GENERALIZED \textit{st}-NUMBERING FOR SIMPLY CONNECTED GRAPHS

Letícia Rodrigues Bueno
CMCC, Universidade Federal do ABC (UFABC)
Santo André – SP – Brazil
leticia.bueno@ufabc.edu.br

Rodrigo de Alencar Hausen, Candido Ferreira Xavier Mendonça
EACH, Universidade de São Paulo (USP)
São Paulo – SP – Brazil
hausen@usp.br, cfxavier@usp.br

ABSTRACT

Given an undirected biconnected graph \( G = (V, E) \) and an edge \( st \in E \), an \( st \)-numbering is a numbering of the vertices of \( G \) such that \( s \) is numbered 1, \( t \) is numbered \( n \), and every vertex different from \( s \) and \( t \) is adjacent both to a lower-numbered and to a higher-numbered vertex. Algorithms for determining an \( st \)-numbering, in general, run in time \( O(|V| + |E|) \) and are restricted to biconnected graphs. Therefore, for general graphs, applications that need \( st \)-numberings for its maximal biconnected subgraphs have to preprocess the input graph to identify those subgraphs. We present an algorithm that provides a numbering of the vertices of \( G \), which is trivially transformed into an \( st \)-numbering for every maximal biconnected subgraph of \( G \), that also runs in time \( O(|V| + |E|) \), but it provides the numbering of the vertices without a preprocessing step. We show an application for graph drawing in a linear layout.

KEYWORDS. \textit{st}-numbering, graph drawing, crossing number

Main area: Theory and Algorithms in Graphs

1. Introduction

Let \( G = (V, E) \) be an undirected graph with \( n \) vertices and \( m \) edges. A numbering of \( V \) is an assignment of a unique number in the range 1, \ldots, \( n \) to every vertex of \( G \). A graph \( G \) is biconnected if there is no vertex whose removal disconnects \( G \); if there exists \( v \in V \) such that its removal from \( G \) disconnects the graph, then \( G \) is non-biconnected. Given an edge \( st \) of \( G \), an \( st \)-numbering (also known as bipolar orientation or \( st \)-orientation [Papamanthou and Tollis 2008]) is a numbering of the vertices of \( G \) such that \( s \) receives number 1, \( t \) receives number \( n \), and every vertex different from \( s \) and \( t \) is adjacent both to a lower-numbered and to a higher-numbered vertex (see an example in Figure 1 (c)).

There are a variety of applications of \( st \)-numbering in graph drawing, such as orthogonal drawings, hierarchical drawings, visibility representations, planarity-testing, and graph planarization. Also, the length of the longest \( st \)-path (an \( st \)-path is a directed path from \( s \) to \( t \)) has been studied [Papamanthou and Tollis 2008, Sadasivam and Zhang 2009] for applications such as network routing and area-bound graph drawing algorithms.

The \( st \)-numbering concept was introduced in [Lempel et al. 1967] as part of an efficient planarity-testing algorithm. The authors proved that an \( st \)-numbering exists if and only if the graph is biconnected. Their \( st \)-numbering algorithm had a time complexity of \( O(nm) \). A more efficient \( O(n + m) \) time algorithm for finding an \( st \)-numbering was devised in [Even and Tarjan 1976]. Later, a simplified version of the algorithm was proposed by Ebert [Ebert 1983]. The methods in [Even and Tarjan 1976, Ebert 1983] decompose \( G \) into a collection of edge-disjoint paths and
then process the paths to produce an $st$-numbering. Tarjan [Tarjan 1986] proposed a simpler algorithm that bypasses the path decomposition phase. Additionally, a parallel algorithm is described in [Maon et al. 1986] and another linear-time algorithm in [Brandes 2002].

Even and Tarjan [Even and Tarjan 1976] had suggested a combination of a block-finding algorithm and an $st$-numbering algorithm but they have not given any details on it. Since then, for applications in general graphs that use an $st$-numbering, researchers have run the first algorithm followed by the second one. For instance, some methods of orthogonal graph drawing [Biedl and Kant 1998, Calamoneri et al. 1999, Calamoneri and Petreschi 1995, Di Battista et al. 1997] divide the graph $G$ in blocks, which are defined as its maximal biconnected subgraphs $C_1, C_2, \ldots, C_k$, and then find an $st$-numbering for each block. Each block is drawn and the whole drawing of $G$ is obtained by properly connecting the drawings of all different blocks. Another approach is to add dummy edges to $G$ until $G$ is biconnected, and then apply an $st$-numbering method.

