Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2017) 25: 3960 – 3969
© TÜBİTAK
doi:10.3906/elk-1608-281

S-visibility problem in VLSI chip design

Marzieh EKANDARI[*], **Mahdieh YEGANEH**
Department of Mathematical sciences, Faculty of Computer Sciences, Alzahra University, Tehran, Iran

Received: 30.08.2016 • **Accepted/Published Online:** 09.02.2017 • **Final Version:** 05.10.2017

Abstract: In this paper, a new version of very large scale integration (VLSI) layouts compaction problem is considered. Bar visibility graph (BVG) is a simple geometric model for VLSI chip design and layout problems. In all previous works, vertical bars or other chip components in the plane model gates, as well as edges, are modeled by horizontal visibilities between bars. In this study, for a given set of vertical bars, the edges can be modeled with orthogonal paths known as staircases. Therefore, we consider a new version of bar visibility graphs (BsVG). We then present an algorithm to solve the s-visibility problem of vertical segments, which can be implemented on a VLSI chip. Our algorithm determines all the pairs of segments that are s-visible from each other.

Key words: Symbolic layout, visibility, dynamic programming, VLSI design

1. Introduction

Visibility problems are very popular in many application areas, particularly in very large scale integration (VLSI) circuit layout. Bar visibility graph (BVG) is a simple geometric model for VLSI chip design and layout problems [1–4]. In this model, vertical bars in the plane model gates, other chip components, and edges are modeled by horizontal visibilities between bars. The problem of determining the visible pairs of n given vertical segments in the horizontal visibility model was studied by Lodi et al. in 1986 [5]. They proposed a parallel algorithm to solve the visibility problem among n vertical segments in a plane, considered by Bose et al. in 1997 [6]. They studied the problem of computing rectangle visibility graphs (RVGs) of a set of vertical bars [7–9]. In 2015, Carmi et al. studied the problem of computing the visibility graph of a set of vertical segments (known as walls). There is an edge between two walls if, and only if, they are weakly visible to each other. They presented an $O\left(n^3 \log n\right)$ time algorithm for a scene consisting of n walls of varying heights parallel to the yz-plane, where visibility is restricted to directions with horizontal projections on the xy-plane [10].

2. Preliminaries

In this section, we will introduce a new version of the VLSI chip design problem. In general, the problem concerns finding the BVG of a set of given vertical bars in the plane with model gates or other chip components. In the previous studies, the edges were modeled by horizontal visibilities between bars. In other words, two vertical bars may connect to each other or "see each other" if there exists a horizontal line segment h, whose endpoints are on these bars and h does not hit the other bars.

[*]Correspondence: eskandari@alzahra.ac.ir

In our model, the edges can be modeled by orthogonal paths called staircases. A set of n vertical line segments is given. We aim to determine all pairs $(b_{i,}b_j)$ in such a way that a "staircase path" exists between them.

Let x_0 and x_n be two points in the plane. An orthogonal path from x_0 to x_n is a sequence of points $x_0, x_1, x_2, \ldots, x_n$, so that $x_{i-1}x_i$ is a horizontal or vertical line segment, where $i = 1, 2, \ldots, n$

Definition 1 *A staircase is a monotone orthogonal path, whose turns alternate between* 90° *left and* 90° *right.*

If the segments of a staircase are oriented from x_0 *to* x_n *, a staircase can be classified into four types:* SE, NE, SW *, and* NW*. If the segments are either toward the south or the east, it is called a* SE *staircase. The other types are defined similarly.*

Definition 2 *Let* p *and* q *be two points and* O *be a set of obstacles in the plane. We say that* q *is* s*-visible from* p *if there exists a staircase path from* p *to* q *that does not intersect with obstacles. Additionally, if this staircase path is a* SE *staircase, we say that* q *is* SE*-visible from* p*. The other types of* s*-visibility (*NE*-visible,* SW*-visible, and* NW*-visible) are defined similarly. Note that if* q *is* SE*-visible from* p*, then* p *is* NW*-visible from* q*.*

Let $B = \{b_1 b_2, \ldots, b_n$ *be a set of* n *vertical line segments called bars. The endpoints of* b_i *are named by* $L_i = (x_i l_i)$ *and* $H_i = (x_i h_i)$*, such that* $l_i < h_i$*. Without loss of generality, assume that* $x_i \neq x_j$ *for all* $i \neq j$*. Note that these bars may block visibility; this means that* $O = B$*.*

Definition 3 *Let* b_i *and* b_j *be two bars in* B*. We say that* b_i *is* s*-visible from* b_j *if there exists a point on* b_i *that is* s*-visible from a point on* b_j*.*

Note that if b_j *is* SE*-visible from* b_i*, for some* $i < j$*, then* b_i *is* NW*-visible from* b_j*.*

Definition 4 *The* s*-visibility graph of* B *has a node* v_i *for every bar* b_i *and an edge for every pair of* s*-visible bars in* B*. We denote this graph as BsVG.*

In this model, we calculate some extra paths between bars, which were not visible in previous models. In other words, $BsVG$ reports more visibility paths than BVG. See Figures 1 and 2 for an illustration.

