Stree: An Interactive Program for Weighted Steiner Trees

Valério Ramos Batista[1*], Marcelo Zanchetta do Nascimento[2], Wendhel Raffa Coimbra[3]
[1] UFABC, av Estados 5001, 09210-580 St André, Brazil.
[2] UFU-FACOM, av João N Ávila 2121, Bl.B, 38400-902 Uberlândia, Brazil.
[3] UFMS, av Pedro Pedrossian 725, 79500-000 Paranaíba, Brazil.

* Corresponding author. Tel.: +55-11-4996-0077; email: valerio.batista@ufbc.edu.br
Manuscript submitted June 19, 2016; accepted July 14, 2016.
doi: 10.17706/jsw.11.10.1062-1072

Abstract: Here we present the extended version of a previous short publication about stree, which is a fully written program with a semi-automatic method that generates weighted Steiner trees. Our choice of the programming language, and the use of well-known theorems from Geometry and Complex Analysis, allowed this method to be implemented with only 764 lines of effective source code. This simplifies the understanding and the handling of this beta version for future developments.

Key words: Programmed code, weighted Steiner minimal trees.

1. Introduction

One of the main problems at implementing multicast in Wide Area Networks (WAN) is the high cost of transmissions between terminals. Cost reduction is attained by adding routers to the network but this increases complexity (see [1]–[2]). Steiner trees have long been used in order to optimise routes aiming at the lowest cost possible (see [3]–[4]).

Although the Steiner Minimal Tree (SMT) problem belongs to the NP-hard class (see [5]), it can be exactly solved by fast algorithms for terminals in thousands. The best example is the GeoSteiner algorithm. Essentially, it checks for terminals that are as close as possible to vertices of equilateral triangles. Afterwards, it prunes sub-optimal trees. See http://www.diku.dk/hjemmesider/ansatte/martinz/geosteiner for details.

GeoSteiner is amazingly fast for terminals positioned at random. However, it is not the case when they follow a pattern. For instance, we use 4GB of RAM, microprocessor Intel Core i5 3.2GHz, and operating system Linux Ubuntu 12.04. With this setting GeoSteiner takes 73.02s to generate Figure 1. This time drops to only 0.06s when the 31 terminals are at random. Fig. 1 was obtained through the datafile `pat.tsp` contained in `test_codes.zip`, which we shall discuss in Section 4. Compare it with Fig. 2, in which the SMT *does* follow a pattern.

Moreover, the GeoSteiner algorithm cannot be adapted to find *weighted* SMT (WSMT). This fact, together with the slowness in patterned cases, is precisely due to the strategy of looking for equilateral triangles.

Given a graph $G=(V,E,w)$, a subset $S\subset V$ and a weight function w, we say that a tree $T\subset$G is a WSMT if it spans all vertices of S and also minimises the total weight. This classical definition can be specialised to edge- or node-weight when the domain of w is either E or V, respectively. The problem has further variations, like for *unity disk graphs* and restrictions on w, that have been studied recently [6]–[9].

These and other works make use of heuristics. They are devoted to automatic methods that are fast at

generating weighted Steiner trees with good chances of approaching the minimum weight. But if one really seeks a WSMT there are little chances that automatic methods will find it, unless applied to a few number of vertices.

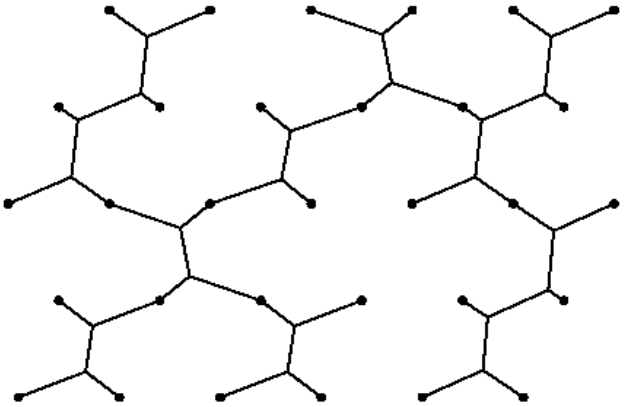

Fig. 1. Non-patterned GeoSteiner output.

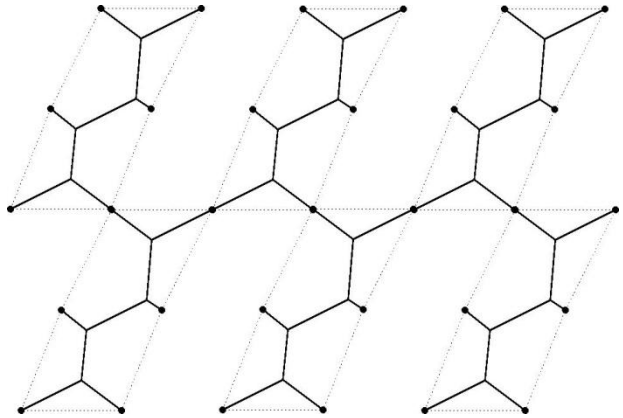

Fig. 2. Patterned SMT.

