

Exploiting Signal Flow and Logic Dependency in Standard Cell Placement

Jason Cong and Dongmin Xu

Computer Sci. Dept., UCLA, Los Angeles, CA 90024

Abstract -- Most existing placement algorithms consider only connectivity information during the placement process, and ignore other information available from the higher levels of design process. In this paper, we exploit the use of signal flow and logic dependency in standard cell placement by using the maximum fanout-free cone (MFFC) decomposition technique. We developed a containment tree based algorithm for splitting large MFFCs into smaller ones to get clusters with restricted sizes. We also developed a placement algorithm, named MFFC-TW, which first clusters the circuit based on MFFC decomposition and then feeds the clustered circuit to the Timberwolf6.0 placement package. Very promising experimental results were obtained.

1. Introduction

Placement is an important and difficult step in VLSI layout synthesis. The goal of the placement is to map the circuit components onto positions of a layout surface. The components must be placed in such a way that the chip can be routed efficiently and the timing requirements can be satisfied. Due to the theoretical and practical importance, the placement problem has been studied extensively in the past two decades. However, rapid increase in chip packing density (by a factor of 10 to 100 in the past decade) on a single VLSI chip and increasing significance of interconnect delay (affected greatly by the placement result) have led to renewed interest in new and efficient placement algorithms for handling the design complexity and performance constraint.

The existing placement algorithms can be divided in two major categories: *constructive placement* or *iterative placement*. *Constructive placement algorithms* start with an unplaced netlist and construct a complete placement. The algorithms in this class include the cluster growth algorithms, the partitioning-based algorithms, the analytical placement algorithms, and the branch-and-bound algorithms. On the other hand, *iterative placement algorithms* start with a given complete placement and go through a number of local refinement steps to obtain an improved placement solution. The algorithms in the class include the pairwise interchange methods, the force directed methods, and the simulated annealing based methods. More detailed survey of the existing placement algorithms can be found in [PrLo88] and [Le90].

Most of existing placement methods ignore the signal flow and logic dependency in the circuit, and simply treat the netlist as an undirected graph or hypergraph, considering only the connectivity information. The approach proposed in this paper is to exploit the use of the signal flow and logic dependency to guide the placement process. Our study shows that the signal flow and logic dependency provide additional information not captured in the connectivity measurement. For example, the two small circuits shown in Figure 1 have the same connectivity. However, cell *A* and cell *C* are closely related in Figure 1(a) as the output of *C* depends on the output of *A*, while cell *A* and cell *C* are loosely related in Figure 1(b) since they can generate output signals

independently and provide them to cell *B*. Such relationship is implied by signal flow and logic dependency.

In this paper, we developed an effective circuit clustering method based on the theory of maximum fanout-free cone (MFFC) decomposition. The MFFC decomposition technique for combination circuits was first proposed by Cong and Ding [CoDi93] for duplication-free area-optimal technology mapping of lookup-table based FPGAs. Most existing clustering algorithms, such as the density-based clustering algorithm [CoSm93], random-walk based clustering [HaKa92], absorption based clustering [SuS93], and multi-dimensional spectral embedding based clustering [AlKa93, ChSZ93], use solely the connectivity information among the components, while MFFC decomposition considers both signal directions and logic gate dependency in the circuit. Recently, it has been applied to area-balanced multi-way circuit partitioning [CoLB94] and showed 30% to 50% reduction of net cut-size when compared with the well-known K-way FM partitioning algorithm [Sa89]. These results motivate us to study the impact of the MFFC based clustering method on circuit placement.

In this work, we generalized the MFFC decomposition method to sequential circuits and developed a containment tree based algorithm for splitting large MFFCs into smaller ones to obtain clusters with the certain sizes. To study the impact of such MFFC based clustering method on standard cell placement, we developed a placement algorithm, named MFFC-TW, which first clusters the circuit using MFFC based clustering and then feeds the clustered circuit to the Timberwolf6.0 placement package [SeLe87]. Experimental results on MCNC layout benchmark circuits show that MFFC-TW reduces the total number of tracks by up to 15.5% (with almost identical width), total wirelength by up to 13.7%, critical path delay by up to 21.2%, and runtime by up to 50.3% when compared to Timberwolf6.0. Our results showed convincingly that the current placement techniques can be further improved using additional information, such as signal flow and logic dependency, available from higher levels of design process. Our MFFC based clustering technique can also be used with other placement techniques, such as min-cut or quadratic programming based placement algorithms.

