Construction of All Multilayer Monolithic Rectilinear Steiner Minimum Trees on the 3D Hanan Grid for Monolithic 3D IC Routing

Sheng-En David Lin
dl23ee.lin@wsu.edu
Washington State University
Pullman, Washington

Dae Hyun Kim
daehyun@eecs.wsu.edu
Washington State University
Pullman, Washington

ABSTRACT

Monolithic three-dimensional (3D) integration enables stacking multiple ultra-thin silicon tiers in a single package, thereby providing smaller footprint area, shorter wirelength, higher performance, and lower power consumption than conventional planar fabrication technologies. Physical design of monolithic 3D integrated circuits (ICs) requires several design steps such as 3D placement, 3D clock-tree synthesis, 3D routing, and 3D optimization. Among the steps, 3D routing is very time-consuming due to numerous routing blockages. Thus, 3D routing is typically performed in two sub-steps, monolithic inter-layer via (MIV) insertion and tier-by-tier routing. In this paper, we propose an algorithm to build a routing topology database that can be used to construct all multilayer monolithic rectilinear Steiner minimum trees on the 3D Hanan grid. The database will help 3D routers reduce the runtime of the MIV insertion step and improve the quality of the 3D routing.

CCS CONCEPTS

• **Hardware → 3D integrated circuits**; **Wire routing**.

KEYWORDS

Monolithic 3D Integrated Circuits, Rectilinear Steiner Minimum Tree, RSMT, Routing, Wirelength, Congestion

ACM Reference Format:
Sheng-En David Lin and Dae Hyun Kim. 2019. Construction of All Multilayer Monolithic Rectilinear Steiner Minimum Trees on the 3D Hanan Grid for Monolithic 3D IC Routing. In *2019 International Symposium on Physical Design (ISPD '19), April 14–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3299902.3309749

1 INTRODUCTION

Monolithic three-dimensional (3D) integration stacks very thin silicon tiers and electrically connects transistors in different tiers by monolithic inter-layer vias (MIVs). An MIV is similar to a through-silicon via (TSV), but the width and the z-directional length of an MIV are much shorter than those of a TSV. Thus, MIV insertion has almost negligible area and capacitance overheads in the monolithic 3D integrated circuit (IC) layout design. However, inserting too many MIVs into a 3D IC layout causes serious routing congestion because planar wires should be connected to the MIVs and routing of the planar wires requires much larger area than the MIV area. Thus, 3D placement in general tries to minimize the number of MIVs inserted into a layout [1, 10] and 3D routing should also minimize the number of MIVs.

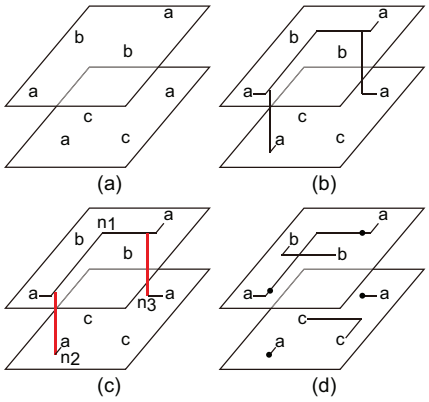

Figure 1: 3D-net-first routing. (a) Three nets to route. (b) 3D routing topology generation for the 3D net. (c) MIV insertion. (d) Tier-by-tier routing.

3D routing routes 2D and 3D nets.[1] 3D routing algorithms could route all the 2D and 3D nets in a given design separately or simultaneously. For example, the routing methodology in [3] routes 3D nets first, then routes 2D nets. On the other hand, the routing methodology used in [9] routes 2D and 3D nets simultaneously by using a commercial tool and modified library files. The latter, however, has some drawbacks compared to the former. According to our own routing simulations using modified library files, the runtime of the simultaneous routing of 2D and 3D nets increases significantly as the complexity (the average net degree, the number of tiers, and the number of instances) of a given design goes up. On the other hand, if 3D nets are routed first, we can route 2D nets in each tier separately (tier-by-tier routing). Thus, the 3D-net-first routing methodology has been used extensively in the literature [3, 5, 10, 11].

The 3D-net-first routing methodology finds MIV locations for each 3D net, inserts MIVs into the locations, and decomposes the 3D

[1] A 2D (3D) net is a net connecting instances placed in a tier (different tiers).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ISPD '19, April 14–17, 2019, San Francisco, CA, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6253-5/19/04…$15.00
https://doi.org/10.1145/3299902.3309749

net into multiple 2D nets. Figure 1 shows an example. In Figure 1(a), eight pins are spread out in two tiers. The net connecting four a pins is a 3D net, whereas the other two nets connecting two b and two c pins are 2D nets. In Figure 1(b), a 3D routing topology using two z-directional edges is constructed for the 3D net. The z-directional edges are replaced by MIVs and the 3D net is decomposed into three 2D nets, n_1, n_2, and n_3 in Figure 1(c). By the decomposition of the 3D net into the three 2D nets, routing of the 3D net is converted into routing of the three 2D nets, two in the bottom tier and one in the top tier. Finally, the 2D nets are routed in each tier in Figure 1(d).

As mentioned above, 3D routing should minimize the number of MIVs used to route 3D nets. In addition, 3D routing should evenly distribute the MIVs over the entire layout area to minimize routing congestion around the MIVs. At the same time, 3D routing should evenly distribute planar wires routing 3D nets over the layout area so that routing of 2D nets does not suffer from serious routing congestion in specific tiers. The MIV insertion methodologies used in the literature, however, do not control the MIV count, MIV locations, and planar wires of 3D nets effectively. For example, the 3D rectilinear Steiner tree (RST) algorithms used in [4, 5, 10, 11] do not guarantee the minimization of the MIV count. The MIV insertion algorithm used in [3] minimizes the MIV count, but fails to minimize the planar wirelength. It is also possible to use multilayer obstacle-avoiding rectilinear Steiner tree (MLOARST) construction algorithms to minimize both the planar wirelength and the number of MIVs [6, 8]. However, the MLOARST construction algorithms do not generate multiple topologies having different MIV locations and planar wire distributions.

In this paper, we propose an algorithm to build a database that can be used to construct all multilayer monolithic rectilinear Steiner minimum trees (MMRSMTs) on the 3D Hanan grid for given pin locations for up to six-pin nets and four tiers. MMRSMTs have the shortest planar wirelength with the minimum number of MIVs, so MIV insertion algorithms can use the database to effectively optimize MIV locations and planar wires of 3D nets. We also propose database size reduction techniques for practical use of the database in academia and industry. To the best of our knowledge, this is the first work on constructing all MMRSMTs on the 3D Hanan grid.

2 PRELIMINARIES AND PROBLEM FORMULATION

In this section, we explain terminologies used in this paper, review the construction of an RSMT in FLUTE [2] and construction of all RSMTs in [7], and formulate the problem we solve in this paper.