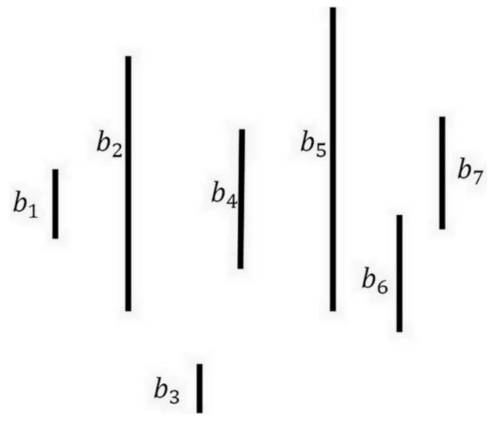

Figure 1. The set of seven bars in the plane.

Graphs BVG and BsVG can yield from a given set of bars as follows: for every bar b_i, graph BVG has a node i (see Figures 1 and 2). If b_i and b_j are horizontally visible to each other, we add an edge between nodes

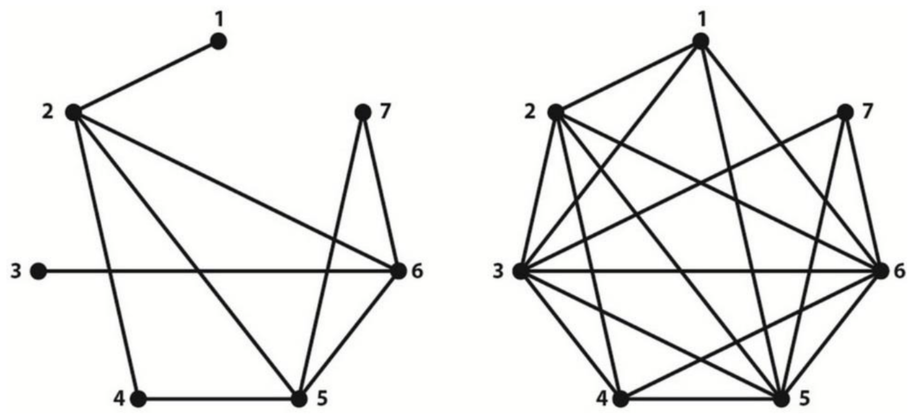

Figure 2. BVG (left) vs. BsVG (right), derived from the bars of Figure 1 in two visibility models: horizontal and staircase visibilities.

i and j. Similarly, we can construct graph BsVG for the bars of Figure 1. For every bar b_i, we add a node i to graph BsVG, and for each s-visible pair b_i and b_j, an edge will be added between $i\,and\,j$.

Lemma 1 *b_j is s-visible from b_i for $i < j$ if, and only if, one of its endpoints is s-visible from one of the endpoints of b_i.*

Proof It is obvious that if one of the endpoints of b_j is s-visible from one of the endpoints of b_i, then b_j is s-visible from b_i. Assuming that b_j is s-visible from b_i, there are two points, $p \in b_i$ and q $\in b_j$, so that q is s-visible from p. We will show that one of the endpoints of b_j is s-visible from one of the endpoints of b_i. There are nine cases for these two points, shown in the Table.

Table. The two visible points p and q of bars of b_i and b_j may jointly have nine positions on b_i and b_j.

	$p = L_i$	$p \neq H_i,L_i$	$p = H_i$
$q = L_j$	done	Figure. 4.a	done
$q \neq H_j,L_j$	Figure. 4.b	Figure. 4.c	Figure. 4.d
$q = H_j$	done	Figure. 4.e	done

Clearly, for all these nine cases, each staircase path from p to q can be extended to a staircase path from the endpoints of the bars. See Figure 3 for an illustration of these extensions. The evidence is summarized in the Table. □

From now on, we only consider s-visibility of the endpoints of the bars.

3. SE-visible pairs

Without loss of generality, assume that $\mathbf{B} = \{\mathbf{b_1}, \ldots, \mathbf{b_n}$ is a set of bars sorted on increasing $\mathbf{x}$-coordinate, i.e. $\mathbf{x_1} < \mathbf{x_2} < \ldots \mathbf{x_n}$.

In this section, we propose an algorithm to compute all **SE**-visible pairs $(\mathbf{b_i}, \mathbf{b_j})$ of $\mathbf{B}$ for $\mathbf{i} < \mathbf{j}$. In the next section, we propose the main algorithm, which reports all s-visible pairs.

Initially, we construct a graph called *orthogonal graph*, which contains all **SE**-staircase paths between bars in $\mathbf{B}$. We denote the orthogonal graph of a set of $\mathbf{B}$ as $\mathbf{OG(B)}$. Then we compute all **SE**-visible pairs $(\mathbf{b_i}, \mathbf{b_j})$ of $\mathbf{B}$ for $\mathbf{i} < \mathbf{j}$ with a dynamic programming approach.

