Journal of Engineering and Technology for Industrial Applications

ITEGAM-JETIA

Manaus, v.12 n.57, p. 661-670. January/February, 2026.
DOI: https://doi.org/10.5935/jetia. v12i57.2957

ISSN ONLINE: 2447-0228

RESEARCH ARTICLE OPEN ACCESS

TO3: A COMPACT AND EFFICIENT ENCODING SCHEME FOR SKEWED SLICING FLOORPLANS IN VLSI DESIGN

Biswojit Nayak*[1] and Mrutynjaya Panda[2]

[1, 2]Department of Computer Science and Applications, Utkal University, Vani Vihar, Bhubaneswar- 758044, Odisha, India.

[1]https://orcid.org/0000-0002-7912-6065, [2]https://orcid.org/0000-0001-5713-9220

Email: *biswojit22@gmail.com, mrutyunjaya.cs@utkaluniversity.ac.in

ARTICLE INFO

Article History
Received: November 16, 2025
Revised: December 20, 2025
Accepted: January 15, 2026
Published: February 28, 2026

Keywords:
VLSI,
Slicing Floorplan,
Slicing Tree,
TO3 Encoding.

ABSTRACT

A slicing floorplan is a recursive process of drawing a floorplan horizontally and vertically until all the blocks are accommodate into it. In the nanometer era, the escalating transistor density in VLSI chips underscores the necessity of designing compact floorplan representations for VLSI circuits. This work introduces a novel approach, employing a tree of order three (TO3) for the effective coding of a skewed slicing tree that corresponds to a slicing floorplan. The key innovation lies in the utilization of a TO3 structure, which adds a distinctive dimension to the encoding scheme. The proposed encoding method for a skewed slicing tree achieves a notable reduction in code size compared to previous approaches. Experimental findings from the MCNC benchmark and artificial circuits indicate that the proposed code uses, on average, almost 80% fewer bits than Breadth First code (BFS), 46% fewer than Slicing Pair (SP) code, and 49% fewer than Improved Slicing Pair (ISP) code for encoding a slicing tree, while also reducing CPU running time through a more compact representation of the electronics modules.

I. INTRODUCTION

Floorplan design is one of the most significant phases in VLSI physical design, which involves arranging a collection of rectangular circuit modules on a chip to optimize the overall area and the length of interconnection wires. This arrangement defines the layout of a VLSI circuit, which is a well-known NP-hard problem [1] that has attracted a lot of interest in recent years. Floorplans are generally divided into two categories: slicing floorplans and non-slicing floorplans. A slicing floorplan is created by recursively drawing horizontal and vertical lines from a given rectangle as shown in Figure 1 (a). In contrast, a non-slicing floorplan cannot be recursively divided into two halves to obtain all of the blocks of the given rectangle. Non-slicing floorplans are sometimes known as spiral or wheel floorplans, as shown in Figure 1 (b).

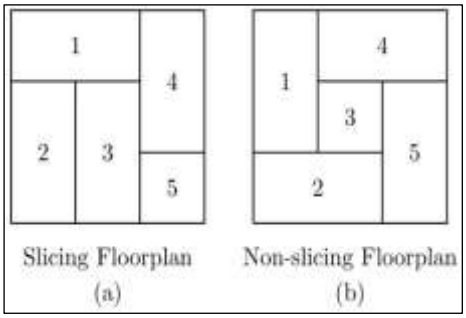


Figure 1: Slicing and non-slicing floorplans.
Source: Authors, (2026).

A slicing floorplan is commonly represented using a binary tree known as a slicing tree. In this representation, each internal node is labelled with '+' or '*' based on line drawing types (for horizontal lines, '+' and for vertical lines, '*'), and leaf nodes represent electronics blocks.

However, slicing trees are not unique representations of the floorplan since the same layout can often be represented by different slicing trees depending on how the lines are chosen. To address this issue, [2] introduced the concept of a skewed slicing tree in 1986, in which the parent and the right child are always opposite line drawing types, as illustrated in Fig 2. However, the non-slicing floorplan is a generalized floorplan such as the one shown in Fig. 1 (b), can be represented in several alternative models, including Sequence Pair (SP), O-Tree, Transitive Closure Graph (TCG), Corner Block List (CBL), SKB-tree, and B*-tree [3-7].

Later, in 2001, [8] proved that a slicing tree can serve as a complete representation for general floorplans. They also showed that, when combined with a simple compaction process, slicing tree representations are capable of producing maximally compact module placements. The proposed work focuses on slicing floorplans because of their hierarchical structure and suitability for machine learning-based generation, as demonstrated in recent work by [9]. Building on this idea, the slicing tree is presented using a binary encoding scheme, enabling faster computation and reduced storage compared to symbolic tree traversal used in [9].

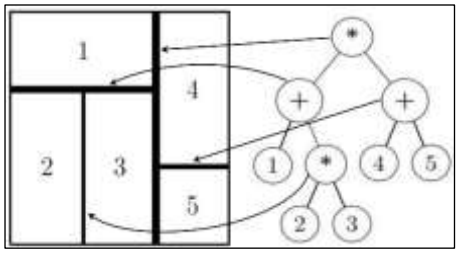


Figure 2: Slicing floorplan and its corresponding skewed slicing tree.
Source: Authors, (2026).

Over the years, several binary representations of floorplans have been proposed in the literature [10-17]. In 2006, [10] proposed a $(5n-5)$ bits representation of a floorplan, where n denotes the number of inner rectangles. Later [11] introduced a more compact $(4n-4)$-bits representation of floorplan and also showed that the number of different rectangles satisfies $R(n) <= 13.5(n-1)$. In 2012, [12] proposed two compact encodings of size $6f-2n^4+6$ bits and $5f-5$ bits for floorplans, where n represents the number of edges, f is the number of inner faces, and n^4 is the number of vertices of degree four. The optimum encoding is dependent on the number of vertices of degree four. The first encoding is more efficient when $2n^4 > f+11$, while the second performs better otherwise. In a subsequent work, [13] proposed an even more compact encoding of size $5f-n^4-6$ bits for $f >= 2$ of general rectangular drawings, which is independent of n^4.

