

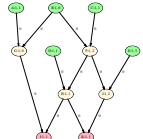
NEW GRAPH-BASED ALGORITHMS FOR PARTITIONING VLSI CIRCUITS

Chris Augeri
Department of Computer Science
United States Air Force Academy
christopher.augeri@usafa.af.mil

Hesham H. Ali
Department of Computer Science
University of Nebraska at Omaha
hesham@unomaha.edu

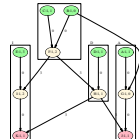
Abstract

It is often necessary to partition a complex circuit into subcircuits based on the available technology. One example would be when modeling a critical timing element within a gate generating a mask for the chip. To model the device, one often uses a set of PPGAs to implement it, thus requiring partitioning. It is desirable to minimize the number of PPGAs necessary to model the device, both for cost and efficiency. We have developed two novel, heuristic partitioning algorithms. One is partitioning-based, while the other is graph-based. Both algorithms are implemented and compared with known partitioning algorithms. Both of our algorithms can be used to optimize devices, while both the comparison algorithms are device-specific. We also used a random search algorithm to provide a control comparison to the new algorithms. A sample graph is used to represent the discussion.



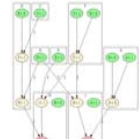
ISCAS C-17 (1985)
 $D_1 = 6$, {EIK, CFJK}

This graph is fitting given this venue, and is used in literature to describe the best predecessor algorithm. It partitions quickly, but is large enough to demonstrate each algorithm's key features.



Optimal
 $D_1 = 11$, {CFHK}

We enumerated all possible (15,400) partitionings to generate this optimal solution. The constraint is 3 groups of 3 nodes and 1 group of 2 nodes.



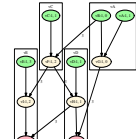
BPD
 $D_1 = 9$, {EIK, CFJK}

Best Predecessor, an existing algorithm, permits replication and is optimal. It targets circuit design.



DSC
 $D_1 = 9$, {DHK, CFJK}

Dominant Sequence Clustering, an existing algorithm, assumes an unlimited number of partitions and is optimal. It targets parallel processing.



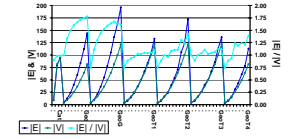
CTR
 $D_1 = 13$, {CFHK}

CTR (Center), our first algorithm, groups nodes based on eccentricity values. This algorithm appears to show more promise than GEA and its domain-independent.



GEA
 $D_1 = 12$, {EIK, CFJK}

GEA (Generic Algorithm), our second algorithm, uses a new encoding scheme we developed and a robust fitness function. It is also intended to be domain-independent.



Input Graph Summary

This chart relates information regarding the graphs we used and their characteristic. Note, we used graphs from 4 different families (Caterpillar, Unit Grid, Unit Grid-Cross, and Random Unit) graphs. Caterpillar graphs have a central spine with legs on one side and outputs on the other. The unit grid is a matrix of connected 4-cycles, the u grid cross is a matrix of 4-vertex complete graphs, and the random unit graph is a graph of randomly connected vertices with varying densities.

GEA

Genetic algorithms have been used for some time in graph partitioning. Previous individuals are produced via vertex extending schemes and operators. The other encoding scheme, which is a hybrid of two existing ones and offers the benefits of both. In addition, we demonstrate the flexibility of our fitness function and generally described, and introduce the concept of migration. This is necessary to the crossover/recombination process. The crossover/recombination process of individuals in the search space of the search space, rather than with GEA would include implementing in a complex language, and experimentation with the fitness function.

Sorted Vertex IDs	A	B	C	D	E	F	G	H	I	J	K					
Group Number Encoding	4	1	1	3	2	1	4	3	2	4	2					
Permutation w/Separators	B	C	F	;	E	I	K	;	D	H	;	A	G	J	;	
Separated Permutations	B	C	F	E	I	K	D	H	A	G	J	-	3	3	2	3

Encoding Schemes

Previous literature describes group number and permutation with separator encodings (rows 2 and 3 above). We introduce the separated permutation encoding scheme. This allows offspring generation to remain uncorrupted in an issue with permutation with separators, and improves the access speed, a problem with both of the existing schemes. Note, a node's position within the encoding has no impact - it is only an issue with the cluster it is assigned to.

