GENERALIZED 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. 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.

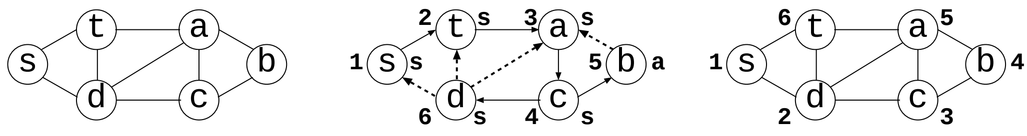
2. Preliminaries

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-numberig 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.



(a) A graph G (b) A spanning tree of G (c) An st-numbering of G

Figure 1: In (b) the vertices of a graph G (a) are numbered in preorder. The *low* values are denoted by letters. Tree edges are solid, back edges are dashed. An st-numbering of G is shown in (c).

Table 1: List L generated by the second phase of Tarjan's st-numbering algorithm [Tarjan 1986] for the graph G (Figure 1 (a)). The st-numbering is depicted in Figure 1 (c).

Vertex Added	List L
	$s-, t$
a	$s-, a, t+$
c	$s-, c, a+, t+$
b	$s-, c-, b, a+, t+$
d	$s-, d, c+, b, a+, t+$

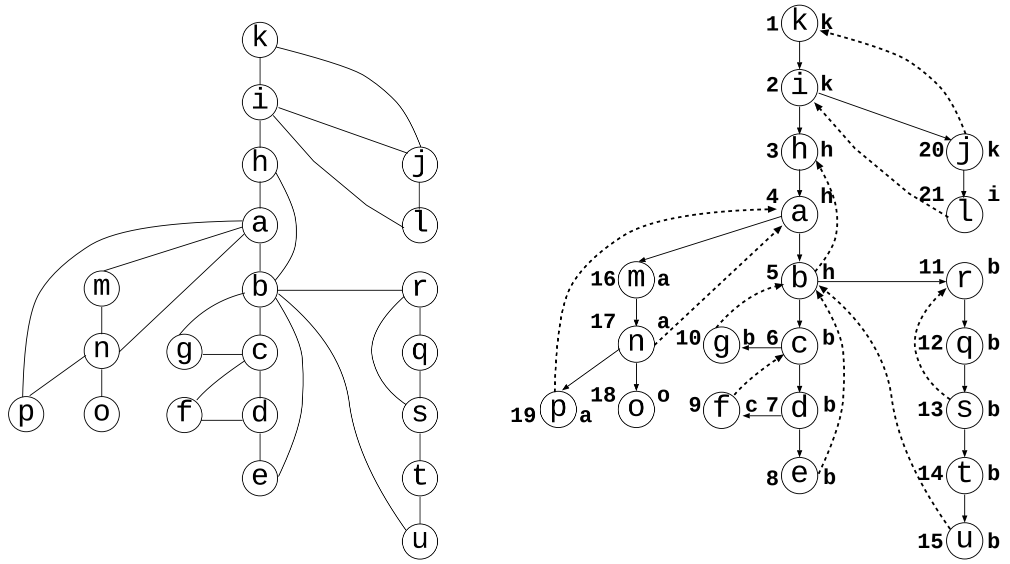
Suppose we number the vertices from 1 to n in the order they are visited during the depth-first search. This numbering is a preorder numbering of the spanning tree [Knuth 1974]. We shall denote the preorder number of a vertex v by $pre(v)$. For each vertex v, let $low(v)$ be the vertex of smallest number 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 succedes 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).

```
procedure INSERT_VERTEX(L,v)
   if sign(low(v))=minus then
