

# New Graph-Based Algorithms for Partitioning VLSI Circuits

Christopher J. Augeri  
Department of Computer Science  
United States Air Force Academy  
USAF Academy, CO 80840  
[christopher.augeri@usafa.af.mil](mailto:christopher.augeri@usafa.af.mil)

Hesham H. Ali  
Department of Computer Science  
University of Nebraska at Omaha  
Omaha, NE 68182  
[hesham@unomaha.edu](mailto:hesham@unomaha.edu)

## Abstract

When designing a circuit, it may be too large to fit on a single layer of a chip, on a single chip, or on a single board. Regardless of the design level, the same objectives remain. Normally, it is desirable to minimize the number of layers, chips, or boards, along with minimizing the delay. Additional constraints, such as the number of interconnections and power consumption, must often be considered. We have developed two  $k$ -way bounded partitioning algorithms; one is evolutionary-based, while the other is a hierarchical graph center-based approach. The algorithms are implemented and compared with known partitioning algorithms. Since VLSI circuits can be naturally modeled by graphs, experiments were conducted by partitioning graphs from various graph families against both simulated and real-world partitioning criteria. A direct result of this research is a high-level abstract graph-partitioning model. This model allows one to specify mathematical evaluation metrics and control parameters, permitting inter-domain comparison of algorithms and allowing one to identify the particular scenarios they are best applicable to.

## 1. Introduction

In VLSI design, partitioning is used to subdivide a circuit into multiple circuits when a circuit is too large to fit on a single device. This device may be a module, chip, board or other comparable object. A circuit, for these purposes, is equivalent to a directed acyclic graph (DAG). The resulting sub-circuits are considered a partition. Hardware components are equivalent to vertices and wires are modeled by edges. Additionally, in contrast to many others, the algorithms we have developed directly handle  $k$ -way partitioning, rather than recursive bi-partitioning.

One of our comparison algorithms has its roots in the work done by Lawler et al during the 1960s, the Unit Delay Model (UDM). In this model, delay within a subgraph of a partition is considered negligible. Additionally, the delay through any gate is considered to be zero and the delay between subgraphs, or clusters, is set at a unit, hence the model's name. This algorithm, Best Predecessor (BPD), and its successors, consist of a three-phase routine [7, 10, 13]. BPD permits vertex replication across subgraphs, often generating a larger output graph than it started with. The algorithm generates an optimal partitioning solution for a single constraint, i.e. it optimizes for maximal delay according to the weight or size of the gates. The general delay model (GDM), developed by Murgai et al, expands upon the UDM:

1. Each vertex may have a unique delay.
2. No delay exists between vertices in the same subgraph.
3. There is a constant delay between different subgraphs.

The GDM allows for the modeling of more realistic circuits and equipment. For instance, a long wire length within a device

can be considered a delay. This may be modeled by inserting dummy vertices with the delay value of this wire length. Additionally, this model can be tailored to handle the layer of circuit design abstraction we are modeling, be it chip, board or device level. This model is used by Wong et al to extend Lawler's algorithm for this model. The core algorithm remains the same, and in fact, the UDM may be implemented via the GDM [10].

The other comparison algorithm used, Dominant Sequence Clustering (DSC), was developed for the parallel processing domain, whereas BPD was intended for circuit partitioning. DSC is also an optimal algorithm, as it assumes an unlimited number of processing units is available. Essentially, it partitions vertices whose ancestors have already been partitioned based on which currently incurs the maximum delay.

Our primary focus is on  $k$ -way partitions, where  $k$  is determined by size bounds and cut bounds. We studied this in partitions with small and large relative inter-partition delay values. Mathematically,  $k$ -way partitioning may be stated as given a graph  $G = (V, E)$ , with a set of partitions  $P = \{P_1, P_2, \dots, P_k\}$ , determine a mapping  $\Phi: V \rightarrow P$ , such that the size,  $m$ , and edge,  $\text{degree}(P)$ , constraints of each partition  $P_i$ ,  $1 \leq i \leq k$  is satisfied. Furthermore, it may be acceptable to allow replication, such that a vertex,  $V_i$ , may be present in two partitions,  $P_i$  and  $P_j$ , where  $i \neq j$ .

