Fast Legalization for Standard Cell Placement with Simultaneous Wirelength and Displacement Minimization

Tsung-Yi Ho and Sheng-Hung Liu

Department of Computer Science and Information Engineering
National Cheng Kung University

Abstract. Legalization is one of the most critical steps in modern placement designs. Since several objectives like wirelength, routability, or temperature are already optimized in global placement stage, the objective of legalization is not only to align the cells overlap-free to the rows, but also to preserve the solution of global placement, i.e., the displacement of cells needs to be minimized. However, minimizing displacement only is not enough for current timing-driven SoC designs. Blind displacement minimization may increase the half-perimeter wirelength (HPWL) of nets significantly that degrades the chip performance. In this paper, we propose a fast legalization algorithm for standard cell placement with simultaneous wirelength and displacement minimization. The main contributions of our work are: (1) a fast row selection technique by using k-medoid clustering approach; (2) an exact linear wirelength model to minimize both wirelength and total displacement; (3) a constant time approach to determine the median in trial placement stage. Compared with the state-of-the-art legalization algorithms, experimental results show that our legalizer acquires much better achievement in terms of HPWL, total and maximum displacements, and running time on legalized NTUplace3 global placement results on both ISPD 2005 and 2006 placement contest benchmarks.

Keywords: Displacement, Legalization, Placement, Wirelength

1 Introduction

The conventional standard cell placement includes three stages: global placement, legalization, and detailed placement. Typically, global placement generates an initial placement with minimum total half-perimeter wirelength (HPWL) and tries to optimize some objectives such as routability, timing, temperature, and etc. This results in few cell overlap and the cells are not aligned to the rows. There has been several researches in the area of global placement in the past few years [3, 5, 4, 11, 16, 18]. After global placement stage, legalization targets (1) to remove overlaps between cell instances, (2) to put all instances on the rows in the core area, and (3) to minimize the displacements of instances between the initial placement and the legalized one. That is, legalization tries to make the placement legal and to preserve the result from the global placement stage. After legalization, detailed placement is used to improve the solution quality of the legal placement. Legalization with only displacement minimization may significantly induce the solutions with more HPWL than global placement stage. For example, if cell a is to be legalized in Figure 1 (a), blind legalization with only displacement minimization may increase the HPWL as shown in Figure 1 (b). However, by taking HPWL into consideration, we can legalize cell a with both displacement and HPWL minimization.

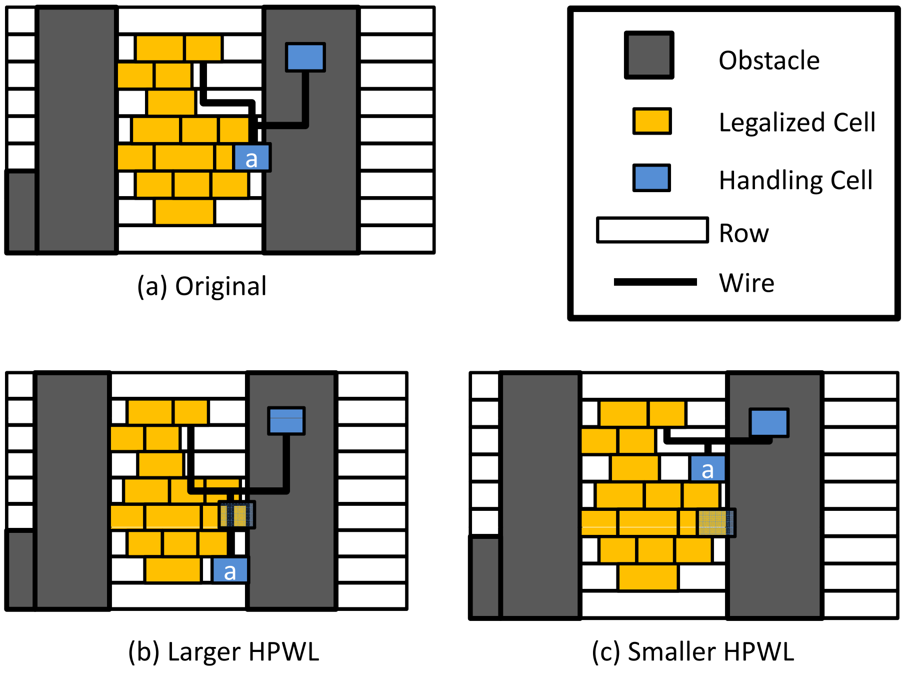


Fig. 1. (a) Original placement before legalizing cell a. (b) Placement with larger HPWL after legalizing cell a. (c) Placement with smaller HPWL after legalizing cell a.

1.1 Previous Work

Existing legalization techniques include network flow [2, 6], ripple cell movement[8], dynamic programming [1], simulated annealing[17], single row optimization[10], diffusion-based method [15], computational geometry [13], packing [7], and quadratic programming [18]. Domino [6] partitions cells in subcells (all having the same height and width) and rows in places, and assigns them by using a min-cost-max-flow approach. Similar to Domino, [2] assigns sets of modules to row regions. Mongrel [8] uses a greedy heuristic to move cells from overflowed bins to under capacity bins in a ripple fashion based on total wirelength gain. Fractional Cut [1] assigned cells to rows by dynamic programming and the cells of each row are packed from left to right. Sarrafzadeh [17] uses simulated annealing for legalization. The authors of [10] assigned cells to the rows by cell juggling and the cells of each row are placed by finding a shortest path in a graph. Diffusion-based placement migration is presented in [15] to remove cell overlap incrementally. In [13], cells are spread and aligned to rows by computational geometry techniques. Sarrafzadeh [17] uses a cell shifting technique to reduce the maximum bin density. HPWL-driven legalizers only target on HPWL minimization within a row by dynamic programming approach [9, 10].

Tetris is a fast greedy heuristic which is widely used in industry [7]. Tetris sorts the cells first, and legalizes one cell at a time then. The legalization of one cell is done by moving the cell over the rows, and within a row by moving the cell over free sites. This movement is done until the best free site is found. Once a cell has been legalized, it will not be moved anymore. This results in a high total cell displacement during legalization. Recently, Spindler et al. present a similar approach, called Abacus [19], that it places a cell from row to row until the location with the smallest displacement is found. However, as a cell is placed into a row, the legalized cells in that row are re-placed by dynamic programming technique to minimize the total displacement. Both Tetris and Abacus scan rows with minimum cost vertically. Tetris scans all rows (see Figure 2 (a)). Abacus sorts the rows by the y-position. A lower bound of the cost is computed by assuming cells are only moved vertically. Abacus moves the cell to the best row first, and then tries to move the cell to the rows with the lower bound not exceeding the minimal cost of an already found legal position (see Figure 2 (b)). In this paper, our legalizer constructs the binary index tree for the rows via clustering techniques. By traversing the binary index tree, our legalizer only scans some candidate rows then the running time is significantly reduced (see Figure 2 (c)).

1.2 Our Contribution

The main contributions of our work are: (1) a fast row selection technique by using k-medoid clustering approach; (2) an exact linear wirelength model to minimize both total displacement and HPWL; (3) a constant time approach to determine the median in trial placement stage. The remainder of this paper is organized as follows. Section 2 details our algorithm. Section 3 shows the experimental results. Finally, concluding remarks are given in Section 4.

