

On Using Maple V
to Partition
Directed Acyclic Graphs

CHRISTOPHER J. AUGERI
United States Air Force Academy

HESHAM H. ALI
University of Nebraska-Omaha

Maple Summer Workshop
Waterloo, Canada
July 29th, 2002

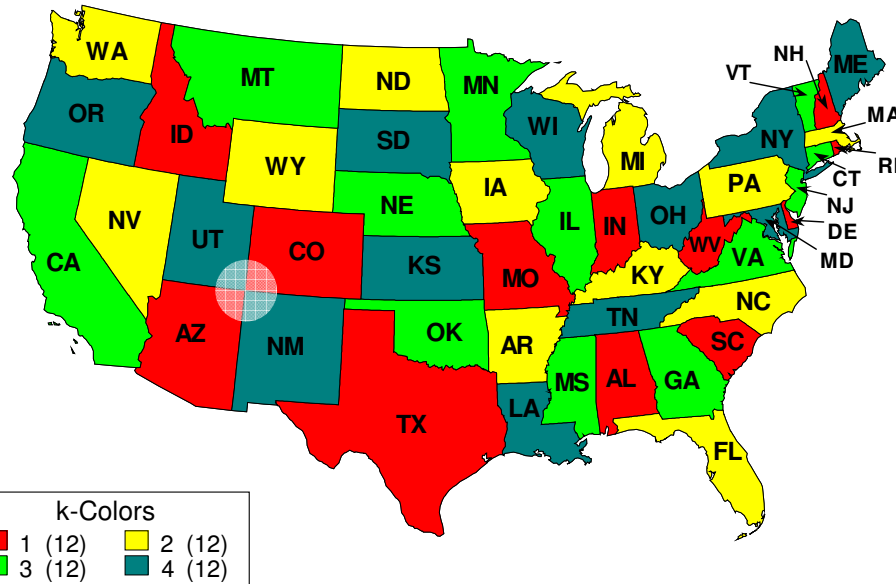
Graph partitioning involves decomposing a relational graph into k sub-graphs, subject to domain-specific constraints. Applications include database querying, map coloring, bin packing, service location determination, electronic circuit design, and parallel processing. Constraints include the number and size of partitions, the number and delay of inter-partition connections, and the permitted edge and vertex replication levels.

We have developed, within Maple, two k -way bounded partitioning algorithms; one is genetic-based, while the other is a hierarchical graph center-based method. These algorithms are compared with ones from the electronic circuit design and parallel processing domains, best predecessor and dominant sequence clustering, respectively. In addition, a random partitioning algorithm is used for control purposes.

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 straightforward abstract graph partitioning model. This model allows one to specify the key evaluation metrics and control parameters from a mathematical perspective, permitting inter-domain comparison of algorithms and allowing one to identify the particular scenarios they are best applicable to.

Other related areas given some attention include optimal partitioning, map coloring, job allocation, maze generation and graph visualization. All research was conducted within the C/C++, Tcl/Tk and Maple environments, with an emphasis on Maple. In particular, all partitioning execution was done remotely on the École Polytechnique's mathematics research computing cluster (Medicis) in Palaiseau, France.

48 Contiguous States of the U.S.A. (4-coloring w/area & edge balancing)



This is an alternative application of our Genetic/Evolutionary algorithm. It is a 4-coloring with 65% elitism, 12 states for each color of the 48 contiguous US states. There was a 90% weighting on balancing adjacent edges for the other 3 colors and a 10% weighting on balancing the total geographic area covered for each color.

Notice the adjacent four corners of Arizona, Colorado, Utah & New Mexico. These edges, as listed in the graph each of have a distance of 0 and are transparent to the cut-detection routine of the algorithm. This could be modified, but was a good illustration of the various subtleties in graph partitioning. It was accomplished in approximately 10.5 hours on our Alpha-based machine, Apollo.

Research Motivation

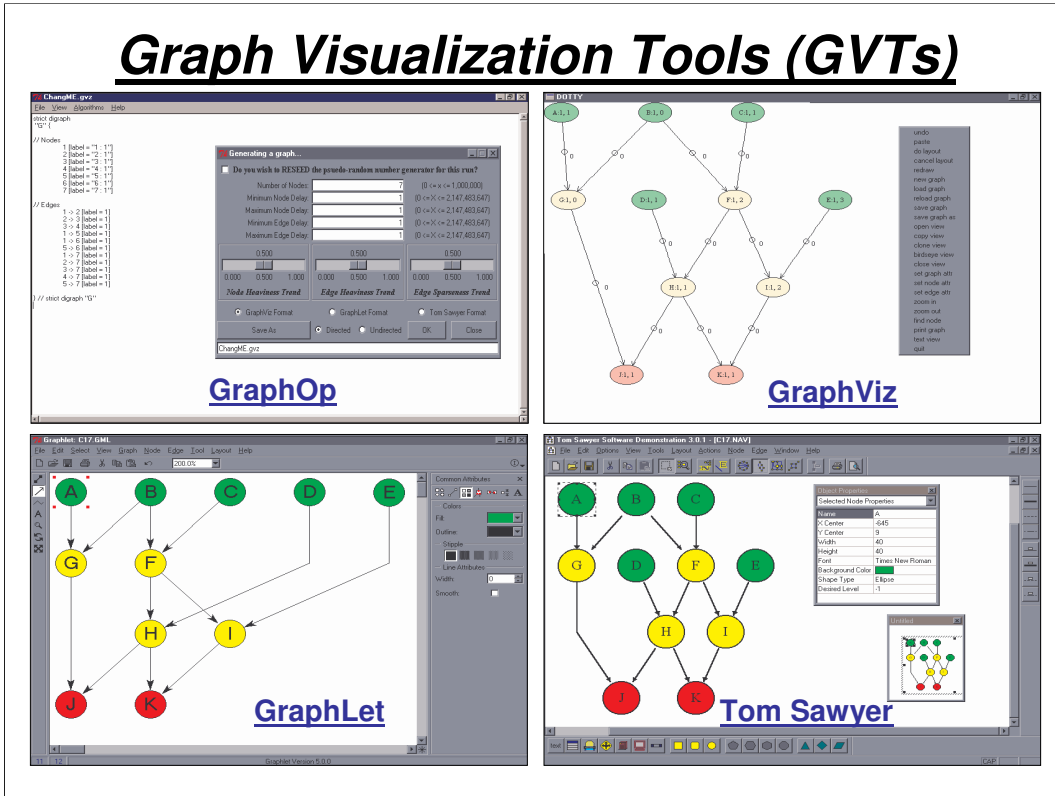
- **Motivation for Graph Partitioning**
 - used & studied in many domains
 - is NP-Hard
 - Decomposing a graph G into an ordered pair, P , containing
 - Set of induced subgraphs of G
 - Edge set for connected vertices in 2 subgraphs of G
- **Objectives**
 - Demonstrate need for an abstract graph partitioning model
 - Genetic and Center-Based algorithms
 - Evaluate graph visualization tools (GVTs)
- **Contributions: GraphPar, GraphOp, & PRRS**
- **Maple V**
 - Provides well-developed graph primitives
 - Set operators conducive to algorithm development

Motivation: Partitioning is used in many application domains. For instance, we may desire to cluster workstations to create a high-end parallel processing environment, assign tasks to employees, group related data after observing database query similarities, gather items which are un-related, e.g. graph coloring, or minimize the number of shipping containers to ship a set of objects

Objectives: Our primary objectives are to demonstrate two new algorithms, along with demonstrating the need for a mathematical model to classify partitioning requirements and algorithms. In addition, we also offer a brief analysis of visualization tools.

Contributions: We developed a number of tools to facilitate our research. The primary tool is a well-developed partitioning code base, *GraphPar*, for the Maple V mathematics research package. This tool includes the genetic and center-based algorithms we developed. Also developed during this research was a Tcl/Tk-based GUI front-end, *GraphOp*, to join together various aspects of this research: graph generation, graph partitioning and graph visualization. A prototype graph visualization tool was also experimented with, based on Ousterhout's sample script. Included is a discussion of three graph visualization tools, both compared and used in this research. Another contribution of this research is the Project & Reference Research System, *PRRS*, useful for tracking research oriented projects.

Graph Visualization Tools (GVTs)



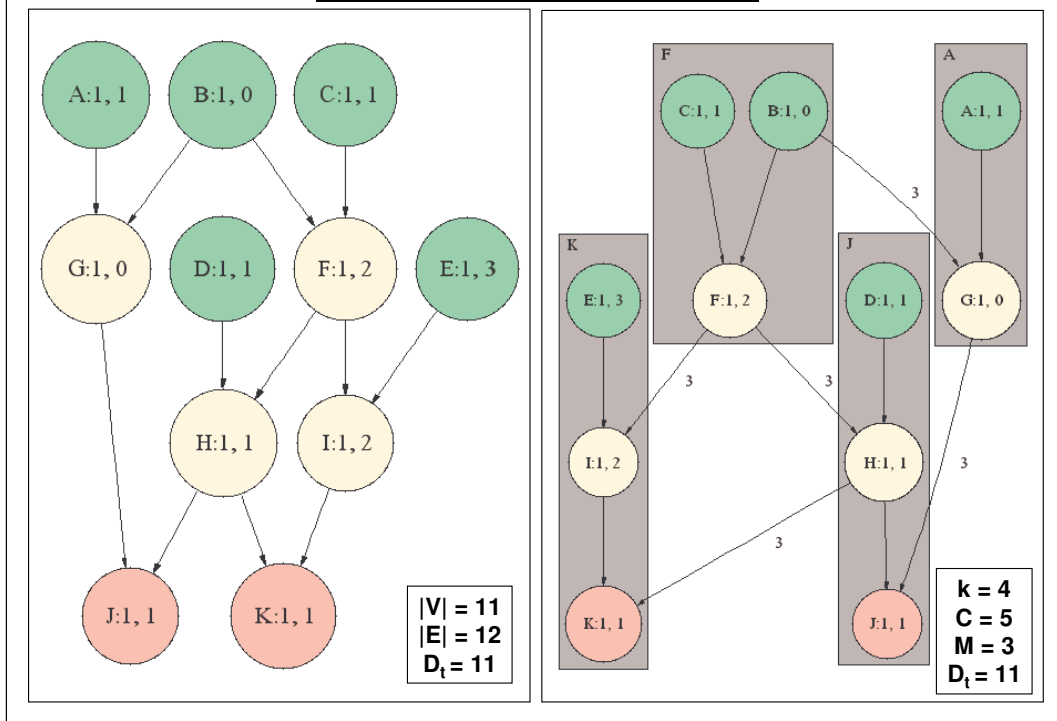
We now present a sampling of the large number of graph visualization tools (GVTs) we used in the course of our research. The upper-left shows *GraphOp*, our Tcl-Tk-based tool. It allows us to directly generate random graphs, edit them, and send them to our various visualization tools.