Of course, a semi-automatic method that includes some feedback to the user increases the chances of finding this tree. Specially if we can rely on the skill and good guesses of a trained user. In the last paragraph from §6 of [10] the very authors had already made this kind of comment. Of course, their work was devoted to the standard Euclidean case and is previous to GeoSteiner by three decades. However, their comment is still modern in the sense that one cannot always predict whether a fast algorithm exists to solve a given problem.

In that same work Gilbert and Pollak conjectured that an SMT must have its length in the interval $Lprim \cdot [\sqrt{3}/2,1]$, where *Lprim* is the length of the minimal spanning tree (MST). This one can be obtained by Prim's algorithm [11]. According to [12] this conjecture is right. If so, any Steiner tree obtained from the MST will be at most 13.4% shorter.

It seems little, but if a connection is used extensively this 13.4% represents a great saving in the long-term. Moreover, in [13] the authors claim that Gilbert and Pollak's conjecture is not completely answered yet. Thus, it might even happen that we get an SMT *under* this ratio.

Of course, a semi-automatic program is not suitable for thousands of terminals, except for a long-term project distributed to a team of users. But even this exception does not apply to extreme cases, like VLSI-design through *rectilinear* Steiner trees, in which *millions* of terminals are needed. However, terminals in hundreds are still the case in several applications, like sound and video cards. Their frequent access makes it desirable to minimise delay as much as possible. Even a 1% improvement would count in this case.

Indeed, weighted Steiner trees have many practical applications. Among others, they are used in network formation games, computational sustainability and electric power networks [14]–[16]. For specific applications in industry, please see [17]. The Steiner tree problem can also be studied in 3D and higher dimensions [18]. Moreover, some of its variations are applicable to computational biology [19] and social networks [20], among others. This is the case of *prize-collecting Steiner trees* (see [21] for details).

In [22] we introduced a fully written programmed code to generate weighted Steiner trees. That reference is in fact a short version finally detailed in this present paper. Our choice of a programming language was made in order to spare the graphical environment, which Matlab already brings in a very well built-in way. Since we have not used toolboxes, this code can be easily adapted to a free software like Octave, though this one has a simpler graphical user interface. Anyway, either Matlab or Octave produce a much shorter code, easier to understand and to handle for future developments. Indeed, the present version of

our program has didactic purposes, for it is accessible even to undergraduate students in Computer Science.

Many original ideas were used to write our code. They are based on theorems normally relegated to Math courses only, specially Geometry and Complex Analysis. The chosen programming language allowed these theoretical results to be implemented into elegant and efficient codes. The reader can download them through the link *Softwares* of

https://sites.google.com/site/vramos1970

As an example, the file `estim.m` (contained in `stree.zip`) runs with only 21 lines of effective source code. This program draws the weighted MST (WMST) and computes its total weighted length. All together, `stree.m` and its related to programs have only 855 lines, which drop to 764 if we do not count `sread.m` and `swrite.m` (for graphical input and output of terminal points).

The aim of this paper is to detail the program `stree.m` totally written with original ideas we have just mentioned in the previous paragraph. The rest is organised as follows: in Section 2 we prove some results used in our paper. Section 3 is devoted to explaining some theorems that we have used to write the program. Section 4 presents some of the geometrical and analytical ideas that were used to implement `stree.m`. In Section 5 we present a practical application of `stree`, and we finally draw our conclusions in Section 6.

2. Preliminaries

As we mentioned in the Introduction, the MST is frequently used to construct a Steiner tree. Prim's algorithm is easily adaptable to find the WMST for terminals that are weighted as follows:

Definition 2.1. Consider the tree $T=(V,E)$ with a weight function $w:V\to \mathrm{IR}_+$ and 0–1 ad-jacency matrix a_{ij}. Then T is an MST if it minimises the total cost C given by

$$C=\sum_{i<j} a_{ij}||V_i-V_j||, \tag{1}$$

where $||V_i - V_j|| = ½(w_i - w_j)|V_i - V_j|$ is the connection cost between terminals V_i and V_j, and $|V_i - V_j|$ is the (Euclidean) distance between V_i and V_j.

REMARK: The cost $||\cdot||$ coincides with the Euclidean distance when $w:V\to\{1\}$. To the best of our knowledge Definition 2.1 is new in the literature. We apply it to solve a practical problem described in Section 5.

