Efficient Deployment of Base Stations in Wireless Communication Networks

Zimao Li[1], YingyingWang[1], and Maode Ma[2]
[1]College of Computer Science, South-Central University for Nationalities, China
[2] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore
Email: lizm@mail.scuec.edu.cn; 2279457429@qq.com; emdma@ntu.edu.sg

Abstract —In the design of wireless communication networks, we may have to interconnect *n* stations locating at given points in the plane such that the distance among each stations is as small as possible by introducing at most *k* extra stations subjective to a budget limit. In this paper, our goal is to determine the locations of the extra *k* stations interconnecting the existing *n* stations to minimize the longest distance among stations. This is so-called bottleneck Steiner tree problem, which also has applications in VLSI routing, WDM optical networks design and phylogenetic tree reconstruction. The problem has been proved to be NP-hard and cannot be approximated in the performance ratio 2 in polynomial time in both Euclidean and rectilinear plane and approximation algorithms in the best possible performance ratios presented for the problem in both planes. In this paper, we improve the time complexity of the approximation algorithms and conduct simulations to demonstrate the validness of our improvements.

Index Terms—Wireless communication networks, bottleneck Steiner tree, approximation algorithm, performance ratio

I. INTRODUCTION

Wireless communication networks have been applied in a variety of defense and civil domains. The efficiency and reliability of these applications rely on the availability and distances among base stations in wireless systems. In the design of a wireless communication network with a given number of base stations, generally these stations must be deployed at given locations, and subjective to the budget limit, only a limited number of additional stations can be introduced to ensure the information successfully delivered through the networks. Thus determining the locations of these extra stations becomes an important issue.

The problem can be formulated as the bottleneck Steiner tree problem, which is a variation of the traditional Steiner tree problem [1]. Unlike the traditional Steiner tree problem, which is to find a tree spanning the required points by introducing some additional points such that the length of the tree is minimized [2], [3], the bottleneck Steiner tree problem is to minimize the length of the longest edge in the tree. We call the required point terminals as the additional point Steiner points, and such a tree as a Steiner tree.

Since 1990s, along with the conquest of a series of famous conjectures, the traditional Steiner tree problem has attracted numerous scientists' considerable attentions and interests from both theoretical point of view and its applicability, which has been once a focus in the emerging theory of approximation algorithms. The problem is MAX-SNP hard even when the edge length is only 1 or 2 [2]. For the Steiner tree problem in the Euclidean plane, it is still NP-hard and there is a Polynomial-Time Approximation Scheme (PTAS) for the Euclidean Steiner trees [3]. It is not known whether the Euclidean Steiner tree problem is NP-complete, since membership to the complexity class NP is not known.

New applications of the Steiner tree problem in VLSI routing [4], wireless communications [5] and phylogenetic tree reconstruction in biology [6] have been extensively explored and studied. These applications have triggered the study of variations of the traditional Steiner tree problem. Algorithms for the two variations, the *bottleneck Steiner tree problem* [7]-[10] and the *Steiner tree problem with minimum number of Steiner points and bounded edge-length* [11], [12], have been fully investigated.

In this paper, we consider the bottleneck Steiner tree problem, which is defined as follows. Given a set $P=\{p_1, p_2, ..., p_n\}$of n terminals and a positive integer k, to find a Steiner tree with at most k Steiner points such that the length of the longest edges in the tree is minimized.

The solution of the problem can be applied in the design wireless sensor networks to prolong the lifetime of the wireless sensor networks by introducing additional sensors at proper locations to achieve the goal of minimizing the length of the longest edge in the network [13].

The problem is NP-hard. In [7], it has been proved that unless P=NP, the problem cannot be approximated in a polynomial time within performance ratios 2 and $\sqrt{3}$ in the rectilinear plane and the Euclidean plane, respectively. Moreover, an approximation algorithm with performance ratio 2 for both the rectilinear plane and the Euclidean plane has been generated. For the rectilinear plane, the performance ratio is the best possible, that is, the performance ratio is tight. For the Euclidean plane,

Manuscript received March 22, 2016; revised June 22, 2016.
This work was supported by the National Science Foundation of China (Projects 61103248 and 61379059)
Corresponding author email: lizm@mail.scuec.edu.cn.
doi:10.12720/jcm.11.6.609-615

©2016 Journal of Communications

however, the gap between the lower bound and upper bound 2 is still big. Based on the existence of a 3-restricted Steiner tree, we have presented a randomized polynomial approximation algorithm with performance ratio 1.866 for the Euclidean problem [8]. Later, a further improvement on the performance ratio to $\sqrt{3}$ has been reported in [14], which is so far the best possible approximation result.