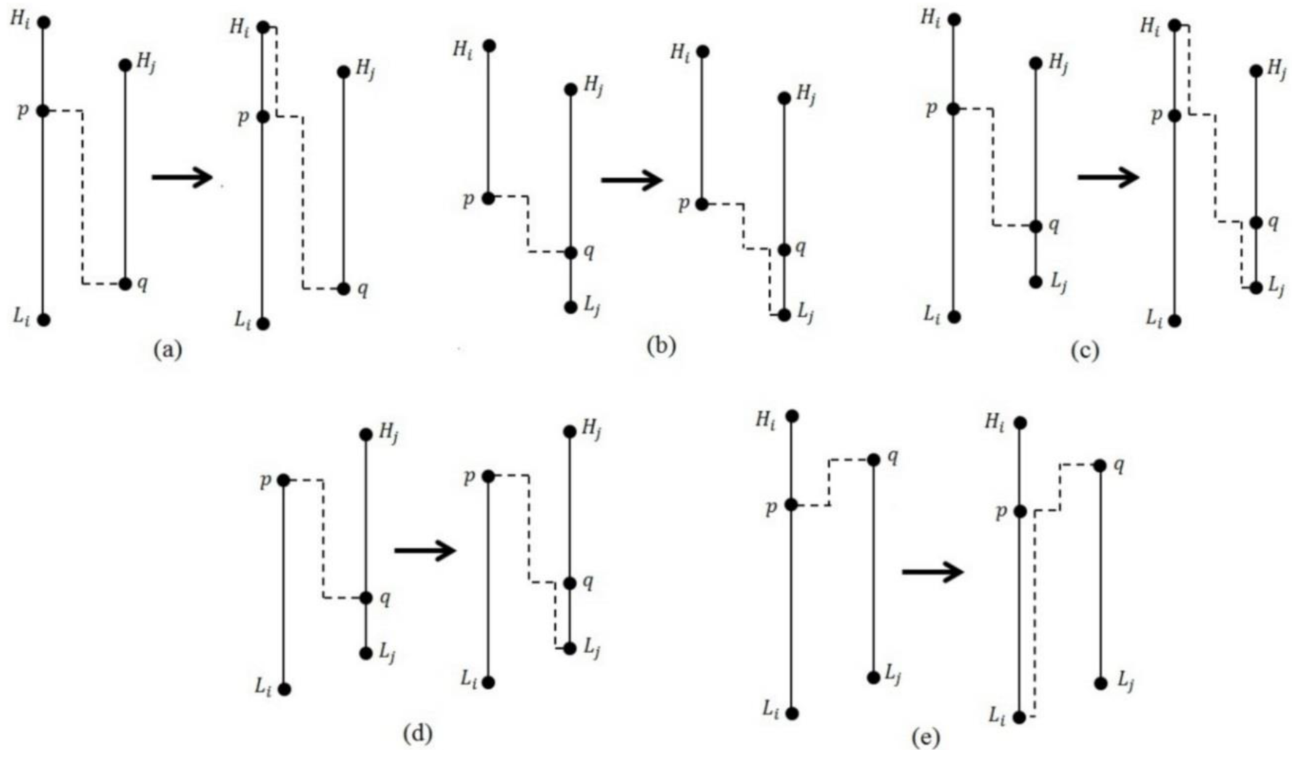

Figure 3. Illustration of Lemma 1.

The set of the vertices and the edges of $\mathbf{OG(B)}$ are denoted as $\mathbf{V}$ and $\mathbf{E}$, respectively. Set $\mathbf{V}$ includes all endpoints of bars and several intermediate vertices. Set $\mathbf{E}$ includes all east- or south-oriented edges between the vertices, as will be detailed below.

There are three types of $\mathbf{OG\,(B)}$ vertices:

- Type I: the endpoints of bars, i.e. $\mathbf{L_iH_i | 1 \leq i \leq n}$.
- Type II: lie on the bars.
- Type III: lie on the supporting line of the bars (not on the bars).

The second- and third-type vertices will be explored in detail in the algorithm.

For $\mathbf{1 \leq i \leq n}$, let $\mathbf{V_i}$ be a subset of $\mathbf{V}$ that contains the vertices from all three types with the same $\mathbf{x}$-coordinate, precisely $\mathbf{x_i}$. Then $\mathbf{V} = \bigcup\limits_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{V_i}$.

Let us assume that the vertices of each $\mathbf{V_i}$ are sorted on the decreasing $\mathbf{y}$-coordinate. Let $\mathbf{V_i}$= $\{\mathbf{v_1^i, v_2^i}, \ldots, \mathbf{v_{m_i}^i}$, where $\mathbf{m_i}$ is the number of vertices in $\mathbf{V_i}$. By this notation, if we set $\mathbf{v_j^i}$= $(\mathbf{x_iy_j^i})$, for all $\mathbf{1 \leq i \leq n}$ and $\mathbf{1 \leq j \leq m_i}$, then $\mathbf{y_{j+1}^i} < \mathbf{y_j^i}$.

The orthogonal graph of the set of bars in Figure 1 is shown in Figure 4. The white nodes are the endpoints of the bars (Type I). The green nodes are second-type and the blue ones are third-type.

Let $\mathbf{A}$ be the adjacency matrix of $\mathbf{OG(B)}$. The first two rows (columns) correspond to the vertices of $\mathbf{V_1}$, i.e. $\mathbf{H_1}$ and $\mathbf{L_1}$, respectively. The next $\mathbf{m_2}$ rows (columns) of $\mathbf{A}$ correspond to $\mathbf{v_1^2, v_2^2}, \ldots, \mathbf{v_{m_2}^2}$, i.e. the

Algorithm 1 Constructing orthogonal graph

Input: Set of bars $\mathbf{B}$.