Generate initial population
While generation limit not reached
Create Offspring: cross-over, mutation, migration
Evaluate offspring via fitness function
Select offspring: elitism and random selection

GEA Algorithm Pseudo-Code

Our genetic algorithm is fairly normal. We generate a random population and then use a number of operators to generate subsequent generations. We do use an operator we termed migration, to simulate "movement" in the real world. It is really just the occasional introduction of a randomly-produced offspring. It may refer to it as gross mutation. We then use a combination of elitism and random selection to choose the next generation. This process is repeated until a stable generation is reached or the maximum time has elapsed.

$$G_i = \begin{cases} C_i/C_j > 1C_i/C_j \\ C_i/C_j \leq 1C_i/C_j * C_j \end{cases} + \begin{cases} D_i/D_j > 1D_i/D_j \\ D_i/D_j \leq 1D_i/D_j * D_j \end{cases}$$

$$\begin{cases} W_i/W_j > 1W_i/W_j \\ W_i/W_j \leq 1W_i/W_j * W_j \end{cases} \quad C_i + D_i + W_i = 1$$

GEA Fitness Function

Our fitness function uses a combination of individual partition data on whether this individual obeys the partitioning constraints, along with its overall delay value. This provides both a micro and macro evaluation of the individual. Rather than destroying individuals which violate the constraints, they will eventually be weeded via a combination of elitism and random selection. This simplifies creating following generations. The current fitness function compares the actual cut, vertex count and delay to the max limits.



GEA'S 4-Coloring of the U.S.

This demonstrates the flexibility of the genetic algorithm we developed. We used it to allocate our jobs (the equal piles problem) to various servers of the MEDICIS cluster via secure shell. The 4-coloring fitness function modification averages a color's surface area, the linear distance shared with other colors, and the number of objects which are that same color.

CTR

The CTR algorithm essentially groups values based on their longest paths from other vertices. We use eccentricity to determine the longest paths. We use the information to add the remaining vertices. The remaining vertices on the CTR algorithm would include a better selection criteria, where adding vertices, speeding up the eccentricity computation, and alternative seed vertex selection systems. The iterative time is also in keeping an object as a parent in Japan.

	A	B	C	D	E	F	G	H	I	J	K	Max	RMS
A	0	1	1	1	1	1	1	1	1	1	1	12	10.526
B	1	0	1	1	1	1	1	1	1	1	1	12	10.526
C	1	1	0	1	1	1	1	1	1	1	1	12	10.526
D	1	1	1	0	1	1	1	1	1	1	1	12	10.526
E	1	1	1	1	0	1	1	1	1	1	1	12	10.526
F	1	1	1	1	1	0	1	1	1	1	1	12	10.526
G	1	1	1	1	1	1	0	1	1	1	1	12	10.526
H	1	1	1	1	1	1	1	0	1	1	1	12	10.526
I	1	1	1	1	1	1	1	1	0	1	1	12	10.526
J	1	1	1	1	1	1	1	1	1	0	1	12	10.526
K	1	1	1	1	1	1	1	1	1	1	0	12	10.526
Max	12	12	12	12	12	12	12	12	12	12	12	144	12.833
RMS	10.526	10.526	10.526	10.526	10.526	10.526	10.526	10.526	10.526	10.526	10.526	12.833	10.526

ISCAS C-17 Eccentricity Matrix

This table shows the all-pairs shortest path information for the ISCAS C-17 graph. This was computed based on each edge being equal to the delay of the nodes it is incident to, it's own delay, and the inter-partition delay (worst-case, each vertex in it's own partition). The last 2 columns are the eccentricity and root mean square delay for each node. We also experimented with the status (summed delay).

Compute all-pairs shortest paths for the graph
Select S, where S will be the set of k seed vertices
For each V_i in S do Partition Assignment Loop
Add each V_i in V / S do Partition Assignment Loop (right)
For inter-subgraph edges for best partition solution(s)

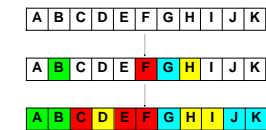
CTR Algorithm Pseudo-Code

The CTR algorithm, after computing the previous matrix, uses a number of selection criteria to create the seed partitions. As the results below indicate, the minimum root mean square delay across the nodes provided the best results. After these partitions were created, nodes are added based on their delay to each of the seed vertices and how full a partition is. This step needs the most improvement. Once nodes are allocated, the necessary inter-partition edges are added.

