**Abstract**

We present a circuit clustering technique based on graph coloring. A given netlist is modeled as an undirected graph and its vertices are colored. Based on this coloring information and the notion of rank of a node and the number of adjacent unique colors it sees, we derive cost functions for the graph edges. We identify cliques in the graph and use these cliques, starting from the max_clique, as building blocks for our clusters. A cost function is derived using the cluster density notion and edge costs. Finally, we use this cost function to identify the critical edges, which when deleted, yield good clusters in the original circuit.

1 Introduction

1.1 Motivation

Recent advances in VLSI technology has made it possible for system designs to contain millions of gates. Increasing gate complexity has made it necessary for CAD tools to come up with ways to keep these designs tractable. Circuit clustering and partitioning are important techniques in keeping "strongly connected" components of designs together for efficient placement and routing.

A "cluster" can be viewed as a group of strongly inter-connected nodes in a circuit such that the number of edges connecting these nodes to one another is much greater relative to the number of edges connecting this subset of nodes to the remaining nodes in the circuit. Clustering can hence be viewed as a bottom-up approach to dividing a larger problem into smaller problems by identifying these regions of densely inter-connected nodes. Efficient algorithms are therefore needed to identify these regions of strongly connected nodes within a circuit. Greedy clustering methods fall into the class of iterative approaches and are more popular. The famous Fiduccia-Mattheyses(FM) iterative algorithm is widely used for circuit bisection. Circuit clusters can be generated using recursive application of the FM algorithm. However, the solution quality hence generated is not very predictable. Flow oriented approaches use the concept of maximum flow minimum cut.

Yeh, Lin and Cheng [6] used this approach in their clustering work which utilized the "ratio cut" approach. Yeh explores the trade-offs between the flow oriented clustering quality and running times in [7].

Density based clustering approaches define cluster density as \(|E| / C(n,2)|\), where \(|E|\) is the number of edges and \(n\) is the number of nodes in the cluster. Cong and Smith find r-cliques in a graph and use this cluster density criteria to find clusters in a circuit. Cluster density is used to determine the quality of the generated clusters.

Clusters with a higher density are considered to be of a higher quality. This clustering metric, however, favors smaller clusters as the term \(C(n,2)|\) increases quadratically in \(n\).

In this paper, we model a given netlist as an undirected graph, find the max-clique in the graph and use the max-clique as a preprocessing step to exactly color the graph. Recently, Coudert [1] showed that most real-time graphs appear to be 1-perfect, i.e., the size of the max clique is equal to the chromatic number of that graph, and hence can be colored exactly with little time overhead. We use the results of the coloring algorithm to extract information about the nodes in a given circuit. After exactly coloring the graph, we assign rank and color values to each node and edge in the graph. This assignment is detailed in the subsequent section. These rank/color values are then used to assign costs to the edges in the graph. A cluster is formed when the sum of the edge costs in a subset of nodes is relatively much larger than the sum of edge costs connecting this subset of nodes to the remainder of the graph. This in a sense is an extension of the cluster density criteria.

1.2 Previous Work

Previous work on clustering can be categorized as either bottom-up or top-down. Top-down approaches partition a given netlist into smaller subclusters. On the other hand, a bottom-up approach starts by assigning each cell into its own cluster and then merges these small clusters into larger clusters. These two approaches achieve different objectives. Top-down approaches have the advantage of having available an overall "global" view of the netlist, yet these approaches suffer from having to perform a global analysis at every step. Bottom-up approaches have the advantage of being more time efficient, yet they may suffer from making mistakes and having to backtrack.

The bottom-up approach relies on the distance between the nodes in a cluster, e.g., greedy cluster growth, random walk and k-l connectedness. These approaches are analogous to the process of growing crystals. Density based approaches fall into this bottom-up category. Other works in bottom-up clustering have utilized the compaction algorithm [9] and the k-l connectedness algorithm. The compaction algorithm increases the average degree of a graph by finding a maximal random matching on the graph. In this algorithm, each edge in the matching represents a cluster and 2 nodes which are connected by a matching edge are collapsed to form a single node in the compacted graph. However, the compaction algorithm fails to find natural clusters in a graph. The k-l connectedness algorithm is a constructive algorithm that uses the transitive closure property of the relation [10] to form clusters in a given netlist. However, the main drawback of this algorithm is how to choose the best values for k and l.

