Parallel Greedy Adaptive Search Algorithm for Steiner Tree Problem

Rashid Bin Muhammad
Department of Computer Science
Kent State University
Kent, OH, USA
e-mail: rmuhamma@cs.kent.edu

Abstract - In this paper, a parallel algorithm for the Steiner tree problem is presented. The algorithm is based on the well-known multi-start paradigm the GRASP and the well-known proximity structures from computational geometry. The main contribution of this paper is the \(O(n^2 \log^2 n + \log n \log(\frac{n \log n}{n}))\) parallel algorithm for computing Steiner tree on the Euclidean plane. The parallel algorithm used proximity structures from computation geometry like, Voronoi diagram, Delaunay triangulation, Gabriel graph, and relative neighborhood graphs to compute additional or Steiner points.

Keywords: Steiner tree, parallel algorithm, local search, and proximity structures.

1 Introduction

Most decision problems in mathematics and computer science may be formulated as optimization problems. These problems arise in situation where discrete choices must be made, and solving them amounts to finding an optimal solution among finite number of alternatives [1]. With these problems the difficulty are twofold. Not only these are too difficult to solve in polynomial time, the quality of a solution is not very attractive. In this situation heuristics becomes the methods of choice. For example, in case of Steiner tree problem computing a feasible solution is not satisfactory, but the quality of solution is also very important.

Several strategies have been proposed and applied to different optimization problems. Among those, local search is very popular mainly because it gives rise to algorithms that are quite simple and robust. Simplicity and robustness come from the facts that these algorithms use some kind of greediness or priority based procedures. In general, a local search algorithm works as follows. It starts with an initial (current) solution and repeatedly replaces current solution with a better one until no better solution can be found in its neighborhood. The main drawback of these techniques is that their search operations trapped in a local optimum. This is a clear indication that we need a technique for guiding these algorithms out of this “trap”. Following this theme, this paper explores the possibility of extracting useful information from unsuccessful searches to improve the time complexity and quality of the final solution.

In the light of above arguments and with the explosion of parallel computers, parallel implementations of local search heuristics appear quite naturally as a promising alternative to speedup the search for approximate solutions and to reach a better solution. Even though parallelism is not yet systematically used to speedup local search algorithms, parallel implementations are plentiful in the literature. A greedy randomized adaptive search procedure (GRASP) [2], [3] is a well-known multi-start paradigm. Most parallel implementation of GRASP is independent-thread multiple-walk strategies and based on the distribution of the iterations over the processors. Martins et al. [4] implemented a parallel GRASP for the Steiner problem in graphs and is based on the distance network [5]. The same approach was used in [6] for a preliminary parallel implementation of the GRASP for the Steiner problem in graphs. Canuto et al. [7], [8] used path-relinking as a post-optimization step to implement a multi-walk parallel GRASP algorithm for the prize-collecting Steiner tree problem. For local searches, they implement an independent multiple-walk strategy in parallel. Handa et al. [9] proposed local search technique, so called neighborhood composition, and used it to reduce the processing time of the traveling salesperson problem. The neighborhood composition can be described as follows. Suppose a local search starts with an initial solution \(s_0\) and tries to obtain an improved solution. Search ‘i’ suggests a substitution of a current solution \(s_0 \in S\) with an \(s\) to achieve a improved solution \(s'\). Since the size of an instance is huge and search ‘i’ can explore only a restricted neighborhood of \(S\) in a reasonable amount of time, for many searches subsets (solutions spaces) do not overlap. Obviously, \(s'\) can be further improve by replacing \(s'\) with \(s''\) using information in ignored searches for
other neighborhood. The basic idea of this technique is to extract useful information from unsuccessful local searches to speedup the parallel local search algorithm for traveling salesperson problem.

While the fundamental approach is based on the neighborhood composition [9] and proximity structure [11], our proposed algorithm diverges in a significant aspect: the way algorithm finds the additional neighborhood composition [9] and proximity structures to speedup the parallel local search algorithm points called Steiner points in parallel by first dividing the neighborhood structure into geometrical subgraphs. Furthermore, the implementation details are very different due to the fact that the Steiner tree problem requires to include Steiner points. Zachariasen et al. proposed sequential-local-search algorithm for Steiner tree problem in [11]. The approach is based on a listing of full Steiner trees, FSTs, in preprocessing phase. The list of full Steiner trees is solution space. For neighborhood structure, they used the well-known structures from computational geometry, such as Voronoi diagram, Delaunay triangulation, Gabriel graph, relative neighborhood graph, etc. In a straightforward parallel implementation of local search algorithm for Steiner tree problem, the given set terminals is divided into several subsets (subsolution spaces), which are distribute over the processors. Each processor performs a sequential search for better solution, defined by the conditioning the terminals appearing in the solution, in its neighborhood. Before the next local search starts, acknowledge the result of one search (that looks promising at that time) and ignore the results of all other searches despite of the fact that they would contain useful information for further searches. Our experiments showed that this kind of straight forward parallel implementation will trapped the local searches into a local minimum and severely damaged the quality of the final solution.

