An O-Tree Representation of Non-Slicing Floorplan and Its Applications

Pei-Ning Guo, Chung-Kuan Cheng, Takeshi Yoshimura

Mentor Graphics Corp.
San Jose, CA

Department of Computer Science and Engineering
University of California, San Diego

NEC Corp., Japan

Outlines

1. Introduction
2. Problem Statement
3. Admissible placement and constraint graph
4. O-tree and placement
5. Floorplan algorithms using O-tree
6. Experimental results
7. Conclusion

Previous Works

1. Slicing and non-slicng

2. Preas et al.: binary tree - block as leaf, V/H relation as internal nodes

   $O(N!2^{3N-2}/N^{1.5})$ configurations (N: # of blocks)

3. Onedera et al.: non-slicing, branch-and-bound

   $O(2^{N(N+2)})$ configurations

4. Murata et al.: sequence pair

   $O((N!)^2)$ configurations

5. J. Xu et al: cluster refinement

   $O(N^{2+K/2})$ complexity/iteration

6. O-tree:

   $O(N!\, 2^{2N-2}/N^{1.5})$ configurations

Storage Needed for Individual Configuration

1. Preas et al.: binary tree - block as leaf, V/H relation as internal nodes

$$N\,(\,3 + \lceil \lg N \rceil\,) \text{ bits } (N\text{: \# of blocks})$$

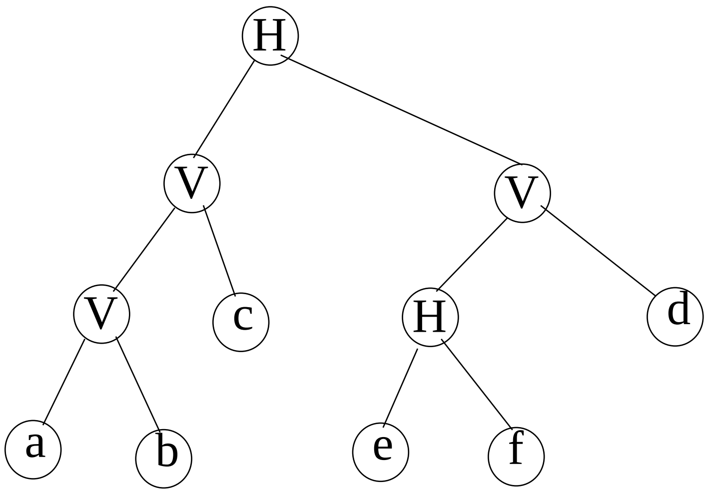
2. Murata et al.: sequence pair

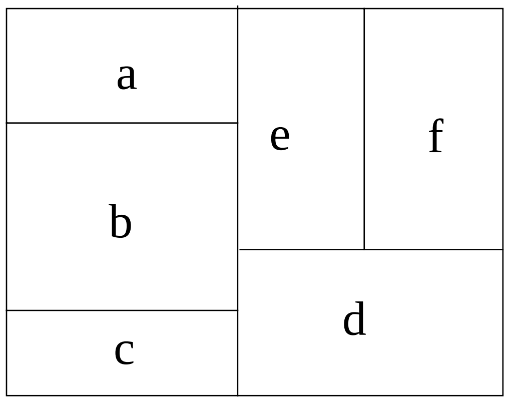
$$2\,N \lceil \lg N \rceil \text{ bits}$$

3. O-tree:

$$N\,(\,2 + \lceil \lg N \rceil\,) \text{ bits}$$

Slicing Structure

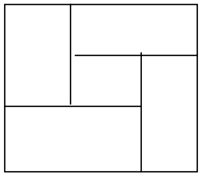


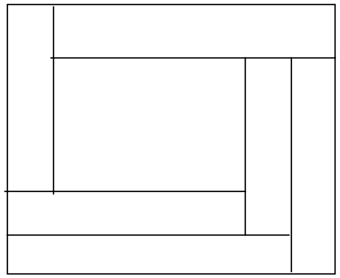


Polish Expression: (((abV)cV)((efH)dV)H)

Redundancy: (abV)cV or a(bcV)V

Non-slicing:

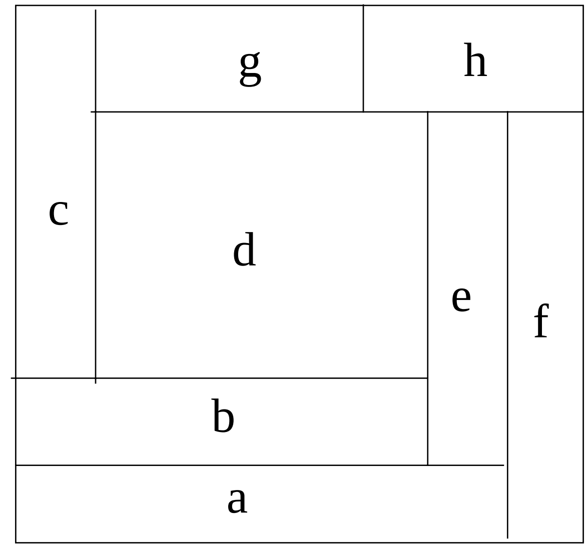




Sequence Pair

π_1: cghdbeaf

π_2: abcdefgh



Redundancy:

π_1: cghdbeaf

π_2: abcdegfh

Problem Statement

$B = \{ B_1, B_2, ... , B_n \}$: set of rectangular blocks

(w_i, h_i) : width and height of block B_i

$P = \{(x_i, y_i): i = 1, ... , n\}$: placement with block B_i at (x_i, y_i)

Assumptions:

1. known net list
2. known chip bounding and I/O pads information
3. known orientations

Goals:

Find a placement P to minimize a preset cost function

Constraint Graph

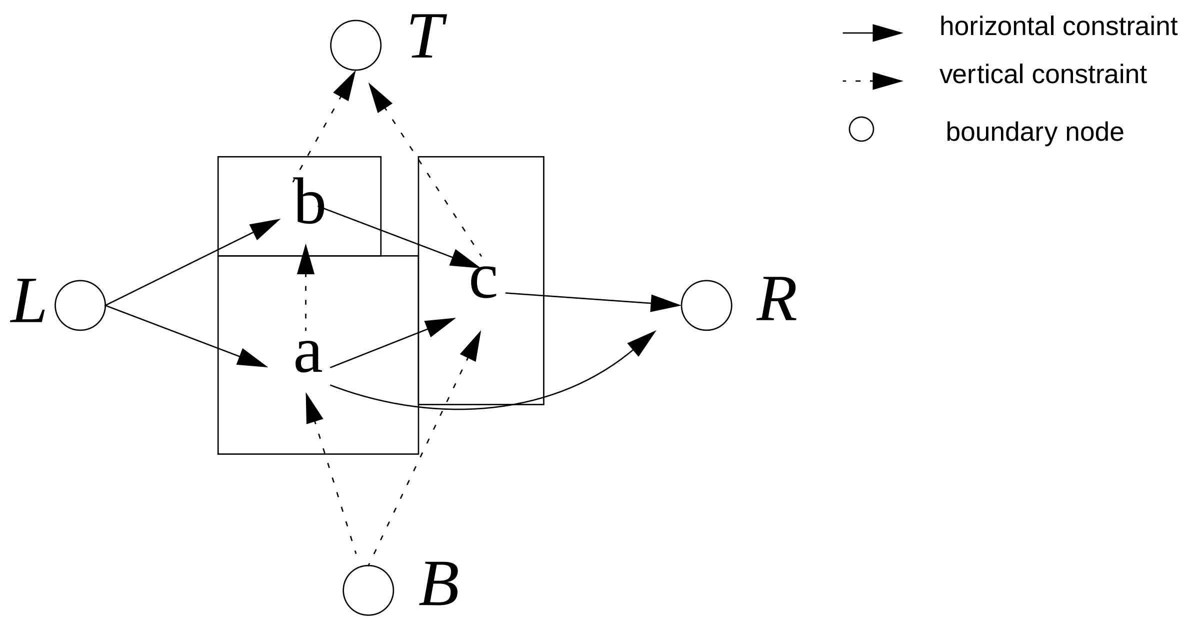


Fig. 1 constraint graph

Admissible Placement

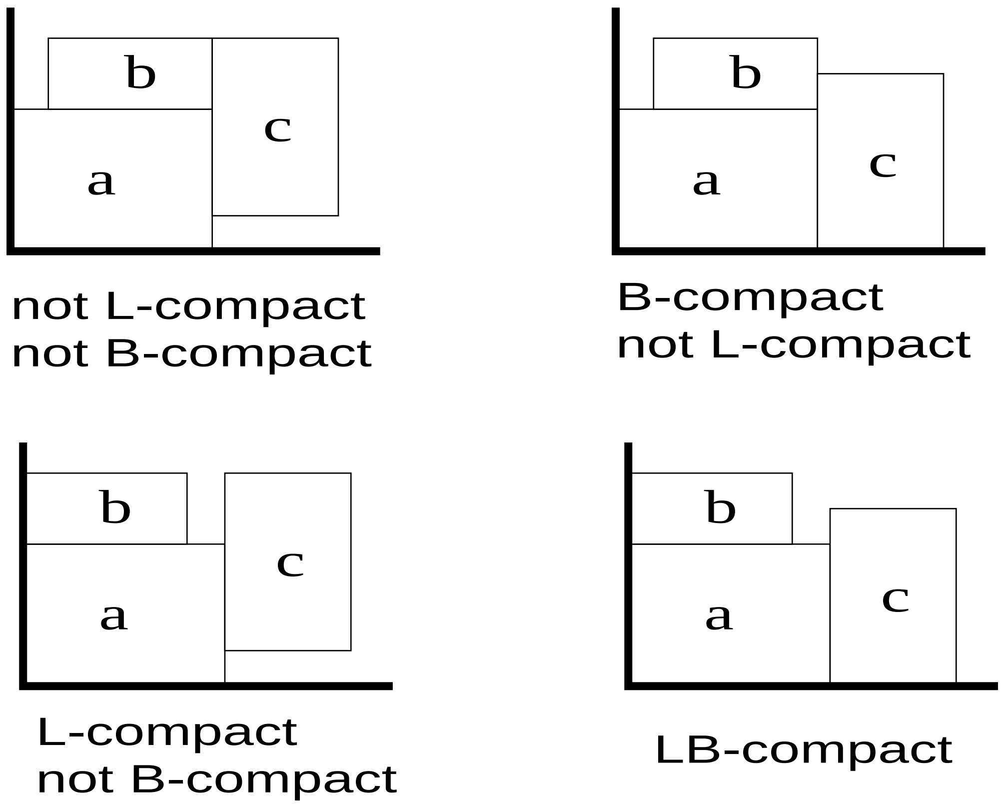
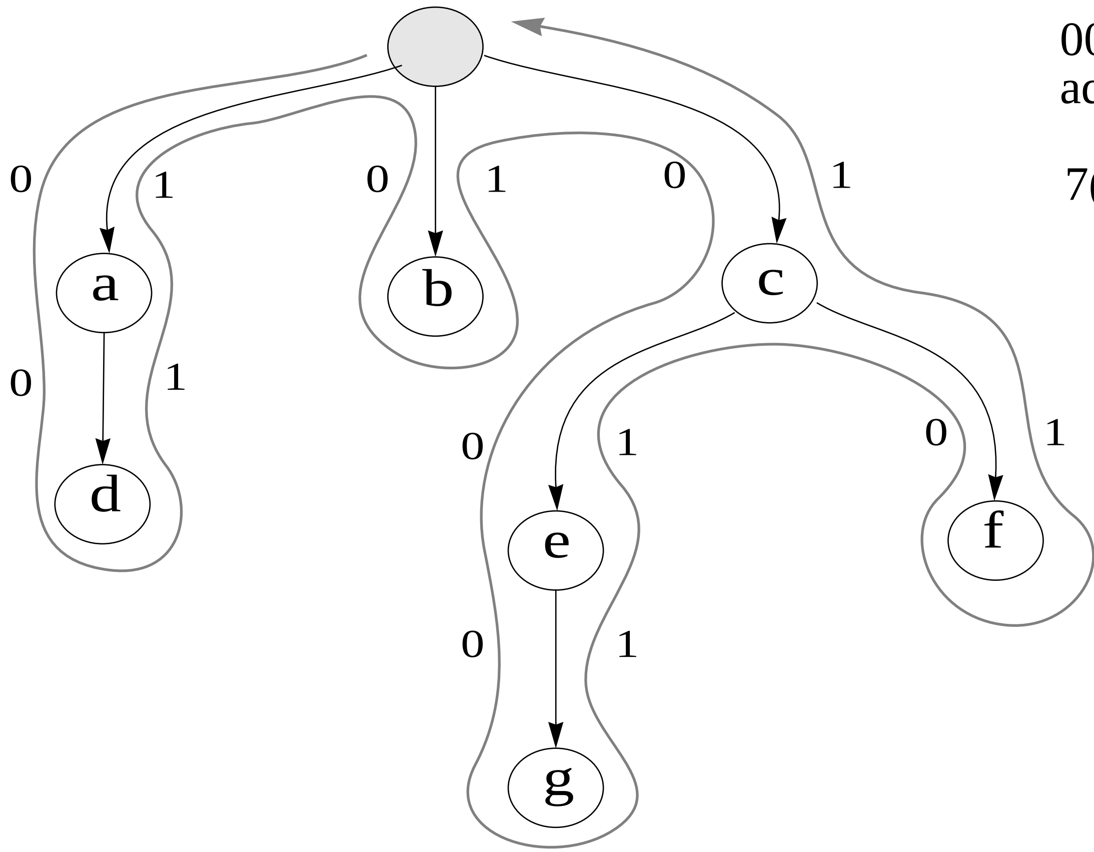