2 Algorithm

Figure 3 shows the overview of our legalization algorithm. First, a binary index tree for rows is constructed for fast row scanning (line 1). After that, cells are sorted according to the sum of their x-position and the half of their width (line

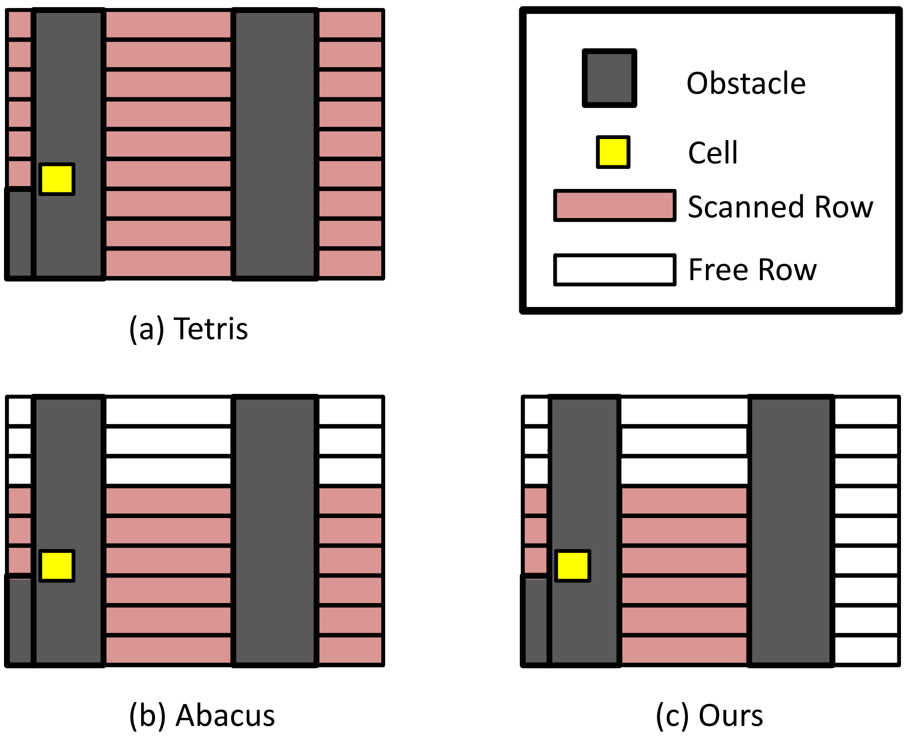


Fig. 2. (a) Tetris scans all rows. (b) Abacus scans the rows by y-position. (c) Our legalizer only scans the rows when they are necessary.

2). Then, cells are legalized one by one according to the sorting order (line 3-8). "DFS" (line 5) adopt depth-first search manner to search the best row for each cell; i.e., the cost (the sum of HPWL and displacement of the cell) of the row is minimum by using the binary index tree for row scanning (line 1). Then the cell is inserted to the best row for HPWL and displacement minimization (line 6).

2.1 Binary Index Tree

Fast row scanning can be achieved by searching a binary index tree constructed by top-down row clustering technique. In this paper, we adopt k-medoid [12] clustering algorithm as the main method for clustering rows. Typically, k-medoid algorithm clusters items by using their similarity. To cluster rows, we define the similarity between rows as the distance between each row. If the distance between each row is far, the similarity is small, and vice versa. Initially, all rows are in a cluster, called **root cluster**. Then all rows are clustered into two clusters by using k-medoid algorithm (k=2) where the centers are central rows of row clusters. Iteration of clustering terminates until the number of rows in every

Algorithm 1: Overall Legalization Algorithm

```
1   Construct a binary index tree for rows;
2   Sort cells according to (x-position + width / 2);
3   foreach cell do
4       $row_{best}$ = null;
5       $row_{best}$ = DFS(root_cluster,cell);
6       Insert cell to $row_{best}$;
7   end
```

Fig. 3. Overall algorithm of our legalizer.

cluster is less than 3. Finally, we construct a binary index tree for row clusters. The relation between rows and binary index tree is shown in Figure 4.

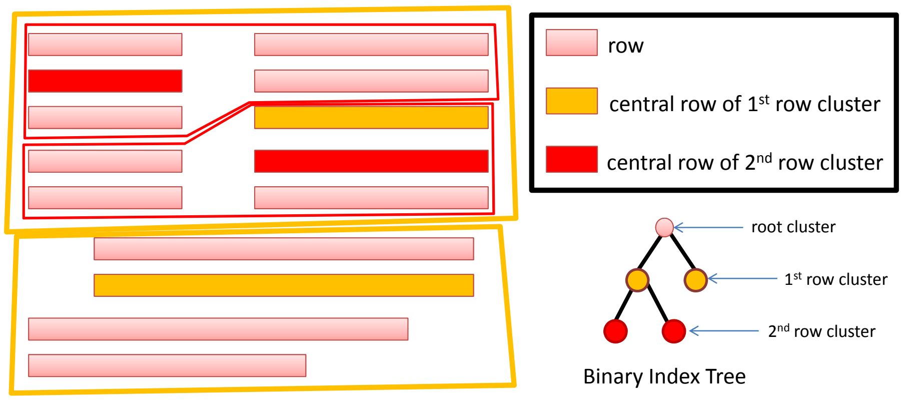


Fig. 4. Top-down clustering constructs a binary index tree for rows.

2.2 Row Scanning

Before placing cells into the rows, legalizers perform row scanning then do trial placement for cells. Different from previous legalizers that scan all rows, our legalizer can find the best row efficiently by searching on the binary index tree. As mentioned earlier, the nodes in the binary index tree represent row clusters. The edge cost in the binary index tree is the distance from a cell to the row cluster. To compute the distance from a cell to the row cluster, we need to compute the distance between rows first. $D_{min}(row_i, row_j)$ and $D_{max}(row_i, row_j)$ are two kinds of the distance from row_i to row_j. We adopt $D_{min}(row_i, row_j)$ as the distance for clustering rows by using k-medoid algorithm and $D_{max}(row_i, row_j)$ as the distance used for computing the edge cost of the binary index tree. If row_i overlaps row_j horizontally, $D_{min}(row_i, row_j)$ is the vertical distance from row_i to row_j (see Figure 5 (a)). If row_i does not overlap row_j horizontally, $D_{min}(row_i, row_j)$ is the sum of vertical and horizontal distance from row_i to row_j (see Figure 5 (b)). $D_{max}(row_i, row_j)$ is the sum of the vertical distance and maximum horizontal distance from the left of row_i to the left (see Figure 5 (c)) of row_j or from the right of row_i to the right of row_j (see Figure 5 (d)). Moreover, the computation of distance from cells to row clusters is the same with distance between rows.