Another bottom-up clustering algorithm uses the random walk approach [11]. This approach is based on traversing the graph at one end and taking \(n^2\) steps through the graph. The clusters are based on cycles during the random walk. A major disadvantage of this approach is the \(O(n^3)\) time it takes. So for very large netlists, this becomes time inefficient.

Top-down clustering approaches have used the Kernighan-Lin/ FM partitioning algorithm. Ratio-cut approaches fall into this...
category. A modification of this ratio-cut approach is the cluster ratio approach. These approaches generalize the concept of max-
imum-flow-minimum-cut. Nodes in the circuit are modeled as com-
munication hosts and nets are modeled as communications links
having finite capacities. Yeh and Cheng used this approach in
devising a stochastic flow injection method. The main goal of this
flow-based clustering approach is to produce optimal flow distribu-
tion as quickly as possible. However, the running time is prohibi-
tively long and not very feasible for large scale applications.

Substantial work has also been done in the area of spectral
partitioning/clustering. In spectral partitioning/clustering, the
eigenvectors and eigenvalues of a graph matrices are computed,
and a cost function obtained. This cost function is then minimized.
Iterative heuristics are then devised to map the information pro-
vided by the eigenvectors into an actual partition. These heuristics
explore the solution space by making a large number of usually
greedy moves to attempt and achieve a global minimum.

Several different clustering objectives exist that determine the
quality of the clusters generated. These include:

- **DS Ratio**: The cluster degree is the average number of nets
incident to each component in the cluster and the cluster separation
is the average distance between 2 components in a given cluster.
This degree/separation cluster metric utilizes the fact that clus-
ters with higher DS values are of higher quality. This metric takes
cubic time so is very time-inefficient.

- **Split**: This objective evaluates clustering results with respect
to a netlist bisection that partitions the netlist into 2 equal-area par-
titions. A better clustering result will have fewer clusters that are
split by this netlist bisection.

- **Density**: This metric evaluates a given cluster consisting of
n nodes according to its density, i.e. |E| / C(n,2). We use this crite-
ria for our cluster evaluation.

Kahng and Sharma in [12] study more clustering metrics in
detail. Specifically, they define their own cluster criteria and study
different clustering heuristics including matching-based clustering
and cone-based variants. In the matching-based clustering, a set
of disjoint pair of cells are found such that each pair shares a com-
mon net. A maximal random matching is then found and each pair
of cells is merged into a single cluster. It is interesting to note that
each matching-based variant is done in the context of hypergraph
partitioning. They also outline several heuristics that perform clus-
tering separately on sequential and combinational parts of a given
netlist. They implement different variations of algorithms in the
work on [12]. In related work, Cong and Ding [13] find maxi-
mum fanout free cones in the DAG of a combinational network for
clustering purposes.

### 2 Circuit Model and Terminology

We use the information about graph coloring to find natural
clusters in a given circuit. We do not predetermine the size or the
number of clusters to be generated. We model a given netlist as a
graph G(V,E) with node set V = {V_i | i = 1,2,3,...n} and edge set E
= {e_u | u = 1,2,3,...m}.

![Graph representation](image)

**Figure 1: Graph representation of a net**

**Figure 1** shows how a given net is mapped into edges and
nodes in a graph. We use this model instead of mapping an r-ter-
mental net into an r-clique. Our model forms a node for each mod-
ule/cell in the net and then forms an edge from the source terminal
node to the nodes representing the remaining modules/cells in the
net.

Recently, Courdert [1] showed that most real-life graphs can be
colored exactly with little time overhead since most of them are
1-perfect. This allows us to model a given netlist as a graph, ex-
actly color the graph and then use the exact coloring of the graph
to extract connectivity information on the graph.

We define the color of a node as the number of unique colors
that are adjacent to it. The rank of a node is the number of its adja-
cent nodes. A node is then given a color/rank value. Edge rank/
color values are assigned as follows. Consider an edge ab con-
necting nodes a and b. Then

\[
\text{Rank}_{ab} = \text{Rank}_a + \text{Rank}_b - 2 \quad (2.1)
\]

\[
\text{Color}_{ab} = \text{Sum}(\# \text{ unique colors adjacent to nodes } a \text{ and } b
\text{ on removing edge } ab) \quad (2.2)
\]

These Edge rank/color values are used in finding the eventual
costs on the graph edges. p-cost is the compensatory cost assigned
to each edge and is defined in the next section.