Output: Orthogonal graph of $\mathbf{B}$, $\mathbf{OG(B)}$.

```
1. $\mathbf{V_i}=\emptyset$;   $\mathbf{E}=\{(\mathbf{H_1},\mathbf{L_1})\}$;   $\mathbf{i}=\mathbf{1}$;
2. $\mathbf{V_i}=\{\mathbf{L_i H_i}$;   $\mathbf{L_i}$ and $\mathbf{H_i}$ are labeled as first-type vertices.
3. For all $\mathbf{u}=(\mathbf{x_{i-1}y})\in\mathbf{V_{i-1}}$ do
   If ($\mathbf{u}$ is not a second type vertex), then
      • Add $\mathbf{u}'=(\mathbf{x_i y})$ to $\mathbf{V_i}$ so that $\mathbf{V_i}$ remains sorted.
      • Add edge $(\mathbf{uu})$ to E.
      • If $(\mathbf{l_i}<\mathbf{y}<\mathbf{h_i})$, then $\mathbf{u}$ is labeled as a second type vertex, otherwise it is labeled as a third type vertex.
4. For all $\mathbf{u_j}=(\mathbf{x_i y_j})\in\mathbf{V_i}$, add edge $(\mathbf{u_j u_{j+1}})$ to $\mathbf{E}$.
5. $\mathbf{i}++$; if $(\mathbf{i}<\mathbf{n})$, go to 2.
```

By Algorithm 1, note that $\mathbf{m_i}\leq\mathbf{2i}$.

next $\mathbf{m_i}$ rows (columns) of $\mathbf{A}$ correspond to $\mathbf{v^i_1},\mathbf{v^i_2},\ldots,\mathbf{v^i_{m_i}}$ for $\mathbf{2}<\mathbf{i}\leq\mathbf{n}$ (Figure 5). Therefore, the number of rows and columns of $\mathbf{A}$ is $\mathbf{m}=\sum_{\mathbf{i=1}}^{\mathbf{n}}\mathbf{m_i}$.

The directed graph $\mathbf{OG(B)}$ contains all possible $\mathbf{SE}$-staircase paths between the vertices. To report all $\mathbf{SE}$-staircase paths between the type-one vertices that are $\mathbf{SE}$-visible, we define a solution matrix $\mathbf{P}$ by using a dynamic programming approach. Matrix $\mathbf{P}$ is an $\mathbf{m}\times\mathbf{m}$ matrix, whose rows (and columns) are labeled the same as the rows (and columns) of $\mathbf{A}$. Its elements, $\mathbf{P}[\mathbf{v^k_a v^l_c}]$, are defined below:

$$\mathbf{P}\left[\mathbf{v^k_a},\mathbf{v^l_c}\right]=\begin{cases}\mathbf{1} & \mathbf{v^l_c\,is\,SE-visible\,from\,v^k_a\ or\ v^k_a{=}v^l_c}\\ \mathbf{0} & \mathbf{otherwise}\end{cases}$$

For all $\mathbf{i}$ and $\mathbf{j}$, where $\mathbf{1}\leq\mathbf{ij}\leq\mathbf{n}$, we define an $\mathbf{m_i}\times\mathbf{m_j}$ matrix as follows: it is a submatrix of $\mathbf{A}$ that is restricted to the rows labeled by $\mathbf{V_i}$ and columns labeled by $\mathbf{V_j}$, and is denoted as $\mathbf{A_{i,j}}$. Its rows and columns are labeled as the vertices of $\mathbf{V_i}$ and $\mathbf{V_j}$, respectively. Therefore, we have:

$$\mathbf{A_{i,i}}\left[\mathbf{v^i_a},\mathbf{v^i_c}\right]=\begin{cases}\mathbf{1} & \mathbf{c-a=1}\\ \mathbf{0} & \mathbf{otherwise}\end{cases}$$

Additionally, for $\mathbf{i}\neq\mathbf{j}$ and $\mathbf{i}\neq\mathbf{j-1}$, we always have: $\mathbf{A_{i,j}}\left[\mathbf{v^i_a},\mathbf{v^j_c}\right]=\mathbf{0}$. Similarly, we define $\mathbf{P_{i,j}}$. Obviously, we have:

$$\mathbf{P_{i,i}}\left[\mathbf{v^i_a},\mathbf{v^i_c}\right]=\begin{cases}\mathbf{1} & \mathbf{a\leq c}\\ \mathbf{0} & \mathbf{a>c}\end{cases}$$

Furthermore, for $\mathbf{i}>\mathbf{j}$, we have $\mathbf{P_{i,j}}\left[\mathbf{v^i_a},\mathbf{v^j_c}\right]=\mathbf{0}$. We propose an algorithm to compute Matrix $\mathbf{P}$.In this