Fig. 2 L-compact and B-compact

Tree Encoding



00110100011011
adbcegh

$7(2+\lceil \lg 7 \rceil)$ bits

Fig. 3 encoding of an 8-node tree

Tree Storage and Its Configurations

Space needed to store a tree:

$n\,(2 + \lceil \lg n \rceil)$-bits

Count of possible configurations:

Unlabeled tree

$$\frac{1}{n}\binom{2n-2}{n-1}$$

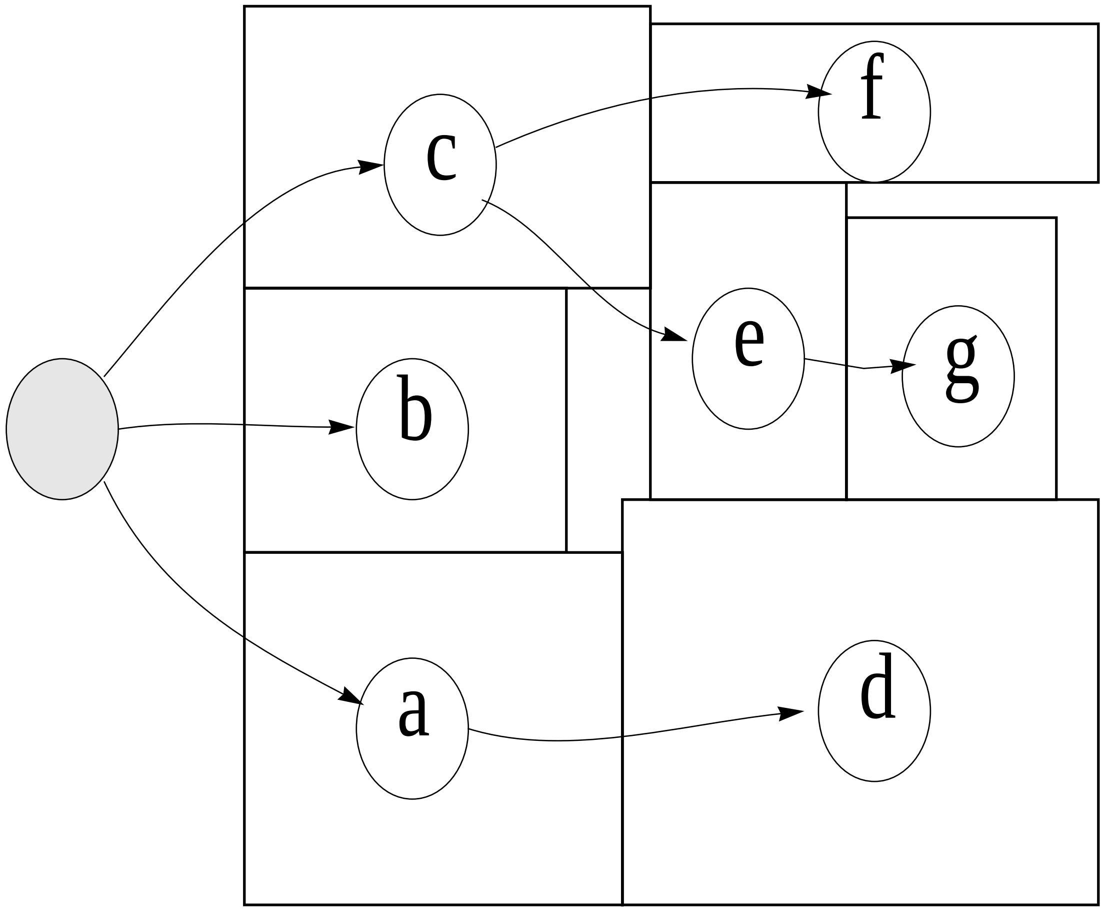
Stirling's approximation

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

After adding label permutation

$O(n!2^{2n-2}/n^{1.5})$

Horizontal O-Tree



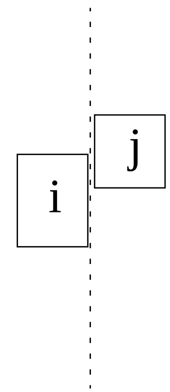
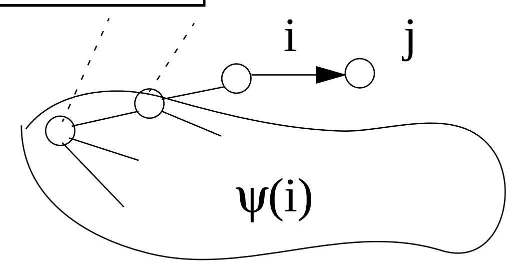


Fig. 4 O-tree and placement



Horizontal O-Tree (definition)

Definition:

The root of the O-tree represents the left boundary of the chip. Thus, we set its x-coordinate $x_{root} = 0$ and its width $w_{root} = 0$. The children are on the right side of their parent with zero separation distance in x-coordinate. Let B_i be the parent of B_j, we have

$$x_j = x_i + w_i$$

The permutation π determines the vertical position of the component when two blocks have proper overlap in their x-coordinate projections. For each block B_i, let $\psi(i)$ be the set of block B_k with its order lower than B_i in permutation π and interval $(x_k, x_k + w_k)$ overlaps interval $(x_i, x_i + w_i)$ by a non-zero length. If $\psi(i)$ is non-empty, we have