GraphViz, from AT & T Bell Labs, functions in a UNIX-style pipe & filters fashion, although it does have some interactive GUI support. Overall, it proved to be the most useful tool, in particular, it had direct support for partitioning. However, it had the worst user interface, to include the requirement for a 3-button mouse. It also will output in various graphics file formats and can be run from the command line.

GraphLet, a Tcl-Tk-based tool from the Univ. of Passau, offered the most graph manipulation algorithms, however, it was the slowest interface and offered little partitioning support.

Tom Sawyer is a commercial product, and offered tremendous layout speed and significant control over the type of layout desired. It did have partitioning support, although it was accessible only via individual subgraphs.

ISCAS 1985 (C17)



The left image depicts the ISCAS '85 C17 circuit. We first came while implementing Rajaraman & Wong's Best Predecessor algorithm during a VLSI design course. The alphabetic designators are for node identification. Green & red nodes indicates a primary input or primary output, respectively.

The yellow nodes are interior nodes. This circuit will be used to demonstrate our new algorithms and the older methods used for comparative purposes. The maximum delay is 6 (along paths $E \rightarrow I \rightarrow K$ and $C \rightarrow F \rightarrow I \rightarrow K$).

C17 is a directed, acyclic graph (DAG) with 1 component and no transitive edges. However it is not a tree (e.g node K has two parents, H & I). There are a total of 11 nodes and 12 edges. All edges are assumed to have a delay of 0.

The right image shows an optimal partitioning of ISCAS '85 C17, as displayed in our primary GVT, *GraphViz*. It is optimal for delay, cut per partition, and total cut for all paths. It was computed via direct examination of *all* 15,400 possible partitions for this graph with $D = 3$ (inter-cluster delay), $k = 4$ (# of clusters), $C = 5$ (cut) & $M = 3$ ($G_n(|V|)$). This is computed by $C(11, 3) * C(8, 3) * C(5, 3) / P(3)$. The max delay in this partition is CFHK (11).

BPD (Best Predecessor): VLSI Design

- UDM: unit delay model (1960s)
- GDM: general delay model (1990s)
- Generation: bottom-up
- Replication: a graph of 1,000 vertices can become a partition of ~10,000 vertices.
- In original form & with replication, the resulting partition is optimal for a single constraint.
- We extended the algorithm for two constraints, however, it is non-optimal.

The best predecessor algorithm was developed by Rajaraman & Wong (30th ACM/IEEE Design Automation Conference). It is based on the work done on circuit partitioning in the late 1960s by Lawler, Levitt and Turner. The late Dr. Eugene Lawler was a noted mathematician, then working with the Univ. of Michigan.

Essentially, one first computes the longest delay based on each nodes predecessors. After this, working bottom-up, you choose the best nodes for each primary output. Any external vertices are added to the pool and the process is repeated. This technique is optimal for a single monotonic constraint. We implemented a non-optimal second constraint for cut limitations.

Pseudo-Code

- Labeling Phase
 - Generate maximum delay from output output
 - For each vertex, in topological order
 - Create subgraph as single member
 - Add vertices until M is exceeded
 - Remove vertices until E is met
- Clustering Phase
 - S = {POs}
 - While S <> {}
 - Add partition for a member V of S and remove from S
 - Add all external incoming vertices to P_v to S
- Add external edges to graph (unused subgraphs are discarded)

DSC (Dominant Sequence Clustering): **Parallel Processing & Scheduling**

- Assumes maximally connected & infinite set of computing devices (e.g. CPUs or nodes).
- Generation: top-down
- Partitioning & scheduling capabilities
- Currently, no provision for
 - replication
 - k-constraints

The Dominant Sequence Clustering algorithm was developed by Drs. Gerasoulis & Yang @ Rutgers University. It is primarily designed for the parallel processing domain. Therefore, there is a strong emphasis on maintaining linear clusters, as a processing element can only execute one task at a time.

This is a top-down iterative technique. Nodes are maintained on a pair of lists, free and partially free. Free nodes are those that have all their ancestors assigned to a partition, partially free nodes have at least one ancestor processed. Only nodes from the free list may be added or used to create a partition.

Pseudo Code

- $EG = \emptyset$; $UEG = V$
- Compute *blevel* for each node & set *tlevel* = 0 for each free node.
- While $UEG \neq \emptyset$ Do
 - $n_x = \text{head (FL)}$; // free task w/highest priority
 - $n_y = \text{head (PFL)}$; // partial free task w/highest priority
 - If ($\text{PRIO}(n_x) \geq \text{pPRIO}(n_y)$) Then
 - Call the minimization procedure to reduce *tlevel*(n_x)
 - If no zeroing accepted, n_x remains in a unit cluster
 - Else
 - Call the minimization procedure to reduce *tlevel*(n_x) under constraint DSRW
 - If no zeroing accepted, n_x remains in a unit cluster
 - ENDIF
 - Update priority of n_x 's successors and place n_x in EG
- ENDWHILE

GEA: Genetic & Evolutionary Algorithm

- *Controlled* random search
- Generation, History & Population (G_n , H_n & P_n)
- Elitism: retain specified % of “fittest” solutions
- Flexible, e.g. RAN control algorithm, U.S. Map
- Encoding Scheme: [ABCDEFGHIJK]
 - Group number: [12121324344]
 - Permutation w/separators: [CAE ; DGB ; FI ; HKJ]
 - Permutation & separators: [CAEDGBFIHKJ] - [3 , 3 , 2 , 3]

• Operators

- *Migration*, Mutation, Cross-Over

- Intelligence: Low High

- Scope: Narrow Broad

Cross-Over

	1	2
Parent	GKICDHJABEF	CIKFEDHAGJB
OX	ICJFEDHABGK	KFECDHJAGBI
PMX	GKIFEDHABDC	FIKCDHJAGEB

Genetic, or evolutionary, algorithms are essentially a controlled random search. The initial population is formed by a random string permutation of a legal solution. Primary control variables include the # of generations, how long to keep a history before assuming no improvement is likely and the population size. Elitism is the level of good solutions kept before randomly selecting the rest of the next generation’s population.

Genetic algorithms are flexible, e.g. our implementation can be used for job allocation or map coloring as well. The group # scheme uses the same positions and replaces the position with the marker indicating its group membership. The permutation w/separators uses the values of the string and embeds separators in the string. We propose a new encoding scheme, permutation & separators, combining these schemes. It allows us to use genetic operators without any exceptions, and is not necessary to scan the entire string to determine a partition(s).

Migration is a technique we introduced that creates a number of random individuals in the same manner as the initial population. This serves to eliminate the potential local optimum stagnation that can occur with genetic algorithms. It is essentially a permutation of any solution. Mutation is the process where certain percentage of random pairs of data locations are swapped. A relatively small percentage is necessary to achieve the desired effect.

There are two types of cross-over operators used, OX & PMX. In both cases a specific segment from two parents is selected. In OX, starting w/the right-side of the segment and wrapping around, the remaining items are added to the offspring, preserving their relative order. With the PMX operator, the same position (& therefore, order) is maintained for as many as possible of the remaining elements. Any other members are functionally mapped until a unused element is reached.

GEA: Pseudo-Code

- Generate initial population (from k-constraints)
- While $G_i \neq G_n$ & $H_i \neq H_n$ do
 - Create Offspring
 - Cross-over ($P_n / 2$)
 - Mutation (P_n)
 - Migration (P_n)
 - Evaluate *offspring* via weighted equation
 - Selection
 - Elitism: $E_r * P_n \quad E_n$
 - Parental Selection: $(P_n - E_n) * PS_r \quad PS_n$
 - Random Offspring Remnant: $(P_n - E_n - PS_n)$

During an experiment, the initial population is generated, based on the k-constraints for the run. The generation and history values dictate the number of times the algorithm will execute, based on the resulting populations. During each iteration, a specified number of parents are selected for cross-over and mutation. Additionally, a certain number of migrations are generated.

All new individuals are evaluated by a value & feasibility equation. This equation is domain dependent and in our runs measured the k-constraints and delay value. The next generation is formed by selecting a certain percentage of elite members, while the remainder are randomly selected from the parent generation.