      insert v in L before p(v);
      sign(p(v)) := plus;
   else if sign(low(v))=plus then
      insert v in L after p(v);
      sign(p(v)) := minus;
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,



(a) A non-biconnected graph H (b) A spanning tree of H

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.

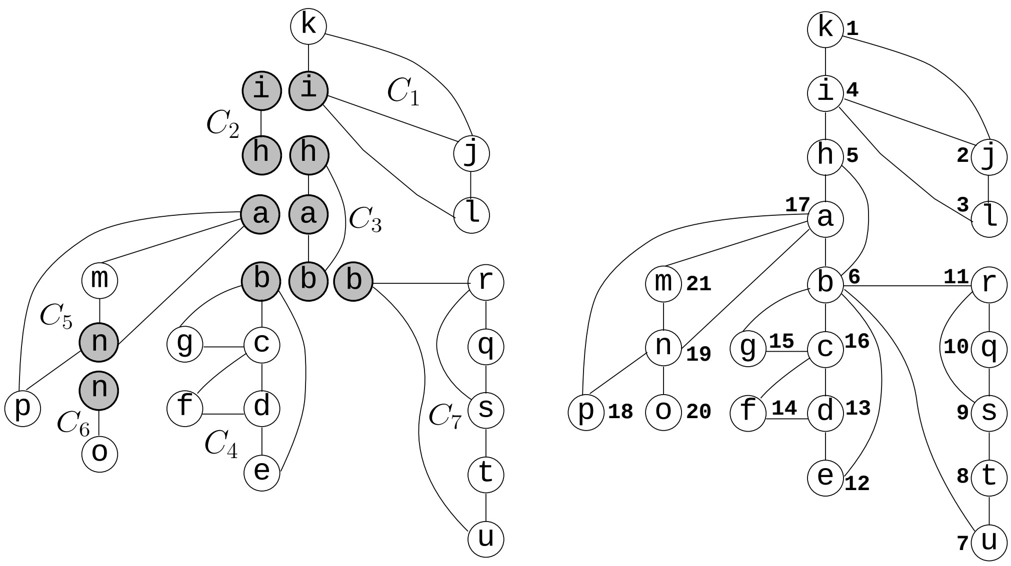
we mean that an edge s_it_i is selected for each block C_i such that every other vertex $v \neq s_i, t_i$ is adjacent both to a lower-numbered and to a higher-numbered vertex. For instance, Figure 3 (b) illustrates an st-numbering of the blocks of the graph H in Figure 2 (a). Notice that the properties 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 the graph but also the values $pre(v)$, $low(v)$ and $p(v)$ needed in the st-numbering phase. Then we modify the second phase of Tarjan's st-numbering algorithm such that the algorithm returns an st-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 $v = low(v)$, 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 $v \neq low(v)$, then add v after $low(v)$ and set the edge s_it_i of C_i as $s_i = low(v)$ and $t_i = v$. Add all the other vertices of C_i between $s_i = low(v)$ and $t_i = v$ according to the



(a) Blocks of H (b) An st-numbering of H

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 $low(v)$ 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.