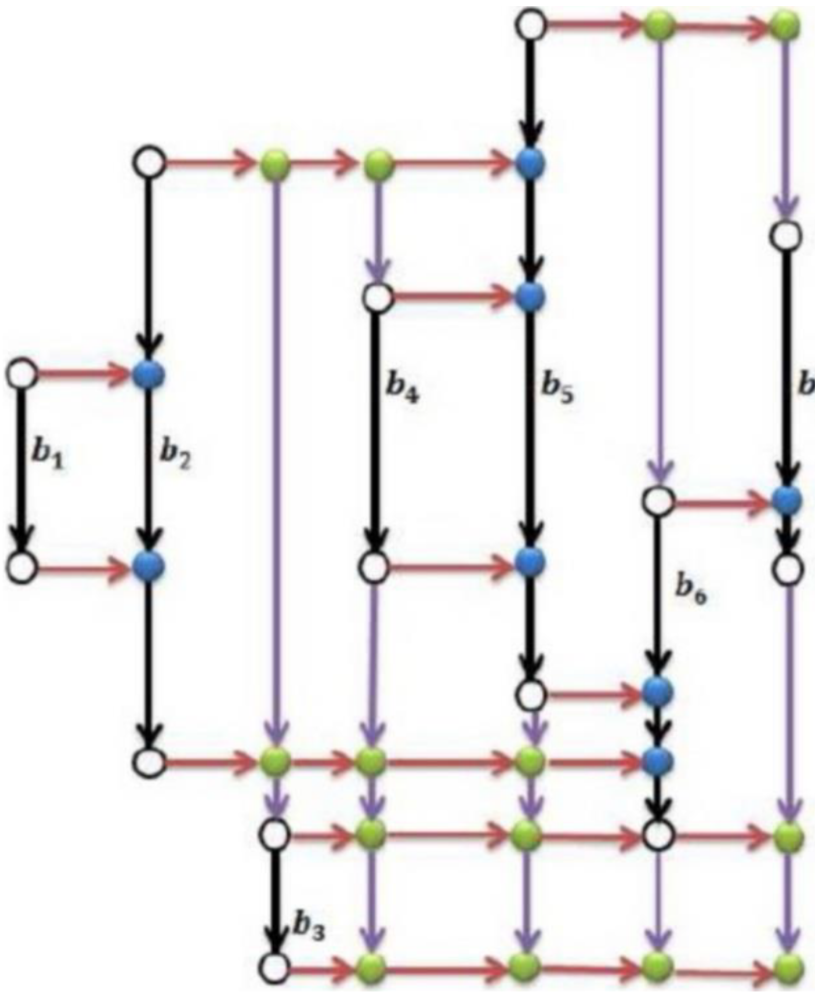

Figure 4. Orthogonal graph of the set of bars of Figure 1.

algorithm, $\mathbf{i \geq j}$, we initialize the values of submatrices $\mathbf{P_{i,j}}$ as described above. Next, for $\mathbf{1 \leq i \leq n-1}$, we compute $\mathbf{P_{i,i+1}}$ directly from $\mathbf{A}$. Then, for $\mathbf{1 \leq i \leq n-2}$, we compute $\mathbf{P_{i,i+2}}$. Next, we can report all pairs $(\mathbf{v_a^k\, v_c^l})$ that are type-one vertices and $\mathbf{P}\left[\mathbf{v_a^k, v_c^l}\right] = \mathbf{1}$ In fact, if for two type-one vertices $\mathbf{v_a^k}$ **and** $\mathbf{v_c^l}$ we have $\mathbf{P}\left[\mathbf{v_a^k, v_c^l}\right] = \mathbf{1}$ this means that $\mathbf{b_l}$ is **SE**-visible from $\mathbf{b_k}$.

In Step 2, we initialize the values of $\mathbf{P_{i,i}}$. In Step 3, we compute the elements of $\mathbf{P_{i,i+1}}$. For this purpose, in Step 3.1 we compute the last row of $\mathbf{P_{i,i+1}}$ and in Step 3.2 we compute the remaining rows with an inductive approach. Note that in Step 3.1 there exists only a unique index $\mathbf{a}$, so that the vertex $\mathbf{v^i_{m_i}}$ is connected to $\mathbf{v_a^{i+1}}$ by an edge. Therefore, we can store these indices for each $\mathbf{i}$. In other words, the time complexity of searching for index $\mathbf{a}$ is constant. Clearly, if there exists an edge between $\mathbf{v^i_{m_i}}$ and $\mathbf{v_a^{i+1}}$, then for all $\mathbf{c \geq a}$, $\mathbf{v_c^{i+1}}$ is **SE**-visible from $\mathbf{v^i_{m_i}}$. In Step 3.2, we calculate the **SE**-visible vertices from the remaining vertices of $\mathbf{V_i}$. Firstly, we consider that if $\mathbf{v_c^{i+1}}$ is **SE**-visible from $\mathbf{v^i_{j+1}}$, then it is **SE**-visible from $\mathbf{v^i_j}$ (Step 3.2.1). Next, in Step 3.2.2, if there exists an edge between $\mathbf{v^i_j}$ and $\mathbf{v_a^{i+1}}$, then all the next vertices of $\mathbf{V_{i+1}}$ after $\mathbf{v_a^{i+1}}$ are **SE**-visible from $\mathbf{v^i_j}$.

