UNIVERSITY OF CRETE
SCHOOL OF SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE

DUAL GRAPH PARTITIONING OF CELLULAR NETWORKS: INFERRING AND QUANTIFYING THE GRAPH PROPERTIES THAT GUARANTEE A PARTITION RESULT

by

Michael Moudatsos

A thesis submitted in partial fulfillment of the
requirements for the degree of Masters in
Computer Science

2008

Author _____

Supervisor: _____
Ioannis G. Tollis, Professor

Readers: _____
Apostolos Traganitis, Professor

_____
Ioannis Tsamardinos, Assistant Professor

Approved by Chairperson of Supervisory Committee:

_____
Panos Trahanias, Professor

Date _____

HERAKLION, CRETE

ABSTRACT

Graphs are frequently used to model networks of all sorts. Objects of interest are modeled with graph nodes and communication between them is modeled with edges between nodes. The communication cost, whose semantics vary on the application, is modeled with edge weights.

Graph partitioning is currently one of the most interesting NP-Hard problems, due to its applications in networks. Its aim is to create a specified number of groups of objects while minimizing the communication cost between these groups, which equals the sum of all edge weights whose nodes belong to different groups.

Specifically, in cellular networks, the number of communication operations between cells defines the communication cost. Cells are assigned to switches, which in turn are assigned to routers. Communication between cells belonging to different switches or routers demands extensive signaling, which slows down the process. Graph partitioning aids in the assignment of cells to switch and router groups, minimizing the inter-group communications. Thus, less frequent communications will have to bear the signaling cost.

There is an extensive work in the literature, proposing heuristics and comparison results on graph partitioning. Most partitioning solutions concern iterative algorithms whose performance is determined by the partition quality.

In this work, an existing planar graph partitioning algorithm is analyzed. The proposed algorithm has a polynomial run time, constituting a faster partitioning solution compared to iterative algorithms, with the trade-off of lower quality. The Dual Graph Partitioning algorithm (DGP) exploits certain properties of a planar graph's dual. Each path between two nodes at the external border of the dual graph, corresponds to a set of edges in the original graph. The removal of these edges results in two disconnected subgraphs. This kind of partitioning is called bisection. DGP searches shortest paths between all border nodes and chooses the shortest that separates the graph in two subgraphs of almost equal size, depending on a maximum allowed imbalance.

However, it is not guaranteed that there will always be a shortest path between any pair of border nodes that would bisect the graph. This work aims to investigate the graph characteristics that affect the successful completion of the algorithm, through partitioning runs on generated hexagonal graphs. The PRAGMA mobility model was used to infer a sample distribution of edge weights. Graphs were generated using similar weight distributions.

Results showed that the spatial distribution of high and low weights in a graph, affects the route of border-to-border shortest paths, and therefore, whether they bisect the graph with a desired section size ratio. Weight spatial and value distribution was modeled with the notion of population centers, nodes from which edge weights decrease with distance. The distribution of distances between population center to graph border nodes was used to identify whether a graph can be bisected using DGP.

A variation of the algorithm, which guarantees a partition result with the trade-off of higher complexity, is also presented. The partition quality is compared to that of combinatorial optimization techniques for a number of graphs used in the literature.

Περίληψη

Οι γράφοι χρησιμοποιούνται πολύ συχνά για την μοντελοποίηση παντός τύπου δικτύων. Τα αντικείμενα ή έννοιες του υπό μοντελοποίηση θέματος αναπαριστώνται με κόμβους και η επικοινωνία ή σχέση μεταξύ αυτών αναπαριστώνται με ακμές μεταξύ κόμβων. Το κόστος αυτής της επικοινωνίας, η σημασιολογία του οποίου ποικίλει ανάλογα με την εφαρμογή, μοντελοποιείται με βάρη ακμών.

Η διαμέριση ή αλλιώς, τμηματοποίηση γράφων είναι ένα απο τα πιο ενδιαφέροντα τρέχοντα προβλήματα της κλάσεως NP-Hard, λόγω της εφαρμογής του στα δίκτυα. Ο στόχος του προβλήματος είναι να δημιουργηθεί ένας καθορισμένος αριθμός συνόλων από αντικείμενα του εκάστοτε προβλήματος, ελαχιστοποιώντας το κόστος επικοινωνίας μεταξύ των συνόλων αυτων, το οποίο ισούται με το άθροισμα των βαρών των ακμών των οποίων οι κόμβοι ανήκουν σε διαφορετικά σύνολα.

Ειδικότερα στα ασύρματα δίκτυα κυψελών, το κόστος επικοινωνίας ορίζεται ως τον αριθμό λειτουργιών επικοινωνίας μεταξύ κυψελών. Οι διεπαφές επικοινωνίας των κυψελών συνδέονται με μεταγωγείς, οι οποίοι με τη σειρά τους συνδέονται με δρομολογητές. Η επικοινωνία μετξύ κυψελών που ανήκουν σε διαφορετικούς μεταγωγείς ή δρομολογητές απαιτεί την εκτεταμένη χρήση λειτουργικών σημάτων, καθυστερώντας την όλη διαδικασία. Η τμηματοποίηση γράφων βοηθάει στην ανάθεση κυψελών σε σύνολα μεταγωγέων και δρομολογητών, ελαχιστοποιώντας το πλήθος των επικοινωνιών μεταξύ συνόλων. Συνεπώς, τα ζεύγη κυψελών που είναι λιγότερο πιθανά να επικοινωνήσουν υπόκεινται του επιπλέον κόστους σήμανσης.

Υπάρχουν εκτεταμένες μελέτες στην βιβλιογραφία, που προτείνουν ευρεστικές λύσεις και συγκρίνουν τα αποτελέσματά τους, σχετικά με την τμηματοποίηση γράφων. Οι περισσότερες λύσεις αφορούν επαναληπτικούς αλγόριθμους, των οποίων η επίδοση καθορίζεται απο την ποιότητα της διαμέρισης.

Στην παρούσα εργασία αναλύεται ένας υπάρχων αλγόριθμος τμηματοποίησης. Ο προτεινόμενος αλγόριθμος τερματίζει σε πολυωνυμικό χρόνο, αποτελώντας μια γρηγορότερη λύση τμηματοποίησης σε σχέση με τους επαναληπτικούς αλγόριθμους, με το κόστος της χαμηλότερης ποιότητας του αποτελέσματος. Ο αλγόριθμος Τμηματοποίησης μέσω Δυικού Γράφου (DGP) εκμεταλλεύεται συγκεκριμένες ιδιότητες του δυικού, ενός επίπεδου γράφου. Οποιοδήποτε μονοπάτι μεταξύ δύο κόμβων που βρίσκονται στο εξωτερικό σύνορο του δυικού γράφου (συνοριακοί κόμβοι), αντιστοιχεί σε ένα σύνολο από ακμές στον αρχικό γράφο. Η αφαίρεση αυτών των ακμών οδηγεί στην δημιουργία δυο αποσυνδεμένων υπογράφων. Αυτή η διαμέριση ονομάζεται διχοτόμιση. Ο αλγόριθμος DGP ψάχνει όλα τα συντομότερα μονοπάτια μεταξύ συνοριακών κόμβων και επιλέγει το ελάχιστο από αυτά που χωρίζουν τον γράφο σε δύο περίπου ίσα μέρη, σύμφωνα με μια δεδομένη μέγιστη επιτρεπτή απόκλιση στο μέγεθος.

Εν τούτοις, δεν υπάρχει εγγύηση ότι θα υπάρχει πάντα ένα συντομότερο μονοπάτι μεταξύ συνοριακών κόμβων (διασυνοριακό μονοπάτι) που θα διχοτομεί τον αρχικό γράφο. Στόχος της παρούσης εργασίας είναι να ερευνήσει τα χαρακτηριστικά ενός γράφου που επηρεάζουν την επιτυχή εύρεση διαμέρισης απο τον αλγόριθμο, εφαρμόζοντας μια σειρά απο διαμερίσεις σε τεχνητούς εξαγωνικούς γράφους. Το μοντέλο κίνησης PRAGMA χρησιμοποιήθηκε για την εξαγωγή ενός δείγματος κατανομής των βαρών των ακμών. Οι τεχνητοί γράφοι παράχθηκαν χρησιμοποιώντας παρόμοια κατανομή βαρών.