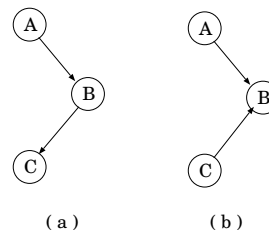


Figure 1 Simple examples to illustrate the use of signal flow and logic dependency

The remainder of the paper is organized as follows. Section 2 reviews the concept of MFFC decomposition and its properties. Section 3 describes our MFFC clustering based placement algorithm. The experimental results are presented in Section 4. Finally, conclusions and future research are included in Section 5.

2. Review of MFFC and Its Properties

The MFFC decomposition technique was first proposed for combination circuits [CoDi93]. Let $input(v)$ denote the set of nodes which are the fanins of node v , and $output(v)$ the set of nodes which are the fanouts of node v . For a node v in the network, a cone of v , denoted by C_v , is a subgraph of logic gates (excluding primary inputs (PIs)) consisting of v and its predecessors such that any path connecting a node in C_v and v lies entirely in C_v . We call v the root of C_v . A fanout-free cone (FFC) at v , denoted by FFC_v , is a cone of v such that for any node $u \neq v$ in FFC_v , $output(u) \subseteq FFC_v$. The maximum fanout free cone (MFFC) of v , denoted by $MFFC_v$, is an FFC of v such that for any non-PI node w , if $output(w) \subseteq MFFC_v$, then $w \in MFFC_v$. Figure 2 shows the MFFC of each node (see the smallest shadowed area of that node) in a network. It is not difficult to show that MFFC is unique for every node, and any FFC of v is contained in $MFFC_v$. Clearly, if a gate u is in $MFFC_v$, its value is used solely for generating the output at gate v (and its descendants). Therefore, it is very natural to cluster u and v together. In general, all the gates in a single $MFFC_v$ can be considered to be closely related, since they are used solely for computation of v . The results in [CoDi93, CoDi94b] showed that MFFCs have the following important properties:

- (P1) If $w \in MFFC_v$, then $MFFC_w \subseteq MFFC_v$.
- (P2) Two MFFCs are either disjoint or one must contain another.

Based these results, we can decompose a combinational circuit N into a set of disjoint MFFCs as follows: (i) choose a primary output (PO) node v from N and compute $MFFC_v$, (ii) let N be $N - MFFC_v$ and include those nodes in N with fanouts to $MFFC_v$ also as POs of N , and (iii) decompose N recursively. It is easy to see that such a decomposition is unique. Figure 3 shows the MFFC decomposition of the network of Figure 2. Note that MFFC decomposition is different from tree decomposition, because an MFFC can contain reconvergent fanout. The following results have been shown for MFFC decomposition:

- (P3) For technology mapping of lookup-table (LUT) based FPGAs, we can compute an area-optimal duplication-free mapping of each MFFC in the MFFC decomposition of N independently to get an area-optimal duplication-free mapping of N [CoDi93].

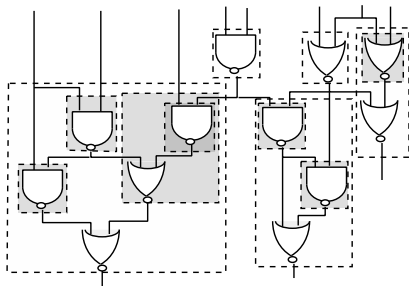


Figure 2 Maximum fanout free cone (MFFC) of each node

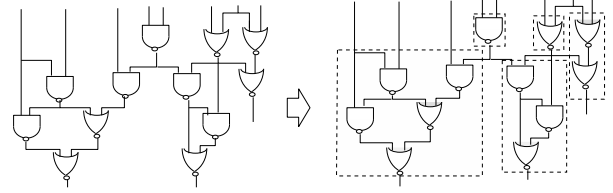


Figure 3 MFFC decomposition of the network in Figure 2

- (P4) For acyclic partitioning, if we do not consider the area constraint, there is an optimal acyclic two-way partition (X, Y) of N , such that for each $MFFC_i$ in the MFFC decomposition of N , either $X \cap MFFC_i = \emptyset$ or $MFFC_i \subseteq X$ [CoLB94]. (i.e. the optimal cut will not cut through any MFFCs).

