COL351: Analysis and Design of Algorithms

Ragesh Jaiswal, CSE, IITD

Dynamic Programming

Problem

There are n items with unlimited copies of each item available. Each copy of the i^{th} item has weight $w(i)$ and value $v(i)$. You have a knapsack of capacity W. All $w(i)$'s and W are positive integers. Your goal is to determine how many copies of every item to put in the knapsack so that the capacity of the knapsack is not exceeded and the total value of items is maximized.

- Dynamic Programming solution:
  - $V(i, w)$: The maximum value that can be obtained using items $\{1, ..., i\}$ given the knapsack has capacity w.
  - If $w(i) > w$, then $V(i, w) =$?
  - If $w(i) \leq w$, then $V(i, w) =$?
  - $\forall w \leq W$, $V(1, w) =$?

Problem

There are n items with unlimited copies of each item available. Each copy of the i^{th} item has weight $w(i)$ and value $v(i)$. You have a knapsack of capacity W. All $w(i)$'s and W are positive integers. Your goal is to determine how many copies of every item to put in the knapsack so that the capacity of the knapsack is not exceeded and the total value of items is maximized.

- Dynamic Programming solution:
  - $V(i, w)$: The maximum value that can be obtained using items $\{1, ..., i\}$ given the knapsack has capacity w.
  - If $w(i) > w$, then $V(i, w) = V(i-1, w)$
  - If $w(i) \leq w$, then $V(i, w) = \max\{V(i-1, w), V(i, w - w(i)) + v(i)\}$
  - $\forall w \leq W$, $V(1, w) = \lfloor \frac{w}{w(1)} \rfloor \cdot v(1)$.
- What is the running time of the table-filling algorithm based on the above recursive formulation?

Problem

There are n items with unlimited copies of each item available. Each copy of the i^{th} item has weight $w(i)$ and value $v(i)$. You have a knapsack of capacity W. All $w(i)$'s and W are positive integers. Your goal is to determine how many copies of every item to put in the knapsack so that the capacity of the knapsack is not exceeded and the total value of items is maximized.

- Dynamic Programming solution:
  - $V(i, w)$: The maximum value that can be obtained using items $\{1, ..., i\}$ given the knapsack has capacity w.
  - If $w(i) > w$, then $V(i, w) = V(i-1, w)$
  - If $w(i) \leq w$, then $V(i, w) = \max\{V(i-1, w), V(i, w - w(i)) + v(i)\}$
  - $\forall w \leq W,\ V(1, w) = \lfloor \frac{w}{w(1)} \rfloor \cdot v(1)$.
- What is the running time of the table-filling algorithm based on the above recursive formulation? $O(n \cdot W)$

- The 0-1 Knapsack problem falls into the category of very difficult problems called NP-hard problems.
- It is unlikely that we can solve such problems in time polynomial in the input size.

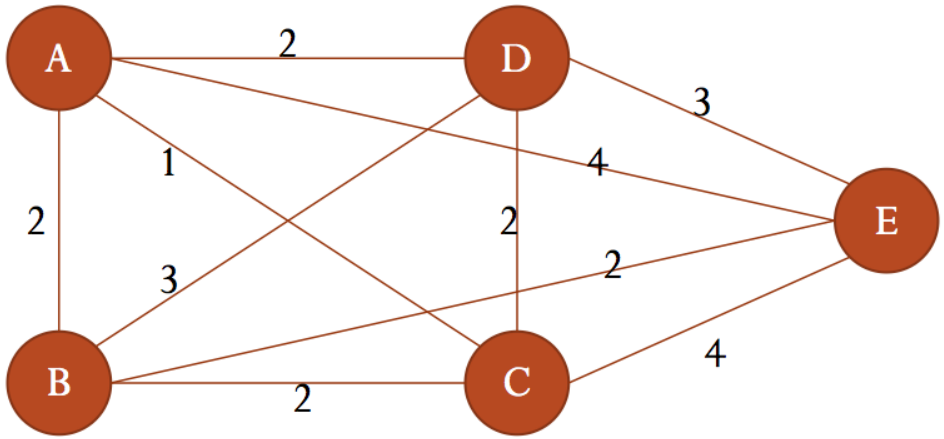
- The 0-1 Knapsack problem falls into the category of very difficult problems called NP-hard problems.
- It is unlikely that we can solve such problems in time polynomial in the input size.
- Suppose each weight is specified using n bits. Then the input size is $m = n^2$. The running time of the Dynamic Programming algorithm is $O(n \cdot 2^n)$!