Τα αποτελέσματα έδειξαν ότι η χωρική κατανομή των υψηλών και χαμηλών βαρών στο γράφο επηρεάζει τη διαδρομή των διασυνοριακών συντομότερων μονοπατιών και συνεπώς, το αν διχοτομούν το γράφο με ένα επιθυμητό λογο των μεγεθών των τμημάτων που προέκυψαν. Η χωρική κατανόμή όπως και η κατανομή των τιμών των

βαρών μοντελοποιήθηκε με την έννοια των «κέντρων πληθυσμού». Τα βάρη των ακμών μειώνονται αντιστρόφως ανάλογα με την απόστασή τους από τον κοντινότερο κόμβο-κέντρο πληθυσμού. Η κατανομή των τιμών των αποστάσεων μεταξύ των κέντρων πληθυσμού και του εξωτερικού συνόρου του γράφου βοηθάει στην αναγνώριση των γράφων που μπορούν να διχοτομηθούν επιτυχώς από τον DGP.

Επίσης, παρουσιάζεται μια παραλλαγή του αλγορίθμου που εγγυάται την εύρεση μιας διαμέρισης, με το κόστος της μεγαλύτερης χρονικής πολυπλοκότητας. Η ποιότητα διαμέρισης του αλγορίθμου συγκρίνεται με αυτή των τεχνικών συνδυαστικής βελτιστοποίησης, εφαρμόζοντας τον αλγόριθμο σε γράφους που χρησιμοποιούνται στη βιβλιογραφία.

Acknowledgements

I would like to thank the University of Crete and the Computer Science Department for the opportunity which gave me to learn and move in the world of science and for the means and facilities that provided to aid in developing my skills and interests. I would like to thank all the people I cooperated with, for the scientific as well as life experiences I acquired. Special thanks to my supervisor, for his guidance towards the completion of my work, and all the people that helped me in this long course. Finally, many thanks to all my friends and intimates for their understanding and support during this period.

TABLE OF CONTENTS

LIST OF FIGURES

Number *Page*

LIST OF TABLES

Number *Page*

Introduction

Graphs have been used to model relations between objects and to solve context-related problems. Objects are represented with a set of vertices V and relations between objects are represented with a set of edges E, which can be weighted. Their most straightforward representation concerns networks of all sorts. The first problem that is considered to have been solved with the aid of graph theory is the problem of the bridges of Königsberg, solved by Euler in 1736 [5]. Since then, graphs have been extensively used to model various types of networks such as hardware networks [19], data networks [100], [62], [73], [86], processing networks [16], [32], [92], and communication networks [27], [35], [93].

One of the most important graph theoretic problems which applies to these networks is graph partitioning. It is classified as an NP-Hard problem due to the computational burden that is required in order to find an optimal solution. The problem consists of dividing a graph into k disjoint sets of vertices, having a maximum allowed size, such that the sum of weights of edges between nodes of different sets is minimized.

In communication networks, graph partitioning can be used to optimize the hierarchical aggregation and distribution of resources. Communication endpoints are connected in groups to switches, which in turn aggregate their traffic in routers. Communication across endpoints which belong to different switches and/or routers requires additional signaling, which slows down this process. Graph partitioning aids in grouping end points in sets, in such a way that communications between endpoints of different sets are minimized.

An algorithm for partitioning planar graphs is presented in [53]. The authors present the Dual Graph Partitioning (DGP) algorithm, which is applicable to large scale wireless communication networks, such as mobile telephony networks, where the underlying graph structure is inherently planar, that is, no edge crosses another edge. DGP is a bisection algorithm, i.e., it divides the graph in two sections. This technique provides a trade-off between speed and partition quality. It computes partition results by searching shortest paths between all outer face-nodes of the Dual graph, which yields an average complexity of $O(V^{3/2} \cdot \log V)$.

The goal of this work derives from the fact that it is not guaranteed that there will be any such shortest path that divides the graph in two sections complying with the maximum allowed size. We investigated the graph properties that affect the succession and performance of DGP by performing a series of bisections on artificially generated graphs. The partitioned graphs had a hexagonal grid structure. We modeled the value and spatial distribution of weights with the notion of population centers: we randomly chose a number of nodes as population centers and edge weights decreased as distance from their nearest population center increased. In order to get a sample distribution of the ratio between the maximum and minimum edge weight per node, we performed a number of simulations using the PRAGMA mobility model [82].

Bisection runs focused on max/min per node edge weight ratio, graph size, global weight range, the number and position of population centers, as well as the tolerance of imbalance between resulting sections, with respect to their

effect on the algorithm succession and performance. Results showed that the primary factor that affected DGP was the spatial distribution of weights. Spatial aggregation of higher and lower weights affected the route of shortest paths in the Dual graph and hence the resulting partition sizes. It is important that these routes are able to pass through the central region of the graph. The concentration of higher weights in an area near the center of the graph, which contains a large fraction of graph nodes, would force all shortest paths to route around this area resulting in partitions violating the maximum allowed size. The length of these paths was also an indication of how "difficult" was to partition a certain graph. Global weight range also affected results since tight ranges reduced the unevenness between higher and lower weights, resulting in higher successful partition ratios.

We constructed a procedure used to determine whether the weight distribution of a graph favors DGP. We devised a heuristic for detecting population centers in a graph. Since the clustering of high weights in the center of the graph affects results, the distribution of distances between population centers and the outer border of the graph was computed, separately for bisected and unbisected graphs. The distribution of the average of these distances of each graph exhibited a Normal-like distribution. Bisected graphs yielded a distribution with a lower mean than the one of unbisected graphs, which was in accordance with previous speculations about the effect of higher weight spatial distribution. We verified this observation statistically using a two sample T-test.

A variation of the algorithm proposed in [53], which guarantees a partition result under certain conditions, with an additional computational cost, was also analyzed as a complement to the original version of DGP. This variation returned a partition result for all unbisected graphs as well as decreased the bisection cost for some of the graphs, with respect to the original version, using the same partition tolerance.

We examined the trade-off between the apparent gain in run-time and the decrease in partition quality, with respect to partitioning based on combinatorial optimization techniques. A number of planar graphs which were used in other research works [79] and are publicly available along with their best partitions found so far [101] were partitioned with DGP and results were compared. Recursive bisection was used in order to perform k-way partitions, for $k > 2$ and the partition cost ratio was computed. Results were almost equal for more regular graphs and quality decreased as k increased. On the other hand, graphs with less regular structure yielded low quality results for low k values, which improved as k increased.

The structure of this thesis is as follows. In Chapter 1, we provide the basic graph theoretical notations and algorithms, which are used in this work. In Chapter 2, we formulate the graph partitioning problem and present the basic techniques which are used in order to find a solution, as well as the related work. Then, we describe the DGP algorithm, we analyze its complexity and discuss its inherent characteristics. In Chapter 3, we perform a series of bisection runs, focusing on different graph characteristics, in order to infer the properties that primarily affect the succession of the algorithm and its partition quality. Initially, we describe how weights are distributed throughout the graph and we perform a number of simulations using the PRAGMA mobility model in order to get a sample weight distribution. Then, bisection results follow. In Chapter 4, we provide the comparison of DGP results with those of other techniques, over

some planar graph instances, which have been used in the literature and are publicly available. Finally, in Chapter 5 we summarize our conclusions and propose some directions of future work as well as DGP applications.