For our discussion, we will use the graph in Figure 1, ISCAS C-17. This graph was first presented at the 1985 International Symposium on Circuits and Systems [1]. The graph contains 11 vertices and 12 edges, with a maximum delay of 6, along paths {CFIK, EIK}. It contains five primary inputs, {A, B, C, D, E}, and two primary outputs, {J, K}.

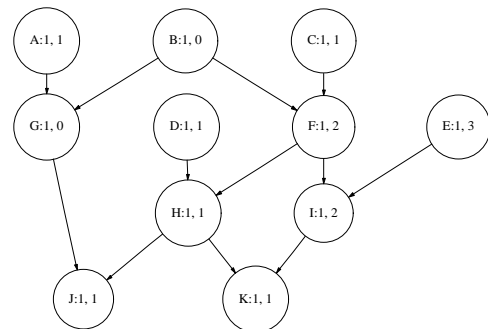


Figure 1: ISCAS C-17

## 2. Graph Partitioning Algorithms

**2.1 GEA: A Genetic Algorithm:** Our first algorithm is an evolutionary based algorithm. There are several key concepts involved with genetic-based algorithms. They are the encoding scheme, initial population creation, offspring creation, successive

generation selection, and generational run control. The key quality of genetic algorithms, because of the controlled randomness involved, is their ability to overcome convergence on local optima across the solution space. There are several new concepts we introduce, including a new encoding scheme, a new genetic operator and unique methods of establishing various experimental constraints.

Our encoding scheme is based on ideas present in two existing encoding schemes [5, 9]. A significant problem with these encoding schemes is the inability to identify all the vertices present in a given partition without scanning the entire solution. Furthermore, while one scheme has difficulties allowing for dynamic partition sizes, the other introduces difficulties with genetic operators due to the presence of the separator symbols.

To address these problems, we propose a two-strings coding scheme. The first string is a sequence of the node identifiers. The second string is a sequence of the length of each of the partition subsequences and is readily converted to a monotonically increasing sequence of absolute position values. In essence, this scheme extracts the separator information of the second scheme discussed, while permitting simple operators to be used when generating offspring.

Recalling a key principle in genetic algorithms, we know they are able to avoid convergence on local optima. To aid this process, we introduce the migration operator. Migration introduces a certain number of random individuals each generation as migrating, or “moving into the neighborhood”. They are still subject to the selection process, but early tests proved they are beneficial.

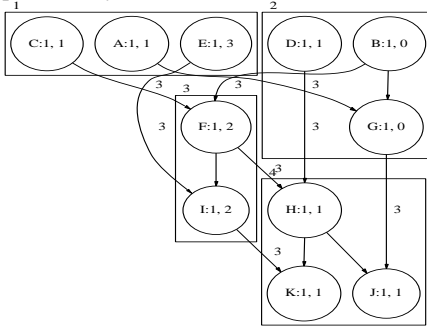


Figure 2: GEA partitioning of ISCAS C-17

Our genetic algorithm first randomly determines a initial population, where population size is determined by  $P = \log_2(|V|) * \log_2(k)$ . This allowed the number of vertices and partitions to both affect the population in a monotonically increasing manner. Considering that the number of unique combinations for a 1,024 element graph partitioned into 16 subgraphs is somewhere around 90 digits would have  $P = 10 * 4 = 40$ , this yields a reasonable population able to generate useful solutions.

Once an initial population is created, generation controls must be used to limit how long the algorithm will execute. This control is typically either a measurement of CPU time or the number of generations created. In our algorithm,  $G = \log_2(|V|) * \log_2(|E|)$ , where  $4 \leq |V| \leq 121$  and  $(|V| - 1) \leq |E| \leq (|V| * (|V| - 1) / 2)$ . Essentially, we desired to base the number of algorithm iterations on the density of the graph, while keeping the computation time reasonable, especially as  $|V|$  and  $|E|$  become large. For each generation, a number of offspring are created via creation, mutation, and exchange of “genetic material”. In this case, the “genetic material” is the encoded representation of a

given partitioning(s). Depending on the construction of these genetic operators, they are designed to induce varying sizes and types of changes in the offspring.