It is clear that MFFC decomposition considers both signal flow and logic dependency. The properties P1-P4 further suggest that MFFC decomposition produces natural circuit decomposition. A few other clustering/decomposition methods have been developed in the past which consider signal directions in the circuits. These methods include the corolla based decomposition method by Dey, Brglez, and Kedem [DeBK90] based on the analysis of convergent fanouts, and the simple cone-based decomposition by Saucier, Brasen, and Hiol, which allows the overlapping cones. [SaBH93]. In a very recent work [TsLi95], Tsay and Lin introduced a cone structure identified by a clustering heuristic which is similar to MFFC clustering. They applied this clustering technique to standard cell placement. Their algorithm searches the cones in the given circuit and merges small cones into larger clusters first. Then, it uses TimberWolfMC to perform a macro cell placement which each cone is treated as a soft macro cell. A mapping procedure is used to map cells in each macros to the standard cell rows. Finally, TimberWolf 6.0 is used to refine the placement result by starting it at a low annealing temperature. The experimental results showed that this method improves the wire length and track density on most examples with shorter CPU time, when comparing with TimberWolf6.0.

In general, the properties P1-P4 do not hold for corolla based decomposition and other cone-based decompositions.

3. MFFC Clustering Based Standard Cell Placement

Our new placement algorithm, MFFC-TW, first computes MFFC based clustering and then uses TimberWolf6.0 to place the clustered circuit. It consists of the following procedures:

- (i) Use MFFC decomposition technique to obtain a natural circuit decomposition.
- (ii) Use a containment tree based approach to split large MFFCs into smaller ones.
- (iii) Generate linear placement for all cells in each cluster and then group them as a large cell to obtain the clustered circuit.
- (iv) Use TimberWolf6.0 to place the clustered circuit and perform global routing.
- (v) Calculate the critical path delay to evaluate the circuit performance.

The remainder of this section explains each step in detail.

3.1. MFFC Decomposition

We use the MFFC decomposition method for combinational circuits from [CoDi93] and extend it to the sequential circuits by allowing directed cycles in some MFFCs.

Given a combinational or sequential circuit N , we first remove clock and reset signals, since they are connected to all sequential elements. Let set Q , initialized to empty, store the resulting MFFCs from our decomposition. We repeat the following three steps to obtain MFFC decomposition of network N : (i) choose an arbitrary PO or the input to a sequential element (SE) v from N , initialize $MFFC_v = \{v\}$; (ii) construct $MFFC_v$ by repeatedly including a transitive fanin u of v as soon as $output(u) \subseteq MFFC_v$ and stop when no such cell exists. (iii) put $MFFC_v$ into Q , let $N = N - MFFC_v$, and update the PO list to include those gates which fanout to some gates in $MFFC_v$. The process stops when N is empty. Figure 4 shows the MFFC decomposition of a sequential circuit. Notice that the left shadowed rectangle is an MFFC with a signal loop inside. Since this algorithm traverses each connection only once during the MFFC decomposition, it is not difficult to show that the time complexity of the MFFC decomposition algorithm is $O(m)$, where m is the number of connection in N .

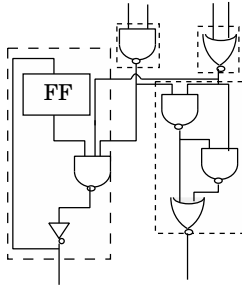


Figure 4 MFFC decomposition of a sequential circuit

3.2. Containment Tree Based MFFC Splitting

After MFFC decomposition, an MFFC splitting process is necessary to split large MFFCs into smaller ones such that the number of different cell widths is not increased significantly, compared with that of the original network. If this is not done, the placement results may be poor, since the exchange of two cells with different widths will cause a large penalty in the TimberWolf cost function and make the calculation of total wire length inaccurate.