Chapter 1

Background

1.1. Graph Terminology

A graph is a means of modeling physical objects or concepts and relations between them. Graph *nodes* model the objects and *edges* between nodes model an existing relation between the corresponding objects. An undirected graph is represented by $G = (V, E)$ and consists of a finite set of vertices V and a set of edges E upon V, that is, unordered pairs of distinct vertices. Therefore, if $e = \{w, v\} \in E$ and $v, w \in V$ we say that edge e is *attached to* or *has as endpoints* the nodes v, w. The number of attached edges to a vertex v is called its *degree* $deg(v)$.

A graph $G'=(V', E')$ is called *subgraph* of a graph $G = (V, E)$, when $V' \subseteq V$ and $E' \subseteq E$. A partition of the node set V in two disjoint sets V', V'' defines a *cut* (V', V'') of the graph. The edges that connect vertices of V' with vertices of V'' are called *cut edges*.

A sequence $(v_1, v_2, ..., v_k)$ of k vertices where

$$\{v_i, v_{i+1}\} \in E \quad \forall i \in [1, k-1]$$

constitutes a *path* of length k-1. When a path is constituted by distinct vertices it is called a *simple path*. The longest length among the shortest, and thus simple, paths between two graph vertices represents the *diameter* of the graph. In case that $v_1 = v_k$ the vertex sequence is characterized as a *cycle*, either being simple or not.

A graph is defined as *connected*, when for any two vertices, there is a path that connects them. In the opposite case, it is disconnected and it is constituted by a set of *connected components*, that is, by maximal connected subgraphs.

When a number, also called *weight* is corresponded to each edge the graph is called *weighted*. A *path cost* is defined as the sum of the weights of its constituent edges. The *shortest path cost* from u to v is defined as the minimum of the path costs, which equals infinity if there's no path between u and v. Consequently, as *shortest path* between u and v is defined any path whose cost is equal to the shortest path cost.

A graph is called *planar* if it admits a planar drawing, that is, if it can be drawn on a plane in a way that no two distinct edges intersect. A planar drawing of a graph partitions the plane into topologically connected regions called *faces*. The unbounded face is usually called the *external face*. A planar drawing determines a circular ordering on the neighbors of each vertex v according to the clockwise sequence of the incident edges around v.

The *dual graph* G^* of a planar graph G has a vertex for each face of G and an edge $\{f, g\}$ between two faces f and g for each edge that is shared by f and g. The dual graph G^* is also planar and its dual graph corresponds to the original graph G.

1.2. Graph Algorithms

1.2.1. Breadth-First Search

Breadth-First Search (BFS) is one of the simplest algorithms for searching a graph and an archetype for many important graph algorithms. Dikstra's single-source shortest-paths algorithm use ideas similar to those in BFS. Given a graph $G=(V, E)$ and a distinguishing source vertex s, BFS systematically explores the edges of G to "discover" every vertex that is reachable from s.

Breadth-First Search is so named because it expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier. That is, the algorithm discovers all vertices at distance k from s before discovering any vertices at distance $k+1$.

To keep track of progress, BFS colors each vertex white, gray, or black. All vertices start out white and may later become gray and then black. A vertex is *discovered* the first time it is encountered during the search, at which time it becomes nonwhite. Gray and black vertices, therefore, have been discovered, but BFS distinguishes between them to ensure that the search proceeds in a breadth-first manner. If $u, v \in E$ and vertex u is black, then vertex v is either gray or black; that is, all vertices adjacent to black vertices have been discovered. Gray vertices may have some adjacent white vertices; they represent the frontier between discovered and undiscovered vertices.

The algorithm uses a First-In First-Out queue Q where discovered vertices are inserted. BFS works as follows. Every vertex is painted white. The source vertex is painted gray, since it is considered to be discovered when the procedure begins. Q is initialized as to contain just the vertex s; thereafter, Q always contains the set of gray vertices.

The main loop of BFS then begins. It iterates as long as there remain gray vertices, which are discovered vertices that have not yet had all their neighbors examined. The gray vertex u at the head of the queue Q is determined. For each vertex v which is a neighbor of u, if v is white, then it has not yet been discovered. Thus, it is first grayed and then placed at the tail of the queue Q. When all neighbors of u have been examined, u is removed from Q and blackened.

The run time complexity of an algorithm expresses the computational burden that is required to finish. Considering BFS running on a graph $G=(V, E)$, after initialization, no vertex is ever whitened, and thus each vertex is enqueued at most once, and hence dequeued at most once. The operations of enqueuing and dequeuing take $O(1)$ time, so the total time devoted to queue operations is $O(V)$. Because each neighbor of each vertex is scanned only when the vertex is dequeued, each neighbor of each vertex is scanned at most once. Since the sum of neighbors of all vertices is $\Theta(E)$, at most $O(E)$ time is spent in total scanning neighbors. The overhead for initialization is $O(V)$, and thus the total running time of BFS is $O(V + E)$. Thus, Breadth-First Search runs in time linear in the size of nodes and edges of G.

1.2.2. Dijkstra's Single Source Shortest Path

Dijkstra's algorithm solves the single-source shortest-paths (SSSP) problem on a weighted graph $G = (V, E)$, for the case in which all edge weights are nonnegative.

Dijkstra's algorithm maintains a set S of vertices whose final shortest path costs from the source s have already been determined. For all vertices $u \in S$, $d[v]$ denotes this cost. The algorithm repeatedly selects the vertex u $\in V - S$ with the minimum shortest path estimate, inserts u into S, and relaxes all edges leaving attached to u. A priority queue Q that contains all the vertices in $V - S$, keyed by their d values, is maintained. The next hop vertex v from u that belongs in the shortest path between the source s and u is denoted by $\pi[u]$.

The algorithm first initializes the d and p values to infinity and null respectively, and the set S to the empty set. The priority queue Q is initialized to contain all vertices in $V - S = V - \varnothing = V$. Then, the algorithm iterates while Q is not empty. Each time through this loop, a vertex u is extracted from $Q = V - S$ (*extractMin* operation) and inserted into set S. (The first time through this loop $u = s$.) Vertex u therefore, has the smallest shortest path estimate of any vertex in $V - S$. Then, each edge $\{u, v\}$ attached to v is relaxed, thus updating the estimate $d[v]$ and $\pi[v]$, as well as the Q structure (*decreaseKey* operation) if the shortest path to v can be improved by going through u. Once this iteration begins, nodes are never inserted into Q and each vertex is extracted from Q and inserted to S exactly once, so that the algorithm iterates exactly $|V|$ times.

As far as the running time of Dijkstra's algorithm is concerned, consider a binary heap implementation of the priority queue Q. Each *extractMin* operation takes time $O(\log V)$. There are $|V|$ such operations. The time to build the binary heap is $O(V)$. The edge relaxation is accomplished by a *decreaseKey* operation, which takes time $O(\log V)$, and there are still at most $|E|$ such operations. The total running time is therefore $O((V + E)\cdot \log V)$, which is $O(E\cdot \log V)$, if all vertices are reachable from the source.

Chapter 2

Partitioning Problem Definition and Dual Graph Partitioning Algorithm

2.1. The *k*-way Graph Partitioning Problem

The *k*-way Graph partitioning problem consists of creating *k* distinct sets of nodes, such that the connection cost between these sets is minimized and each set has approximately the same cost. In the case of an unweighted graph $G=(V,E)$, the problem is formed as dividing the graph in *k* sets of $|V|/k$ nodes, such that the number of edges, which are attached to nodes which belong to different sets, is minimized. In a weighted graph, the cost of a set of nodes is defined as the sum of its nodes' weights and the connection cost between sets, as the sum of weights of all edges between different sets. The goal is to create sets of approximately equal cost, while minimizing the connection cost between them. The definition of the problem can also allow a certain amount of imbalance *t* (also called tolerance), concerning the maximum allowed cost of a set. It must hold that $\max(|V_i|) \leq (1+e)\cdot|V|/k$, where V_i any set of nodes, let it be called section, $|V_i|$ the cost of all the nodes of a section, let it be called size, and *e* the maximum allowed fraction of the average section size, which represents the imbalance. Therefore, $t = e\cdot|V|/k$. A special case of *k*-way partitioning, when $k=2$, is also called bisection. The graph is divided in two parts with a possible imbalance and the cost of the set of edges between the two sections, also called as edge cut, has the minimum possible value. The two sections of the bisection can either consist of contiguous or non-contiguous nodes. In the former case, a section is a connected subgraph, whereas in the latter case a section can consist of several subgraphs which are not connected.