The main contribution of this paper is a proficient and implementable algorithm based on local search algorithm for Euclidean Steiner tree problem. This work parallelize the well-known GRASP paradigm and efficiently combines the sequential approach based on computational geometry structure [11] with the concept of neighborhood composition [9] to speedup the processing time of Euclidean Steiner tree problem.

In this paper, we study parallel local search approach to the Steiner tree problem. The general outline of the paper is as follows. Some basic definitions and problem definitions are given in Section 2. Section 3 deals with local search and Steiner tree problem. Section 4 presents the overall strategy of parallel local search algorithm for Steiner tree problem in Euclidean space. Section 5 presents the algorithm for Steiner tree problem based on client-server paradigm. Finally, some concluding remarks are collected in Section 7.

2 Preliminaries

This section presents well-known closest-point or proximity structures from computational geometry.

Let \( P = \{p_1, p_2, \ldots, p_n\} \) be the set of points in the Euclidean plane. Also, let \( H(p_1, p_2) \) be the half-plane and defined as the set of points equidistant to \( p_1 \) and \( p_2 \). Then \( V(p_i, P) = \bigcap_{p_j \in P, p_j \neq p_i} H(p_j, p_i) \) be the Voronoi region of \( p_i \). \( V(p_i, P) \) is convex and its interior is the locus of points closer to \( p_i \) than to any other point. Therefore, \( V(p_i, P) = \{q \in E^2 : |p_i, q| \geq |p_j, q|, \forall p_j \in P \setminus \{p_i\}\} \). Let \( P(p_i, P) \) denote the boundary of \( V(p_i, P) \). The union of these boundaries for all points in \( P \) forms the Voronoi diagram for \( P \), denoted by \( VD(P) \). The \( k-th \) order Voronoi diagram \( VD_k(P) \) is a partition of the plane into regions \( V(P_k, P) \), where \( P_k \subseteq P \) and \( |P_k| = k \). The interior of \( V(P_k, P) \) is the locus of points closer to every point in \( P_k \) than to any point in \( P \setminus P_k \).

Therefore, \( V(P_k, P) = \{q \in E^2 : |p_i, q| \leq |p_j, q|, \forall p_j \in P_k, \forall p_j \in P \setminus P_k\} \). Note that \( VD_1(P) = VD(P) \). The Voronoi diagrams of all orders up to \( K-th \) order can be determined in \( O(Kn \log n) \) time [12].

The straight-line dual of the Voronoi diagram for \( P \) is a triangulation of \( P \), known as Delaunay triangulation and denoted by \( DT(P) \). Also, defined as the unique triangulation in which the circumsphere of each triangle does not contain any other point in its interior. Delaunay Triangulations can be computed in \( \Theta(n \log n) \) time and contains at least one minimum spanning tree (MST) for the given set \( P \). The MST for \( P \) can be computed in \( \Theta(n) \) time once Delaunay triangulation of \( P \) is given [13].

The Gabriel graph, denoted by \( GG(P) \), of \( P \) is defined by the graph in which \( p_i, p_j \) is an edge of \( GG(P) \) if and only if the circle having \( p_i, p_j \) as a diameter is an empty circle, that is, if an only if it contains no point of \( P \) in its interior. \( GG(P) \) can be computed in \( \Theta(n \log n) \) time by removing from \( DT(P) \) edges not intersecting their Voronoi edges. Note that \( GG(P) \) is a subgraph of \( DT(P) \) and contains at least on MST for \( P \).

The relative neighborhood graph, denoted by \( RNG(P) \), is defined as a geometric graph in which \( RNG(P) \) has an edge between \( p_i \) and \( p_j \) if and only if \( d(p_i, p_j) = \min_{k \neq i, j} \max(d(p_i, p_k), d(p_j, p_k)). \) \( RNG(P) \) can be computed in \( \Theta(n \log n) \) time. Furthermore, \( RNG(P) \) is a subgraph of \( DT(P) \) and contains at least one MST for \( P \).