For standard cell design we assume that cells have the identical height and variable width. Let $NC = \{nc_1, nc_2, \dots, nc_n\}$ denote all cells in a given network N , $W(nc_i)$ the width of cell nc_i ($1 \leq i \leq n$). We define a cell width set for network N as $CW_N = \{w_1, w_2, \dots, w_d\}$ ($d \leq n$), which holds all different $W(nc_i)$ ($1 \leq i \leq n$). Since we place each cluster in a single row in our current implementation, the width of the cluster C is defined as

$$W(C) = \sum_{c \in C} W(c).$$

We define the size of a set S , denoted by $|S|$ to be the number of elements in S .

In our program, we first carry out MFFC decomposition to obtain the clustered network N_c of a given network N . Then, TimberWolf is used for the placement of N_c . Note that in general $|CW_{N_c}| \geq |CW_N|$, i.e. the number of cell widths increases after clustering due to the new cell widths introduced by the clusters. We

want to control $|CW_{N_c}|$, i.e. the number of cell widths in the clustered circuit so that it is easy for TimberWolf to exchange two cells (recall that TimberWolf allows cell overlap. The TimberWolf cost function penalizes the exchange of cells with different widths, discouraging this type of move). Therefore, we introduce the constraint $|CW_{N_c}| = |CW_N| + k$ to limit the number of new cell widths being introduced by clustering, where k is the parameter provided by the designer.

Given a network N to be implemented with standard cell design, let $MFFCS = \{MFFC_1, MFFC_2, \dots, MFFC_m\}$ be the set of MFFCs in the clustered network N_c after MFFC decomposition of N . If $|CW_{N_c}| > |CW_N| + k$, we choose an MFFC with the largest size to split. Suppose $MFFC_i$ ($1 \leq i \leq m$) is chosen, then $MFFC_i$ is decomposed into a number of clusters $C_{i1}, C_{i2}, \dots, C_{ip}$ such that $C_{ik} \cap C_{il} = \emptyset$ ($1 \leq k, l \leq p$) and $|C_{ij}|$ should be as large as possible. Notice that not all C_{ij} ($1 \leq j \leq p$) are MFFCs. Let $SC_i = \{C_{i1}, C_{i2}, \dots, C_{ip}\}$. We define $SM_i = \{C_{ij} | C_{ij} \in SC_i \text{ and } C_{ij} \text{ is an MFFC}\}$. In order to take advantage of the inherent features of MFFC, we want to maximize $|SM_i|$, i.e. we want as many clusters in SC_i to be MFFCs as possible. We can show that when the number of clusters in SC_i is larger than one, there exists at least one cluster in SC_i which is not an MFFC.

After $MFFC_i$ is split, the clusters in N_c are updated by replacing $MFFC_i$ with $C_{i1}, C_{i2}, \dots, C_{ip}$. Then, we update CW_{N_c} . If $|CW_{N_c}|$ is still larger than $|CW_N| + k$, we repeat the above process again by choosing an MFFC with largest size to split. This iteration is continued until $|CW_{N_c}| = |CW_N| + k$.

To describe the MFFC splitting algorithm which can satisfy the above requirements, we first give some definitions.

For a given MFFC of node v (e.g. $MFFC_v$), we assume that the MFFCs of all $u_i \in MFFC_v$ are $MFFC_{u_1}, MFFC_{u_2}, \dots, MFFC_{u_n}$ ($n = |MFFC_v|$). For each $MFFC_{u_i}$ ($1 \leq i \leq n$), it may contain a number of $MFFC_{u_j}$, e.g. $MFFC_{u_i} \supseteq MFFC_{u_j}$ ($1 \leq j \leq n, i \neq j$).

Def 1: $MFFC_{u_i}$ directly contains $MFFC_{u_j}$, if $MFFC_{u_i} \supseteq MFFC_{u_j}$ and $MFFC_{u_i}$ is the smallest MFFC (in size) containing $MFFC_{u_j}$.

Def 2: For a given MFFC M , its containment tree, denoted by $CT_M = (V, E)$, is defined as a directed tree, where each vertex $M_i \in V$ is a vertex which corresponds to $MFFC_{u_i}$ ($u_i \in M$), each directed edge $(M_i, M_j) \in E$ represents that $MFFC_{u_i}$ directly contains $MFFC_{u_j}$.

