

An interactive programme for weighted Steiner trees

Marcelo Zanchetta do Nascimento¹, Valério Ramos Batista²,
Wendhel Raffa Coimbra³

¹ UFU-FACOM, av. João Neves de Ávila 2121, Bl.A, 38400-902, Uberlândia-MG, Brazil

Tel: +55-34-3239-4571, *Email:* marcelo.zanchetta@gmail.com

² CMCC-UFABC, r. Sta. Adélia 166, Bl.B, 09210-170 St. André-SP, Brazil

³ UFMS, rod. BR 497 km 12, 79500-000 Paranaíba-MS, Brazil

Abstract. We introduce a fully written programmed code with a supervised method for generating 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 eases the understanding and the handling of this beta version for future developments.

Keywords: *weighted minimal Steiner trees, programmed code*

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 [14, 17]). Steiner trees have long been used in order to optimise routes, aiming at the lowest cost possible (see [3, 12]).

Although the Steiner Minimal Tree (SMT) problem belongs to the NP-hard class (see [9]), 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. Figure 1 was obtained through the datafile `pat.tsp` contained in `stree.zip`, which we shall discuss in Section 3. Compare it with Figure 2.

Moreover, the GeoSteiner algorithm cannot be adapted to find *weighted* minimal

Steiner trees. This fact, together with the slowness in patterned cases, is precisely due to the strategy of looking for equilateral triangles.

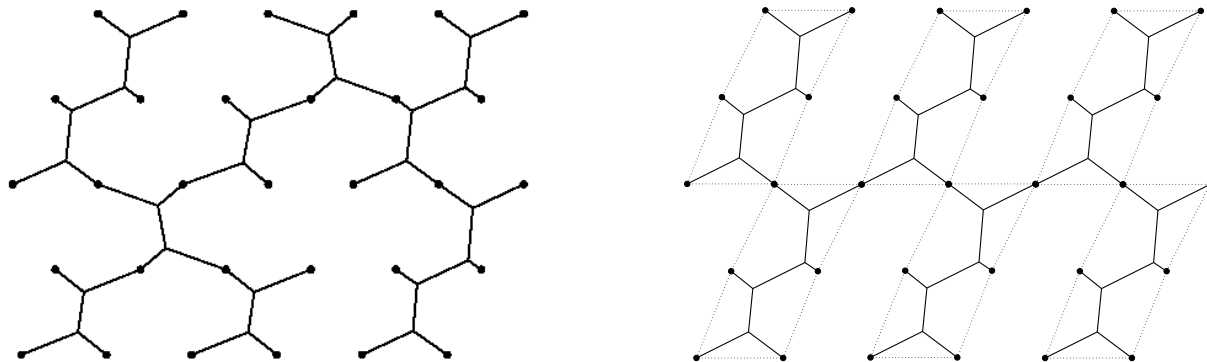


Figure 1: Non-patterned GeoSteiner output. Figure 2: Patterned SMT drawn with Xfig.

Given a graph $G = (V, E, w)$, a subset $S \subset V$ and a weight function w , a *weighted minimal Steiner tree* $T \subset G$ is one that 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 [1, 5, 15, 19].

These and other works make use of heuristics. They are devoted to non-supervised methods that are fast at generating weighted Steiner trees with good chances of approaching the minimal weight. But if one really seeks a weighted *minimal* Steiner tree, there are little chances that non-supervised methods will find it, unless applied to a few number of vertices.

Of course, a supervised 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 a Steiner minimal tree must have its length in the interval $L_{prim} \cdot [\sqrt{3}/2, 1]$, where L_{prim} is the length of the minimal spanning tree. This one can be obtained by Prim's algorithm [16], and here we shall name it after him for concision and clarity. According to [7], this conjecture is right. If so, any Steiner tree obtained from Prim's can improve it of at most 13.4%.

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 the 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 supervised programme 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, it is used in network formation games, computational sustainability and electric power networks [6, 11, 2].

We introduce a fully written programmed code for generating weighted Steiner trees. 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, this produces a much shorter code, easier to understand and to handle for future developments. Indeed, the present version of our programme 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. But the chosen programming language allows these theoretical results to be implemented into elegant and efficient codes. The reader can download some of them from the link *Softwares* on the webpage

<http://www.facom.ufu.br/~nascimento>

and the full programme in p-code at the same address in order to make tests. As an example, the file `Prim.m` runs with only 21 lines of source code. In total, `stree.m` and its related to programmes 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 introduce the programme `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. In Section 3 we briefly describe how the programme works. It draws full tree stretches automatically with the mouse, and Section 4 gives details and hints about it. Section 5 is devoted to explaining some theorems that we have used to write the programme. Section 6 presents some of the geometrical and analytical ideas that were used to implement `stree.m`, and we finally make our conclusions in Section 7. The full source code will be available in future.