2.1 Terminologies

2.1.1 2D and 3D Hanan Grids. Suppose a finite set S of points are given in the 2D plane. The ***2D Hanan grid*** constructed from S is a graph $G_S = (V_S, E_S)$. The vertex set of G_S is $V_S = \{(x, y)|x \in X_S, y \in Y_S\}$ where $X_S = \{x_1, ..., x_L\}(x_1 < ... < x_L)$ and $Y_S = \{y_1, ..., y_M\}(y_1 < ... < y_M)$ are the sets of the x- and y-coordinates of all the points in S, respectively. The edge set of G_S is $E_S = E_{S,X} \cup E_{S,Y}$ where

$$E_{S,X} = \{(v_1, v_2)|v_1 = (x_{i<L}, y_j) \in V_S, v_2 = (x_{i+1}, y_j) \in V_S\},$$
$$E_{S,Y} = \{(v_1, v_2)|v_1 = (x_i, y_{j<M}) \in V_S, v_2 = (x_i, y_{j+1}) \in V_S\}.$$

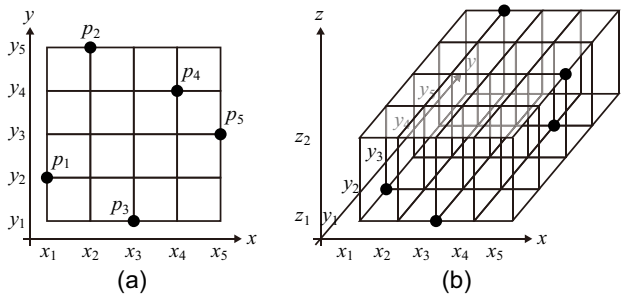

Figure 2: 2D and 3D Hanan grids.

In other words, $E_{S,X}$ and $E_{S,Y}$ are the sets of the x- and y-directional edges of G_S, respectively. Figure 2(a) shows the 2D Hanan grid constructed for the five points located at (x_1, y_2), (x_2, y_5), (x_3, y_1), (x_4, y_4), and (x_5, y_3).

Suppose a finite set T of points are given in the 3D space. The ***3D Hanan grid*** constructed from T is a graph $G_T = (V_T, E_T)$. The vertex set of G_T is $V_T = \{(x, y, z)|x \in X_T, y \in Y_T, z \in Z_T\}$ where $X_T = \{x_1, ..., x_L\}(x_1 < ... < x_L)$, $Y_T = \{y_1, ..., y_M\}(y_1 < ... < y_M)$, and $Z_T = \{z_1, ..., z_N\}(z_1 < ... < z_N)$ are the sets of the x-, y-, and z-coordinates of all the points in T, respectively. The edge set of G_T is $E_T = E_{T,X} \cup E_{T,Y} \cup E_{T,Z}$ where

$$E_{T,X} = \{(v_1, v_2)|v_1 = (x_{i<L}, y_j, z_k) \in V_T, v_2 = (x_{i+1}, y_j, z_k) \in V_T\},$$
$$E_{T,Y} = \{(v_1, v_2)|v_1 = (x_i, y_{j<M}, z_k) \in V_T, v_2 = (x_i, y_{j+1}, z_k) \in V_T\},$$
$$E_{T,Z} = \{(v_1, v_2)|v_1 = (x_i, y_j, z_{k<N}) \in V_T, v_2 = (x_i, y_j, z_{k+1}) \in V_T\}.$$

In other words, $E_{T,X}$, $E_{T,Y}$, and $E_{T,Z}$ are the sets of the x-, y-, and z-directional edges of G_T, respectively. Figure 2(b) shows the 3D Hanan grid constructed for the five points located at (x_1, y_2, z_1), (x_2, y_5, z_2), (x_3, y_1, z_1), (x_4, y_4, z_1), and (x_5, y_3, z_2).

2.1.2 Position Sequence. x_+- and x_--directions are the directions along which x-coordinates increase and decrease, respectively. y_+-, y_--, z_+-, and z_--directions are defined similarly.

Suppose a finite set $P = \{p_1, ..., p_n\}$ of n distinct pins[2] are given in an $n \times n$ grid. Let the x-coordinates of the y-directional edges be x_1 to x_n from the left and the y-coordinates of the x-directional edges be y_1 to y_n from the bottom as shown in Figure 2(a). Then, we denote sorting the pins in the increasing (+) and decreasing (−) order of their c-coordinates (c is x or y) by c_+ and c_-, respectively. In Figure 2(a), for example, y_+ sorting leads to the ordered list $L_1 = (p_3, p_1, p_5, p_4, p_2)$ and x_- sorting leads to the ordered list $L_2 = (p_5, p_4, p_3, p_2, p_1)$. Suppose we obtain an ordered list $L = (l_1, ..., l_n)$ from c_+ or c_- sorting. Then, we can obtain the $\bar{c}$-coordinates (if c is x (or y), $\bar{c}$ is y (or x)) of the pins in the $\bar{c}_+$- or $\bar{c}_-$-direction from L. For example, we obtain (31542) and (35124) if we extract the x-coordinates of the pins in L_1 in the x_+ and x_- directions, respectively. The ***position sequence*** $\Gamma_{(R,C)}$ for P is a sequence $(s_1 s_2 ... s_n)$ where $R \in \{c_+, c_-\}$, $C \in \{\bar{c}_+, \bar{c}_-\}$, and s_i is the $\bar{c}$-coordinate of the i-th pin in the $\bar{c}_+$ or $\bar{c}_-$-direction in the list of the pins sorted by c_+ or c_- sorting. For example, assume that (R, C) is (y_+, x_+) and P is the set of pins in Figure 2(a). Then, we first sort the pins along the y_+-direction, which leads to the ordered list

[2] If the coordinate of p_i is (x_{p_i}, y_{p_i}), $x_{p_i} \neq x_{p_j}$ and $y_{p_i} \neq y_{p_j}$ for any i and j $(i \neq j)$.

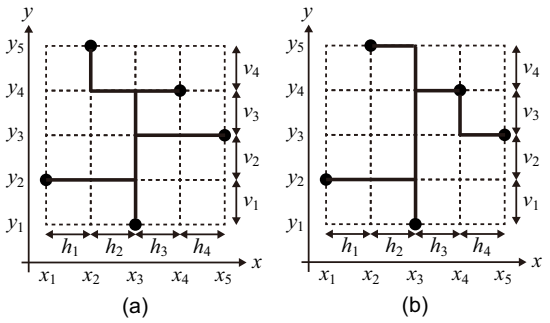

Figure 3: Two trees constructed on the 2D Hanan grid.