To represent the mosaic (dead space free) floorplan, [14] introduced a string-based data structure known as the quarter-state sequence or Q sequence. The number of bits needed to represent mosaic floorplans with n blocks using the Quarter-State Sequence approach is 4n bits. Subsequently, He [15] proposed more compact binary encoding of length $(3n-3)$ bits and also proved that any binary string representation of an n block mosaic floorplan must use at least $(3n-o(n))$ bits. In 2014, [16] introduced two encoding schemes for slicing floorplans: the breadth-first code (BFS code) and the slicing-pair code (SP code). For a slicing floorplan with n blocks, the BFS code requires $3n-2$ bits. Building on this idea, the authors introduced the SP code by improving the BFS code.

The length of the SP code is $5n/2+m_1-p_{10}-4$ bits, Where n denotes the number of blocks, m_1 represents the number of internal nodes in a slicing tree that have exactly one child as a leaf node (i.e., a block), and p_{10} counts the number of slicing pairs (siblings) in which the left node is a leaf and the right node is an internal node. More recently, [17] proposed an improved slicing-pair (ISP) code designed for a balanced skewed slicing floorplan. They also proved that the ISP code has a length of $2(n+m_1)-3$ bits. Meanwhile, [9] proposed a subitizing-inspired approach to VLSI floorplanning that fine-tunes large language models (LLMs) to generate slicing trees representing near-optimal floorplans. Their method encodes the layout as post-order traversals of slicing trees and achieves high success and low dead-space ratios, especially when using GPT-4o-mini.

Unlike their post-order symbolic representation, the present work introduces a binary encoding of slicing floorplans, providing a more compact and machine-interpretable form that facilitates efficient optimization and learning. The motivation for creating compact representations of VLSI floorplans stems from the necessity of being able to quickly optimize very large and complex designs using minimal computational resources. Compact representations also require less memory to store and optimize the floorplan, which is particularly advantageous for large scale designs. Thus, even complex designs can be optimized with minimal resources as well as reduced execution times due to quicker evaluation of the design space during optimization processes [18]. The perturbation technique allows for an efficient examination of the design space without having to recreate the entire floorplan, thus speeding up the optimization process [14], [18].

In addition, compact encodings lead to faster runtime of both the floorplan generation and verification process, which are especially important in large industrial designs with a tight schedule for market entry [19]. Although hybrid placement methods such as [20] have shown promising results, they still require significant computational effort for their search processes. The reinforcement learning-based optimization technique proposed by [21] reduces the burden of explorations by learning efficient perturbations of sequence-pair floorplans but still depends on the compactness and flexibility of the underlying binary encoding. The use of a more compact encoding could help lower the complexity of that search by reducing the number of bits required to represent the solution space. Building on this motivation, the current work improves existing solutions such as SP code [16] and ISP code [17].

The present contribution introduces a new approach to encode skewed slicing floorplans referred to as the Tree of Order Three (TO3) encoding. To summarize, this paper makes the following key contributions.

- A tree of order three (TO3) code is introduced to encode skewed slicing floorplans.

- On average, the length of TO3 code reduces the length by 80%, 46%, and 49% compared to BFS, SP, and ISP codes, respectively.
- The range of the TO3 code length (i.e., lower and upper bounds) is determined based on [16] and [17], and its performance is compared with the lengths of the SP and ISP codes.
- Interestingly, the upper bound of the TO3 code length matches the lower bound of the SP and ISP codes, thus proving the TO3 code is more compact than the SP and ISP codes.

The remainder of this paper is organized as follows. Section II reviews the existing BFS, SP, and ISP encoding schemes. Section III introduces the proposed TO3 code for skewed slicing floorplans. The implementation details are discussed in Section IV, and Section V concludes with a summary of findings and direction for future work.

II. PRELIMINARIES

This section presents essential definitions and important finding from state-of-the-art research, which collectively guide the development of the TO3 code discussed in Section III.

II.1 DEFINITIONS

II.1.1 Slicing Floorplan

The floorplan consisting of exactly one block is referred to as a slicing floorplan. A slicing floorplan F with n blocks can be constructed by drawing horizontal and vertical cuts recursively on each slicing floorplan that partitions floorplan into two smaller slicing floorplans, F_1 and F_2. This partition process continues until all the blocks fit into the floorplan. In case of a vertical cut, the right-most boundary of F_1 is equivalent to the left-most boundary of F_2, while a horizontal cut aligns the top-most horizontal boundary of F_1 is equivalent to the bottom-most boundary of F_2.

II.1.2 Slicing Tree

A slicing floorplan F consisting of n blocks is represented by a binary tree known as a slicing tree T with $2n-1$ vertices or nodes. In this tree, every internal node represents either '*' (for vertical cut) or '+' (for horizontal cut), and the leaf node correspond to the electronics blocks. A slicing tree is called skewed if an internal node and its right child are of opposite cuts, as shown in Fig. 2; otherwise, the tree is non-skewed.

II.1.3 Slicing Pair

A slicing pair (V_i, V_{i+1}) for each $i = 1,3,\ldots,2n-3$, consists of two vertices in the slicing tree T (with n blocks) that have the same parent P, where $(V_i, V_{i+1}) \in \{*, +, L\}$, $P \in \{*, +\}$, and L represents a leaf node, as illustrated in Figure 3.