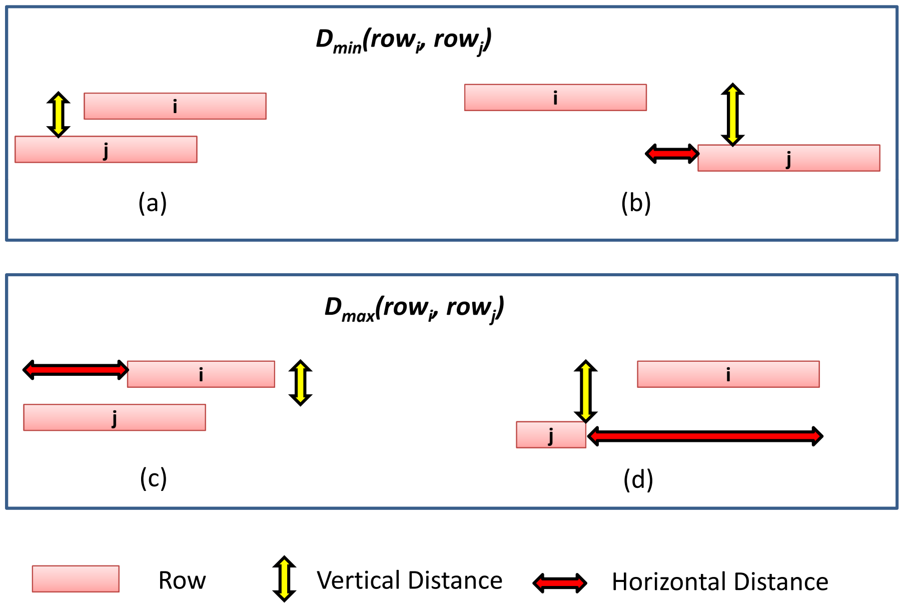


Fig. 5. (a) $D_{min}(row_i, row_j)$ with horizontal overlaps. (b) $D_{min}(row_i, row_j)$ without horizontal overlaps. (c) $D_{max}(row_i, row_j)$ with horizontal overlaps. (d) $D_{max}(row_i, row_j)$ without horizontal overlaps.

After the distance between rows is computed, we defined radius of a row cluster is the farthest distance between its central row to other rows. According to triangle inequality, the distance from a cell to the row cluster, i.e. $D_{min}(cell, row\_cluster)$, can be derived as shown in Figure 6.

A depth-first search (DFS) approach is used for finding the best row. Algorithm 2 describes the function DFS. DFS starts from root cluster. The cell is always moved into the nearer cluster for trial placement first (line 5-18). The cell

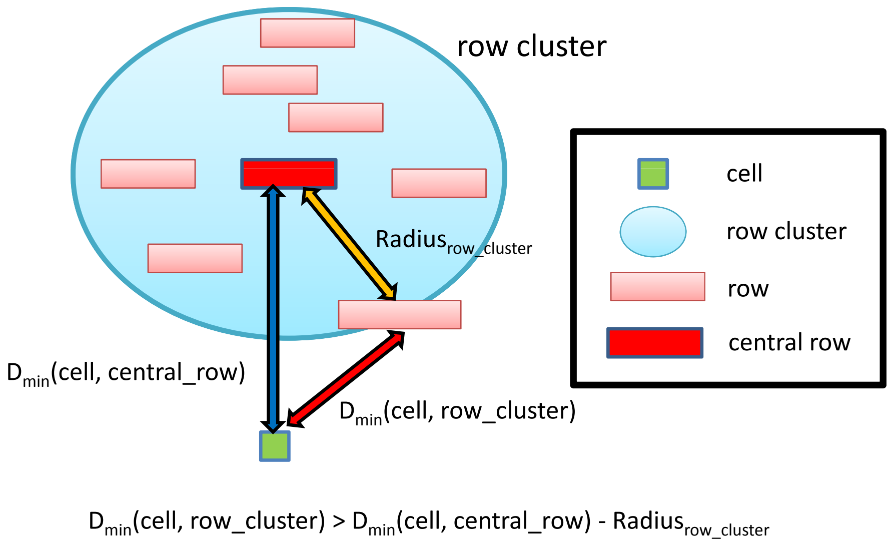


Fig. 6. Illustration of the minimum distance computation between a *cell* and a *row_cluster*.

is not moved to cluster if $D_{min}(cell, row\_cluster)$ is larger than the current best displacement. The cost of the row is the sum of both HPWL and displacement of the cell after the cell is moved to the row (line 21). After DFS, the best row can be found quickly, then cells will be inserted into it. For simultaneous wirelength and displacement minimization, the cost of row for placing a cell depends on the displacement and the HPWL of it. Since a cell may be connected by multiple nets, the cost by simply adding the displacement and the HPWL of a cell may not be precise. Furthermore, we not only target to minimize total displacement but also to minimize maximum displacement, we consider both factors by a combined cost function. The cost function is defined as follows:

$$Cost(i) = HPWL + N_i(\alpha \cdot (DP_S) + \beta \cdot (DP_P)) \tag{1}$$

where N_i represents the number of the net belongs to cell i. DP_S is the cell displacement between original position and legalized position. DP_P is the total increased displacement caused by inserting one cell into row. α, β are user-specified parameters. An example of cost computation is shown in Figure 8.

2.3 Obstacle-Aware Cell Ordering

Modern chip designs often consist of many preplaced blocks, such as analog blocks, memory blocks, and/or I/O buffers, which are fixed in the chip and cannot overlap with other blocks. These preplaced blocks, i.e., obstacles, impose more constraints on the legalization problem. A legalization algorithm without considering obstacles may significantly induce increasing cell displacement or inferior solutions. Different from all previous works that insert cell according to its x-position in the global placement, we insert cell according to its central position (i.e., x-position of cells + cell width/2). As illustrated in Figure 9, an obstacle lies in the middle of the row shred the row into two subrows. If cells are inserted by x-position, the larger cell 1 will be inserted first but cause larger displacement for cell 2 and 3 since they are inserted to another subrow that across the obstacle (See Figure 9 (a)). It is not worth because only cell 1 is inserted into the left subrow with smaller displacement and cell 2 and 3 are squeezed to right subrows with larger displacement. If cells are inserted by their central position, most of the smaller cells that are capable of being inserted to fragmentary subrows can reduce the total displacement significantly (See Figure 9 (b)). Moreover, it is capable of handling the relative order problem.

2.4 Cell Insertion

Most of previous works use simple quadratic wirelength model to measure cell displacement. The drawback lies in the preciseness since the square of larger cell displacement will have dramatically difference. Different from Abacus that uses quadratic model, we adopt an exact linear wirelength model for measuring cell displacement. Cells of one row are abutting in the legal placement and form a cluster. Assume the row has c clusters and $cluster_c$ has n cells which are sorted by their global x-position + width/2. To minimize the cell displacement, the objective function is defined as follows:

$$min \sum_{i=1}^{n} |x_i^c - x_i'^c| \tag{2}$$

where $x_i'^c$ and x_i^c are initial and legalized position of cluster c. The constraint (2) assures that there is no overlap between the cells and is defined as follows:

$$x_i^c - x_{i-1}^c \geq w_{i-1}^c \qquad i = 2, ..., n \tag{3}$$

where w_i^c is cell width. However, solving linear programs with "$\geq$" constraints is time consuming in general. If the same solution of the linear program is found by "=" constraints, then the linear program is solved quite fast by solving one linear equation. The situation that "=" constraints are sufficient is given if all cells of one row are abutting in the legal placement and form a cluster. There, two cells are "abutting" if there is no free space between them in the legal placement. With only "=" constraints, (2) is transformed to:

$$x_i^c = x_1^c + \sum_{k=1}^{i-1} w_k^c \qquad i = 2, ..., n \tag{4}$$

By (3), we can transform (1) into (4) such that the optimal value only depends on variable x_1^c.

$$min \sum_{i=1}^{n} |x_1^c - \underbrace{[x_i^c - \sum_{k=1}^{i-1} w_k^c]}_{d_i^c}| \tag{5}$$

where d_i^c is the candidate position of cluster c for cell i and all d_i^c form a set D_c. The optimal x_1^c can be derived by finding the median m of all d_i^c.

2.5 Find Median

As shown in Figure 10, a row has several clusters. The positions of these clusters are determined by X_c. If the clusters are overlapped, they will be merged and form a new cluster.