For a given set of terminals $V=\{V_1,...,V_n\}$ and a weight function $w:V\to \mathrm{IR}_+$ we can find the edges E that minimise the cost C in (1) and result in an MST $T=(V,E)$. But if we can add extra points $S=\{S_1,...,S_m\}$ to V, namely $\tilde{V}=V\cup S$, we shall find the corresponding $\tilde{T}=(\tilde{V},\tilde{E})$ such that $\tilde{C}\le C$. We claim that $\tilde{C}<C$ exactly when $S\neq\emptyset$, providing one chooses a suitable extension $\tilde{w}:\tilde{V}\to \mathrm{IR}_+$ of the weight function.

The following lemma shows how to make this choice when $(n,m)=(3,1)$. See Fig. 3.

Lemma 2.1. *Consider a triangle with vertex weights a, b and c, and suppose it admits a classical (Euclidean) Steiner point. If* $s\le \min\{a,b,c\}$ *is the weight of the Steiner point, then its connection with the vertices will cost less than any other connection through the vertices only.*

Proof. Let ℓ_1, ℓ_2, ℓ_3 be the distance from the Steiner point to the vertices a, b and c, respectively. We want to prove that

$$(a+s)\ell_1+(b+s)\ell_2+(c+s)\ell_3<(a+b)L+(b+c)L. \tag{2}$$

Case 1: $b\le a\le c$. The law of cosines implies $\ell_1+\ell_2/2<L$ and $\ell_3+\ell_2/2<L$. Hence $a\ell_1+b\ell_2+c\ell_3<aL+cL$.

Moreover, $s \le b$ and $\ell_1+\ell_2+\ell_3<$L$+L$. This implies (2).

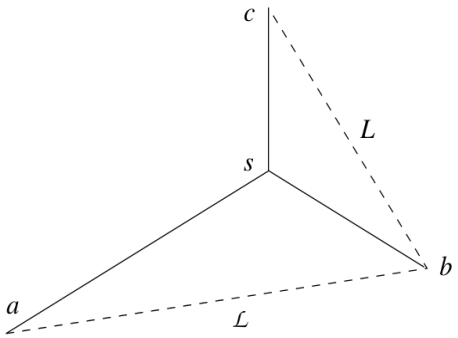

Fig. 3. Adding a Steiner point to three weighted terminals.

Case 2: $a \le b \le c$. We have $(a\ell_1+b\ell_2/2)+(b\ell_2/2+c\ell_3)<bL+cL$. Since $s \le a$ and $aL \le bL$, then (2) follows.

Case 3: $a \le c \le b$. Notice that $(a\ell_1+b\ell_2/2)+(b\ell_2/2+c\ell_3)<b(L+L)$. Since $s \le a$ and $aL \le cL$, then (2) holds again.

q.e.d.

Definition 2.2. A *full* Steiner tree is one in which any terminal connects to a Steiner point.

Lemma 2.2. *Consider a* full *Steiner tree with* $n \ge 3$ *terminals* $A_1,...,A_n$ *and Steiner points* $S_1,...,S_{n-2}$. *Add weights to their respective terminals as* a_i, $1 \le i \le n$, *and to the Steiner points as* $s_i=\min\{a_1,...,a_n\}$, ∀ *i. The resulting weighted tree is then a local minimum of* C.

Proof. Based on Lemma 2.1 we see that a sufficiently small displacement of any S_i will increase the weighted length of the tree. Therefore, it characterises a local minimum of C.

q.e.d.

Regarding the Euclidean non-weighted case, in §§3.7 of [10] the authors show that any Steiner tree can be decomposed into a union of full Steiner trees. By adding weights as described in Lemma 2.2 we can run through all Steiner trees $T\tilde{}=(V\tilde{},E\tilde{})$ determined by V and compute the corresponding C. There is a finite number of such trees, hence the least C will determine $T\tilde{}$ as an MST.

The most important consequence of Lemmas 2.1 and 2.2 is that Steiner points can be added exactly as in the Euclidean case for arbitrary $n \ge 3$. The resulting MST $T\tilde{}=(V\tilde{},E\tilde{})$ coincides with a Euclidean non-weighted Steiner tree, which will not be necessarily the Euclidean SMT. Anyway, many properties proved in [10] still hold for $T\tilde{}$: *Convex hull*, *Maxwell's Theorem*, *Lune* and *Wedge* properties.

3. Some Theorems Used in Our Code

The following theorems were used to implement `rprim.m` and `mksaw.m`, respectively.

Theorem 3.1. *Let S be a finite set of points with at least two elements. In this case, there are P,Q*∈ *S such that the distance between P and Q is maximal. The segment PQ is called the diameter of S.*

Proof. We can write $S=\{V_1,...,V_n\}$ and consider the set $D=\{|V_i - V_j| : 1 \le i,j \le n\}$, which has at most $(n-1)!$ elements. Then max D is surely given by certain indexes $\hat{\imath}, \hat{\jmath}$ that determine P as $V_{\hat{\imath}}$ and Q as $V_{\hat{\jmath}}$. Finally we get PQ as stated.

