Estimating the Storage Requirements of the Rectangular and L-Shaped Corner Stitching Data Structures

Dinesh P. Mehta
Department of Computer Science
University of Tennessee Space Institute
Tullahoma, TN 37388-8897

Abstract

This paper proposes a technique for estimating the storage requirements of the Rectangular Corner Stitching (RCS) data structure [1] and the L-Shaped Corner Stitching (LCS) data structure [2] on a given circuit by studying its (the circuit's) geometric properties. This provides a method for estimating the storage requirements of a circuit without having to implement the Corner Stitching data structure, which is a tedious and time-consuming task. This technique can also be used to estimate the amount of space saved by employing the LCS data structure over the RCS data structure on a given circuit.

1 Introduction

Corner stitching is a data structuring technique proposed by Ousterhoudt [1] for representing rectangular objects in interactive VLSI layout editing systems. It is superior to other data structuring techniques for layout editors such as linked lists, bins, and neighbor pointers because it allows fast, localized algorithms for operations such as searching, deletion, creation, and compaction. The original corner stitching data structure was subsequently extended to accommodate trapezoidal [3] and curved [4] tiles. Both extensions are similar to the original in that the basic topology of the data structure remains unchanged (in the sense that each tile continues to have four corner stitching pointers, etc.).

In [2], the corner stitching data structure was extended so that it also represented L-shaped objects. This extension required a modification of the topology of the corner stitching data structure. It was seen that the LCS data structure, while retaining the advantages of corner stitching over simpler data structuring methods, required less memory than the alternative approach of partitioning each L-shaped object into two rectangular objects and then applying the original corner-stitching techniques of [1]. It was, however, expected to require more computing time to carry out its operations. The philosophy used to represent L-shaped objects can also be used on trapezoidal and curved objects to yield L-shaped trapezoidal and curved tiles.

Implementing the corner stitching data structure is a rather tedious task as indicated by the following quote [4] attributed to John Ousterhoudt, the inventor of the Corner Stitching data structure: “Corner-Stitching is pretty straightforward at a high level, but it can become much more complicated when you actually sit down to implement things, particularly if you want the implementation to run fast....”

Our own experience supports this opinion. Further, we note that the LCS data structure is even more complex than the RCS data structure, which exacerbates the problem for the LCS data structure. In view of this, we propose a technique for estimating the storage requirements of both the RCS data structure and the LCS data structure on a given circuit by studying its (the circuit’s) geometric properties. This provides a method for estimating the storage requirements of a circuit without having to implement or run the Corner Stitching data structure. This technique can also be used to estimate the amount of space saved by employing the LCS data structure over the RCS data structure on a given circuit.

We note that [1] analyses the worst case memory requirement of the RCS data structure which is realized if the circuit satisfies certain unrealistic restrictions. Our analysis is general in that it imposes no restrictions on the input circuit.

2 The Rectangular Corner Stitching (RCS) data structure

The corner-stitching data structure is obtained by partitioning the layout area into a number of rectangular tiles. There are two types of tiles: solid and vacant, both of which are stored in the corner-stitching data structure. Vacant tiles are obtained by extending horizontal lines from each of the four corners of a solid tile until they encounter another solid tile or a boundary of the layout region. The set of vacant tiles so obtained is unique for a given set of solid tiles. Further, the partitioning scheme ensures that vacant tiles are horizontally maximal; i.e., no two vacant tiles share a vertical side. Each tile has associated with it two coordinates and four pointers. For details, we refer the reader to [1].
<table>
<thead>
<tr>
<th>Tile type</th>
<th>No. of Coords</th>
<th>No. of Pts</th>
<th>No. of auxiliary bits</th>
<th>No. of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>33</td>
</tr>
<tr>
<td>L2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>33</td>
</tr>
<tr>
<td>L3</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>41</td>
</tr>
<tr>
<td>L4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>33</td>
</tr>
<tr>
<td>R in LCS</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td>R in RCS</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>25</td>
</tr>
</tbody>
</table>