### 3 Analysis of our Approach

The key idea involved in using coloring information about a
graph for clustering purposes is that the number of unique colors a
graph node sees is a measure of the connectivity of that node. For
example, a node that has 4 neighbors but sees only 1 unique color
is less strongly connected to it’s neighbors than a node that has 4
distinct neighbors and sees 4 distinct colors. In the latter case, we
have a very strongly connected group of nodes. In the case where a
node sees 4 distinct colors despite not being strongly connected to
it’s neighbors (this may happen because of the way we color our
graph) ,we use a compensatory cost to distinguish between this
node and a node that sees 4 unique colors and is strongly con-
ected to it’s neighbors. We call this the p-cost (Figure 2). Here p-
cost are indicated in ( ).

Each edge has 2 costs associated with it, r-cost and p-cost. The
r-cost of an edge is simply obtained by observing the rank (number of
neighbors seen by a node) and color (number of distinct colors the
node can see) of the opposite nodes to that edge.

\[
r\text{-cost node} = (\text{rank/color}) * 100 \quad (3.1)
\]

\[
r\text{-cost edge} = (\text{source node-cost target node-cost} ) * 100 \quad \text{where source node is the higher costing node. (Here source node does not mean the source node in the original net list)} \quad (3.2)
\]

![Figure 2: Calculating p-costs](image)

\[
p\text{-cost Edge}_{ab} = \text{Sum}(r\text{-cost Edge}_{ac}, r\text{-cost Edge}_{bc}) \text{ if node c has edges whose opposite ends are nodes } a \text{ and } b. \quad (3.3)
\]

An example graph and the necessary steps are shown in Figure
3. All the nodes have been given a rank/color value and all edges
have been assigned costs (p-cost + r-cost) according to our algo-
rum. The total cost for each edge as calculated is shown along-
side each edge.
The critical edge costs are indicated by arrows.

![Diagram with arrows indicating critical edge costs]

Figure 3: Edge rank/color and edge costs for sample graph

It is necessary to have the p-cost measure while coming up with the edge costs. Consider Figure 4 for instance. This figure is the same as Figure 2, except that p-costs have been excluded.

![Diagram with only r-costs shown]

Figure 4: Example Graph with only r-costs shown

It can be seen that edge costs with p-cost excluded identifies edge ab to be the least costing edge though it is evident that edge ab is part of a max-clique consisting of nodes a,b,c. Hence an exclusion of the p-cost factor incorrectly identifies ab to be the critical edge (the least costing edge) as opposed to edge cd as shown in Figure 2. p-cost takes into account the fact that deleting edge ab also involves the cost of deleting edges ac and bc. Since edge cd has nodes c and d as its opposite nodes which in turn have no common nodes between them, p-cost for edge cd is zero.

Hence, it is extremely important that we add the p-cost factor to the total edge costs. This compensates for the fact that there may be nodes in the graph that see many unique colors, thereby indicating that they are strongly connected when in fact they are not. The p-cost factor actually finds out if a particular node is strongly connected to its neighbors by considering if it is a part of a clique.

Since the graph coloring technique [1] involves finding the max-clique in the graph, we can store information about this max-clique and other cliques found during the max-clique search to identify removable edges. Consider the example in Figure 5 (this example is the same as in Figure 3). A max-clique search finds A, B and C as the possible max-cliques, (it is possible that other smaller cliques are found en-route to finding the max-clique, this smaller cliques information is also stored in the data structure).

![Diagram showing clique search to find critical edges]

Figure 5: Using Clique search to find critical edges

Edges a and b are the edges that connect these max-cliques to one another. A combination of the coloring information and the information generated during this max-clique search can be effectively used in identifying edges a and b as being critical and hence removable.

Our clustering approach is analogous to that of growing crystals. Our seed each time is a max-clique. We add nodes and edges to this initial group of nodes comprising the max-clique until adding additional nodes is not beneficial as determined by our cost function, outlined below in Section 4.