- The 0-1 Knapsack problem falls into the category of very difficult problems called NP-hard problems.
- It is unlikely that we can solve such problems in time polynomial in the input size.
- Suppose each weight is specified using n bits. Then the input size is $m = n^2$. The running time of the Dynamic Programming algorithm is $O(n \cdot 2^n)$!
- Advantage of DP solution: For certain problems where W is small (e.g., small constant independent of n), we get a fast algorithm.

Problem

Traveling Salesperson: There are n cities and all the inter-city distances are given.Let $D(i, j)$ denote the distance between cities i and j. You are a salesperson and would like to visit each of the n cities starting and ending at your home city. Give a tour that minimizes the total distance you have to travel.

- Example:



Problem

Traveling Salesperson: There are n cities and all the inter-city distances are given.Let $D(i, j)$ denote the distance between cities i and j. You are a salesperson and would like to visit each of the n cities starting and ending at your home city. Give a tour that minimizes the total distance you have to travel.

- Suppose our home city is city 1
- How many different tours are possible?

Problem

Traveling Salesperson: There are n cities and all the inter-city distances are given.Let $D(i, j)$ denote the distance between cities i and j. You are a salesperson and would like to visit each of the n cities starting and ending at your home city. Give a tour that minimizes the total distance you have to travel.

- Suppose our home city is city 1
- How many different tours are possible? $(n-1)!$
- Let S be a subset of cities containing city 1 and city j
- Let $T(S, j)$ denote the shortest path between cities 1 and j such that all cities in S are visited once on this path.
- Try writing $T(S, j)$ in terms of $T(S', j')$ where $|S|'$ is smaller than $|S|$

Problem

Traveling Salesperson: There are n cities and all the inter-city distances are given.Let $D(i,j)$ denote the distance between cities i and j. You are a salesperson and would like to visit each of the n cities starting and ending at your home city. Give a tour that minimizes the total distance you have to travel.

- Suppose our home city is city 1
- How many different tours are possible? $(n-1)!$
- Let S be a subset of cities containing city 1 and city j
- Let $T(S,j)$ denote the shortest path between cities 1 and j such that all cities in S are visited once on this path.
- $T(S,j) = \min_{i \in S, i \neq j} \{T(S-\{j\}, i) + d(i,j)\}$
  - The first city on the path above is 1 and the last city is j. Check all possibilities for second-to-last city.

Problem

Traveling Salesperson: There are n cities and all the inter-city distances are given.Let $D(i,j)$ denote the distance between cities i and j. You are a salesperson and would like to visit each of the n cities starting and ending at your home city. Give a tour that minimizes the total distance you have to travel.

Algorithm

`TSP-length`

- $T(\{1\},1) \leftarrow 0$
- For $s = 2$ to n
  - For all subsets S of $\{1,...,n\}$ of size s containing 1
    - $T(S,1) \leftarrow \infty$
    - For all $j \in S$ s.t. $j \neq 1$
      - $T(S,j) \leftarrow \min_{i \in S, i \neq j} \{T(S-\{j\},i) + d(i,j)\}$
- return$(\min_j \{T(\{1,...,n\},j) + d(1,j)\})$

- What is the running time of the above algoithm?

Problem

Traveling Salesperson: There are n cities and all the inter-city distances are given.Let $D(i,j)$ denote the distance between cities i and j. You are a salesperson and would like to visit each of the n cities starting and ending at your home city. Give a tour that minimizes the total distance you have to travel.

Algorithm

`TSP-length`

- $T(\{1\},1) \leftarrow 0$
- For $s = 2$ to n
  - For all subsets S of $\{1,...,n\}$ of size s containing 1
  - $T(S,1) \leftarrow \infty$
  - For all $j \in S$ s.t. $j \neq 1$
    - $T(S,j) \leftarrow \min_{i \in S, i \neq j} \{T(S-\{j\},i) + d(i,j)\}$
- return$(\min_j \{T(\{1,...,n\},j) + d(1,j)\})$

- What is the running time of the above algoithm? $O(n^2 \cdot 2^n)$

Course Overview

- Graph Algorithms
- Algorithm Design Techniques:
  - Greedy Algorithms
  - Divide and Conquer
  - Dynamic Programming
  - Network Flows
- Computational Intractability

Network Flow

- Main Idea: Reduction
  1. We will obtain an algorithm A for a *Network Flow* problem.
  2. Given a new problem, we will *rephrase* this problem as a Network Flow problem.
  3. We will then use algorithm A to solve the rephrased problem and obtain a solution.
  4. Finally, we build a solution for the original problem using the solution to the rephrased problem.