```
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;
```

Table 2: List L generated by the second phase of `GEN_ST_NUMBER` for the graph H (Figure 2). The most recently inserted vertex is underlined. Vertices $s_i t_i$ in each block C_i are in bold. Irrelevant signs are omitted.

Vertex Added	List L	Current Block
	$\mathbf{k}-, \mathbf{i}$	C_1
h	$k-, \mathbf{i}, \underline{\mathbf{h}}$	C_2
a	$k-, i, \mathbf{h}-, \underline{\mathbf{a}}$	C_3
b	$k-, i, \mathbf{h}-, \underline{b}, \mathbf{a}+$	C_3
c	$k-, i, h-, \mathbf{b}-, \underline{\mathbf{c}}, a+$	C_4
d	$k-, i, h-, \mathbf{b}-, \underline{d}, \mathbf{c}+, a+$	C_4
e	$k-, i, h-, \mathbf{b}-, \underline{e}, d+, \mathbf{c}+, a+$	C_4
f	$k-, i, h-, \mathbf{b}-, e, d-, \underline{f}, \mathbf{c}+, a+$	C_4
g	$k-, i, h-, \mathbf{b}-, e, d-, f, \underline{g}, \mathbf{c}+, a+$	C_4
r	$k-, i, h-, \mathbf{b}-, \underline{\mathbf{r}}, e, d-, f, g, c+, a+$	C_7
q	$k-, i, h-, \mathbf{b}-, \underline{q}, \mathbf{r}+, e, d-, f, g, c+, a+$	C_7
s	$k-, i, h-, \mathbf{b}-, \underline{s}, q+, \mathbf{r}+, e, d-, f, g, c+, a+$	C_7
t	$k-, i, h-, \mathbf{b}-, \underline{t}, s+, q+, \mathbf{r}+, e, d-, f, g, c+, a+$	C_7
u	$k-, i, h-, \mathbf{b}-, \underline{u}, t+, s+, q+, \mathbf{r}+, e, d-, f, g, c+, a+$	C_7
m	$k-, i, h-, b, u, t, s, q, r, e, d, f, g, c, \mathbf{a}-, \underline{\mathbf{m}}$	C_5
n	$k-, i, h-, b, u, t, s, q, r, e, d, f, g, c, \mathbf{a}-, \underline{n}, \mathbf{m}+$	C_5
o	$k-, i, h-, b, u, t, s, q, r, e, d, f, g, c, a-, n, \underline{\mathbf{o}}, \mathbf{m}+$	C_6
p	$k-, i, h-, b, u, t, s, q, r, e, d, f, g, c, \mathbf{a}-, \underline{p}, n, o, \mathbf{m}+$	C_5
j	$\mathbf{k}-, \underline{j}, \mathbf{i}+, h-, b, u, t, s, q, r, e, d, f, g, c, a, p, n, o, m$	C_1
l	$\mathbf{k}-, j-, \underline{l}, \mathbf{i}+, h-, b, u, t, s, q, r, e, d, f, g, c, a, p, n, o, m$	C_1