Theorem 3.2. *Let Q be a* quadrilateral *with consecutive vertices* P_i, $i=1,...,4$. *Let* $vec_{i-2} = P_i-P_1$, $i \ge 2$, *and* $Vec_{i-1} = P_i-P_4$, $i \le 3$. *Then Q is convex precisely when* vec_1 *is between* vec_0, vec_2, *and also* Vec_1 *is between* Vec_0, Vec_2.

Proof. By definition, Q is convex exactly when P_2, P_4 are at opposite sides of the straight line determined by P_1P_3 and P_1, P_3 are at opposite sides of the straight line determined by P_2P_4. This is equivalent to the

assertion of the Theorem.

Theorem 3.3. *Let $A_1,...,A_n$ be $n\geq 3$ terminals in the complex plane connected by a full Steiner tree S. Let $V=\{V_1,...,V_s\}$ be the set of Steiner points of S. Suppose that each element in V admits a terminal such that both are the extremes of a segment in S. In this case there exists $V_i \in V$ that determines all points in $V \setminus \{V_i\}$.*

Proof. By following the arguments from §3.4 of [10], we have $s=n-2$ and only one segment in S with extreme A_k for each $k \in \{1,...,n\}$. Since $s\geq 1$ the arguments from §6 of [10] apply. Namely, if each element of V were connected to a single terminal, then we would have $s=n$, a contradiction. Hence, there exists $V_i \in V$ that connects two terminals. Up to re-indexing, these are A_1, A_2 and $i=1$.

Now consider the ray given by

$$r(t) = V_1 + t \cdot \left(\frac{V_1 - A_1}{|V_1 - A_1|} + \frac{V_1 - A_2}{|V_1 - A_2|} \right), \ t \in \mathbb{R}_+.$$

If $n=3$, then it is clear that a unique positive τ gives $r(\tau)=A_3$.

Now take $n > 3$. For each positive t consider the rays $\rho_t = r(t)+(V_1 - A_1)\cdot \mathrm{IR}_+$ and $\rho_t = r(t)+ (V_1-A_2)\cdot \mathrm{IR}_+$. Let $A=\{A_1,...,A_n\}$ and take all positive t such that $(\rho_t \cup \rho_t)\cap A \neq \emptyset$. They will make a finite set $\{t_1,...,t_m\}$ with $m\leq n-2$.

Since V_1 must be connected to another Steiner point, say V_2 (after re-indexing), then $V_2= r(t_k)$ for a certain $k \in \{1,...,m\}$. Of course, $r(t_k)$ is connected to a terminal in A. Up to re-indexing, it is A_k.

In order to find k we must repeat the same arguments with $V_1, A_k, r(t_k)$ respectively in the place of A_1, A_2, V_1, and so on. This will give at most $s!$ full trees. One of them is minimal, whence all points $V_2,...,V_s$ are determined. This concludes the proof of the Theorem.

Our next section explains a bit of `stree.m`, `rprim.m` and `mksaw.m`, which exemplify the applications of the theorems discussed in this present section.

4. Efficient Codes from Geometry and Complex Analysis

Figure 4 describes our pseudocode, which works recursively while the connection matrix is not complete. There the WMST is called Tree_of_Prim because we have adapted Prim's classical algorithm to the weighted case.

`stree` **Pseudocode**
Input: Set of terminals
Output: Steiner tree
1. Pts ← terminals, Connexion_Matrix ← 0
2. Tree_of_Prim ← Compute_Tree_of_Prim(Pts)
3. Steiner_hull ← Compute_Steiner_hull(Pts)
4. While Connexion_Matrix implies Stree non-connected
i. Subset_Pts ← User's Choice of a Subgroup(Pts)
ii. Subtree ← Compute_Connection(Subset_Pts)
iii. If Subtree not OK, redo Step **i**
iv. Else Connexion_Matrix ← Connection(Subset_Pts)
5. return Steiner tree (and its weighted length)

Fig. 4. The pseudocode of `stree.m` algorithm.

Initially, `stree.m` calls `lune.m` and `cvxhull.m` to inscribe the terminal points into the Steiner hull, namely a polygon coloured cyan as shown in Figure 2 of the user manual. In general, there will be isolated terminals inside the polygon. The ones that build its vertices are marked by `stree.m` with either **0** or **1**,

which mean ``greater'' and ``lesser'' than 120 °, respectively. The characters **0** and **1** are stored in a string `s`. For instance, if `s` contains a stretch **010**, this hints to three terminals joined by a Steiner point.

But they are not connected automatically, for this hint might not lead to the shortest tree. We use the variable `s` for purposes like building zigzags out of stretches **01...10**. Of course, not all zigzags are good, but the program will try them back and forth, and even split them into parts.