GEA uses a 3-valued implicit fitness function based on weighted evaluations of edge cut ( $C$ ), maximum delay ( $D$ ), and vertex weights ( $W$ ). Subscript  $a$  identifies a particular subgraph,  $m$  is a pre-determined maximum value, and  $f$  is the weighting factor. The sum of the three weighting factors equals one. The complete fitness function is shown in Equation 1.

$$G_c = \begin{cases} C_a/C_m > 1, C_a/C_m \\ C_a/C_m \leq 1, C_a/C_m * C_f \\ D_a/D_m > 1, D_a/D_m \\ D_a/D_m \leq 1, D_a/D_m * D_f \\ W_a/W_m > 1, W_a/W_m \\ W_a/W_m \leq 1, W_a/W_m * W_f \end{cases}, C_f + D_f + W_f = 1$$

Equation 1: GEA Fitness Function

Assuming an individual is fully feasible and meets constraints, the value of  $G_c \leq 1$ , with smaller values being better. After evaluating the new offspring formed during a generation, we use elitism to determine the next generation’s population. A certain percentage of low fitness function-rated, or high-quality individuals are first selected. The remaining individuals are selected based on a parental selection rate or randomly chosen from the high fitness function-rated, or low-quality offspring.

Using GEA, the partitioning was formed, with a maximum delay of 12, along path {CFIK}. Subgraphs are labeled numerically according to where they are in the encoding scheme. The representation with our new encoding method would be ([CAEDBGFIHJK], [3323]). Figure 2 depicts GEA’s partitioning of the ISCAS C-17 graph. Note the partitions are numbered rather than referenced by vertex labels. This is due to the algorithm not using vertex-specific information when labeling a partition.

**2.2 CTR: A Graph Center Based Algorithm:** Our second algorithm, CTR, is based on graph center metrics. The proposed algorithm was a recursive algorithm involving determining a graph center, extract a subgraph based on this center, and repeat this process until all vertices were allocated to a subgraph. However, after our historical literature review, we decided to extend the efforts of F. Harary and F. Buckley [2]. To understand the approach, we shall first define several center-related concepts.

The intuitive idea in CTR is suppose a graph is a pictorial representation of some physical object and the vertices are potential grip points. A balanced  $k$ -center is a set of vertices in this object, which if physically grasped at these  $k$ -center grip points, balances the load. For example, a 4-center of a large box might be the four corners of the box. Similarly, if the vertices in four subgraphs of a partitioning were all “close”, by some metric, to a pre-determined vertex in their respective subgraphs, the partitioning would be a good partitioning.

Center distances may be measured in a several means. We shall concern ourselves with two measurements, eccentricity and status. Eccentricity is defined as the longest shortest path from the vertex,  $v$ , to all other vertices in the graph. The eccentricity is often provided in tabular format. Each row and column intersection defines the distance between that pair of vertices. The status is the sum of the eccentricities for a given vertex,  $v$ .

A  $k$ -center is defined as the set(s) of vertices whose number of elements is  $k$ , and whose eccentricity is equal to the minimum eccentricity, or radius of the graph. For instance, a 3-center is the set of all sets of combinations of 3 vertices with the lowest eccentricity. A 1-center is the set of all nodes with the lowest eccentricity. Groupings based on status are compared to the summed eccentricity of the subgraphs. The premise is if we group those nodes closest to each other together, the overall delay will remain lower. Putting these steps together, the CTR is:

1. Set edge values to the sum of the end vertex delays and the inter-partition delay
2. Generate all-pairs shortest path table of underlying simple (undirected) graph
3. For each combination  $S$  of  $C(V, k)$  do
  - a. For each  $V_n$  in  $S$  create a subgraph  $G_n$  populated by  $V_n$
  - b. For  $V_i$  in  $V_n / S$  do
    - i. Find minimum distance to all members of  $S$
    - ii. Add  $V_i$  to the  $G_x$  with  $\min(d_t)$  and  $\min(|V|)$
  - c. Verify  $k$ -constraints for each  $G_n$  (optional)
    - i.  $M(|V|)$
    - ii.  $C(|E| = \text{sum}(\text{inputs, outputs, and edges}))$
  - d. Add inter-partition edges

For example, the following would be a result of finding the 4-center of the ISCAS 17 Graph, assuming  $M = 3$ ,  $C = 5$ , and  $D = 3$ . The minimum eccentricity for a 4-center is 19. One of the 4-centers with this eccentricity has a maximum delay of 11, along path {CFHK}. Subgraphs are labeled according to the vertices chosen to be the center cores from the eccentricity calculations.

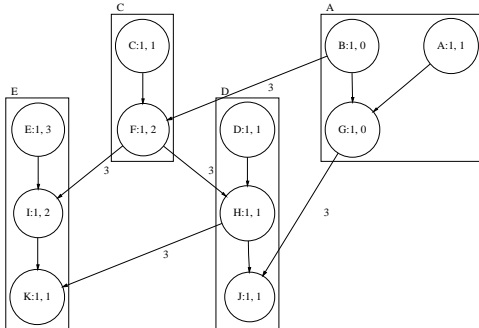


Figure 3: CTR Partitioning of ISCAS C-17

### 3. Implementation of Partitioning Algorithms

We used Maple V environment for our experimental study. Our experiments were conducted by partitioning graphs from various graph families against both simulated and real-world partitioning criteria. We used a wide spectrum of test graphs to evaluate the partitioning algorithms. There were 59 graphs, spread over 7 graph variants, with  $4 \leq |V| \leq 121$ . The simplest graphs to construct are caterpillar graphs. A caterpillar is defined as a path of  $x$  vertices, with  $y$  inputs and outputs for each vertex  $x_i$ . It is designed to be representative of electronic circuits.

We also worked with several graphs based on the unit square. Geometric grid and cross graphs have vertices spread equidistant from each other in a square, and are examples of certain types of array processing tasks, seen in both circuit design and parallel processing [6]. Geometric touch graph were also used. To construct one, we place  $n$  vertices randomly on the unit square and then specify a distance,  $t$ , which if two vertices are less than that

far apart, are connected. We added a modification to connect components such that a directed acyclic graph with a single component would be formed.

The edge density,  $d$ , for these graphs is based on the equation  $d = \pi * |V| * t^2$ , where  $t$  is the maximum geometric distance between vertices to have an edge connecting them. Four variants of the geometric touch graph were used: constant distance with variable edge density, variable distance with constant edge density percentage, variable distance with constant edge density, and constant distance where  $d = t = 1 / \pi$ .

Some additional key functions we implemented include a directed graph generator, directed graph inducer, transitive edge reducer, subgraph power set generator, vertex permuter, root determiner, topological sorter, Hasse diagram determiner and a scheduler. There were several other minor functions and metrics determiners written that are not discussed here. The directed graph generator allowed the input of a simple graph, and then based on vertex label names, the conversion of undirected edges to directed edges. This method guaranteed no cycles would be created and was used in our graph generators described later.

The ability to induce a copy of a graph was necessary, however, Maple's version did not copy edge or vertex attributes. We added this capability to our inducer, while retaining the capability to specify subgraphs or the entire graph. The transitive edge reducer determines, after a topological sort, which vertices are ancestors of the current vertex and then removes edges directly between the current vertex and ancestors via other vertices.