```
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 facilite 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 $(n-1)!\ 2^{m-2}$. 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 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 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 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.

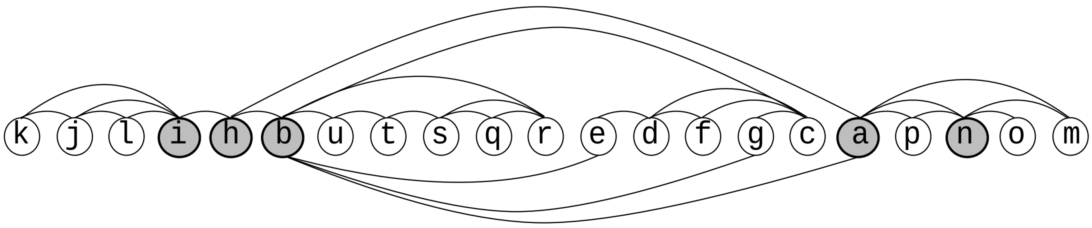


Figure 4: Linear layout of the graph H. Vertices in the spine are positioned according to the generalized st-numbering in Figure 3 (b).

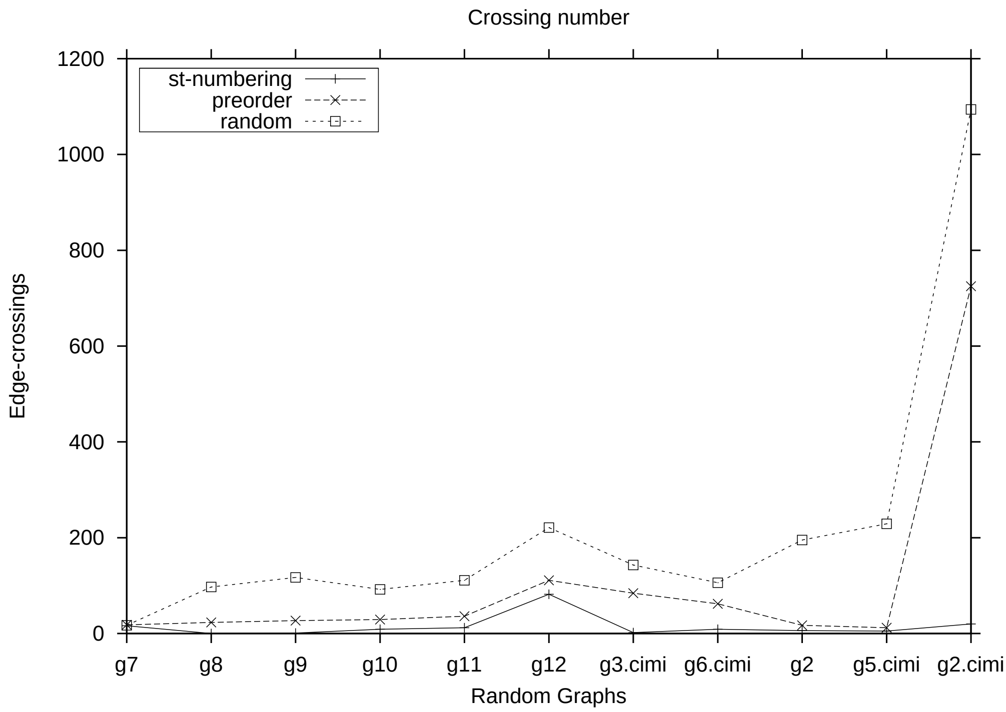


Figure 5: Number of crossings to different ways of positioning of the vertices.

5. Conclusions

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

The algorithm can be immediately used by applications that need st-numberings of graphs such as crossing number minimization, algorithms for orthogonal drawing [Biedl and Kant 1998,

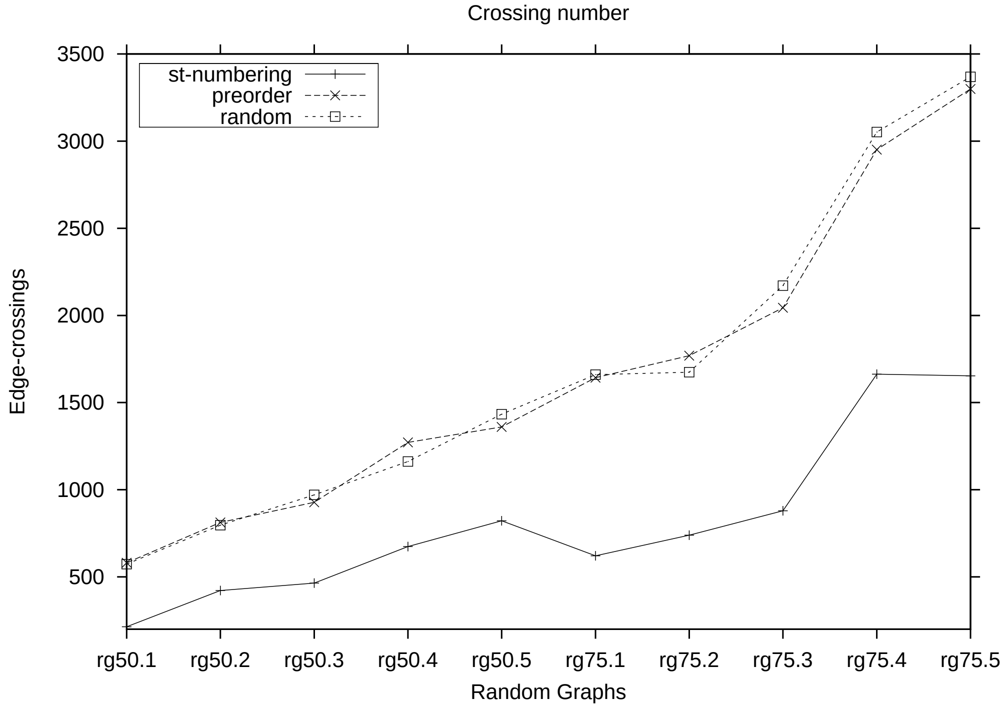


Figure 6: Number of the crossings to different ways of positioning of the vertices.

Calamoneri and Petreschi 1995, Calamoneri et al. 1999, Di Battista et al. 1997] and planarity testing algorithms [Lempel et al. 1967]. The generalized 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 st-numbering is a superior strategy.

It is simple and straightforward to get an st-numbering of only one block in a generalized 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

Biedl, T. and Kant, G. (1998). A better heuristic for orthogonal graph drawings. *Computational Geometry: Theory and Applications*, 9(3), 159–180.