In this work, we present an algorithm that provides a numbering of the vertices of a connected graph $G$ that can be straightforwardly transformed into an $st$-numbering for every maximal biconnected subgraph of $G$. This generalized $st$-numbering of $G$ is returned without the additional phase to divide the graph into blocks. Moreover, the numbering provided by the algorithm has the property that every vertex in more than one block, which is the case of cut-vertices, is numbered in such a way that it can be properly put into the $st$-numberings of its blocks.

The notion of a generalized $st$-numbering is not completely new. Indeed, for planar graphs, [Didimo and Pizzonia 2003] presented an $O(n^{3/2})$ algorithm that computes an upward orientation with the minimum number of sources and sinks. This kind of orientation is equivalent to a generalization of an $st$-orientation to simply connected graphs. We extend the notion of generalized $st$-orientations to non-planar graphs and provide a linear-time algorithm that finds such an orientation.

Our work is organized as follows. In Section 2, we present some preliminary definitions. In Section 3, we discuss the algorithm and Section 4. contains some experimental results. Concluding remarks are presented in Section 5.

2. Preliminaries

In this section, we introduce some notation that will be used to construct a generalized $st$-numbering algorithm. For basic concepts such as graph, path, cycle, etc., we borrow the definitions from [Even and Tarjan 1976].

Let $G = (V, E)$ be an undirected graph with $|V| = n$ and $|E| = m$. A graph is said to be connected if there is a path between any pair of vertices. If $v$ is a vertex of $G$ whose removal disconnects $G$, we call $v$ a cut-vertex. Similarly, if there is an edge $e \in E$ whose removal disconnects $G$, we call $e$ a cut-edge. Notice that a cut-edge does not belong to any cycle of $G$.

A connected graph that has no cut-vertex is called a biconnected graph. A maximal biconnected subgraph of $G$ is called a block of $G$. Notice that a vertex may belong to more than one block of $G$, but every edge belongs to exactly one block.

In order to develop the algorithm, we need to review some of the properties of both depth-first search algorithm and Tarjan’s $st$-numbering algorithm [Tarjan 1986]. Assume $G$ is connected. Suppose we carry out a depth-first search in $G$, starting at vertex $s$ and traversing first the edge $st$. The search traverses every edge of $G$, orienting them in the direction along which the search advances. The resulting directed edges belong to two types: tree edges, which define a spanning tree rooted at $s$ and having paths from $s$ to every vertex, and back edges, which lead from a vertex to one of its proper ancestors in the spanning tree.
Suppose we number the vertices from 1 to n in the order they are visited during the depth-first search. This numbering is a preorder numbering of the spanning tree [Knuth 1974]. We shall denote the preorder number of a vertex v by pre(v). For each vertex v, let low(v) be the vertex of smallest number reachable from v by a path consisting of zero or more tree edges followed by at most one back edge. The vertex low(v) is guaranteed to be an ancestor of v in the spanning tree (see an example in Figure 1). The preorder number and the low values can be computed in linear time by a single depth-first search [Tarjan 1986].

Tarjan’s st-numbering algorithm [Tarjan 1986] finds an st-numbering of a graph in time $O(n + m)$ and consists of two phases. First, a depth-first search is executed and the preorder numbering and low values are computed, as well as the parent $p(v)$ of each vertex v in the spanning tree. In the second phase, a list $L$ of the vertices is constructed such that the vertices are numbered in the order they occur in $L$ resulting in an st-numbering. This is achieved by performing a preorder traversal of the spanning tree, attributing a minus sign to every ancestor u of a vertex v if u precedes v in $L$, or a plus sign if u succeeds v in $L$. Initially $L = [s, t]$ and $sign(s) = minus$. Each vertex $v \notin \{s, t\}$ is inserted into $L$ following the preorder previously obtained, as per procedure INSERT_VERTEX (see an example in Figure 1 and Table 1).

```
1 procedure INSERT_VERTEX(L, v)
2     if sign(low(v))=minus then
3         insert v in L before p(v);
4         sign(p(v)) := plus;
5     else if sign(low(v))=plus then
6         insert v in L after p(v);
7         sign(p(v)) := minus;
8 end.
```

3. A Generalized st-Numbering Algorithm

Since an st-numbering exists if and only if the graph is biconnected [Lempel et al. 1967], we propose an algorithm that returns an st-numbering for a biconnected graph $G$ and, if $G$ is non-biconnected, the algorithm returns a generalized st-numbering. By a generalized st-numbering,
Figure 2: The first phase of GEN_ST_NUMBER: finding blocks. In (b) a spanning tree of the graph $H$ where vertices are numbered in preorder. The letters labelling vertices are the low values. Tree edges are solid, back edges are dashed.

we mean that an edge $s_it_i$ is selected for each block $C_i$ such that every other vertex $v \neq s_it_i$ is adjacent both to a lower-numbered and to a higher-numbered vertex. For instance, Figure 3 (b) illustrates an st-numbering of the blocks of the graph $H$ in Figure 2 (a). Notice that the properties of the st-numbering are valid for each block, although some cut-vertices exist and belong to several blocks.