The problem of partitioning a graph in *k* sets belongs to the family of combinatorial optimization problems. Contrast to the impractical solution of an exhaustive search of all possible partitions, different methodologies have been employed in order to approximate an optimal solution, such as Tabu Search, Simulated Annealing, Evolutionary Search and Spectral Clustering. In the following section we analyze these techniques and present the corresponding related work.

2.2. Related Work

2.2.1. Preface

There has been extensive work on graph partitioning algorithms with applications on various contexts of science, using different approaches and techniques. Due to the magnitude of this information, a brief presentation of the most important techniques accompanied with an enumeration of the related work, will be provided here. In Appendix A, we provide an elaborate presentation for each technique and a descriptive summary of each related work.

2.2.2. Subsequent Maximization Algorithms

This category of algorithms is based on the fact that the problem of minimizing the external cost (that is, the cost between partitions) is equivalent with the problem of maximizing the internal cost. Given that a partition must have at most p nodes (and thus, $\lceil n/p \rceil$ partitions will be created, for a graph with n vertices), a Subsequent Maximization Algorithm (SMA) chooses a starting node and creates the partition by adding each time a node based on a defined metric, whose implementation differs in the literature and depends on the context. This is repeated until all n/p partitions are created. The choice of the starting nodes of each partition can be implemented with various ways: randomly or by using again a metric with respect to already created partitions. In [61], Ferracioli et al. propose some sub-optimal heuristic partitioning algorithms.

2.2.3. Iterative Refinement Algorithms

This family of algorithms takes a non-optimal partition and iteratively tries to decrease its cost by exchanging nodes which belong to different partitions, until no improvement can be achieved. The choice of the sequence of the nodes to be exchanged varies. It can be implemented using the preexisting pre-ordering, a random pre-ordering, and a topological pre-ordering. The Kernighan-Lin algorithm [3] is one of the most characteristic techniques that represent Iterative Refinement Algorithms (IRA). It is often used as a basic means of comparison with new proposed techniques. In [17], Gilbert et al. develop a parallel algorithm for partitioning the vertices of a graph into $k \geq 2$ sets in such a way that few edges connect vertices in different sets. In [18], the maximum concurrent flow problem (MCFP) is readily illustrated by problems such as traffic flow in road networks and message transfer in packet switched networks. In [25], Tragoudas et al. consider the general partitioning problem, namely, how to partition the elements of a circuit into sets of size less than a small constant, so that the number of nets which connect elements in different sets is minimized. In [43], Saab addresses the network partitioning problem with the use of node clustering. In the context of VLSI circuit partitioning, Dutt and Deng [52] propose new iterative-improvement methods that select cells to move with a view to moving clusters that straddle the two subsets of a partition into one of the subsets. Given n heterogeneous traffic sources which generate multiple types of traffic among themselves, Lim and Lee [54] consider the problem of finding a set of disjoint clusters to cover n traffic sources. In [61], the authors focus on the topological pre-ordering concerning the nodes

exchanged in their IRA implementation. In [70], Begerano et al. propose a Location Area Planning technique which guarantees a bounded quality of the solution in the worst case. In [88], Meals proposes a novel fast technique for partitioning a system into hardware and software components and scheduling the resulting components for execution.

2.2.4. Simulated Annealing

As described by S. Kirkpatrick et al. in [8], the analysis of the movement of atoms of matter at given temperatures and the search of this system's ground state can be corresponded to combinatorial optimization problems. The initial energy of the system at a high temperature can be corresponded to the initial state of a combinatorial problem having a certain cost. In the simulation of the displacement of atoms under a certain temperature, at each step, an atom's displacement result in a change of the system's energy *DE*. If the energy is reduced, the new system configuration is accepted. However in the case of $DE > 0$ the new configuration is accepted with probability $P(DE) = \exp(-DE/(k \cdot T))$, where k is the Boltzmann's constant and T the temperature. Savage et al. in [22], show that the Kernighan-Lin and simulated annealing heuristics are logspace complete for the classes P and BPP, respectively, which means that they are hard to parallelize. In [31], the authors introduce a meta-heuristic to combine simulated annealing with local search methods for CO problems. In the same context, Boukerche et al. [36] consider the problem of partitioning a conservative parallel simulation for execution on a multi-computer, assuming that the synchronization protocol makes use of null messages [6]. In [67], Ricci and Alfeld explore the network testbed mapping problem. Yet another work where simulated annealing is used in order to solve the network partitioning problem in the context of Location Area planning (cell-LA assignment) and cell to switch assignment in cellular networks, is analyzed in [71].

2.2.5. Tabu Search

This technique introduced by Glover and Laguna in [33], incorporates various structures and strategies in order to escape from local optima while searching for a solution. The main concepts of the technique are: *a)* the move, which is an operation which takes us from a current solution to a neighboring solution. *b)* The memory structures, which include information across four dimensions, recency, frequency, quality and influence of a move. The relationship between prohibition-based diversification (Tabu Search) and the variable-depth Kernighan-Lin algorithm is discussed in [60]. Saab [75] introduces a new approach to partition graphs and hypergraphs. In [80] Aringhieri et al. consider two problems that arise in the design of optical telecommunication networks when a ring-based topology is adopted, namely the SONET Ring Assignment Problem and the Intraring Synchronous Optical Network Design Problem. In [84], Chamberland and Pierre deal with the problem of how to design cellular networks in a cost-effective way.

2.2.6. Evolutionary Search

This technique belongs to the category of **genetic algorithms**, since as A.J. Soper, C. Walshaw and M. Cross elaborate in [79], it is a stochastic search technique that generates new points in a search space using information from a finite population of already evaluated points. Maini et al. [37] present a genetic

algorithm for the problem of partitioning in parallel computing. In [50], hybrid genetic algorithms (GAs) for the graph partitioning problem are described. In [58], Shazely et al. present new techniques for discovering more than one solution to the partitioning problem using genetic algorithms. Hwoang et al. [91] propose a new gene reordering scheme for the graph bisection problem. James et al. in [97], consider the registration area planning (RAP) problem. In the context of the network partitioning problem, Soper, Walshaw and Cross combine a multilevel graph partitioner with an evolutionary algorithm in [79].

2.2.7. Spectral Clustering

In the spectral method, the vertices in a graph can be mapped into the vectors in *d*-dimensional space, thus the vectors are partitioned instead of vertices to obtain graph partitioning. Incremental graph partitioning is examined by Ou and Ranka in [38]. Hendrickson and Leland in [44] present a multilevel algorithm for graph partitioning in which the graph is approximated by a sequence of increasingly smaller graphs. In [55], the authors show a method to obtain optimal two-way vector partitioning based on an optimal direction vector. In the same context, recursive spectral bisection (RSB) is a heuristic technique for finding a minimum cut graph bisection. In [56], Chan et al. prove that the median cut method is optimal in the sense that the partition vector induced by it is the closest partition vector, in any l_s norm, for $s \geq 1$, to the second eigenvector.