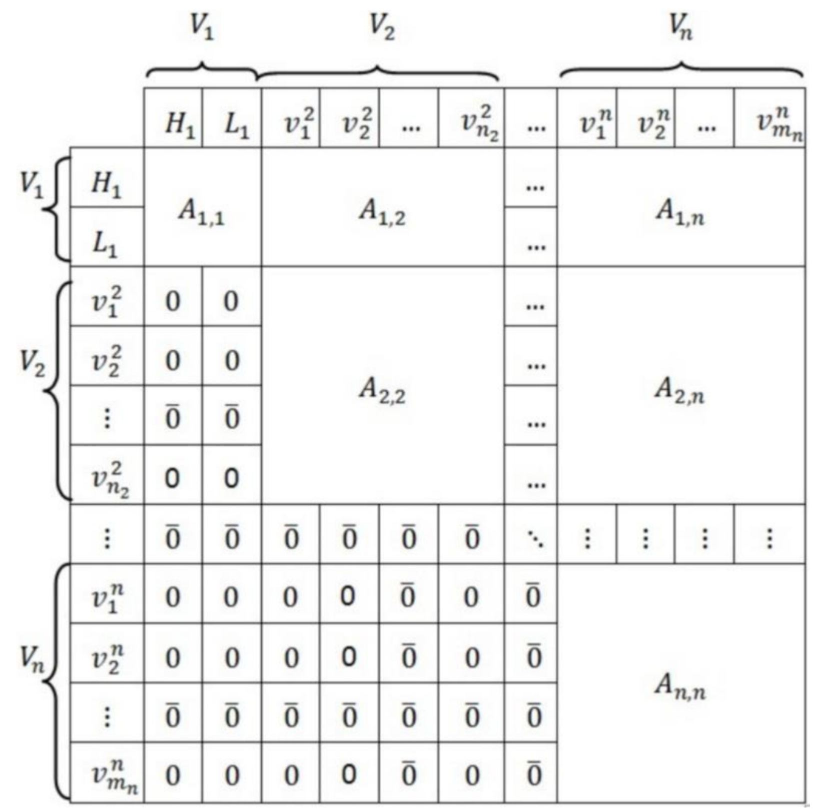

Figure 5. Structure of Matrix A.

Finally in Step 4, we calculate the elements of $\mathbf{P_{i,i+d}}$ by using $\mathbf{P_{i+1,i+d}}$, which is computed before it. This step is similar to Step 3. Initially we compute the last row of $\mathbf{P_{i,i+d}}$ and in Step 4.2 we compute the remaining rows with an inductive approach.

In Step 4.1, if there exists an edge between $\mathbf{v^i_{m_i}}$ and $\mathbf{v^{i+1}_a}$, then all vertices of $\mathbf{V_{i+d}}$ that are **SE**-visible from $\mathbf{v^{i+1}_a}$ are **SE**-visible from $\mathbf{v^i_{m_i}}$.

In Step 4.2, we calculate the **SE**-visible vertices from the remaining vertices of $\mathbf{V_i}$. In Step 4.2.1, we consider that if $\mathbf{v^{i+d}_c}$ is **SE**-visible from $\mathbf{v^i_{j+1}}$, then it is **SE**-visible from $\mathbf{v^i_j}$. Next, in Step 4.2.2, if there exists an edge between $\mathbf{v^i_j}$ and $\mathbf{v^{i+1}_a}$, then all vertices of $\mathbf{V_{i+d}}$ that are **SE**-visible from $\mathbf{v^{i+1}_a}$ are **SE**-visible from $\mathbf{v^i_j}$.

Now we can report all **SE**-visible pairs $(\mathbf{b_k},\mathbf{b_l})$ by scanning Matrix $\mathbf{P}$ for two type-one vertices $\mathbf{v^k_a}$ and $\mathbf{v^l_c}$ so that $\mathbf{P}\left[\mathbf{v^k_a},\mathbf{v^l_c}\right] = \mathbf{1}$

Now we consider the time complexity of Algorithm 2. Steps 1– 3 can run in $O\left(n^3\right)$, and the time complexity of Step 4 is $O\left(n^4\right)$. Note that the number of nodes of $OG(B)$ is $m = \sum_{i=1}^{n} m_i \le \sum_{i=1}^{n} 2i = O(n^2)$. This algorithm can compute all $O(n^4)$ paths between all pairs of nodes in time $O(n^4)$.

4. Results

In this section, we consider the main problem, which concerns reporting all **s**-visible pairs of bars. Note that **s**-visible pairs are either **SE**-visible or **NE**-visible; therefore, we should compute all **SE**-visible and **NE**-visible pairs. We compute **SE**-visible pairs by deploying Algorithms 1 and 2. Then, to compute **NE**-visible pairs,

