---

1 Dynamic Programming Optimization

Dynamic programming is a coding style that allows us to find optimal solutions to problems more efficiently. In order to make this process easier, there are important steps that should be followed.

1.1 Steps of Dynamic Programming Optimization:

1. Establish a recursive property that gives an optimal solution to the problem.
2. Compute the value of the optimal solution.
3. Construct the optimal solution.

Steps 2 and 3 are implemented with bottom-up algorithms.

1.2 Example of optimization process using Floyd's algorithm

1. Establish a recursive property that gives an optimal solution to the problem.

   In order to find a shortest path between any two vertices, any smaller path within that path would also have to be a shortest path between the two intermediate vertices.

2. Compute the value of the optimal solution.

   Computed matrix D, which is a matrix of all of the weights of the shortest paths. This matrix holds the value(s) of the optimal solution(s).

3. Construct the optimal solution.

   Constructed the optimal solution P, which is the path we follow to get one of the weights in D.

2 The principle of optimization

Some optimization problems *cannot* be solved with dynamic programming. For a problem to be optimized using dynamic programming, the principle of optimization must be satisfied.

The principle of optimization states that an optimal solution to an instance of a problem always contains optimal solutions to all sub-problems.

2.1 Proof using Floyd's algorithm

Floyd's algorithm works as a great example in order to better understand how the principle of optimization works.

2.1.1 Proof of the Principle of Optimization:

Let G = (V, E) be a graph, and let $p =< v_1, v_2, \ldots, v_k >$ be a shortest path in G from v_1 to v_k.

Claim: For $1 \leq i < j \leq k$, the path $p =< v_i, v_{i+1}, \ldots, v_k >$ is a shortest path from v_i to v_j.

Proof (by contradiction):

Suppose $< v_i, v_{i+1}, \ldots, v_j >$ is *not* a shortest path in G from v_i to v_j.

Then there exists another path, p', from v_i to v_j that is shorter.

Then the path v_1 to v_i–$p' - v_j$ to v_k is a shorter path from v_1 to v_k.

This contradicts that p is a shortest path from v_1 to v_k.

Therefore, p is a shortest path from v_i to v_j.

2.2 Example of a Problem without the Principle of Optimization

The longest path problem is an optimization problem that cannot be solved using dynamic programming. This is because the principle of optimization does not apply.

Goal: Find the longest path between two vertices.
Only consider simple paths (Do not repeat vertices.). This stops an "infinite loop".

Consider the following graph:

Goal: Find the longest path from v_1 to v_4.
The optimal (longest) path is $< v_1, v_3, v_2, v_4 >$ which has a total length of $1 + 2 + 4 = 7$.

However, the principle of optimization states that all the sub-paths must be the longest available sub-paths. This does not hold.

Proof:
v_1 has two options, v_2 and v_3. The weights of each of the edges are 1.
However, v_3 to v_2 is 2 and v_2 to v_3 is 3.
So the two sub-paths have different total weights:
$< v_1, v_3, v_2 >$ has the total weight of $1 + 2 = 3$
$< v_1, v_2, v_3 >$ has the total weight of $1 + 3 = 4$

The sub-path from v_1 to v_3 is a longer path from v_1 to v_2. However, we cannot include this in our final answer since there is a longest path if we include the edge from v_3 to v_4. Thus, the principle of optimization does not hold.

3 Traveling Salesperson Problem (TSP)

In order to solve the traveling salesperson problem (TSP), we must find a Hamiltonian cycle of minimal length. A cycle is a path that ends at the same vertex which it started. A Hamiltonian cycle is a cycle which visits each vertex exactly once. It is also known as a tour. To find the cycle, you can start at any vertex. However, the convention is to start at v_1.

Goal: Find an optimal tour to minimize the total distance traveled.

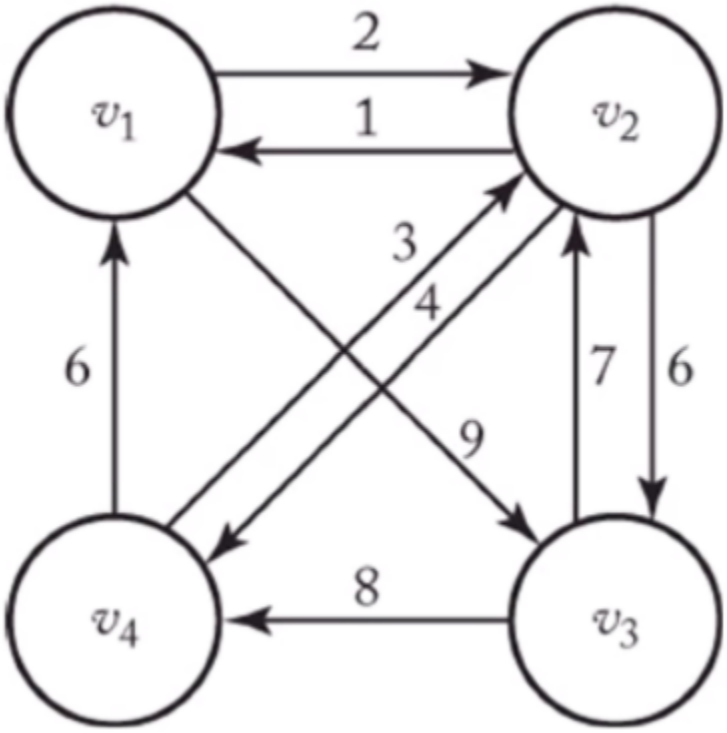

Figure 3.16

3.1 Brute Force Algorithm

There are three tours in this graph.

$H_1 =< v_1, v_2, v_3, v_4, v_1 >$ with a total weight of 22.
$H_2 =< v_1, v_3, v_2, v_4, v_1 >$ with a total weight of 26.
$H_3 =< v_1, v_3, v_4, v_2, v_1 >$ with a total weight of 21.

Since 21 is the smallest total weight, H_3 is the most optimal tour.

However, this type of algorithm is extremely inefficient because the total number of possible tours is $(n-1)!$ with n being the number of vertices. This would make the runtime of this brute force algorithm worse than exponential time.

3.2 Dynamic Programming Approach

In order for this problem to be solved using dynamic programming, the principle of optimization must hold. So, does the principle of optimization hold? Yes.

> Let G = (V,E) be a graph, and H be a tour with $H =< v_1, v_2, \ldots, v_n, v_1 >$.
>
> Claim: $H' =< v_i, v_i + 1, \ldots, v_j >$ is a shortest path from v_i to v_j that passes through each vertex in $\{v_{i+1}, v_{i+2}, \ldots, v_j\}$ exactly once.
>
> Proof (by contradiction):
>
> > Suppose the claim is false.
> > Then there is a path from v_i to v_j that passes through $\{v_{i+1}, v_{i+2}, \ldots, v_j\}$ exactly once that is shorter than H'.
> > This contradicts the definition of a tour.