$$y_i = max_{k \in \psi(i)}\ y_k + h_k$$

Otherwise,

$$y_i = 0$$

Admissible O-tree

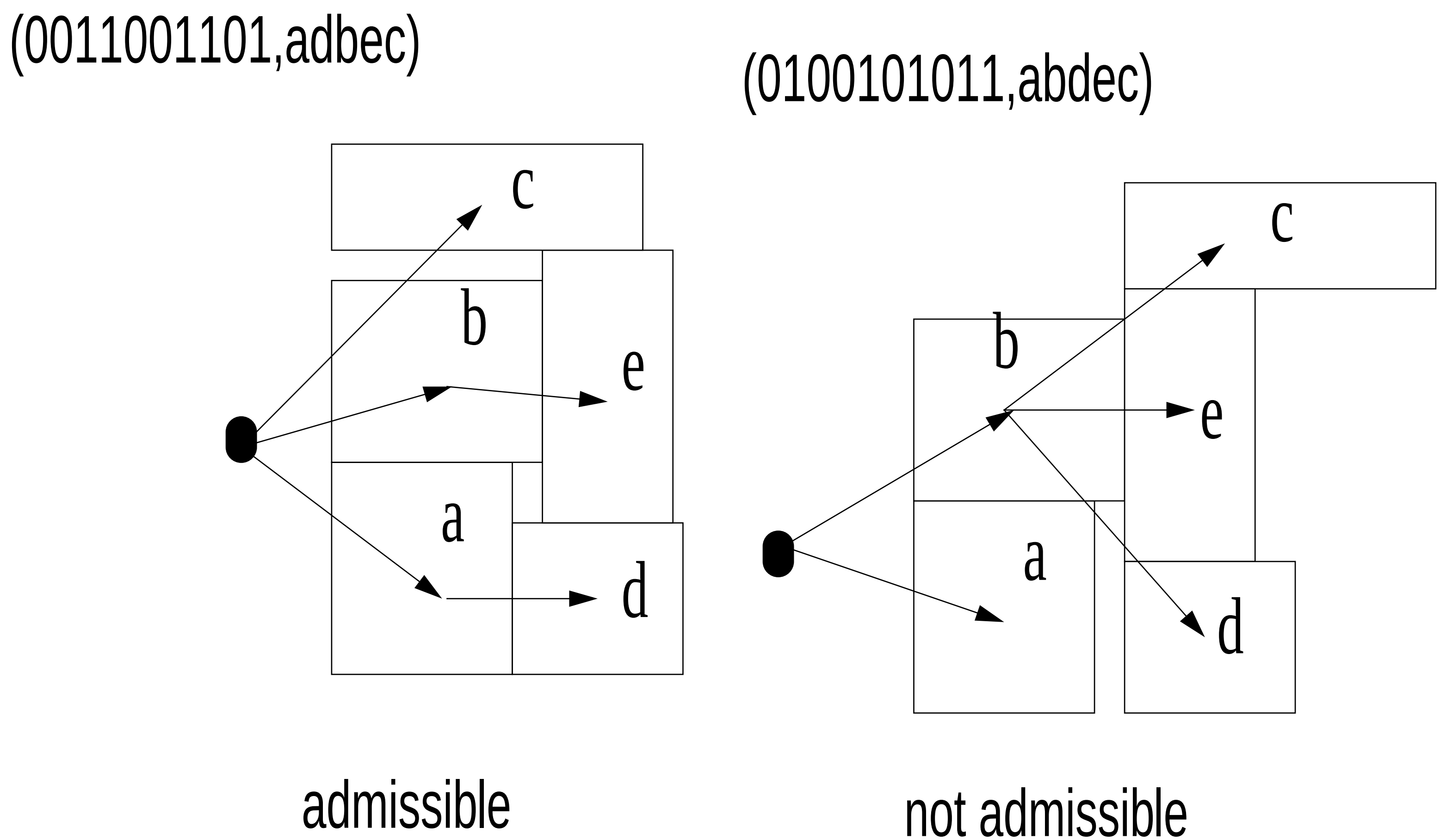


Fig. 5 admissible o-tree

O-Tree to Orthogonal Constraint Graph Algorithm

```
Algorithm OT2OCG
    Input: O-tree(T[0:2n-1], Π[0:n])
    Output: orthogonal constraint graph G=(V,E) and placement x[1:n], y[1:n]
    V = Π + {V_s,V_t};
    perm = 1;
    contour = NULL;
    current_contour = 0;
    for code = 0 to 2n - 1
        if T[code] = 0
            current_block = Π[perm];
            if current_contour = 0
                then x[current_block] = x[current_contour] + w[current_contour];