We start with the max-clique of the graph as a seed cluster, find the sum of the internal edge costs of this max-clique, find the sum of the external edge costs leaving this max-clique, and find the ratio of the two. Depending on this ratio, we choose those edges leaving the max-clique which have the most cost and add the nodes opposite to these edges to our current max-clique and check the feasibility of our cluster according to our cost function. We outline the algorithm in section 4.

4 Algorithm

Based on our above analysis, we propose the algorithm outlined below:

```plaintext
Algorithm Cluster(G)
1) Graph G(V,E)
2) Mc = Maximum-Clique(G)
3) int best_color = Exact_Color(G,Mc)
4) Forall nodes n \in G
   a) find rank
   b) find color
   c) node-cost = (rank/color) * 100
5) Forall edges e \in G
   a) find r-cost /* (sourcenode-cost/targetnode-cost) * 100 */
   b) find p-cost /* see Figure 2 */
   c) total-cost = r-cost + p-cost
6) Best_Cost = 0
7) Cluster_list = Find_Clusters(Mc, G, Best_Cost)

The Flow for identifying good clusters is outlined below:

Algorithm Find_Clusters(Mc, G, Best_Cost) /* initially Best_Cost = 0 */
1) Cluster = Mc
2) Mc = Max-Clique of remaining Graph
3) Cd = Cluster Density = |E| / C(n,2)
4)Wd = Sum (edge_costs internal to the cluster)/Sum(edge_costs leaving the cluster)
5) Cluster Cost = Wd * Cd
6) If (Cluster Cost > Best_Cost) then Mc = Mc + N ! ; GOTO (3)
   else
   Cluster_list.Add(Cluster) ; /* We have generated a new Cluster */
7) G = G - Cluster
8) If G != empty
   then Cluster_list = Find_Clusters(Mc, G,Cluster_Cost)
   else return(Cluster_list)
```

$N = \text{set of nodes which are opposite to the nodes in } M_c \text{ and connected by the highest costing edges leaving } M_c$. We determine when to stop adding these opposite nodes when we cannot get a better Cluster Cost as explained above.
Consider the sample graph shown in Figure 6. All the edges have costs assigned to them according to our algorithm. This graph has a max-clique consisting of nodes A,B,C and D and is connected to the remainder of the graph by edges AF, AE and DE. So this max-clique is our starting point for critical edge identification. The sum of the edge costs in this max-clique equals 3485. Note that since this is a clique, $C_d = 1$. The sum of the edge costs leaving this max-clique is 1396. The ratio of these two costs comes out to be 2.5. So we add these edges in the set of non-critical edges. Add nodes E and F to our current cluster (since edges AB, AD and DE all have high costs and satisfy our cost criteria) and next consider edges in the clique EFG. As can be observed, neither of edges EF, EG and FG can be identified as critical edges since their costs add up to 1679. and deleting any of these edges will not lead to any good cluster formation.

This graph has edge EF missing. An analysis of this graph yields the following:

- Sum of edge costs in max_clique A,B,C,D = 4337 (4.5)
- Sum of edge costs of edges AF, AE and DE = 814 (4.6)
- Ratio of (4):(5) = 5.3 (4.7)
- Sum of edges EG and FG = 350 (4.8)

From the above analysis, we observe that deleting edges AE, AF and DE will not lead to an optimum clustering. Instead, if we add node E to the set of nodes comprising the max_clique, we will get a set consisting of nodes A,B,C,D and E. Note that we find that edge AF has a cost of 120 and adding the node opposite to it, i.e. F to $M_c$ will not benefit our current cluster $M_c$. Sum of edge costs in this node set = 5151. Sum of edge costs leaving this node set = 270 giving us a cluster cost of $19.01 * 0.8 = 15.21$ (Here $C_d = 0.8$) This leaves us with nodes F and G in the remainder of the graph. They can be in turn be grouped into another set since they form a max-clique in the remaining graph. Hence we get 2 clusters. Cluster 1 consisting of nodes A,B,C,D and E and Cluster 2 consisting of nodes F and G. By dynamically adding node E to the starting node set of nodes A,B,C and D, we have managed to generate good clusters.

