3-D Floorplanning Using Labeled Tree and Dual Sequences

Renshen Wang[†] Evangeline F. Y. Young[‡] Yi Zhu[†]
Fan Chung Graham[†] Ronald Graham[†] Chung-Kuan Cheng[†]

[†]Department of Computer Science and Engineering, University of California, San Diego
[‡]Department of Computer Science and Engineering, The Chinese University of Hong Kong
rewang@cs.ucsd.edu fyyoung@cse.cuhk.edu.hk {y2zhu, fan, graham, ckcheng}@ucsd.edu

ABSTRACT

3-D packing is an NP-hard problem with wide applications in microelectronic circuit design such as 3-D packaging, 3-D VLSI placement and dynamically reconfigurable FGPA design. We present a complete representation for general non-slicing 3-D floorplan or packing structures, which uses a labeled tree and dual sequences. For each compact placement, there is a corresponding encoding. The number of possible tree-sequence combinations is $(n+1)^{n-1}(n!)^2$, the lowest among complete 3-D representations up to date. The construction of placement from an encoding needs $O(n^2)$ in the worst case, but in practical cases we expect $O(n^{4/3} \log n)$ time on average for circuit blocks with limited length/width ratios. Experimental results show promising performance using the labeled tree and dual sequences on 3-D floorplan and placement optimizations.

Categories and Subject Descriptors

J.6 [**Computer Applications**]: Computer-aided design

General terms - Algorithms, Design

Keywords - 3-D packing, labeled tree, sequence

1. INTRODUCTION

The 3-D packing problem is a classic combinatorial problem. The basic formulation of 3-D packing is to place a set of cuboidal blocks in a box such that no two blocks overlap and the box volume is minimized. In microelectronic circuit design areas, there are similar problems on block placement with various applications besides minimizing volume, such as optimizing wire length and heat distributions. Also in dynamically reconfigurable FPGAs, the resources can be reused for different tasks, and the task scheduling is a 3-D packing problem regarding time as the third dimension.

2-D and 3-D packing are NP-hard problems because they reduce to the NP-complete 1-D bin packing. Some approximation algorithms with bounded performance ratio (in terms of volume or dimension) are devised in theoretical works like [5]. For some floorplan optimizations on more strict or complex objectives, we often need search schemes like simulated annealing based on a compact encoding called floorplan representations. A representation encoding decides the geometric relationships among the blocks and thus determines the structure of a floorplan. A good representation should be easy to operate and efficient for the optimization approaches.

1.1 Previous Works on 2-D and 3-D Representations

2-D floorplan representations are first developed. Sequence based models like Sequence pair [7], Corner block list [4], tree based models like O-tree [3], Twin binary tree [11] and graph based model like Bounded slice grid [8], Transitive constraint graph [12], *etc* are proposed.

These 2-D representations are all complete for non-slicing structures, which means that any compact placement has an instance in the encoding set of the representation. The transformations between placement and encoding are mostly in linear time or $O(n \log\log n)$. Together with local searching algorithms, the 2-D representations are very effective tools for current placement optimization in microelectronic designs.

3-D VLSI circuits and packaging are emerging, because first, the signal delay are dominated by interconnections, which can be reduced by stacking chips in the third dimension; second, 3-D integration technologies are now being put into practical use. One of the big challenges in 3-D IC industry is in the design tools.

Previous works of 3-D representations include Seq-triple [10], Seq-quintuple [10], which are extended from Seq-pair [7], and other representations like 3-D subTCG [12], 3-D BSG [13], 3-D CBL [6], *etc.* Each one is extended from the corresponding 2-D model. A problem with most of the extended models is that the completeness property in the original 2-D model is lost: the special "cyclic" cases (Fig.1, more discussed in section 2) are missing in most representations. The complete 3-D models among them are Seq-quintuple, which are highly redundant by using 5 sequences, and subTCG which is a direct copy of the geometric relations.

Results of previous research indicate that there is some inherent complexity in 3-D packing. A more compact and efficient encoding on the 3-D geometric relations is desired for effective optimization algorithm on 3-D placement.

