Applications of Parameterized *st*-Orientations in Graph Drawing Algorithms

Charalampos Papamanthou[1,2] and Ioannis G. Tollis[1,2]

[1] Department of Computer Science, University of Crete, P.O. Box 2208, Heraklion, Greece
{cpap, tollis}@csd.uoc.gr
[2] Institute of Computer Science, FORTH, Vasilika Vouton, P.O. Box 1385, Heraklion, GR-71110, Greece
{cpap, tollis}@ics.forth.gr

Abstract. Many graph drawing algorithms use *st*-numberings (*st*-orientations or bipolar orientations) as a first step. An *st*-numbering of a biconnected undirected graph defines a directed graph with no cycles, one single source s and one single sink t. As there exist exponentially many *st*-numberings that correspond to a certain undirected graph G, using different *st*-numberings in various graph drawing algorithms can result in aesthetically different drawings with different area bounds. In this paper, we present results concerning new algorithms for parameterized *st*-orientations, their impact on graph drawing algorithms and especially in visibility representations.

1 Introduction

st-orientations (*st*-numberings) or bipolar orientations are orientations of undirected graphs that satisfy some certain criteria, i.e., they define no cycles and have exactly one source s and one sink t. Starting with an undirected biconnected graph $G = (V, E)$, many graph drawing algorithms, such as hierarchical drawings [1], visibility representations [2] and orthogonal drawings [3], use an *st*-orientation of G in order to compute a drawing of G. Therefore, the importance of *st*-orientations in Graph Drawing is evident.

Given a biconnected undirected graph $G = (V, E)$, with n vertices and m edges, and two nodes s, t, an *st*-orientation (also known as bipolar orientation or *st*-numbering) of G is defined as an orientation of its edges such that a directed acyclic graph with exactly one source s and exactly one sink t is produced. An *st*-orientation of an undirected graph can be easily computed using an *st*-numbering [4] of the respective graph G and orienting the edges of G from *low* to *high*. An *st*-numbering of G is a numbering of its vertices such that s receives number 1, t receives number n and every other node except for s, t is adjacent to at least one lower-numbered and at least one higher-numbered node.

st-numberings were first introduced in 1967 in [5], where it is proved (together with an $O(nm)$ time algorithm) that given any edge $\{s, t\}$ of a biconnected undirected graph G, we can define an *st*-numbering. However, in 1976 Even and

P. Healy and N.S. Nikolov (Eds.): GD 2005, LNCS 3843, pp. 355–367, 2005.

Tarjan proposed an algorithm that computes an *st*-numbering of an undirected biconnected graph in $O(n + m)$ time [4]. Ebert [6] presented a slightly simpler algorithm for the computation of such a numbering, which was further simplified by Tarjan [7]. The planar case has been extensively investigated in [8] where a linear time algorithm is presented which may reach any *st*-orientation of a planar graph. Finally, in [9] a parallel algorithm is described. An overview of the work concerning bipolar orientations is presented in [10].

However, all developed algorithms compute an *st*-numbering at random, without expecting any specific properties of the oriented graph. In this paper we present new techniques that produce such orientations with specific properties. Namely, our techniques are able to control the length of the longest path of the resulting directed acyclic graph. This provides significant flexibility to many graph drawing algorithms such as [2, 3]. Actually, *st*-orientations play a very important role in defining certain aesthetics in the drawings produced by algorithms they use them. The length of the longest path of the final directed graph that is produced is vital in determining the area bounds of the drawing. In this paper, we try to answer these questions by connecting a newly developed algorithm for the computation of *st*-orientations with graph drawing applications.

The paper is organized as follows. In Section 2 we present the problem, the objectives and some preliminary definitions. In Section 3 we give a brief description of the algorithm and show its implication in defining the longest path length of the final directed graph. A detailed presentation of the algorithm can be found in [11]. In Section 4 we comment on primal and dual *st*-orientations and Section 5 presents experimental results. Finally, some conclusions are presented in Section 6.