Restricted versions of the bottleneck Steiner tree problem have been investigated by many researchers. S. Bae, C. Lee, and S. Choi have studied the Euclidean bottleneck Steiner tree problem when k is restricted to 1 or 2 to produce the exact solutions to this problem in [10]. Due to the restriction of the adjacency of Steiner points in [9], [15], [16], when requiring that no two Steiner points are adjacent, or only degree 2 Steiner points are adjacent, or only Steiner points of degree at least 3 are not allowed to be adjacent in the optimal Steiner tree, these versions of Euclidean bottleneck Steiner tree problem cannot be approximated within $\sqrt{2}$ in the polynomial time and the existence of approximation ratio has been proved.

In this paper, we first address the Euclidean bottleneck Steiner tree problem, on which, the best possible approximation is provided by a randomized approximation algorithm with performance ratio of $\sqrt{3}+\varepsilon$ and time complexity $\frac{1}{\varepsilon}\times poly(n, k)$, for any $\varepsilon > 0$. By an idea similar to the binary search, we can decrease the number of loops of the algorithm from $\frac{1}{\varepsilon}$ to $\log\frac{1}{\varepsilon}$ to improve the efficiency from pseudo-polynomial to polynomial. Then, we investigate the rectilinear bottleneck Steiner tree problem, on which, D.-Z Du and L. Wang have proved that the problem could not be approximated within ratio 2 in the polynomial time and provided a 2-approximation algorithm which runs in $O(n\log n + kn + k^2)$ time rather than as they claimed as $O(n\log n + k\log n)$ [7]. The performance ratio is the best possible and any improvement of the ratio will lead to P=NP. By introducing two advanced data structures, the binary heap and the Fibonacci heap, we can reduce the time complexity of their algorithm to $O(n\log n + k\log n)$ and amortized $O(n\log n + k\log n)$, respectively. To help understanding the algorithms, we first introduce a notion of the *k-restricted* Steiner tree. A *full component* of a Steiner tree is a subtree in which each terminal is a leaf and each internal node is a Steiner point. A Steiner tree for n terminals is a k-restricted Steiner tree if each full component spans at most k terminals.

The rest of the paper is organized as below. In Section II, we briefly introduce the approximation algorithm with performance ratio $\sqrt{3}$ and our improvement. Section III introduces the approximation algorithm with performance ratio 2 for rectilinear bottleneck Steiner problem. Section IV implements the algorithm by introducing two kind of heaps. Simulation results are shown in Section V and the concluding remark appears in Section VI.

II. $\sqrt{3}+\varepsilon$–Approximation for Euclidean and Improvements

D.-Z Du, L. Wang and B. Xu have shown the existence a performance ratio $\sqrt{3}$ by the existence of a 3-restricted Steiner tree with the same number of Steiner points as T such that the longest edge in the tree is at most $\sqrt{3}$ times of the optimum in [14]. Then, they have transformed the computing of an optimal 3-restricted Steiner tree into the computation of the minimum spanning tree problem for 3-hypergraphs in [17]. They have proposed a randomized approximation algorithm to find an optimal 3-restricted Steiner tree, which resulting an algorithm with performance ratio $\sqrt{3}+\varepsilon$ for any positive number ε. The algorithm works mainly based on the following lemma.

Lemma 1[14]**:** *Let T be an optimum Steiner tree for Euclidean bottleneck Steiner tree problem. Then, there is a 3-restricted Steiner tree with the same number of Steiner points as T such that the longest edge in the tree is at most* $\sqrt{3}$ *times of the optimum.*

A *hypergraph* $H = (V, F)$ is a generalization of a graph where the edge set F is an arbitrary family of subsets of vertex set V. A 3-*hypergraph* $H_3 = (V, F)$ is a hypergraph, each of whose edges has cardinality at most 3. A *weighted* 3-*hypergraph* $H_3 = (V, F; W)$ is a 3-hyperpgrah with each edge associated with a weight. A minimum spanning tree for a weighted 3-hypergraph $H_3 = (V, F; W)$ is a subgraph T of H_3 that is a tree containing every node in V with the least weight.