Our GEN_ST_NUMBER algorithm constructs a generalized st-numbering for connected graphs by replacing the depth-first search step in Tarjan’s st-numbering algorithm [Tarjan 1986] by a block-finding algorithm, done in $O(n + m)$ time as well [Tarjan 1972]. The block-finding algorithm in [Tarjan 1972] is a variation of the depth-first search and returns not only the blocks of $G$ numbering of the whole graph $G$ if $G$ is biconnected. Otherwise, the algorithm returns a numbering such that all blocks of $G$ are locally st-numbered and each vertex has only one number, even for cut-vertices that are in more than one block.

The algorithm GEN_ST_NUMBER works by attributing an edge $s_it_i$ to every block $C_i$ such that the position of each vertex $v \in C_i$ in the list $L$ depends on the position of vertices $s_i$ and $t_i$ in $L$. The idea is similar to the one used in Tarjan’s st-numbering algorithm: each vertex $v \in C_i$ must be placed between $p(v)$ and low($v$) so that $v$ is adjacent both to a lower-numbered and to a higher-numbered vertex in the numbering.

Since an edge $s_it_i$ in a block $C_i$ cannot be in another block, for each vertex $v \in C_i$ to be added in the list $L$, if $s_it_i$ was already defined, then part of the block $C_i$ is already in the list $L$ as well. Thus, $v$ can be simply added to $L$ in its correct position among the vertices of $C_i$. On the other hand, if $s_it_i$ was not defined yet, then there are two possibilities:

1. If $v = \text{low}(v)$, then $C_i$ is a cut-edge. Add $v$ in $L$ after $p(v)$ and set the edge $s_it_i$ of $C_i$ as $s_i = p(v)$ and $t_i = v$;
2. If $v \neq \text{low}(v)$, then add $v$ after low($v$) and set the edge $s_it_i$ of $C_i$ as $s_i = \text{low}(v)$ and $t_i = v$. Add all the other vertices of $C_i$ between $s_i = \text{low}(v)$ and $t_i = v$ according to the
Figure 3: In (a) the blocks $C_1,\ldots,C_7$ of $H$ obtained by the first phase of GEN_ST_NUMBER for the graph $H$ (Figure 2). Cut-points are gray. In (b) the $st$-numbering obtained from List $L$ (Table 2).

rule in Tarjan’s $st$-numbering algorithm given in the procedure INSERT_VERTEX.

The insertion of $v$ after $\text{low}(v)$ into $L$ joins the vertices in two blocks. Clearly, any two blocks do not share more than one cut-vertex, therefore there are no edge-crossings between any blocks. The procedure GEN_ST_NUMBER implements the algorithm. The routine FIND_BLOCK returns the block containing an edge $st$.

```plaintext
procedure GEN_ST_NUMBER(v)
L := [s,t];
current := FIND_BLOCK(s,t);
current(s,t) := s,t;
sign(s) := minus;
for each v in preorder do
    if v is in the block current then
        INSERT_VERTEX(L,v);
    else
        if v is in a block i that has s,t already defined then
            current := i;
            sign(s) := minus;
            INSERT_VERTEX(L,v);
        else if v <> low(v) then
            insert v to the right low(v);
            current := FIND_BLOCK(v,low(v));
            current(s,t) := low(v),v;
            sign(s) := minus;
        else
            insert v to the right p(v);
            current := FIND_BLOCK(v,p(v));
            current(s,t) := p(v),v;
```

2552
Table 2: List $L$ generated by the second phase of \texttt{gen.st.number} for the graph $H$ (Figure 2). The most recently inserted vertex is underlined. Vertices $s,t_i$ in each block $C_i$ are in bold. Irrelevant signs are omitted.