                else x[curent_block] = 0;
            end if
            y[current_block] = find_max_y(contour, current_block)
            update_constraint_graph(G, contour, current_block)
            update_contour(contour, current_block)
            current_contour = current_block ;
            perm = perm + 1
        else
            current_contour = prev[current_contour];
        end if
    end for
```

Contour Structure

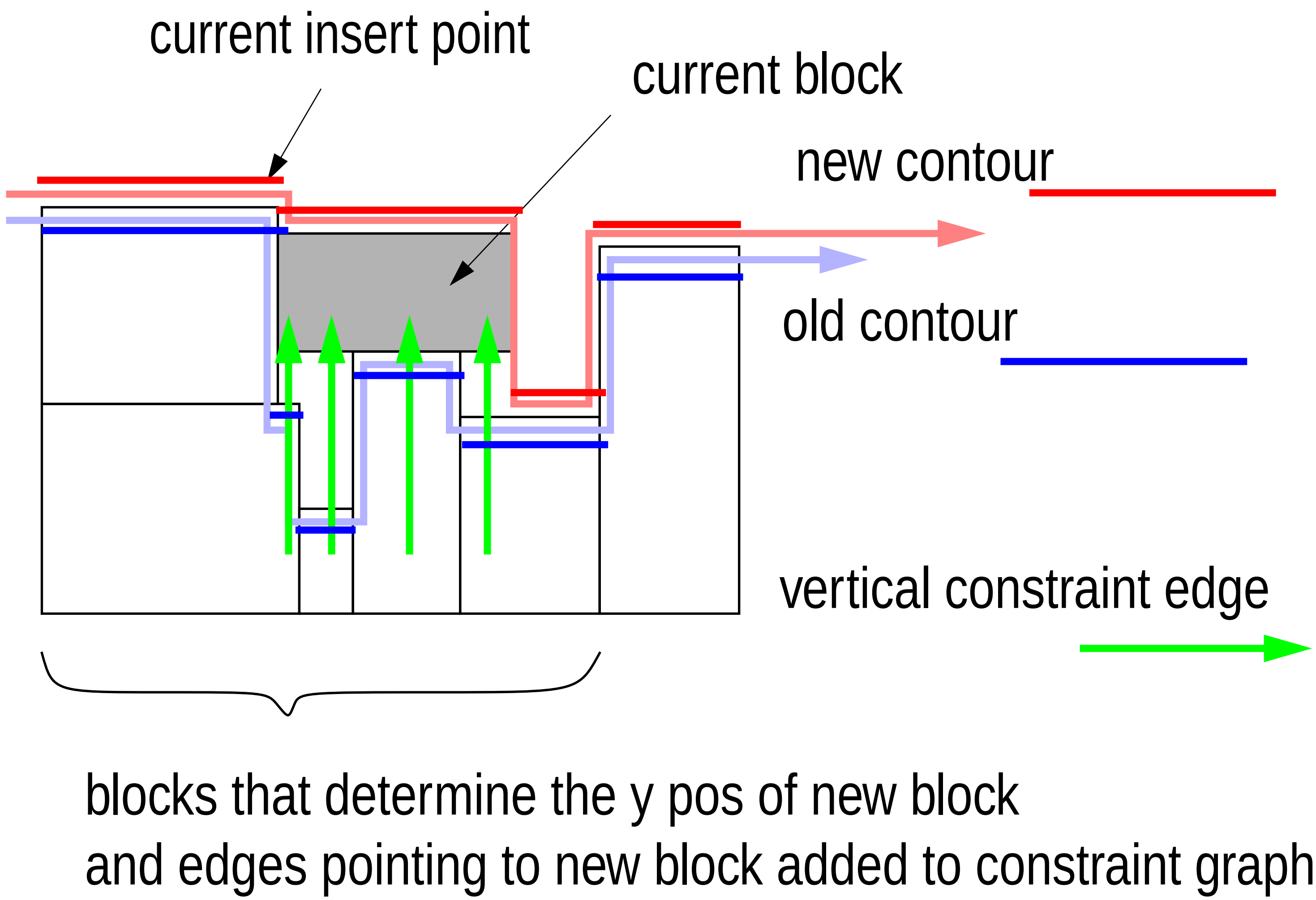
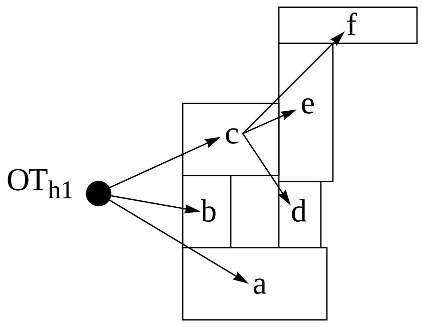


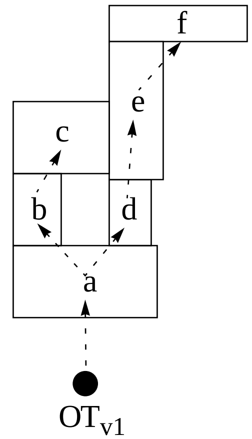
Figure 6 updating constraint graph and contour