The following Lemma proves the existence of a randomized algorithm for computing a minimum spanning tree for a weighted 3-hypergraph [17].

Lemma 2[17]**:** *There exists a randomized algorithm for the minimum spanning tree problem for weighted* 3-*hypergraphs, with probability at least* 0.5*, running in poly(n,*w_{max}*) time, where n is the number of nodes in the hypergraph and* w_{max} *is the largest weight of edges in the hypergraph.*

A weighted 3-hypergraph $H_3=(V, F; W)$ can be constructed from the set P of terminals. Here $V = P$, and $F = \{(a, b)|a \in P \text{ and } b \in P\} \cup \{(a, b, c)| a \in P \text{ and } b \in P \text{ and } c \in P\}$. To obtain the weight of each edge in F, we need to know B, the length of the longest edges in an optimal solution. It is hard to find the exact value of B in an efficient way because of the hardness result of the problem. However, we can finds a value B' that is at most $(1+\varepsilon)B$ for any $\varepsilon>0$ in time[8].

Fig. 1 shows the algorithm presented by D.-Z Du, L. Wang and B. Xu.

Theorem 3[8]: *For any given ε, there exists a randomized algorithm that computes a Steiner tree with n terminals and k Steiner points with probability at least 0.5 such that the longest edge in the tree is at most* $\sqrt{3}+\varepsilon$ *times of the optimum, and the algorithm's running is* $\frac{1}{\varepsilon}\times poly(n, k)$.

In fact, by using a binary search strategy, we can decrease the number of loops in Step 2 from $\frac{1}{\varepsilon}$ to $\log\frac{1}{\varepsilon}$,

thus improve the time complexity of the Algorithm of Euclidean-Bottleneck-by-Du Wang and Xu.

The improved algorithm is presented in Fig. 2.

Theorem 4: *For any given positive number ε, there exists a randomized algorithm that computes a Steiner tree with n terminals and k Steiner points with probability at least 0.5 such that the longest edge in the tree is at most* $\sqrt{3}+\varepsilon$ *times of the optimum, and the algorithm's running is* $(\log\frac{1}{\varepsilon})\times poly(n, k)$.

Algorithm Approximation-Euclidean-Bottleneck-by-Du Wang and Xu

Input: a set *P* of *n* terminals in the rectilinear plane and an integer *k* and a positive number ε

Output: a 3-restricted Steiner tree T with at most k Steiner points.

1. Call the ratio-2 approximation algorithm for bottleneck Steiner tree problem in [7] and obtain a number *X* as the length of the longest edge.
2. **for** B = $\frac{X}{2}$, $\frac{X}{2}(1+\varepsilon)$, $\frac{X}{2}(1+2\varepsilon)$,..., $\frac{X}{2}(1+\varepsilon\times\lceil\frac{1}{\varepsilon}\rceil)$ **do**
   a) Construct a weighted hypergraph H_3=(*V*, *F*; *W*) according to the value of *B*.
   b) Call the randomized algorithm in [19] to compute a minimum spanning tree *T* for H_3 =(*V*, *F*; *W*);
   c) Consider the solution *T*' of the smallest B such that $w(T') \le k$.
3. Replace every edge *f* of the minimum spanning tree *T*' on H_3 (*V*, *F*; *W*) with a Steiner tree with *w*(*f*) Steiner points such that the maximum length of each edge in the tree is at most B and output the obtained tree.

Fig. 1. Original approximation algorithm for Euclidean bottleneck Steiner tree problem

Algorithm Faster Approximation for Euclidean-Bottleneck

Input: a set *P* of *n* terminals in the rectilinear plane and an integer *k* and a positive number ε

Output: a 3-restricted Steiner tree T with at most k Steiner points.

