COMP3121/9101: Greedy Algorithms

Module 3 Lecture Notes

Term 2, 2025

1 Overview

- **Motivation**: Introduction to greedy algorithms and their applications.
- **Optimal Selection**: Problems like activity selection and cell tower placement.
- **Optimal Ordering**: Problems like tape storage and minimizing job lateness.
- **Optimal Merging**: Huffman coding for efficient data compression.
- **Applications to Graphs**: Shortest paths, minimum spanning trees, and directed graph structures.
- **Puzzle**: A problem involving secure transmission using padlocks and boxes.

2 Greedy Algorithms

2.1 Definition

A greedy algorithm solves a problem by making locally optimal choices at each stage, hoping to find a global optimum. However, these choices may not always lead to the best overall solution.

2.2 When to Use

- Problems where local optima lead to global optima (e.g., fractional knapsack).
- Problems with optimal substructure (e.g., shortest paths).

2.3 Proof Techniques

- **Greedy Stays Ahead**: Show that the greedy choice is never worse than any other choice at each step.
- **Exchange Argument**: Transform any alternative solution into the greedy solution without worsening the objective.

3 Key Problems and Solutions

3.1 Fractional Knapsack

- **Problem**: Select items with weights w_i and values v_i to maximize total value without exceeding weight limit W. Items can be split.
- **Solution**: Sort items by $\frac{v_i}{w_i}$ in descending order and take as much as possible of the highest ratio item first.
- **Time Complexity**: $O(n \log n)$ due to sorting.

3.2 0-1 Knapsack

- **Problem**: Similar to fractional knapsack, but items cannot be split.
- **Challenge**: Greedy approach fails; dynamic programming is required.
- **Complexity**: NP-hard.

3.3 Activity Selection

- **Problem**: Select the maximum number of non-overlapping activities with given start and finish times.
- **Solution**: Sort activities by finish time and greedily select the earliest finishing activity that doesn't conflict with the last selected activity.
- **Proof**: Exchange argument shows optimality.
- **Time Complexity**: $O(n \log n)$ for sorting.

3.4 Cell Tower Placement

- **Problem**: Place the minimum number of cell towers to cover all houses along a straight road, each with a 5km range.
- **Solution**: Place towers 5km east of the leftmost uncovered house iteratively.
- **Proof**: Greedy stays ahead shows optimality.
- **Time Complexity**: $O(n)$ if houses are sorted; $O(n \log n)$ otherwise.

3.5 Tape Storage

- **Problem**: Order files on a tape to minimize average retrieval time.
- **Solution**: Sort files by increasing length (or by $\frac{p_i}{L_i}$ if probabilities differ).
- **Proof**: Adjacent inversion argument.
- **Time Complexity**: $O(n \log n)$ for sorting.

3.6 Minimizing Job Lateness

- **Problem**: Schedule jobs to minimize maximum lateness, where lateness is the difference between finish time and deadline.
- **Solution**: Sort jobs by deadlines (earliest deadline first).
- **Proof**: Exchange argument resolves adjacent inversions.
- **Time Complexity**: $O(n \log n)$ for sorting.

3.7 Huffman Coding

- **Problem**: Construct a prefix code to minimize expected message length given symbol frequencies.
- **Solution**: Build a binary tree by repeatedly combining the two least frequent symbols.
- **Proof**: Optimality follows from the properties of prefix codes and greedy choices.
- **Time Complexity**: $O(n \log n)$ using a priority queue.

4 Applications to Graphs

4.1 Strongly Connected Components (SCCs)

- **Problem**: Find SCCs in a directed graph where every vertex is reachable from every other vertex in the same component.
- **Solution**: Use Kosaraju's or Tarjan's algorithm (DFS-based).
- **Time Complexity**: $O(|V| + |E|)$.

4.2 Dijkstra's Algorithm

- **Problem**: Find shortest paths from a single source in a graph with non-negative edge weights.
- **Solution**: Greedily select the closest vertex and relax its outgoing edges.
- **Proof**: Correctness relies on non-negative weights and greedy choices.
- **Time Complexity**: $O((n+m)\log n)$ with a priority queue.

4.3 Kruskal's Algorithm

- **Problem**: Find a minimum spanning tree (MST) in a weighted graph.
- **Solution**: Sort edges by weight and add them if they don't form a cycle (using Union-Find).
- **Proof**: Correctness follows from the cut property.
- **Time Complexity**: $O(m \log n)$ for sorting and Union-Find operations.

5 Puzzle: Secure Transmission

5.1 Problem

Bob wants to send a teddy bear to Alice using locked boxes and padlocks. Constraints:

- Non-empty boxes must be locked during transit.
- Keys cannot be sent; they must remain with the sender.
- Padlocks can only be sent if locked.

5.2 Solutions

- **AND Solution** (3 mailings):
  1. Bob locks the box with his padlock and sends it to Alice.
  2. Alice adds her padlock and sends it back to Bob.
  3. Bob removes his padlock and sends the box back to Alice.
- **OR Solution** (2 mailings):
  1. Bob sends an empty box with his padlock to Alice.
  2. Alice places the teddy bear in the box, locks it with her padlock, and sends it back to Bob.