$(p_3, p_1, p_5, p_4, p_2)$, and obtain $\Gamma_{(y_+,x_+)} = (31542)$. Similarly, $\Gamma_{(y_+,x_-)}$ is (35124), $\Gamma_{(y_-,x_+)}$ is (24513), $\Gamma_{(y_-,x_-)}$ is (42153), $\Gamma_{(x_+,y_+)}$ is (25143), $\Gamma_{(x_+,y_-)}$ is (41523), $\Gamma_{(x_-,y_+)}$ is (34152), and $\Gamma_{(x_-,y_-)}$ is (32514).

2.1.3 Potentially Optimal Wirelength Vector and Potentially Optimal Steiner Tree. The wirelength of an RST on the 2D Hanan grid can be expressed as a linear combination of the x- and y-directional edge vectors representing the tree as explained in [2]. For example, the wirelength of the tree in Figure 3(a) is

$$L = 1 \cdot h_1 + 2 \cdot h_2 + 2 \cdot h_3 + 1 \cdot h_4 + 1 \cdot v_1 + 1 \cdot v_2 + 1 \cdot v_3 + 1 \cdot v_4, \quad (1)$$

which can also be expressed as

$$L = (1, 2, 2, 1, 1, 1, 1, 1) \cdot (h_1, h_2, h_3, h_4, v_1, v_2, v_3, v_4). \quad (2)$$

The first vector $(1, 2, 2, 1, 1, 1, 1, 1)$ is called a ***coefficient vector*** and the second vector $(h_1, h_2, h_3, h_4, v_1, v_2, v_3, v_4)$ is called an ***edge length vector***. The edge length vector is a constant vector for a given set of pin locations. However, the coefficient vector is dependent on the topology of the constructed tree. For example, the wirelength of the tree in Figure 3(b) is

$$L = 1 \cdot h_1 + 2 \cdot h_2 + 1 \cdot h_3 + 1 \cdot h_4 + 1 \cdot v_1 + 1 \cdot v_2 + 2 \cdot v_3 + 1 \cdot v_4, \quad (3)$$

whose coefficient vector is $(1, 2, 1, 1, 1, 1, 2, 1)$. Thus, the two trees in Figure 3(a) and (b) have the same edge length vector, but different coefficient vectors.

For a given set of pin locations, a coefficient vector $V = (c_1, ..., c_k)$ becomes a ***potentially optimal wirelength vector (POWV)*** if it satisfies the following conditions [2]:

- There exists an RST that connects all the pins and uses the edges specified in the coefficient vector V on the Hanan grid constructed for the pins.
- For the same pin locations, there is no other coefficient vector $V' = (c'_1, ..., c'_k)$ satisfying $c'_i \leq c_i$ for all $i = 1, ..., k$.

An RST corresponding to a POWV is called a ***potentially optimal Steiner tree (POST)*** [2]. The RST shown in Figure 3(a) is a POST for POWV $(1, 2, 2, 1, 1, 1, 1, 1)$, which belongs to position sequence (31542). Similarly, the RST shown in Figure 3(b) is a POST for POWV $(1, 2, 1, 1, 1, 1, 2, 1)$, which belongs to the same position sequence.

2.2 Construction of an RSMT and All RSMTs on the Hanan Grid

FLUTE constructs an RSMT by a lookup table [2]. The lookup table consists of all position sequences, all POWVs belonging to each position sequence, and one POST for each POWV. Whenever a set of pin locations is given, FLUTE first finds the position sequence of the pin locations, then compares the wirelengths of all the POWVs belonging to the position sequence by calculating the inner product of each POWV and the edge length vector for the given pin locations. Then, FLUTE returns the POST of the POWV having the minimum wirelength. If multiple POWVs have the same wirelength, FLUTE can return the POSTs of all the POWVs having the minimum wirelength. The returned POSTs are RSMTs for the pin locations. For more details, we refer readers to [2].

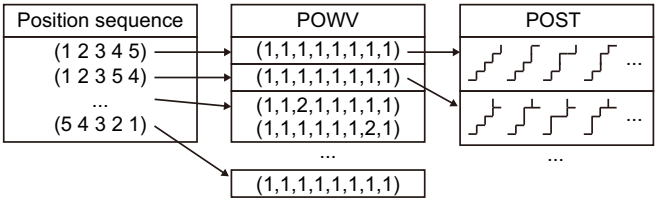

Figure 4: An overview of the lookup table of all POSTs in [7].

FLUTE contains only one POST per POWV, but a POWV can have multiple POSTs. If the lookup table contains all POSTs for each POWV, numerous CAD algorithms can also benefit from the POSTs. For example, congestion-aware global routing can reduce routing congestion by finding all RSMTs and choosing the best one for each net. Thus, Lin generated a lookup table storing all POSTs in [7]. Figure 4 shows an overview of the lookup table of all POSTs. The algorithm of finding all POSTs uses a binary decision tree with several speed-up techniques to reduce the runtime. For more details, we refer readers to [7].

2.3 Multilayer Monolithic Rectilinear Steiner Minimum Trees

The following three definitions define a 3D rectilinear tree, a 3D rectilinear Steiner tree, and a 3D rectilinear Steiner minimum tree.

Definition 1. *A **3D rectilinear tree** is a tree having only x-, y-, and z-directional edges and connecting all given pins.*

Definition 2. *A **3D rectilinear Steiner tree (3D RST)** is a 3D rectilinear tree with Steiner points. A Steiner point is a non-pin vertex having more than two edges.*

Definition 3. *A **3D rectilinear Steiner minimum tree (3D RSMT)** is a 3D RST having the minimum wirelength.*

The wirelength of a 3D rectilinear tree is computed by the sum of the lengths of all the x-, y-, and z-directional edges in the tree. When 3D RSTs or 3D RSMTs are used for routing of 3D IC layouts, the length of a vertical via is adjusted so that the actual overhead of the vertical via can be properly taken into account in the design. In the monolithic 3D IC layout design, the area and capacitance overhead of an MIV is negligible, so we can set the length of an MIV to zero during 3D routing. However, minimizing the number of MIVs inserted in the layout is still crucial. Thus, we define a multilayer monolithic rectilinear Steiner minimum tree as follows:

Definition 4. *A **multilayer monolithic rectilinear Steiner minimum tree (MMRSMT)** is a 3D RSMT using the minimum number of z-directional edges with zero z-directional edge length.*

Since an MMRSMT is a 3D RSMT, it has the shortest planar wirelength. Thus, if we project all the edges in an MMRSMT onto the xy plane, the projection becomes a 2D RSMT. In other words, *an MMRSMT can be constructed from a 2D RSMT by properly placing x- and y-directional edges of the 2D RSMT in a 3D grid and inserting z-directional edges.* In addition, we obtain 2D RSMTs from POSTs as mentioned in the previous section. Thus, we can construct an MMRSMT from a POST. We define a 3D potentially optimal Steiner tree as follows:

Definition 5. *Suppose a set of xy-distinct*[3] *pin locations is given. Let the set be* $P = \{(x_1, y_1, z_1), ..., (x_n, y_n, z_n)\}$. *Let the set of the projections of the pins onto the xy plane be* $P' = \{(x_1, y_1), ..., (x_n, y_n)\}$. *Let a POST constructed for* P' *be* $G' = (V', E')$. *Let the coordinate of* e' *in* E' *be* $e'(i, j)$. *A* ***3D potentially optimal Steiner tree (3D POST)*** *is a tree* T *that connects all the pins in* P, *uses the minimum number of z-directional edges in the 3D Hanan grid* G *constructed from* P, *and uses one of the edges among* $e(i, j, k = 0, ..., t-1)$ *in* G *for each* $e'(i, j) \in E'$. t in the definition is the number of tiers. From now, we denote the POSTs in the database of all POSTs in [7] by 2D POSTs to distinguish them from 3D POSTs.