1. Call the ratio-2 approximation algorithm for bottleneck Steiner tree problem in [7] and obtain a number *X* as the length of the longest edge.
2. Initialize *low* ← 0 and *high* ← $\lceil\frac{1}{\varepsilon}\rceil$
3. **while** (*low* < *high*) **do**
   a) *mid*←(*low* + *high*)/2 and *B*← $\frac{X}{2}(1+mid\times\varepsilon)$
   b) Construct a weighted 3-hypergraph H_3(*V*,*F*; *W*) according to the value of *B*.
   c) Call the randomized algorithm in [17] to compute a minimum spanning tree *T* for H_3 =(*V*, *F*; *W*).
   d) Consider the solution *T*', **if** $w(T') > k$, **then** *low*←*mid*+1; **else** *high*←*mid*-1.
4. Replace every edge *f* of the minimum spanning tree *T*' on H_3 (*V*, *F*; *W*) with a Steiner tree with *w*(*f*) Steiner points such that the maximum length of each edge in the tree is at most B and output the obtained tree.

Fig. 2. Improved approximation algorithm for Euclidean bottleneck Steiner tree problem

III. 2–Approximation Algorithm

The existence of performance ratio 2 has been proved by constructing a Steinerized spanning tree under the triangle inequality property in the rectilinear plane. The algorithm first constructs a minimum spanning tree for the set of *n* terminals in *P*, then the degree-2 Steiner point to long edges in the minimum spanning tree has been repeatedly added. We call such a tree as a *steinerized spanning tree*. The approximation algorithm has been derived based on the following two lemmas.

Lemma 5[7]: *Given a set of n terminals P in the rectilinear plane, let T be an optimal tree for the rectilinear bottleneck Steiner tree problem. Then there exists a steinerized spanning tree T' for P with the same number of Steiner points as T such that the length of the longest edges in T' is at most 2 the optimum.*

Lemma 6[7]: *Let* $e_1,e_2,...,e_{n-1}$ *be all edges in a spanning tree T and* $e_1^*,e_2^*,...,e_{n-1}^*$ *be all edges in a minimum spanning tree* T^* *for the same set P of terminals. Suppose* $c(e_i) \le c(e_{i+1})$ *and* $c(e_i^*) \le c(e_{i+1}^*)$ *for all* $1\le i \le n-1$ *where c(e) denotes the length of edge e. Then,* $c(e_i^*) \le c(e_i)$ *for all* $1\le i \le n-1$.

It follows immediately from Lemma 1 and 2 that when we use the same number of Steiner points to steinerize a spanning tree and a minimum spanning tree, the result from the latter cannot produce the longest edge of length exceeding that from the former. That is, the optimal steinerized spanning tree can be found among steinerized minimum spanning trees. Since only degree-2 Steiner points are possibly adjacent, we only need to add *k* Steiner points to a minimum spanning tree in order to obtain an optimal steinerized spanning tree.

The idea is explained as follows: for each edge e_i= (*u*, *v*) in the minimum spanning tree, if we add l_i degree-2 Steiner points to it, the length of the longest edge in the resulting path from *u* to *v* has the minimum value $c(e_i)/(l_i+1)$, where $c(e_i)$ is the original length of edge e_i. This minimum value can be achieved when the l_i Steiner points divide e_i evenly. Denote $l(e_i) = c(e_i)/(l_i+1)$. At the beginning of the algorithm, $l(e_i) = c(e_i)$. Each time a degree-2 Steiner point is added to the edge e_i with the largest $l(\cdot)$ value. After e_i receives one more degree-2 Steiner point, l_i is updated by $l_i=l_i+1$ and $l(e_i)$ is updated by $c(e_i)/(l_i+1)$ and the position of all the degree-2 Steiner points in the edge e_i is re-organized by dividing e_i evenly.

Note that e_i is defined in the rectilinear plane. The process is repeated until k degree-2 Steiner points have been added.

Fig. 3 shows D.-Z Du and L. Wang's approximation algorithm with performance ratio 2 [7].

Algorithm Approximation-Rectilinear-Bottleneck-by-Du and Wang

Input: a set P of n terminals in the rectilinear plane and an integer k.