CTR: k-Centered Partitions

- What is a center?
 - Eccentricity: Longest Shortest Longest path
 - Summed Min/Max vs. RMS Min/Max
- What is a k-center?
 - $k = |\{ \}$
 - all combinations: $(|V|, k)$
 - Finding a single center is $O_t(n)^3$
- Useful for small k, however, combinatorial choices for large k is prohibitive.
- May be able use a sample of V for large k

Our second algorithm, CTR, is based on metrics to determine centric values of graphs. The specific metric chosen here is eccentricity, or the longest, shortest, longest path from a vertex to all other vertices in the underlying simple graph. The two versions studied used both the maximum and summed eccentricity of the subgraphs for comparative purposes between potential k-center solutions.

A k-center solution is a set of k vertices, each in a different subgraph and populated with the remaining vertices. To examine all solutions for a graph, one must evaluate all combinations of $(|V|, k)$. We used four variants of this algorithm, the combined options for pure minimum and maximum, along with the RMS evaluation.

The algorithm, albeit slow, for the k-value of 2, but becomes prohibitive as larger k values are desired. One alternative, unexamined here, is to use a selective & small sample of V, limiting the number of possibilities to be equal to $(|V|, 2)$. For example $(18,4) = 3,060 < (81,2) = 3,240 < (19,4) = 3,876$.

CTR: Pseudo-Code

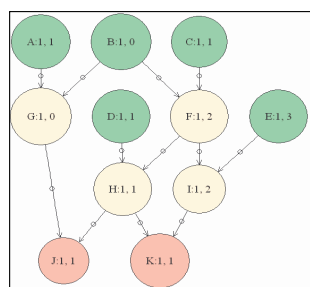
- Generate all-pairs shortest path of underlying simple (undirected) graph
- For each combination S of $C(|V|, k)$ do
 - for each V_n in S create a subgraph populated by V_n
 - for V_i in V_n / S do
 - find minimum distance to all members of S
 - add to subgraph of vertex $w/\min(d_i) \& \min(|V|)$
 - compare $w/\max(\text{eccentricity})$
 - verify k -constraints for each G_n
 - $M: |V|$
 - $C: \text{Cut (inputs, outputs, \& edges)}$
 - if *valid*, compare $w/\text{current min(eccentricity)}$
- Add inter-subgraph edges for best partition solution(s)

First, the all-pairs shortest path distance is found for the underlying simple graph. We then iterate over all combinations of (V, k) . For each combination we form partitions where each element of S is allocated to an empty subgraph. For the remaining vertices, allocate them to the subgraph whose center vertex is “closest” based on the all-pairs shortest path distance.

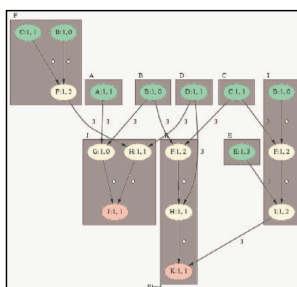
If there is greater than 1 subgraph, the one with the lowest $|V|$ value is used. In the event of a tie, one of these subgraphs is randomly chosen & updated. This process continues for all remaining vertices. The eccentricity of the subgraphs is set to either the max eccentricity within it or the summed eccentricity of the vertices in it.

The k -solutions are compared based on the lowest values for the max or summed eccentricity. After forming a k -partition, the cut constraints must be validated for the subgraphs in it. If valid and minimal, the inter-subgraph edges necessary are added to the solution.

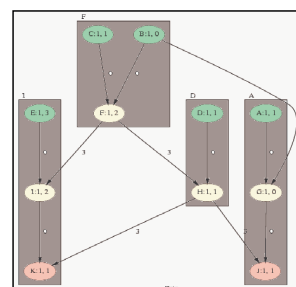
ISCAS '85 C17 Partitionings



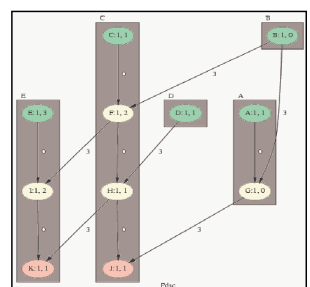
C17, $D_t=6$



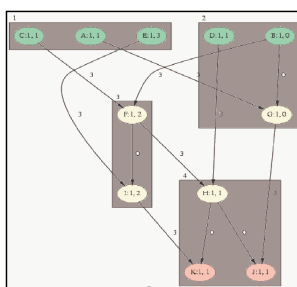
BPD, $D_t=12$



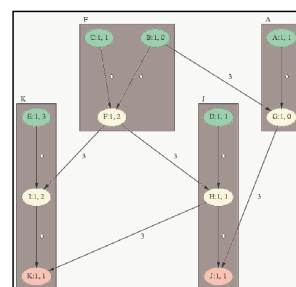
CTR, $D_t=11$



DSC, $D_t=12$



GEA, $D_t=12$



Optimal, $D_t=11$

This slide shows C17 partitioned by each algorithm --the optimal partitioning is repeated.

BPD: replication & independent partitions. As shown here, this algorithm adds subgraphs for those outside of the current subgraph. This process leads to a non k-constrained solution that also incorporates vertex & edge replication. Additionally, the algorithm permits non-linear clusters, e.g. {B,G,J}, beneficial in the VLSI domain. The maximum delay is CFIK (12).

CTR: aesthetically pleasing, linear & non-linear, note is similar to optimal solution. Again, this algorithm, like GEA, does not currently support replication. It also has no limitation with regards to linearity, although the vertices will tend to be connected within a subgraph, by the very nature of the algorithm, i.e. closer nodes are more likely to be allocated to a subgraph. The maximum delay is on CFHJ, where $D_t = 11$.

DSC: linear, non k-limited. Note all clusters are linear, i.e. they have no independent tasks. This is driven by the domain it is typically used in, parallel processing, where an independent task could potentially be performed on a different CPU. This algorithm is not limited to linear cluster, but it tends to primarily generate linear clusters. Also, it does not obey k-constraints, but it does not allow replication either. The maximum delay here is 8 (BFIK).

GEA: linear & non-linear, no specific structure or approach (top, bottom, in out). GEA has no limitations with regards to linearity. The maximum delay is 12 (EIK).

Input Metrics

1	G_f	Graph Family
2	$\sum (\text{deg}(G))$	roots
3	$\min ((a, \dots, b)_{dt})$	min delay along any path
4	$\max ((a, \dots, b)_{dt})$	max delay along any path
5	$ E , V $	# edges & # vertices
6	$\sum E_{wt}, \sum V_{dt}$	sum of edge & vertex delays
7	$\sum V_{wt}$	sum of vertex weights

The various metrics chosen for input, constraints and output were based on graph theory constructs and existing research in graph partitioning. Several metrics, in our final analysis, were not critical factors of separation and are not included in future slides.

1. Graph Family = (Caterpillar, Geometric Grid, Geometric Grid-X or Geometric ($T_1 - T_4$))
2. Sum of primary input & output vertices
3. Shortest path from any input to any output
4. Longest path from any input to any output
5. # edges & vertices (respectively)
6. Sum of edge & vertex delays (respectively)
7. Sum of vertex weights (e.g. space it would need on a VLSI device)

Parameters

1	D_f	Domain Family
2	P_a	Partition Algorithm
3	C	max cut/partition (including root vertices)
4	D_{max}	max delay along any path, assuming worst-case multi-partite partitioning
5	M	max vertex weight/partition
6	K	# of desired partitions
7	D	inter-partition delay constant

1. Domain of interest (e.g. VLSI Design (circuit) or Parallel Processing (schedule))
2. Partition algorithm to use (BPD, CTR, DSC, GEA, or RAN)
3. Cut limitation for subgraphs (# inputs, interconnections, & outputs)
4. Essentially, $\sum(V_{dt}) + D * |E|$, assumes every edges becomes an interconnect
5. $\sum(V_{wt})$ per subgraph
6. # of desired partitions or pieces to divide the graph into
7. Delay value when going between subgraphs

Output Metrics

1	$\kappa(G)$	# components in partitioned graph
2	$\max (\text{deg}(P))$	max cut/partition (including roots)
3	$\sum (\text{deg}(P))$	sum of partition cuts
4	$\min ((a, \dots, b)_{dt})$	min delay along any path
5	$\max ((a, \dots, b)_{dt})$	max delay along any path
6	$ E , V $	# edges & # vertices (including replicas)
7	$\sum E_{wt}$	sum of edge delays
8	$ P $	# partitions
10	$\sum V_{dt}$	sum of vertex delays
11	$\sum V_{wt}$	sum of vertex weights
12	$\sigma (V_{wt})$	standard deviation of partition vertex weights sums
13	$\max (\sum V_{wt})$	max of partition vertex weights sums
14	$O_s(n)$	processing space (bytes)
15	$O_t(n)$	processing time (seconds)

1. # of individual (or disconnected) graphs in final partitioning
2. max cut for all subgraphs (if $\leq C$, is a valid partitioning)
3. sum of subgraph cut values
4. Shortest path from any input to any output (in partitioned graph)
5. Longest path from any input to any output (in partitioned graph)
6. # edges (including potentially new replicas) & # vertices (respectively)
7. sum (E_{wt})
8. # of partitions (if = k satisfies k-partition constraint)
9. # vertices (including replicated vertices)
10. sum of vertex delays
11. sum of vertex weights (space || area)
12. standard deviation of sum of V_{wt} , indicates how evenly vertices are distributed
13. max sum of subgraph vertex weights
14. # bytes of memory were processed by CPU during the run
15. # seconds used by CPU for this run

Input Graph Specifications

- 1-Component DAG: “Connected” Directed Acyclic Graph
- Transitive Reduction: eliminate direct ancestors if inherited
- $4 \leq |V| \leq 1024$
- $V_{dt} = V_{wt} = 1$
- $E_{wt} = 0$ (within a partition)
- $E_{wt} = D_t = [10, 1000]$
- Families: Caterpillar, Geometric (Grid, Cross, & $T_1 - T_4$)

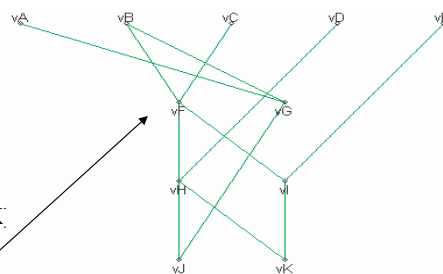
Partition Category	C	M	K
E & V-bipartitioning	2	2	2
V-bipartitioning	1	2	2
E-bipartitioning	2	1	2
Xilinx Spartan XCS05/XL	77	238	$\max(E_n/77, G_s[V_{wt}]/238)$
Lucent ORCA 2C	160	400	$\max(E_n/160, G_s[V_{wt}]/400)$

Summarized here are primary requirements and controls used in our experiments. Experiments were limited to single component DAGs that were transitively reduced. Graph size was limited to 1,024 vertices and the delay for each vertex was assumed to be 1, with an internal partition delay of 0 (UDM, as in BPD). External delay was simulated at two different orders of magnitude, 10 & 1,000.