In this above example we have generated clusters by building on a group of nodes that comprise a max-clique. We dynamically add nodes that are adjacent to the nodes in the max-clique but not a part of it, by evaluating the feasibility of adding them according to our cluster density and edge cost criteria. Recall that, $C_d = \text{Cluster Density} = |E| / C(n,2)$ and our edge cost criteria is defined as $W_d = \text{Sum(edge costs internal to the cluster) / Sum(edge costs leaving the cluster)}$. In essence, we create a threshold value dynamically, and the building process stops when we can no longer improve on the cluster quality as defined by our cluster quality functions outlined above. The edge costs determine which nodes need to be added and which nodes need not be added. If an edge with a higher cost connects a node in a cluster to a node outside the current cluster, then this node is a better candidate to be added to the cluster than a node which is connected by an edge with a low cost. The way we determine the edge costs initially helps us identify these possible node candidates. Note that instead of adding one node at a time to the cluster (which is expensive) we add a set of nodes to the cluster.

5 Experimental Results

Table 1 shows the size of the test cases on which we tried our clustering algorithm, as well as the exact coloring obtained for these circuits (based on our graph model). Most test cases are from the standard MCNC set of circuits. However, we have also incorporated 3 of the smallest ibm circuits that were released in [ISPD98]. Work is currently under way to try this clustering algorithm on the remaining ibm circuits.

We compare our results to the Breadth First Clustering (BFC) approach in [7], the Shortest Path Clustering (SPC) approach [6] and the Recursive Ratio Approach (RR) [14]. Table 2 shows the obtained cluster sizes for all the benchmark circuits. The netlists in Table 1 have been modeled as graphs as explained earlier in section 2, instead of the traditional hypergraph. This is to facilitate the exact coloring of the graph. This may affect the clustering results we obtain since we model our netlists as graphs while most published results model them as hypergraphs. Even when other published results use a graph model, they use the standard clique-
based hypergraph transformations.

Table 1: Our Cluster Sizes

<table>
<thead>
<tr>
<th>Test Circuit</th>
<th># Nodes</th>
<th># Nets</th>
<th>Exact Color</th>
<th>Cluster sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary1</td>
<td>833</td>
<td>904</td>
<td>5</td>
<td>611 : 127 : 95</td>
</tr>
<tr>
<td>Primary2</td>
<td>3014</td>
<td>3029</td>
<td>5</td>
<td>2358 : 411 : 245</td>
</tr>
<tr>
<td>Test02</td>
<td>1663</td>
<td>1721</td>
<td>5</td>
<td>12255 : 438</td>
</tr>
<tr>
<td>Test03</td>
<td>1672</td>
<td>1699</td>
<td>5</td>
<td>1176 : 496</td>
</tr>
<tr>
<td>Test04</td>
<td>1551</td>
<td>1738</td>
<td>5</td>
<td>1073 : 478</td>
</tr>
<tr>
<td>Test05</td>
<td>2658</td>
<td>2910</td>
<td>5</td>
<td>2018 : 640</td>
</tr>
<tr>
<td>Test06</td>
<td>1821</td>
<td>1745</td>
<td>5</td>
<td>767 : 621 : 433</td>
</tr>
<tr>
<td>19ks</td>
<td>12752</td>
<td>14111</td>
<td>6</td>
<td>8259 : 3436 : 1057</td>
</tr>
<tr>
<td>ibm01</td>
<td>19601</td>
<td>19584</td>
<td>6</td>
<td>8259 : 3436 : 1057</td>
</tr>
<tr>
<td>ibm02</td>
<td>23136</td>
<td>27401</td>
<td>7</td>
<td>15422 : 7713</td>
</tr>
<tr>
<td>ibm03</td>
<td>2658</td>
<td>2910</td>
<td>5</td>
<td>2018 : 640</td>
</tr>
<tr>
<td>ibm04</td>
<td>1821</td>
<td>1745</td>
<td>5</td>
<td>767 : 621 : 433</td>
</tr>
<tr>
<td>ibm05</td>
<td>1551</td>
<td>1738</td>
<td>5</td>
<td>1073 : 478</td>
</tr>
<tr>
<td>ibm06</td>
<td>1663</td>
<td>1721</td>
<td>5</td>
<td>12255 : 438</td>
</tr>
<tr>
<td>ibm07</td>
<td>1672</td>
<td>1699</td>
<td>5</td>
<td>1176 : 496</td>
</tr>
<tr>
<td>ibm08</td>
<td>1551</td>
<td>1738</td>
<td>5</td>
<td>1073 : 478</td>
</tr>
<tr>
<td>ibm09</td>
<td>1663</td>
<td>1721</td>
<td>5</td>
<td>12255 : 438</td>
</tr>
<tr>
<td>ibm10</td>
<td>1672</td>
<td>1699</td>
<td>5</td>
<td>1176 : 496</td>
</tr>
<tr>
<td>ibm11</td>
<td>1551</td>
<td>1738</td>
<td>5</td>
<td>1073 : 478</td>
</tr>
<tr>
<td>ibm12</td>
<td>1663</td>
<td>1721</td>
<td>5</td>
<td>12255 : 438</td>
</tr>
<tr>
<td>ibm13</td>
<td>1672</td>
<td>1699</td>
<td>5</td>
<td>1176 : 496</td>
</tr>
<tr>
<td>ibm14</td>
<td>1551</td>
<td>1738</td>
<td>5</td>
<td>1073 : 478</td>
</tr>
<tr>
<td>ibm15</td>
<td>1663</td>
<td>1721</td>
<td>5</td>
<td>12255 : 438</td>
</tr>
</tbody>
</table>