## 4. Experimental Results

Several constraints were adjusted to simulate various scenarios. We adjusted inter-partition delay,  $D$ , to simulate both small and large relative inter-partition delay delays. The number of partitions,  $k$ , was varied by adjusting subgraph  $|E|$  and  $|V|$  values. In addition, we experimented with the maximum cut to simulate objects with both small and large relative numbers of available inter-connections. In two cases, we set the parameters to simulate existing FPGAs, Xilinx's Spartan and Lucent's ORCA chips [12, 11]. We used several evaluation metrics to compare the algorithms described above. In addition, a control algorithm (RAN), executed for the root mean square computation time of the other algorithms, continuously evaluated random solutions. Based on our results, we determined the key metrics, as compared to the maximum delay, to be the graph family, inter-partition delay, sub-graph constraints for  $|E|$  and  $|V|$ , and the edge density,  $|E| / |V|$ .

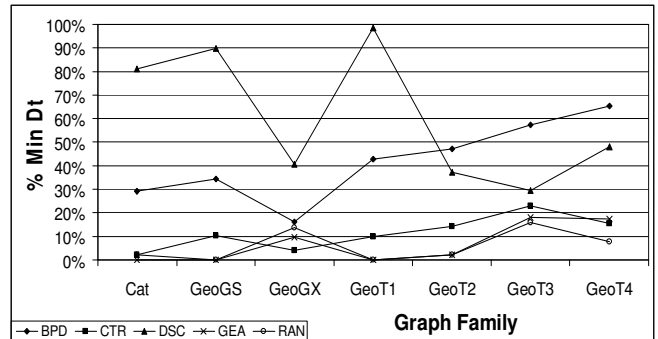


Figure 4: Min Delay for Different Graph Families

Experiments were conducted on seven different graph families using the minimum delay  $D_i$  as the main objective function. Summary results show that GEA and CTR performed

exceptionally well, as compared to BPD and DSC, while still obeying domain constraints (Figure 5). In addition, CTR provided partitionings most similar to the input graph.

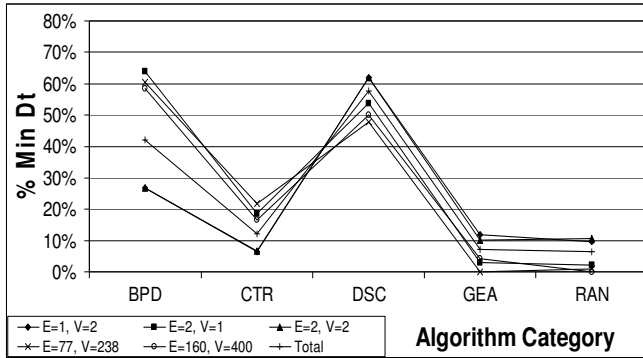


Figure 5: Algorithms vs.  $|E|$  &  $|V|$  Constraints

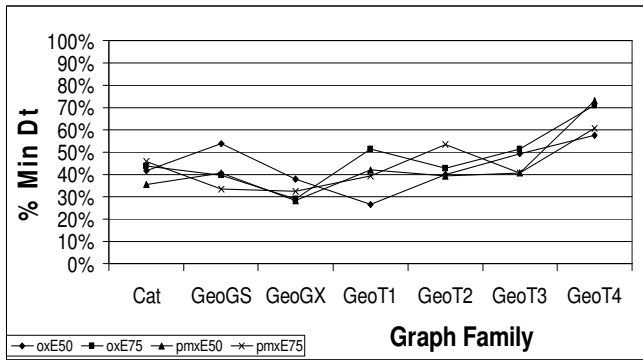


Figure 6: GEA Variants

For both GEA and CTR, we evaluated four variants of each algorithm. With GEA, we pre-selected the cross-over operator and choose either a 50% or 75% elitism rate. For our testing scenarios, 75% elitism and coupled with ordered crossover (oxE75) experienced the best performance, as Figure 6 shows. We also used partially mapped cross-over (pmx) and matched both cross-over operators with 50% and 75% elitism [5, 9].

For CTR, the root mean square minimum eccentricity (RIN) metric dominated results amongst its variants (Figure 7). More significantly, it never violated tested domain constraints. The other variants of CTR used were the root mean square maximum eccentricity (RAX), summed maximum eccentricity (SAX), and summed minimum eccentricity (SIN).