Example

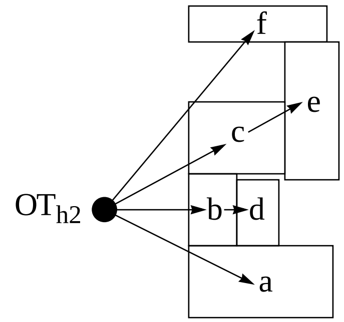
(a) OT_{h1} = (010100101011, abcdef)



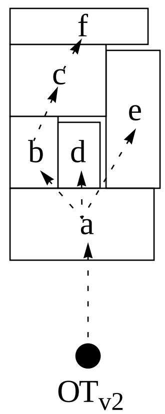
(b) OT_{v1} = (000110001111, abcdef)



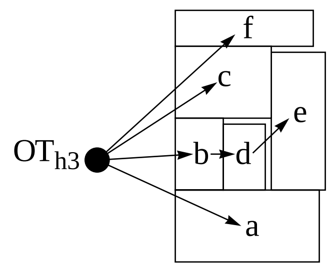
(c) OT_{h2} = (010011001101, abdcef)



(d) OT_{v2} = (000011101011, abcfde)



(e) OT_{h3} = (010001110101, abdecf)



Time Complexity

Theorem:

The time complexity for OT2OCG (O-Tree to Orthogonal Constraint Graph) is linear to the number of blocks.

Proof:

Without loss of generality, assume we can construct a vertical constraint graph from a horizontal O-Tree by OT2OCG. Suppose we have n blocks.

1) The loop executes exactly $2n$ times.

2) In the loop, we perform *find_max_y*, *update_contour*, and *update_constraint_graph* for any block B_i inserted.

3) With maintaining contour structure and *current_contour* pointing to the current starting point in the contour, we can keep tracing the contour until the y-coordinate > $x_i + w_i$.

4) By (3), we need only to pass a limited set of blocks to three operations in (2). The number of blocks is equal to the number of edges inserted in vertical constraint graph.

5) The constraint graph is planar, so the number of edges is less then $3n$-6.

The overall complexity for OT2OCG is linear because we add each vertical edge exactly once.

Constraint Graph to O-Tree Algorithm

Algorithm CG2OT

Input*: constraint graph G=(V,E)*
Output*: O-tree(T[0:2n-1],* Π*[0:n])*
set all mark to false
perm = 0;
code = 0;
DFS traverse on the graph G
n = current node
p = parent[n]
if *not mark[n]* ***and*** *the weight[edge(p,n)] = 0* ***then***
mark[n] = true
Π*[perm++] = 0;*
T[code++] = n;
for *c in children[n]*
traverse(c);
end for
Π*[perm++] = 1;*
end if

Admissible O-Tree Algorithm

Algorithm AOT

Input*: O-tree T*
Output*: Admissible O-tree*
set changed = true
while *changed*
    set changed = false
    set Gy = OT2OCG(T)
    set Ty = CG2OT(Gy)
    set Gx = OT2OCG(Ty)
    set Tx = CG2OT(Gx)
    if *(T is not equal to Tx)* ***then***
        set T = Tx
        set changed = true
    end if
end while
output(T)