In summary, if we have a database of all 3D POSTs, we can build all MMRSMTs for a given set of pin locations in a very short period of time. In this paper, we build a database of all 3D POSTs for all possible relative pin locations in two, three, and four tiers.

3 CONSTRUCTION OF ALL 3D POSTS

In this section, we present an algorithm to construct all 3D POSTs on the 3D Hanan grid for a given set of pin locations and a 2D POST for them. Figure 5 shows a 3D grid, pin and non-pin vertices, x-, y-, and z-directional edges, and notations used in this paper.

3.1 Construction of All 3D POSTs

The input to the algorithm is a set P of pin locations and a 2D POST, $G_2 = (V_2, E_2)$ constructed for the projections of the pins onto the xy plane. In Figure 6(a), for example, the coordinates of the three pins are (0, 0, 0), (1, 2, 2), and (2, 1, 1). The position sequence $\Gamma_{(y_+, x_+)}$ of the projections is (132) and the 2D POST shown in Figure 6(a) belongs to the position sequence.

Algorithm 1 shows the algorithm for constructing all 3D POSTs. We first set the *visited* variables of all the edges in E_2 to false (Line 1). Then, we sort the edges in E_2 and store the result in an ordered set E'_2 (Line 2). The function **sort_edges** chooses a pin vertex in G_2 and performs the breadth-first search starting from the pin vertex until all the pin vertices are reached. Whenever it goes through an edge, the function inserts the edge into E'_2. This order reduces the runtime of the algorithm. For example, the sort_edges function starts from the pin vertex (0, 0, 0) in Figure 6(a). Then, E'_2 becomes $(e_x(0, 0), e_y(0, 1), e_x(1, 1), e_y(1, 1))$. Then, we construct a 3D grid $G_3 = (V_3, E_3)$ from G_2 and P (Line 3). The **construct_3D_grid** function expands the 2D graph G_2 to a 3D graph G_3 as shown in Figure 6(b). The expansion adds vertices $v(i, j, k = 0, \ldots, t-1)$ to G_3 for each vertex $v'(i, j)$ in G_2 where t is the number of tiers. Similarly, the expansion adds $e_x(i, j, k = 0, \ldots, t-1)$ to G_3 for each

[3]If the coordinate of p_i is $(x_{p_i}, y_{p_i}, z_{p_i})$, $x_{p_i} \neq x_{p_j}$ and $y_{p_i} \neq y_{p_j}$ for any i and j $(i \neq j)$.

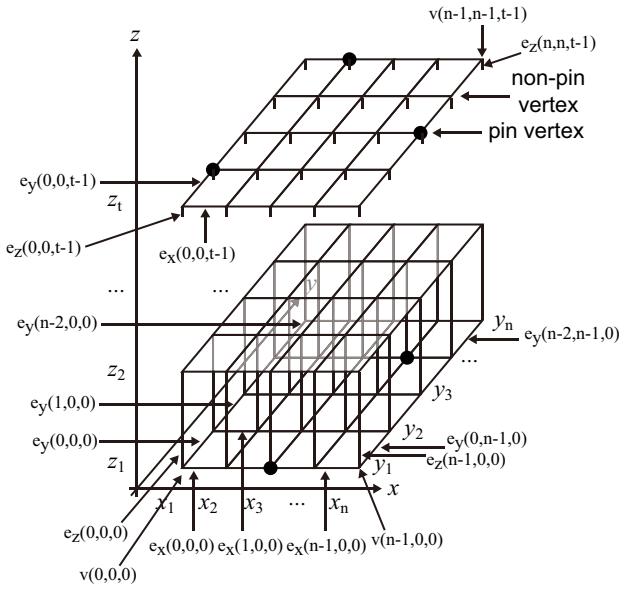

Figure 5: An $n \times n \times t$ 3D grid, pin and non-pin vertices, and indices for x-, y-, and z-directional edges.

$e'_x(i, j)$ in G_2, $e_y(i, j, k = 0, \ldots, t-1)$ to G_3 for each $e'_y(i, j)$ in G_2, and $e_z(i, j, k = 0, \ldots, t-2)$ to G_3 for each $v'(i, j)$ in G_2. Then, we set the *used* variables of all the edges in E_3 to false, which means that the edges are not used yet (Line 4). T is a set of graphs storing all the 3D POSTs, nr_MIVs is a variable storing the number of MIVs used in G_3, and min_nr_MIVs is a variable storing the minimum number of MIVs used in the 3D POSTs (Line 5). Then, we call the **recur_con** function to recursively construct all 3D POSTs for the given pin locations and the 2D POST (Line 6). Once the algorithm ends, we return T (Line 7).

The **recur_con** function starts from checking the given index, which is used to access the edges in E'_2. The edge index is greater than the number of edges in E'_2 when there is no more edge to process in G_3 (Line 1), which means that G_3 is a 3D graph connecting all the pins. In this case, if the total number of MIVs used in G_3 is equal to the minimum number of MIVs used in the best graphs found until now, we add it to T (Line 3). However, if the total number of MIVs used in G_3 is less than the minimum number of MIVs used in the best graphs found until now, all the graphs in T use more MIVs than G_3, so we empty T (Line 5), add G_3 to T, and update min_nr_MIVs (Line 6).