2 Preliminaries

2.1 Motivation and Objectives

Many algorithms in Graph Drawing use *st*-Orientations as a first step. Additionally, the length of the longest path from s to t of the specific *st*-orientation determines certain aesthetics of the drawing:

- **Hierarchical Drawings.** One of the most common algorithms in hierarchical drawing is the longest path layering [1]. This algorithm applies to directed acyclic graphs. The height of such a drawing is always equal to the length of the longest path of the directed acyclic graph, l. If we want to visualize an undirected graph G using this algorithm, we must firstly *st*-orient G. The height of the produced drawing will be equal to the length of the longest path l of the produced *st*-orientation.
- **Visibility Representations.** In order to compute visibility representations of planar graphs, we must compute an optimal topological numbering of an *st*-orientation of the input graph [2]. This can be done if we assign unit-weights to the edges of the graph and compute the longest path to each one of its vertices from source s. The y-coordinate of each vertex u in the

visibility representation is equal to the length of the longest path from s to u. Hence the length of the longest path of the used *st*-orientation is decisive in visibility representations of undirected graphs. Moreover, in the visibility representations, the length of the longest path of the dual graph is also important. How a different primal *st*-orientation impacts on the dual orientation is very crucial for visibility representations.

– **Orthogonal Drawings.** The first step of algorithms that compute orthogonal drawings [3] is to compute an *st*-numbering of the input undirected graph G. These algorithms compute some variables (such as the row pairs or the column pairs in [3]) that are functions of the *st*-orientation and which determine the width and the height of the drawing. Applying different *st*-orientations for the orthogonal drawing of a graph G, can result in different drawing area bounds.

Figure 1 depicts an undirected graph G (Figure 1a) and two different *st*-orientations of it. Figure 2 shows two different longest path and visibility representation layouts for the two different *st*-orientations (1b), (1c) of the same graph (1a). Note that the drawings have different characteristics, which depend on the length of the longest path of the different *st*-orientations.

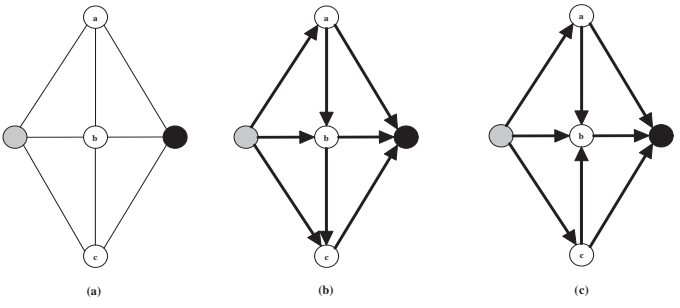

Fig. 1. An undirected graph (a) and two (b), (c) possible *st*-orientations of it

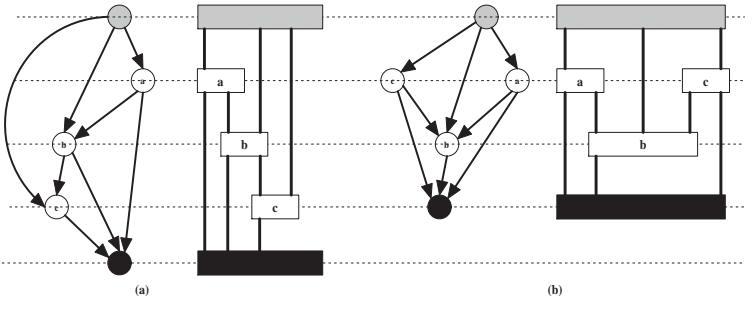

Fig. 2. Longest path layering and visibility representation layouts for the *st*-orientation of Figure 1b (a) and for this of Figure 1c (b)

In order to develop an algorithm for the computation of (longest-path) parameterized *st*-orientations, there are mainly two things that we should carefully consider: (1), the correctness of the final *st*-orientation and (2), the algorithm should give us the opportunity to control the length of the longest path of the final directed graph. The idea behind the algorithm is that, beginning with an undirected biconnected graph G and two nodes of it s, t, we repeatedly remove a node v_i (different from t), orienting at the same time all its incident edges from v_i to its neighbors. In this way we build up a directed graph F. The first node removed is the source s, of the desired *st*-orientation. Thus, the problem of computing a correct *st*-orientation is reduced to this of removing the vertices of the graph with a correct order $v_1, v_2, \ldots, v_n$ with $v_1 = s$ and $v_n = t$ and simultaneously maintaining a data structure that will allow us to compute such a correct order.

2.2 Terminology

In this section, we present some terminology and useful observations. Throughout the paper, $N_G(v)$ denotes the set of neighbors of node v in graph G, s the source and t the sink of the graph. Additionally, l is the length of the longest path of the primal graph from s to t, whereas l^* denotes the length of the longest path of the respective dual graph. Let $G = (V, E)$ be a one-connected undirected graph, i.e., a graph that contains at least one vertex whose removal causes the initial graph to disconnect and $T = (B \cup C, U)$ be the respective block-cutpoint tree [12]. The edges $(i, j) \in U$ of the block-cutpoint tree always connect pairs of blocks (biconnected components) and cutpoints such that the cutpoint of a tree edge belongs to the vertex set of the corresponding block (see Figure 3).