2.2.8. Heuristic Algorithms and Case Studies

Apart from the techniques described above, there's an extensive work in the literature concerning problem-specific techniques, innovative approaches, as well as comparison or combination of known techniques and even theoretical work on computational bounds and approximation ratios of result quality, with respect to an optimal solution. An extensive report on such related work is reported in Appendix A

2.3. The Dual Graph Partitioning Algorithm

2.3.1. Description

DGP is a graph bisection algorithm. As described in [53], it is applicable to planar graphs, taking advantage of the existence of a dual graph. The outer face node v_o with degree $d(v_o)$, is replaced by $d(v_o)$ special outer face nodes, let them be called border nodes. Since each face of the original graph corresponds to a node of the dual graph, and each edge separating two faces corresponds to an edge connecting the relative nodes of the dual graph, it is evident that an acyclic path between two border nodes of the dual graph corresponds to an edge cut of the original graph. The cut consists of a sequence of edges between neighboring faces, starting from an edge between the outer face and an inner face (a border edge), and all subsequent edges which separate neighboring inner faces, until the last edge, which is between an inner face and once again the outer face. Let us call this a border to border cut. Finding the minimum of these border to border paths in the dual graph leads to a minimum border to border cut in the original graph. The algorithm searches the shortest path between border nodes, which divides the original graph in two sections, whose size accords to a given tolerance.

In more detail, the algorithm is implemented as follows. The dual graph is created and the special border nodes are detected. The Dijkstra algorithm is run for each border node of the dual graph as a source. From a source, paths to all other border nodes are checked in terms of cost. If the cost of a certain path is less than the current minimum cost, the corresponding edges of the original graph are marked as candidate bisection edges, and the size of each resulting section is computed. From each node, if it hasn't been assigned a section, a BFS starts assigning all neighboring nodes to the current section. Neighbors across bisection edges are not added to the BFS FIFO. Thus, at the end of this procedure all sections have been defined and all nodes are assigned a section. If the sizes of the resulting sections comply with the given imbalance, the minimum path and cost are updated.

The SSSP algorithm of Dijkstra in its original form, returns the first minimum path that it discovers, in the case of multiple equal such paths. However, in this application of the algorithm, the choice among different routes of a path is important since it corresponds to different edge cuts in the original graph, which in turn divide the graph in different sections, whose size may or may not comply with the given imbalance. Thus, we modified the Dijkstra algorithm as to return all possible next hops from a node. These hops represent different sub routes of same cost, which result in different partition results.

When a path cost is less than the current minimum, a stack is populated with the path nodes, from the destination to the source, along with the number of possible next hops from each node. The sections are defined and the partition is updated if the imbalance criterion is met. Then, the nodes are popped out of the stack until the first node from which there are more than one next hops to the source. The next alternative hop is pushed in the stack and the corresponding number is reduced by one. The stack is then populated, as it was described before, up to the source. Following this procedure until the stack is empty

guarantees that all possible paths from destination to the source are tested, in terms of the partition they yield.

Here follows the pseudo code of DGP

DGP Pseudo Code

```
DGA(G, tolerance)
»  Create dual graph Gd, from G
»  currentMin = INFINITY
»
»  for each outer face node 'from' of Gd
»  »  compute 'd' and 'nextHop' matrix using dijkstraSSSP(Gd, from)
»  »
»  »  for each outer face node 'to' of Gd
»  »  »  if(d[to] < currentMin)
»  »  »  »  push 'to' in 'pathStack'
»  »  »  »  push nextHop[to].size in 'nextHopCountStack'
»  »  »  »
»  »  »  »  while pathStack is not empty
»  »  »  »  »  push node = nextHop[pathStack.top][0] in 'pathStack', while(node != from)
»  »  »  »  »  push nextHop[node].size in 'nextHopCountStack', while(node != from)
»  »  »  »  »
»  »  »  »  »  compute size ratios 't' from nodes in 'pathStack'
»  »  »  »  »
»  »  »  »  »  if(t <= (1+tolerance))
»  »  »  »  »  »  currentMin = d[to]
»  »  »  »  »  »  update partition cut in G
»  »  »  »  »  »  break
»  »  »  »  »
»  »  »  »  »  while (nextHopCountStack is not empty) and (nextHopCountStack.top == 1)
»  »  »  »  »  »  pop pathStack.top
»  »  »  »  »  »  pop nextHopCountStack.top
»  »  »  »  »
»  »  »  »  »  if(nextHopCountStack is not empty)
»  »  »  »  »  »  nextHopCountStack.top -= 1
»  »  »  »  »  »  push in 'pathStack', next = nextHop[pathStack.top][nextHopCountStack.top]
»  »  »  »  »  »  push in 'nextHopCountStack', nextHop[next].size
»  »  »  »  »  »
»  »  »  »  »
»  »  »  »
»  »  »
»  »
»  return currentMin
```

In Figure 1, an example of Dual Graph Partitioning is presented, over a randomly weighted graph. The original graph is composed of grey nodes, which form a hexagonal grid. Its dual is represented by yellow nodes. The outer face node is replaced by special outer face nodes (let them be called border nodes), one for each outer face edge. The algorithm searched the border to border node shortest paths that separated the graph in two sections with a maximum allowed tolerance of zero nodes. It returned the shortest of these paths, which is depicted with red dual nodes. This path separates the original graph in two sections with the minimum edge cut cost, since each dual edge along this shortest path corresponds to an edge of the original graph, forming an edge-cut.