1.2 Contributions

We use a labeled tree [1] and two sequences to encode a packing process of 3-D placement. The idea is similar to the 2-D O-tree [3] representation: the blocks are sequentially pushed into position and thus the placement can be recovered from the encoding.

The advantages of the labeled tree and dual sequences include: it is a complete representation of compact 3-D placements; the total number of combinations is $(n+1)^{n-1}(n!)^2$, so it has a smaller solution space to search within; to transform the encoding into a placement needs at most $O(n^2)$ time, which can be lowered to $O(n^{4/3}\log n)$ in average cases of circuit blocks with limited length/width ratios. These properties are the key factors for optimization algorithms to find better solutions in shorter time.

The rest of the paper is organized as follows. Section 2 states the 3-D floorplanning problem and some difficulties for representations. Section 3 describes the labeled tree and dual sequences representation we propose. Section 4 discusses the conversions between packing instances and encodings. Finally, experimental results and conclusions are in sections 5 and 6.

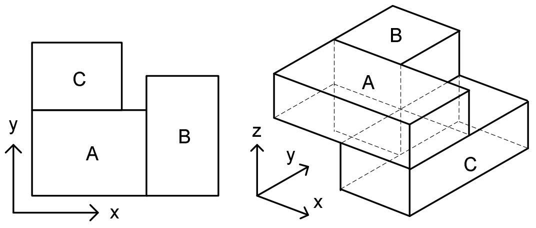


Figure 1: 2-D and 3-D packing

2. PROBLEM STATEMENT

A set of cuboidal blocks $B = \{B_1, B_2, \ldots, B_n\}$ lying parallel to the coordinate axes. Each block B_i has 3 dimensional sizes (w_i, l_i, h_i). A placement $P = \{(x_i, y_i, z_i) : 1 \le i \le n\}$ is an assignment of coordinates at the lower-left-front point of each block such that no two blocks overlap each other. A representation of a placement is the encoding E from which the placement can be recovered together with $\{(w_i, l_i, h_i)\}$.

A *compact* placement is defined as a placement where each block is pushed to the lower-left-front corner, so that for each B_i, the 3 surfaces at its $-x, -y$ and $-z$ side are all touching either the boundary or another block. Fig.1 shows a 2-D compact placement as well as a 3-D one. For most optimization objectives such as volume and interconnections, good solutions are usually compact because we always need the solution to be as small as possible. Even for the interconnection objective, where sometimes we need non-compact placement, the optimal solution can be obtained from a compact solution by small movements on some blocks.

A loop of *cyclic* geometric relations exists in the 3-D placement in Fig.1, which is among block A, B and C, "$A \Rightarrow^y B$", "$B \Rightarrow^x C$", "$C \Rightarrow^z A$". Here "$\alpha \Rightarrow^d \beta$" means α's $+d$ surface has a d-coordinate value no greater than β's $-d$ surface, and the two surfaces have overlapping projections on d's orthogonal plane. In this way, the three blocks prop up one another, and none of them can reach the corner point $(0, 0, 0)$. The packing process of this case has a cyclic dependency problem: To place block A, we need to know the coordinates of C; to place C we need to know B; to place B we need to know A. Therefore it is not feasible to sequentially push the blocks one by one to the $(-x, -y, -z)$ corners.

The cyclic case does not exist in 2-D placement where the lower-left corner on the boundary is always occupied by a block. This is one of the reasons why sequence based representations which work perfectly for 2-D placement fail to encode 3-D placement in the same way. We need to take special care of the cyclic dependency issue when extending 2-D representations.

By the statements above, we need a representation model with following properties:

• Completeness. Every possible compact placement of blocks can be represented.

• The number of combinations should be as small as possible so that the optimization algorithm can search in a relatively small solution space.

• Operations on the encoding are fast and convenient.

Based on the representation model, various algorithms can be applied to search for best solutions in terms of low volume, low wire length (interconnection delay & power), or low concentration of heat sources, *etc.*

3. REPRESENTATION OF LABELED TREE AND DUAL SEQUENCES

We introduce a representation using a labeled tree [1] and two sequences, which is complete for 3-D compact placement with relatively small solution space, and is easy to operate. The completeness of this encoding is due to the separation of z coordinate encoding: once the coordinate values on z axis is decided, the dependency cycle is broken and the "cyclic case" in section 2 can be represented. Details of the tree and sequences are described in the following subsections.