Figure 1: Space requirements of tiles in LCS and RCS

3 The L-shaped Corner Stitching (LCS) data structure

The LCS data structure is obtained by partitioning the layout area into rectangular and L-shaped tiles. Once again, there are vacant tiles and solid tiles, each of which can either be rectangular or L-shaped. Vacant tiles are obtained by extending horizontal lines from some or all corners of the solid tiles, until they encounter another solid tile or a boundary of the layout region. However, the set of vacant tiles obtained is not unique for a given set of solid tiles. The partitioning scheme, however, does ensure that vacant tiles are horizontally maximal. Any L-shape may be viewed as having been obtained by coalescing two rectangles. There are four L-shape types, one for each orientation of the L-shape. The L-shapes are numbered according to the quadrant represented by the two lines meeting at the single concave corner of the L-shape. Details of how the coordinates and corner-stitching pointers for each of the four L-shapes are obtained may be found in [2].

4 Space Estimation

4.1 Memory Requirements of Individual Tiles

From the definitions of the RCS and LCS data structures, we obtain Figure 1 which is a table describing the contents of L-shapes and rectangles in LCS and rectangles in RCS. The actual memory requirements of a node in bytes (last column of the table) are obtained by assuming that pointers and coordinates, each, require 4 bytes of storage, and by placing all the remaining bits into a single byte. Note that the space required by any L-shape is less than the space required by two rectangles in RCS (therefore, it is always beneficial to coalesce two rectangles into an L-shape) and that the space required by a rectangle in LCS is equal to the space required by a rectangle in RCS.

4.2 LCS always requires less memory than RCS

Next, we show that the memory required by the entire LCS data structure is less than the memory required by the entire RCS data structure for the same set of solid tiles, even if all solid tiles are rectangles; i.e., it is more memory efficient to use the LCS approach even if the layout has no solid L-shaped tiles.

Lemma 1. For a given set of solid tiles, there is a one-to-one correspondence between the set \( S_1 \) of vacant rectangles in RCS and the set \( S_2 \) consisting of (i) vacant rectangles in LCS and (ii) rectangles obtained by partitioning each vacant L-shape in LCS into two vertically adjacent, vacant rectangles, such that each pair of related rectangles (one from \( S_1 \), the other from \( S_2 \)) have identical dimensions and positions in the layout.

Proof The Tile Creation and Tile Deletion operations on the LCS data structure [2] ensure that (i) vacant tiles are horizontally maximal and (ii) no two vacant tiles have an entire horizontal side in common. The proof follows from the fact that Tile Creation and Tile Deletion are the only two operations that directly modify the LCS data structure. Other operations that modify the LCS data structure do so by calling Tile Creation and Tile Deletion.

Theorem 1 The LCS data structure never requires more memory than the RCS data structure for the same set of solid tiles.

Proof The theorem follows from the results shown in Figure 1 and Lemma 1.

4.3 Space Requirements of Circuits that satisfy the CV property

Definition: A layout consisting of solid, rectangular tiles is said to have the CV property if (i) no two solid tiles touch each other and (ii) no two horizontal edges of these tiles are collinear and mutually visible; i.e., it is not possible to draw a horizontal line segment through two horizontal edges without the segment intersecting a third solid tile.

Lemma 2 Any two vertically adjacent, vacant rectangles in a layout satisfying the CV property can always be combined to form a vacant L-shape.

Lemma 3 Any vacant \( L_3 \) or \( L_4 \)-shape in a layout satisfying the CV property can have at most two neighboring vacant rectangles that touch its upper edge; any vacant \( L_1 \) or \( L_2 \)-space in a layout satisfying the CV property can have at most three neighboring vacant rectangles that touch either of its two upper edges.
Let $R_0$ denote the rectangle representing the bottommost vacant horizontal strip in a layout that extends from $-\infty$ to $+\infty$

**Lemma 4** Any vacant rectangle other than $R_0$ in a layout satisfying the CV property has at least one vacant L-shape that neighbors its lower edge.

**Theorem 2** The LCS data structure require between approximately $75N$ and $93.25N$ bytes of memory for a set of $N$ solid, rectangular tiles that satisfies the CV property. The RCS data structure requires approximately $100N$ bytes of memory for the same set of tiles.

**Proof** The RCS data structure representing a set of $N$ solid, rectangular tiles that satisfies the CV property contains $3N + 1$ vacant tiles [1]. Therefore, the total memory required by the RCS data structure is approximately $(4N + 1)25$ or $100N$, from row 6 of Figure 1.