Of course, `lune.m` and `cvxhull.m` are based on the Convex Hull and Lune properties described in [10]. They do not give a polygon with zigzags, but with stretches like in Figure 5. Hence, `stree.m` calls `mksaw.m` in order to get the zigzag.

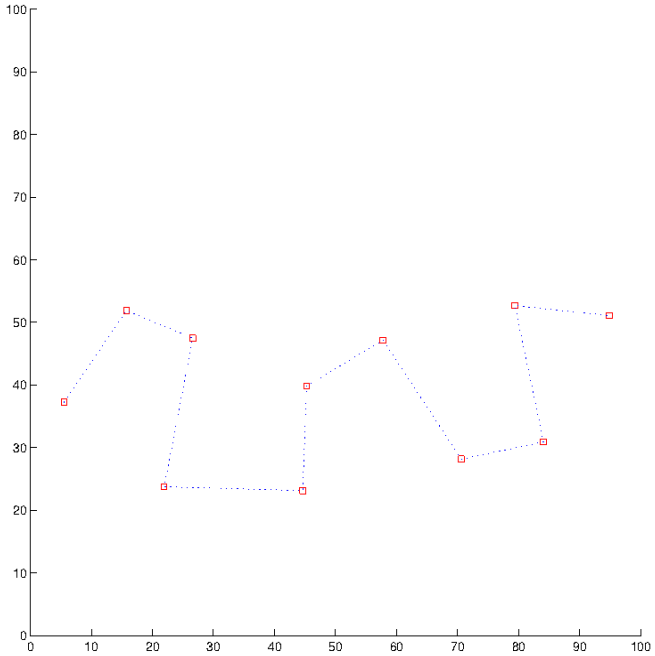

Fig. 5. A ''**01...10**'' stretch.

If you want to test these procedures, extract the attached file `test_codes.zip` in a folder and start Matlab inside it. At the Matlab prompt run ''Cvxhull'' for the terminal data `test2` and save the output with the name `test3`. A figure will show the original input order of the terminals (dotted lines), and a blue polygon involving them (the convex hull). Then run ''Lune'' for `test3` and name the output `test4`.

Now we are going to explain `Rprim.m` and `Mksaw.m`, which will finally generate a zigzag out of `test2.txt` (the initial datafile). According to Theorem 3.1, there is a diameter for the points in `test2.txt`, and lines 2-3 of `Rprim.m` are precisely the commands to find them, namely `Pts(h(k))` and `Pts(k)`. However, you *must* run ''Rprim.m'' for the re-ordered data `test4`.

Now we project all points along the diameter and re-order them by increasing distance between their projection and the extreme `Pts(k)`. That is what the while-loop does in lines 11 to 19. The re-ordered input is restored in line 20, and may be saved in line 21. We did it in `test5.txt`, which `Mksaw.m` finally re-orders into a zigzag.

Notice that the terminals from `test5.txt` are in zigzag order *except* for one single square wavelet. However, `Mksaw.m` works well even if you have a totally square wave. Enter the terminal points in an order that makes such a wave. We have used `test6.txt` to generate Figure 5. This precaution is not necessary when you run `stree.m`, for it will rearrange the points as explained above.

By walking along the square wave from the very first point on, two consecutive steps will always lead to a

vertex that makes a quadrilateral Q with the following two. That is why the while-loop ends in `pts=pts(2:end)`.