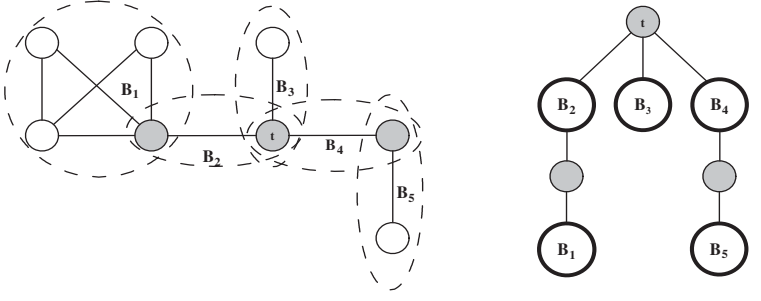

Fig. 3. A one-connected graph and the t-rooted block-cutpoint tree

The block-cutpoint tree is a free tree, i.e., it has no distinct root. In order to transform this free tree into a rooted tree, we define the t-rooted block-cutpoint tree with respect to the sink t. Consequently, the root of the block-cutpoint tree is the block that contains t (see Figure 3).

Finally, we define the leaf-blocks of the t-rooted block-cutpoint tree to be the blocks, except for the root of the block-cutpoint tree that contain a single

cutpoint. The block-cutpoint tree can be computed in $O(n + m)$ time with an algorithm similar to DFS [12].

Following, we give some results that are necessary for the development of the algorithm.

Lemma 1 *([11]). Let $G = (V, E)$ be an undirected biconnected graph and s, t be two of its nodes. Suppose we remove s and all its incident edges. Then there is at least one neighbor of s lying in a leaf-block of the t-rooted block-cutpoint tree. Moreover, this neighbor is not cutpoint.* □

The main idea of the algorithm is based on the successive removal of nodes and the simultaneous update of the t-rooted block-cutpoint tree. We call each such node a source, because at the time of its removal it is effectively chosen to be a source of the remainder of the graph. We initially remove s, the first source, which is the source of the desired *st*-orientation and give direction to all its incident edges from s to all its neighbors. After this removal, the graph either remains biconnected or is decomposed into several biconnected components but the number of leaf-blocks remains the same or is decomposed into several biconnected components and the number of leaf-blocks changes.

This procedure continues until all nodes of the graph but one are removed. As it will be clarified in the next sections, at every step of the algorithm there will be a set of potential sources to choose from. Our aim is to establish a connection between the current source choice and the length of the longest path of the produced *st*-oriented graph.

3 Parameterized *st*-Orientations

3.1 The Algorithm

Now we describe the procedure in a more formal way. We name this procedure STN. Let $G = (V, E)$ be an undirected biconnected graph and s, t two of its nodes. We will compute an *st*-orientation of G. Suppose we recursively produce the graphs $G_{i+1} = G_i - \{v_i\}$, where $v_1 = s$ and $G_1 = G$ for all $i = 1, \ldots, n-1$.

During the procedure we always maintain a t-rooted block-cutpoint tree. Additionally, we maintain a structure Q that plays a major role in the choice of the current source. Q initially contains the desired source for the final orientation, s. Finally we maintain the leaf-blocks of the t-rooted block-cutpoint tree. During every iteration i of the algorithm node v_i is chosen so that

- it is a non-cutpoint node that belongs to Q (1)
- it belongs to a leaf-block of the t-rooted block-cutpoint tree (2)

Note that for $i = 1$ there is a single leaf-block (the initial biconnected graph) and the cutpoint that defines it is the desired sink of the orientation, t. When a source v_i is removed from the graph, we have to update Q in order to be able to choose our next source. Q is then updated by removing v_i and by inserting all of the neighbors of v_i except for t.

By Lemma 1, after the removal of a node v_i, there will always exist at least one node satisfying both (1) and (2). In this way we can reach the final sink of the orientation, t, without disconnecting the graph. Additionally, each time a node v_i is removed we orient all its incident edges from v_i to its neighbors. The procedure continues until Q gets empty. Let $F = (V', E')$ be the directed graph computed by this procedure. We claim [11] that $F = (V', E')$ is an st-oriented graph:

Theorem 2 *([11]). The directed graph $F = (V', E')$ computed by STN is st-oriented.* □

STN is a recursive algorithm for computing an st-orientation of a biconnected undirected graph G. The full pseudocode and an illustrative example can be found in [11]. During the execution of the algorithm we can also compute an st-numbering f of the initial graph. Actually, for each node v_i that is removed from the graph, the subscript i is the final st-number of node v_i. Finally, each node v inserted into Q is associated with a timestamp value $m(v)$ (which will finally determine the longest path length). $m(v)$ is set equal to i, every time that v is discovered by a removed node v_i, i.e., v is a neighbor of v_i. This means that $m(v)$ can be updated many times until the algorithm terminates.