3.1 Labeled Tree

First, a labeled tree L specifies the z coordinates of the blocks. A labeled tree [1] is an unordered tree with a label on each node. For an n-block case, we use a labeled tree with $n + 1$ nodes, one label indicating the root node and others each indicating a block B_i.

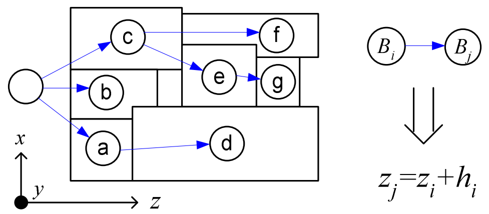


Figure 2: A labeled tree which decides z coordinates

The root of L is the lower boundary $z_{root} = 0$ with zero height $h_{root} = 0$. The children are on the $+z$ side of their parents with zero separation distance. For each tree branch $B_i \to B_j$, $z_j = z_i + h_i$. As shown in Fig.2, a set of blocks are piled up in z direction. We construct the labeled tree by connecting each block B_i to its parent which is the block touching B_i's $-z$ surface. On the other hand, by traversing the tree, the z coordinate of each block can be recovered.

The way a labeled tree determines z coordinates is just like an O-tree [3] determines x coordinates. The difference is that the nodes in the O-tree have an order which can determine the y coordinates, while the nodes in the labeled tree are unordered. For example in Fig.2, if we exchange the positions of the two branches of node c, "f" and "$e \rightarrow g$", the resulting tree is equivalent to the original one. We choose the unordered tree because the traversal orders in trees are not enough to cover the packing orders in 3-D placement. As a result, we have fewer combinations than those of ordered trees. For the tree with $n+1$ nodes, the total number of combinations is $(n+1)^{n-1}$ by the results in [1].

In summary, we use labeled tree L to decide the z coordinates with the idea of O-tree [3]. The z coordinates are determined separately, independent of the x and y coordinates. Thus the relations of cyclic dependency among the blocks are broken, and the rest part of the representation needs to pack the blocks to decide their x and y coordinates.

3.2 Permutation and X-sequence

The x and y coordinates are decided by two sequences. We use a permutation $P[1 \cdots n]$ and a number sequence $X[1 \cdots n]$ to decide the packing process, and thus decide the placement of each block. Since the cyclic relation problem is resolved, the blocks can be packed in a sequential order like in 2-D packing: Push the blocks one by one, each to a corner until the block cannot be pushed further towards $-x$ or $-y$.

The order in which the blocks are pushed is by the permutation P, and the x coordinates are decided by X: each block $P[i]$ is placed upon block $P[X[i]]$ on the x axis, i.e. $x_{P[i]} = x_{P[X[i]]} + w_{P[X[i]]}$, where $0 \le X[i] < i$ and block $P[i]$ is touching the $-x$ boundary if $X[i] = 0$. Number $X[i]$ has the same effect as a branch in the labeled tree. By its definition, the X-sequence has $n!$ combinations, same as the number of permutations.

With (x, z) known, the y coordinate of each block is determined by pushing the block towards $-y$ direction until it touches the boundary or another block. For block B_i, its y coordinate $y_i = \max_{k \in \psi(i)} y_k + h_k$, where $\psi(i)$ is the set of blocks covered by B_i, i.e. set of B_k which is placed previously and has a non-zero area overlapping with B_i. The $-y$ boundary is regarded as a block B_0 with $y_0 = 0$ and $h_0 = 0$.