2 Preliminaries

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

Definition 2.1. Consider the tree $T = (V, E)$ with weight function $w : V \rightarrow \mathbb{R}^+$ and 0-1 adjacency matrix a_{ij} . Then T is an MST if it minimises the total cost C given by

$$C = \sum_{i < j} a_{ij}(w_i + w_j) \|V_i - V_j\|, \quad (1)$$

where $\|V_i - V_j\| = \frac{1}{2}(w_i + w_j)|V_i - V_j|$ is the connection cost between terminals V_i and V_j .

REMARK: When $w : V \rightarrow \{1\}$ we have the Euclidean non-weighted case.

For a given set of terminals $V = \{V_1, \dots, V_n\}$ and a weight function $w : V \rightarrow \mathbb{R}^+$, 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 $S = \{S_1, \dots, S_m\}$ to V , namely $\tilde{V} = V \cup S$, we shall find the corresponding $\tilde{T} = (\tilde{V}, \tilde{E})$ such that $\tilde{C} \leq C$. We claim that $\tilde{C} < C$ exactly when $S \neq \emptyset$, providing one chooses a suitable extension $\tilde{w} : \tilde{V} \rightarrow \mathbb{R}^+$ of the weight function.

The following lemma shows how to make this choice when $n = 3$ and $m = 1$ (see Figure 3).

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

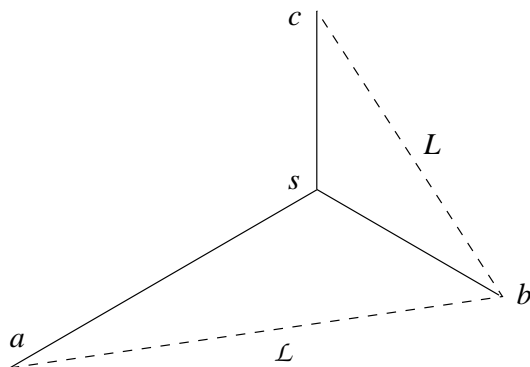


Figure 3: Adding a Steiner point to three weighted terminals.

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

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

Case 1: $b \leq a \leq c$. The law of cosines imply $\ell_1 + \ell_2/2 < \mathcal{L}$ and $\ell_3 + \ell_2/2 < L$. Hence $a\ell_1 + b\ell_2 + c\ell_3 < a\mathcal{L} + cL$. Moreover, $s \leq b$ and $\ell_1 + \ell_2 + \ell_3 < \mathcal{L} + L$. This implies (2).

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

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

q.e.d.

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

Proof. Based upon 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 $\tilde{T} = (\tilde{V}, \tilde{E})$ determined by V and compute the corresponding C . There is a finite number of such trees, hence the least C will determine \tilde{T} 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 $n \geq 3$. The resulting MST $\tilde{T} = (\tilde{V}, \tilde{E})$ 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 \tilde{T} : *Convex hull*, *Maxwell's Theorem*, *Lune* and *Wedge* properties.

3 Getting started

This present section is like a quick manual to the programme. The readers that are more interested in mathematical results can read it through or simply go straight to Section 4. Download `stree.zip` from <http://www.facom.ufu.br/~nascimento/software.html> and extract it in a folder. Enter “stree” at the Matlab prompt. You will get the following message:

Please adjust terminal window to show full picture.
Give a filename to open or press Enter to choose points.

Adjusting the terminal window will prevent figures from hiding it. Now you may either give an existing datafile of terminal points for the tree, or plot one as follows: position the mouse where you want to add a terminal, then either type any number from 0 to 9 or click the mouse left-button.

For the time being, you can enter only integer weights from 1 to 9, namely the number you typed. As a matter of fact, either 0 or the mouse left-button will attribute weight 1 to the new terminal, the only value that is omitted by default (see Figures 4 and 5). However, the terminals can be saved in a text file. Edit arbitrary weights there if you want to.

Some test-files with extension “.txt” can be found in **stree.zip**, and we shall take them here for comments.