Output: a bottleneck Steiner tree T for P with at most k Steiner points

1. Compute a minimum spanning tree of P. Suppose $e_1, e_2, \ldots, e_{n-1}$ are all edges in it.
2. Compute $l(e_1)$ for each edge e_i.
3. Sort the edges in a non-increasing order of $l(\cdot)$.
4. Add a degree-2 Steiner point to $e_1=(u,v)$ with the largest $l(\cdot)$ value.
5. Update $l(e_1)$ for the selected edge in Step 2.
6. Re-organize the Steiner points in the path from u to v.
7. Re-set e_i's position in the ordering.
8. Repeat Steps 4-7 until k degree-2 Steiner points are added.

Fig. 3. Du and wang's 2-approximation algorithm for rectilinear bottleneck Steiner tree problem

Algorithm Faster-Approximation-for Rectilinear-Bottleneck

Input: A set P of n terminals in the rectilinear plane and an integer k.

Output: A bottleneck Steiner tree T for P with at most k Steiner points.

1. Compute a minimum spanning tree for P, suppose $e_1, e_2, \ldots, e_{n-1}$ are all edges in it.
2. Initialize $l(e_i) \leftarrow c(e_i)$ and $l_i \leftarrow 0$ for $1 \le i \le n-1$, where $c(e_i)$ denote the length of edge e_i, l_i denote the number of Steiner points added to edge e_i.
3. Bottom-up manner to construct a max-heap for $e_1, e_2, \ldots, e_{n-1}$ by keys $l(\cdot)$.
4. Find $e_i = (u, v)$ with the largest key $l(e_i)$ from the constructed max-heap.
5. Update l_i by l_i+1 and $l(e_i)$ by $c(e_i)/(l_i+1)$ in the max-heap. //add a Steiner point to e_i.
6. Repeat step 4 to 5 until k Steiner points are added, that is $\Sigma l_i = k$.
7. Organize the l_i Steiner points by evenly partition edge $e_i = (u, v)$ for $1 \le i \le n-1$.

Fig. 4. Improved 2-approximation algorithm for rectilinear bottleneck Steiner tree problem

The algorithm's time complexity is analyzed as below: The first step can be implemented in $O(n\log n)$ time as described in [19], [20]. Step 2 takes a linear time. Sorting in Step 3 takes $O(n\log n)$ time. In each loop of Step 4-7, Step 4 and 5 use a constant time to find the longest edge and update $l(\cdot)$, Step 6 uses the time linear to the number of Steiner points on that edge, and the step 7 of resetting e_i's position needs $O(n)$ time in the worst case. As Step 4-7 only loops k times, the entire time complexity of the algorithm is $O(n\log n+kn+k^2)$.

IV. The Faster 2–Approximation Algorithms

The most time consuming steps in the loop of the Du and Wang's algorithm is Step 6 and 7, either linear to number of Steiner points or to the number of terminals in the worst case. First, we find that moving step 6 in Figure 1 out of the loop as the final step can decrease the time of organization of Steiner points from $O(k^2)$ to $O(k)$. Then, the step to find an edge with the largest $l(\cdot)$ and step to update $l(\cdot)$ are frequently executed, together with Step 5 and 7 combined as a single step, which inspires us to use a priority queue to maintain all the edges associated with priority $l(\cdot)$. The priority queue should support two operations efficiently: finding an edge with the highest priority and update an edge's priority.

Over the introduced data structure, a binary max-heap [18] is suitable to implement the priority queue. A binary max-heap is a heap data structure created using a binary tree with two additional constraints: (1) shape property, the tree is a complete binary tree; that is, all levels of the tree, except possibly the last one are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right; (2) heap property, the key at each node is greater than or equal to that of its children.

A max-heap supports the operations of a priority queue efficiently. We can construct a heap in a linear time, and a max-heap returns a node with the largest key in $O(1)$ time, and updates a node key in $O(\log n)$ time. In fact, the introduction of the max-heap can also decrease the implementation time of Step 7 from $O(n)$ to $O(\log n)$.

Now we can formulate our improved algorithms as shown in Fig. 4.

It is clear that the time complexity of the above algorithm is $O(n\log n+k\log n)$. Obviously, Step 2 only takes the time linear to n. Constructing a max-heap in bottom-up fashion needs only $O(n)$ time. Step 4 runs in a constant time because the root of the heap indicating the edge with the largest $l(\cdot)$, while Step 5 takes $O(\log n)$ time to update an edge's key. Considering that the two steps will be in the loops for k times, Step 4 and Step 5 will run in $O(k\log n)$ time in total. Step 7 can be implemented in

O(n+k) time locating the positions of the added Steiner points. Remember that the first step runs in O($n$log$n$), the entire time complexity of the improved algorithm is O($n$log$n$+$k$log$n$).