3 Steiner Tree Problem and Local Search

A Steiner minimal tree (SMT) for a given set \( P \) of points in the Euclidean plane is the shortest interconecting \( P \). Any intersections of edges which are not in \( P \) are called Steiner points. It is well-known [14]
that each Steiner point is of degree three and any two edges in an \textit{SMT} intersect at an angle with at least degree 120. An interconnecting tree satisfying the above two conditions is called a Steiner tree. It is also well-known [14] that a Steiner tree for \( n \) given points can have at most \( n-2 \) Steiner points. A Steiner tree is full if it has \( n-2 \) Steiner points and denoted by FST. The use of a local search algorithm presupposes definitions of a problem and a neighborhood. The \textit{FST}. The use of a local search algorithm presupposes definitions of a problem and a neighborhood. The Euclidean Steiner tree problem is given by a finite set \( F \) of full Steiner trees \( f_1, f_2, \ldots, f_m \), where each \( f_i \) in \( F \) has cost \( \text{cost}(i) \). The objective is to find a solution \( f'_i \) with minimum cost. In general, local search algorithm for \textit{SMT} work as follows. A local search starts from an initial solution \( s \), which is usually a \textit{MST} and repeatedly replace with a better solution in its neighborhood \( N(s) \), where \( N(s) \) is a set of solutions obtainable by slight perturbations. This can be formalized as follow.

Step 1. For a given set \( P \) of \( n \) points, set initial solution \( s \leftarrow \text{MST}(n) \).

Step 2. Search for improved solution \( s' \) in \( N(s) \) such that \( \text{cost}(s')<\text{cost}(s) \).

Step 3. IF such a solution \( s' \) exits Then improve the solution by setting \( s \leftarrow s' \) and GOTO Step 2 otherwise GOTO Step 4

Step 4. Output resultant tree \( s \) and halt.

Clearly, the performance of Steiner local search algorithm depends upon the choice of neighborhood. Choosing a larger neighborhood may give the better solutions but increases the time complexity. On the other hand, if we choose small neighborhood, we might missed the good, possibly optimal, solution. In this paper, we consider a local search algorithms for \textit{SMT} whose neighborhood is defined by proximity structures from computational geometry that is, Voronoi diagram (\textit{VD}), Delaunay triangulation (\textit{DT}), Gabriel graph (\textit{GG}), and relative nearest neighborhood graph (\textit{RNG}) [11].

4 Overall Strategy

The proposed parallel Steiner local search algorithm is based on [9] and [11] and used the server-client paradigm (generally known as master-slave paradigm). It has two phases, namely search phase and upgrade phase. These two phases accomplished by server and client processors. Server processor does the upgrading in the solution while searches are accomplished by client processors. The algorithm is based on the decomposition of the solution space. The server processor reduces the \textit{ESTP} to a selection problem by constructing a list of full Steiner trees \( F = \{f_1, f_2, \ldots, f_m\} \), where \( m = O(n) \) is the number of FSTs [11]. The server processor divides the \textit{ESTP} problem by generating subsets of terminals by enumerating all connected subgraphs of proximity structure (for example, \textit{RNG}) up to \( k = 5 \) terminals. Then, it generates lists of full Steiner trees (\textit{FSTs}) associate with each subgraph. The subgraphs and lists of \textit{FSTs} associated with each subgraph are distributed over the client-processors.

In a local search phase, a local minimum in the neighborhood of the constructed solution is sought. Each client-processor searches a solution space i.e., list of \textit{FSTs} on a different subgraph, defined by condition the vertices appearing in the solution. The best solution over all searches is kept as the result by client processor. When a client finds a better solution in its neighborhood, it computes the \textit{enhanced} solution from its current solution and newly calculated one, and sends the \textit{enhancement} (not the best solution) to the server processor. We define \textit{enhancement} as the difference between the original solution and the best solution so far. After the \textit{enhancement} has been received, the server tries to apply the \textit{enhancement} to its current solution. If this tryout is successful that is, it actually improves the overall solution, the server sends a new copy of its solution to the client. Otherwise, server processor updates the segment of the solution spaces assigned to the client processor. Since the current solution of the client may be completely different from the server’s current solution.

The advantage of this method is that it has no communication overhead (theoretically) since it does not require synchronization of client processors. That is, each client independently searches its neighborhood while other clients communicate with server.

5 Parallel Algorithm

This section presents the parallel algorithm, based on [11] and [9], for Euclidean Steiner tree problem. For the ease of presentation, we consider only relative neighborhood graph in the following algorithm.