Press the Enter key after you either finish plotting points or type the filename *without* extension, for instance **test0**. Figure 4 will appear with the weighted tree of Prim, and Figure 5 is for you to draw a weighted Steiner tree. Our purpose is to find the *cheapest* tree, and Figure 4 can help make good choices.

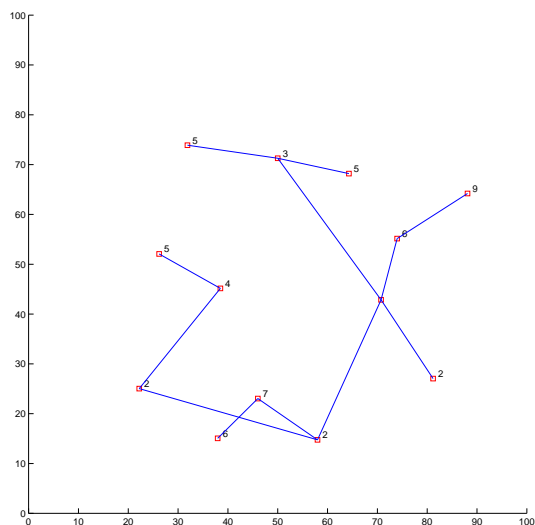


Figure 4: The weighted tree of Prim.

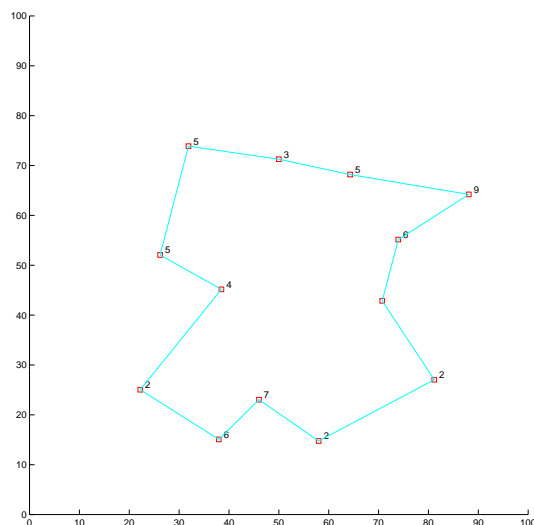


Figure 5: Terminals joined by cyan edges.

Notice that a weighted tree of Prim can have self-intersections, as shown in Figure 4. This never happens to a weighted Steiner tree, because Lemmas 2.1 and 2.2 imply that any self-intersection can be split into two Steiner points in order to lower the cost.

In Figure 5 the cyan polygon is obtained by the Convex Hull and Lune properties described in [10]. It is called “Steiner hull”. From this point on some hints and instructions to proceed are printed in the Matlab terminal window, and all actions will be with the

mouse. For the weighted tree of Prim, its length is printed as follows:

```
The length of a weighted Prim is
ans =
802.0910
```

and so the user can compare this value with the one obtained after a new tree is build. From the *first mouse menu*, now printed in the Matlab terminal window, you can read the drawing options:

```
Press button
l(+l)+r to omit point(s);
r alone to try full tree;
middle for more options.
```

For instance, choose (39, 15) as the first and (65, 70) as the second point with the left button, and then press the right button. A dotted black polygonal will blink to indicate a remaining cyan polygon. You will be left with a group of terminals very likely to give a full Steiner subtree. It will be drawn by pressing the right mouse button alone, no matter the cursor is. See Figures 6 and 7.

From the first to the second point, one goes *counterclockwise* along the black polygon. It is a standard of `stree.m`, but if you mistake it *before* clicking the right button, choose extra point(s) and the programme will treat the latter two as “first” and “second”. Otherwise, if you mistake that standard and realise it only *after* having clicked the right button, it is again no problem. The first menu is re-printed on the screen, and now you can undo your previous step as follows: press the middle button for the *second mouse menu*, namely:

```
Press button
left to return drawing;
right to undo previous step;
middle for retouches.
```

The right button undoes your mistake and you get back again to the first menu.

You might now want to take (75, 55) and then (39, 15) to get a Steiner subtree out of a quadrilateral. That will work out, but Figure 4 does not hint this way. So try instead (75, 55) and (22, 24) and you will get Figure 8. As a matter of fact, our strategy is to build full Steiner subtrees that contain a cheap terminal in order to minimise the total cost. That is why Prim’s weighted tree is not the *only* hint to build the MST.