Let us now comment on the execution time of the algorithm. Each time a vertex is removed, we have to update the block-cutpoint tree, which takes time $O(n + m)$ [12]. As all the vertices are removed, the algorithm runs clearly in $O(nm)$ time. However we can use the algorithm for biconnectivity maintenance (which supports edge deletions in $O(\log^5 n)$ time) proposed in [13] and drop the bound to $O(m \log^5 n)$ [11].

3.2 Control of the Length of Longest Path

This section presents methods which can be implemented in order to *control* the length of the longest path of an st-orientation computed with STN. Actually, we take advantage of the timestamps $m(u)$ in order to choose our next source. During iteration j of the algorithm, we have to pick a leaf-block B_j^l of the t-rooted block-cutpoint tree and we always have to make a choice on the structure $Q' = B_j^l \cap Q \sim \{h_j^l\}$, where h_j^l is the cutpoint that defines B_j^l. Our investigation has revealed that if vertices with high timestamp are chosen then long sequences of vertices are formed and thus there is higher probability to obtain a long longest path. We call this way of choosing vertices MAX-STN. Actually, MAX-STN resembles a DFS traversal (it searches the graph at a *maximal* depth). Hence, during MAX-STN, the next source v is arbitrarily chosen from the set

$$\{v \in Q' : m(v) = \max\{m(i) : i \in Q'\}\}.$$

On the contrary, we have observed that if vertices with low timestamp are chosen, then the final st-oriented graph has relatively small longest path. We call this way of choosing vertices MIN-STN, which in turn resembles a BFS traversal. Hence, during MIN-STN, the next source v is arbitrarily chosen from the set

$$\{v \in Q' : m(v) = \min\{m(i) : i \in Q'\}\}.$$

The length of a longest path from s to t computed with MAX-STN is denoted with $\ell(t)$ whereas this computed with MIN-STN is denoted with $\lambda(t)$. As it has already been reported, it would be desirable to be able to compute *st*-oriented graphs of length of longest path within the interval $[\lambda(t), \ell(t)]$. This is called a parameterized *st*-orientation. So the question that arises is: Can we insert a parameter into our algorithm, for example a real constant $p \in [0, 1]$ so that our algorithm computes an *st*-oriented graph of length of longest path that is a function of p?

This is feasible if we modify STN. As the algorithm is executed exactly n times (n vertices are removed from the graph), we can execute the procedure MAX-STN for the first pn iterations and the procedure MIN-STN for the remaining $(1-p)n$ iterations. We call this method PAR-STN(p) and we say that it produces an *st*-oriented graph with length of longest path from s to t equal to $\Delta(p)$. Note that PAR-STN(0) is equivalent to MIN-STN, thus $\Delta(0) = \lambda(t)$ while PAR-STN(1) is equivalent to MAX-STN and $\Delta(1) = \ell(t)$. PAR-STN has been tested and it seems that when applied to *st*-Hamiltonian graphs (biconnected graphs that contain at least one path from s to t that contains all the nodes of the graph) there is a high probability that $\Delta(p) \geq p(n-1)$. Actually, $\Delta(p)$ is very close to $p(n-1)$. Additionally, it has been observed that if we switch the order of MAX-STN and MIN-STN execution, i.e., execute MIN-STN for the first pn iterations and MAX-STN for the remaining $(1-p)n$ iterations, there is a high probability that $\Delta(p) \leq p(n-1)$. In this case, $\Delta(p)$ is again very close to $p(n-1)$.

4 Primal and Dual *st*-Orientations

4.1 General

Now we present some results concerning the impact of parameterized *st*-orientations on *st*-planar graphs. If we *st*-orient such a graph, we can define a single orientation for the dual graph G^* which is also an s^*t^*-orientation.

This method is used in the visibility representations algorithms [2], when we have to compute the dual s^*t^*-oriented graph. The length of the longest path of this graph determines the width of the geometric representation. Thus, the questions that arise are natural. What is the impact of the parameter p on the length of the longest path of the dual s^*t^*-oriented graph G^* of an *st*-planar graph G, which (the graph G) has been *st*-oriented with PAR-STN(p)? Intuitively, we would expect that l^* (the length of the longest path of the dual graph G^*) will grow inversely proportional to l. As we will see, this is not always the case.