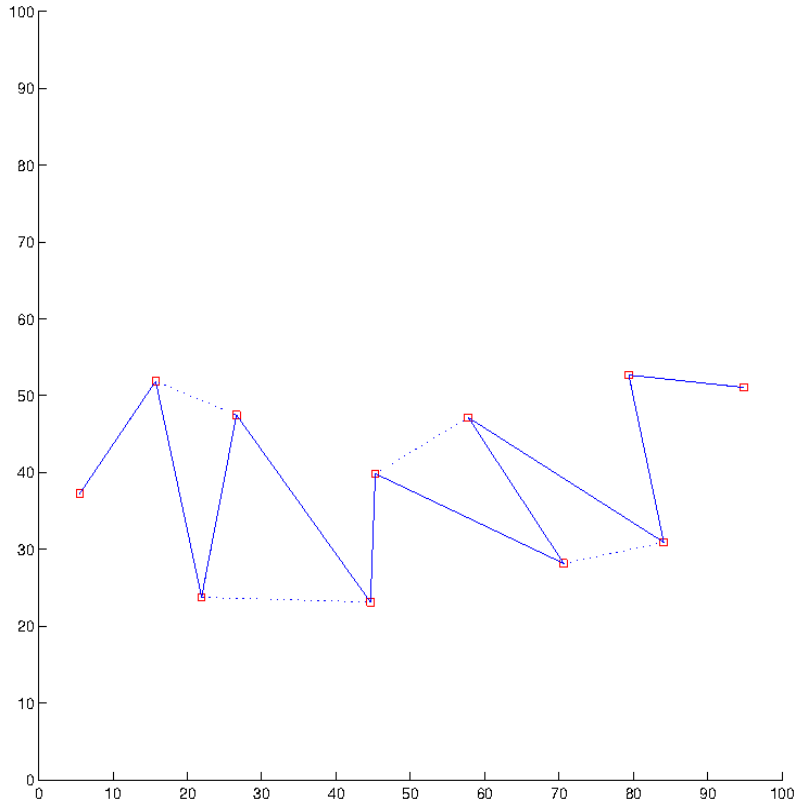

Fig. 6. Its rearrangement by "Mksaw".

Now look at lines 4 to 9 of `Mksaw.m` and apply Theorem 3.2 to Q. The symbol × will indicate the *vector product*. From Geometry, the convexity criteria given by this theorem is equivalent to $vec_0 \times vec_1$, $vec_2 \times vec_1$ pointing in opposite directions. The same holds for $Vec_0 \times Vec_1$, $Vec_2 \times Vec_1$. Taking these vectors as complex numbers in $\mathbf{C} \times \mathrm{IR}$, say $vec_1 = a + ib = (a,b,0)$ and $vec_0 = c + id = (c,d,0)$, we have $vec_0 \times vec_1 = (0, 0, bc - ad)$, namely $bc - ad = Im\{vec_1 \cdot \mathrm{conj}(vec_0)\}$. Thus, from Complex Analysis the vectors $vec_0 \times vec_1$, $vec_2 \times vec_1$ will point in opposite directions precisely when the signs of $Im\{vec_1 \cdot \mathrm{conj}(vec_0)\}$ and $Im\{vec_1 \cdot \mathrm{conj}(vec_2)\}$ are opposite. Hence, lines 10-11 from `Mksaw.m` check if Q is convex. In this case, the program makes a sawtooth out of Q. Then we go two steps forward with the command `pts=pts(2:end);` and the while-loop repeats the process, unless we have already come to the end of the line.

5. Practical Application

Suppose a fibre optic company plans to install cables that will provide connection within a group of cities. If we consider only the cost to install cables underground, the SMT will minimise it providing the soil is free from barriers like groundwater, rocky earth, and so on. Let us suppose we have this favourable soil everywhere in the region that includes these cities.

As an example, let us consider the terminal points of `test0.txt` as our group of cities. This file is also used as an example in the user manual. Figure 6 shows the GeoSteiner output by ignoring weights, whereas Figure 7 was generated by `stree`.

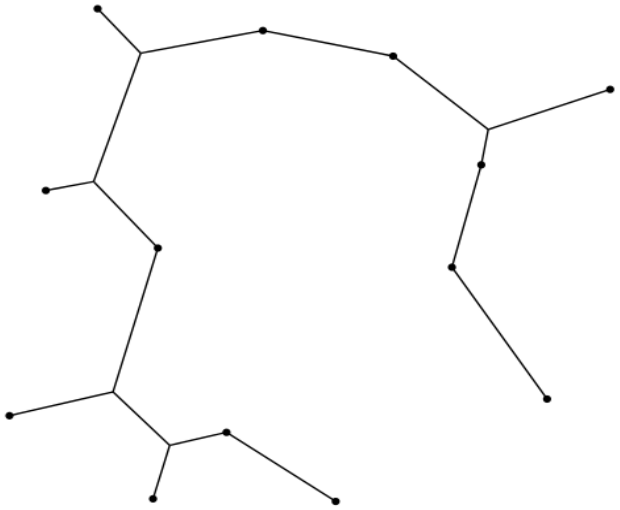

Fig. 6. GeoSteiner output of `test0`.