From the second mouse menu, we can now click on the middle button for retouches. By the *third mouse menu*,

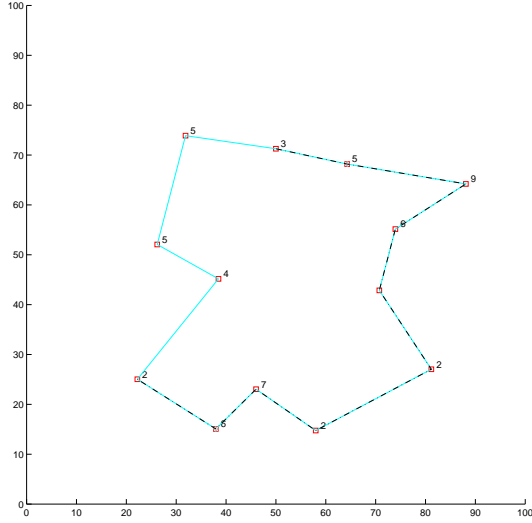


Figure 6: The dotted black polygonal.

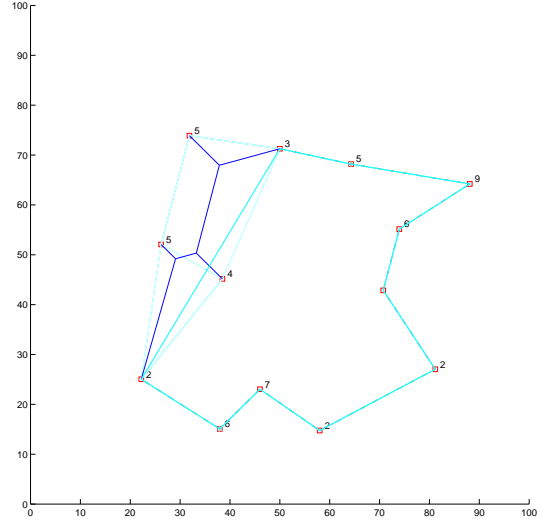


Figure 7: A Steiner subtree.

Thrice Left = Steiner Point
Left + n x Right = Polygonal
Middle = Terminate

which is also the last one, you should click left on (71, 42), (39, 45) and (75, 55) to get a Steiner point out of three terminals (see Figure 9). The order you choose these points is unimportant. Now click left on (90, 62) and then right on (75, 55), or vice-versa. Finally, connect both (65, 68) and (50, 71). The middle mouse button concludes your drawing (see Figure 10) and informs that $Ltree = 598.3593$. Also, a full border is printed when you terminate execution.

It is worthwhile to perform several tests no matter how good our strategies may seem. Figure 11 shows another try on `test0` which however resulted in $Ltree = 646.2392$.

4 Full Tree Stretches

In the previous section we mentioned Figure 11, which contains a group of terminals connected by a full Steiner subtree. This subtree is obtained by the second option of the first mouse menu. Now we present some details and hints, so that the intuitive mind will hardly be wrong at identifying such groups.

When will a subgroup of terminals admit connection by a full Steiner tree? For the time being, we answer this question providing each Steiner point is directly connected to a terminal of the subgroup (see Theorem 5.3 below). In general, it is when you have a “good” zigzag polygonal connecting them. For instance, run “Mksaw” for the terminal data `test1.txt`, and do the same to “stree”. You will get Figures 12 and 13.

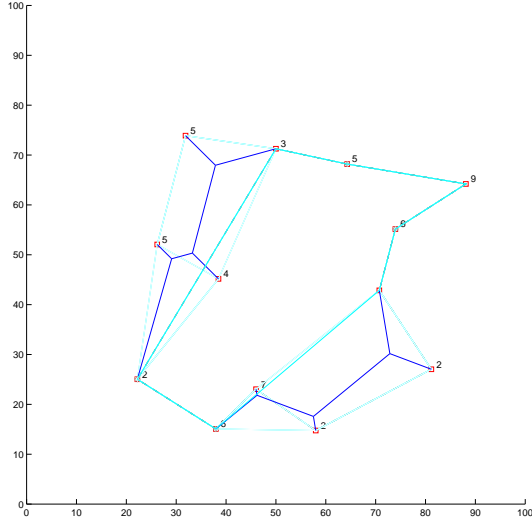


Figure 8: Trying cheap Steiner subtrees.