4.2 A Special Class of Planar Graphs

In this section we investigate certain classes of *st*-planar graphs that can be *st*-oriented in such a way that certain lengths of primal and dual longest paths can be achieved. This is actually a good reason to justify the fact that different *st*-orientations are indeed important in many applications.

Definition 3. *We define an n-path planar graph* ($n \geq 5$) $G = (V, E)$ *to be the planar graph that consists of a path* $P = v_2, v_3, \ldots, v_{n-1}$ *of* $n-2$ *nodes and two other nodes* v_1, v_n *such that* $(v_1, v_i) \in E$, $(v_i, v_n) \in E$ $\forall i = 2, \ldots n-1$ *and* $(v_1, v_n) \in E$.

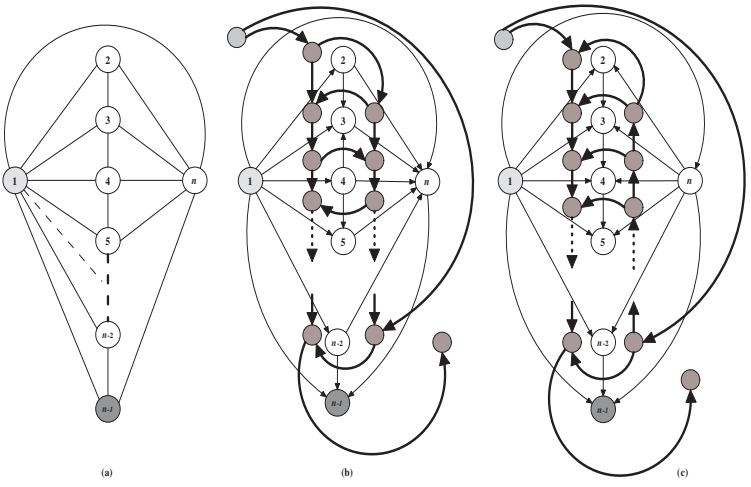

Fig. 4. (a) An n-path planar graph. We define node 1 to be the source of the graph and node $n-1$ to be the sink of the graph. (b) Primal and Dual st-orientation with $l = 4$ and $l^* = 2n-4$. (c) Primal and Dual st-orientation with $l = n-1$ and $l^* = 2n-4$.

In Figure 4a, one n-path planar graph is depicted. Its source is node 1 whereas its sink is node $n-1$. Note that an $(n+1)$-path planar graph G_{n+1} can be obtained from an n-path planar graph G_n if we add a new node and connect it with nodes v_1, v_2 and v_n (nodes v_1 and v_n are the rightmost and leftmost nodes of G_n's embedding in Figure 4). Let now G_n be an n-path planar graph and $\lambda(G_n), \ell(G_n)$ denote the minimum and the maximum longest path length $1(n-1)$-orientations over the set of all the $1(n-1)$-orientations of G_n respectively. In Figure 4b, the primal orientation of minimum longest path length (together with the respective dual orientation of longest path length $\lambda^*(G_n)$) is depicted while in Figure 4c the orientation of maximum longest path length (together with respective dual orientation of longest path length $\ell^*(G_n)$) is depicted. Inductively, we can prove that for an n-path planar graph the following holds:

Theorem 4. *For all* $n \geq 5$, *it is* $\lambda(G_n) = 4$, $\ell(G_n) = n-1$ *and* $\lambda^*(G_n) = \ell^*(G_n) = 2n-4$.

According to Theorem 4, the impact of different st-orientations of an n-path planar graph on the area of their visibility representation is evident. By using the minimum st-orientation, we will need an area equal to

$$\lambda(G_n)\lambda^*(G_n) = 4(2n-4) = 8n-16 = O(n)$$

If we use the maximum *st*-orientation, we will need an area equal to

$$\ell(G_n)\ell^*(G_n) = (n-1)(2n-4) = 2n^2 - 6n + 4 = O(n^2)$$

Note that while $\ell(G_n) + \ell^*(G_n) = 3n - 5 > 2n$, it is $\lambda(G_n) + \lambda^*(G_n) = 2n \leq 2n$. We therefore introduce the following conjecture:

Conjecture 5. *For every n-node planar biconnected graph G, two nodes s, t of its vertex set, there exists at least one st-orientation of G such that* $l+l^* \leq 2n+c$ *, where c is a constant.*

In order to face this conjecture, one should try to devise an algorithm that deterministically *st*-orients a planar graph in a way that the produced length of the dual longest path grows at most as much as the primal one does.