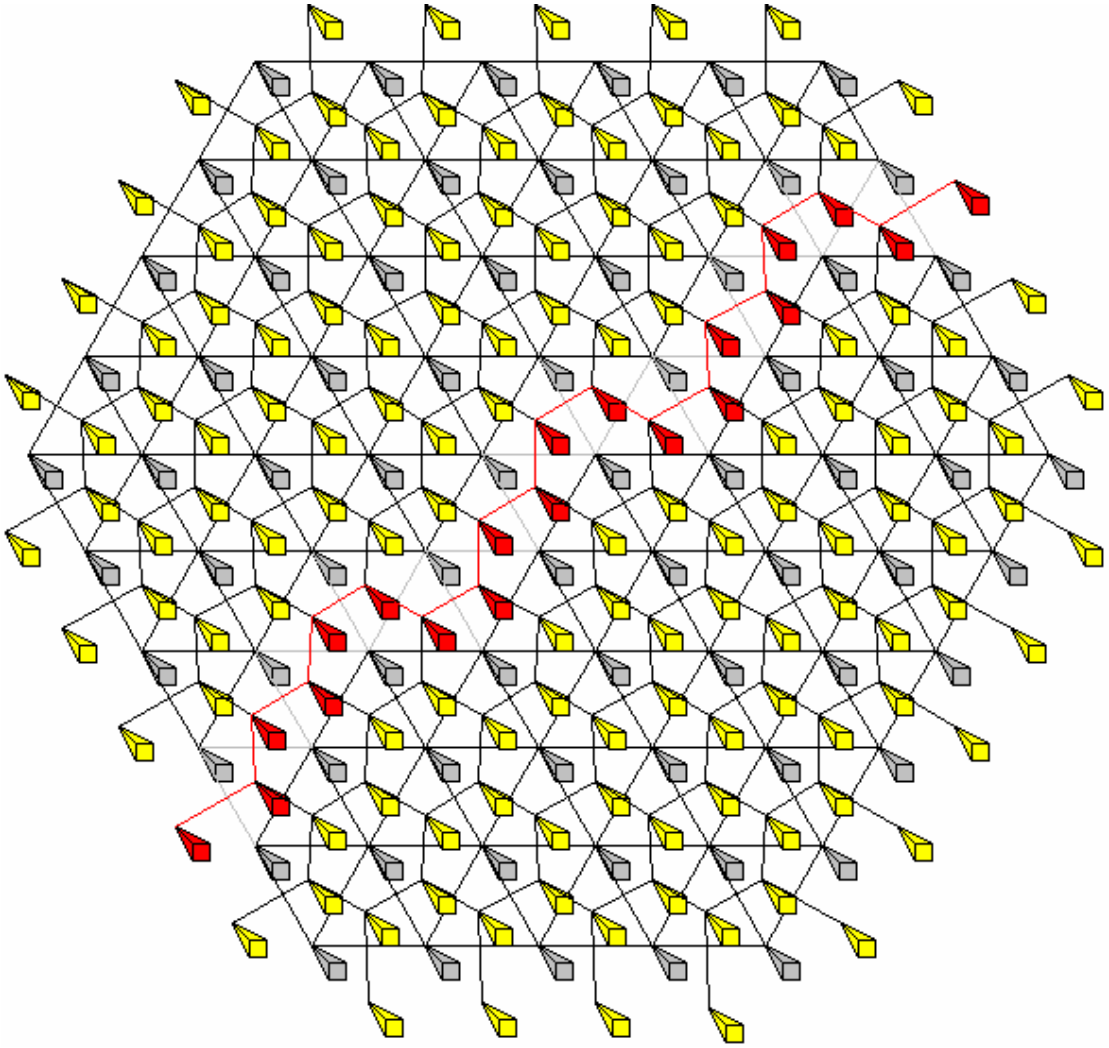

Figure 1.Dual Graph Partitioning example

2.3.2. Complexity

DGP is applied to the dual of a planar graph, embedded in the plane. The construction of a dual graph takes O(V) time. The complexity of DGP is governed by the complexity of the Dijkstra algorithm, computed over the number of dual graph nodes $|V_d|$ and edges $|E_d|$, the number of dual graph border nodes B, and the frequency of occurrence of better solutions. The Dijkstra algorithm costs $O((V_d + E_d)\cdot \log V_d)$. For a regular graph shape, whose height and width are of the same order of magnitude, it can be proved that the number of outer border nodes B is $O(V_d^{1/2})$. Hence, the cost complexity of creating all border to border paths, is $O(V_d^{3/2} \cdot \log V_d)$.

Considering, an undirected graph where there are no equal minimum routes from a node to a source, $(B^2\text{-}B)/2$ paths are created $((B\text{-}1) + (B\text{-}2) + ... + 1)$. The complexity is affected by the number of the corresponding partitions, whose size will be tested for compliance, that is, by the number of times these paths will have a cost lower to the current minimum partition. Note that the previous reasoning implies that there will be checked at least, all resulted partitions until the first which complies with the given imbalance. In the worst case scenario, all paths will be tested. As it was previously described, a BFS is used to define

sections, thus the cost of checking all possible partitions is $|V_d|\cdot(B^2-B)/2 = 2\cdot\pi\cdot|V_d|^2 - \pi^{1/2}\cdot|V_d|^{3/2}$ which is of $O(V_d^2)$.

In practice not all paths will be checked, however, this number is subject to the number of border nodes which are inside each source's neighborhood at a distance equal to the current minimum partition cost. Nevertheless, this version of DGP does not guarantee that a compliant partition will be found, in which case all paths will be checked.

In case where there are multiple minimum routes from a node to a source, the worst case cost of the algorithm would be exponential. In this work, such a case was highly improbable, since graph weight values had a double precision decimal part, apart from their integer range. However, the multiple route variation of the Dijkstra algorithm was used in case such an improbable event occurred.

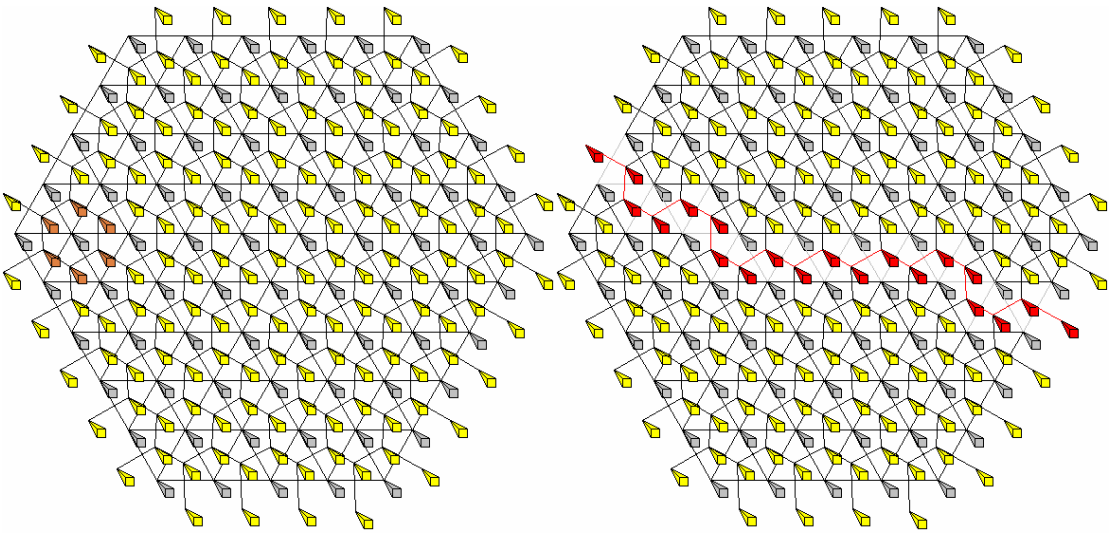

Figure 2. DGP variation example

2.4. Variation of the algorithm

2.4.1. Intermediate Region

A variation of DGP, using path combinations is also presented. Instead of testing border to border node shortest paths, let them be called **BB** shortest paths, a set of intermediate nodes is defined from which minimum paths to all border nodes are computed. For each node of this set, all pairs of paths to border nodes are combined to form a partition. If the sum of the cost of the two paths is less than the current minimum partition cost, the resulting partition is checked for compliance with the given imbalance. If the resulting sections comply, the partition cut and cost are updated.

The example graph of Figure 1 is partitioned using the variation of the algorithm in Figure 2, using the same tolerance. On the left instance of this graph, the orange nodes were chosen as the set of intermediate nodes from which, paths to all border nodes would be combined in pairs to form a partition. On the right instance of the graph the effect on the resulting partition is presented. The resulting edge cut was different since the border to border paths were formed via the intermediate nodes which did not belong to the resulting path of Figure 1. Hence, this allows one to "guide" the algorithm's search, as well as to form paths that the original version doesn't check, i.e, paths which include nodes that they do not belong in any border to border node shortest path.

2.4.2. Complexity

The choice of the set of intermediate nodes must be such that, the number of paths to be checked will not be prohibiting, yet it should be able to check all paths that the original version of the algorithm would check and if possible, to guarantee the discovery of a solution. For example, a set of intermediate nodes which consists of all internal nodes of the graph would ensure that a solution would be found, however, the cost of checking all resulting partitions would be raised at an $O(V_d^3)$. In this work, the number of intermediate nodes was bounded to an $O(V_d^{1/2})$. Thus, the cost of producing all border to border paths remained the same as the original version, since the Dijkstra algorithm was run $O(V_d^{1/2})$ times.

There are $(B^2-B)/2$ paths to be checked for each intermediate node, that is $O(V_d^{1/2})$ more times than the original algorithm. Thus, the cost of checking all resulting partitions can reach an $O(V_d^{5/2})$, in the worst case.

The location of these nodes affects which paths will be checked. Creating the intermediate set of nodes by choosing all nodes across a pair of vertical lines that cross in the center of the graph, ensures that if there is a BB shortest path that bisects the graph, it will definitely consist of at least one of these nodes. For a symmetrical graph shape, this set of nodes has a size of $4 \cdot R = 4 \cdot (|V_d|/\pi)^{1/2}$, which is indeed of an $O(V_d^{1/2})$.

Another choice is to create the intermediate node set, by choosing the nodes on the perimeter of a circle, which contains $|V_d|/2$ nodes. A BB path which optimally bisects this graph will definitely contain one of these intermediate nodes. For a near optimal bisection, that is, a small imbalance, this inner circle can be enlarged so that the perimeter of this circle can contain the additional nodes to guarantee that all existing BB bisecting shortest paths will be checked.