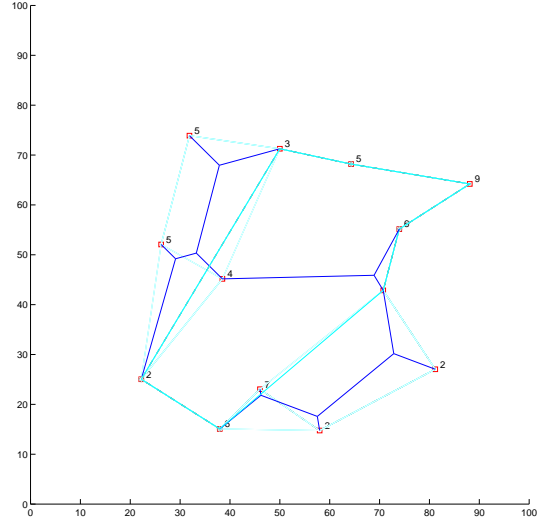


Figure 9: Retouching the last connections.

This example shows that **stree** will give an answer, even if no full tree exists. However, it relies on your guesses. Whenever they are wrong, choose the “undo” option from the second mouse menu, or simply run again “stree” if most of your terminals were a group like in Figure 12. This second case is illustrated in Figures 14 and 15. It shows what is wrong: the zigzag is quite irregular.

Here are some hints to identify good zigzags: the group of points should form a strip, which can scroll in any direction. The strip can be slightly bent or waved, but its width may oscillate even quite a lot. However, despite all hints we give only the practice will really make you recognise the good zigzags very quickly.

The next section discusses some theorems from Geometry and Complex Analysis that made the **stree**-codes very short.

5 Some Theorems used in stree

The following theorems were used to implement **Rprim.m** and **Mksaw.m**, respectively.

Theorem 5.1. *Let S be a finite set of points with at least two elements. In this case, there are $P, Q \in S$ such that the distance between P and Q is maximal. The segment PQ is called the diameter of S .*

Proof. It is already explicitly given by lines 2-3 of the open code **Rprim.m**, and so we shall omit it here.

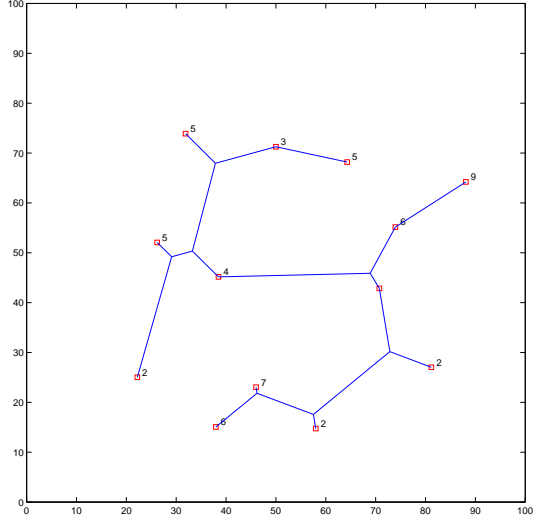


Figure 10: The completed Steiner tree.

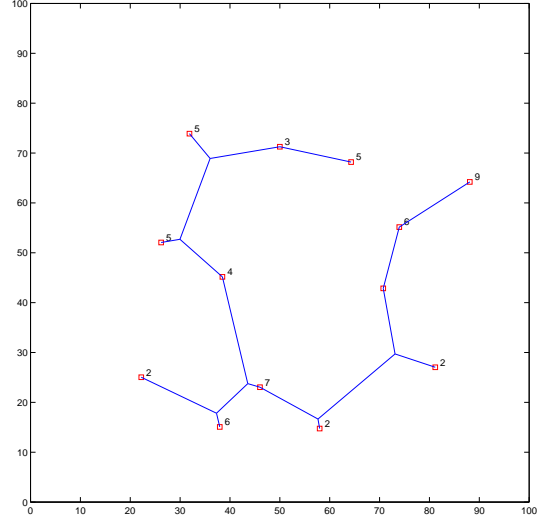


Figure 11: Another try on `test0`.

Theorem 5.2. *Let Q be a quadrilateral with consecutive vertices P_i , $i = 1, \dots, 4$. Let $vec_{i-2} = P_i - P_1$, $i \geq 2$, and $Vec_{i-1} = P_i - P_4$, $i \leq 3$. 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 $\overleftrightarrow{P_1P_3}$ and P_1, P_3 are at opposite sides of $\overleftrightarrow{P_2P_4}$. This is equivalent to the assertion of the Theorem.