If the edge index is less than than the size of E'_2 (Line 10), we visit the edge in E'_2 indexed by the edge index variable (Line 11) and try using edges in G_3 corresponding to the indexed edge (Line 12 to Line 32). First, suppose $e'_d(i, j)$ is E'_2[index] where d is either x or y. Then, we try using $e_d(i, j, k)$ in G_3 for each $k = 0, ..., t-1$ (Line 14). In Figure 6(c), $e_x(0, 0, 0)$ in G_3, which corresponds to the first edge $e_x(0, 0)$ in E'_2, is used. Then, if e is x-directional, we obtain its left vertex in G_2, otherwise we obtain its bottom vertex in G_2 and assign it to v (Line 15). Then, we find the bottommost and topmost tiers that should be connected along the z-axis through v in G_3 by the **get_min_max_tier** function (Line 16 to Line 19). The function

Function: Construct all 3D POSTs for P and G_2.
Input: Pin locations (P) and a 2D POST $G_2 = (V_2, E_2)$.
1: e.visited = false for all $e \in E_2$;
2: Ordered set E_2' = **sort_edges** (G_2);
3: $G_3 = (V_3, E_3)$ = **construct_3D_grid** (G_2, P);
4: e.used = false for all $e \in E_3$;
5: $T = \{\}$; nr_MIVs = 0; min_nr_MIVs = ∞;
6: Call **recur_con** (T, G_2, G_3, E_2', 0, nr_MIVs, min_nr_MIVs);
7: Return T;
Function: recur_con (T, G_2, G_3, E_2', index, nr_MIVs, min_nr_MIVs)
1: **if** index $\geq |E_2'|$ **then**
2: **if** nr_MIVs == min_nr_MIVs **then**
3: Add G_3 to T;
4: **else if** nr_MIVs < min_nr_MIVs **then**
5: Clear T;
6: Add G_3 to T; min_nr_MIVs = nr_MIVs;
7: **end if**
8: return;
9: **end if**
10: $e = E_2'$[index];
11: e.visited = true;
12: **for** tier = 0 ; tier < # tiers ; tier = tier + 1 **do**
13: $e_3 = E_3[e.x][e.y][tier]$;
14: e_3.used = true;
15: v = e.left (or e.bottom);
16: min_t1 = max_t1 = 0;
17: **if** All the edges connected to v in G_2 have been visited **then**
18: min_t1, max_t1 = **get_min_max_tier** (v, G_2, G_3);
19: **end if**
20: v = e.right (or e.top);
21: min_t2 = max_t2 = 0;
22: **if** All the edges connected to v in G_2 have been visited **then**
23: min_t2, max_t2 = **get_min_max_tier** (v, G_2, G_3);
24: **end if**
25: delta = (max_t1 - min_t1) + (max_t2 - min_t2);
26: nr_MIVs = nr_MIVs + delta;
27: **if** nr_MIVs $\leq$ min_nr_MIVs **then**
28: **recur_con** (T, G_2, G_3, E_2', index+1, nr_MIVs, min_nr_MIVs);
29: **end if**
30: nr_MIVs = nr_MIVs - delta;
31: e_3.used = false;
32: **end for**
33: e.visited = false;

Algorithm 1: Construction of all 3D POSTs for a given 2D POST and pin locations in 3D.

finds all the visited edges connected to e in G_2, obtains the tiers of the edges in G_3 corresponding to the visited edges, and finds the bottommost and topmost tiers. In addition, if v is a pin vertex, the z-coordinate of the pin should be included in the computation of the range of the tiers. We repeat the same process for the right vertex of e (or top vertex if e is y-directional) (Line 20 to Line 24).

If we have visited all the edges connected to the left (bottom) and/or right (top) vertices of e, we can find the z-directional edges required to connect to the pin and the edges along the z-axis at the vertices. From the z-directional edges, we obtain the number of MIVs (Line 25). If the total number of MIVs currently used in G_3 is less than or equal to the minimum number of MIVs used in the best graphs found until now, we move on to the next edge in

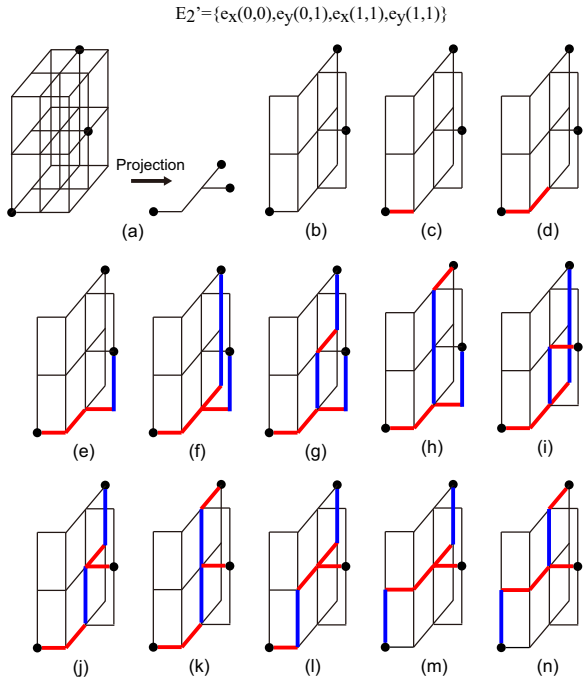

Figure 6: Construction of all 3D POSTs in three tiers for pins $(0, 0, 0)$, $(1, 2, 2)$, $(2, 1, 1)$. (a) A 2D POST is given. (b) The construct_3D_grid function creates a 3D grid structure. (c)-(n) 3D POST construction. The red edges are used planar edges and the blue edges are used z-directional edges.

E_2' (Line 28). Otherwise, the current graph uses more MIVs than the best graphs found until now, so we do not need to proceed to the next edge. Once the recursive function call ends (Line 28), we readjust the number of MIVs used in G_3 (Line 30) and try using the edge above e (Line 31 and Line 32).

In Figure 6(c), for example, $e_x(0, 0, 0)$ is in Tier 0 and the left vertex of $e_x(0, 0, 0)$ is a pin vertex, which is also in Tier 0. Thus, both the bottommost and topmost tiers for the vertex are Tier 0. Then, we move on to $e_y(0, 1)$ in E_2' and try using $e_y(0, 1, 0)$ in G_3 in Figure 6(d). The *used* variables of all the edges connected to the bottom vertex of $e_y(0, 1)$ are *true* at this point and all the edges are placed in Tier 0. Thus, we do not need to add any z-directional edges above the vertex. Then, we process the next edge $e_x(1, 1)$ in E_2' in Figure 6(e). The right vertex of $e_x(1, 1)$ is connected to the pin located at $(2, 1)$, which corresponds to the pin located at $(2, 1, 1)$ in G_3, so the bottommost and topmost tiers at the vertex are Tier 0 and Tier 1, respectively. Thus, we use $e_z(2, 1, 0)$ in G_3, which is inserting an MIV into the location. When we also try using $e_y(1, 1, 0)$ in Figure 6(f), we finally construct a 3D graph connecting all the pins. The total number of MIVs is three. Similarly, the total numbers of MIVs in the 3D graphs in Figure 6(g), (h), and (i) are all three. However, the 3D graph in Figure 6(j) uses two MIVs. At this time, T contains all the 3D graphs found in Figure 6(f), (g), (h), and (i), so we delete all of them from T and add the 3D graph found in Figure 6(j) to T. There are four more 3D graphs using two MIVs as shown in Figure 6(k), (l), (m), and (n). Thus, when the algorithm

finishes, T contains the five 3D graphs found in Figure 6(j), (k), (l), (m), and (n). All of them become 3D POSTs for the given pin locations and 2D POST.