Table 2 shows the size of the generated clusters using the SPC, BFS and RR cluster metrics.

Table 2: Published Clustering Results

<table>
<thead>
<tr>
<th>Test Circuit</th>
<th>SPC (Shortest Path Clustering)</th>
<th>BFS (Breadth First Clustering)</th>
<th>RR (Recursive Ratio)</th>
<th>Best Net Cut (SPC)</th>
<th>Best Net Cut (BFS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary1</td>
<td>681 : 76 : 76</td>
<td>3 clusters</td>
<td>742 : 91</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>Primary2</td>
<td>2283 : 731</td>
<td>3 clusters</td>
<td>2275 : 739</td>
<td>83</td>
<td>77</td>
</tr>
<tr>
<td>Test02</td>
<td>1596 : 34 : 33</td>
<td>3 clusters</td>
<td>794 : 429 : 345 : 46</td>
<td>152</td>
<td>9</td>
</tr>
<tr>
<td>Test03</td>
<td>1502 : 105</td>
<td>2 clusters</td>
<td>1286 : 321</td>
<td>48</td>
<td>15</td>
</tr>
<tr>
<td>Test04</td>
<td>1442 : 73</td>
<td>2 clusters</td>
<td>1121 : 394</td>
<td>51</td>
<td>6</td>
</tr>
<tr>
<td>Test05</td>
<td>2492 : 103</td>
<td>4 clusters</td>
<td>2308 : 287</td>
<td>48</td>
<td>8</td>
</tr>
<tr>
<td>19ks</td>
<td>1059 : 909 : 876</td>
<td>2 clusters</td>
<td>2771 : 7311</td>
<td>127</td>
<td>12</td>
</tr>
<tr>
<td>ibm01</td>
<td>19601 : 19584</td>
<td>6 clusters</td>
<td>10272 : 4813 : 4516</td>
<td>15422 : 7713</td>
<td></td>
</tr>
<tr>
<td>ibm02</td>
<td>23136 : 27401</td>
<td>7 clusters</td>
<td>15422 : 7713</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Our Cluster Quality