Theorem 5.3. *Let A_1, \dots, A_n be $n \geq 3$ terminals in the complex plane \mathbb{C} connected by a minimal full Steiner tree S . Let $V = \{V_1, \dots, V_s\}$ 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 , $\forall k \in \{1, \dots, n\}$. 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

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

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 $\rho_t = r(t) + (V_1 - A_1) \cdot \mathbb{R}_+$ and $\varrho_t = r(t) + (V_1 - A_2) \cdot \mathbb{R}_+$. Let $A = \{A_3, \dots, A_n\}$ and take all positive t such that $(\rho_t \cup \varrho_t) \cap A \neq \emptyset$. They will make a finite set $\{t_3, \dots, t_m\}$ with $m \leq n - 2$.

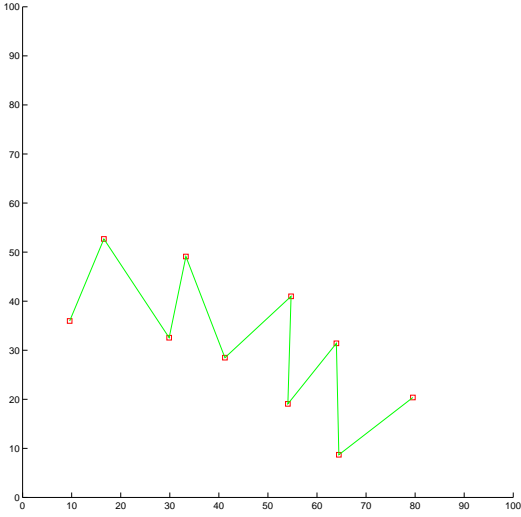


Figure 12: A “good” zigzag.

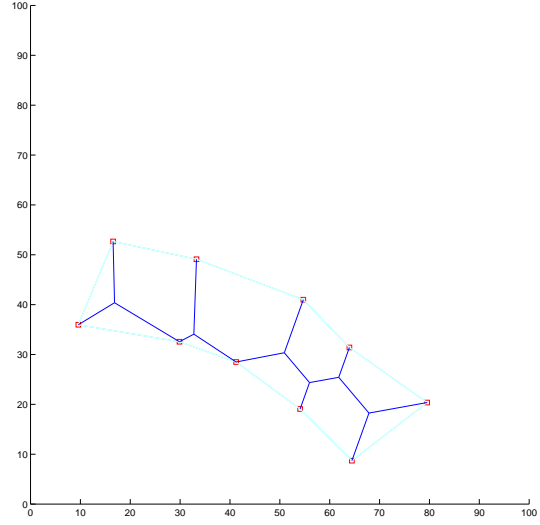


Figure 13: An “almost” full tree.

Since V_1 must be connected with another Steiner point, say V_2 (after re-indexing), then $V_2 = r(t_k)$ for a certain $k \in \{1, \dots, m\}$. 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, \dots, V_s are determined. This concludes the proof of the Theorem.

Our next section explains a bit of `stree.m` and the open codes `Rprim.m` and `Mksaw.m`, which exemplify the applications of the theorems discussed in this present section. They correspond to `rprim.p` and `mksaw.p`, which are two of the files that `stree.zip` contains.

6 Efficient Codes from Geometry and Complex Analysis

Figure 16 describes our pseudocode, which works recursively while the connection matrix is not complete.

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 5. 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.

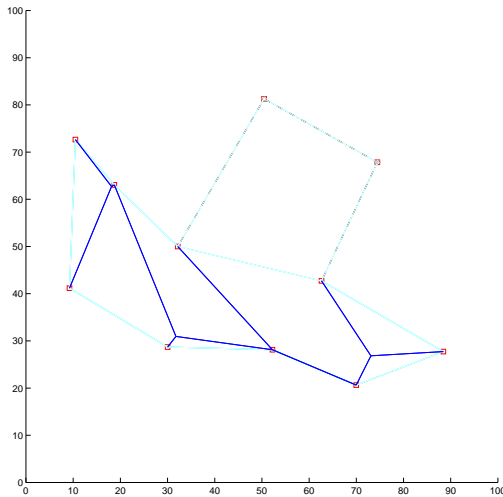


Figure 14: A wrong guess.