3.2 Congruence of 3D POSTs

The runtime of the algorithm shown in Algorithm 1 is prohibitively long. In addition, there are numerous 3D POSTs in the database, so it is crucial to reduce the runtime and the size of the database. In this section, we show congruent properties of the 3D POSTs, which are used to skip generating and storing some 3D POSTs.

3.2.1 Congruence of Position Sequences. As mentioned in [2], two position sequences are congruent if rotating one of them leads to the other. For example, Figure 7(a) shows position sequence (31542). If we rotate it counterclockwise by 90, 180, and 270 degrees, we obtain position sequences (41523), (42153), and (34152) as shown in Figure 7(b), (c), and (d), respectively. If two position sequences are congruent to each other, a 2D POST constructed for one of them can be for the other position sequence. Thus, we do not need to generate 2D POSTs for some position sequences.

In addition to the rotation, reflection also generates congruent position sequences. If we reflect the pin locations in Figure 7(a) over a y-directional line results in position sequence (35124) shown in Figure 7(e). Now, rotating the position sequence counterclockwise by 90, 180, and 270 degrees leads to position sequences (32514), (24513), and (25143) shown in Figure 7(f), (g), and (h), respectively.

Rotating and reflecting a position sequence has the same effect as generating position sequences by $\Gamma_{(d_1,d_2)}$. For example, generating the position sequence in Figure 7(a) is the same as generating the position sequence $\Gamma_{(y_+,x_+)}$. The position sequences obtained by rotating the position sequence $\Gamma_{(y_+,x_+)}$ by 90, 180, and 270 degrees are the same as the position sequences $\Gamma_{(x_+,y_-)}$, $\Gamma_{(y_-,x_-)}$, and $\Gamma_{(x_-,y_+)}$, respectively. Similarly, reflecting $\Gamma_{(y_+,x_+)}$ over a y-directional line is the same as generating the position sequence $\Gamma_{(y_+,x_-)}$. Then, rotating $\Gamma_{(y_+,x_-)}$ by 90, 180, and 270 degrees are the same obtaining the position sequences $\Gamma_{(x_-,y_-)}$, $\Gamma_{(y_-,x_+)}$, and $\Gamma_{(x_+,y_+)}$, respectively.

If multiple position sequences are congruent, we can store POSTs for only one (called a *base position sequence*) of them. Then, we can obtain POSTs for the other position sequences by properly transforming the POSTs stored for their base position sequences. Notice that a position sequence can be congruent to multiple position sequences as shown in Figure 7, so we need a rule to choose a base position sequence among congruent position sequences. In this paper, we use the following rule to determine base position sequences. Suppose a set of pin locations is given in the 2D plane. Then, we find all the eight position sequences $\Gamma_{(y_\pm,x_\pm)}$ and $\Gamma_{(x_\pm,y_\pm)}$ and choose the smallest position sequence for their base sequence. In Figure 7, for example, (24513) in Figure 7(g) is the smallest number, so (24513) becomes the base position sequence for all the position sequences in Figure 7.

3.2.2 Congruence of 3D POSTs. Suppose pin locations are given in the 3D space. Then, we can characterize the pin locations by two sequences, a position sequence and a tier sequence. The position sequence $PS = (s_1 \ldots s_n)$ is based on the projections of the pins onto the xy plane and the tier sequence $TS = (t_1 \ldots t_n)$ is a sequence of the z-coordinates of the pins where t_i corresponds to s_i. Figure 8 shows an example. In Figure 8(a), the z-coordinates of the pins corresponding to the position sequence elements 3, 1, 5, 4, 2 are 0, 1, 0, 1, 0, respectively. Thus, the tier sequence for the pin locations is $TS = (01010)$.

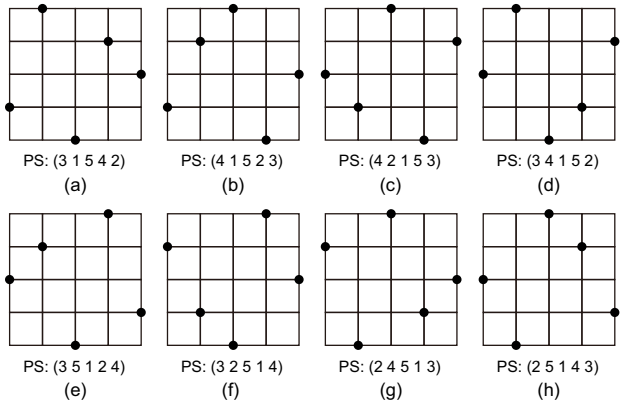

Figure 7: Congruence of eight position sequences.

If we rotate the two tiers in Figure 8(a) counterclockwise by 90, 180, and 270 degrees around the z-axis, we obtain the position and tier sequences shown in Figure 8(b), (c), and (d), respectively. In addition, if we reflect the two tiers in Figure 8(a) over a plane parallel to the yz plane, we obtain the position and tier sequences in Figure 8(e). Rotating the two tiers in Figure 8(e) counterclockwise by 90, 180, and 270 degrees around the z-axis leads to the position and tier sequences in Figure 8(f), (g), and (h), respectively. Moreover, reflecting the two tiers in Figure 8(a) and (e) over a plane parallel to the xy plane generates the position and tier sequences in Figure 8(i) and (m), respectively. Rotating the position and tier sequences in Figure 8(i) and (m) counterclockwise by 90, 180, and 270 degrees around the z-axis generates the position and tier sequences in Figure 8(j), (k), and (l), and Figure 8(n), (o), and (p), respectively.

To find a congruence between two sets of position and tier sequences, we define a *3D position sequence* $\Gamma_{(R,C,T)}$, which consists of a pair of sequences. The first sequence is the 2D position sequence $(s_1 \ldots s_n)$ obtained from $\Gamma_{(R,C)}$. The second sequence is the tier sequence along the T-direction ($T \in \{z_\pm\}$) as defined above. Then, the 3D position sequence for the pins in Figure 8(a) is denoted by $\Gamma_{(y_+,x_+,z_+)}$. Similarly, 3D position sequences for the pins in Figure 8(b), (c), (d), (e), (f), (g), and (h), are $\Gamma_{(x_+,y_-,z_+)}$, $\Gamma_{(y_-,x_-,z_+)}$, $\Gamma_{(x_-,y_+,z_+)}$, $\Gamma_{(y_+,x_-,z_+)}$, $\Gamma_{(x_-,y_-,z_+)}$, $\Gamma_{(y_-,x_+,z_+)}$, and $\Gamma_{(x_+,y_+,z_+)}$, respectively. Since the reflection over a plane parallel to the xy plane reverses the tier sequence, 3D position sequences for the pins in Figure 8(i), (j), (k), (l), (m), (n), (o), and (p) are $\Gamma_{(y_+,x_+,z_-)}$, $\Gamma_{(x_+,y_-,z_-)}$, $\Gamma_{(y_-,x_-,z_-)}$, $\Gamma_{(x_-,y_+,z_-)}$, $\Gamma_{(y_+,x_-,z_-)}$, $\Gamma_{(x_-,y_-,z_-)}$, $\Gamma_{(y_-,x_+,z_-)}$, and $\Gamma_{(x_+,y_+,z_-)}$, respectively. If two sets of pin locations are congruent, we can use the 3D POSTs belonging to one of them for the other by properly transforming the 3D POSTs.