5 Experimental Results

Following we present our results for different kinds of graphs, *st*-Hamiltonian graphs (undirected graphs that have at least one Hamilton path from s to t and hence an upper bound for the longest path length equal to $n-1$) and planar graphs. All experiments were run on a Pentium IV machine, 512 MB RAM, 2.8 GH under Windows 2000 professional.

5.1 *st*-Hamiltonian Graphs

We have implemented the algorithm in Java, using the Java Data Structures Library (www.jdsl.org) [14]. The graphs we have tested are n-node-undirected

Table 1. Results for density 3.5 *st*-Hamiltonian graphs

n	**p=0**		**p=0.3**		**p=0.5**		**p=0.7**		**p=1**	
	l	$\%(n-1)$	l	$\%(n-1)$	l	$\%(n-1)$	l	$\%(n-1)$	l	$\%(n-1)$
100	14.00	0.141	38.90	0.393	59.20	0.598	76.50	0.773	92.20	0.931
200	18.60	0.093	74.10	0.372	113.00	0.568	147.90	0.743	186.60	0.938
300	23.30	0.078	104.80	0.351	165.10	0.552	219.20	0.733	280.70	0.939
400	23.30	0.058	139.10	0.349	213.80	0.536	289.30	0.725	376.30	0.943
500	29.20	0.059	169.40	0.339	267.30	0.536	361.20	0.724	470.70	0.943
600	27.90	0.047	202.10	0.337	318.90	0.532	428.90	0.716	566.60	0.946
800	30.00	0.038	264.90	0.332	415.30	0.520	566.50	0.709	755.60	0.946
900	31.70	0.035	294.30	0.327	469.90	0.523	640.20	0.712	848.10	0.943
1000	36.20	0.036	322.10	0.322	518.20	0.519	709.30	0.710	940.00	0.941
1100	38.90	0.035	353.90	0.322	576.30	0.524	782.90	0.712	1033.40	0.940
1200	34.40	0.029	387.00	0.323	622.10	0.519	845.50	0.705	1127.80	0.941
1300	34.30	0.026	421.10	0.324	674.50	0.519	917.00	0.706	1223.10	0.942
1400	38.90	0.028	448.80	0.321	718.40	0.514	983.90	0.703	1319.90	0.943
1500	38.00	0.025	478.30	0.319	775.70	0.517	1056.40	0.705	1417.10	0.945
1600	39.30	0.025	515.00	0.322	824.30	0.516	1137.20	0.711	1499.10	0.938
1700	38.50	0.023	539.30	0.317	872.00	0.513	1190.40	0.701	1604.00	0.944
1800	41.10	0.023	571.90	0.318	923.60	0.513	1263.80	0.703	1691.30	0.940
1900	41.40	0.022	605.60	0.319	978.60	0.515	1331.80	0.701	1786.30	0.941
2000	44.00	0.022	632.40	0.316	1023.80	0.512	1403.50	0.702	1883.90	0.942

st-Hamiltonian graphs of density d where $n = 100, 200, 300, \ldots, 2000$ and $d = 3.5$. For each pair (n, d) we have tested 10 different randomly generated graphs (and we present the mean of the length of the longest path) in order to get more reliable results. We have similar results for other values of density as well (see Figure 5).

As we can see, the results (Table 1 and Figure 5) are remarkably consistent with the parameter p. The computed longest path length for $p = p_0$ is always very close to $p_0(n-1)$. The computed results are similar for increasing graphs size and density.

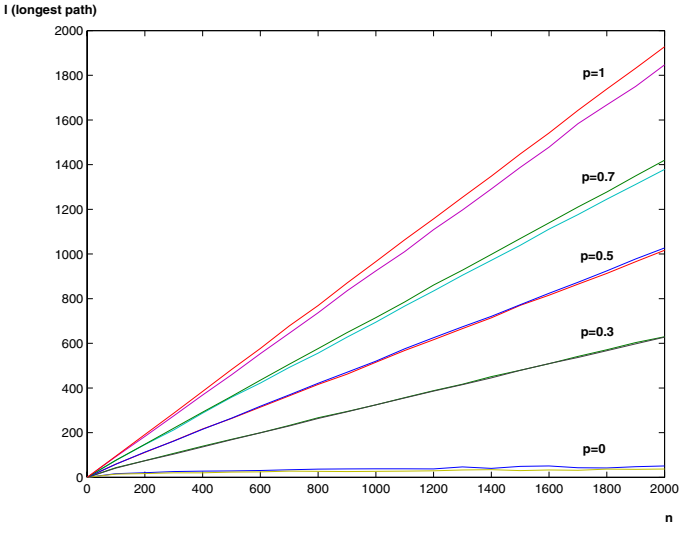

Fig. 5. Longest path length as a function of n, d, p ($d = 2.5, 6.5$)