<table>
<thead>
<tr>
<th>Vertex Added</th>
<th>List $L$</th>
<th>Current Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h$</td>
<td>$k-, i$</td>
<td>$C_1$</td>
</tr>
<tr>
<td>$a$</td>
<td>$k-, i, h-$</td>
<td>$C_2$</td>
</tr>
<tr>
<td>$b$</td>
<td>$k-, i, h-, a$</td>
<td>$C_3$</td>
</tr>
<tr>
<td>$c$</td>
<td>$k-, i, h-, b-, a+$</td>
<td>$C_4$</td>
</tr>
<tr>
<td>$d$</td>
<td>$k-, i, h-, b-, c+, a+$</td>
<td>$C_4$</td>
</tr>
<tr>
<td>$e$</td>
<td>$k-, i, h-, b-, d+, c+, a+$</td>
<td>$C_4$</td>
</tr>
<tr>
<td>$f$</td>
<td>$k-, i, h-, e, d-, f, c+, a+$</td>
<td>$C_4$</td>
</tr>
<tr>
<td>$g$</td>
<td>$k-, i, h-, e, d-, f, g, c+, a+$</td>
<td>$C_4$</td>
</tr>
<tr>
<td>$r$</td>
<td>$k-, i, h-, e, d-, f, g, c+, a+$</td>
<td>$C_4$</td>
</tr>
<tr>
<td>$q$</td>
<td>$k-, i, h-, b-, q, r+, e, d-, f, g, c+$</td>
<td>$C_7$</td>
</tr>
<tr>
<td>$s$</td>
<td>$k-, i, h-, e, d-, f, g, c+$</td>
<td>$C_7$</td>
</tr>
<tr>
<td>$t$</td>
<td>$k-, i, h-, e, d-, f, g, c+$</td>
<td>$C_7$</td>
</tr>
<tr>
<td>$u$</td>
<td>$k-, i, h-, e, d-, f, g, c+$</td>
<td>$C_7$</td>
</tr>
<tr>
<td>$m$</td>
<td>$k-, i, h-, b, u, t, s, q, r, e, d, f, g, c, a-$</td>
<td>$C_5$</td>
</tr>
<tr>
<td>$n$</td>
<td>$k-, i, h-, b, u, t, s, q, r, e, d, f, g, c, a-$</td>
<td>$C_5$</td>
</tr>
<tr>
<td>$o$</td>
<td>$k-, i, h-, b, u, t, s, q, r, e, d, f, g, c, a-$</td>
<td>$C_5$</td>
</tr>
<tr>
<td>$p$</td>
<td>$k-, i, h-, b, u, t, s, q, r, e, d, f, g, c, a-$</td>
<td>$C_5$</td>
</tr>
<tr>
<td>$j$</td>
<td>$k-, j, i+$</td>
<td>$C_1$</td>
</tr>
<tr>
<td>$l$</td>
<td>$k-, j, i+$</td>
<td>$C_1$</td>
</tr>
</tbody>
</table>
end.

The algorithm GEN_ST_NUMBER determines an st-numbering of the blocks and properly inserts them in the sequence, resulting in a generalized st-numbering. If \( G \) is biconnected, then the block reorganization operation returns an st-numbering of the entire graph.

If the graph is a tree, the obtained numbering is not an st-numbering as defined in [Lempel et al. 1967] nor an generalized st-numbering, because each block is only an edge. Even though, the algorithm returns a numbering, this numbering is not very much useful; however, for most applications of st-numberings it does not matter; for instance, it does not make sense to test the planarity of a tree since trees are planar. A st-numbering has been used in graphs more dense than trees since it requires biconnected graphs. Thus, a generalized st-numbering is particularly useful for those graphs that can contains some bridges and cut-vertices.

Since we replaced the depth-first search by the finding-blocks algorithm in [Tarjan 1972], which is a variation of the depth-first search, the complexity of GEN_ST_NUMBER is still \( O(n+m) \). The second phase to organize the vertices in \( L \) requires a structure to keep the edges st in each block. In the worst case, when \( G \) is a tree and all edges are cut-edges, the structure has a space complexity of \( n-1 = O(n) \). The list \( L \) must be doubly linked to facilitate insertions and, finally, a pointer to the position of each vertex in \( L \) can be stored in the same structure that contains the preorder numbers, the low values and the parent of each vertex in the spanning tree. Therefore, as in Tarjan’s algorithm, the second phase of GEN_ST_NUMBER has time complexity of \( O(n) \). The space complexity is \( O(n) \) as well.