We also define a *3D base position sequence* as follows. Suppose a set of pin locations is given in the 3D space. Then, we find all the 16 3D position sequences $\Gamma_{(y_\pm,x_\pm,z_\pm)}$ and $\Gamma_{(x_\pm,y_\pm,z_\pm)}$ for them and choose the smallest 3D position sequence. If multiple 3D position sequences have the same 2D position sequence, we choose

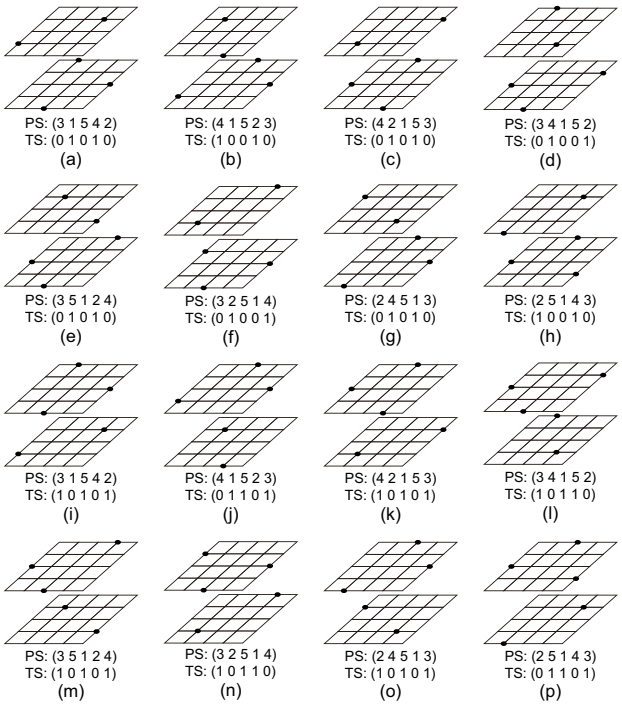

Figure 8: Congruence of 16 position and tier sequences.

the smallest tier sequence among them. In Figure 8, for example, the smallest 2D position sequence is (24513) in Figure 8(g) and (o). Between these two, the tier sequence (01010) in Figure 8(g) is smaller than (10101) in Figure 8(o), so the 3D position sequence of Figure 8(g) becomes the base 3D position sequence for all the 3D position sequences in Figure 8.

4 SIMULATION RESULTS

In this section, we present simulation results obtained from the construction of all 3D POSTs on the 3D Hanan grid. We implemented the proposed algorithm using C/C++ and ran the code in an Intel Core i5-6600K 3.3GHz CPU system with 64GB memory. We used the 2D POST database in [7].

Table 1 shows statistics of the construction of all 3D POSTs for two- to six-pin nets and two to four tiers. The number of position sequences is $n!$ where n is the number of pins. The number of 2D POSTs comes from the database of all 2D POSTs in [7].

Our first observation is that as the tier count goes up from two to four, the total number of 3D POSTs increases exponentially. This is because the number of combinations of placing pins in different tiers increases exponentially as the tier count goes up. The recurrent relation for counting the number of the combinations is as follows:

$$f(n,t) = t^n - \sum_{i=1}^{t-1}\{(t-i+1)\cdot f(n,i)\} \tag{4}$$

where $f(n,t)$ is the number of combinations of placing n pins in t consecutive tiers. A closed-form expression for $f(n,t)$ is as follows:

$$f(n,t) = t^n - 2\cdot(t-1)^n + (t-2)^n, \tag{5}$$

$$f(n,1) = 1. \tag{6}$$

Thus, as t increases, $f(n,t)$ goes up exponentially. In addition, as the pin count goes up, the number of 2D POSTs also increases exponentially as shown in the table. Thus, the total number of 3D POSTs increases extremely fast as the pin and tier counts go up.

We also observe that the number of generated 3D POSTs is approximately 16% of the total 3D POSTs. As explained in Section 3.2, using the congruence properties of position sequences and 3D POSTs significantly reduces the number of 3D POSTs actually generated from the proposed algorithm. Thus, we reduce the construction time and the database size effectively. The database contains the generated 3D POSTs for all the 3D base position sequences. For non-base topologies, the database contains their base topologies and rules transforming the non-base topologies to their base topologies.

The construction efficiency measured by the ratio between the number of generated 3D POSTs and the total construction time decreases almost exponentially as the pin count and the tier count go up. The proposed algorithm can still construct approximately 130,000 3D POSTs for the six-pin four-tier case. However, there are almost 15-billion 3D POSTs to generate for the case, so the construction time is about 30 hours. The table size is approximately 135 GB, which can be easily handled in server computers.

Figure 9 shows two 3D POSTs constructed for pin locations $(0,0,0)$, $(3,1,3)$, $(2,2,2)$, $(5,3,1)$, $(1,4,0)$, $(4,5,3)$ and the same 2D POST. The red edges are planar wires and the blue edges are MIVs. The 3D POST in Figure 9(a) has five planar edges in Tier 0 (the bottommost tier) and eight planar edges in Tier 1, respectively. On the other hand, the 3D POST in Figure 9(b) has 12 planar edges in Tier 2 and a planar edge in Tier 3 (the topmost tier). In addition, the planar coordinates of the MIVs in Tier 1 in Figure 9(a) are $(2,1)$ and $(2,3)$, whereas those in Tier 1 in Figure 9(b) are $(0,0)$ and $(1,4)$. Similarly, the planar coordinates of the MIVs in Tier 2 in Figure 9(a) are $(3,1)$, $(2,2)$, and $(4,5)$, whereas those in Figure 9(b) are $(0,0)$, $(1,4)$, and $(5,3)$. The planar coordinates of the MIVs in Tier 3 in Figure 9(a) are $(3,1)$ and $(4,5)$, whereas those in Figure 9(b) are $(2,1)$ and $(4,5)$. Thus, among the seven MIVs inserted in the two 3D POSTs, only one MIV located at $(4,5,3)$ is common and the other six MIVs are located at different locations. We have also compared various 3D POSTs and found similar trends. Overall, we expect that the database of the 3D POSTs can be used for 3D routing to evenly distribute planar wires and MIVs across the tiers in a given layout.

