and whether there was a better alternative
Problem: How to recognise which moves were important?

Adjusting Search Parameters

Aim: Learn how to make search more efficient
e.g. Learn a preferred order of generating possible moves to maximise effectiveness of $\alpha - \beta$ pruning
e.g. Learn a classifier to predict what depth we should search to based on current states (e.g. more breadth at the start of a game of chess, more depth later in the game)

Adjusting Weights in Evaluation Functions

Aim: Adjust weights in evaluation function based on experience of their ability to predict the true utility

5.2 Types of Machine Learning

Supervised learning

Use a set of training examples corresponding to the set of features for a state and the true minimax utility value of the state $d =< f_1(s),\ ...,\ f_n(s), U(s) >$ to learn a set of weights $w =< w_1,\ ...,\ w_n >$ so that the output $z = EVAL(s; w)$ closely approximates the true output $U(s)$ on the training examples (and hopefully on new states)

Problems: - delayed reinforcement: reward from an action may not be received until several time steps later
→ no immediate feedback
-credit assignment: need to know which move was responsible for the outcome

Temporal Difference Learning (TD)

For multi-step prediction, e.g. predict outcome of game based on first move, then update prediction as more moves are made
- correctness of prediction not known until several steps later
- intermediate steps provide information about correctness of prediction
- a form of reinforcement learning

e.g. TDLeaf(λ) algorithm: combines TD learning with minimax search
Update weight in evaluation function to reduce differences in rewards predicted at different levels in search tree (should be stable from one move to next)

5.3 Monte Carlo Tree Search

5.3.1 Motivation

For games that are hard to find a good evaluation function (due to high branching factor, changing positions of the pieces) as it does not depend on an evaluation function
Plays the game all the way to the end multiple times and see how often we win or lose and use it as a guide to how good the move is