Computational Complexity
Assignment 2

Luis José Quintana Bolaño

May 16, 2014

Abstract

This document examines the design and implementation of a naïve algorithm for the Stainer Tree problem (ST). It provides the worst-case time analysis, the design of a parametric graph for experimentation and observations of the experiments conducted on this graph for 0 to 20 optional graphs.

1 Introduction

In this assignment we are required to propose a solution for the Steiner Tree problem.

The Steiner tree problem is very similar to the minimum spanner tree, in the sense that both are required to find the network (graph) of shortest length (i.e. the sum of the length of all edges) that interconects all given vertex, with the difference being that, in the Steiner tree problem, optional vertices and edges may be added to the graph in order to reduce the length of the spanning tree. It's been proven that the solution is always a tree, and several trees may solve the same given set of initial vertices.

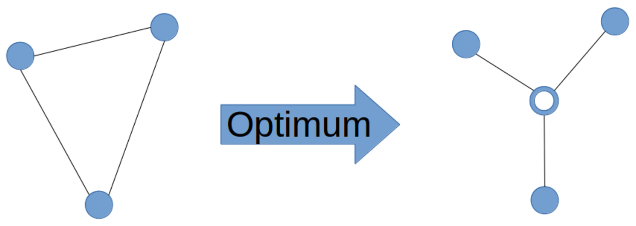

Figure 1: Example of the reduction of the spanning tree with three initial vertices by adding an optional one.

2 Implementation

As required in the assignment, the Steiner tree problem solver has been implemented in C++.

Since the problem is a variation on the minimum spanning tree problem, Prim's algorithm has been used as a base, implemented according to the pseudocode provided witht the documentation for the assignment.

In order to find the Steiner tree, the computation of Prim with every single possible combination of optional vertices is necessary, to find the shortest one. For the generation of the k-combinations, a recursive solution has been proposed, in wich the recursive function includes one of the optional vertices in the graph (received as a parameter), then computes Prim and calls itself with the current graph and the next vertex to be included for all optional vertex remaining, returning the minimum value between it's own computed Prim and those returned by the recursive calls. This process continues until all possible combinations have been tested, thus obtaining the minimum spanning tree amongst all posible graphs (i.e. the Steiner tree).

```
Data: Graph $G$, Vertex $V$
Result: minimum ST found by Prim's algorithm
Include vertex $V$ in graph $G$;
Set $Prim(G)$ as current best result ;
for every optional Vertex after $V$, $O_i$ do
    if current best > recursive call(G, $O_i$) then
        Set recursive call's result as current best;
    end
end
return current best;
```

Algorithm 1: Recursive Steiner function

3 Analysis

3.1 Worst-case time

As the algorithm tries Prim on every possible combination of optional vertices, for n optional vertices it must do 2^n iterations of Prim (which itself belongs to $O(m^2)$ with m being the number of vertices on the graph), and the same amount of iterations of the cost function, which runs at a time cost of $O(m)$.
Since we're studying the worst-case time, we can asume all vertex to be optional, thus having a total cost of $2^n * (n^2 + n)$. We can now conclude that our algorithm runs in $O(2^n * n^2 + 2^n * n)$, and, according to the rule of the maximum, that is $O(2^n * n^2)$

3.2 Experiments

In this section we proceed to run the program with a graph created for this pourpose, using from 0 to 20 optional vertices, to compare the empirical time results.

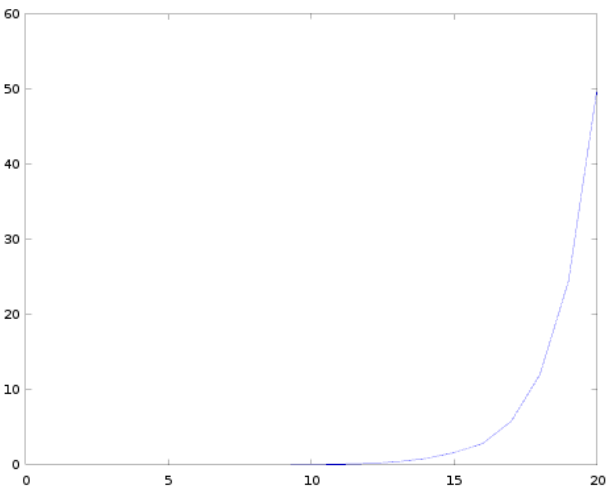

Figure 2: Plot of the time vs. number of optional vertices.

3.3 Observations

As we can see int he previous plot, the time obviously increases exponentially. Even if the implementation used in this document is not the most effective one known, this should serve as an example of how inefficient this problem is to solve.