Determine minimum distance to all S
Add vertex to subgraph of S, based on:
delay to seed vertices
vertices in seed subgraphs
Verify k-constraints still obeyed

Partition Assignment Loop

The allocation of each node is a complicated and non-vertex-performed process. In addition to deciding which partition a node is allocated to, we also determine if it will violate constraints when added (k-vertices, cut size, etc.). We currently average the number of vertices already in a partition too strongly in the allocation process.

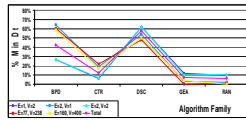


CTR Algorithm Phases

Each row depicts a CTR algorithm phase. The first row represents determination of the all-pairs shortest path information. The second row indicates selection of the seed nodes, and the bottom row matches the solution depicted in the first row of this poster.

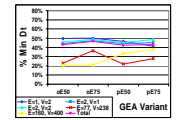
Results

We provide a subset of our key results. The relevant charts compare each algorithm's average performance and the specific variants with respect to the minimum delay after partitioning. The middle graph shows how each algorithm performs on different graph families (or datasets) relative to partitioning constraints. The rightmost chart compares the new algorithms and their variants graph families. The bottom graph compares each graph by each algorithm and a series of partitioning constraints. We also provide a comparison of the new algorithms and their variants graph families. The bottom graph compares each graph by each algorithm and a series of partitioning constraints. We also provide a comparison of the new algorithms and their variants graph families.



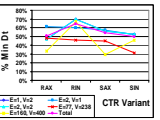
Algorithm Family vs. Min D_1

This chart compares the collective results of each algorithm against whether it found the minimum possible delay after partitioning. Given the advantage DSC and BPD have, they often performed "better". However, in the context of obeying the partitioning constraints, our algorithms performed well. Future research would include comparison against sub-optimal algorithms which obey constraints.



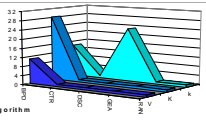
GEA Variants vs. Min D_1

This details the individual scenarios GEA was tested in. We created 4 variants, each a different selection criterion during seed vertex determination. The minimum root mean square delay performed best on average. The OX operator with 75% elitism was best on average.



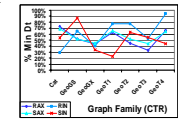
CTR Variants vs. Min D_1

This details the individual scenarios CTR was tested in. It compares their performance across the respective graph families. Besides pure graph metrics, this was one of the best ways to categorize an algorithm's performance.



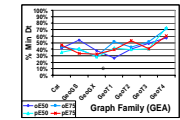
Constraint Analysis

This chart depicts how BPD and DSC do not obey the partitioning constraints, in some cases excessively. The increased components for both BPD (from replication) and DSC (unlimited components) is readily apparent. Our algorithms always provide at least one solution which obeys constraints.



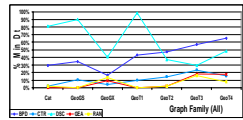
Graphs vs. Min D_1 (CTR)

This details the individual scenarios CTR was tested in. It compares their performance across the respective graph families. Besides pure graph metrics, this was one of the best ways to categorize an algorithm's performance.



Graphs vs. Min D_1 (GEA)

This details the individual scenarios GEA was tested in. It compares their performance across the respective graph families. The OX cross-over operator with 75% elitism performed best.



Graphs vs. Min D_1 (ALL)

One can clearly see the wide variance each algorithm experiences, depending on the graph family a graph belongs to. One goal of our (not completed) is to build a taxonomy of partitioning algorithms based on both the density of interest and the type of graph or a graph's metrics.

Summary

In sum, we considered research of partitioning algorithms, their potential, provide a new algorithm (CTR), and updated constraints to an existing algorithm (GEA). Opportunities for future research include comparing our systems, implementing both algorithms in a compiled language, experimenting with improved seed vertex selectors for CTR, and improving GEA's fitness function.

Acknowledgements

Ecole Polytechnique MEDICIS Lab

AT & T's GraphVis
Maple V