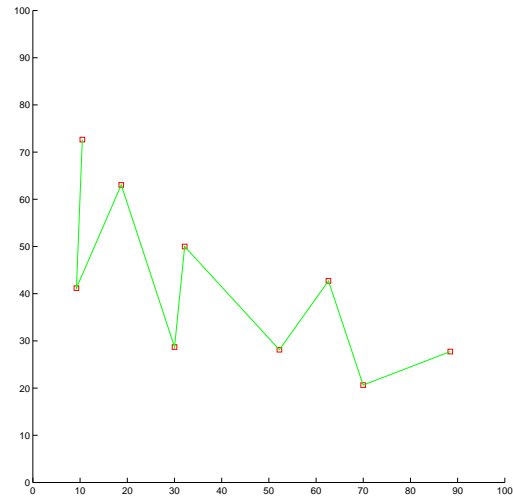


Figure 15: A “bad” zigzag.

stree Pseudocode

Input: Set of terminals

Output: Steiner tree

1. $Pts \leftarrow$ terminals, $Connexion_Matriz \leftarrow 0$
2. $Tree_of_Prim \leftarrow Compute_Tree_of_Prim(Pts)$
3. $Steiner_hull \leftarrow Compute_Steiner_hull(Pts)$
4. While $Connexion_Matriz$ implies Stree non-connected
 - i. $Subset_Pts \leftarrow$ User’s Choice of a Subgroup(Pts)
 - ii. $Subtree \leftarrow Compute_Connection(Subset_Pts)$
 - iii. If Subtree not OK, redo Step i
 - iv. Else $Connexion_Matriz \leftarrow Connection(Subset_Pts)$
5. return Steiner tree (and its length)

Figure 16: The pseudocode of **stree.m** algorithm.

But **stree.m** does not connect them 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 programme will try them back and forth, and even split them in parts.

Of course, **lune.m** and **cvxhull.m** are based upon the Convex Hull and Lune properties described in [10]. They do not give a polygon with zigzags, but with stretches like in Figure 17. Hence, **stree.m** calls **mksaw.m** in order to get the zigzag.

If you want to test these procedures, run “Cvxhull” for the terminal data **test2.txt** 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

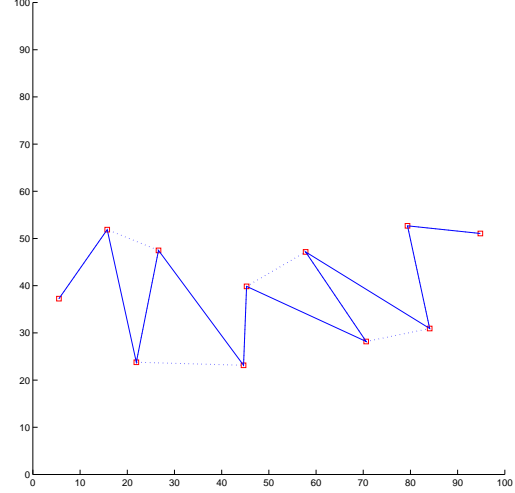
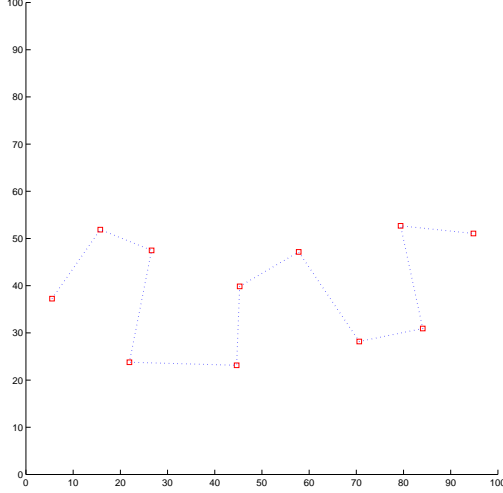


Figure 17: A “01...10” stretch. Figure 18: Rearrangement of Figure 17 by “Mksaw”.

run “Lune” for `test3.txt` and name the output `test4`.

Now we are going to explain the open codes `Rprim.m` and `Mksaw.m`, which will finally generate a zigzag out of `test2.txt` (the initial datafile). According to Theorem 5.1, 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, the open code `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 17. 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)`.