- We want to model various kinds of networks using graphs and then solve real world problems with respect to these networks by studying the underlying graph.
- One problem that arises in network design is routing "flows" within the network.
  - Transportation Network: Vertices are cities and edges denote highways. Every highway has certain traffic capacity. We are interested in knowing the maximum amount commodity that can be shipped from a source city to a destination city.
  - Computer Networks: Edges are links and vertices are switches. Each link has some capacity of carrying packets. Again, we are interested in knowing how much traffic can a source node send to a destination node.

- To model these problems, we consider weighted, directed graph $G = (V, E)$ with the following properties:
  - Capacity: Associated with each edge e is a capacity that is a non-negative integer denoted by $c(e)$.
  - Source node: There is a source node s with no in-coming edges.
  - Sink node: There is a sink node t with no out-going edges. All other nodes are called *internal nodes*.

- To model these problems, we consider weighted, directed graph $G = (V, E)$ with the following properties:
  - Capacity: Associated with each edge e is a capacity that is a non-negative integer denoted by $c(e)$.
  - Source node: There is a source node s with no in-coming edges.
  - Sink node: There is a sink node t with no out-going edges.
    All other nodes are called *internal nodes*.
- Given such a graph, an "$s - t$ flow" in the graph is a function f that maps the edges to non-negative real numbers such that the following properties are satisfied:
  (a) Capacity constraint: For every edge e, $0 \leq f(e) \leq c(e)$.
  (b) Flow conservation: For every internal node v:

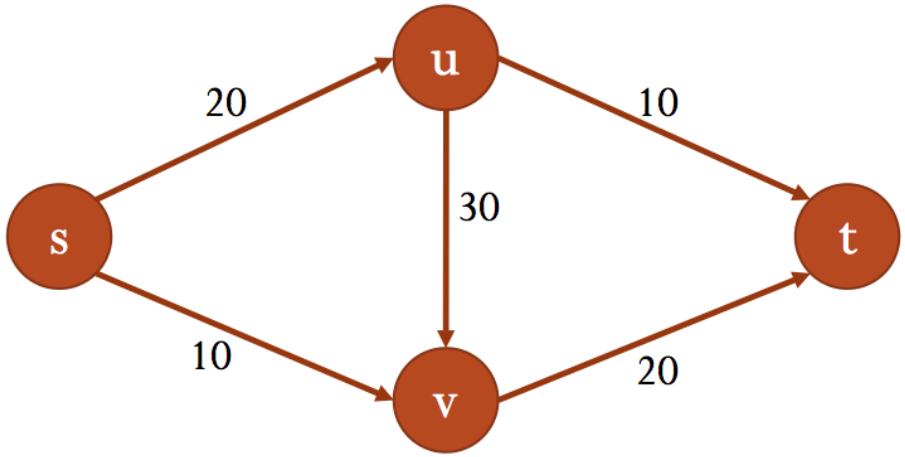
$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

Problem

Find an $s - t$ flow f in a given network graph such that the following quantity is maximized:

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

- Example:



Problem

Find an $s - t$ flow f in a given network graph such that the following quantity is maximized:

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

- Example:

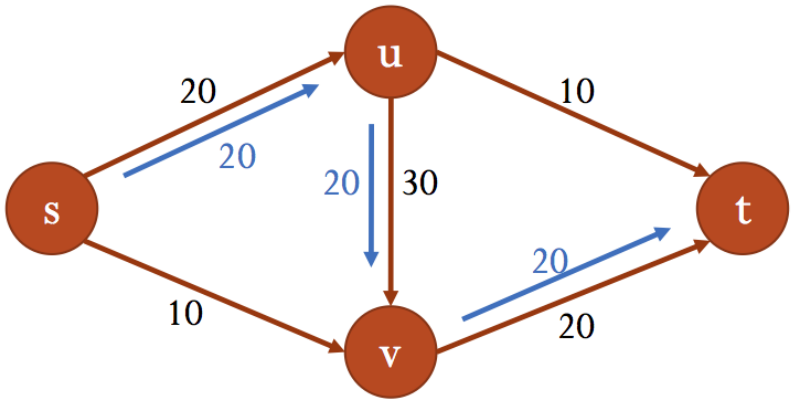


Figure: Routing 20 units of flow from s to t. Is it possible to "push more flow"?

End