The intuitive method for finding the median is to sort n elements in D_c while the time complexity of sorting for trial placement and insertion are both $O(nlogn)$. An efficient method for finding the median is to apply the deterministic partitioning algorithm from quicksort while the time complexity for trial placement and insertion are both $O(n)$. In this paper, we adopt red-black trees for finding the median while the time complexity for trial placement and insertion are both $O(logn)$. Moreover, red-black trees can be used to record already sorted D_c since D_c will not be changed after cells are placed. Thus, we can derive the set of X_c as follows:

$$X_c = \{x_c(i)\} \qquad i = 2, ..., n \tag{6}$$

where $x_c(i)$ corresponds to the already sorted elements in D_c.

Since cells are not really placed on rows in trial placement stage, unnecessary movement of cells can be pruned in this stage. By exploring all possible movements of clusters, we found only the rightmost cluster (i.e., $cluster_c$) will move to the left and others will stay still. By this observation, we have 3 lemmas for our trial placement stage. By these 3 lemmas, the time complexity of our trial placement is only O(1) since we only need to compare 3 possible positions then we can find the best position for all clusters. The comparison of time complexity in listed in Table 1.

Lemma 1 *If the number of the cells in $cluster_c$ is even, the cell i is placed to the right of $cluster_c$ and the position of $cluster_c$ will not be changed.*

Proof: Let c is the number of clusters in a row. N_c is the number of the cells in $cluster_c$. $W_c$is the width of $cluster_c$. The candidate position for cell i is $\overline{x_i}$ which is the difference of W_c and the original x position of cell i. If cell i and $cluster_c$ overlap horizontally, $\overline{x_i}$ is less than the original x position of $cluster_c$.

As shown in Figure 11, assume N_c is an even number. The original x position of $cluster_c$ is the median of X_c (i.e., $\frac{N_c}{2}$). Therefore, there are $\frac{N_c}{2} - 1$ elements that are smaller than the original x position of $cluster_c$, and $\frac{N_c}{2}$ elements that are larger than the original x position of $cluster_c$. After cell i and $cluster_c$ are merged, $\overline{x_i}$ is added into X_c. Since $\overline{x_i}$ is smaller than the x position of cluster c, the number of X_c which smaller than the original x position of cluster c is $\frac{N_c}{2}$. Therefore, the median of X_c does not changed. The new x position of $cluster_c$ is the original x position of $cluster_c$. $\square$

Lemma 2 *If the number of the cells in $cluster_c$ is odd, the cell i is placed to the right of $cluster_c$ and the position of $cluster_c$ will shift to the left.*

Proof: As shown in Figure 12, assume N_c is a odd number and the original x position of $cluster_c$ is the median of X_c (i.e., $\frac{N_c+1}{2}$). Therefore, the number of X_c which smaller than the original x position of $cluster_c$ is $\frac{N_c+1}{2} - 1$. The number of X_c which larger than the original x position of $cluster_c$ is $\frac{N_c+1}{2} - 1$. After cell i and $cluster_c$ are merged, $\overline{x_i}$ is added into X_c. Since $\overline{x_i}$ is smaller than the original x position of $cluster_c$, the number of X_c which smaller than the original x position of $cluster_c$ is $\frac{N_c+1}{2}$. Therefore, the median of X_c will be changed. The new x position of $cluster_c$ is $\frac{N_c+1}{2}$ of new X_c instead of the original x position of $cluster_c$. In other words, new x position of $cluster_c$ is the maximum one between ($\frac{N_c+1}{2} - 1$) of original X_c and $\overline{x_i}$). $\square$

By Lemma 2, $cluster_c$ may overlap with $cluster_{c-1}$ when $cluster_c$ shifts to the left. In this situation, $cluster_{c-1}$ and $cluster_c$ are merged and then form a new and bigger $cluster_{c-1}$ and the number of the clusters of the row becomes $c-1$. Under this circumstance, we further observe that although $cluster_c$ shifts to the left, the positions of other $c-1$ clusters will not be influenced by $cluster_c$. The details are described in Lemma 3.

Lemma 3 *$cluster_c$ has no influence on other $n-1$ clusters.*

Proof: As shown in Figure 13, X'_c is the difference between X_c and W_{c-1}. If a new cell added to the row, it will cause $cluster_c$ overlapped with $cluster_{c-1}$, then $cluster_{c-1}$ and $cluster_c$ will be merged together. After $cluster_{c-1}$ and $cluster_c$ are merged, the position of the new $cluster_{c-1}$ is found by the median of new X_{c-1} (i.e., original X_{c-1} merged with X'_c). The median of original X'_c (i.e., $X'_c(N_c/2+1)$) is larger than the median of X_{c-1} (i.e., $X_{c-1}((N_{c-1}+1)/2)$), because $cluster_{c-1}$ and original $cluster_c$ are not overlapped. Once a new cell added to the row, it will cause $cluster_c$ overlaps $cluster_{c-1}$, the median of new X'_c (i.e., $X'_c(N_c/2)$) will be smaller than the median of X_{c-1} (i.e., $X_{c-1}((N_{c-1}+1)/2)$). Therefore, after $cluster_{c-1}$ and $cluster_c$ are merged, the median of new X_{c-1} is unchanged (i.e., $X_{c-1}(N_{c-1}+1)/2)$. It means that the x position of new $cluster_{c-1}$ is unchanged, and the position of cell is the sum of $(X_{c-1}(N_{c-1}+1)/2)$, W_{c-1}, and L_c.

In trial place step, the position of the cell on the row can be calculated by finding the maximum of the following three numbers: x', the sum of the $\frac{N_c+1}{2}$-1) of X_c and W_c, and the sum of the x position of $cluster_{c-1}$ (i.e., $(X_{c-1}((N_{c-1} + 1)/2)))$, W_{c-1}, and L_c. $\square$

3 Experimental Results

We have implemented our legalizer in the C++ language on a 2-GHz 64-bit Linux machine with 16GB memory. For fair comparison, we evaluate NTUplace3 [5], Tetris [7], Abacus [19], and Kahng [9] on the same platform. Furthermore, to verify the efficiency and effectiveness of our legalizer, we perform experiments on two benchmark suites, such as ISPD 2005 and 2006 placement contest benchmarks [22]. The global placement results are obtained from NTUplace3 [5]. Extensive experiments demonstrate that in terms of HPWL, total and maximum displacements, and running time, we acquire much better achievement than all the state-of-the-art algorithms such as Tetris [7], Abacus [19], and Kahng [9] in any aspect.

3.1 Benchmarks

The cell numbers of the ISPD 2005 benchmarks range from 210K to 2169K, the fixed macro numbers range from 543 to 23084, and the row numbers range from 890 to 2694. The cell numbers of the ISPD 2006 benchmarks range from 330K to 2507K, the fixed macro numbers range from 337 to 26582, and the row numbers range from 930 to 4182. The benchmark information is listed in Table 2.

3.2 HPWL

In the first experiment, we evaluate the HPWL of our legalizer on the both benchmarks. On legalized NTUplace3 [5] global placements on ISPD 2005 placement contest benchmarks, the HPWL is 1.92X, 1.19X and 1.92X of our legalizer compared to Tetris [7], Abacus [19], and Kahng [9], respectively. It should be noted that Kahng [9] only focusing on HPWL minimization. Moreover, we also compare the HPWL after our legalizer with the original HPWL of NTUplace3 global placement results [5], we can further reduce HPWL by 18%. The details of HPWL experiments on ISPD 2005 benchmarks are listed in Table 3.