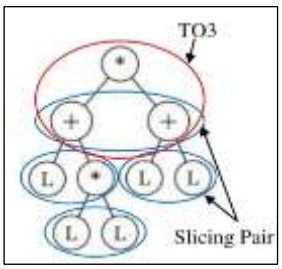


Figure 3: Slicing pairs and TO3 representation of a skewed slicing tree T.
Source: Authors, (2026).

II.2 PRIOR WORKS

Theorem 1. For a slicing floorplan F with n blocks, the BFS representation, i.e., $|B(F)|$, requires $3n-2$ bits. In addition the floorplan can be encoded and decoded in linear time, that is, $O(n)$ [16].

Theorem 2. Consider a slicing floorplan F with n blocks and its associated skewed slicing tree T. If |SP(F)| represents the number of bits required to encode F using the slicing pair code, then the encoding length is $|SP(F)| = 5n/2 + m_1/2 - p_{10} - 4$ bits, Where m_1 and p_{10} are same parameters introduced in Section I. Moreover, both encoding and decoding using slicing pair code runs in $O(n)$ time [16].

Theorem 3. The length of $SP(F)$ is bounded by the inequality $2n-3 \leq |SP(F)| \leq 3n-5$. When the constant terms in these expressions are ignored, the worst case of SP and BFS codes are almost the same [16].

Theorem 4. Consider a slicing floorplan F with n blocks and its corresponding skewed slicing tree is T. Let $|ISP(F)|$ represent the length of the encoded bits string of T using the improved slicing pair code, i.e., $|ISP(F)| = 2(n + m_1) - 3$, where m_1 is the number of internal vertices that have exactly one leaf node [17].

Theorem 5. For a floorplan F with n blocks, if the corresponding skewed slicing tree is balanced, then $|ISP(F)| < |SP(F)| < |B(F)|$ holds[17].

III. PROPOSED TO3 CODE FOR SKEWED SLICING FLOORPLAN

This section presents a novel encoding technique specially designed for skewed slicing floorplans. It explains the underlying principle of encoding. In addition, this section presents detailed derivation of the code length and the range length of the proposed code.

III.1 ENCODING

Consider a slicing floorplan F with n blocks and let T(F) be its associated skewed slicing tree. A tree of order three, in short TO3 (i.e., a subtree with three vertices) is obtained form T(F), which is represented as a triplet (x, y, z). In this triplet, $x \in \{+,*\}$ represents the root node, while $y, z \in \{+,*, L\}$, denotes the left and right children of x respectively. The rules to obtain TO3s of a skewed slicing tree are outlined below.

1 Internal vertices at even levels of the slicing tree, starting from the root (considering the root at level 0) are candidates for roots of TO3s.
2 If the root of a TO3 is at level i, then two leaves of the TO3 are at level $(i + 1)$, and they can't be roots of any other TO3s, i.e., TO3s are disjoint trees.
3 A leaf labelled by L can not be the root of a TO3.

Using the above rules and the property of a skewed slicing tree (an internal node always has a label opposite to that of its right child), the possible TO3s for $x = '+'$ of the type $(+, y, z)$ are: $(+, +,*)$, $(+, +, L)$, $(+,*,*)$, $(+,*, L)$, $(+, L,*)$, $(+, L, L)$. Similarly, TO3s of the type $(*, y, z)$ are: $(*, +, +)$, $(*, +, L)$, $(*,*, +)$, $(*,*, L)$, $(*, L, +)$, $(*, L, L)$. In the skewed slicing tree T shown in Figure 3, the corresponding TO3 is inscribed within a circle. The TO3 code, analogous to the SP and ISP codes, is shown in Table 1. Column 1 of Table 1 represents the TO3 labels, and the encoded bit pattern used for each TO3 label is shown in Column 2. Although L is not a TO3 label, it is included in the last row of the table and assigned a bit value of 1, as the label L is required to encode when it is followed by TO3 labels in the TO3's level-wise traversal. The complete encoding procedure for the TO3 code is described in Algorithm 1.

Table 1: Patterns of TO3 and their assigned codes.

TO3 Labels	Assigned Bits
(+, +, *)	011
(+, +, L)	010
(+, *, *)	001
(+, *, L)	000
(+, L, *)	01
(+, L, L)	00
(*, +, +)	111
(*, +, L)	110
(*, *, +)	101
(*, *, L)	100
(*, L, +)	11
(*, L, L)	10
L	1

Source: Authors, (2026).

Algorithm 1 takes the root of the skewed slicing tree as its input and generates the corresponding encoded bit string. The algorithm employs a queue data structure Q to traverse the skewed slicing tree level-wise, during which all the TO3s are identified and encoded using the bit pattern listed in Table 1. Since Algorithm 1 visited every node of the slicing tree exactly once, and a slicing tree with n leaves contains $n - 1$ internal nodes, the overall time complexity of the algorithm is $O(n)$. By applying bit patterns from Table 1 and following Algorithm 1, the TO3 code length for the skewed slicing tree shown in Fig. 3 is found to be 6 bits. In comparison, the lengths of the BFS [16], SP [16], and ISP [17] codes for the same tree are 13 bits, 8 bits, and 9 bits, respectively. Therefore, among the BFS, SP, ISP, and TO3 codes, the proposed TO3 code required the fewest number of bits and provide the most compact.

Algorithm 1: Encoding.