Brandes, U. (2002). Eager *st*-ordering. In *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*, 247–256, London, UK. Springer-Verlag.

Calamoneri, T., Jannelli, S., and Petreschi, R. (1999). Experimental comparison of graph drawing algorithms for cubic graphs. *Journal of Graph Algorithms and Applications*, 3(2), 1–23.

Calamoneri, T. and Petreschi, R. (1995). An efficient orthogonal grid drawing algorithm for cubic graphs. In *Proceedings of the 1st Annual International Conference on Computing and Combi-*

natorics (COCOON'95), volume 959 of *Lecture Notes in Computer Science*, 31–40. Springer-Verlag.

Chung, F. R. K., Leighton, F. T., and Rosenberg, A. L. (1987). Embedding graphs in books: a layout problem with applications to VLSI design. *SIAM Journal of Algebra Discrete Methods*, 8, 33–58.

Cimikowski, R. (1995). An analysis of some heuristics for the maximum planar subgraph problem. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, 322–331.

de Souza, P. S. and Talukdar, S. N. (1993). Asynchronous organizations for multi-algorithm problems. In *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing (SAC'93)*, 286–293, New York, NY, USA. ACM.

Di Battista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., and Vargiu, F. (1997). An experimental comparison of four graph drawing algorithms. *Computational Geometry*, 7(5-6), 303–325.

Didimo, W. and Pizzonia, M. (2003). Upward embeddings and orientations of undirected planar graphs. *Journal of Graph Algorithms and Applications*, 7(2), 221–241.

Ebert, J. (1983). st-ordering the vertices of biconnected graphs. *Computing*, 30(1), 19–33.

Even, S. and Tarjan, R. E. (1976). Computing an *st*-numbering. *Theoretical Computer Science*, 2, 339–344.

Garey, M. R. and Johnson, D. S. (1983). Crossing number is NP-Complete. *SIAM Journal on Algebraic and Discrete Methods*, 4(3), 312–316.

Goldschmidt, O. and Takvorian, A. (1994). An efficient graph planarization two-phase heuristic. *Networks*, 24, 69–73.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. and Thatcher, J., editors, *Complexity of Computer Computations*, 85–103, New York. Plenum Press.

Knuth, D. E. (1974). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, Reading.

Lempel, A., Even, S., and Cederbaum, I. (1967). An algorithm for planarity testing of graphs. In *Theory of Graphs: International Symposium (Rome 1966)*, 215–232, New York. Gordon and Breach.

Maon, Y., Schieber, B., and Vishkin, U. (1986). Parallel ear decomposition search (eds) and st-numbering in graphs. *Theoretical Computer Science*, 47, 277 – 298.

Masuda, S., Nakajima, K., Kashiwabara, T., and Fujisawa, T. (1990). Crossing minimization in linear embeddings of graphs. *IEEE Transactions on Computers*, 39(1), 124–127.

Nicholson, T. A. J. (1968). Permutation procedure for minimising the number of crossings in a network. *Proceedings of the Institution of Electrical Engineers*, 115(1), 21–26.

Papadimitriou, C. H. and Steiglitz, K. (1976). Some complexity results for the traveling salesman problem. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, 1–9, New York. Association for Computing Machinery.

Papamanthou, C. and Tollis, I. G. (2008). Algorithms for computing a parameterized *st*-orientation. *Theoretical Computer Science*, 408(2-3), 224–240.

Sadasivam, S. and Zhang, H. (2009). NP-completeness of *st*-orientations for plane graphs. In *FCT'09: Proceedings of the 17th International Conference on Fundamentals of Computation Theory*, 298–309, Berlin, Heidelberg. Springer-Verlag.

Talukdar, S. N. (1998). Autonomous cyber agents: rules for collaboration. In *Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS-31)*, volume 3, 57.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2), 146–160.

Tarjan, R. E. (1986). Two streamlined depth-first search algorithms. *Fundamenta Informaticae*, 9, 85–94.