To illustrate the above definitions, let us look at a small circuit with six cells in Figure 5(a). All six cells are grouped in a MFFC. To split it, we find MFFCs of each node first. We obtain six MFFCs listed below:

$$\begin{aligned} MFFC_{u_1} &= \{u_1, u_2, u_3, u_4, u_5, u_6\} \\ MFFC_{u_2} &= \{u_2, u_4\} \\ MFFC_{u_3} &= \{u_3, u_5\} \\ MFFC_{u_4} &= \{u_4\} \\ MFFC_{u_5} &= \{u_5\} \\ MFFC_{u_6} &= \{u_6\} \end{aligned}$$

Let vertices $M_1 \sim M_6$ in the containment tree correspond to $MFFC_{u_1} \sim MFFC_{u_6}$, then the containment tree is shown in Figure 5(b). Note that although cell u_6 is not connected to cell u_1 in the circuit, there is an edge pointing from M_1 to M_6 . A possible

splitting solution for this example is $\{u_1\}$, $\{u_2, u_4\}$, $\{u_3, u_5\}$ and $\{u_6\}$.

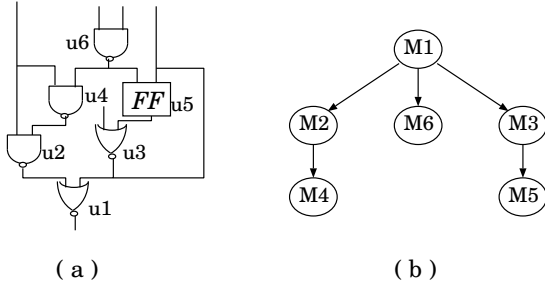


Figure 5 Containment tree of an MFFC

Given an MFFC M , let $root(M)$ be the root cell of M . If we need to split M , we always have $\{root(M)\}$ as a single cluster. Then, we traverse all edges of (M, M_i) in the containment tree CT_M to introduce MFFCs, corresponding to each M_i , into the cluster list. We recompute the cell width and find the next MFFC to split. This procedure can be carried out recursively until $|CW_{N_c}| \leq |CW_N| + k$.

3.3. Linear Placement for Cells in the Clusters

After MFFC splitting, we compute a linear placement for all cells in each cluster to reduce wirelength among them. The cells are abutted according to the linear ordering to form a large cell. When the number of cells in each cluster is small (less than 6), we use an exhaustive searching method to find the optimal linear ordering. When the number of cells in each cluster is large, we use the second smallest eigenvector of the Laplacian matrix of the MFFC to compute a linear placement [Ha70, HaKa91].

3.4. Placement of the Clustered Circuit

In general, we can use any placement algorithm at this stage. In our implementation, we choose TimberWolf6.0 [SeLe87] to place the clustered circuit, because of the availability of the tools and the demonstrated quality of TimberWolf6.0 placement solutions.

3.5. Timing Analysis

Since TimberWolf6.0 package provides only a global routing solution, to calculate the critical path delay without implementing the detailed router, we measure the wirelength using the maximum bounding box which encloses all pins of a signal net. The gate delay formula was given below [Ko91]:

$$delay = delay_constant + delay_fan \times output_capacitance$$

where $delay_constant$ is the intrinsic gate delay from an input pin to an output pin, $delay_fan$ the equivalent driver resistance, and $output_capacitance$ the total capacitance seen by the driver. Parameters $delay_constant$ and $delay_fan$ are specified in the cell library.

To calculate the wire delay, we used a π -type lumped RC model.

$$wire_delay = R_{wire} \times (C_{wire}/2 + C_{sinks})$$

where R_{wire} and C_{wire} are the lumped wire resistance and capacitance, respectively. C_{sinks} is the sum of the gate capacitances

of the sinks. We calculated the maximum delay of a given design from PIs (or flip-flops) to POs (or flip-flops) as the circuit delay.