Consider the LCS data structure representing the same set of solid tiles. The vacant space of the layout is now represented by vacant rectangular and L-shaped tiles. Let the number of vacant L-shaped and rectangular tiles be $l$ and $r$, respectively. Then, from Lemma 1, the number of vacant, rectangular tiles is $3N + 1 - 2l$. Let $l_i$ be the number of $L_i$-shaped tiles in LCS, for $1 \leq i \leq 4$. Then, the total memory required by the LCS data structure is $(4N + 1 - 2l)25 + (l - l_4)33 + l_441$. In the best case, $l = [3N + 1/2]$ and $l_4 = 0$ giving a memory requirement of approximately $25N + (3N/2)33$ or $75N$, a saving of 25%.

We omit the derivation of the worst case expression.

Actual VLSI circuits typically do not satisfy the CV property. Hence, the bounds of Theorem 2 do not apply to actual VLSI circuits. We now generalize Theorem 2 by removing the restriction that the set of input rectangles must satisfy the CV property.

### 4.4 Space Requirements of Circuits that do not satisfy the CV property

We first classify the violations of the CV property as follows:

1. A horizontal violation is said to occur between two solid tiles $A$ and $B$ iff there exists a horizontal segment $h$ such that
   1. A segment of a horizontal side of $A$ and a segment of a horizontal side of $B$ are contained in $h$, and
   2. no portion of any other tile is contained in $h$.

There are four types of horizontal violations:

- (a) Same side, no touching (sn) horizontal violation: Both $A$ and $B$ are on the same side of the horizontal segment, and $A$ and $B$ do not touch each other. Let $h_{sn}$ denote the number of such violations in the layout.

- (b) Same side, touching (st) horizontal violation: Both $A$ and $B$ are on the same side of the horizontal segment, and $A$ and $B$ touch each other. Let $h_{st}$ denote the number of such violations in the layout.

- (c) Opposite side, no touching (on) horizontal violation: $A$ and $B$ are on opposite sides of the horizontal segment, and $A$ and $B$ do not touch each other. Let $h_{on}$ denote the number of such violations in the layout.

- (d) Opposite side, touching (ot) horizontal violation: $A$ and $B$ are on opposite sides of the horizontal segment, and $A$ and $B$ touch each other. Let $h_{ot}$ denote the number of such violations in the layout.

**Figure 2:** Example layout with violations

$A$ and $B$ are on opposite sides of the horizontal segment, and $A$ and $B$ do not touch each other. Let $h_{ot}$ denote the number of such violations in the layout.

(d) Opposite side, touching (ot) horizontal violation: $A$ and $B$ are on opposite sides of the horizontal segment, and $A$ and $B$ touch each other. Let $h_{ot}$ denote the number of such violations in the layout.

(2) Vertical violations are of two types:

- (a) A same side (ss) vertical violation is said to occur between two solid tiles $A$ and $B$ iff there exists a vertical segment $v$ such that $v$ is the union of a vertical side of $A$ and a vertical side of $B$, and $A$ and $B$ are on the same side of $v$.

- (b) Let $v$ denote the number of such violations in the layout. A vertical group of tiles is a maximal list of solid tiles ($T_1, T_2, ..., T_m$) such that there exists a vertical segment $v$ where (i) $v$ is the union of vertical sides of $T_i$ for $0 < i \leq m$, and (ii) $T_i$, for $0 < i \leq m$, are on the same side of $v$. $v$ is known as the common side of the vertical group.

An opposite side (os) vertical violation is said to occur between two vertical groups $G$ and $H$ iff there exists a vertical segment $v$ such that (i) $v$ is the intersection of the common side of $G$ and the common side of $H$, (ii) $G$ and $H$ are on opposite sides of $v$, and (iii) $v$ consists of more than one point. Let $v_o$ denote the number of such violations in the layout.

**Lemma 5** A layout satisfies the CV property if and only if it does not contain any of the violations described above.

Figure 2 shows a layout, consisting of 19 solid tiles and 26 vacant tiles, that violates the CV property. We first enumerate the horizontal violations: (a) sn violations: (C,G), (G,J), (J,Q), (D,H), (O,R). (b) st violations: (B,C), (c) on violations: (G,I), (E,I), (R,S). (d) ot violations: (A,B), (C,D), (G,H), (I,L), (L,N), (J,K), (K,M), (O,P), (Q,R).