Theorem 7: *There is an O(*nlogn+klogn*) time approximation algorithm with performance ratio* 2 *for the bottleneck Steiner tree problem in the rectilinear plane.*

If we use a Fibonacci heap [18] to implement the priority queue, the algorithm can be implemented in the amortized time with O($n$log$n$+$k$log$n$). This is because heap construction takes only O(n) of the amortized time, while determining the edge with the largest key and decreasing an edge's key takes only O(1) and O(log$n$) amortized time. Thus we have,

Theorem 8: *There is an amortized O(*nlogn+klogn*) time approximation algorithm with performance ratio* 2 *for the bottleneck Steiner tree problem in the rectilinear plane.*

V. Simulation of Algorithms

To demonstrate the differences of the time complexities among the original, binary-heap-based and Fibonacci-heap-based approximation algorithms with performance ratio 2, we code the algorithms to obtain their actual execution times. We have used a computer with Intel(R) Core(TM) i5-3470 CPU @ 3.2GHz+3.2GHz and 3.4GB RAM. Upon the number of fixed and additional base stations (terminals and Steiner points) given, we randomly generate the x-coordinate and y-coordinate of each terminal. Fig. 5-Fig. 8 demonstrate the trends of the execution time with the increasing of terminals when the number of Steiner points is fixed. They show that the advantages of our proposed algorithm become more and more clear with the increasing number of Steiner points. In each figure, the horizontal axis represents the number of terminals and the vertical axis represents the execution time. The blue, green, and red curves represent the trends of running time of the original, the Fibonacci-heap-based, and the binary-heap-based algorithms, respectively.

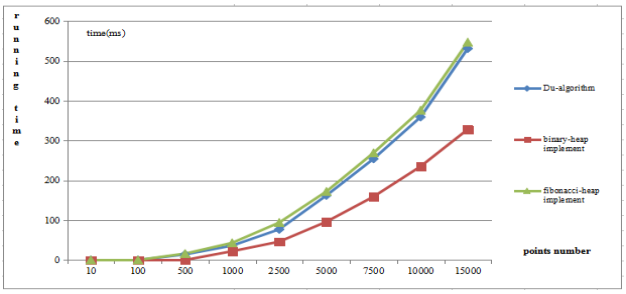

Fig. 5. Trends of running time with Steiner points 50

When the number of Steiner points is 50, the two curves representing the original and Fibonacci-heap-based algorithm almost coincide and the latter is a little worse than the original, while the binary-heap-based algorithm performs much better than the other two, e.g., when the number of terminals is 15000, the binary-heap-based algorithm saves about 40% running time of the original one shown in Fig. 5.

When the number of Steiner points is 100, both of binary-heap-based and the Fibonacci-heap-based algorithms perform better than the original one when the number of terminals exceed 1000. When the number of terminals is 15000, the binary-heap-based and the Fibonacci-heap-based algorithms saves about 52% and 26% running time of the original one, respectively shown in Fig. 6.

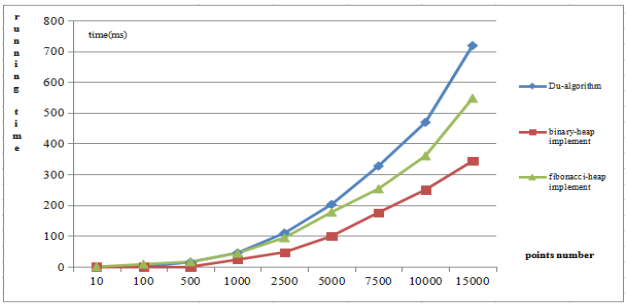

Fig. 6. Trends of running time with Steiner points 100

When the number of Steiner points is 500, the curves representing the binary-heap-based and the Fibonacci-heap-based algorithms almost coincide, and the curve of the Fibonacci-heap-based is a little worse than the binary-heap-based but both of them perform much better than the original one when the number of terminals exceed 2500. When the number of terminals is 15000, both of the binary-heap-based add the Fibonacci-heap-based algorithms saves about 80% running time of the original one as shown in Fig. 7.