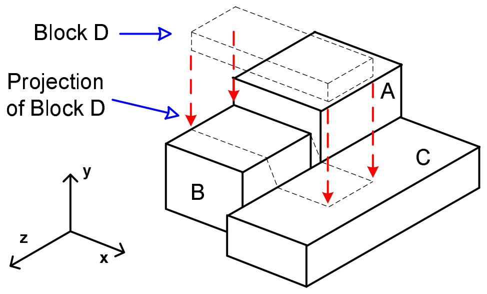


Figure 3: Pushing a block towards $-y$

An example of the pushing process is shown in Fig.3. Block A, B and C are placed and block D is pushed towards $-y$ with its projection covering B and C. In this case, the y coordinate of D should be the height of B. With n pushing processes, a compact 3-D placement can be recovered from the encoding. A complete representation of the example in Fig.2 is shown in Fig.4.

In summary, 3-D placement can be represented by a labeled tree L, a permutation P and a number sequence X. The total number of combinations is $(n+1)^{n-1}(n!)^2$. To recover the placement, we first traverse labeled tree L to compute z coordinates, and then follow permutation P and sequence X to push the blocks into position. If the blocks have limited length/width ratio, the entire placement can be recovered in average $O(n^{4/3} \log n)$ time by maintaining a map on the $+y$ surface.

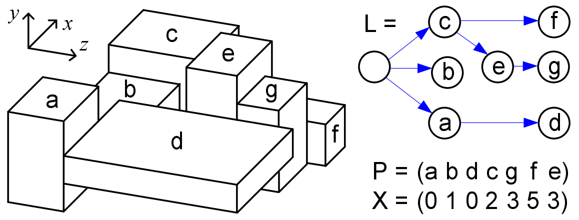


Figure 4: Full representation of an example

4. CONVERSIONS BETWEEN ENCODING AND PLACEMENT

In floorplan optimization algorithms, an important property of the representation is completeness, i.e. every possible placement can be represented in the encoding. The conversion from any compact 3-D placement to our representation (L, P, X) is shown in section 4.1.

On the other hand, an efficient conversion from encoding to placement is crucial for a representation, because heavy amount of these conversions are needed in optimization algorithms. In our representation of labeled tree and dual sequences, we maintain a 2-D map on the height of each block that can be seen from top view (Fig.5). The blocks are placed one by one with the map updated with each block.

4.1 Placement to Representation

Given any compact placement of blocks, we can always generate a labeled tree and dual sequences by their definitions in the last section.

THEOREM 1. *The representation of labeled tree and dual sequences is complete for 3-D compact placement.*

PROOF. First, the labeled tree can be constructed from the placement. Since the placement is compact, each block must be touching a neighbor or the boundary on its $-z$ surface. Take each block and the boundary as a node, and connect each block to its parent which is the touching neighbor at $-z$ direction (in case of multiple neighbors, connect to one of them). The result is a tree rooted at the boundary node, because each node has one and only one parent except for the boundary node.

The permutation and X sequence are then constructed from the x and y coordinates of the blocks. By the definition of the sequence, we reverse the pushing process (from $P[n]$ to $P[1]$): at each step we pick a block as $P[i]$ which has the largest x coordinate from the set completely viewable from $+y$ top view, i.e. the set $\{B_k : B_k \notin \psi(j) \text{ for } j \notin \{P[i+1], \ldots, P[n]\}\}$. Clearly the "cover" relation is noncyclic and the set is always non-empty at each step. After the permutation is constructed, we find each block's touching neighbor at $-x$ direction and thus construct the X sequence. □

With the completeness guaranteed, we can apply this representation in automatic searching algorithms, and the optimal compact packing solution is always included in the searching space. However, not all the instances in the representation are compact placements. So there is still some redundancy in this encoding, which also exists in O-tree ([3], "non-admissible"). This indicates that more efficient encoding is possible.

4.2 Representation to Placement

To recover a placement from representation, we need a pushing process for the placement of each block, which involves a set of blocks being covered on a 2-D map. The map needs to be stored in some data structure and be maintained by certain operations. When block B_i is being pushed, we need to know $\psi(i)$, the set of blocks covered by B_i, and then add B_i into the map.

For general cases with arbitrary blocks, there are some packing instances with $O(n^2)$ pairs of blocks in "cover" relation, and we need at least $O(n^2)$ time for the entire packing process. However, these cases are not common. In practice, the blocks usually have a limited length/width ratio, because practical boxes or circuit modules are usually not too "long" or "thin". With the ratio limit set as k, maintaining the 2-D map in average cases needs $O(n^{4/3}\log n)$ time. The details of the algorithm are as follows.