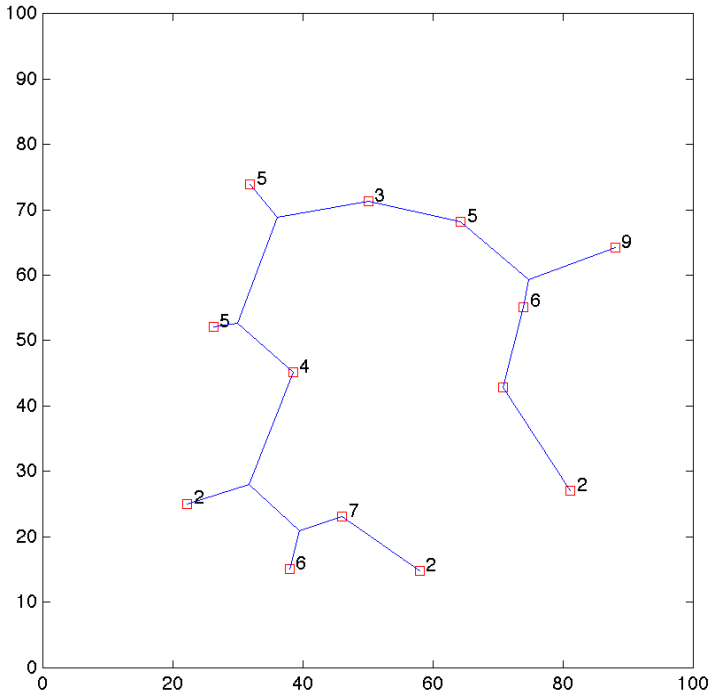

Fig. 7. The corresponded weighted tree.

However, there are other costs besides the underground installation of optic fibre cables. For example, land taxes, local maintenance expenses, etc. Suppose the extra costs for each city were directly related to the corresponding weights in Figure 7. Our program `stree` prints 715.4669 as the weighted length. However, in the user manual we showed a tree of length 598.3593, which is very likely to be the minimum. Anyway, that tree is already 16% cheaper than the one given by GeoSteiner, which cannot handle weights.

6. Conclusions

Differently from the approach of trying a fully automated method, we propose to take advantage of the good choices that a user can make. Many attributes like intuition, guess, practice and a bird's-eye view are valuable means that one cannot translate into any programming language. Hence, as long as a task is feasible with the help of supervision we suggest taking it into account, besides the fully automated methods. This proposal is not new, but we endeavour to obtain a code that is both easy to run and to understand.

The program `stree` is still in the beta version. Further improvements will include more feedback to the user. For instance, the tree will be also checked with Maxwell's Theorem and the (Double) Wedge properties (see [10] for details). Some tests can be implemented to run while the users are drawing, so that they may also undo steps which just seem successful with this present version.

Moreover, `stree` still works strongly devoted to *real* Steiner trees, which in fact should be adapted to practical purposes. For instance, outputs consider even Steiner points extremely close to a terminal. By implementing such a tree to a multicast network, those Steiner points can be unnecessary and even costly. In future, the user will decide on the tolerance regarding the minimum distance that terminals and Steiner points will keep apart.

By the way, it is even preferable to implement multicast networks with a minimum number of Steiner points, because of the high cost of the routers. This is also the case of WDM optical networks (see [23]). Therefore, it will be useful to have future versions of `stree` devoted to the construction of such trees. The *rectilinear* Steiner trees are also of interest (see [24] and [25]), and then another option like `stree` to be developed.

Acknowledgment

Many improvements in this paper were due to the careful analyses carried out by referees. We thank them for their valuable help. We are also grateful to Cláudio Nogueira de Meneses, professor at the Federal University of ABC, for his assistance with weighted graphs.

References

[1] Jia, X.-H., Du, D.-Z., Hu, X.-D., Lee, M.-K., & Gu, J. (2001). Optimization of wavelength as-signment for qos multicast in wdm networks. *IEEE Transactions on Communications*, *49(2)*, 341–350.

[2] Sahasrabuddhe, L. H., & Mukherjee, B. (2000). Multicast routing algorithms and pro-tocols: A tutorial. *Network*, *14(1)*, 90–102.

[3] Ausiello, G. (1999). Complexity and approximation: Combinatorial optimization pro-blems and their approximability properties. *Springer Science and Business Media*.

[4] Hu, X.-D., Shuai, T.-P., Jia, X., & Zhang, M.-Z. (2004). Multicast routing and wavelength as-signment in wdm networks with limited drop-offs. *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*.

[5] Garey, M. R., Graham, R. L., & Johnson, D. S. (1997). The complexity of computing Stei-ner minimal trees. *SIAM Journal on Applied Mathematics*, *32(4)*, 835–859.

[6] Angelopoulos, S. (2006). *The Node-Weighted Steiner Problem in Graphs of Restricted Node Weights, Algorithm Theory–SWAT*.

[7] Demaine, E. D., Hajiaghayi, M., & Klein, P. N. (2009). *Node-Weighted Steiner Tree and group Steiner Tree in Planar Graphs, Automata, Languages and Programming*.

[8] Li, X., Xu, X.-H., Zou, F., Du, H., Wan, P., Wang, Y., Wu, W. (2009). *A Ptas for Node-Weighted Steiner Tree in unit Disk Graphs, Combinatorial Optimization and Applications*.

[9] Zou, F., Li, X., Gao, S., & Wu, W. (2009). Node-weighted steiner tree approximation in unit disk graphs. *Journal of Combinatorial Optimization*, *18(4)*, 342–349.

[10] Gilbert, E., & Pollak, H. (1968). Steiner minimal trees. *SIAM Journal on Applied Mathematics*, *16(1)*, 1–29.