According to the MOSIS SCMOS 2.0 μ technology file, the wire width of metal 1 and metal 2 for circuit *fract*, *struct* and *biomed* are 3 μ m. The other parameters are listed in Table 1. The same delay estimator and parasitic parameters are used to evaluate the delay of TimberWolf6.0 and MFFC-TW placement solutions.

sheet resistance(Ω/\square)		capacitance ($fF/\mu m^2$)	
metal 1	metal 2	M1 over sub	M2 over sub
0.108	0.045	0.027	0.021

Table 1 Parameters of MOSIS SCMOS 2.0 μ technology

4. Experimental Results

The algorithm proposed in this paper, MFFC-TW, was implemented in C under UNIX on SUN SPARC workstations. In the MCNC standard cell benchmark suite, we found out five circuits (listed in Table 2) which have signal direction information. Three of them (*fract*, *struct* and *biomed*) have timing information. The description of these circuits is shown in Table 2. Column 2 to 4 show the number of standard cells, the number of I/O pads and the number of nets in the designs, respectively. The fifth column shows the number of rows used in our tests. Column 6 and 7 show the number of MFFCs after MFFC decomposition and the number of clusters after MFFC splitting (with parameter $k=2$ as defined in Section 3.2), respectively. Finally, the eighth column shows the runtime (on SUN SPARC 5) MFFC-TW spends to generate the clustered circuit, which is negligible comparing to the runtime of the subsequent step of using TimberWolf6.0 to place the clustered circuits.

circuit	cells	pads	nets	rows	clusters after decomposition	clusters after splitting	runtime(s)
fract	125	24	147	5	60	107	0.8
primary1	752	81	904	17	414	705	8.7
struct	1888	64	1920	18	1183	1186	28.0
primary2	2907	107	3029	29	1117	2429	71.2
biomed	6417	97	5742	44	2176	2897	261.8

Table 2 Benchmark circuits and efficiency of MFFC based clustering algorithm

The specification given by MCNC about the design rules was used for these benchmarks. We compared the results by applying TimberWolf6.0 to both original and clustered circuit. For each circuit, we run TimberWolf6.0 five times. The values of the best designs by TimberWolf6.0 and MFFC-TW were reported in Table 3. In this table, column 3 to 7 show the number of tracks, the total wire length in meters, the width of maximum cell row in microns, the critical path delay in nano-seconds and the runtime of TimberWolf6.0 in seconds. The runtime is recorded on a SUN SPARC1000 server. The last two rows in Table 3 show the maximum and average reduction in percentage by MFFC-TW compared to TimberWolf6.0 for all five circuits, respectively.

From Table 3, we can find that the clustered circuits used much less CPU time for all the five circuits. The runtime is reduced by up to 50%. We can also find that MFFC-TW is especially effective for large circuits. For circuits *struct*, *primary2* and *biomed*, the track count is reduced by 15.0%, 6.7% and 15.5%, respectively; the wirelength is reduced by 5.8%, 2.8% and 13.7%, respectively,

with almost identical maximum row length. For circuit *fract*, *struct* and *biomed*(the only three circuits with timing information), the critical path delay is reduced by 21.2%, 12.1% and 18.6%, respectively. It is not surprising to see that MFFC-TW reduces the interconnect delay substantially as it groups logically dependent

gates close to each other. Note that TimberWolf6.0 is not a performance-driven placement algorithm. So, it is reasonable to expect that our MFFC based clustering algorithm will produce even better timing results when integrated with a performance-driven placement algorithm.

circuit	algorithm	track	wireLen(m)	width(μm)	delay(ns)	time(s)
fract	TW	40	0.062	1590	64.51	328.8
	MFFC-TW	38	0.061	1576	50.82	289.6
	DC-TW	37	0.066	1590	60.77	340.3
primary1	TW	174	1.07	4760	*	1057.0
	MFFC-TW	175	1.05	4760	*	802.0
	DC-TW	166	1.04	4750	*	851.2
struct	TW	177	0.69	5544	699.30	3200.1
	MFFC-TW	150	0.65	5506	615.04	1589.6
	DC-TW	174	0.69	5541	698.25	2697.1
primary2	TW	507	3.89	7972	*	6050.7
	MFFC-TW	473	3.78	7986	*	5495.5
	DC-TW	479	3.73	8014	*	6517.4
biomed	TW	756	3.88	10385	173.50	16173.2
	MFFC-TW	639	3.35	10415	141.30	9999.5
	DC-TW	758	3.80	10427	238.38	21554.0
max. reduction by MFFC-TW		-15.5%	-13.7%	-0.9%	-21.2%	-50.3%
ave. reduction by MFFC-TW		-8.3%	-5.2%	-0.2%	-17.3%	-26.7%
max. reduction by DC-TW		-7.5%	-4.1%	-0.2%	-5.6%	-19.5%
ave. reduction by DC-TW		-3.9%	-0.5%	+0.1%	+10.6%	+1.9%

