New approximation algorithm for Steiner forest

1Seyed Farid Seyfosadat and 2Ali Moeini

1Islamic Azad University south Tehran Branch, Ahang street, Abouzar street, Piroozi street, Tehran, Iran
2University of Tehran, Enghelab Street, Tehran, Iran

ABSTRACT

As we know NP problems have no optimized solution and solving these problems with super computers can take over 300 years! Because of their huge range of usability of these problems introduce an optimized solution can change industry, science and our life. Steiner forest problem have huge usability in designing VLSI circuit, network routing and Phylogenetic trees. In this paper we review Steiner forest problem and introduce an optimized solution for it. This solution use approximation algorithms. For this purpose we review existing solutions and their objections and then presenting new approximation algorithm that can cover these objections.

INTRODUCTION

In computer science problems divided into two categories: solvable and unsolvable. When we can solve a problem with an algorithm we say that problem is solvable and otherwise that is unsolvable. Both of these definitions based on Turing machine. If one problem is solvable, our solution can be efficient or inefficient. If our solution has polynomial time \( T(n) = O(n^2) \) this is an efficient solution and if it has exponential time \( T(n) = O(n^a) \) we say this is inefficient solution.

We have various calculation complexity classes, which means classifying problems based on calculation complexity, for running time and memory. The best known calculation complexities for running time are \( P \), \( NP \), \( NP-Complete \) and \( NP-Hard \) and for memory are \( NL \), \( L \), \( \ldots \). A problem is \( NP \) when verifying a sample solution can be done in polynomial time. A problem said \( NP-Complete \) when it has two conditions: first it’s \( NP \) and second we can reduce every other \( NP-complete \) with a reduction procedure to this problem.(Shirazi, 2008)

In 1972 Richard Crap using Stephan cook’s theory and proof 21 classic problems is \( NP-complete \). He starts with \( SAT \) problem and proves some problems are \( NP \). By using them proved some other problems are \( NP \) and \( \ldots \) (Krap,1972). One of these problems is Steiner Tree and Steiner forest.

Unfortunately most of important problems are in \( NP \) class. We can solve these problems with three approaches: first, if size of inputs is small we can solve problem with existing exponential algorithm. Second restriction problem for achieve a simple problem that can be solved in polynomial time. Third suffice to approximate solution instead of exact one.(Cormen)

Steiner forest: definition and classic solution:

Given an undirected graph \( G = (V,E) \), a cost function on edges \( c : E \rightarrow Q^+ \), and a collection of disjoint subsets of \( V \), \( S_1,S_2,\ldots,S_k \), find a minimum cost subgraph in which each pair of vertices belonging to the same set \( S_i \) is connected.

Let us restate the problem; this will also help generalize it later. Define a connectivity requirement function \( r \) that maps unordered pairs of vertices to \([0, 1]\) as follows:

\[
r(u, v) = \begin{cases} 
1 & \text{if } u \text{ and } v \text{ belong to the same set } S_i \\
0 & \text{otherwise}
\end{cases}
\]

Now, the problem is to find a minimum cost subgraph \( F \) that contains a \( u-v \) path for each pair \((u, v)\) with \( r(u, v) = 1 \). In general, the solution will be a forest. (Vazirani,2003)

Also we can define this problem like this:
In order to give an integer programming formulation for this problem, let us define a function on all cuts in: \( 2^V \rightarrow \{0,1\} \), which specifies the minimum number of edges that must cross each cut in any feasible solution.

\[
f(S) = \begin{cases} 1 & \text{if } \exists u \in S \text{ and } v \in \overline{S} \text{ such that } r(u, v) = 1 \\ 0 & \text{otherwise} \end{cases}
\]

Let us also introduce a 0/1 variable \( x_e \) for each edge \( e \in E \); \( x_e \) will be set to 1 iff \( e \) is picked in the subgraph.

The integer program is:

\[
\text{minimize } \sum_{e \in E} c_e x_e \\
\text{subject to } \sum_{e \in \delta(S)} x_e \geq f(S), \quad S \subseteq V
\]

Where \( \delta(S) \) denotes the set of edges crossing the cut \((S, \overline{S})\). (Lopez.A,2006)

We will introduce a new idea in the primal–dual schema for approximation algorithms, setting it apart from the way this schema is used for designing exact algorithms. The later algorithms work on demand in each iteration, we pick one unsatisfied complementary slackness condition, and satisfy it by modifying the primal and dual solutions suitably. The new idea is that of raising duals in a synchronized manner. The algorithm is not trying to rectify a specific condition. Instead, it tries many possibilities simultaneously, one of which leads to primal improvement.

Some figurative terminology will help describe the algorithm more easily. Let us say that edge \( e \) feels dual \( y_e > 0 \) if \( y_e > 0 \) and \( e \in \delta(S) \). Say that set \( S \) has been raised in a dual solution if \( y_e > 0 \). Clearly, raising \( S \) or \( \overline{S} \) has the same effect. Sometimes we will also say that we have raised the cut \((S, \overline{S})\). Further, there is no advantage in raising set \( S \) with \( f(S) = 0 \), since this does not contribute to the dual objective function. Thus, we may assume that such cuts are never raised. Say that edge \( e \) is tight if the total amount of dual it feels equals its cost. The dual program is trying to maximize the sum of the dual variables \( y_e \) subject to the condition that no edge feels more dual than its cost, i.e., no edge is overtight. (Huang, 2011)

Next, let us state the primal and relaxed dual complementary slackness conditions. The algorithm will pick edges integrally only. Define the degree of set \( S \) to be the number of picked edges crossing the cut \((S, \overline{S})\).