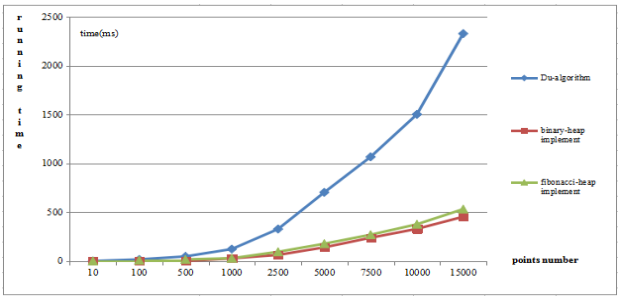

Fig. 7. Trends of running time with Steiner points 500

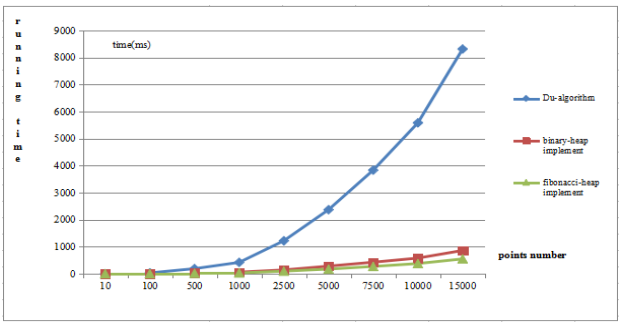

Fig. 8. Trends of running time with Steiner points 2000

When the number of Steiner points is 2000, the curves representing the binary-heap-based and the Fibonacci-heap-based algorithms almost coincide. The Fibonacci-

heap-based algorithm outperforms a little than the binary-heap-based algorithm. But both of them perform much better than the original one when the number of terminals exceed 2500. When the number of terminals is 15000, both of the binary-heap-based and the Fibonacci-heap-based algorithms save about 80% execution time of the original one shown in Fig. 8.

From Fig. 5-Fig. 8, we can find that the Fibonacci-heap-based algorithm can obtain much more efficiency, which is the most improvement with increasing number of Steiner points, while the binary-heap-based algorithm performs much better than the original algorithm regardless of the number of Steiner points. The simulation results match the time complexity analysis on the three algorithms.

VI. Conclusion

In this paper, we have mainly considered the bottleneck Steiner tree problem. The problem asks to find a Steiner tree with n fixed terminal nodes in the plane and up to k Steiner nodes such that the length of the longest edge in the tree is minimized. We first introduced the randomized approximation algorithm with performance ratio $\sqrt{3}+\varepsilon$. Further based on the idea similar to the binary search, we have improved the time complexity of the algorithm from $\frac{1}{\varepsilon}\times \mathrm{poly}(n, k)$ to $(\log\frac{1}{\varepsilon})\times \mathrm{poly}(n, k)$. Then, we have introduced the approximation algorithm with performance ratio 2 for the rectilinear bottleneck Steiner tree problem. By introducing the binary heap and the Fibonacci heap, we have designed a new algorithm to improve the time complexity of the approximation algorithm to $O(n\log n+k\log n)$ and amortized $O(n\log n+k\log n)$, respectively. However, the complexity of Fibonacci-heap-based algorithm with a considerable overhead makes the improvement primarily of theoretical value.

An observation is that our improvements can be directly applied to the polynomial approximation algorithm with performance ratio 2 for the Euclidean bottleneck Steiner tree problem. As an application, the algorithms can be used to efficiently determine the locations of additional base stations in the design of wireless communication networks.

References

[1] M. R. Garey, R. L. Graham, and D. S. Johnson, "The complexity of computing steiner minimal trees," *Siam Journal on Applied Mathematics*, vol. 32, pp. 835-859, 1977.

[2] M. Bern and P. Plassmann, "The steiner problem with edge lengths 1 and 2," *Information Processing Letters*, 1989, vol. 32, pp. 171-176.

[3] S. Arora, "Polynomial time approximation scheme for euclidean TSP and other geometric problems," in *Proc. 37th Annual Symposium on Foundations of Computer Science*, Burlington VT, 1996, pp 2-11.