5 CONCLUSION

Monolithic 3D integration uses ultra-small MIVs, so routing of 3D nets in the design of monolithic 3D IC layouts should minimize the planar wirelength and the number of MIVs at the same time while evenly distributing planar wires across tiers. In this paper, therefore, we have developed an algorithm to build a database of 3D POSTs, which can help routers generate MMRSMTs in no time and use them to route monolithic 3D ICs and optimize 2D and 3D interconnects. The database size is manageable for up to four-tier

Table 1: Statistics of the construction of all 3D POSTs for two- to six-pin nets for two, three, and four tiers. "# PS" is the number of 2D position sequences for projected pins. "# all 3D POSTs" is the total number of 3D POSTs and "# gen. 3D POSTs" is the number of 3D POSTs generated from the proposed algorithm. r is the ratio between the number of generated 3D POSTs and the total number of 3D POSTs. "Con. time" is the construction time for all the generated 3D POSTs. "Con. eff." is the construction efficiency, which is measured by the ratio between the number of generated 3D POSTs and the construction time in seconds.

# pins (n)	# PS ($n!$)	# 2D POSTs	# tiers	# all 3D POSTs	# gen. 3D POSTs	r	Con. time	Con. eff.	Table size
2	2	4	2	24	12	0.5	0.0 s	-	< 1 KB
			3	48	24	0.5	0.0001 s	-	< 1 KB
			4	80	40	0.5	0.0001 s	-	< 1 KB
3	6	16	2	224	84	0.375	0.0001 s	-	1 KB
			3	896	336	0.375	0.0003 s	-	2 KB
			4	2,352	888	0.378	0.0006 s	-	4 KB
4	24	284	2	20,056	5,372	0.268	0.0043 s	1,249,302	35 KB
			3	226,800	60,120	0.265	0.0457 s	1,315,536	313 KB
			4	1,396,944	367,424	0.263	0.3465 s	1,060,387	2 MB
5	120	4,260	2	719,864	125,360	0.174	0.1484 s	844,744	850 KB
			3	14,876,928	2,575,092	0.173	5.2478 s	490,699	16 MB
			4	142,195,680	24,482,354	0.172	95.77 s	255,637	167 MB
6	720	120,212	2	85,530,040	13,831,206	0.162	20.13 s	687,094	93 MB
			3	4,318,826,472	697,355,262	0.161	42.2 m	275,417	5.1 GB
			4	90,473,628,112	14,586,090,890	0.161	30.2 h	134,162	129 GB

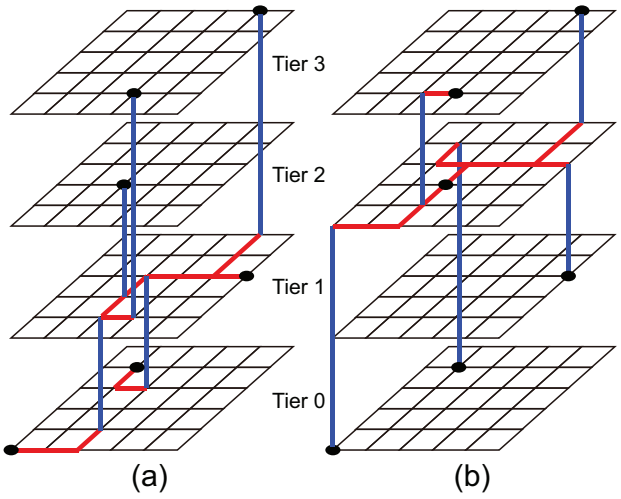

Figure 9: Comparison of two 3D POSTs constructed for pin locations $(0,0,0)$, $(3,1,3)$, $(2,2,2)$, $(5,3,1)$, $(1,4,0)$, $(4,5,3)$**. 3D position sequence:** $PS = (143625)$, $TS = (032103)$**.**

six-pin 3D POSTs. Thus, the proposed algorithm and the database of 3D POSTs will help various VLSI CAD algorithms optimize 3D IC layouts more effectively and serve as a baseline algorithm for better monolithic 3D IC routing algorithms.

ACKNOWLEDGMENTS

This work was supported by the Defense Advanced Research Projects Agency Young Faculty Award under Grant D16AP00119 and the New Faculty Seed Grant (125679-002) funded by the Washington State University.

REFERENCES

[1] Yiting Chen and Dae Hyun Kim. 2017. A Legalization Algorithm for Multi-Tier Gate-Level Monolithic Three-Dimensional Integrated Circuits. In *Proc. Int. Symp. on Quality Electronic Design*. 277–282.
[2] Chris Chu and Yiu-Chung Wong. 2008. FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 27. 70–83.
[3] Dae Hyun Kim, Krit Athikulwongse, and Sung Kyu Lim. 2013. Study of Through-Silicon-Via Impact on the 3D Stacked IC Layout. In *IEEE Trans. on VLSI Systems*, Vol. 21. 862–874.
[4] Dae Hyun Kim, Rasit Onur Topaloglu, and Sung Kyu Lim. 2012. Block-level 3-D IC Design with Through-Silicon-Via Planning. In *Proc. Asia and South Pacific Design Automation Conf.* 335–340.
[5] Bon Woong Ku, Kyungwook Chang, and Sung Kyu Lim. 2018. Compact-2D: A Physical Design Methodology to Build Commercial-Quality Face-to-Face-Bonded 3D ICs. In *Proc. Int. Symp. on Physical Design*. 90–97.
[6] Chung-Wei Lin, Shih-Lun Huang, Kai-Chi Hsu, Meng-Xiang Lee, and Yao-Wen Chang. 2008. Multilayer Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Spanning Graphs. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 27. 2007–2016.
[7] Sheng-En David Lin and Dae Hyun Kim. 2018. Construction of All Rectilinear Steiner Minimum Trees on the Hanan Grid. In *Proc. Int. Symp. on Physical Design*. 18–25.
[8] Chih-Hung Liu, Chun-Xun Lin, I-Che Chen, D. T. Lee, and Ting-Chi Wang. 2014. Efficient Multilayer Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Geometric Reduction. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 33. 1928–1941.
[9] Shreepad Panth, Kambiz Samadi, Yang Du, and Sung Kyu Lim. 2014. Design and CAD Methodologies for Low Power Gate-level Monolithic 3D ICs. In *Proc. Int. Symp. on Low Power Electronics and Design*. 171–176.
[10] Shreepad Panth, Kambiz Samadi, Yang Du, and Sung Kyu Lim. 2015. Placement-Driven Partitioning for Congestion Mitigation in Monolithic 3D IC Designs. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 34. 540–553.
[11] Shreepad Panth, Kambiz Samadi, Yang Du, and Sung Kyu Lim. 2017. Shrunk-2D: A Physical Design Methodology to Build Commercial-Quality Monolithic 3D ICs. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 36. 1716–1724.