Input: Root of the skewed slicing tree
Output: Encoded String (EncodeStr)
1: Initialize EncodeStr ← Null
2: Insert root of the skewed slicing tree into a queue Q (Enqueue(Q, root))
3: while Q ≠ Ø do
4: if Front → ch == L then
5: Append encoded TO3 label (Front → ch) using Table 1 to EncodeStr
6: else
7: Append encoded TO3 label (Front → ch, Front → Left → ch, Front → Right → ch) using Table 1 to EncodeStr
8: end if
9: if Front → Left → Left ≠ NULL then
10: Enqueue(Q, Front → Left → Left)
11: end if

```
12:   if Front → Left → Right ≠ NULL then
13:      Enqueue(Q, Front → Left → Right)
14:   end if
15:   if Front → Right → Left ≠ NULL then
16:      Enqueue(Q, Front → Right → Left)
17:   end if
18:   if Front → Right → Right ≠ NULL then
19:      Enqueue(Q, Front → Right → Right)
20:   end if
21:   Dequeue(Q)
22: end while
23: return EncodeStr
```

Source: Authors, (2026).

III.2 DECODING

The decoding algorithm described in Algorithm 2 takes an encoded binary string as input and reconstructs the corresponding skewed slicing tree. The bit(s) assigned (in Table 1) for the TO3 label either start with 0 (for the parent of the TO3 label is '+') or start with 1 (for the parent of the TO3 label is '*'). An encoded bit string starting with '0' has an assigned length of either three or two bits, whereas a bit string starting with '1' may have a length of three, two, or one bit. Algorithm 2 is a recursive algorithm that makes decisions non-deterministically based on the current symbol pointed to by the forward pointer (FP). During execution the algorithm generates TO3 labels level-wise and stores them in a queue data structure Q. Since Algorithm 2 non-deterministically generates TO3 labels, there is the possibility of more than one slicing tree, which is held by an array named ArraySkewed. Algorithm 2 then systematically builds and validates all slicing trees using skewed slicing properties, ultimately selecting a valid skewed slicing tree as the output. In the worst case, the slicing tree has n leaf nodes; the running time complexity of Algorithm 2 is $O(3^{(2n-1)})$, since the algorithm explore non-deterministically on bits directed by the forward pointer (FP).

Algorithm 2: Decoding.

```
Input: Encoded String (EncodeStr)
Output: Skewed Slicing Tree
1: Initialize forward pointer FP to the first character of the input encoded string
2: Initialize a queue Q to hold level-wise TO3 labels
3: Create an array named ArraySkewed to store the skewed slicing tree
4: Call AlgoDecode(EncodeStr, FP, Q, ArraySkewed)
5: if FP ≠ Null then
6:    if FP = 1 then
7:       TempStr = FP
8:       Decode TempStr to the TO3 label using Table 1
9:       Append TO3 label to Q
10:   AlgoDecode(EncodeStr, FP + 1, Q, ArraySkewed)
11:   end if
12:   if (FP = 1 ∨ FP = 0) ∧ (FP + 1 ≠ Null) then
13:      Append two bits from FP to TempStr
14:      Decode TempStr to the TO3 label using Table 1
15:      Append TO3 label to Q
16:      AlgoDecode(EncodeStr, FP + 2, Q, ArraySkewed)
17:   end if
18:   if (FP = 1 ∨ FP = 0) ∧ (FP + 2 ≠ Null) then
19:      Append three bits from FP to TempStr
20:      Decode TempStr to the TO3 label using Table 1
21:      Append TO3 label to Q
22:      AlgoDecode(EncodeStr, FP + 3, Q, ArraySkewed)
23:   end if
24: else
25:   Append Q to ArraySkewed
26:   return ArraySkewed
27: end if
28: Generate level-wise slicing trees on the array of queues and check whether
the tree is skewed or not
29: return Skewed Slicing Tree
```

Source: Authors, (2026).

III.3 ESTIMATING LENGTH OF TO3 CODE

This subsection derives an expression for the length of the TO3 code. The analysis follows the same approach used to determine the length of the SP code in [16]. To support this derivation, the following additional notations are considered for the skewed slicing tree T(F), as illustrated in Table 2.

Table 2: Notations and Definitions for TO3 Encoding.

Notation	Definition
|TO3(F)|	Length of TO3 code of T(F) in bits for a floorplan F.
t_{000}	Number of TO3s (x, y, z) such that x, y, z ∈ {+, *}.
t_{001}	Number of TO3s (x, y, z) such that x, y ∈ {+, *}, z ∈ {L}.
t_{010}	Number of TO3s (x, y, z) such that x, z ∈ {+, *}, y ∈ {L}.
t_{011}	Number of TO3s (x, y, z) such that x ∈ {+, *}, y, z ∈ {L}.
m	Number of internal nodes of T(F).
p	Number of leaf nodes (labeled by L) of T(F) that are not part of any TO3.

Source: Authors, (2026).

For the skewed slicing tree shown in Fig. 3, $p = 3$, there are three such Ls at level 2 of T. The expression for the bit length of the TO3 code corresponding to the slicing tree T(F) is given in Theorem 6.

Theorem 6. Let T(F) be a skewed slicing tree for slicing floorplan F with n blocks. Then the length of the TO3 code for encoding T(F) is $2n - t_{010} - t_{011} - 3$ bits. i.e., $|TO3(F)| = 2n - t_{010} - t_{011} - 3$.