Algorithm 2 Computing **SE**-visible pairs

Input: The orthogonal graph of a set of bars **B**, **OG**(**B**) and its adjacency matrix, **A**.
Output: Solution Matrix **P**.

```
1. $\mathbf{P} = \bar{0}$;
2. For $\mathbf{i} = \mathbf{1}$ to $\mathbf{n}$ do
   $\forall \mathbf{ac} : \mathbf{1} \le \mathbf{a} \le \mathbf{c} \le \mathbf{m_i}$, set $\mathbf{P_{i,i}}\left[\mathbf{v_a^i}, \mathbf{v_c^i}\right] = \mathbf{1}$;
3. For $\mathbf{i} = \mathbf{1}$ to $\mathbf{n} - \mathbf{1}$ do
   3.1. If $(\mathbf{A}[\mathbf{v_{m_i}^i v_a^{i+1}}] == \mathbf{1})$ then
            For $\mathbf{c} = \mathbf{a}$ to $\mathbf{m_{i+1}}$ do
                set $\mathbf{P_{i,i+1}}\left[\mathbf{v_{m_i}^i}, \mathbf{v_c^{i+1}}\right] = \mathbf{1}$;
   3.2. For $\mathbf{j} = \mathbf{m_i} - \mathbf{1}$ to $\mathbf{1}$ do
        3.2.1. For $\mathbf{c} = \mathbf{1}$ to $\mathbf{m_{i+1}}$ do
                   If $(\mathbf{P_{i,i+1}}\left[\mathbf{v_{j+1}^i}, \mathbf{v_c^{i+1}}\right] == \mathbf{1})$ then
                       set $\mathbf{P_{i,i+1}}\left[\mathbf{v_j^i}, \mathbf{v_c^{i+1}}\right] = \mathbf{1}$;
        3.2.2. If (there exists an index $\mathbf{a}$ so that $\mathbf{A}[\mathbf{v_j^i v_a^{i+1}}] == \mathbf{1}$) then
                   For $\mathbf{c} = \mathbf{a}$ to $\mathbf{m_{i+1}}$ do
                       set $\mathbf{P_{i,i+1}}\left[\mathbf{v_j^i}, \mathbf{v_c^{i+1}}\right] = \mathbf{1}$;
        3.2.3. For $\mathbf{d} = \mathbf{2}$ to $\mathbf{n} - \mathbf{1}$ do
                   For $\mathbf{i} = \mathbf{1}$ to $\mathbf{n} - \mathbf{d}$ do
   4.1. If $(\mathbf{A}[\mathbf{v_{m_i}^i v_a^{i+1}}] == \mathbf{1})$ then
            For $\mathbf{c} = \mathbf{1}$ to $\mathbf{m_{i+d}}$ do
        If $(\mathbf{P_{i+1,i+d}}\left[\mathbf{v_a^{i+1}}, \mathbf{v_c^{i+d}}\right] == \mathbf{1})$ then
            set $\mathbf{P_{i,i+d}}\left[\mathbf{v_{m_i}^i}, \mathbf{v_c^{i+d}}\right] = \mathbf{1}$;
   4.2. For $\mathbf{j} = \mathbf{m_i} - \mathbf{1}$ to $\mathbf{1}$ do
        4.2.1. For $\mathbf{c} = \mathbf{1}$ to $\mathbf{m_{i+d}}$ do
                   If $(\mathbf{P_{i,i+d}}\left[\mathbf{v_{j+1}^i}, \mathbf{v_c^{i+d}}\right] == \mathbf{1})$ then
                       set $\mathbf{P_{i,i+d}}\left[\mathbf{v_j^i}, \mathbf{v_c^{i+d}}\right] = \mathbf{1}$;
        4.2.2. If (there exists an index $\mathbf{a}$ so that
                   $\mathbf{A}[\mathbf{v_j^i v_a^{i+1}}] == \mathbf{1}$) then
                   For $\mathbf{c} = \mathbf{1}$ to $\mathbf{m_{i+d}}$ do
                   If $(\mathbf{P_{i+1,i+d}}\left[\mathbf{v_a^{i+1}}, \mathbf{v_c^{i+d}}\right] == \mathbf{1})$ then
                       set $\mathbf{P_{i,i+d}}\left[\mathbf{v_j^i}, \mathbf{v_c^{i+d}}\right] = \mathbf{1}$;
```

we temporarily change the coordinate system and swap "South" and "North" in Algorithms 1 and 2 to report **NE**-visible pairs. For this purpose, we should modify the edges of **OG**(**B**) (constructed by Algorithm 1) by adding upward edges, i.e. in Step 4 of Algorithm 1, we add $(\mathbf{u_j u_{j+1}})$ to **E** instead of downward ones. With this new **OG**(**B**) as the input, Algorithm 2 will calculate all **NE**-visible pairs. Consequently, all **s**-visible pairs can be reported.