Table 2. Primal and dual longest path length for triangulated st-planar graphs

		p=0			**p=0.5**			**p=1**			$l \times l^*$		
n	$2n$	l	l^*	$l+l^*$	l	l^*	$l+l^*$	l	l^*	$l+l^*$	**p=0**	**p=0.5**	**p=1**
109	218	31	167	198	75	95	170	100	74	174	5177	7125	7400
310	620	44	503	547	186	319	505	280	163	443	22132	59334	45640
535	1070	98	785	883	240	534	774	402	293	695	76930	128160	117786
763	1526	144	1114	1258	385	780	1165	691	241	932	160416	300300	166531
998	1996	83	1419	1502	425	862	1287	846	340	1186	117777	366350	287640
1302	2604	134	2024	2158	704	1154	1858	1173	451	1624	271216	812416	529023
1501	3002	119	2203	2322	784	1073	1857	1403	224	1627	262157	841232	314272
1719	3438	131	2550	2681	856	1661	2517	1555	515	2070	334050	1421816	800825
1990	3980	208	2339	2547	1013	1581	2594	1773	400	2173	486512	1601553	709200
2159	4318	142	3238	3380	930	1816	2746	1823	445	2268	459796	1688880	811235
2268	4536	148	3136	3284	952	1666	2618	1887	336	2223	464128	1586032	634032
4323	8646	356	5852	6208	2238	3589	5827	3957	841	4798	2083312	8032182	3327837

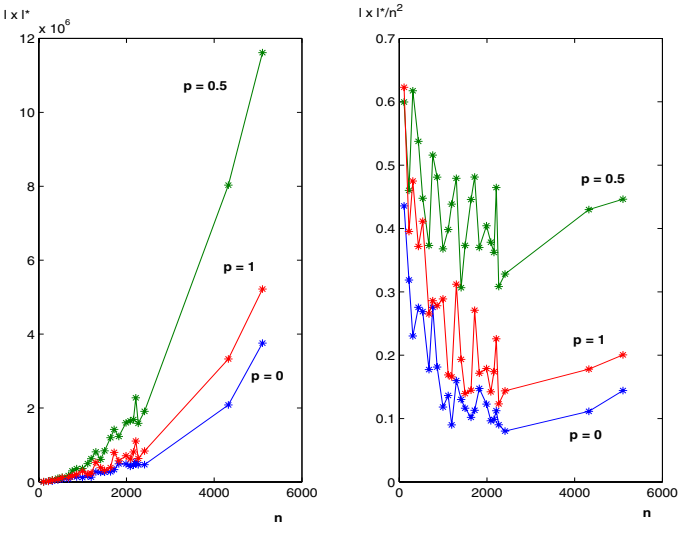

Fig. 6. Absolute (left) and normalized (divided by n^2) (right) results for visibility representation area requirement for different values of the parameter p and triangulated planar graphs. The parameter $p = 0$ (low longest path length *st*-oriented graphs) is clearly preferable.

5.2 Planar Graphs

In this section we present some results for maximum density (triangulated) *st*-planar graphs. We also have similar results for low density planar graphs. We mainly present the impact of the parameter p on the primal and dual longest path length of the planar graphs. From Table 2, it is clear that the primal and the dual longest path length are inversely proportional for various values of the parameter p. We have used the values $p = 0, 0.5, 1$, as the most representative ones. The last three columns of Table 3 show the product $l \times l^*$. This is actually the area that is needed in order to construct a visibility representation of the given graph using the algorithms proposed in [2].

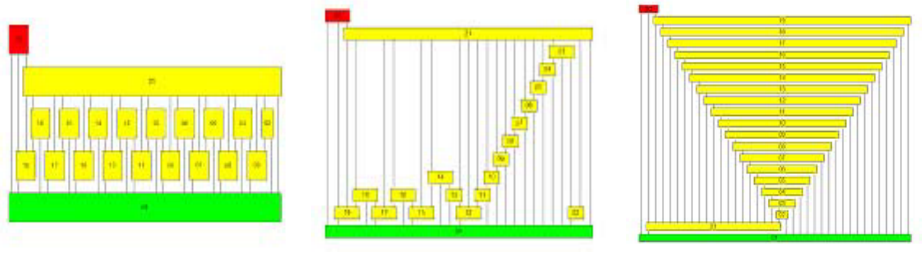

Fig. 7. Visibility Representations of a 21-path planar graph for different *st*-orientations ($p = 0, 0.5, 1$)