The size of this intermediate node set is $2 \cdot \pi \cdot r$, r being the radius of the inner circle. It holds that $\pi \cdot r^2 = |V_d|/2 \Rightarrow r = (|V_d| / (2 \cdot \pi))^{1/2}$ and thus the set size has an $O(V_d^{1/2})$.

In this work the intermediate node set was constructed with the former technique (vertical lines). Creating an intermediate node set that guarantees that a partition result will be found greatly reduces the probability the algorithm run time reaches the worst case order of magnitude.

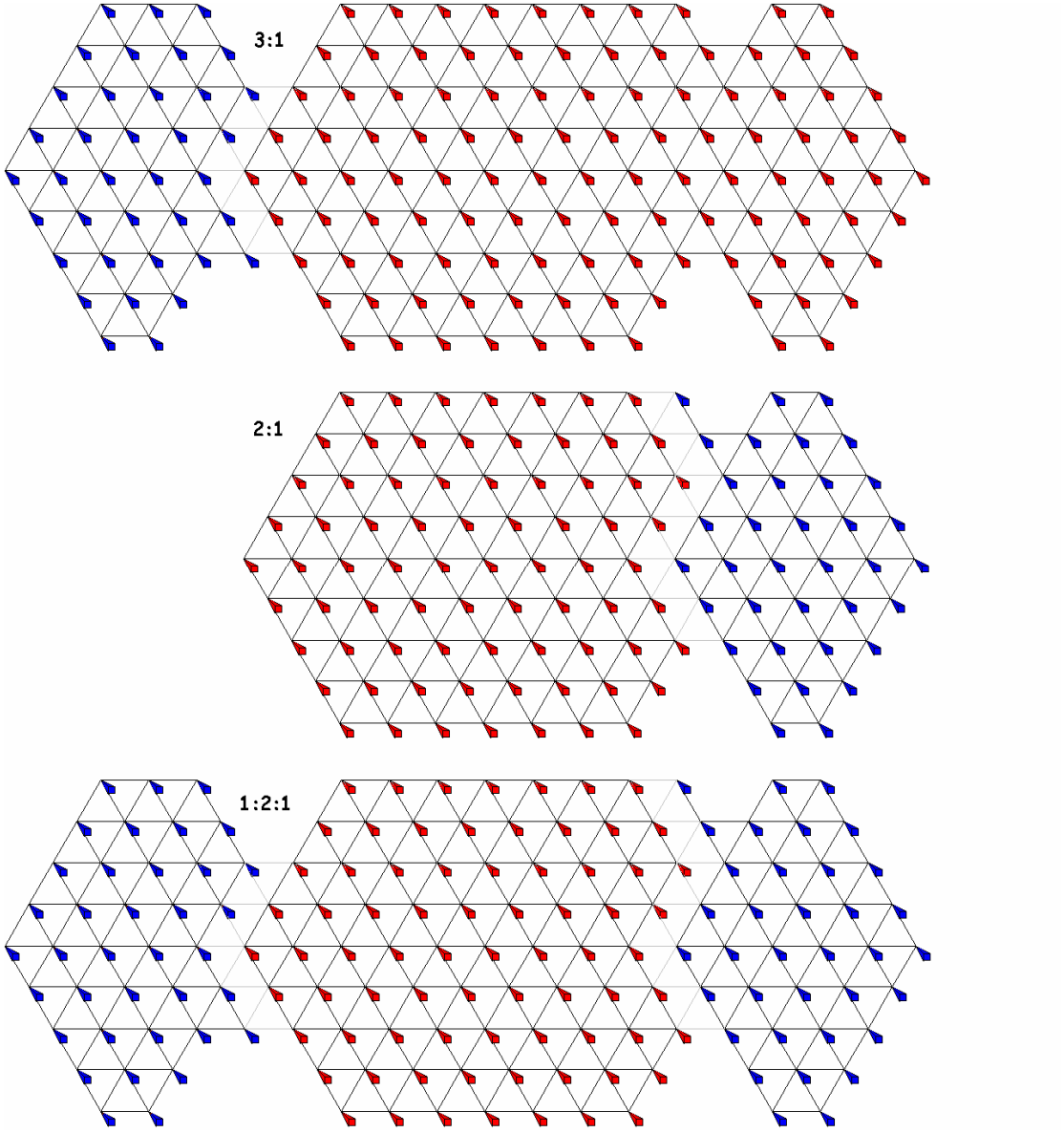

Figure 3. Creating disconnected sections using DGP

2.5.Analysis

2.51.. Motivation and Assumptions

DGP can only be applied to planar graphs, due to its described inherent characteristics. Therefore, it is applicable to problems concerning such graph structures. Moreover, its primary advantage, speed, implies that DGP is favorable for problems posing real-time constraints. A practical application which is closer to these characteristics concern wireless communication networks.

In large scale wireless networks, such as mobile telephony networks, the underline graph structure that represents the wireless service endpoints is inherently planar: In a cellular network, cells represent the coverage area of transceivers, which are modeled with graph nodes. A mobile client can leave the coverage area of a transceiver *a* and enter the one of a neighboring transceiver *b*. The adjacency of *a* and *b* is modeled with an edge and the number of mobile client transitions from the coverage area of *a* to that of *b* and vice-versa, is modeled with the edge weight. Due to the nature of the wireless communication medium, overprovisioning the network with wireless endpoints results in lower or no-connectivity; there is interference and collision of communications. Thus, wireless endpoints are positioned uniformly by triangulating the covered space, as to minimize the number of transceivers used. This, results in an underline planar graph structure, where each transceiver is adjacent to at most six other transceivers, modeled with a graph having a hexagonal grid structure. An example of this popular representation of wireless networks is presented in Figure 1, where grey nodes represent wireless endpoints and edges between nodes reflect transceiver adjacency.

A partitioning problem which is applied to such networks concerns grouping the wireless endpoint devices in order to scale the network without compromising the provided service. This is achieved with a hierarchical aggregation of resources. Transceivers are assigned in groups, to switches, which in turn are assigned to routers. When a mobile clients *c*, served by transceiver *a*, enters the coverage area of a transceiver *b*, *b* informs the system that it will serve the client. If *a* and *b* are assigned to different switches, or even in different routers, extensive signaling operations are performed in order to reroute information towards *c*, via another router and switch. Hence, neighboring transceivers between which there's a lot of client migration, must be assigned to the same switch and vice-versa. Graph partitioning aids in minimizing inter-switch and inter-router signaling communications.

The real-time constraints posed by such an application come from networks with very mobile clients or networks which demand a flexible rearrangement of the connectivity structure for security reasons. In large-scale networks, wireless endpoints can be assigned in Virtual LANs automatically, in order to respond to high client mobility towards different areas, during a day. In wireless networks designed to "follow" very mobile users, devices can be activated and deactivated for resource and power efficiency, causing a real-time transformation of the underline graph structure. In such cases, designing an efficient hierarchical aggregation of resources using a fast procedure, is essential.

In this work, the artificial graphs on which algorithm will be tested are chosen from its closest application domain. Thus, graphs will be constructed as to be planar hexagonal grids and edge weights will be concerned as mobile client transitions between nodes. However, we don't constraint the use and analysis of the algorithm only in this domain. Graph characteristics and their effect on the algorithm will be neutrally presented, along with an additional explanation from the wireless application domain point of view.

2.52.. Correctness

The algorithm is correct in terms of finding a bisection. In the context of graphs representing wireless networks, graphs are embedded in the plane and therefore, the outer border of a graph is well defined. For every planar graph embedded in the plane, in the dual graph the outer face node with degree d_o can be replaced by d_o special outer border nodes. Every edge between two nodes also separates two faces and thus, every border-to-border (BB) acyclic path in a dual graph represents a BB cut in the original graph, which divides the graph in two connected subgraphs. A minimum BB path corresponds to a minimum BB edge cut and consequently, a minimum BB path that separates the original graph in two equal subgraphs corresponds to a minimum BB bisection in the original graph.