Proof. Since the slicing tree T(F) is a complete binary tree with n blocks, it contains $m = n - 1$ internal nodes. To derive an expression for m, observe that each TO3 of the type t_{000} has three internal nodes, TO3s of the types t_{001} and t_{010} have two internal nodes, and each TO3 of the type t_{011} has one internal node and two leaf nodes. Additionally, the expression for the number of leaf nodes p that are not part of the TO3 labels is given in Eq. 2. Thus, it follows that

$$m = n - 1 = 3t_{000} + 2t_{001} + 2t_{010} + t_{011} \quad (1)$$

$$p = n - t_{001} - t_{010} - 2t_{011} \quad (2)$$

To establish an expression for |TO3(F)|, referencing Table 1 reveals that TO3s of types t_{000} and t_{001} require 3 bits each, while types t_{010} and t_{011} utilize 2 bits each. Additionally, the leaf L, which is not part of any TO3, is represented with just one bit. In summary, this information contributes to the derivation of the total bit count for |TO3(F)| as present in Eq. 3.

$$\begin{aligned} |TO3(F)| &= 3t_{000} + 3t_{001} + 2t_{010} + 2t_{011} \\ &+(n - t_{001} - t_{010} - 2t_{011} - 2) \end{aligned} \quad (3)$$

Recall that in the calculation of |SP(F)| (See Theorem 2, Section II) the authors of [16] saved the last two bits for slicing pair (L, L) subtracting 2 from the expression of |SP(F)|. Similarly, in order to have fair comparison between |SP(F)| and |TO3(F)|, last two bits for slicing pair (L, L) is also saved by subtracting 2 from the right hand side of Eq. 3. By substituting Eq. 2 into Eq. 3, the resulting expression is derived.

$$\begin{aligned} |TO3(F)| &= n + 3t_{000} + 2t_{001} + t_{010} - 2 \\ &= n + 3t_{000} + 2t_{001} + 2t_{010} - t_{010} \\ &\qquad + t_{011} - t_{011} - 2 \\ &= n + (3t_{000} + 2t_{001} + 2t_{010} + t_{011}) \\ &\qquad - t_{010} - t_{011} - 2 \\ &= n + n - 1 - t_{010} - t_{011} - 2 \\ &= 2n - t_{010} - t_{011} - 3 \end{aligned} \quad (4)$$

Equation 4 estimates the bit-length of the TO3 code of T(F). Hence the proof. The lengths of the BFS, SP, ISP, and TO3 codes are computed for the skewed slicing tree in Fig. 3. For the parameter values $n = 5, m_1 = 1, p_{10} = 1, t_{010} = 0, t_{011} = 1$, Theorem 1, 2, 4 and 7 yields $|B(F)| = 3 * 5 - 2 - 4 = 13$ bits, $|SP(F)| = 12.5 + 0.5 - 1 - 4 = 8$ bits, $|ISP(F)| = 2 * (5 + 1) - 3 = 9$ bits and $|TO3(F)| = 10 - 0 - 1 - 3 = 6$ bits respectively. Hence, the proposed TO3 code encodes T(F) using a smaller number of bits compared to the other codes.

III.4 RANGE LENGTH OF TO3 CODE

The authors of [16] showed that, in the worst case, the length of SP code is almost the same as that of the BFS code, indicating that SP offers a more compact representation (see Theorem 3, Section II). Further, [17] proved that, for balanced skewed slicing trees, the ISP code achieves better compression than the SP code (see Theorem 5, Section II). Extending these results, the following section demonstrates that the worst-case length of the TO3 code is shorter than the best-case length of the ISP and SP codes. In summary, the following Theorem 7 is established.

Theorem 7. If T(F) be the skewed slicing tree of a floorplan F with n blocks. Then $\frac{3n-5}{2} \leq |TO3(F)| \leq 2n - 3$.

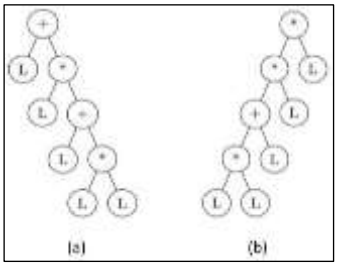


Figure 4: Minimum and Maximum saved skewed slicing trees.
Source: Authors, (2026).

Proof. The best and worst case length of TO3 code are evaluated using the skewed slicing trees shown in Fig. 4 (a) and Fig. 4 (b). The best case of $|TO3(F)|$ occurs when the slicing tree skews along the right branch of the tree shown in Fig. 4 (a). In this case: for n: odd, $t_{010} = \frac{n-1}{2}$, $t_{011} = 0$, and for n: even, $t_{010} = \frac{n-2}{2}$, $t_{011} = 1$. Using these values in $|TO3(F)| = 2n - t_{010} - t_{011} - 3$, the following result is obtained.

$$|TO3(F)| = \frac{3n-5}{2} \quad \text{if } n : \text{odd}$$
$$= \frac{3n-6}{2} \quad \text{if } n : \text{even}$$

Therefore, the tight lower bound of $|TO3(F)|$ is $\frac{3n-5}{2}$. i.e.

$$\frac{3n-5}{2} \leq |TO3(F)| \tag{5}$$

Similarly, for the worst case of $|TO3(F)$ occurs when the slicing tree skews along the right branch of the tree shown in Fig. 4 (b). In such case: for n: odd, $t_{010} = 0$, $t_{011} = 0$, and for n: even, $t_{010} = 0$, $t_{011} = 1$. Using the values of t_{010} and t_{011} in $|TO3(F)| = 2n - t_{010} - t_{011} - 3$, it follows that

$$|TO3(F)| = 2n - 3 \quad if\ n : odd$$
$$= 2n - 4 \quad if\ n : even$$

Thus the tight upper bound of $TO3(F)$ is $2n - 3$. i.e
]

$$|TO3(F)| \leq 2n - 3 \tag{6}$$

From inequality 5 and inequality 6, it follows $\frac{3n-5}{2} \leq |TO3(F)| \leq 2n - 3$. □
Following Theorem 3, 5, and 7, Corollary 1 can be derived.

Corollary 1: $\frac{3n-5}{2} \leq |TO3(F)| \leq 2n - 3 \leq |SP(F)|$ or $|ISP(F)| \leq 3n - 5$
Thus, To3 code is more compact than BFS, SP and ISP codes.