4.2.1 Data Structure

The map can be treated as a 2-D mosaic floorplan [4] [11] consisting of a set of rectangular blocks. The geometric information of the blocks can be stored as a corner stitching graph [9] $G = (V, E)$ where each node in V is a block and has at most 8 edges: 2 contacting neighbors at each corner times 4 corners. Each block has a y value as its height in the 2-D map.

To quickly locate the blocks covered by B_i, we also construct a quaternary search tree (same idea as the binary search tree in [2]) which contains all the block corners. Each tree node has x and z coordinates as index values. For node V_0 with coordinate (x_0, z_0), the four children are the roots of four subtrees:

· the upper-left subtree with nodes $\{P : x_p \le x_0, z_p > z_0\}$,
· the upper-right subtree with nodes $\{P : x_p > x_0, z_p > z_0\}$,
· the lower-left subtree with nodes $\{P : x_p \le x_0, z_p \le z_0\}$,
· the lower-right subtree with nodes $\{P : x_p > x_0, z_p \le z_0\}$.

The center point of each area can be taken as the root of the subtree. Average time for adding and deleting a node in the quaternary search tree is $O(\log n)$.

After locating a block covered by B_i, the corner stitching structure helps to find all the blocks in $\psi(i)$ in linear time by the neighbor finding algorithms in [9].

4.2.2 2-D Map

With the basic data structure and operations of the 2-D map, we can recover the 3-D placement by sequentially pushing the blocks onto the map. The y coordinate of each block $B_{P[i]}$ is the highest block it covers on the map, with its x and z coordinates computed from L, P and X.

The map is updated each time a block is pushed. The new block $B_{P[i]}$ is included in the new map, replacing the area it covers. Fig.5 shows an example of map updating: block A and B are already placed and block C is newly added. Originally, there are block A and B plus three "filling" blocks

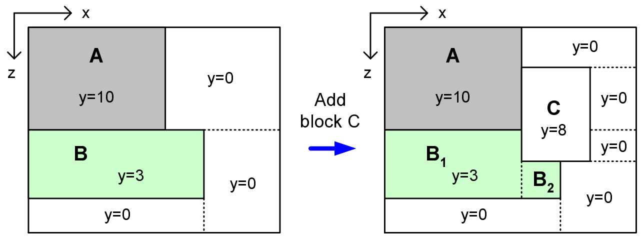


Figure 5: Top view of a 2-D map

which fill the empty space. When a new block C is added, it covers part of block B and two filling blocks. The y coordinate of C is decided by the highest covered block which is B. Then we add block C into the map, while we also need to update the blocks covered by C. There are basically 3 cases for a block (B_k) in the map covered by another block:

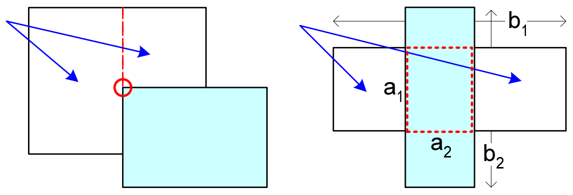
1) Completely covered: delete B_k from the map;

2) Partially covered, with a rectangular remaining part: update the corners' coordinates and neighbors;

3) Partially covered, with non-rectangular remaining part(s): partition the remaining part into rectangular small blocks and add new blocks into the map.

In case 3 above, we keep the map always consisting of rectangular blocks by partitioning each non-rectangular shape into rectangular parts (as in Fig.5). The upper bound of the length/width ratio helps to bound the average number of blocks in the map.

Theorem 2. If *all* the blocks appearing in the map have length/width ratio bounded by k, the total number of 2-D blocks in the map does not exceed $(k + 5)n$.



(a) Corner induced **(b) Non-corner induced**

Figure 6: Two cases of partitioning

Proof. Starting from the flat 2-D map which is a big rectangle, we add n blocks, and each block may produce additional blocks because of partial covering. Two cases of partial covering can induce additional blocks:

1) Non-rectangular shape by corners (Fig.6a). Each corner induces at most one additional block. With n blocks, the number of corner induced blocks is bounded by $4n$. Consider that the first block must touch the $-x$ boundary, we can reduce the upper bound to $4n - 1$.