Perturbing the O-tree

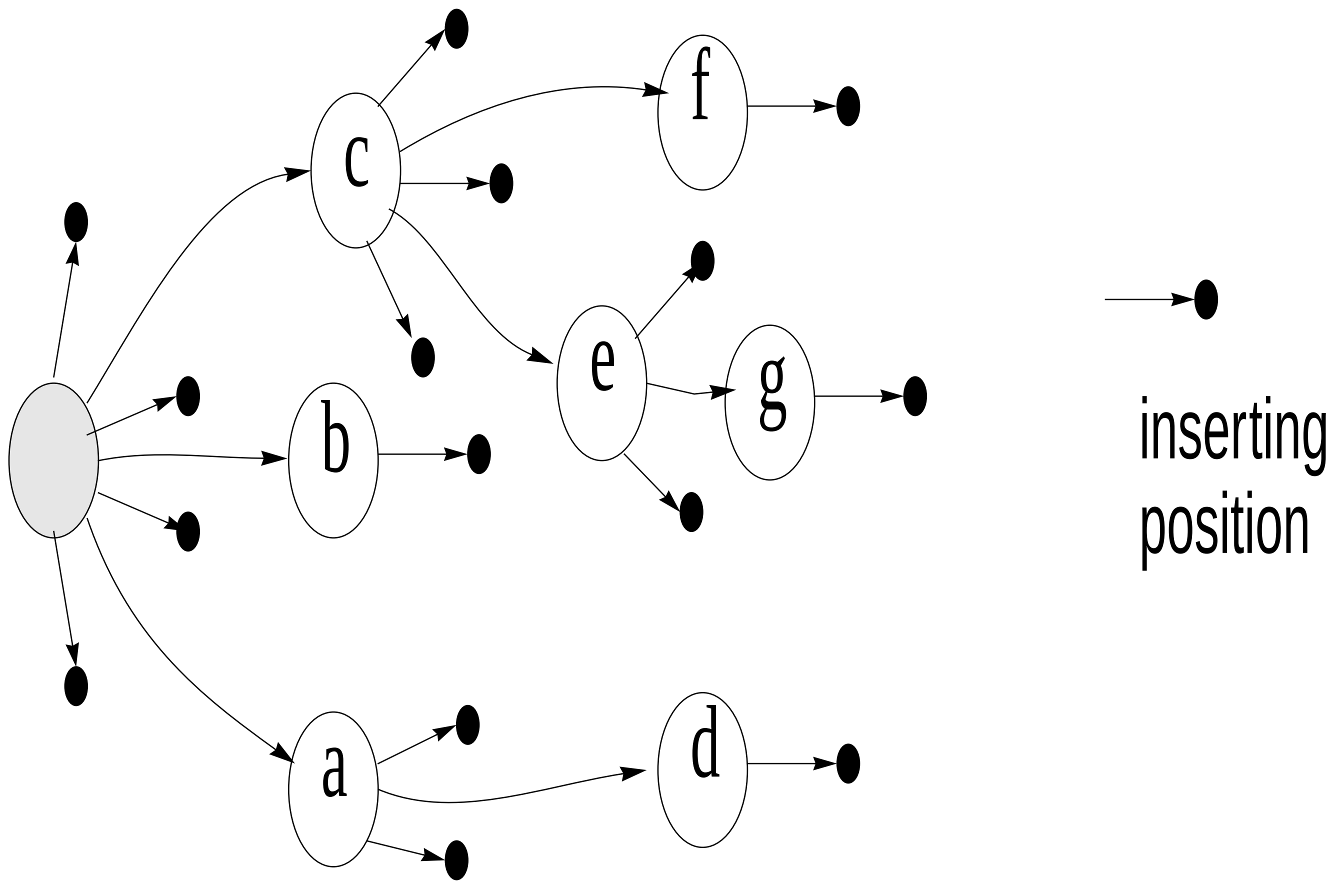


Fig. 7 possible inserting positions as an external node

Deterministic Algorithm

for *each block b*
    set min_cost = infinite
    remove (T, b)
    for *each possible position p of b in T and T's orthogonal*
        set T_1 *= new O-tree and placement for p*
        get admissible T_1 *using AOT*
        set c = cost (T_1*)*
        if *c < min_cost* ***then***
            set min_cost = c
            set min_T = T_1
        end if
    end for
    set T = min_T
end for

Experimental Result

circuit (area/w.l./cpu)	cluster refinement	initial placement	deterministic algorithm
apte	48.4/321/224*	63.3/330/0.14	63.3/330/0.65
xerox	20.3/477/18.8*	25.9/506/0.44	23.8/478/0.99
hp	9.58/185/18.0*	14.3/178/0.26	9.91/167/6.32
ami33	1.21/64/603*	1.69/61.9/2.83	1.34/50.9/24.3
ami49	37.7/764/1860*	54.6/676/11.2	45.5/673/177.5

Area/Wirelengh/CPU comparison
Use original input sequence for initialization
Cost function is mainly by wirelength

* CPU time measured on Sparc-20

Experimental Results (Cont'd)

circuit (min/ avg)	w_1=0, w_2=1		w_1=w_2=0.5		w_1=1, w_2=0		improve over CR (area/wire)
	area	wire	area	wire	area	wire	
apte	48.3/56.9	317/347	47.6/53.2	317/370	47.1/50.6	343/544	3% / 1%
xerox	20.4/24.1	368/426	20.4/22.4	367/447	20.1/21.4	444/702	1% / 23%
hp	9.71/11.2	153/163	9.21/10.5	153/167	9.21/9.97	162/226	4% / 17%
ami33	1.26/1.41	51.5/57.2	1.26/1.34	51.6/59.8	1.25/1.32	61.1/87.4	-3% / 20%
ami49	41.3/49.8	636/734	39.1/42.0	671/777	37.6/39.9	819/1375	0% / 17%

