Algorithms Course Project Report

John Wilkins

CSC 513 Algorithms

Dr. Bo Li

December 3rd, 2024

Report

Abstract

This project implements a WordNet semantic relationship analyzer that finds relationships between words using graph algorithms. The implementation includes a directed graph structure to represent word relationships, shortest ancestral path (SAP) calculations, and an outcast detection system. The project demonstrates practical applications of graph theory in natural language processing and semantic analysis.

Problem Description

The project addresses three main computational challenges in semantic relationship analysis:

1. Building and representing a WordNet as a directed graph structure, where:
   - Nodes (synsets) represent sets of synonymous words
   - Edges represent hypernym relationships (IS-A relationships)
   - The graph must be a rooted directed acyclic graph (DAG)

2. Finding the shortest ancestral path between word pairs, which involves:
   - Calculating the shortest common ancestor of two vertices in the graph
   - Computing the minimum total path length to this common ancestor
   - Handling multiple starting points when words appear in multiple synsets

3. Identifying semantic outcasts from word groups by:

- Computing semantic distances between all word pairs
- Finding the word with the greatest cumulative semantic distance from other words

Algorithm (Pseudocode)

1. Shortest Ancestral Path (SAP)

```
Function SAP(v, w):
        Initialize minDistance = INFINITY
        Initialize ancestor = -1

        Create BFS1 from vertex v
        Create BFS2 from vertex w

        For each vertex i in graph:
        If BFS1 has path to i AND BFS2 has path to i:
        distance = BFS1.distTo(i) + BFS2.distTo(i)
        If distance < minDistance:
                minDistance = distance
                ancestor = i

        Return [ancestor, minDistance]
```

2. Outcast Detection

```
Function findOutcast(nouns[]):
        Initialize maxDistance = MIN_VALUE
        Initialize outcast = null

        For each noun1 in nouns:
        For each noun2 in nouns:
        distance = wordnet.distance(noun1, noun2)
        If distance > maxDistance:
                maxDistance = distance
                outcast = noun2

        Return outcast
```

Implementation Details

The implementation consists of three main classes:

1. WordNet.java
   - Maintains two key data structures:
      - HashMap<String, List<Integer>> for mapping words to synset IDs
      - HashMap<Integer, String> for mapping synset IDs to their definitions
   - Validates graph properties (DAG, single root)
   - Provides methods for semantic distance calculations

2. SAP.java
   - Implements breadth-first search for finding shortest ancestral paths
   - Uses edu.princeton.cs.algs4.BreadthFirstDirectedPaths for efficient path finding
   - Handles both single vertex and multiple vertex inputs
   - Returns both the common ancestor and the path length

3. Outcast.java
   - Implements semantic distance-based outcast detection
   - Uses the WordNet distance calculations to find semantic relationships
   - Returns the word with maximum cumulative distance from others

Key Implementation Decisions:

- Used immutable Digraph copies to ensure thread safety

- Implemented null checks and input validation throughout
- Optimized memory usage by sharing graph structure between instances
- Used efficient data structures for quick lookups and traversals

Running Results and Analysis

The implementation successfully handles:

- Word relationship queries with accurate semantic distance calculations
- Multiple word senses (polysemy) through synset management
- Complex graph traversals for finding common ancestors
- Outcast detection in word groups

Performance Characteristics:

- Space Complexity: O(V + E) for graph storage
- Time Complexity:
  - SAP calculations: O(V + E) per query
  - Outcast detection: O(n^2 * (V + E)) where n is the number of input words

Conclusions (What I learned)

This project provided valuable insights into:

1. Graph Theory Applications:
   - Practical implementation of directed acyclic graphs

- Real-world applications of graph algorithms in linguistics
- Importance of graph properties validation

2. Algorithm Design:

- Efficient implementation of breadth-first search
- Optimization techniques for graph traversal
- Handling multiple path scenarios

3. Software Engineering Practices:

- Importance of robust input validation
- Benefits of immutable data structures
- Value of clear API design
- Effective use of Java collections framework

4. Performance Considerations:

- Trade-offs between time and space complexity
- Importance of choosing appropriate data structures
- Impact of algorithm choice on scalability

The project successfully demonstrated how graph theory concepts can be applied to solve real-world natural language processing problems while maintaining efficient performance characteristics.

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms (4th ed.). MIT Press.