These experiments were conducted across seven graph families. The families were selected based on previous research using these same families in VLSI Design and Scheduling simulations. The Xilinx & Lucent technology mapping was not conducted for all algorithms, primarily CTR, due to a combinatoric increase in execution time. They were selected as representative FPGAs to partition onto for the graph size we were working with.

Implementation & Execution

- Maple: 10-1000x slower than C/C++
- Maple kernel issues: RAM, Windows OS stability?
- MEDICIS
 - 124 graphs/7 families
 - ~206 TB of data
 - ~365 days of CPU time
 - ~11,000 data points
- GraphPar (support)
 - Transitive edge reducer
 - Vertex permuter
 - Topological sorter
 - [vA, vB, vC, vD, vE, vF, vG, vH, vI, vJ, vK]
 - Hasse diagram determiner



Results analysis is based on the *GraphPar* routines we implemented (1,600 lines of source code). We experienced crashes with the Maple kernel on any Microsoft Windows OS and platform w/in 36 hours. However, the leon* Alpha-based machines at MEDICIS, along with our Alpha-based server, Apollo were rather stable. A single run on one machined lasted 53 days.

Maple, being interpreted, is estimated to be at least (2) orders of magnitude slower than a compiled language. Thus, our 365+ days of CPU time could conceivably be reduced to < 10. The combination of input parameters led to a theoretical ~21K data points, however, we only obtained ~11K. Analysis structuring limited us to using ~6K data points. The 206 TB indicates the total amount of data processed during these algorithms.

This image is of the UMS MEDICIS 658 cluster located at the CNRS (National Center of Scientific Research) and the École Polytechnique of France (<http://medicis.polytechnique.fr>). These machines were used in the computation of our results, along with UNO's server, Apollo. The cluster consists of a large number of robust machines running various mathematical packages, e.g. Maple, Mathematica, & Macysma.

Total computing capacity is on the order of 15 GHz of CPU cycles w/15 GB of RAM in 29 machines w/45 CPUs, connected over a 100 MBs local network with a virtually unlimited amount of disk storage. Operating systems include LINUX, FreeBSD, OSF, SunOS, & Windows NT. Interaction is controlled via SSH & PKI. MEDICIS' resources are managed by Platform's LSF software (<http://www.platform.com/>).

There are several additional functions, some mentioned here: the transitive edge reducer limits computation & ensures we have DAGs. The vertex permuter eliminates any named ordering bias by the partitioners. The topological sort was necessary and sorts based on delay and pure ordering. The hasse diagram determiner will display the hasse diagram for a graph. The one for C17 is shown, as drawn by Maple V.

Results Analysis

- Graphs: V, E, & E/V
- Big-O: $\max(O_s(|V|))$ & $\max(O_t(|V|))$
- Constraint observation
 - |V|: Vertex replication
 - K: Components
 - k: Partitions
- Partition Examples: All families - $|V|=81$, V-bi
- Analysis Examples
 - Graph Family (Cat, Geo: Grid, Cross, & T_1 - T_4)
 - Domain Category (Circuit & Schedule)
 - Standard Deviation (D=10 & D=1000)
 - Partitionings: V-bi, E-bi, & VE-bi, FPGAs

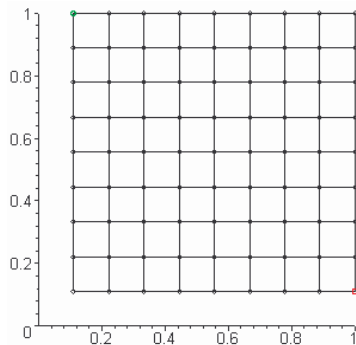
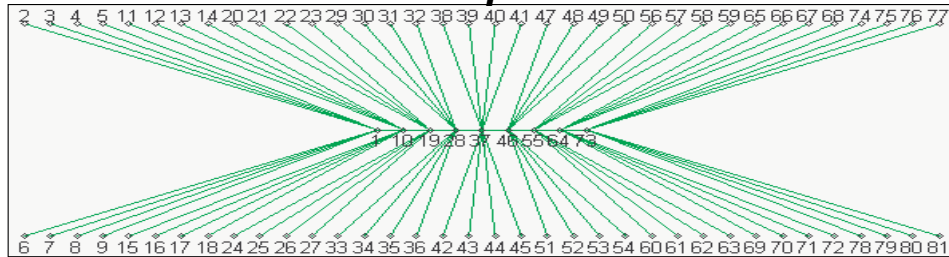
Results presented here include: visual, summary and Algorithm Family, GEA & CTR comparisons. The visual slides show the original graph and representative partitioning from each algorithm family (BPD, CTR, DSC, GEA, & RAN). RAN is a control algorithm using the GEA encoding scheme. It continually generates a random solution and compares it against the best it has generated. It is given the $RMS(G_f(O_t))$ of the other algorithms as it's time limit to find a solution.

The summary slides compares the graphs used in experimentation along w/overall results for O_s , O_t , replication, & multi-component/partition results. Structural metrics, such as graph family and $|E|/|V|$ best serve to distinguish algorithm performance. However, the control metrics become useful when comparing variations within an algorithm family.

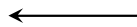
These same comparisons are used to compare variations of both the GEA & CTR algorithms. BPD was only used in its modified form for two constraints (although V-bi is BPD in original form, as the E restriction becomes irrelevant).

Partitions: Graph Families

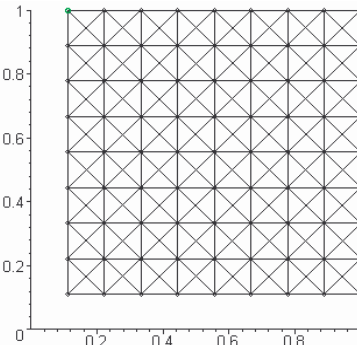
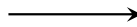
Caterpillar



GeoGrid-S



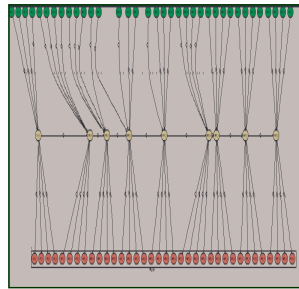
GeoGrid-X



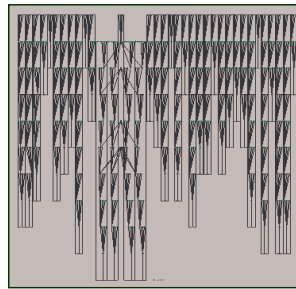
The following slides show a sample graph of 81 vertices for each graph family, along with a partition example for each algorithm in the V-bi & E-bi categories. This slide depicts the Caterpillar graph C9_X_4 (9 central vertices w/4 inputs & outputs each). All images were generated by Maple V.

The graph on the bottom left is from the Geometric Cross family where vertices are connected if directly in line with each other horizontally or vertically. The Geometric Grid-X, or Cross family, also adds directly diagonal vertices to the graph. Both of these contain 81 vertices.

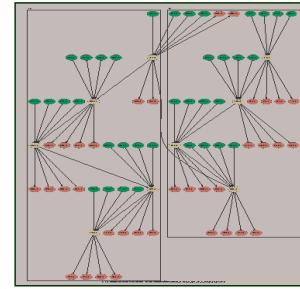
Partitions: Circuit, C9 x 4, V/2



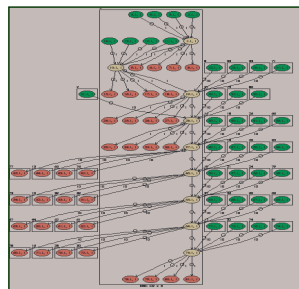
Input, $D_t=11$



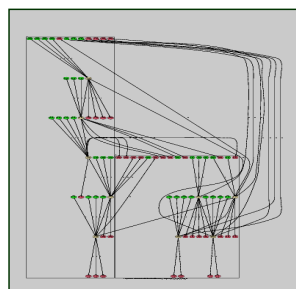
***BPD, $D_t=21$**



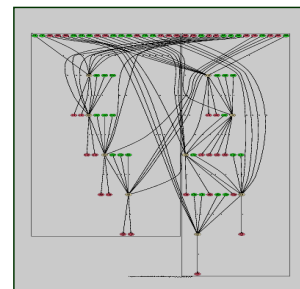
*****CTR, $D_t=27$**



DSC, $D_t=28$



GEA, $D_t=41$



****RAN, $D_t=51$**

This slide shows the partitioning results for the earlier caterpillar example. Note, a new control algorithm, RAN, is shown here also. These images were drawn by GraphViz.