Table 3 Comparison of MFFC-TW and DC-TW with TimberWolf6.0 on MCNC benchmark circuits (* Delay information was only available for *fract*, *struct*, and *biomed*)

To study the effectiveness of MFFC clustering, we also implemented the density-based clustering algorithm[CoSm93] and replaced the MFFC clustering algorithm with this density-based algorithm in MFFC-TW. This new package is named DC-TW. In this density-based clustering approach, a netlist is represented by a graph rather than a hypergraph. An r -terminal net is represented by an r -clique in the graph. The weighting function of $\frac{1}{r}$ is used to weight each edge in r -clique. The density of a cluster is defined as $\frac{E}{M_c}$, where $M_c = \binom{n}{3}$ and E is the total weight of the edges in the cluster. The approach used in[CoSm93] is to collapse small cliques recursively if the size and density of each clique are larger than the pre-defined thresholds (the density threshold is defined as $\alpha_n \times D$, where D is the ratio of the total edge weight to $\binom{n}{3}$, n is the total number of vertices in the graph, $\alpha_n = 4.5$ when $n \leq 1000$, and $\alpha_n = 10$ when $n > 1000$). In our implementation, since we just need small clusters, the recursive process of the original algorithm is removed so as to avoid producing large clusters. We calculated the density of all cliques with the size of 3 to 5. For a clique, if its density is greater than the density threshold, we group all cells in this cluster as a large cell so as to obtain the clustered circuit. The results of this clustering algorithm is shown in Table 4. In Table 4, column 6 shows the number of clusters after the density-based clustering. Column 7 shows the runtime on SUN SPARC 5. We can find that the runtime is much longer for large circuits, when comparing with our MFFC clustering algorithm shown in Table 2. The comparison of DC-TW with TimberWolf6.0 is also shown in Table 3. We can see that the results of DC-TW are comparable with those of TimberWolf6.0. However, it spent even longer CPU time for large circuits, such as *primary2* and *biomed*. Also the critical path delay is nearly same as or larger than that of TimberWolf6.0. So, we

conclude that the MFFC clustering method is more effective than the density-based clustering algorithm.

circuit	cells	pads	nets	rows	no. of clusters after clustering	runtime(s)
fract	125	24	147	5	73	0.9
primary1	752	81	904	17	733	14.5
struct	1888	64	1920	18	1170	22.5
primary2	2907	107	3029	29	2670	1221.5
biomed	6417	97	5742	44	3650	476.9

Table 4 The clustering results of the density-based algorithm

Recently, TimberWolf7.0 has been developed and reported to produce better placement in terms of total wirelength and critical path delay [SuS93]. The source codes of TimberWolf7.0 is not available to us at this point so that we have not integrated the MFFC based clustering algorithm with TimberWolf7.0. We expect to see better results by MFFC-TW when TimberWolf7.0 is used. It is clear that our MFFC based clustering algorithm can be used with other placement algorithms, such as min-cut based algorithm or analytical placement algorithms like Gordian/Domino [KISJ91, DoJS92], or Ritual [SrCK91].

5. Conclusions

We have presented a new MFFC clustering approach for large row-based standard cell designs. Because of the inherent properties of MFFC decomposition, the logic dependency relation of cells is carefully maintained in each MFFC cluster. To apply MFFC clustering technique to standard cell designs, we split large MFFCs into smaller ones. We developed a containment tree based MFFC splitting algorithm: **MFFC-splitting**, to accomplish this

task. Experimental results on MCNC layout benchmark circuits indicate that MFFC-TW is very effective. In many cases, we achieved substantial improvement on total number of tracks, total wirelength, and critical path delay. To study the effectiveness of MFFC clustering, we also implemented the density-based clustering algorithm[CoSm93] and found that the MFFC clustering approach is more effective.