Algorithm Server (\textbf{UPGRADE})

1) For given set \( P \) of \( n \) points, construct \textit{RNG}(\( P \)).
2) Set Server solution to \textit{MST}(\( P \)).
3) Generate subsets of terminals by enumerating all connected subgraphs of \textit{RNG}(\( P \)) up to \( k = 5 \) terminals i.e., \textit{RNG}_i(\( P_k \)) for \( 1 \leq i \leq n \) and \( 1 \leq k \leq 5 \).
4) Generate full topologies for each \textit{RNG}_i(\( P_k \)).
5) Send server solution, \( \sigma_{\text{server}} = \text{MST}(\( P \)) \) and \textit{RNG}_i(\( P_k \)) to each client[i] i.e. client[i] gets \textit{MST}(\( P \)) and \textit{RNG}_i(\( P_k \)), where \( i \) is the number of clients.
6) \textbf{WAIT}.
7) Enhancement in the \( \sigma_{\text{server}} \) solution received from client[i], \( 1 \leq i \leq n \).
8) Check if Enhanced solution $\sigma_{\text{client}}$ received from client[i] is consistent with current server solution, $\sigma_{\text{server}}$.
9) If "yes" improved current server solution, $\sigma_{\text{server}}$, based on the received client’s enhancement.
10) Generate subsets of terminals by enumerating subgraphs of $RNG_{\sigma_{\text{server}}}$.
11) Send the current server solution, $\sigma_{\text{server}}$, and new partitions (subgraphs) $RNG_i$ to the client[i] from which the enhancement is received and GOTO step 6.

Algorithm Client (SEARCH)
1) Receive the initial solution $\sigma_{\text{server}} = \text{MST}(P)$ and $RNG_i(P_k)$ where $1 \leq k \leq 5, 1 \leq i \leq n$ and set client solution $\sigma_{\text{client}} \leftarrow \text{MST}(P)$.
2) A solution $s' \in S$ has $s'_i = 1$, iff $F_i$ is selected, $1 \leq i \leq n$.
3) Compute the Enhancement and send enhancement $\sigma'$ to the server where $\text{Solution}_{\text{Enhanced}} = \text{Solution}_{\text{New}} - \text{Solution}_{\text{Old}}$.
4) WAIT.
5) When received a $\sigma_{\text{server}}$ and $RNG_i(P_k)$ for neighborhood $N_i$ from the server, replace $RNG_i(P_k)$ and set $\sigma_{\text{client}}[i] \leftarrow \sigma_{\text{server}}$.
6) GOTO step 2.

6 Complexity Analysis
The following analysis is based on the [10].

Lemma 1 The parallel time complexity of the algorithm is $O(n^2 \log^2 n + \log n \log (\frac{n}{\log n}))$.

Proof: From Lemma 1 of [10] we know the time complexity of evaluating the neighborhood is $O(n^2 \log n) + O(\log (\frac{n}{\log n}))$ and $O(n)$ be the neighborhood size for instance of size $n$. Let $t_p$ be the time required by an algorithm with $p$ processors (server + clients processors). Then the time required by the algorithm to evaluate neighborhood of size $O(n)$ using one processor is
\[
t_1 = O((\text{Neighborhood evaluation time}) \\
\times (\text{neighborhood size})) \\
= O(n^2 \log n + \log (\frac{n}{\log n}) \times n) \\
O(n^3 \log n + n \log (\frac{n}{\log n}))
\]
From Lemma 1 of [10], we know that the number of processors are $p = \frac{n}{\log n}$. Therefore,
\[
t_p = O((\text{Neighborhood evaluation time}) \\
\times (\text{neighborhood size})) \\
= O((n^2 \log n + \log (\frac{n}{\log n}) \times \log n) \\
O(n^2 \log^2 n + n \log n \log (\frac{n}{\log n}))
\]
Hence, the parallel time complexity of the algorithm is $O(n^2 \log^2 n + \log n \log (\frac{n}{\log n}))$.

7 Conclusion
In this paper, we presented a parallel algorithm that utilizes the local search technique for computing a Steiner tree in the two dimensional plane. We implemented the local search technique in parallel for Steiner tree problem that allows us to solve larger problem and improve the quality of the final solution. The main contribution of this work is the $O(n^2 \log^2 n + \log n \log (\frac{n}{\log n}))$ parallel local search algorithm for computing Steiner tree on the Euclidean plane. The main advantage of the algorithm is that it does not need synchronization.

References