III.5 EXPECTED NUMBER OF SPs AND TO3s

The length of encoded bits required for a slicing floorplan depends on the number of slicing pairs (SPs) or TO3s present in the corresponding slicing tree. This section presents results on the expected number of SPs and TO3s of a slicing tree with n blocks.

Theorem 8. For skewed slicing tree with n blocks and height $h = \log n$, the expected number of SPs is given by $\frac{4(4^h-1)-3h}{9(n-1)} \approx \frac{4n^2-3\log n-4}{9(n-1)}$.

Proof. Let X_i be a random variable that denotes the number of SPs at level $1 \leq i \leq h$ of the slicing tree. Let X be the random variable that denote the total number of SPs in the tree. Then $X = X_1 + X_2 + \ldots + X_h$. Clearly $X_i \in \{1,2,4,\ldots,2^{i-1}\}$, since the root of the tree does not participate in any slicing pair. The tree contains n leaves and $n-1$ internal nodes; hence the cardinality of the sample space is $n-1$. Let $E[X]$ and $E[X_i]$ are expectation of random variables X and X_i respectively. Then $E[X_i] = \frac{1}{n-1}\sum_{i=1}^{l}(2^{i-1})^2$. Thus, by linearity of expectation, and taking $h = \log n$,

$$E[X] = \sum_{l=1}^{h-1} E[X_i] = \frac{1}{3(n-1)}\sum_{l=1}^{h-1} 4^l - 1$$
$$= \frac{4(4^h-1)-3h}{9(n-1)}$$
$$\approx \frac{4n^2-3\log n-4}{9(n-1)}.$$

Theorem 9. Consider a skewed slicing tree with n blocks and height $h = \log n$, the expected number of TO3s is given by $\frac{16(4^{h+1}-1)-3h}{15(4^{\lceil \log_4 n \rceil}-1)}$. Here h is taken as a $h-1$ when h is odd and as $h-2$ when h is even.

Proof. Since nodes at even levels $0,2,4,\ldots m$ (where $m = h-1$ if h is odd and $m = h-2$ if h is even) of the slicing tree are roots of TO3s. Let $X_i (i \in \{0,2,4,\ldots m\})$ be a random variable that depresents the number of TO3s at level i and X be the random variable that denotes the total number of TO3sof the slicing tree, then, $X = X_0 + X_2 + X_4 \ldots + X_m$. The cardinality of the sample space is $\frac{4^{\lceil \log_4 n \rceil}-1}{3}$. Let $E[X]$ and $E[X_i]$, denote the expectations of random variables X and X_i respectively. Then, $E[X_i] = \frac{3}{4^{\lceil \log_4 n \rceil}-1} \sum_{i=1}^{\frac{m}{2}} (4^i)^2$. By linearity of expectation,

$$E[X] = \sum_{m=0}^{h} E[X_i] = \frac{1}{5(4^{\lceil \log_4 n \rceil}-1)} \sum_{m=0}^{h} 16^{\frac{m}{2}+1} - 1$$
$$= \frac{16(4^{h+1}-1)-3h}{15(4^{\lceil \log_4 n \rceil}-1)}$$

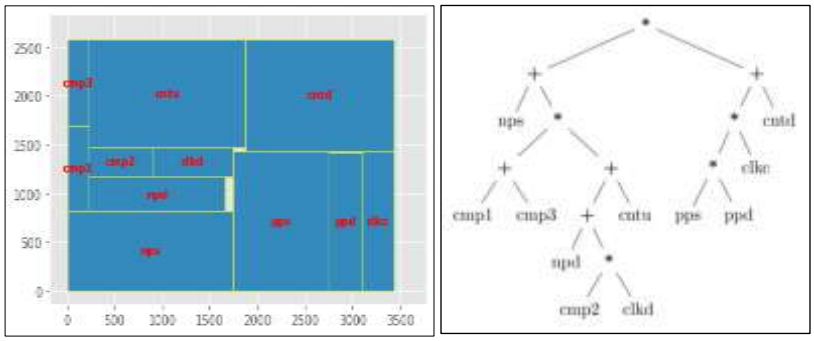
When the height of the slicing tree is odd, $h = h-1$ with $h = \log n$; the expected value is given by $E[X] \approx \frac{16n^2-3\log n-13}{15(n-1)}$. For slicing tree even height, $h = h-2$ with $h = \log n$; in this case, the expected value becomes $E[X] \approx \frac{4n^2-3\log n-10}{15(n-1)}$. □

In the worst case, where both a slicing pair (SP) and a TO3 are encoded using 3 bits. Theorem 8 and Theorem 9 shows that the number of bits required to encode the expected number of TO3s is less than that of SPs.

IV. IMPLEMENTATION AND RESULT ANALYSIS

In this section, the results of the proposed TO3 code are validated with state-of-the-art works using MCNC benchmarks [3] and artificial modules. The proposed and the other algorithms are implemented in Python and executed on a Windows 10 PC equipped with an AMD 3.2 GHz CPU and 8 GB RAM. The length of binary code of slicing trees corresponding to MCNC benchmark floorplans are validated using the B*-tree-based Simulated Annealing (BTBSA) algorithm. Since MCNC benchmark circuits are non-slicing in nature, each floorplan is transformed into a slicing floorplan using the compaction technique discussed in [8]. The resulting slicing trees are then encoded using BFS, SP, ISP, and TO3 coding techniques, which are tabulated in Table 3, which lists the benchmark names, number of modules, and the encoded bit lengths produced by each algorithm.