Now look at lines 4 to 9 of `Mksaw.m` and apply Theorem 5.2 to Q . The symbol \times 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 $\mathbb{C} \times \mathbb{R}$, 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 $bc - ad = Im\{vec_1 \cdot \overline{vec_0}\}$. 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 $Im\{vec_1 \cdot \overline{vec_0}\}$ and $Im\{vec_1 \cdot \overline{vec_2}\}$ are opposite. Hence, lines 10-11 from `Mksaw.m` check if Q is convex. In this case, the programme 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 came to the end of the line.

7 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 programme `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 [4]). 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 [8, 18]), and then another option like `stree` to be developed.

Acknowledgement

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] S. Angelopoulos, The node-weighted Steiner problem in graphs of restricted node weights, in: *Algorithm Theory-SWAT*, Springer, Berlin and Heidelberg, 2006, pp. 208-219.
- [2] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler and T. Roughgarden, The price of stability for network design with fair cost allocation, in: *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, 2004, pp. 295-304.
- [3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela and M. Protasi, *Complexity and approximation: combinatorial optimization problems and their approximability properties*, Springer-Verlag, Berlin and Heidelberg, 1999.
- [4] D. Chen, D.-Z. Du, X.-D. Hu, G.-H. Lin, L. Wang and G. Xue, Approximations for Steiner trees with minimum number of Steiner points, *Theoretical Computer Science* **262** (2001), 83-99.
- [5] E.D. Demaine, M. Hajiaghayi and P.N. Klein, Node-weighted Steiner tree and group Steiner tree in planar graphs, in: *Automata, Languages and Programming*, Springer, Berlin and Heidelberg, 2009, pp. 328-340.
- [6] B. Dilkina and C.P. Gomes, Solving connected subgraph problems in wildlife conservation, in: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Springer, Berlin and Heidelberg, 2010, pp. 102-116.
- [7] D.-Z. Du and F.K. Hwang, A proof of the Gilbert-Pollak conjecture on the Steiner ratio, *Algorithmica* **7** (1992), 121-135.
- [8] U. Fößmeier and M. Kaufmann, Solving rectilinear Steiner tree problems exactly in theory and practice, in: *Proceedings of the 5th European Symposium on Algorithms*, Springer-Verlag, Berlin and Heidelberg, 1997, pp. 171-185.
- [9] M.R. Garey, R.L. Graham and D.S. Johnson, The complexity of computing Steiner minimal trees, *SIAM Journal of Applied Mathematics* **32** (1977), 835-859.
- [10] E.N. Gilbert and H.O. Pollak, Steiner minimal trees, *SIAM Journal of Applied Mathematics* **16** (1968), 1-29.
- [11] S. Guha, A. Moss, J.S. Naor and B. Schieber, Efficient recovery from power outage, in: *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, ACM, 1999, pp. 574-582.
- [12] X.-D. Hu, T.-P. Shuai, X. Jia and M.-H. Zhang, Multicast routing and wavelength assignment in WDM networks with limited drop-offs, in: *IEEE INFOCOM*, 2004, http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1354520
- [13] N. Innami, B.H. Kim, Y. Mashiko and K. Shiohama, The Steiner ratio conjecture of Gilbert-Pollak may still be open, *Algorithmica* **57** (2010), 869-872.
- [14] X. Jia, X.-D. Hu, M. Lee, D.-Z. Du and J. Gu., Optimization of wavelength assignment for QoS multicast in WDM networks, *IEEE Trans. on Communications* **49** (2001), 341-350.
- [15] X. Li, X.-H. Xu, F. Zou, H. Du, P. Wan, Y. Wang and W. Wu, A PTAS for node-weighted Steiner tree in unit disk graphs, in: *Combinatorial Optimization and Applications*, Springer, Berlin and Heidelberg, 2009, pp. 36-48.
- [16] R.C. Prim, Shortest connection networks and some generalizations, *Bell System Tech-*

- nical Journal* **36** (1957), 1389-1401.
- [17] L.H. Sahasrabuddhe and B. Mukherjee, Multicast routing algorithms and protocols: a tutorial, *IEEE Network* **14** (2001), 90-102.
 - [18] M. Zachariasen, *Rectilinear full Steiner tree generation*, Technical Report DIKU-TR-97/29, University of Copenhagen, 1997, <http://www.diku.dk>
 - [19] F. Zou, X. Li, S. Gao and W. Wu, Node-weighted Steiner tree approximation in unit disk graphs, *Journal of Combinatorial Optimization* **18** (2009), 342-349.