2) Block partitioning by segments (Fig.6b). A block crossing another block on the map with no corner involved. Assume the dimensions of these blocks are $a_1 \times b_1$ for B_1 and $a_2 \times b_2$ for B_2. The crossing area of the two blocks is $a_1 \times a_2$, which occupies a_2/b_1 of B_1's upper surface and a_1/b_2 of B_2's lower surface. Since the length/width ratio is limited by k, the sum of these two proportions of occupied areas

$a_1/b_2 + a_2/b_1 \ge \frac{a_1}{ka_2} + \frac{a_2}{ka_1} \ge 2\sqrt{\frac{a_1}{ka_2}\frac{a_2}{ka_1}} = 2/k$

If we add the occupied proportions of all the crossing areas, the result is at most $2n$ because each block has one upper surface and one lower surface. The number of crossing areas, which is also the number of non corner-induced blocks, is therefore no larger than $2n/\frac{2}{k} = kn$.

Adding the numbers together, the total number of blocks does not exceed $1 + n + (4n - 1) + kn = (k + 5)n$ $\square$

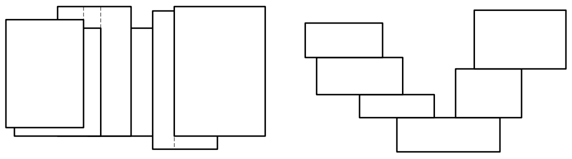


Figure 7: Consecutive "long" blocks in the map

By the deduction of theorem 2, the linear upper bound $(k+5)n$ holds if there are only two layers of blocks, with one layer covering the other. However in actual cases there are most likely $O(n^{1/3})$ layers of blocks, because the n blocks with limited shapes are packed in a box of average size $O(n^{1/3})\times O(n^{1/3})\times O(n^{1/3})$. The additional blocks on the map induced by partial covering do not necessarily have bounded length/width ratio. We may have a set of "long" blocks as in Fig.7a. The only way to form a consecutive long block set is to stack the blocks in $+y$ direction as in Fig.7b. With $O(n^{1/3})$ layers, the size of the set is also $O(n^{1/3})$ and each new block produces at most $O(n^{1/3})$ times more new blocks on the map than in theorem 2. Thus, in average cases the estimated number of blocks appearing in the 2-D map is $O(n^{4/3})$.

4.2.3 Algorithm

Based on the 2-D map manipulations, the algorithm to convert the labeled tree and dual sequences instance to a placement is in the following chart.

```
Encoding_to_Placement (Labeled tree L,
                Permutation P, Sequence X)
1. Compute $z$ coordinates by traversing L
2. For $i = 1$ to $n$ $\{x_{P[i]} \leftarrow x_{P[X[i]]} + w_{P[X[i]]}\}$ endfor
3. Initialize quaternary search tree Q
   and 2-D map G=(V,E)
4. For $i = 1$ to $n$
5.     $\psi_{P[i]} \leftarrow$ the blocks covered by $B_{P[i]}$
6.     $y_{P[i]} \leftarrow \max_{k\in\psi(P[i])} y_k + h_k$
7.     Update Q and G for $B_{P[i]}$ and $B_k \in \psi_{P[i]}$
8. Endfor
```

By quaternary search tree Q, locating a block covered by $B_{P[i]}$ needs $O(\log n)$ time. By theorem 2 and the corner stitching graph G [9], finding $\psi(P[i])$ needs average time $O(n^{1/3})$, and updating tree Q and graph G needs average time $O(n^{1/3}\log n)$. The total estimated time added up is therefore $O(n^{4/3}\log n)$.

5. EXPERIMENTAL RESULTS

Based on the representation, optimization algorithms can be applied on a set of blocks to search for a placement with minimum volume, minimum interconnect length, minimum number of hot spots, or combinations of the three, *etc.*

We use simulated annealing as the basic optimization algorithm. Besides the standardized annealing scheme, we need a perturbation routine on a representation instance so that the neighbor of an instance can be randomly generated. The perturbation includes five basic operations:

1) Rotation of a block. A cuboidal block parallel to coordinate axes has 24 directions. For volume minimization, we can reduce them to 6 directions.

