Advanced Programming Topics

Algorithm analysis and design - dynamic programming and optimisation
Lecture 10
James Marshall

In the last lecture was saw *dynamic programming* used for efficiently computing the results of functions. The technique originates in *decision optimisation*, and the principle of optimal substructure was formalised by Richard Bellman in what is now known as the *Bellman equation.*

Definition - the Bellman Equation

Examples

Example 1 - Getting from A to B (Dijkstra's Shortest Path Algorithm)

Naive solution: try all possible paths from A to B

Algorithm idea: if C is on the shortest path from A to B it doesn't matter how you got there (optimal substructure *and* overlapping subproblems (see naive solution; all possible paths from A to B also includes all possible paths from A to C and from C to B))

Algorithm overview: for each node store the best total cost to get to that node so far, and the preceding node on that path; successively update these. At each update step choose the lowest total distance unvisited node (see 'greedy algorithms' below)

Example 2 - Gambling (One-armed bandits and Gittins indices)

Imagine you are in a casino lobby somewhere like Las Vegas:

- In the lobby you are choosing between two one-armed bandits to play, which you believe have different probabilities of paying out on each pull of the arm (trial)
- You used to work for the manufacturer so you know the distribution of success probabilities of these kinds of machines
- You have a pile of dimes to play with - for each dime you have to decide which bandit to put it in
- You only get information about the bandits' payout probabilities by playing them, and given how valuable future payouts to you are (β in the Bellman equation) you want to maximise your long-term discounted reward by eventually settling on only playing the highest probability bandit
- Bayes' rule and dynamic programming you can give you the provably optimal strategy (calculating Gittins indices)… in other words, the most (expected) cash possible!

Optimisation: Dynamic Programming vs Greedy Algorithms

In optimisation problems, at each step in constructing a solution a *greedy algorithm* makes the *locally best choice*; this can be referred to as a *greedy choice* or a *myopic choice*

Greedy algorithms can have advantages and disadvantages:

- Greedy algorithms are usually (much) faster than dynamic programming algorithms…
- …but may not guarantee the optimal solution will be constructed, *although they can be shown to for many important problems, such as computing shortest paths (e.g. Dijkstra's algorithm)!*

N.B. just because a greedy algorithm exists to optimally solve a problem, that does not mean *any* greedy algorithm for that problem will lead to an optimal solution. *E.g.* "*take the next train departing London St Pancras*" would be a greedy route-planning algorithm, but not one guaranteed to get you where you want to go quickly!