**Primal conditions:**

For each \( e \in E \), \( x_e \neq 0 \implies \sum_{e \in \delta(S)} y_e = c_e \\
\text{Equivalently, every picked edge must be tight.}

**Relaxed dual conditions:**

The following relaxation of the dual conditions would have led to a factor 2 algorithm: \( \forall S \subseteq V, y_e \neq 0 \implies \sum_{e \in \delta(S)} y_e \leq 2 \cdot f(S) \).

This algorithm has iterative procedure which different forests grown to achieve minimal subgraph. At any point, the picked edges form a forest. Say that set \( S \) is unsatisfied if \( f(S) = 1 \), but there is no picked edge crossing the cut \((S, \overline{S})\). Set \( S \) is said to be active if it is a minimal unsatisfied set in the current iteration.

**Lemma:**

Set \( S \) is active iff it is a connected component in the currently picked forest and \( f(S) = 1 \)

By the characterization of active sets given in this Lemma, it is easy to find all active sets in the current iteration. The dual variables of these sets are raised in a synchronized manner, until some edge goes tight. Any one of the newly tight edges is picked, and the current iteration terminates.

When a primal feasible solution is found, say \( F \), the edge augmentation step terminates. However, \( F \) may contain redundant edges, which need to be pruned for achieving the desired approximation factor. Formally, edge \( e \in F \) is said to be redundant if \( F - \{e\} \) is also a feasible solution. All redundant edges can be dropped simultaneously from \( F \). Equivalently, only non-redundant edges are retained.

This algorithm is presented below.

**Steiner Forest algorithm:**

1. **(Initialization)** \( F \leftarrow \emptyset ; \) for each \( S \subseteq V, y_e \leftarrow 0 \).
2. **(Edge augmentation)** while there exists an unsatisfied set do:
   - Simultaneously raise \( y_e \) for each active set \( S \), until some edge \( e \) goes tight;
   - \( F \leftarrow F \cup \{e\} \).
3. **(Pruning)** return \( F = \{ e \in F | F - \{e\} \text{ is primal infeasible} \} \).
Some of the issues of this algorithm:

1- Probably the most important issue of this algorithm is using factor 2 in relaxation step. As mentioned, there is no any scientific reason for that so we can increase or decrease this factor and solving problem. (Panigrahi, 2007)

2- the second issue is simultaneously grown of forest. For implementing that we need distributed environment or using parallel programming. Both solution needs hardware and software facilities for implementing distributed system or parallel running of this algorithm on one data structure.

3- that is difficult to define a data structure that shows grown of forest and can store dual variable, edge’s weight and label of vertices.

4- that is difficult to keep forest consistence during grown of it

5- last issue is the large number of iteration and dependency of these iteration to each other. In large number of iteration we must have large storage to keep these. Also size of this storage depends on which cut of forest processed now.

New approximation algorithm for Steiner forest:

Improved Steiner Forest algorithm

\[ C \leftarrow \emptyset \]

While \( C \neq V \) do

Find minimum edge between \( u \) and \( v \) and remove this edge from \( E \)

\( S = \{u, v\} \)

if(\( f(S) == 1 \))\{

\[ \alpha = \frac{\text{Cost}(S)}{|S-C|} \text{Cost-effectiveness of selected edge} \]

for each \( v \in S - C \) set \( \text{price}(v) = \alpha \)

\( C = C \cup S \)

return \( C \)

Time complexity of this algorithm is \( O(V^2) \)

Calculate approximation of this algorithm:

Assume that we choose \( k \) element from set \( V \) with this algorithm and choose remaining with optimized algorithm. Maximum cost of covering remaining element is \( \text{OPT} \), so there is one element from remaining vertices that its cost-effectiveness is \( \frac{\text{OPT}}{|C|} \). Because we covered \( k \) element, number of remaining element was \( n-k+1 \). Also \( v_k \) selected earlier then it is more cost-effectiveness than other. So:

\[ \text{price}(v_k) \leq \frac{\text{OPT}}{|C|} \leq \frac{\text{OPT}}{n-k+1} \]

Summation of cost-effectiveness is cost of algorithm. So:

\[ \sum_{k=1}^{n} \text{price}(v_k) = (1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}) \cdot \text{OPT} \]

If \( 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \) named \( H(n) \) then the approximation of this algorithm is \( H(n) \) . so the approximation of this algorithm depend on number of inputs.

Benefits of our algorithm:

1- unlike previous algorithm, it has an exact approximation.

2- it is centralized algorithm and no deal with previous algorithm’s challenge due to its distributed sprite.

3- unlike previous algorithm, in isn’t iterative and no need to huge storage.

REFERENCES


Lopez, A., 2006. Steiner Trees and Forests, Massachusetts Institute of Technology, USA.