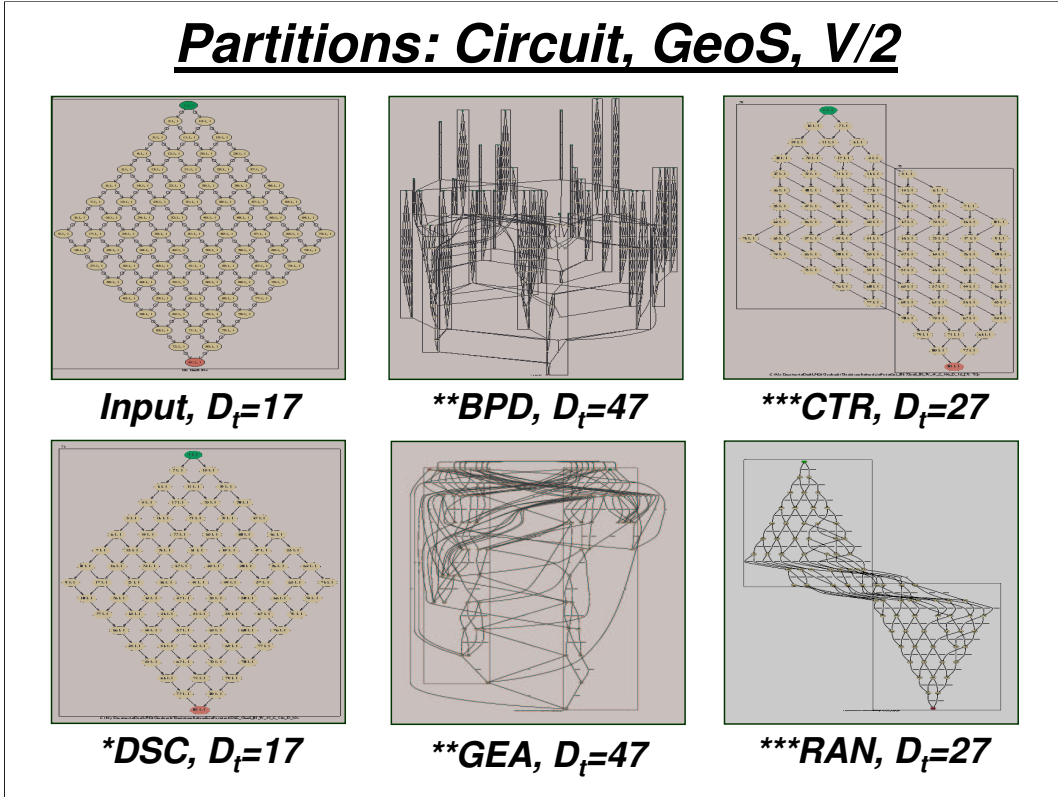
One can see how BPD generates its own components along w/severe replication. DSC generates an excessive amount of partitions, due to its unlimited processing device availability assumption. The randomness behind both GEA & RAN lead to somewhat unpleasing aesthetic results, while CTR finds the natural “center” of the graph., along with the best result.

*: Minimum D_t

** : Maximum D_t

***: Minimum D_t meeting constraints

Partitions: Circuit, GeoS, V/2

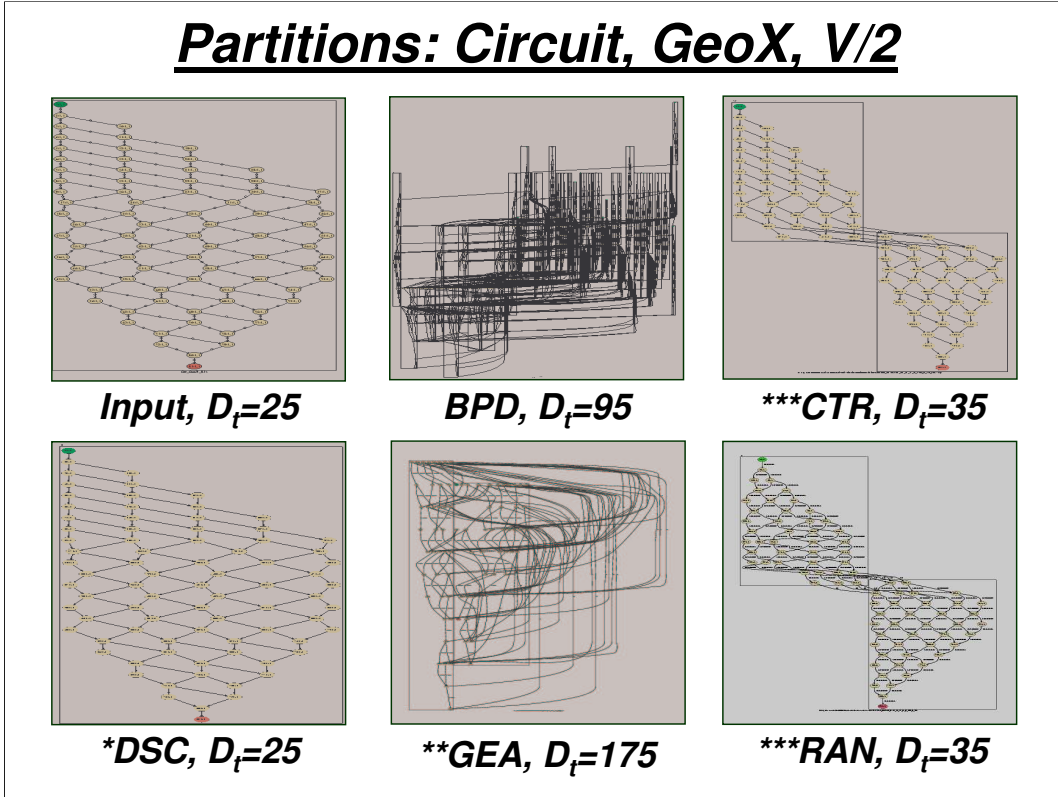


This slide depicts Geometric Grid. Note the single primary input & output vertex. Vertex #s are assigned L R & T B. A lower # vertex will always be the parent of any neighboring higher # vertices. The same geometric algorithm is used to form this graph as in subsequent graphs, with the vertices pre-determined. A complete graph may be generated by saying the radius is the square root of 2.

Note that DSC determines this graph is not worth partitioning. In other words, it determines there are so many dependencies, it isn't worth the extra cost to partition.

- *: Minimum D_t
- ** : Maximum D_t
- ***: Minimum D_t meeting constraints

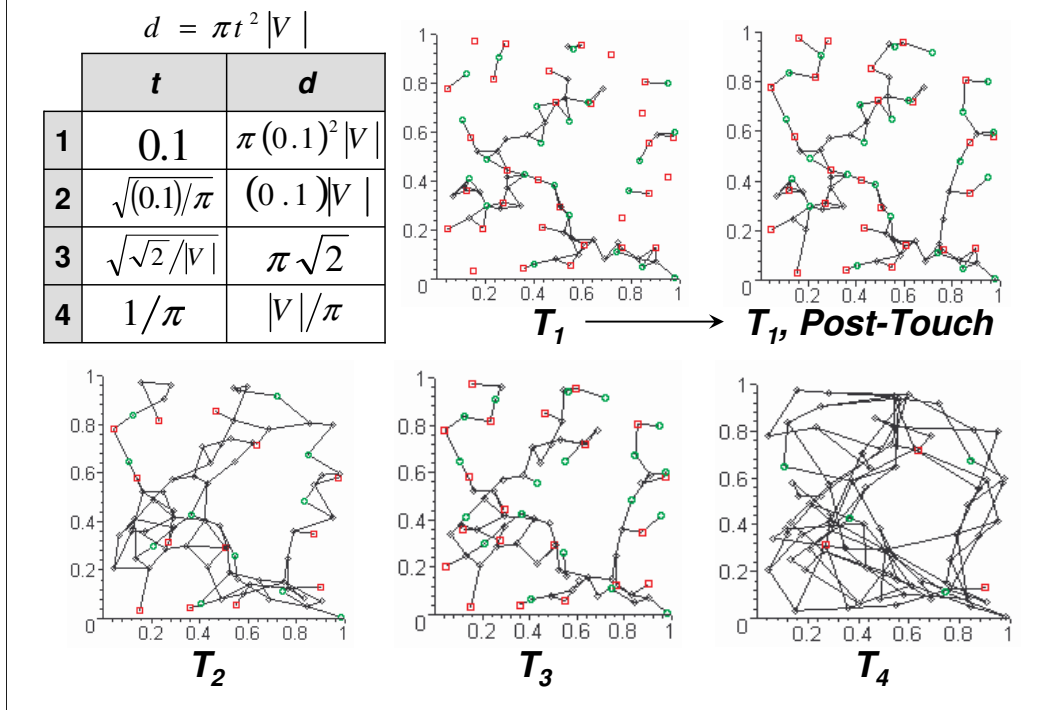
Partitions: Circuit, GeoX, V/2



This slide depicts Geometric Grid-X. Note the single primary input & output vertex. Vertex #s are assigned L R & T B. A lower # vertex will always be the parent of any neighboring higher # vertices. The same geometric algorithm is used to form this graph as in subsequent graphs, with the vertices pre-determined. A complete graph may be generated by saying the radius is the square root of 2. Note that DSC determines this graph also is not worth partitioning.