4. Experimental Results

A drawing \( D \) of a graph \( G \) is optimum if no other drawing of \( G \) has less edge-crossings than \( D \). The edge-crossing number of an optimum drawing is called the crossing number of \( G \) and it is denoted by \( cr(G) \). A graph \( G \) is planar if \( cr(G) = 0 \). The decision problem associated to determining the crossing number of a graph is NP-Complete [Garey and Johnson 1983]. The crossing number problem has many applications in VLSI and printed-circuit board designs.

In a linear layout of \( G \), the vertices of \( G \) are distributed in a spine (a straight line), the edges are drawn as semicircles in one of the two sides (we call them pages); where every edge is completely contained in one of the two pages (see Figure 4). According to Nicholson [Nicholson 1968], the edge-crossing number in a linear layout is exactly equal to the edge-crossing number of the graph \( cr(G) \). Thus, the simplified structure of a linear layout of a graph can help to determine the crossing number of graphs. However, the crossing number problem in a linear layout is still NP-Complete [Chung et al. 1987] even if the order of vertices in the spine is given [Masuda et al. 1990].

In a linear layout, there is an exponential number of solutions to check. If the position of the vertices in the spine is given, and one has to just decide on which page to draw every edge, there are \( 2^{m-1} \) possible solutions. Since there are \( (n-1)!/2 \) possible orders for the vertices along the spine, the total number of possible solutions is \( (n-1)! \cdot 2^{m-2} \). Since determining the crossing number \( cr(G) \) of a graph \( G \) in a linear layout is a combination of a suitable order of the vertices in the spine and a suitable configuration of the edges on the two pages, some strategies have been studied to determine the order of the vertices along the spine. We developed a heuristic algorithm based on Asynchronous Teams [de Souza and Talukdar 1993, Talukdar 1998] to minimize the number of edge-crossings of graphs in a linear layout. We empirically show a correlation between finding an generalized st-numbering of a graph and minimizing the number of edge-crossings in a linear layout. Three strategies were used to determine the order of the vertices along the spine: an generalized st-numbering, a preorder numbering and a random order. Figure 5 and Figure 6 display the number of edge-crossings obtained for some random graphs related in the literature [Goldschmidt and Takvorian 1994, Cimikowski 1995]. Notice that, in all cases, when an
generalized \textit{st}-numbering is used to determine the order of the vertices, the edge-crossing number is smaller than in the other strategies.

In applications to minimize the crossing number of graphs in a linear layout, a generalized \textit{st}-numbering can be particularly useful. For instance, notice the graph $H$ drawn in a linear layout (see Figures 3 (b) and 4). The generalized \textit{st}-numbering naturally gives an order of the blocks of the graph so that the edges between vertices in the same block can be drawn without crossing edges between vertices in other blocks. Therefore, we can restrict the work of minimizing the edge-crossings to the edges within each block.

![Figure 4: Linear layout of the graph $H$. Vertices in the spine are positioned according to the generalized \textit{st}-numbering in Figure 3 (b).](image)

Figure 5: Number of crossings to different ways of positioning of the vertices.

5. Conclusions

In this paper, we presented an algorithm to determine a generalized \textit{st}-numbering for simply connected graphs. The algorithm returns an \textit{st}-numbering for biconnected graphs and, for non-biconnected graphs, it returns a numbering such that all blocks are \textit{st}-numbered.

The algorithm can be immediately used by applications that need \textit{st}-numberings of graphs such as crossing number minimization, algorithms for orthogonal drawing [Biedl and Kant 1998,
Calamoneri and Petreschi 1995, Calamoneri et al. 1999, Di Battista et al. 1997] and planarity testing algorithms [Lempel et al. 1967]. The generalized \textit{st}-numbering used to determine the order of vertices in the spine of a linear layout showed to be as good as an order determined by a hamiltonian cycle. However, since determining if a graph has a hamiltonian cycle is an NP-Complete problem [Karp 1972], even when a hamiltonian path is given as part of the instance [Papadimitriou and Steiglitz 1976], the \textit{st}-numbering is a superior strategy.

It is simple and straightforward to get an \textit{st}-numbering of only one block in a generalized \textit{st}-numbering. It can be obtained by numbering the vertices of the block in the order they appear in the list $L$, ignoring every other vertices.

6. Acknowledgments

The research work of the first author was supported by National Counsel of Technological and Scientific Development (CNPq) under the grant PDJ 150605/2010-2. The third author acknowledges partial support for this research from CNPq grant n. 308462/2007-6. Thanks to three anonymous reviewers for their comments.

References