[4] A. Kahng and G Robins, On Optimal Interconnections for VLSI, Kluwer Publishers, 1995.

[5] A Caldwell, A. Kahng, S. Mantik, *et al*., "On wire length estimations for row-based placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 1265-1278, 1999.

[6] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*, North-Holland, 1992.

[7] L. Wang and D. Z. Du, "Approximations for a bottleneck steiner tree problem," *Algorithmica*, vol 32, pp 554-561, 2002.

[8] L. Wang and Z. Li, "An approximation algorithm for a bottleneck *k*-Steiner tree problem in the euclidean plane," *Information Processing Letters*, vol. 81, pp. 151-156, 2002.

[9] Z. Li and W. Xiao, "Determining sensor locations in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2015, pp. 1-6, 2015.

[10] S. Bae, C. Lee, and S. Choi, "On exact solutions to the euclidean bottleneck steiner tree problem," in *Proc. Third Annual Workshop on Algorithms and Computation*, Kolkata, India, vol. 5431, pp. 105-116, 2009.

[11] M. Sarrafzadeh and C. Wong, "Bottleneck steiner trees in the plane," *IEEE Transactions on Computers*, vol. 41, pp. 370-374, 1992.

[12] G. Lin and G. Xue, "Steiner tree problem with minimal number of Steiner points and bounded edge-length," *Information Processing Letters*, vol. 69, pp. 53-57, 1999.

[13] I. Cardei, M. Cardei, L. Wang, B. Xu, and D. Z. Du, "Optimal relay location for resource-limited energy-efficient wireless communication," *Journal of Global Optimization*, vol. 36, pp. 391–399, 2006.

[14] D. Z. Du, L. Wang, and B. Xu, "The euclidean bottleneck steiner tree and steiner tree with minimum number of steiner points," in *Proc. 7th Annual International Conference on Computing and Combinatorics*, Guilin, China, 2001,vol 2108, pp 509-518.

[15] Z. Li, D. Zhu, and S. Ma, "Approximation algorithm for bottleneck steiner tree problem in the euclidean plane," *Journal of Computer Science and Technology*, vol. 19, pp. 791-794, 2004.

[16] Z. Li and W. Xiao, "Nearly optimal solution for restricted euclidean bottleneck steiner tree problem," *Journal of Networks*, vol. 9, pp. 1000-1004, 2014.

[17] H. J. Prömel and A. Steger, "A new approximation algorithm for the steiner tree problem with performance ratio 5/3," *Journal of Algorithms*, vol. 36, pp. 89–101, 2000.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *et al*., *Introduction to Algorithms*, 3rd ed., MIT Press and McGraw-Hill, 2009.

[19] H. Zhou, N. Shenoy, and W. Nicholls, "Efficient minimum spanning tree construction without delaunay triangulation," *Information Processing Letters*, vol. 81, pp. 271-276, 2002.

[20] L. Guibas and J. Stolfi, "On computing all north-east nearest neighbors in the L1 metric," *Information Processing Letters*, vol. 17, pp. 219-223, 1983.

Zimao Li, Ph.D., born in Linqing, P.R. China, 1974, received his Bachelor's degree in Mathematics, Master's degree in Computer Science from Shandong University in 1996 and 1999, respectively, and his Ph.D. degree in Computer Science from City University of Hong Kong in 2002. His research interests include computational complexity, approximation algorithms and design and analysis of algorithms.

Yingying Wang born in Suizhou, P.R. China, 1991, received her Bachelor's degree in Computer Science and Technology from Hunan Institute of Science and Technology, currently she is a Master degree candidate. Her research interest is design and analysis of algorithms.

Maode Ma, received his Ph.D. degree in computer science from Hong Kong University of Science and Technology in 1999. Now, he is an Associate Professor in the School of Electrical and Electronic Engineering at Nanyang Technological University in Singapore. He has extensive research interests including wireless networking and network security. He has led and/or participated 18 research projects funded by government, industry, military and universities in various countries. Dr. Ma is a Fellow of IET and a senior member of IEEE Communication Society and IEEE Education Society. He is the Chair of the IEEE Education Society, Singapore Chapter. He is serving as an IEEE Communication Society Distinguished Lecturer.