Currently, we plan to perform a floorplan step to handle the clustered circuit, making splitting unnecessary after MFFC decomposition. The outline of our approach is to partition the given netlist based on the MFFC decomposition solution first, followed by floorplan sizing and floorplan ordering. The final placement will be generated based on the obtained floorplan result.

Acknowledgments

This work was partially supported by ARPA/CSTO under Contract J-FBI-93-112 for Computer Aided Design of High Performance Wireless Networked Systems.

REFERENCES

- [AlKa93] Alpert, C. J. and A. B. Kahng, "Geometric Embeddings for Faster (and Better) Multi-Way Netlist Partitioning," *Proc. ACM/IEEE Design Automation Conf.*, pp. 743-748, June 1993.
- [ChSZ93] Chan, P., M. Schlag, and J. Zien, "Spectral K-Way Ratio-Cut Partitioning and Clustering," *Proc. 30th ACM/IEEE Design Automation Conf.*, June 1993.
- [CoDi93] Cong, J. and Y. Ding, "On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping," *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 213-218, June 1993.
- [CoDi94b] Cong, J. and Y. Ding, "On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping," *IEEE Trans. on VLSI Systems*, Vol. 2, pp. 137-148, June 1994.
- [CoLB94] Cong, J., Z. Li, and R. Bagrodia, "Acyclic Multi-Way Partitioning of Boolean Networks," *Proc. ACM/IEEE 31st Design Automation Conf.*, pp. 670-675, June 1994.
- [CoSm93] Cong, J. and M. Smith, "A Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Designs," *ACM/IEEE Design Automation Conf.*, pp. 755-760, June 1993.
- [DeBK90] Dey, S., F. Brglez, and G. Kedem, "Corolla Based Circuit Partitioning and Resynthesis," *Proc. 27th Design Automation Conf.*, pp. 607-612, June 1990.
- [DoJS92] Doll, K., F. M. Johannes, and G. Sigl, "Placement Improvement by Network Flow Methods," *International Workshop on Layout Synthesis*, Vol. 2, pp. 179-182, May 1992.
- [Ha70] Hall, K. M., "An r-dimensional Quadratic Placement Algorithm," *Management Science*, Vol. 17, pp. 219-229, 1970.
- [HaKa91] Hagen, L. and A. B. Kahng, "Fast Spectral Methods for Ratio Cut Partitioning and Clustering," *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 10-13, 1991.
- [HaKa92] Hagen, L. and A. B. Kahng, "A New Approach to Effective Circuit Clustering," *Int'l Conf. on Computer-Aided Design*, pp. 422-427, Nov. 1992.
- [KISJ91] Kleinhans, J. M., G. Sigl, F. M. Johannes, and K. J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization," *IEEE Trans. on Computer-Aided Design*, Vol. 10(3) pp. 356-365, March 1991.
- [Ko91] Kozminski, K., "Benchmarks for Layout Synthesis - Evolution and Current Status," *Proc. ACM/IEEE Design Automation Conference*, pp. 265-270, 1991.
- [Le90] Lengauer, T., *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons (1990).
- [PrLo88] Preas, B. and M. Lorenzetti, *Physical Design Automation of VLSI Systems*, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA (1988).
- [Sa89] Sanchis, L., "Multiple-Way Network Partitioning," *IEEE Trans. on Computers*, Vol. 38, pp. 62-81, 1989.
- [SaBH93] Saucier, G., D. Brasen, and J. Hiol, "Partitioning With Cone Structures," *Proc. of ICCAD-93*, pp. 236-239, 1993.
- [SeLe87] Sechen, C. and K. W. Lee, "An Improved Simulated Annealing Algorithm for Row-Based Placement," *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 478-481, Nov. 1987.
- [SrCK91] Srinivasan, A., K. Chaudhary, and E. S. Kuh, "RITUAL: Performance Driven Placement Algorithm for Small Cell ICs," *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 48-51, Nov. 1991.
- [SuS93] Sun, W.-J. and C. Sechen, "Efficient and Effective Placement for Very Large Circuits," *Proc. of ICCAD-93*, pp. 170-177, 1993.
- [TsLi95] Tsay, Y. and Y. Lin, "A Row-Based Cell Placement Method That Utilizes Circuit Structural Properties," *IEEE Trans. on CAD*, pp. 393-397, March 1995.