On legalized NTUplace3 [5] global placements on ISPD 2006 placement contest benchmarks, the HPWL is 1.44X, 1.25X and 1.43X of our legalizer compared to Tetris [7], Abacus [19], and Kahng [9], respectively. Compare the HPWL after our legalizer with the original HPWL of NTUplace3 global placement results [5], we can further reduce HPWL by 19%. The details of HPWL experiments on ISPD 2006 benchmarks are listed in Table 4.

3.3 Displacement

In the second experiment, we report the total and maximum displacement on both ISPD benchmarks. On legalized NTUplace3 global placements on ISPD

2005 placement contest benchmarks, the total displacement is 25.76X, 1.09X, and 27.16X, the maximum displacement is 1.88X, 1.04X, and 2.40X of our legalizer compared to Tetris [7], Abacus [19], and Kahng [9], respectively. The details of displacement experiments on ISPD 2005 benchmarks are listed in Table 5.

On legalized NTUplace3 global placements on ISPD 2006 placement contest benchmarks, the total displacement is 5.27X, 1.15X, and 5.43X, the maximum displacement is 1.51X, 1.35X, and 1.51X of our legalizer compared to Tetris [7], Abacus [19], and Kahng [9], respectively. The details of displacement experiments on ISPD 2006 benchmarks are listed in Table 6. It should be noted that Abacus [19] achieved the best published results in displacement but our legalizer still get better results due to the exact linear wirelength model.

3.4 Running time

In the third experiment, we evaluate the running time on both ISPD benchmarks. On legalized NTUplace3 global placements on ISPD 2005 placement contest benchmarks, the running time is 5.79X, 36.18X, and 5.94X of our legalizer compared to Tetris [7], Abacus [19], and Kahng [9], respectively. The details of running time experiments on ISPD 2005 benchmarks are listed in Table 7.

On legalized NTUplace3 global placements on ISPD 2006 placement contest benchmarks, the running time is 6.28X, 26.15X, and 6.49X of our legalizer compared to Tetris [7], Abacus [19], and Kahng [9], respectively. The details of running time experiments on ISPD 2006 benchmarks are listed in Table 8. It should be noted that Tetris [7] achieved the fastest published results but our legalizer still get better results due to the row indexing and fast trial placement techniques.

Figure 14 (a) shows the global placement result of “adaptec1” obtained from NTUplace3. To further reveal the difference of Tetris, Abacus, Kahng, and ours, we plot the movement pictures during legalization in Figure 14 (b), (c), (d), and (e).

4 Conclusions

In this paper, we proposed a fast legalization algorithm with simultaneous displacement and HPWL minimization. The main contributions of our work are: (1) a fast row selection technique by using k-medoid clustering approach; (2) an exact linear wirelength model to minimize both wirelength and total displacement; (3) a constant time approach to determine the median in trial placement stage. Compared with the state-of-the-art algorithms, experimental results have shown that our legalizer obtains very high-quality results on legalized NTUplace3 global placements on both ISPD 2005 and 2006 placement contest benchmarks.

References

1. Ameya Agnihorti, Mehmet Can Yildiz, Ateen Khatkhate, Ajita Mathur, Satoshi Ono, and Patrick H. Madden, "Fractional cut: Improved recursive bisection placement," *Proc. ICCAD*, pp. 307-310, 2003.
2. Ulrich Brenner and Jens Vygen, "Legalizing a placement with minimum total movement," *IEEE TCAD*, 23(12):1597-1613, Dec.2004.
3. Ulrich Brenner and Markus Struzyna, "Faster and better global placement by a new transportation algorithm," *Proc. DAC*, pp. 591-596, 2005.
4. Tony Chan, Jason Cong, Kento Sze, and Min Xie, "mPL6: Enhanced multilevel mixed-size placement," *Proc. ISPD*, pp. 212-214, 2006.
5. T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "NTUplace3: A high-quality mixed-size analytical placer considering preplaced blocks and density constraints," *Proc. ICCAD*, pp. 187-192, 2006.
6. Konrad Doll, Frank M. Johannes, and Kurt J. Antreich, "Iterative placement improvement by network flow methods," *IEEE TCAD*, 13(10):1189-1200, Oct. 1994.
7. Dwight Hill, "Method and system for high speed detailed placement of cells within integrated circuit designs," *U.S. Patent 6370673*, Apr. 2002.
8. Sung-Woo Hur and John Lillis, "Mongrel: Hybrid techniques for standard cell placement," *Proc. ICCAD*, pp. 165-170, 2000.
9. Andrew B. Kahng, Paul Tucker and Alex Zelikovsky, "Optimization of Linear Placements for Wirelength Minimization with Free Sites," *Proc. ASP-DAC'99*, pp. 241-244, 1999.
10. Andrew B. Kahng, Igor L. Markov, and Sherief Reda, "On legalization of row-based placements," *Proc. GLS-VLSI*, pp. 214-219, 2004.
11. Andrew B. Kahng and Qinke Wang, "Implementation and extensibility of an analytic placer," *IEEE TCAD*, 24(05):734-747, May 2005.
12. Carlos B. Lucasius, Adrie D. Dane, and Gerrit Kateman, "On k-medoid clustering of large data sets with the aid of a genetic algorithm: background, feasibility and comparison," *Analytica Chimica Acta*, vol. 282, no3, pp. 647-669, 1993.
13. Tao Luo, Haoxing Ren, Charles J. Alpert, and David Z. Pan, "Computational geometry based placement migration," *Proc. DAC*, pp. 41-47, 2007.
14. Min Pan, Natarajan Viswanathan, and Chris Chu, "An efficient and effective detailed placement algorithm," *Proc. ICCAD*, pp. 48-55, 2005.
15. Haoxing Ren, David Z. Pan, Charles J. Alpert, and Paul Villarrubia, "Diffusion-based placement migration," *Proc. DAC*, pp. 515- 520, 2005.
16. Jarrod A. Roy, David A. Papa, Saurabh N. Adya, Haward H. Chan, Aaron N. Ng, James F. Lu, and Igor L. Markov, "Capo: Robust and scalable open-source min-cut floorplacer," *Proc. ISPD*, pp. 224- 226, 2005.
17. M. Sarrafzadeh and M. Wang, "NRG: Global and detailed placement," *Proc. ICCAD*, pp. 532-537, 1997.
18. Peter Spindler and Frank M. Johannes, "Fast and robust quadratic placement based on an accurate linear net model," *Proc. ICCAD*, pp. 179-186, 2006.
19. Peter Spindler, Ulf Schlichtmann, and Frank M. Johannes, "Abacus: Fast legalization of standard cell circuits with minimal movement," *Proc. ISPD*, pp. 47-53, 2008.
20. T. C. Wang and D. F. Wong, "Optimal floorplan area optimization," *IEEE TCAD*, 11 (1992), 992-1002.

21. D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," *Proc. DAC*, pp. 101-107, 1986.
22. http://www.ispd.cc/

```
Algorithm 2: Function DFS
Function DFS(cell, row_cluster)
if row_cluster is not a leaf then
    $D_R$ = $D_{min}$(row_cluster.RightChild, cell);
    $D_L$ = $D_{min}$(row_cluster.LeftChild, cell);
    if ($D_R < D_L$) then
        if $row_{best}$ is null or $displacement_{best} > D_R$ then
            DFS(cell, row_cluster.RightChild);
            if $row_{best}$ is null or $displacement_{best} > D_L$ then
                DFS(cell, row_cluster.LeftChild);
            end
        end
    else
        if $row_{best}$ is null or $displacement_{best} > D_L$ then
            DFS(cell, row_cluster.LeftChild);
            if $row_{best}$ is null or $displacement_{best} > D_R$ then
                DFS(cell, row_cluster.RightChild);
            end
        end
    end
else
    foreach row in row_cluster do
        $displacement_{candidate}$ = TrialPlace(row, cell);
        if $displacement_{candidate} < displacement_{best}$ then
            $displacement_{best}$ = $displacement_{candidate}$;
        end
    end
end
```

Fig. 7. The algorithm of function DFS.