2) Exchanging two nodes in L.

3) Moving a leaf node to another node in L, which changes the tree structure.

4) Exchanging two nodes in P.

5) Changing a number in sequence X.

The five operations guarantee that any instance is reachable from an arbitrary initial instance. Each operation is based on random selection of nodes or numbers. Together with the completeness of representation in theorem 1, the simulated annealing algorithm is capable of finding good solutions. The results of our implementation are as follows.

We use two sets of benchmarks: one is the "Beasley & OKP" cases used in previous works for dead space minimization; another one is from MCNC benchmarks with interconnection optimization.

Table 1 is the comparison of results from previous works and our experiments in comparable running times. The dead space in a solution is $\frac{Packing\ volume\ -\ total\ block\ volume}{Packing\ volume}$ and the time consumptions of our program are listed in the last column. The results of previous works are from [13]. Column "$\#xyz$-loops" of Table 1 is the number of cyclic relation loops (Fig.1) in our solution. Since most of other representations exclude some cyclic relations, this number indicates the effectiveness of our encoding scheme. Though our results on the relatively small "Beasley" cases are not significantly better, on the larger "OKP" cases, where the searching space becomes super-exponentially large, our "Tree+Seq^2" produces greatly improved results than other representations.

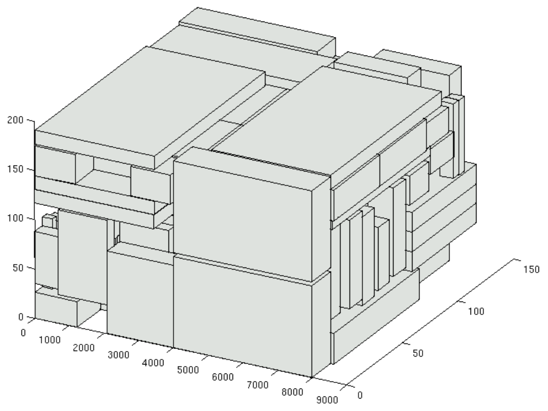


Figure 8: Packing solution of case OKP5

Another set of benchmarks are based on two cases of the MCNC benchmarks. Since the blocks have only two dimensions, we add a third dimension which is the average of the other two, i.e. "$Height = \frac{Length+Width}{2}$". The wire length of a net is $\frac{(x_{max}-x_{min})+(y_{max}-y_{min})+(z_{max}-z_{min})}{2}$, where each pin's coordinate (x_p, y_p, z_p) is assumed to be at the center of the block. We test "ami33", "ami49" and other two cases which are produced by duplicating the blocks 10 times. In the "×10" cases, each pin on the block is connected with the

Table 1: Dead space comparison among various representations

Test case	#blocks	Total volume	Seq-Triple	3D-subTCG	3D CBL	3D-BSG	Tree+Seq^2	#xyz-loops	Time
beasley1	10	6218	28.6%	17.1%	23.5%	17.1%	19.3%	1	4s
beasley2	17	11497	21.5%	7.2%	7.0%	14.7%	15.1%	2	9s
beasley3	21	10362	35.3%	18.0%	17.0%	12.9%	12.0%	6	11s
beasley5	14	16734	26.4%	11.5%	13.5%	12.3%	12.6%	3	8s
beasley6	15	11040	26.3%	16.3%	15.4%	16.8%	14.8%	1	9s
beasley7	8	17168	30.1%	16.5%	24.6%	16.5%	28.5%	0	3s
beasley10	13	493746	25.2%	14.2%	15.2%	17.5%	20.3%	18	8s
beasley11	15	383391	24.8%	12.6%	13.2%	10.9%	16.3%	6	11s
beasley12	22	746158	29.9%	21.5%	21.2%	18.9%	17.3%	21	19s
okp1	50	124358256	42.6%	28.4%	29.1%	24.6%	18.2%	7	151s
okp2	30	85445223	33.2%	22.3%	27.0%	18.9%	17.5%	12	58s
okp3	30	123808466	33.1%	23.0%	26.3%	20.1%	17.2%	33	61s
okp4	61	238860881	42.8%	27.3%	28.6%	26.6%	16.8%	91	238s
okp5	97	189874755	57.7%	35.8%	36.2%	20.4%	16.3%	224	570s
Average			33.7%	20.7%	22.6%	17.7%	17.3%		