[11] Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, *36(6)*, 1389–1401.

[12] Du, D.-Z., & Hwang, F. K. (1992). A proof of the gilbert-pollak conjecture on the Steiner ratio.

Algorithmica, *7(1-6)*, 121–135.

[13] Innami, N., Kim, B., Mashiko, Y., & Shiohama, K. (2010). The Steiner ratio conjecture of Gilbert-Pollak may still be open. *Algorithmica*, *57(4)*, 869–872.

[14] Dilkina, B., & Gomes, C. P. (2010). Solving connected subgraph problems in wildlife conservation, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*.

[15] Guha, S., Moss, A., Naor, J. S., & Schieber, B. (1999). Efficient recovery from power outage. *Proceedings of the Thirty-First Annual ACM Symposiumon Theory of Computing* (pp. 574–582).

[16] Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, E., Wexler, T., & Roughgarden, T. (2008). The price of stability for network design with fair cost allocation. *SIAM Journal on Computing*, *38(4)*, 1602–1623.

[17] Cheng, X., & Du, D.-Z. (2013). Steiner trees in industry. *Springer Science and Business Media*.

[18] Du, D.-Z., Smith, J., & Rubinstein, J. H. (2013). Advances in Steiner trees. *Springer Science and Business Media*.

[19] Liu, L., Song, Y., Zhang, H., Ma, H., & Vasilakos, A. V. (2015). Physarum optimization: A biology-inspired algorithm for the Steiner tree problem in networks. *IEEE Transactions on Computers*, *64(3)*, 818–831.

[20] Rozenshtein, P., Anagnostopoulos, A., Gionis, A., & Tatti, N. (2014). Event detection in activity networks. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1176–1185).

[21] Hegde, C., Indyk, P., & Schmidt, L. (2014). A fast, adaptive variant of the Goemans-Williamson scheme for the prize-collecting Steiner tree problem. *Workshop of the 11th DIMACS Implementation Challenge*.

[22] Nascimento, M. Z., Batista, V. R., & Coimbra, W. R. (2015). An interactive programme for weighted Steiner trees. *Journal of Physics: Conference Series*.

[23] Chen, D., Du, D.-Z., Hu, X.-D., Lin, G.-H., Wang, L., & Xue, G. (2000). Approximations for Steiner trees with minimum number of steiner points. *Journal of Global Optimization*, *18(1)*, 17–33.

[24] Fößmeier, U., & Kaufmann, M. (1997). Solving rectilinear Steiner tree problems exactly in theory and practice. *AlgorithmsESA'97*, 171–185.

[25] Zachariasen, M. (1999). Rectilinear full steiner tree generation. *Networ*ks, *33(2)*, 125–143.

Valério Ramos Batista graduated in computer engineering at the Technological Institute of Aeronautics in SJ Campos, Brazil, Dec 1993. He concluded his master's in mathematics at the University of São Paulo, Brazil, Sep 1996, and his PhD in mathematics at the University of Bonn, Germany, Jul 2000. Nowadays he works as a full professor in computer modelling at the Federal University of ABC, St André, Brazil.

Some recent publications as a co-author include: (1) "LBP operators on curvelet coefficients as an algorithm to describe texture in breast cancer tissues", Expert Systems with Applications, 2016; (2) "A software tool based on the Surface Evolver for precise location of tumours as a preoperative procedure to partial mastectomy", Journal of Physics: Conference Series, 2015; (3) "Programming plantation lines on driverless tractors", Revista SODEBRAS, 2014. His main research areas are Image Processing and Computer Modelling applied to Medicine, Agriculture and Human-Computer Interfaces.

Prof. Ramos Batista was awarded the Scholarship in research productivity by the Brazi- lian National Council for Scientific and Technological Development, from Mar 2008 to Feb 2011.

Marcelo Zanchetta do Nascimento was born in São José do Rio Preto Brazil, in 1976. He received the high-level technologist degree from the University Center of Rio Preto, São

Paulo, in 1996. He received his MSc. and Ph.D. in electrical engineering at the University of São Paulo, São Carlos, Brazil, in 2002 and 2005, respectively. Since 2013, he has been an professor of the Faculty of Computer Sciences at the Federal University of Uberlândia. His research interests include medical image processing, computer vision, and pattern recognition.

Wendhel Raffa Coimbra was born in Guararapes Brazil, in 1985. He concluded his master's in applied mathematics at the Federal University of ABC, Santo André, Brazil, Dez 2009, and his Ph.D. in electrical engineering at the University of São Paulo, São Carlos, Brazil, in 2016. Since 2010 he has been a professor of the Federal University of Mato Grosso do Sul, CPAR-UFMS. His research interests include invariance principle, Fuzzy systems, differential equations, nonlinear systems and dynamic systems.