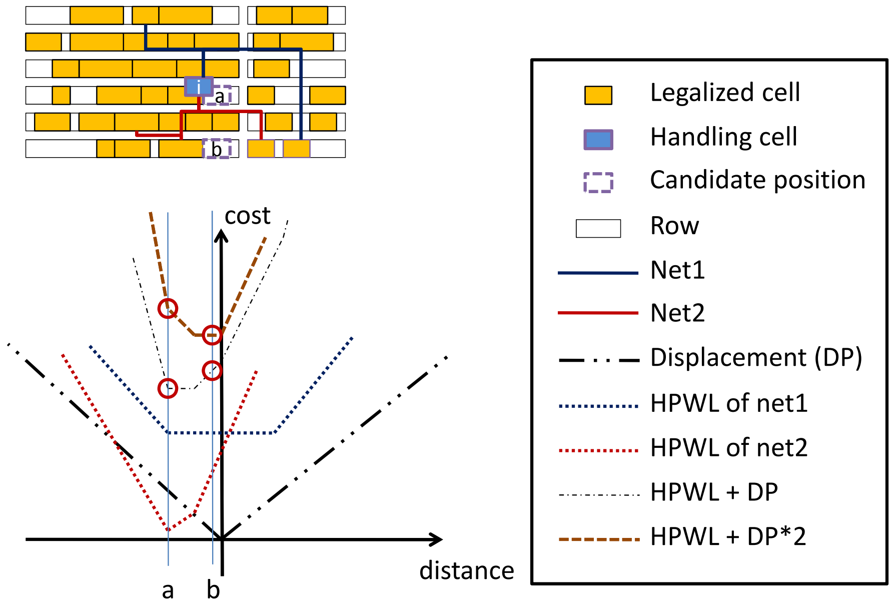


Fig. 8. Illustration of the cost(i) to find the best row.

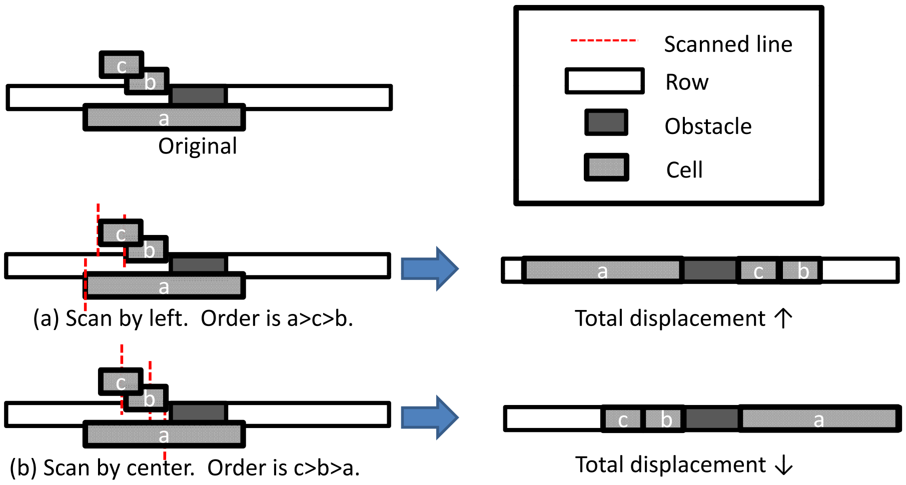


Fig. 9. Ordering for obstacle-aware cell insertion.

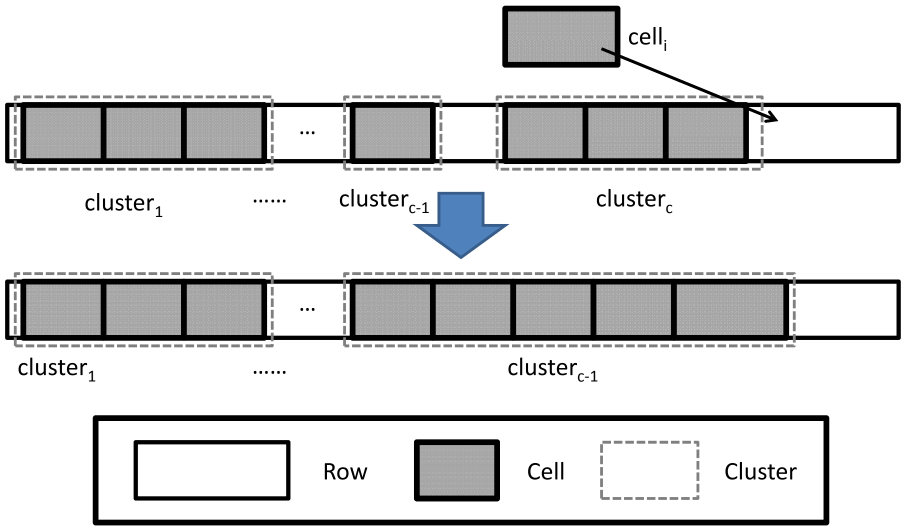
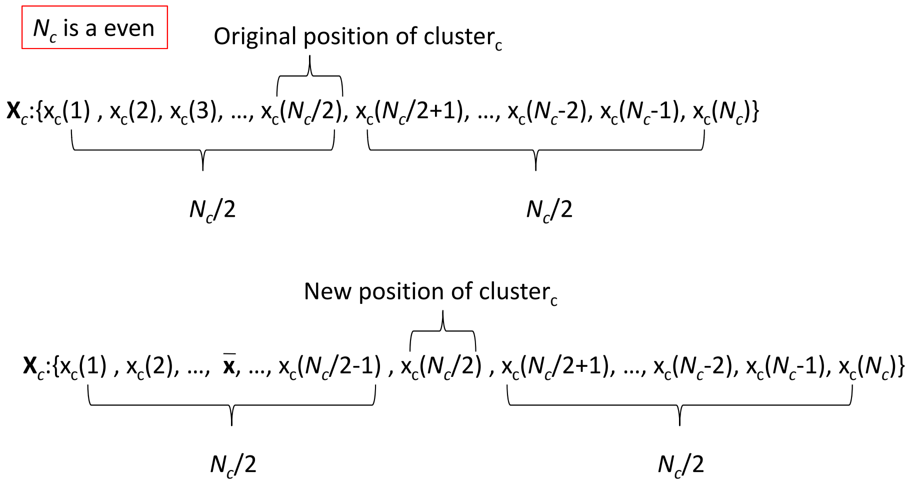


Fig. 10. Illustration of cluster merging.

Table 1: Time complexity for trial placement and cell insertion

	Trial	Insert
Sorting algorithm	O(nlogn)	O(nlogn)
SELECT algorithm	O(n)	O(n)
Red-Black tree	O(logn)	O(logn)
Ours	O(1)	O(logn)



N_c is a even
Original position of cluster_c
$\mathbf{X}_c$:{$x_c(1)$, $x_c(2)$, $x_c(3)$, …, $x_c(N_c/2)$, $x_c(N_c/2+1)$, …, $x_c(N_c-2)$, $x_c(N_c-1)$, $x_c(N_c)$}
$N_c/2$
$N_c/2$
New position of cluster_c
$\mathbf{X}_c$:{$x_c(1)$, $x_c(2)$, …, $\bar{\mathbf{x}}$, …, $x_c(N_c/2-1)$, $x_c(N_c/2)$, $x_c(N_c/2+1)$, …, $x_c(N_c-2)$, $x_c(N_c-1)$, $x_c(N_c)$}
$N_c/2$
$N_c/2$

Fig. 11. N_c is an even number.