<table>
<thead>
<tr>
<th>Test Circuit</th>
<th>Original Graph Ratio (#edges/ #nodes)</th>
<th>Average Cluster Ratio (Edges/ Nodes)</th>
<th>% decrease in # edges</th>
<th># Nets Cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary1</td>
<td>2.41</td>
<td>2.36</td>
<td>0.07</td>
<td>22</td>
</tr>
<tr>
<td>Primary2</td>
<td>2.71</td>
<td>2.55</td>
<td>0.90</td>
<td>90</td>
</tr>
<tr>
<td>Test02</td>
<td>2.66</td>
<td>2.53</td>
<td>0.90</td>
<td>53</td>
</tr>
<tr>
<td>Test03</td>
<td>2.6</td>
<td>2.5</td>
<td>0.84</td>
<td>61</td>
</tr>
<tr>
<td>Test04</td>
<td>2.78</td>
<td>2.69</td>
<td>0.33</td>
<td>56</td>
</tr>
<tr>
<td>Test05</td>
<td>2.8</td>
<td>2.65</td>
<td>0.35</td>
<td>64</td>
</tr>
<tr>
<td>Test06</td>
<td>2.74</td>
<td>2.60</td>
<td>0.12</td>
<td>83</td>
</tr>
<tr>
<td>19ks</td>
<td>2.55</td>
<td>2.40</td>
<td>0.80</td>
<td>124</td>
</tr>
<tr>
<td>ibm01</td>
<td>2.9</td>
<td>2.77</td>
<td>0.48</td>
<td>411</td>
</tr>
<tr>
<td>ibm02</td>
<td>3.14</td>
<td>3.04</td>
<td>0.18</td>
<td>508</td>
</tr>
<tr>
<td>ibm03</td>
<td>2.86</td>
<td>2.78</td>
<td>0.80</td>
<td>1311</td>
</tr>
</tbody>
</table>

Column 2 shows the ratio of the number of edges to the number of nodes in each particular graph. The average of this is 2.74.

Column 3 shows the ratio of the average number of edges in a cluster to the number of nodes in that cluster. For all the tested benchmarks, this average comes out to be 2.62. An interesting fact to be noted is that if we compare the values in column 3 to that of column 2 (which gives us the ratio of the number of edges to the number of nodes in the original graph), we find that in all cases the value in column 3 is less by at most 5.90 % (for Primary2) from the value in column 2. Take for example, the netlist Primary1. Column 4 shows the percentage decrease between the values in columns 2 and 3 which is 2.07 % in this case. This is attributed to the number of inter-cluster edges and this is translated into the number of actual nets being cut. This nets cut value is reported in column 5.

Table 4 compares our cluster quality with that of the clusters generated through the Recursive Ratio (RR), Shortest path Clustering (SPC) and Breadth First Search (BFS) clustering methods using the cluster ratio metric [3] [6]. Given a cut C(V1, V2, ..., Vk), the cluster ratio of C is defined as

\[
R_c(V_1, V_2, ..., V_k) = \frac{\sum_{i=1}^{k} |V_i| \times \sum_{j=i+1}^{k} |V_j|}{\sum_{i=1}^{k} |V_i|^2}
\]

where \(C(V_1, V_2, ..., V_k)\) is the number of nets which join different clusters.

Table 4: Ratio Cut Comparison

<table>
<thead>
<tr>
<th>Test Circuit</th>
<th>Rc (RR)</th>
<th>Rc (SPC)</th>
<th>Rc (BFS)</th>
<th>Rc (Ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary1</td>
<td>16.29</td>
<td>12.81</td>
<td>12.81</td>
<td>14.81</td>
</tr>
<tr>
<td>Primary2</td>
<td>5.15</td>
<td>4.61</td>
<td>5.70</td>
<td>5.46</td>
</tr>
<tr>
<td>Test02</td>
<td>19.68</td>
<td>8.32</td>
<td>8.99</td>
<td>9.87</td>
</tr>
<tr>
<td>Test03</td>
<td>11.70</td>
<td>9.51</td>
<td>9.43</td>
<td>10.45</td>
</tr>
<tr>
<td>Test04</td>
<td>12.17</td>
<td>5.69</td>
<td>5.85</td>
<td>10.91</td>
</tr>
<tr>
<td>Test05</td>
<td>6.72</td>
<td>3.11</td>
<td>4.52</td>
<td>4.96</td>
</tr>
<tr>
<td>Test06</td>
<td>8.87</td>
<td>6.22</td>
<td>7.60</td>
<td>7.70</td>
</tr>
<tr>
<td>19ks</td>
<td>5.44</td>
<td>4.72</td>
<td>5.56</td>
<td>5.43</td>
</tr>
</tbody>
</table>

\(\times 10^{-5}\)
-ent clusters. We use this Cluster metric for the sake of comparison with RR, BFS and SPC. Note that a smaller value of \( R_c \) means a better cluster quality. We observe that our results are in most cases better than the those in RR and close to the ones in BFS except in Test04. We can see that SPC performs best but as we will see in Table 5, has a general trend of using excessive run time.