Cost function = $w_1 * area + w_2 * wire$

Min/Avg for 100 runs of randomized initial sequences

Independent comparison for area and wirelength

Various Cost Functions

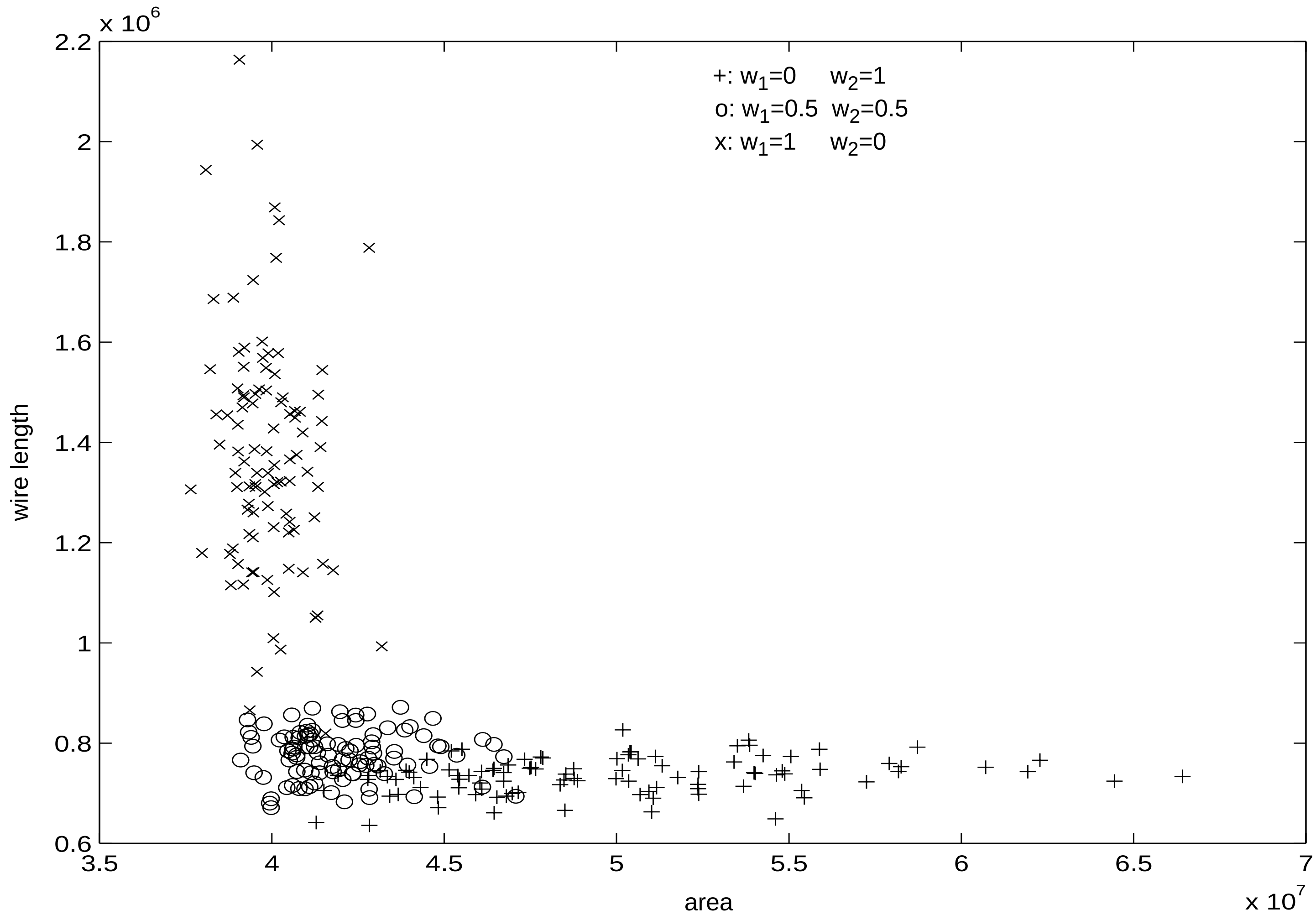
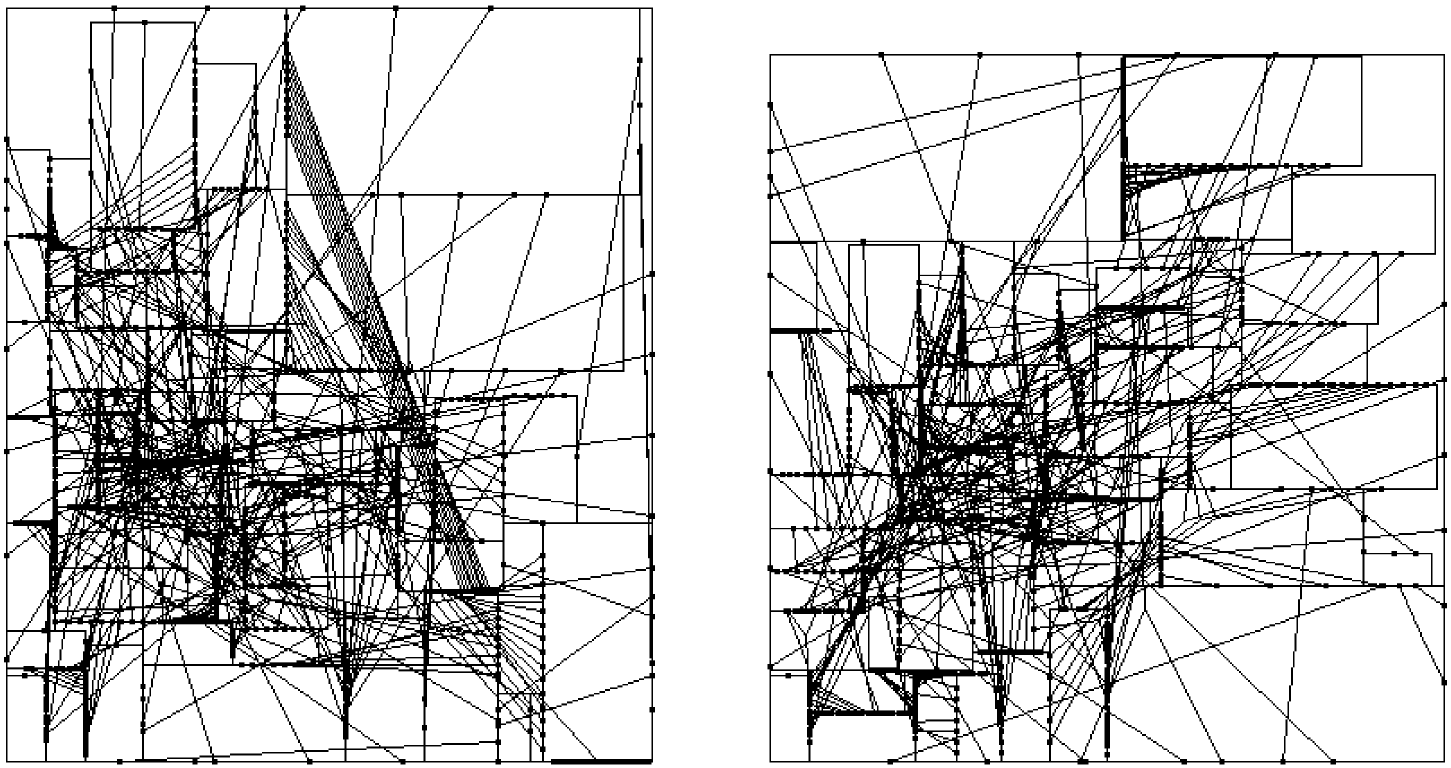


Fig. 8 randomized sequence with different weights (*ami49*)



(a) before improvment
area = 40.8 (5.92 x 6.89)
wire length = 810

(b) after improvment
area = 39.9 (6.17 x 6.47)
wire length = 680

Fig. 9 placements before and after deterministic improvement for *ami49*

Conclusions

- Floorplan and placement problem becomes more important for larger VLSI circuit design (IP blocks, SoC, ...)

- Interconnection is the key issue for the performance in Very Deep Sub Micron (VDSM) technology

- O-tree provides a very effective and efficient representation of building block placement in 2D plane

- The experimental results using O-tree show that much better interconnection (in term of wire length) is achieved in less CPU time