$\boxed{N_c \text{ is an odd}}$

$$\mathbf{X}_c:\{\underbrace{x_c(1)\,, x_c(2), x_c(3), \dots, x_c((N_c-1)/2)}_{(N_c-1)/2}, \overbrace{x_c((N_c+1)/2)}^{\text{Original position of cluster}_c}, \underbrace{x_c((N_c+1)/2+1), \dots, x_c(N_c-2), x_c(N_c-1), x_c(N_c)}_{(N_c-1)/2}\}$$

$$\mathbf{X}_c:\{\underbrace{x_c(1)\,, x_c(2), \dots, \bar{\mathbf{x}}, \dots, x_c\overbrace{(N_c/2-1)}^{\text{New position of cluster}_c}}_{(N_c+1)/\ 2}\,, \underbrace{x_c((N_c+1)/2), x_c((N_c+1)/2+1), \dots, x_c(N_c-2), x_c(N_c-1), x_c(N_c)}_{(N_c+1)/\ 2}\}$$

or

$$\mathbf{X}_c:\{\underbrace{x_c(1)\,, x_c(2), \dots, x_c(N_c/2-1), \overbrace{\bar{\mathbf{x}}}^{\text{New position of cluster}_c}}_{(N_c+1)/\ 2}, \underbrace{x_c((N_c+1)/2), x_c((N_c+1)/2+1), \dots, x_c(N_c-2), x_c(N_c-1), x_c(N_c)}_{(N_c+1)/\ 2}\}$$

Fig. 12. N_c is an odd number.

$\boxed{\boldsymbol{N_{c\text{-}1}} \textbf{ is an odd}}$

$$\mathbf{X}_{c\text{-}1}:\{\boxed{\underbrace{x_{c\text{-}1}(1)\,, x_{c\text{-}1}(2), \dots, x_{c\text{-}1}((N_{c\text{-}1}-1)/2)}_{(N_{c\text{-}1}-1)/2}}, \overbrace{x_{c\text{-}1}((N_{c\text{-}1}+1)/2)}^{\text{Original position of cluster}_{c\text{-}1}}, \boxed{\underbrace{x_{c\text{-}1}((N_c+1)/2+1), \dots, x_{c\text{-}1}(N_{c\text{-}1}-1), x_{c\text{-}1}(N_{c\text{-}1})}_{(N_{c\text{-}1}-1)/2}}\}$$

$\boxed{\boldsymbol{N_c} \textbf{ is an even}}$

$$\mathbf{X'}_c:\{\boxed{\underbrace{x'_c(1)\,, x'_c(2), x'_c(3), \dots, x'_c(N_c/2)}_{N_c/\ 2}}, \boxed{\underbrace{x'_c(N_c/2+1), \dots, x'_c(N_c-2), x'_c(N_c-1), x'_c(N_c)}_{N_c/\ 2}}\}$$

$$\because\ x'_c(N_c/2) < x_{c\text{-}1}((N_{c\text{-}1}+1)/2) < x'_c(N_c/2+1)$$

$$\therefore\ \mathbf{X}_{c\text{-}1}\ \textbf{merge}\ \mathbf{X'}_c:\{:\{\underbrace{\boxed{\dots}}_{(N_{c\text{-}1}-1+N_c)/\ 2}, x_{c\text{-}1}((N_{c\text{-}1}+1)/2), \underbrace{\boxed{\dots}}_{(N_{c\text{-}1}-1+N_c)/\ 2}\}$$

$$\therefore\ x_{c\text{-}1}((N_{c\text{-}1}+1)/2) \text{ is also the } \textbf{median}$$

Fig. 13. Merge process of $cluster_{c-1}$.

Table 2: Statistics of two benchmark suites of the ISPD placement contest

ISPD 2005 Benchmarks				ISPD 2006 Benchmarks			
Name	# Cells	# Fixed Macros	# Rows	Name	# Cells	# Fixed Macros	# Rows
adaptec1	210904	543	890	adaptec5	843128	646	1944
adaptec2	254457	566	1170	newblue1	330073	337	930
adaptec3	450927	723	1944	newblue2	441516	1277	1925
adaptec4	494716	1329	1944	newblue4	646139	3422	1524
bigblue1	277604	560	890	newblue5	1233058	4881	2130
bigblue2	534782	23084	1566	newblue6	1255039	6889	2316
bigblue3	1093034	1293	2316	newblue7	2507954	26582	3258
bigblue4	2169183	8170	2694				

Table 3: Wirelength comparison between NTUplace3, Tetris, Abacus, Kahng, and ours on ISPD 2005 benchmarks

ISPD 2005 Benchmarks	NTUplace3 [5]	Tetris [7]	Abacus [19]	Kahng [9]	Ours
Name	HPWL	HPWL	HPWL	HPWL	HPWL
adaptec1	9.56E+07	1.20E+08	9.50E+07	1.19E+08	8.00E+07
adaptec2	9.98E+07	1.39E+08	1.01E+08	1.39E+08	8.80E+07
adaptec3	2.43E+08	2.61E+08	2.40E+08	2.61E+08	2.00E+08
adaptec4	2.14E+08	2.53E+08	2.09E+08	2.52E+08	1.80E+08
bigblue1	1.10E+08	5.00E+08	1.10E+08	4.98E+08	9.40E+07
bigblue2	1.65E+08	1.79E+08	1.64E+08	1.78E+08	1.30E+08
bigblue3	4.15E+08	5.86E+08	4.67E+08	5.85E+08	3.60E+08
bigblue4	8.97E+08	1.04E+09	9.09E+08	1.03E+09	7.60E+08
AVG.	1.18	1.92	1.19	1.92	1

Table 4: Wirelength comparison between NTUplace3, Tetris, Abacus, Kahng, and ours on ISPD 2006 benchmarks

ISPD 2006 Benchmarks	NTUplace3 [5]	Tetris [7]	Abacus [19]	Kahng [9]	Ours
Name	HPWL	HPWL	HPWL	HPWL	HPWL
adaptec5	4.01E+08	4.95E+08	4.07E+08	4.94E+08	3.14E+08
newblue1	6.42E+07	1.03E+08	6.40E+07	1.03E+08	5.74E+07
newblue2	2.16E+08	2.46E+08	2.20E+08	2.45E+08	2.02E+08
newblue4	2.89E+08	3.20E+08	2.84E+08	3.19E+08	2.32E+08
newblue5	5.06E+08	5.34E+08	5.01E+08	5.32E+08	4.23E+08
newblue6	5.09E+08	6.15E+08	6.18E+08	6.15E+08	4.25E+08
newblue7	1.13E+09	1.30E+09	1.25E+09	1.30E+09	9.24E+08
AVG.	1.19	1.44	1.25	1.43	1

Table 5: Displacement comparison between NTUplace3, Tetris, Abacus, Kahng, and ours on ISPD 2005 benchmarks