Next, we enumerate the vertical violations: (a) same side violations: (A,B), (A,B), (G,H), (I,L), (L,N), (J,K), (J,K), (K,M), (K,M), (O,P), (O,P). (b) opposite side violations: (AB,C), (E,F), (ILNJKM), (ILNLP). So, $h_{sn} = 5$, $h_{st} = 1$, $h_{on} = 3$, $h_{ot} = 9$, $v_s = 10$, and $v_o = 4$; i.e., the total number of violations in the layout is 32. This example verifies the following theorem:
<table>
<thead>
<tr>
<th>$k$</th>
<th>0</th>
<th>$N/3$</th>
<th>$2N/3$</th>
<th>$N$</th>
<th>$2N$</th>
<th>$5N/2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max savings</td>
<td>25</td>
<td>24.2</td>
<td>23.2</td>
<td>22</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>Min savings</td>
<td>6.75</td>
<td>5.7</td>
<td>4.5</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3: Percentage savings in space by using the LCS data structure

**Lemma 6** The RCS data structure representing a set of $N$ solid, rectangular tiles with $k$ violations of the CV property contains $3N + 1 - k$ vacant tiles.

**Proof** A constructive proof is used to prove the lemma. The proof consists of three steps. At each step, violations are eliminated by perturbing sides of the solid tiles. It can be shown that, in each step, the number of violations eliminated is equal to the increase in the number of vacant rectangles. Eventually, the layout contains no violations and, from Lemma 5, the CV property is satisfied. So, the number of vacant rectangles in the final layout is $3N + 1$. The total increase in the number of vacant rectangles is equal to the total number of violations eliminated, which is $k$. So, the number of vacant rectangles in the initial layout must have been $3N + 1 - k$. We omit the details of the proof because of space limitations. □

We now obtain a relationship between the number of vacant rectangles and vacant L-shapes in an LCS representation of a layout consisting of $N$ solid, rectangular tiles and $k$ violations of the CV property:

**Lemma 7** The number of vacant rectangles $v$ in an LCS representation of a layout consisting of $N$ solid, rectangular tiles and $k$ violations of the CV property is at most $2(l_3 + l_4) + 3(l_1 + l_2) + k + 1$.

**Theorem 3** The LCS data structure requires $100(25N - 8.5k)/(100N - 25k)$ to $100(6.75N - 4.5k)/(100N - 25k)$% less memory than the RCS data structure for a set of $N$ solid tiles with $k$ violations of the CV property.

The table in Figure 3 shows minimum and maximum percentage savings in space for various values of $k$. $k$ is expressed as a fraction of $N$, the number of solid tiles in the layout. The table entries are computed by substituting for $k$ in the expressions of Theorem 3. The significance of Theorem 3 is that we can obtain bounds on the space required by a layout by simply knowing the number of solid tiles and the number of violations in the layout. Thus, we can estimate the memory required by a layout without having to create its LCS data structure. Since the algorithms for computing violations are expected to be easier to implement and faster than those for constructing the LCS data structure, the estimation process is faster.

One apparent disadvantage of the estimation process for L-shaped tiles is that the estimate of the space required could lie anywhere in a range of values (as opposed to being an exact value). However, the space required by the LCS data structure is inherently a non-unique number since it is influenced by the order in which tiles are inserted into the data structure. Finally, we note that a tighter lower bound than the one of Theorem 3 may be obtained if the number of each kind of violation in the layout is provided.

5 Conclusions and Future Work

We are currently developing and implementing efficient geometric algorithms for computing the number of violations in a given circuit. From these, we will be able to determine the exact space requirements of the RCS data structure and upper and lower bounds on the space requirements of the LCS data structure. We propose to experiment with the LCS data structure to determine how close to the theoretical lower bound the actual memory requirements are.

We have assumed in this paper that all solid tiles are rectangles. We expect that the techniques of this paper can be extended to solve the case where some solid tiles are L-shaped. We expect that these techniques can be extended to the trapezoidal and curved versions of the Corner Stitching data structure as well. Thus, these techniques are expected to be of use to a CAD System designer in choosing which data structure to use (without having to purchase or implement it locally) for his/her circuits.

References