Our overall results are comparable with those in Table 4 especially the with the ones reported by BFS [7]. We believe that the way we model our graph can explain some of the differences in the results we obtain. The BFS results do not report the sizes of the generated clusters but they evaluate their results based on the cut size and the cluster ratio metric. No available data on clustering exists on the ibm circuits to our knowledge and hence we do not report previous results on these new ibm circuits. However, we need to emphasize that it is very important to include all these new circuits as most of the MCNC circuits are relatively small and hence not very relevant to current technology trends as far as circuit complexity and size are concerned.

SPC refers to the results reported by Yeh, Cheng and Lin [6]. BFS refers to the results in a follow-up paper by Ching Wei Yeh [7].

Table 5 shows the run times on the tested benchmark circuits in comparison with the SPC, BFS and RR results

<table>
<thead>
<tr>
<th>Test Circuit</th>
<th>RR(s)</th>
<th>SPC(s)</th>
<th>BFS(s)</th>
<th>Ours(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary1</td>
<td>2</td>
<td>54</td>
<td>12</td>
<td>6.4</td>
</tr>
<tr>
<td>Primary2</td>
<td>13</td>
<td>1221</td>
<td>59</td>
<td>20.89</td>
</tr>
<tr>
<td>Test02</td>
<td>13</td>
<td>843</td>
<td>30</td>
<td>25.51</td>
</tr>
<tr>
<td>Test03</td>
<td>6</td>
<td>527</td>
<td>19</td>
<td>29.85</td>
</tr>
<tr>
<td>Test04</td>
<td>11</td>
<td>522</td>
<td>18</td>
<td>28.16</td>
</tr>
<tr>
<td>Test05</td>
<td>33</td>
<td>1780</td>
<td>30</td>
<td>52.37</td>
</tr>
<tr>
<td>Test06</td>
<td>7</td>
<td>662</td>
<td>17</td>
<td>27.27</td>
</tr>
<tr>
<td>19ks</td>
<td>11</td>
<td>2085</td>
<td>20</td>
<td>42.73</td>
</tr>
<tr>
<td>ibm01</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>427.1</td>
</tr>
<tr>
<td>ibm02</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>2794</td>
</tr>
<tr>
<td>ibm03</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>4849</td>
</tr>
</tbody>
</table>

We run our C++ code on an Ultra Sparc 1 with 128 MB of memory. The run times reported in Table 5 are in seconds. We report the run times on the remaining clustering methods as published by their authors, hence they should be viewed as only an indication of the run time behavior of the clustering method as a whole. (Note: The BFS results were obtained using a SUN SPARC 2). Our run time results include the time to form a graph from the given netlist and to exactly color it. In cases where our run time seems big, it is because these cases do not model into 1-perfect graphs, i.e. the maximum clique in the graph is not equal to the graph’s chromatic number. However, we should note that all these graphs are exactly colored. In the case of the new ibm circuits, we can see that the run time for our method begins to become prohibitively large. Most portion of this run time is spent in exactly coloring the graph. Work is currently underway to empirically determine the trade-offs in relaxing the exact coloring criteria and the generated cluster quality. This relaxed criteria will help in generating clusters on the remaining large ibm circuits.

6 Conclusions

We have presented a clustering algorithm based on graph coloring. The main motivation for our approach was the work done in [1] which showed that most real life graphs are 1-perfect and hence can be exactly colored with little time overhead. We have used a bottom-up clustering technique to create natural clusters and report cut and cluster sizes on our obtained results. Experimental results show that our cluster quality is comparable to the reported results using the ratio cut metric as a comparison parameter [6]. Slight differences in results can be attributed to the different way in which we model our graph from the given netlists. Since the run times are reported on different machines, we cannot make an accurate comparison on the run times. However, our run times show a general trend of being smaller than the run times reported in SPC [6]. In essence, our results are a mere trade-off between cluster quality and run times. Our netlist graph model is not obtained from the standard clique-based hypergraph transformation and hence, this prevents us from making a true comparison from previously published results. We are currently working on refining our algorithm and relaxing the exact coloring criteria on the larger circuits so that we obtain results on all the ibm circuits from the ISPD98 benchmark suite.

References