ISPD 2005 Benchmarks	Tetris [7]		Abacus [19]		Kahng [9]		Ours	
Name	TOTAL	MAX	TOTAL	MAX	TOTAL	MAX	TOTAL	MAX
adaptec1	7.91E+07	2490	5.89E+06	1247	8.10E+07	2490	6.00E+06	1211
adaptec2	1.31E+08	3432	1.90E+07	3485	1.32E+08	3432	1.90E+07	3239
adaptec3	1.22E+08	4581	5.11E+07	5419	1.23E+08	4577	4.80E+07	5733
adaptec4	1.56E+08	2197	2.74E+07	1702	1.58E+08	3161	2.60E+07	1736
bigblue1	7.38E+08	9319	4.63E+06	1304	7.90E+08	9319	4.80E+06	1342
bigblue2	6.25E+07	1143	1.31E+07	871	6.77E+07	3757	9.20E+06	718
bigblue3	8.01E+08	5886	4.20E+08	6729	8.02E+08	5886	3.50E+08	6756
bigblue4	8.21E+08	2115	6.56E+07	5701	8.36E+08	2115	6.00E+07	5203
AVG.	25.76	1.88	1.09	1.04	27.16	2.40	1	1

Table 6: Displacement comparison between NTUplace3, Tetris, Abacus, Kahng, and ours on ISPD 2006 benchmarks

ISPD 2006 Benchmarks	Tetris [7]		Abacus [19]		Kahng [9]		Ours	
Name	TOTAL	MAX	TOTAL	MAX	TOTAL	MAX	TOTAL	MAX
adaptec5	4.65E+08	5117	9.12E+07	3617	4.68E+08	5117	8.79E+07	4920
newblue1	1.44E+08	3639	1.37E+07	2990	1.45E+08	3639	1.31E+07	1459
newblue2	2.17E+08	6162	1.05E+08	4535	2.18E+08	6162	7.37E+07	2131
newblue4	2.17E+08	2214	4.24E+07	2808	2.30E+08	2214	4.16E+07	2327
newblue5	1.55E+08	4030	3.52E+07	1527	1.61E+08	4030	3.31E+07	1777
newblue6	8.97E+08	2609	2.68E+08	7485	9.07E+08	2609	2.23E+08	7514
newblue7	1.22E+09	4248	3.99E+08	10527	1.34E+09	4248	3.20E+08	7246
AVG.	5.27	1.51	1.15	1.35	5.43	1.51	1.00	1.00

Table 7: Runtime comparison between NTUplace3, Tetris, Abacus, Kahng, and ours on ISPD 2005 benchmarks

ISPD 2005 Benchmarks	Tetris [7]	Abacus [19]	Kahng [9]	Ours
Name	TIME (s)	TIME (s)	TIME (s)	TIME (s)
adaptec1	32	101	34	16
adaptec2	44	321	47	11
adaptec3	163	647	169	24
adaptec4	213	682	218	47
bigblue1	38	110	40	18
bigblue2	360	601	365	72
bigblue3	1453	15033	1465	82
bigblue4	1272	4456	1304	304
AVG.	5.79	36.18	5.94	1

Table 8: Runtime comparison between NTUplace3, Tetris, Abacus, Kahng, and ours on ISPD 2006 benchmarks

ISPD 2006 Benchmarks	Tetris [7]	Abacus [19]	Kahng [9]	Ours
Name	TIME (s)	TIME (s)	TIME (s)	TIME (s)
adaptec5	304	2225	312	51
newblue1	32	123	35	24
newblue2	389	1032	395	53
newblue4	274	1327	281	49
newblue5	598	1869	612	116
newblue6	839	4917	854	188
newblue7	5163	16844	5245	365
AVG.	6.28	26.15	6.40	1

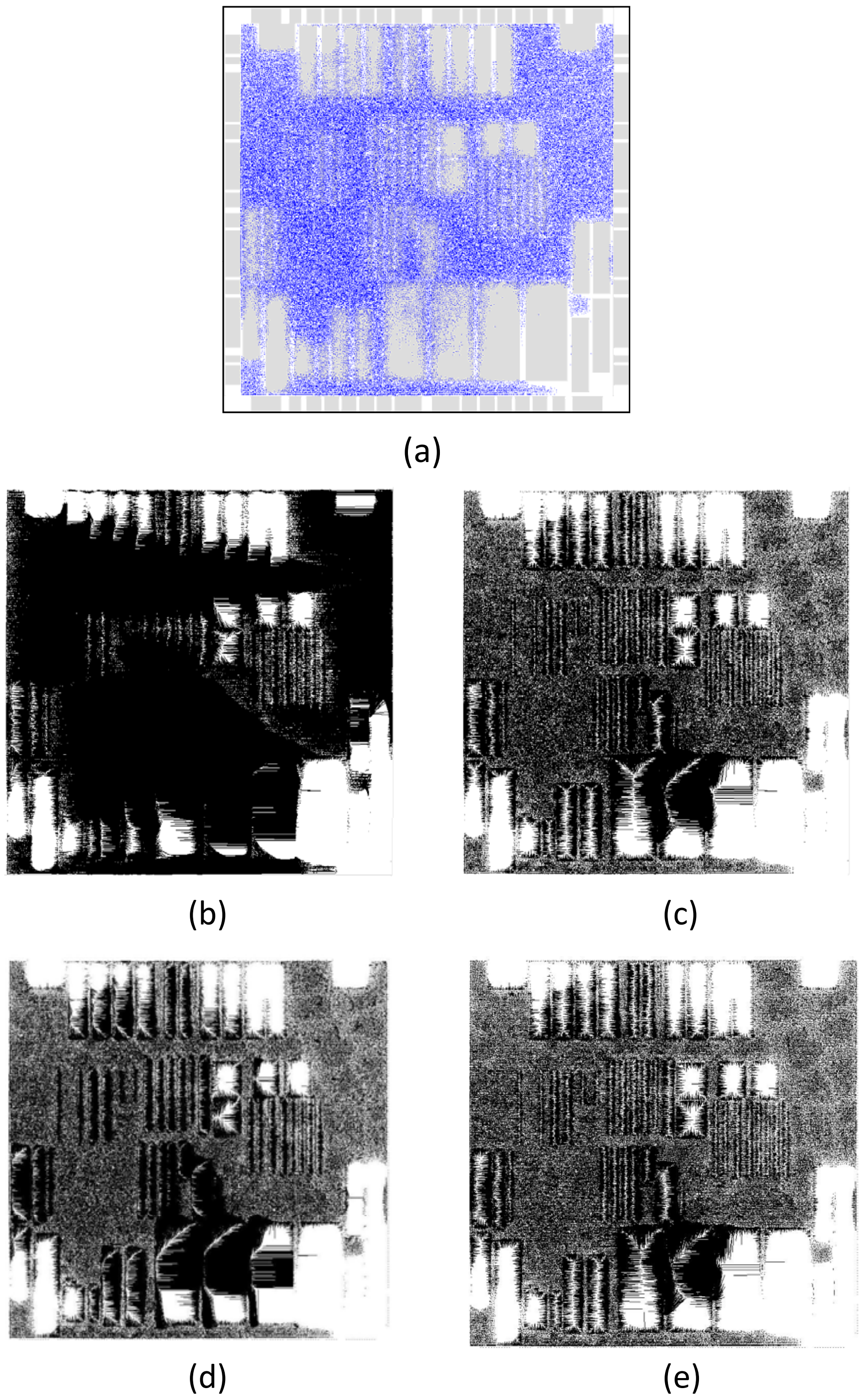


Fig. 14. (a) Global placement result of adaptec1. (b) Cell movement during Tetris [7]. (c) Cell movement during Abacus [19]. (d) Cell movement during Kahng [9]. (e) Ours.