- *: Minimum D_t
- ** : Maximum D_t
- ***: Minimum D_t meeting constraints

GeoTouch Graphs



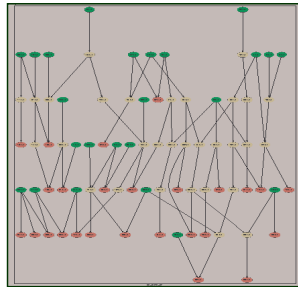
Geometric Touch graphs are populated by randomly placing $|V|$ vertices on a 1×1 grid. They are an extension of geometric graphs we created. The geometric grid and cross families can be expressed via this method as well. Once the vertices are placed, any vertices within the distance t of each other are then connected. The geometric equation is $d = \pi t^2 * |V|$.

For neighbors within the distance t there exists an edge from the lower # vertex to the higher # vertex (prevents cycles). The value of d , density, indicates how many neighbors each vertex may be expected to have. The touch process joins vertices of separate components to its nearest, largest component until a single component is obtained.

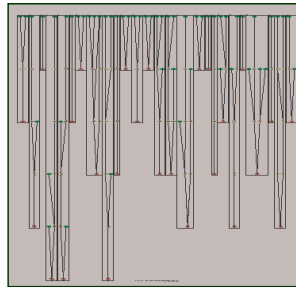
Note, by specifying $t \geq \sqrt{2}$, one can create a complete graph. We used four variants of this graph family. For each variant, we established a t or d value for each of the following scenarios:

- T_1 : constant distance & constant density percentage
- T_2 : variable distance & constant density percentage
- T_3 : variable distance & constant density value
- T_4 : constant distance & variable density value (solving for $d = t, |V| = 1$)

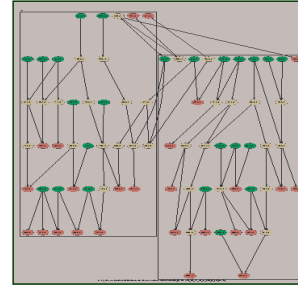
Partitions: Circuit, GeoT-1, V/2



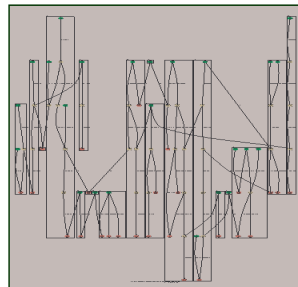
Input, $D_t=6$



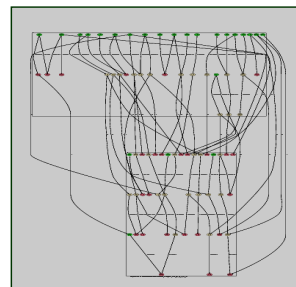
BPD, $D_t=6$



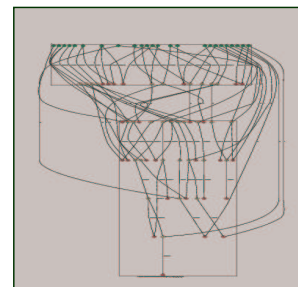
CTR, $D_t=25$



DSC, $D_t=23$



GEA, $D_t=25$



RAN, $D_t=36$

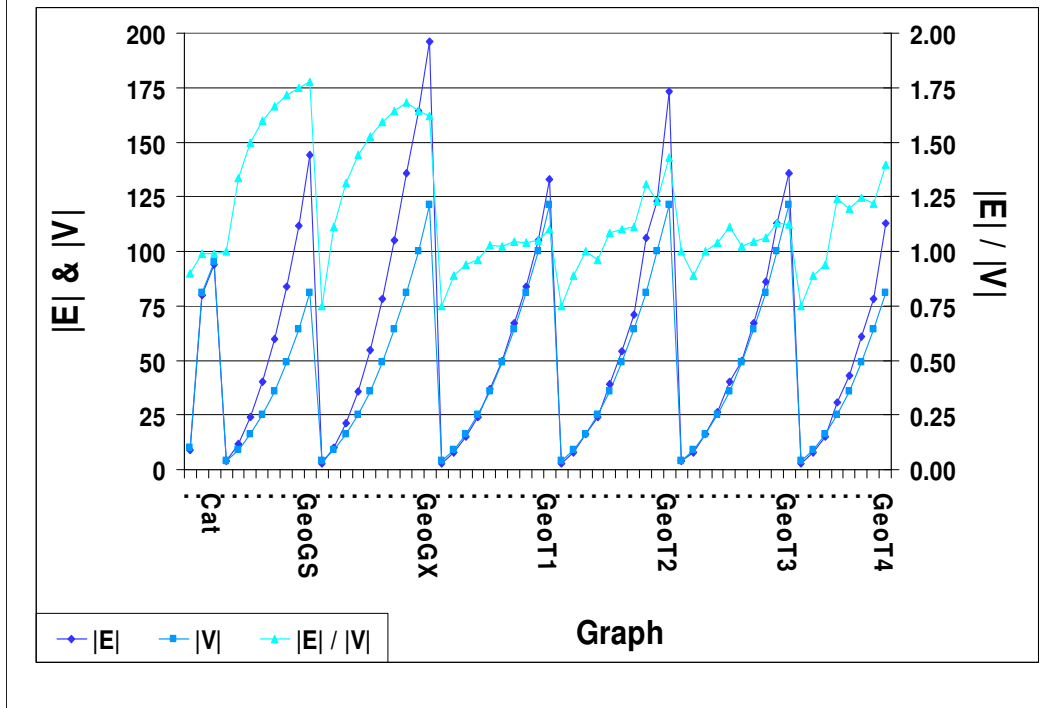
BPD generates less replication as there is a higher # of |roots| as compared to GeoS & GeoX. BPD & DSC also generate a a more reasonable number of partitions for the same reason. Notice the equivalent partitions between CTR & GEA, while CTR tends to have a lower cut.

*: Minimum D_t

** : Maximum D_t

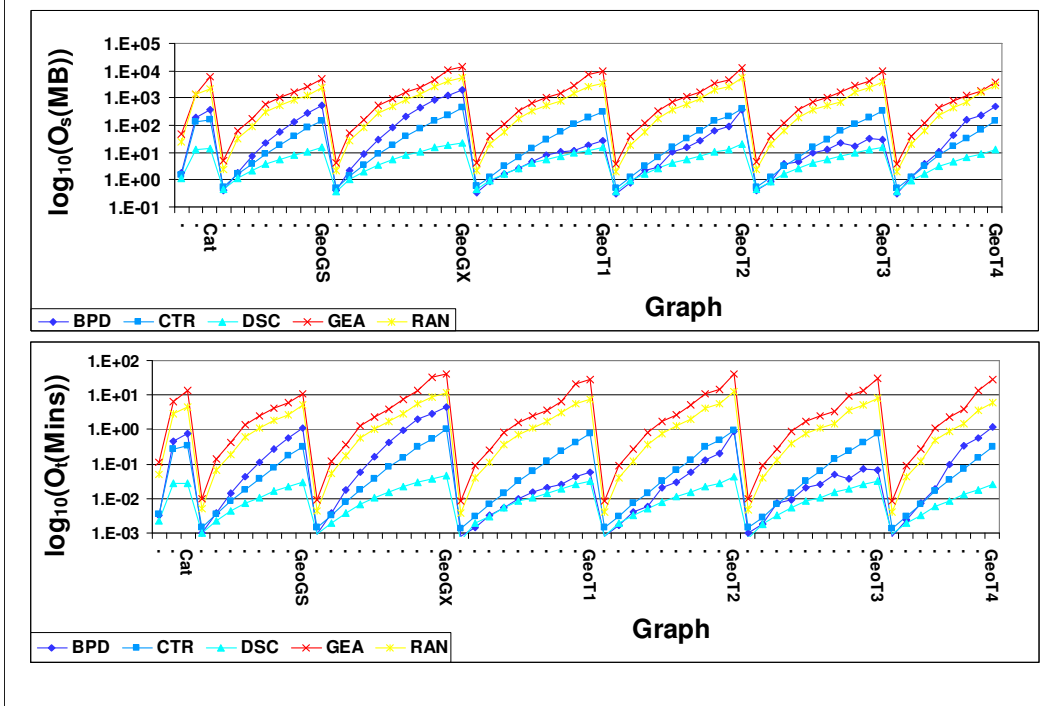
***: Minimum D_t meeting constraints

Graph vs. E, V, & E/V



This graph depicts the $|E|$, $|V|$ and $|E|/|V|$ values for the graphs used in the partitioning experiments. The graphs to the left of a category name are in the same category, e.g. the first graph is a caterpillar graph. Caterpillar and GeoTouch-3 maintain a relatively constant ratio, while the others all increase as the graphs add vertices. Note the ratio never falls below 0.75 as the smallest graph has 4 vertices and therefore must have 3 edges to be connected. $|E| \geq |V| - 1$ for all graphs and thus $|E|/|V| \geq 0.75$.