Table 2: Experiments on minimizing wirelength

Test case	#blocks	#nets	Total volume	Packing dimensions	Wire length
ami33	33	123	261857750	$758 \times 756 \times 770$	1428
ami49	49	408	1271919696	$3542 \times 5152 \times 4690$	7662
ami33×10	330	123	2618577500	$1918 \times 1526 \times 3255$	5960
ami49×10	490	408	12719196960	$10623 \times 8414 \times 14233$	28364

same net, so each net has 10 times number of pins. Table 2 shows the results on wire length minimization.

6. CONCLUSIONS

We introduce an encoding scheme of compact 3-D placement by a labeled tree and dual sequences. It is not the only choice of 3-D representation. Following the idea of O-tree[3], we have encoding schemes such as 3 labeled trees, 2 labeled trees + 1 sequence (similar to [10]), and 1 labeled trees + 2 sequences (adopted in this paper).

More efficient encoding of 3-D placement is possible since our representation still has redundancy in most of the instances. Future work may explore more encoding schemes and further reduce the total number of combinations while preserving the completeness of the encoding. Faster packing algorithms are also helpful to improve the overall 3-D placement optimization.

7. ACKNOWLEDGMENTS

We would like to acknowledge the support of NSF CCF-0618163 and California MICRO Program.

8. REFERENCES

[1] A. Cayley. A theorem on trees. *Quart. J. Math.*, 23:376–378, 1889.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*, pages 311–317, 2001.

[3] P. N. Guo, T. Takahashi, C. K. Cheng, and T. Yoshimura. Floorplanning using a tree representation. *IEEE Trans. on CAD*, 20(2):281–289, Feb. 2002.

[4] X. Hong et al. Corner block list: An effctive and effient topological representation of non-slicing florplan. *Int. Conf. on Computer-Aided Design*, pages 8–12, Nov. 2000.

[5] K. Li and K. H. Cheng. On three-dimensional packing. *SIAM J. Computing*, 19(5):847–867, oct. 1990.

[6] Y. Ma, X. Hong, S. Dong, and C.K.Cheng. 3D CBL: an efficient algorithm for general 3-dimensional packing problems. *Midwest Symp. on Circuits and Systems*, pages 1079–1082, 2005.

[7] M. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI module placement based on rectangle-packing by the sequence pair. *IEEE Trans. on CAD*, pages 1518–1524, Dec. 1996.

[8] S. Nakatake et al. Module packing based on the BSG-structure and IC layout applications. *IEEE Trans. on CAD*, pages 484–491, Jun. 1998.

[9] J. K. Ousterhout. Corner stitching: a data structuring technique for VLSI layout tools. Technical Report UCB/CSD-83-114, EECS Department, University of California, Berkeley, 1983.

[10] H. Yamazaki, K. Sakanushi, S. Nakatake, and Y. Kajitani. The 3D-packing by meta data structure and packing heuristics. *IEICE Trans. Fundamentals*, pages 639–645, Apr. 2000.

[11] B. Yao, H. Chen, C. K. Cheng, and R. Graham. Floorplan representations: complexity and connections. *ACM Trans. on Design Automation of Electronic Systems*, pages 55–80, Jan. 2003.

[12] P. H. Yuh, C. L. Yang, Y. W. Chang, and H. L. Chen. Temporal floorplanning using 3D-subTCG. *Asia and Pacific Design Automation Conf.*, pages 725–730, Jan. 2004.

[13] L. Zhang, S. Dong, X. Hong, and Y. Ma. A fast 3D-BSG algorithm for 3D packing problem. *Int. Symp. on Circuits and Systems*, pages 2044–2047, May 2007.