Figure 5 (a) shows the floorplan of the HP benchmark circuit generated by the BTBSA algorithm, and the corresponding slicing tree is illustrated in Figure 5 (b). For artificial modules, slicing trees are randomly generated for different numbers of modules, and the length of binary code of four encoding techniques is summarized in Table 3. The last row of Table 3 presents the comparison results, where the average values indicate the improvement achieved by the proposed TO3 coding scheme over BFS, SP, and ISP. For a fair comparison, the lengths of the state-of-the-art codes have been adjusted using the TO3 code. From the last row of Table 3, it can be observed that, on average, BFS, SP, and ISP codes need 80%, 46%, and 49% more bits, respectively, than TO3 codes for encoding a slicing tree.



(a) A floorplan generated on HP benchmark. (b) Slicing tree corresponding to floorplan Fig. 5 (a).
Figure 5: A floorplan and its corresponding slicing tree.
Source: Authors, (2026).

Additionally, Figure. 6 (a) and Figure 6 (b) present a comparison of the BFS, SP, ISP, and TO3 code lengths for benchmark and artificial circuits, respectively. The horizontal axis defines different benchmark or artificial circuits, and the vertical axis defines the length of BFS, SP, ISP, and TO3 codes. The vertical bars in red, gray, green, and blue correspond in length to the BFS, SP, ISP, and TO3 codes, respectively. The vertical bar in blue, representing the length of the TO3 code, has the least height among the other encoding schemes for different circuits, as shown in Figure 6. Therefore, compared to BFS, SP, and ISP codes, TO3 code is more compact and uses fewer bits to encode a slicing tree.

Table 3: Comparison of lengths of BFS, SP, ISP, and TO3 codes for benchmark and artificial circuits.

Benchmark / Artificial	No. of Modules / Leaves	BFS Code Length (bits)	SP Code Length (bits)	ISP Code Length (bits)	TO3 Code Length (bits)
ami33	33	97	83	97	57
ami49	49	145	125	153	86
hp	11	31	24	29	18
apte	9	25	19	25	14
xerox	10	28	19	25	14
AM1	5	13	8	9	4
AM2	10	28	21	17	17
AM3	20	58	46	37	27
AM4	35	103	83	69	58
AM5	49	145	118	97	78
Average		1.80	1.46	1.49	1.00

Source: Authors, (2026).

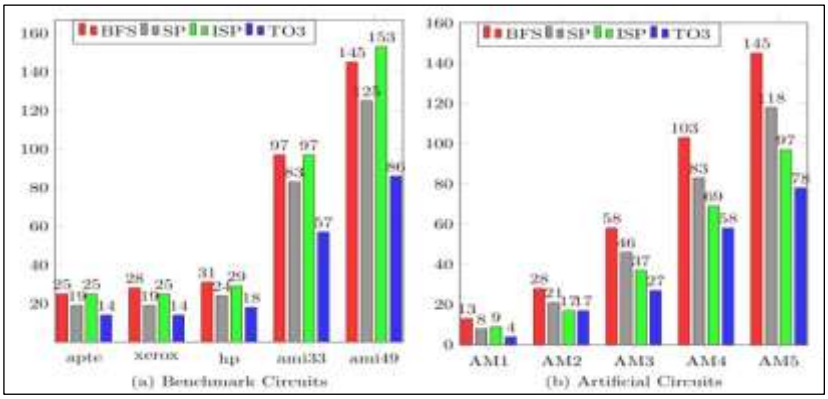


Figure 6: Lengths of BFS, SP, and TO3 codes for benchmark and artificial circuits.
Source: Authors, (2026).

The convergence iteration determines how effectively the algorithm explores and settles into an optimal or near-optimal floorplan layout, which shows an efficient search process and lower computational costs. As shown in Table 4, during the convergence iterations, the slicing tree-based simulated annealing (STBSA) with code representation yields experimental results comparable to those achieved without the code representation of BTBSA. In contrast, the CPU runtime is reduced significantly when the code representation is employed. Specifically, the runtime of STBSA on each benchmark, i.e., ami33, ami49, hp, apte, and xerox, is faster than that of BTBSA by 16%, 11%, 14%, 17%, and 8%, respectively. This improvement arises because the proposed strategy required fewer bits to represent the electronic blocks or modules, thereby reducing computational overhead without compromising the solution quality.

Table 4: Comparison results with and without the slicing tree-based simulated annealing.

Benchmark	Convergence Iterations (With)	Convergence Iterations (Without)	Time (With) (seconds)	Time (Without) (seconds)	Ratio
ami33	59	62	24.23	28.74	0.843
ami49	56	54	41.56	46.65	0.891
hp	19	20	15.52	18.03	0.861
apte	16	14	2.85	3.43	0.831
xerox	11	12	3.95	4.27	0.925

Source: Authors, (2026).

The TO3 code offers the most compact representation when compared with BFS, SP, and ISP, as demonstrated by both the theoretical bounds in Eq. 4 and the implementation results in Table 3. This compactness is consistently observed across small and large circuits, leading to less memory use and faster processing. A key advantage of the TO3 code is that it is faster in terms of CPU runtime because of its simpler structure and efficient encoding process. Since each TO3 is a disjoint subtree, modification or perturbation to any particular TO3 triplet affects only that triplet and its immediate neighbors, unlike in SP or BFS, where dependencies can spread across the entire structure. This disjoint property makes TO3 more efficient for both encoding and optimization tasks.

V. CONCLUSION

This paper proposed a compact binary encoding technique, namely TO3, for the skewed slicing tree that corresponds to a slicing floorplan, and derived an analytical expression for both the length and range length of the TO3 code. The theoretical analysis demonstrates that the expected number of TO3s is smaller than the number of slicing pairs (SPs) required to encode a slicing tree. More importantly, the experimental results on benchmark circuits and artificial modules show that the proposed coding scheme is 80%, 46%, and 49% more compact than the BFS, SP, and ISP codes, respectively. In addition, the code-based simulated annealing reduces CPU running time compared to the B*-tree-based simulated annealing. In the future, this work will be extended to non-slicing floorplans.