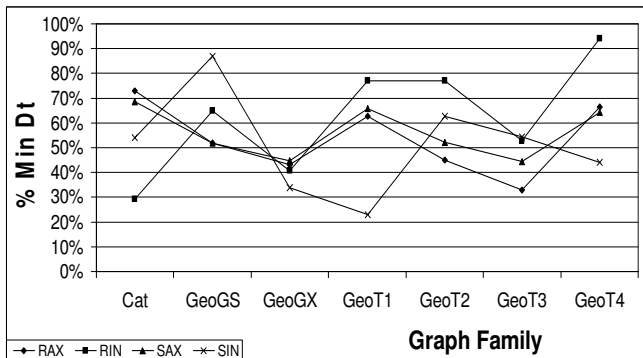


Figure 7: CTR Variants

## 5. Conclusions

Over the course of our research, we found few instances of inter-domain knowledge sharing. In other words, even though a given partitioning approach would work well in other domains, the framework did not exist to discuss the problem at an abstract enough level. From this observation, we determined the need for an abstract graph-partitioning model. Essentially, it is feasible to define the data set, constraints, and evaluation metrics in common mathematical terms. Thus regardless of domain, a taxonomy of various algorithms can be developed. For instance, an approach in database query result generation may be highly applicable in circuit design, and the model would permit us to determine this.

In summary, we have demonstrated metrics can be determined which allow us to examine graphs and partition them by formally specifying domain-specific constraints. Additionally, we have demonstrated the research and development capabilities of Maple. We have also, though not demonstrated here, applied geometric touch graphs to maze generation and have demonstrated GEA's ability for generating map 4-colorings.

Further research would include the translation of these algorithms to a compiled language, to facilitate evaluation of larger graphs. Any of these algorithms would benefit from further development. BPD would benefit from a replication control mechanism, DSC from a  $k$ -bounding technique, CTR from a faster method of evaluating center-oriented metrics, and GEA from alternative operators, optimal population size determination, elitism, and alternative fitness functions.

Finally, we are indebted to the MEDICIS lab at the École Polytechnique, for use of their computing clusters [8] and AT&T Labs for their open-source automated drawing tool, GraphViz [4].

## 6. References

1. D. Bryan. "The ISCAS '85 Benchmark Circuits and Netlist Format", MCNC <<http://zodiac.cbl.ncsu.edu/>>.
2. F. Buckley and F. Harary. Distance in Graphs. Addison-Wesley Publishing Company, 1990.
3. A. Gerasoulis and T. Yang. On the Granularity and Clustering of Directed Acyclic Graphs, Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE Computer Society Press, 1995. pg. 143-202.
4. GraphViz. <<http://www.research.att.com/sw/tools/graphviz/>>.
5. R. L. Haupt and S. E. Haupt. Practical Genetic Algorithms. John Wiley & Sons, New York, 1998.
6. S. Y. Kung. VLSI Array Processors. Prentice Hall, Englewood Cliffs, Pgs. 374-83, 1988.
7. E. L. Lawler, K. N. Levitt, and J. Turner, "Module Clustering to Minimize Delay in Digital Networks," IEEE Trans. on Computers. Vol. C-18 (1), pp. 47-57, Jan. 1969.
8. MEDICIS 2000. <<http://www.medicis.polytechnique.fr>>.
9. Z. Micalewicz. Genetic Algorithms + Data Structures = Evolution. Springer-Verlag, New York, 1994.
10. R. Murgai, R. K. Brayton and A. Sangiovanni-Vincentelli. "On Clustering for Minimum Delay/Area". Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD), pp. 6-9, 1991.
11. ORCA Series 2 Data Sheet. Lucent Technologies, Jun. 1999. <<http://www.lucent.com/micro/fpga/>>.
12. Spartan and Spartan XL Series Data Sheet. Xilinx, Jan. 1999. <<http://www.xilinx.com/products/spartan.htm>>.
13. D. F. Wong and R. Rajamaran, "Optimal Clustering for Delay Minimization," 30th ACM/IEEE Design Automation Conference (309-314), ACM, 1993.