To consider the time complexity of our approach, we need to know the time complexity of Algorithms 1 and 2. Algorithm 1 has a main loop of size $\mathbf{n}$, containing two loops of size $\mathbf{m_{i-1}}$ (Step 3 of Algorithm 1) and size $\mathbf{m_i}$ (Step 4 of Algorithm 1). Hence, Algorithm 1 can run in time $\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{m_i} + \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{m_{i-1}} \leq \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{2i} + \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{2(i-1)} = \mathbf{O(n^2)}$. Then we consider the time complexity of Algorithm 2. Step 2 can run in $\mathbf{O\left(n^3\right)}$, because it has a loop of size $\mathbf{n}$ that contains a loop of size $\left(\frac{\mathbf{m_i}}{\mathbf{2}}\right)$. (See Step 2 of Algorithm 2 for choosing integers $\mathbf{a}$ and $\mathbf{c}$.) Therefore, this step can run in time $\sum_{\mathbf{i=1}}^{\mathbf{n}} \left(\frac{\mathbf{m_i}}{\mathbf{2}}\right) \leq \sum_{\mathbf{i=1}}^{\mathbf{n}} \left(\frac{\mathbf{2i}}{\mathbf{2}}\right) = \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{i(2i-1)} = \mathbf{O(n^3)}$. Step 3 may also run in $\mathbf{O\left(n^3\right)}$ time, because it has a loop of size $\mathbf{n}$ containing two steps, 3.1 and 3.2. Step 3.1 has a loop of size $\mathbf{m_{i+1}}$. Step 3.2 has two nested loops of a total size $\mathbf{(m_i-1)(m_{i+1}+m_{i+1})}$. Then Step 3 runs in time $\sum_{\mathbf{i=1}}^{\mathbf{n-1}} \mathbf{(2m_i-1)m_{i+1}} \leq \sum_{\mathbf{i=1}}^{\mathbf{n-1}} \mathbf{(4i-1)\times 2\,(i+1) = O(n^3)}$. The time complexity of Step 4 of Algorithm 2 is $\mathbf{O\left(n^4\right)}$. This step has two nested loops that contain Steps 4.1 and 4.2. Step 4.1 has a loop of size $\mathbf{m_{i+d}}$. Step 4.2 has two nested loops of a total size $\mathbf{(m_i-1)(m_{i+d}+m_{i+d})}$. Then Step 4 runs in time $\sum_{\mathbf{d=2}}^{\mathbf{n-1}} \sum_{\mathbf{i=1}}^{\mathbf{n-d}} \mathbf{(2m_i-1)m_{i+d}} \leq \sum_{\mathbf{d=2}}^{\mathbf{n-1}} \sum_{\mathbf{i=1}}^{\mathbf{n-d}} \mathbf{(4i-1)\times 2\,(i+d) = O(n^4)}$. Consequently, Algorithm 2 has a time complexity $\mathbf{O(n^4)}$. This means that determining all $\mathbf{s}$-visible pairs can be done in time $\mathbf{O(n^4)}$.

Note that the number of nodes for $\mathbf{OG(B)}$ is $= \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{m_i} \leq \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{2i} = \mathbf{O(n^2)}$, meaning that there are at most $\mathbf{O(n^4)}$ paths between all pairs of nodes of this graph, and this algorithm can compute these $\mathbf{O(n^4)}$ visibility paths in time $\mathbf{O(n^4)}$. It appears that the time complexity is close to efficient.

5. Conclusion

In this study, we considered a new version of the VLSI chip design problem: the problem of finding the bar s-visibility graph ($BsVG$) of a set of given vertical bars in the plane. In previous studies, two vertical bars may connect with each other if there exists a horizontal line segment h whose endpoints are on these bars and h does not hit the other bars. However, in our model, the edges are modeled by staircase paths, i.e. two bars b_i and b_j can see each other if there exists a point on b_i that is s-visible from a point on b_j. In this model, we report more edges. These extra edges of BsVG, which were not presented in BVG, cause extra visibility paths between bars that were not reported in BVG. Actually, if there is no horizontal path between two bars, we cannot report a connection between them in the horizontal visibility model. However, in our model, a path may be found between them. This means that in our model there are more opportunities for connecting bars, and a problem that cannot be solved by previous models may find a solution in our model through these extra paths. We proposed an $O(n^4)$-time algorithm to find $BsVG$ of a given set of n vertical bars.

References

[1] Dean AM, Veytsel N. Unit bar-visibility graphs. In: 2003 Southeastern International Conference on Combinatorics, Graph Theory, and Computing; 3–7 March 2003; Boca Raton, FL, USA. pp. 161-175.

[2] Schlag M, Luccio F, Maestrini P, Lee D, Wong C. Visibility problem in VLSI layout compaction. Adv Comput Res 1985; 2: 259-282.

[3] Tamassia R, Tollis IG. A unified approach to visibility representations of planar graphs. Discrete Comput Geom 1986; 1: 321-341.

[4] Wismath SK. Characterizing bar line-of-sight graphs. In: ACM 1985 Symposium on Computational Geometry; 5–7 June 1985; Baltimore, MD, USA. New York, NY, USA: ACM. pp. 147-152.

[5] Lodi E, Pagli L. A VLSI solution to the vertical segment visibility problem. IEEE T Comput 1986; 35: 923-928.

[6] Bose P, Dean A, Hutchinson J, Shermer T. On rectangle visibility graphs. Graph Drawing 1996; 1190: 25-44.

[7] Dean AM, Ellis-Monaghan JA, Hamilton S, Pangborn G. Unit rectangle visibility graphs. Electron J Comb 2008; 15: 1-24.

[8] Dean AM, Hutchinson JP. Rectangle-visibility representations of bipartite graphs. Discrete Appl Math 1997; 75: 9-25.

[9] Streinu I, Whitesides S. Rectangle visibility graphs: characterization, construction, and compaction. Lect Notes Comput Sci 2003; 2607: 26-37.

[10] Carmi P, Friedman E, Katz MJ. Spiderman graph: Visibility in urban regions. Comp Geom 2015; 48: 251-259.