VI. AUTHOR'S CONTRIBUTION

Conceptualization: Biswojit Nayak and Mrutynjaya Panda.
Methodology: Biswojit Nayak.
Investigation: Biswojit Nayak and Mrutynjaya Panda.
Discussion of results: Biswojit Nayak and Mrutynjaya Panda.
Writing – Original Draft: Biswojit Nayak.
Writing – Review and Editing: Biswojit Nayak and Mrutynjaya Panda.
Resources: Biswojit Nayak.
Supervision: Mrutynjaya Panda.
Approval of the final text: Biswojit Nayak and Mrutynjaya Panda.

VII. REFERENCES

[1] B. Yang, Q. Xu, H. Geng, S. Chen, B. Yu, and Y. Kang, "Floorplanning with edge-aware graph attention network and hindsight experience replay," ACM Trans. Des. Autom. Electron. Syst., vol. 29, no. 5, pp. 1–17, 2024.

[2] D. Wong and C. Liu, "A new algorithm for floorplan design," in 23rd ACM/IEEE design automation conference, IEEE, 1986, pp. 101–107.

[3] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: A new representation for non-slicing floorplans," in Proceedings of the 37th annual design automation conference, 2000, pp. 458–463.

[4] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An o-tree representation of non-slicing floorplan and its applications," in Proceedings of the 36th annual ACM/IEEE design automation conference, 1999, pp. 268–273.

[5] P.-N. Guo, T. Takahashi, C.-K. Cheng, and T. Yoshimura, "Floorplanning using a tree representation," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 20, no. 2, pp. 281–289, 2001.

[6] J.-M. Lin and Y.-W. Chang, "TCG: A transitive closure graph-based representation for non-slicing floorplans," in Proceedings of the 38th annual design automation conference, 2001, pp. 764–769.

[7] X. Hong et al., "Corner block list: An effective and efficient topological representation of non-slicing floorplan," in IEEE/ACM international conference on computer aided design. ICCAD-2000. IEEE/ACM digest of technical papers (cat. No. 00CH37140), IEEE, 2000, pp. 8–12.

[8] M. Lai and D. Wong, "Slicing tree is a complete floorplan representation," in Proceedings design, automation and test in europe. Conference and exhibition 2001, IEEE, 2001, pp. 228–232.

[9] C.-C. Yeh, S.-C. Lu, H.-L. Cho, Y.-C. Lin, and R.-B. Lin, "Subitizing-inspired large language models for floorplanning," in Proceedings of the symposium on assessment of synthesis and integration of mixed information (SASIMI 2025), Yuan Ze University, 2025, pp. 223–228.

[10] K. Yamanaka and S. Nakano, "Coding floorplans with fewer bits," IEICE transactions on fundamentals of electronics, communications and computer sciences, vol. 89, no. 5, pp. 1181–1185, 2006.

[11] T. Takahashi, R. Fujimaki, and Y. Inoue, "A (4 n- 4)-bit representation of a rectangular drawing or floorplan," in Computing and combinatorics: 15th annual international conference, COCOON 2009 niagara falls, NY, USA, july 13-15, 2009 proceedings 15, Springer, 2009, pp. 47–55.

[12] M. Saito and S. Nakano, "Two compact codes for rectangular drawings with degree four vertices," in Proc. 11th forum on information technology (FIT 2012), 2012, pp. 1–8.

[13] T. Takahashi, "A compact code for rectangular drawings with degree four vertices," Journal of information processing, vol. 22, no. 4, pp. 634–637, 2014.

[14] K. Sakanushi, Y. Kajitani, and D. P. Mehta, "The quarter-state-sequence floorplan representation," IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol. 50, no. 3, pp. 376–386, 2003.

[15] B. He, "Optimal binary representation of mosaic floorplans and baxter permutations," in Frontiers in algorithmics and algorithmic aspects in information and management: Joint international conference, FAW-AAIM 2012, beijing, china, may 14-16, 2012. proceedings, Springer, 2012, pp. 1–12.

[16] T. Ohmori, K. Yamanaka, T. Hirayama, and Y. Nishitani, "Compact codes of slicing floorplans," 電子情報通信学会技術研究報告= IEICE technical report: 信学技報, vol. 114, no. 80, pp. 33–37, 2014.

[17] B. Nayak and B. Ray, "ISP: An improved slicing pair code for skewed slicing floorplan," in 2023 36th international conference on VLSI design and 2023 22nd international conference on embedded systems (VLSID), IEEE, 2023, pp. 205–210.

[18] S. Vinay Kumar, P. Rao, and M. K. Singh, "Optimal floor planning in VLSI using improved adaptive particle swarm optimization," Evolutionary Intelligence, vol. 15, no. 2, pp. 925–938, 2022.

[19] B. Srinivasan and R. Venkatesan, "Multi-objective optimization for energy and heat-aware VLSI floorplanning using enhanced firefly optimization," Soft Computing, vol. 25, no. 5, pp. 4159–4174, 2021.

[20] V. V. Ignatyev, A. V. Kovalev, O. B. Spiridonov, V. M. Kureychik, A. S. Ignatyeva, and I. B. Safronenkova, "Hybrid genetic–paired–permutation algorithm for improved VLSI placement," ETRI Journal, vol. 43, no. 2, pp. 260–271, 2021, doi: 10.4218/etrij.2019-0412.

[21] S. Yu, S. Du, and C. Yang, "A deep reinforcement learning floorplanning algorithm based on sequence pairs," Applied Sciences, vol. 14, no. 7, p. 2905, 2024.