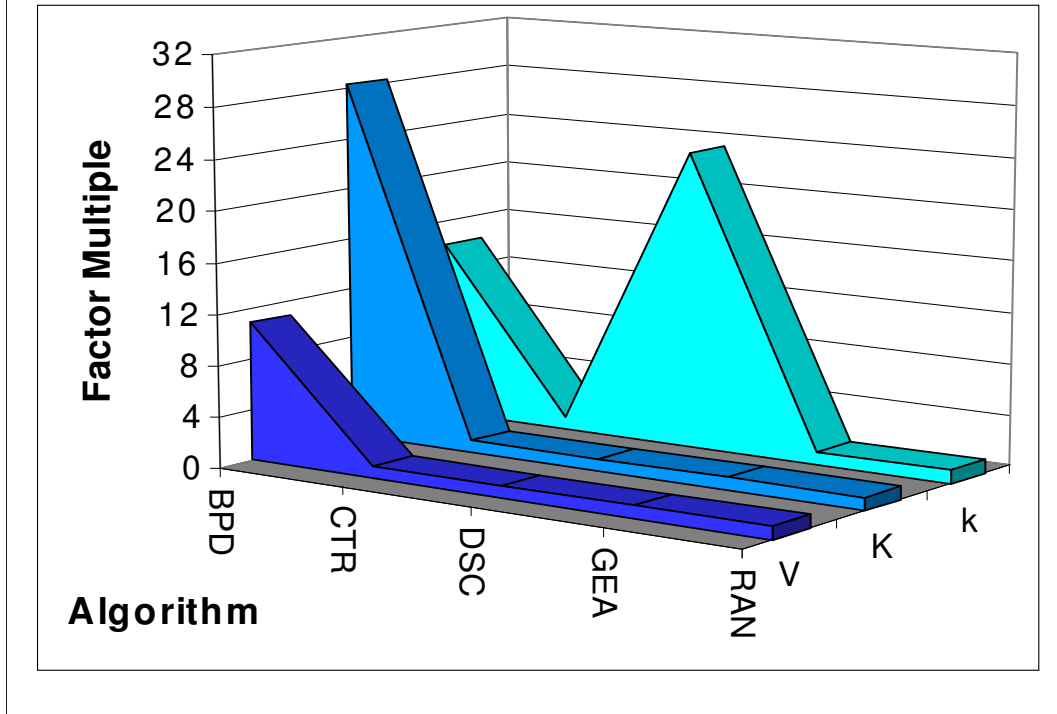
Max O_s & O_t



Although unobserved prior to experimentation, CTR, GEA & RAN use about the same level of space on the same graphs. BPD is highly variable, but the replication factor, especially on denser graphs increases it to roughly the same level as the other. DSC approaches the 1 GB barrier while the others approach the 1 TB barrier. These values seem largely sensitive to $|V|$, excluding DSC.

Similar analysis to O_s , although the scale is in minutes, not MB. We again clearly observed the 2-3 order of magnitude difference between DSC and the other algorithms.

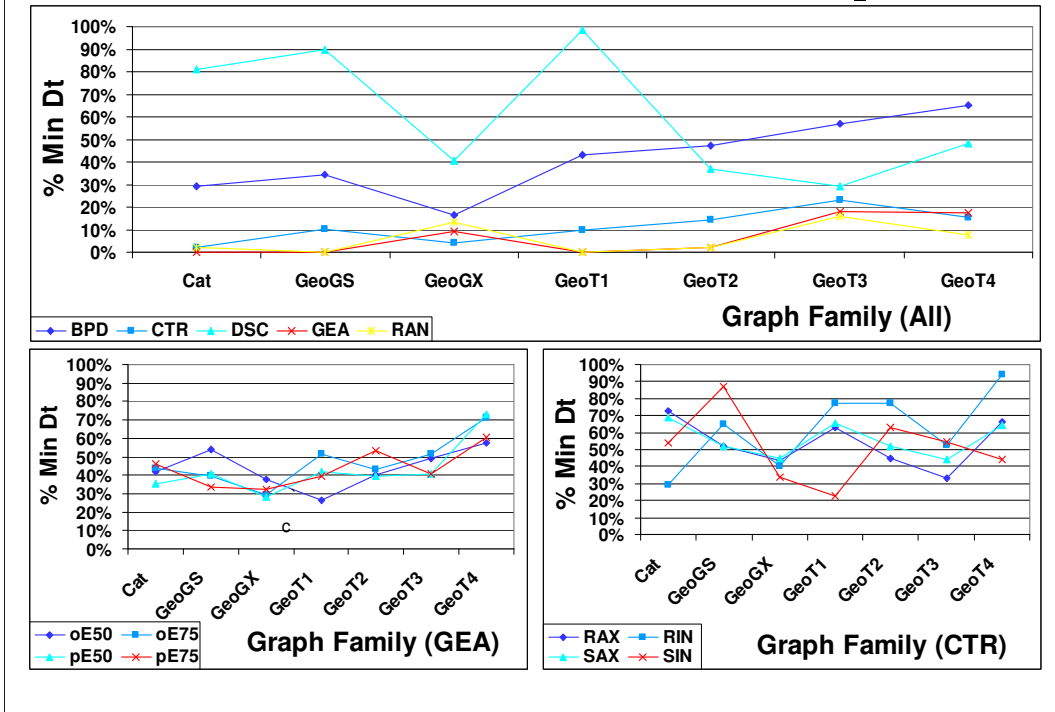
Algorithms vs. V, K, & k



This slide shows the dramatic differences among replication, partitions and components. As seen in the sample partitions of earlier slides, BPD produces a factor of 9 for vertex replication (V). These subgraphs may go all the way to the primary input(s), and this creates a significant number of individual components (K, these components may be spread across multiple subgraphs).

This also directly leads to increase the number of subgraphs (k) produced in a partitioning. DSC, which assumes an unlimited number of processing devices, also sees a dramatic number relative to the number of desired subgraphs. CTR, GEA & RAN are unaffected by any of these factors and see no increase in any of these areas.

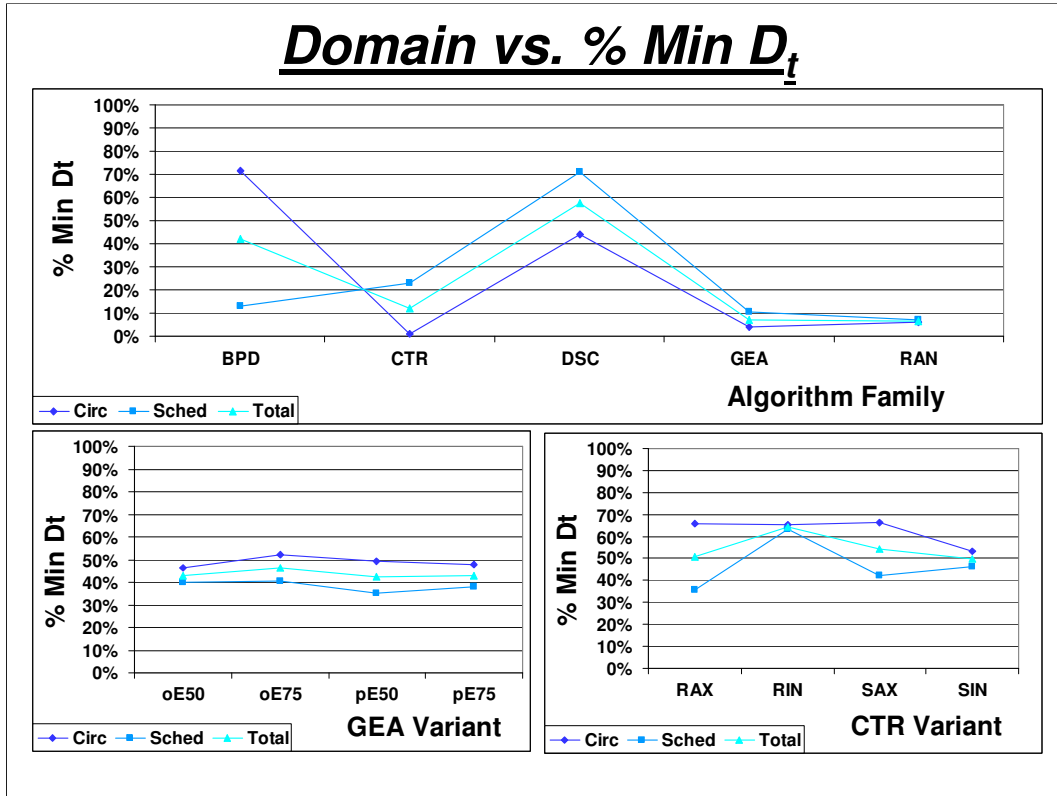
Graph Family vs. % Min D_t



Graph family proves to be one of the most distinguishing metrics when identifying partitioning algorithms. Notably, DSC falls dramatically on circuit type graphs. RAN tends to outperform GEA, although this could be an implementation anomaly. CTR averages better than RAN over the same allotted time. The key item to recall is neither DSC or BPD adhere to k-constraints.

Same as previous, except it is the GEA internal comparison. Note relative dominance of the OX operator, esp. the E75 variant. The sweet spot for elitism could be higher or lower, but is somewhere above E50.