Fig. 8. Visibility Representations of a 85-node triangulated planar graph for different *st*-orientations produced with PAR-STN(p) ($p = 0, 0.5, 1$)

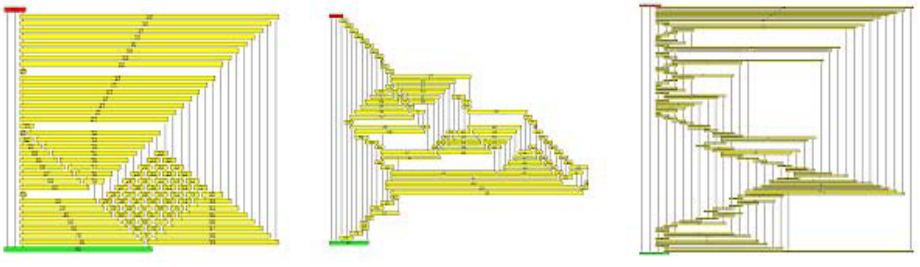

Fig. 9. Visibility Representations of a 10x10 grid graph for different *st*-orientations produced with PAR-STN(p) ($p = 0, 0.25, 1$)

Figure 7 shows 3 visibility representation frames of a 21-path planar graph. The difference in the area is evident. Note that the visibility representation that uses the minimum *st*-orientation ($p = 0$) consumes the least area. Figure 8 contains 3 visibility representations frames of a triangulated graph where the value $p = 0$ is preferable. Finally, in Figure 9 we present some visibility representations frames produced by *st*-orienting a grid graph. In this case, the importance of the parameter is clear. Using a parameterized *st*-orientation with $p = 0.25$ is preferable, as it produces a more *compact* drawing.

5.3 Orthogonal Drawings

The impact of the different *st*-orientations is not very clear in orthogonal drawings. However, for the algorithm described in [3], where the area upper bound is roughly $0.76n^2$, we are able to produce *st*-numberings that produce drawings of area upper bound roughly equal to $0.68n^2$ or less, by using the parameterized *st*-orientation algorithm. Due to space limitations, we cannot describe further details.

6 Conclusions

In this paper the application of parameterized *st*-orientations in graph drawing algorithms (mainly in visibility representations) is presented. It seems that there

is a way to efficiently control the length of the longest path of an *st*-orientation and keep it "short", "long" or "medium". Experimental results not only on planar graphs but also on non-planar graphs reveal the robustness of the algorithm.

Acknowledgements. The authors would like to thank Hubert de Fraysseix, C.N.R.S. for his help on the visualization frames produced with his software P.I.G.A.L.E..

References

1. G.D. Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
2. R. Tamassia and I.G. Tollis. A unified approach to visibility representations of planar graphs. *Disc. and Comp. Geom.*, 1:321–341, 1986.
3. A. Papakostas and I.G. Tollis. Algorithms for area-efficient orthogonal drawings. *Computational Geometry: Theory and Applications*, 9:83–110, 1998.
4. S. Even and R. Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2:339–344, 1976.
5. A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *P. Rosestiehl(ed.) Theory of Graphs: International Symposium July 1966*, pages 215–232, 1967.
6. J. Ebert. st-ordering the vertices of biconenected graphs. *Computing*, 30(1):19–33, 1983.
7. R. Tarjan. Two streamlined depth-first search algorithms. *Fundamentae Informatica*, 9:85–94, 1986.
8. P. Rosehnstiehl and R. Tarjan. Rectilinear planar layout and bipolar orientation of planar graphs. *Discrete Comput. Geom.*, 1:343–353, 1986.
9. Y. Maon, B. Schieber, and U. Vishkin. Parallel ear decomposition search (eds) and st-numbering in graphs. *Theoret. Comput. Sci*, 47:277–298, 1986.
10. H.D. Fraysseix, P.O. de Mendez, and P. Rosenstiehl. Bipolar orientations revisited. *Discrete Applied Mathematics*, 56:157–179, 1995.
11. C. Papamanthou and I.G. Tollis. Algorithms for parameterized st-orientations of graphs, *submitted to ESA2005*.
12. J. Hopcroft and R. Tarjan. Efficient algorithms for graph manipulation. *Comm. ACM*, 16:372–378, 1973.
13. J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge and biconnectivity. *J. ACM*, 48(4):723–760, 2001.
14. Michael T. Goodrich and Roberto Tamassia. *Data Structures and Algorithms in Java*. John Wiley & Sons, 2 edition, 2001.