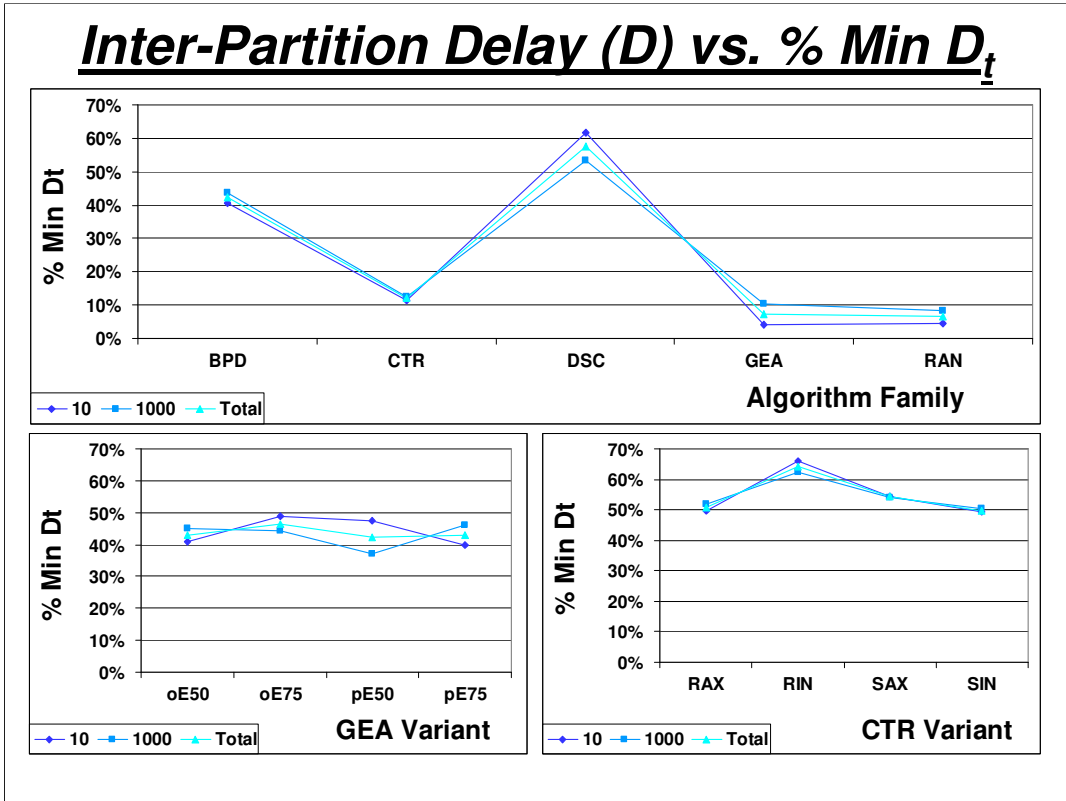
Barring the caterpillar and the geometric cross families, the RIN variant performs best for the CTR family.



In order of performance: DSC, BPD, CTR, RAN, GEA. Essentially, domain evaluation appears insignificant. In other words, a good schedule makes a good circuit & vice versa. It would be worthwhile to expand this to use a granularity or linearity metric instead.

Again, OX is better than PMX and E75 is better than E50. Differences are by roughly the same percentage levels. PMX's better performance on circuit vs. schedule is largely due to the fact it retains the position of the members. This leads to less change being introduced to the population and a slower improvement rate, as scheduling is highly dependent on position.

For CTR, again RIN dominates the results

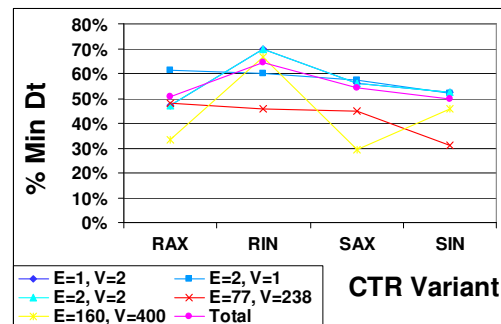
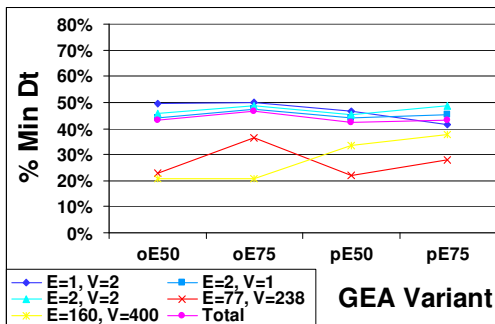
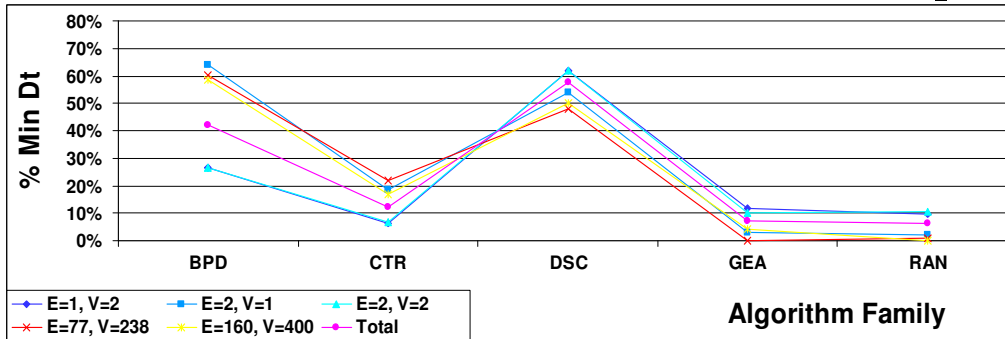


This graph is roughly similar to circuit & schedule across algorithm families. Essentially, the delay between subgraphs is not a significant factor affecting algorithm performance.

Similar to GEA performance for circuit & schedule. Again, note OX & E75's better performance. However, this a difference between the 10 & 1,000 delays for all the algorithm, but we have no explanation at this time.

Yet one more time, RIN is the winner for CTR.

Partition Limits (C, M) vs. % Min D_t

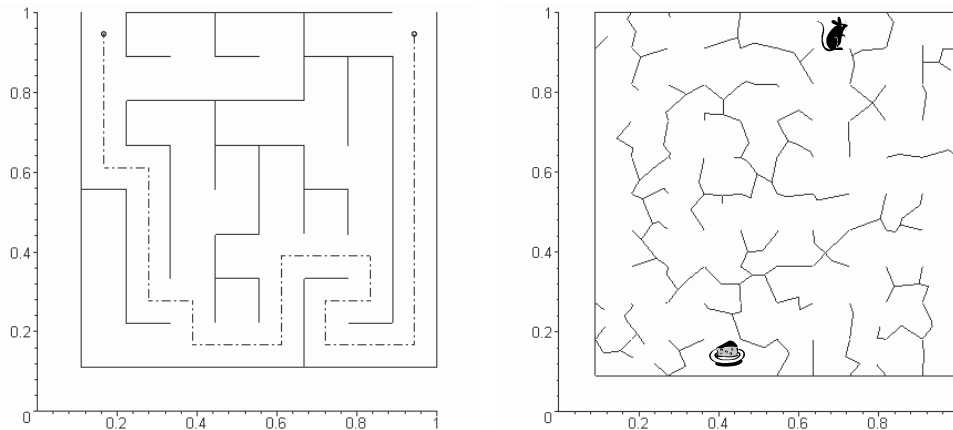


BPD is the only algorithm to have a significant performance difference between the partitioning types. Essentially, splitting of the vertices on our BPD variant causes significant problems, while splitting the edges does not. Although we did not include the original BPD algorithm, it would be worthwhile to compare the results of the original algorithm.

Again, OX & E75 outperform the rest. However, OX handles V-bi partitions, while PMX performs better on E-bi partitions.

As with all other scenarios, RIN appears to be the best variant for CTR.

Maze Generation



- Side effect of geometric touch graphs for empty graph
- In essence, generates a spanning tree of all vertices
- Any point can be reached from any other point
- Can be used for any set of points (non)-orthogonal

The maze generator is a side effect of our geometric touch graph generator. In essence, when passed a fully disconnected (empty) graph, a spanning tree is generated, and once enclosed in a box, serves as a maze. We experimented all the way up to a 2500 vertex graph, thus having 2499 edges. The examples shown here have 81 vertices and 80 edges.

Summary

- **Conclusions**
 - Structural metrics best perf. indicator (family, density)
 - Maple V (*GraphPar*): good for proof-of-concept
 - *GraphViz* provides best partitioning support
- **Contributions**
 - *AGPM*: Abstract Graph Partitioning Model concept
 - *BPD*: post-processing replication reduction
 - *DSC*: incorporation of k-constraints
 - *GEA*: comparison w/other encodings
 - *CTR*: V samples for large k-values
 - *RAN*: V samples for large k-values
- **Future Research**
 - Existing algorithms: Analyze for AGPM/variant analysis
 - Convert to a compiled (faster) development language
 - Better partitioning support in visualization tools

Structural metrics best indicate an algorithmic family's performance, allowing for its' assumptions. However, control metrics (delay, domain, partition type) are useful when comparing variants of the same family.

Maple V: although execution-time problems were experienced on Windows OS family platforms, it facilitated rapid application development & a mathematical mindset.

GVTs: GraphViz had best partitioning support; better GUI would make good tool better.

AGPM serves as a solid taxonomic model to expand upon. It would be best served by adding algorithms & application domains to the taxonomy and determining additional & significant cross-domain metrics. Ideally, one should be able to specify the input graph metrics, partitioning control metrics, and the output metrics desired, along with the desired values. By matching these against the AGPM, the necessary algorithm(s) can then be selected.

BPD: solid, stable performance; creates excessive replication, components and partitions.

DSC: clearly has best results; primarily useful when permitted to have the unlimited # processors. One could try a post-processing step to reduce the # partitions, but would then have to compare if results still good against other algorithms obeying k-constraints

GEA: the OXE75 version deserves further exploration, especially as not all runs for this algorithm were generated in our experiments. Additionally, the encoding scheme provides flexibility, e.g. our usage of it for load balancing & map coloring.

CTR: control parameters should be used when determining which variant to use. Tends to have aesthetic results, useful for comparing w/original graph. Subgraph sampling?

RAN: served as a good control algorithm, as it will generate a reasonable answer.