2.53.. Completeness

The problem of k-way partitioning is an NP-Complete problem. To the best of our knowledge the bisection problem and its application on planar or bounded-degree graphs, such as the ones belonging to the application domain of our interest, belong to the same class of problems. The sole guarantee that exists in the literature concerns proposed solutions which offer approximated upper bounds on the solution quality with respect to the optimal solution, concerning planar and m-regular graphs.

In this work an optimal solution cannot be guaranteed since only a subset of the possible BB paths is checked. The size of this subset differs from one graph structure to another, from different embeddings, and from the choice of the number of intermendiate nodes, thus, we don't provide an upper bound on the quality approximation ratio.

The original form of DGP tests the section sizes that result from all the BB shortest paths. It cannot guarantee that there will always be a path that bisects the original graph with the given imbalance. Given that there are such paths, the algorithm can guarantee that it will find the minimum. This work investigates the characteristics of the graph that affect the existence of such paths.

The variation of DGP can guarantee the discovery of all paths that the original version would return and generally, the discovery of a path, under certain conditions concerning the choice of intermediate nodes, which will also be analyzed in this work.

2.54.. Characteristics

2.54.1. Border to Border vs. Internal Partition

The algorithm can only produce BB cuts. It cannot find internal bisections, since they correspond to cycles in the dual graph that do not contain border

nodes. In wireless networks it is generally undesired to use internal partitions, since in small scales, mobile clients that move along a route that goes through an external and internal partition would always have to cross the partition borer twice.

However, there might be geographical limitations that might favor an internal partition in larger scales (e.g., the existence of a plateau). In order to estimate the importance of this limitation we compare a BB bisection cost with an internal bisection cost.

The worst case scenario of a BB bisection will be compared against the best case scenario of an internal bisection, with respect to graph shape, as well as maximum and minimum weight in the case of a weighted graph.

Assuming a regular, circular, unweighted grid, an optimal partition would cost twice its radius *R*. The assumption of a symmetrical shape ensures that there would be no outer border nodes, which would favor the border to border partitioning. Let *N* be the size of the dual graph, it then holds

$$N = \pi \cdot R^2 \Rightarrow R = \left(\frac{N}{\pi}\right)^{1/2}$$

Now let there be an internal partition which bisects the graph. The partition structure that minimizes the number of edges is a cycle; let it be centered on the center of the graph and have a radius *r*. It holds that

$$\frac{N}{2} = \pi \cdot r^2 \Rightarrow r = \left(\frac{N}{2 \cdot \pi}\right)^{1/2}$$

The internal partition cost equals $2 \cdot \pi \cdot r$. The ratio of the BB partition and the internal partition is then:

$$\frac{2 \cdot R}{2 \cdot \pi \cdot r} = \frac{2^{1/2}}{\pi} < 1$$

Thus, a BB cut consists of $(2^{1/2}) / \pi$ fewer edges than the internal one, in the case of an unweighted graph

In a weighted such graph, consider the extreme scenario that favors the internal partition, that is, all internal partition edges have the minimum weight, *minW*, and all other edges the maximum weight, *maxW*. Then, the BB to internal partition cost ratio changes to

$$\frac{2 \cdot R \cdot maxW - 2 \cdot maxW + 2 \cdot minW}{2 \cdot \pi \cdot r \cdot minW}$$

In order to prove when $2 \cdot R \cdot maxW - 2 \cdot maxW + 2 \cdot minW < 2 \cdot \pi \cdot r \cdot minW$, in which case the BB cost is lower than the internal cost, it is sufficient to prove when

$$2 \cdot R \cdot maxW < 2 \cdot \pi \cdot r \cdot minW \Rightarrow$$
$$\frac{maxW}{minW} < \frac{\pi \cdot r}{R} \Rightarrow$$
$$\frac{maxW}{minW} < \frac{\pi}{2^{1/2}}$$

Thus, BB partition will have a lower cost than an internal partition if *maxW* is at most $\pi / 2^{1/2}$ times the *minW*.

Note that this max/min weight ratio upper bound value $\pi / 2^{1/2}$, is practically higher, since the BB cut cost is less than assumed in the initial inequality. Moreover, note that the distribution of weights which was chosen is extremely improbable and the choice of both graph and inner partition shapes were considered circular since this favors the inner partition (ensures the minimum number of edges) and disfavors the BB partition (there are no end points of a bisecting BB path that are considerably closer than other end points). Thus, the value of $\pi / 2^{1/2}$ is a lower bound on the practical maximum max/min weight ratio that guarantees that a BB bisection has a lower cost than an internal one.

2.54.2. Contiguous Sections

Another characteristic of the algorithm is that it produces partitions with contiguous nodes, i.e., the subgraphs defined by the two resulting sections are connected. This is a precondition in certain graph partitioning applications such as VLSI design or can be a desired attribute for some network applications. However, non-contiguous partitions can be created implicitly, if needed. For example, the original graph can be bisected with an imbalance which yields a section size ratio 1:3 and then subsequently bisect the greatest section with a 2:1 ratio, for one disconnected section. There will be three sections with size fractions 1/4, 2/4, 1/4. The smaller sections can be considered as one disconnected section. Figure 3 depicts this example on a graph, were the disconnected section partitioning reduced the number of cut edges by one. Nodes of same color belong to the same section.

Chapter 3

Experiments using DGP

3..1 Testing hypothesis that *nodeRatio* ≤ 3 ensures bisectability

3..1.1. The *nodeRatio* hypothesis

We create a number of artificial graph sets, in order to test how graph properties affect the partition result. In parallel, the effect of tolerance is analyzed. The structure of these graphs is that of a hexagonal grid, which is often used in mobile network representation. A node is centered at a given point and nodes are subsequently connected at its perimeter, at a spiral order, at positions which respect the hexagonal grid structure. Nodes are subsequently connected at the perimeter of the current shape until the graph has the requested size. From that point, edge weights can be assigned using different policies.

In [53], the author expressed its belief that a constraint on the max and min weight of the edges attached at each node could provide a guarantee that the algorithm would find a result. Thus, as a start point, we test the following hypothesis. For each node v, a certain upper bound on the max/min weight ratio of its attached edges E_v, let it be called *nodeRatio*, can guarantee a bisection result.

3..1.2. Weight assignment

In order to test this hypothesis we must assign weights in a way that for each node, the max/min ratio of the weights of its attached edges would be at most a threshold *nodeRatio*. Thus, if $maxW(E_v)$ and $minW(E_v)$ return the maximum and minimum weights of the edges E_v attached to node v, it must hold for each v:

$$\frac{maxW(E_v)}{minW(E_v)} \leq nodeRatio \ .$$

Moreover, it is desired that the absolute weight values vary, if not significantly at least substantially, between different neighborhoods of the graph. For example, *minW* and *maxW* of the attached edges for a node *A* could be 1000 and 3000 respectively, and the same numbers could be 5000 and 15000 for another node *B*. A *nodeRatio* = 3 invariance is respected, although weights can differ even by an order of magnitude.

In order to achieve this, the notion of "population center" nodes is introduced in the weight assignment procedure. The graph is considered to have some population centers where the population is greater and decreases as the distance from these nodes increases. Consequently, the edge weights, which reflect movement of this population between nodes, vary in value with respect to their hop distance from their closest population center. The number of population centers can be considered as a given parameter in the graph construction procedure, along with the global range of edge weight values, let say [*maxWeight*, *minWeight*) (positive axis goes towards left). The nodes which

would be considered as population centers are chosen randomly among the set of all nodes of the graph. The goal is to assign weight values to edges according to their distance in hops from their nearest population center, drawn from a global range of values. As an immediate consequence, the global edge weight value range has to be split in a number of bins. This number of bins depended on the maximum of the distances of all nodes from their nearest population center.

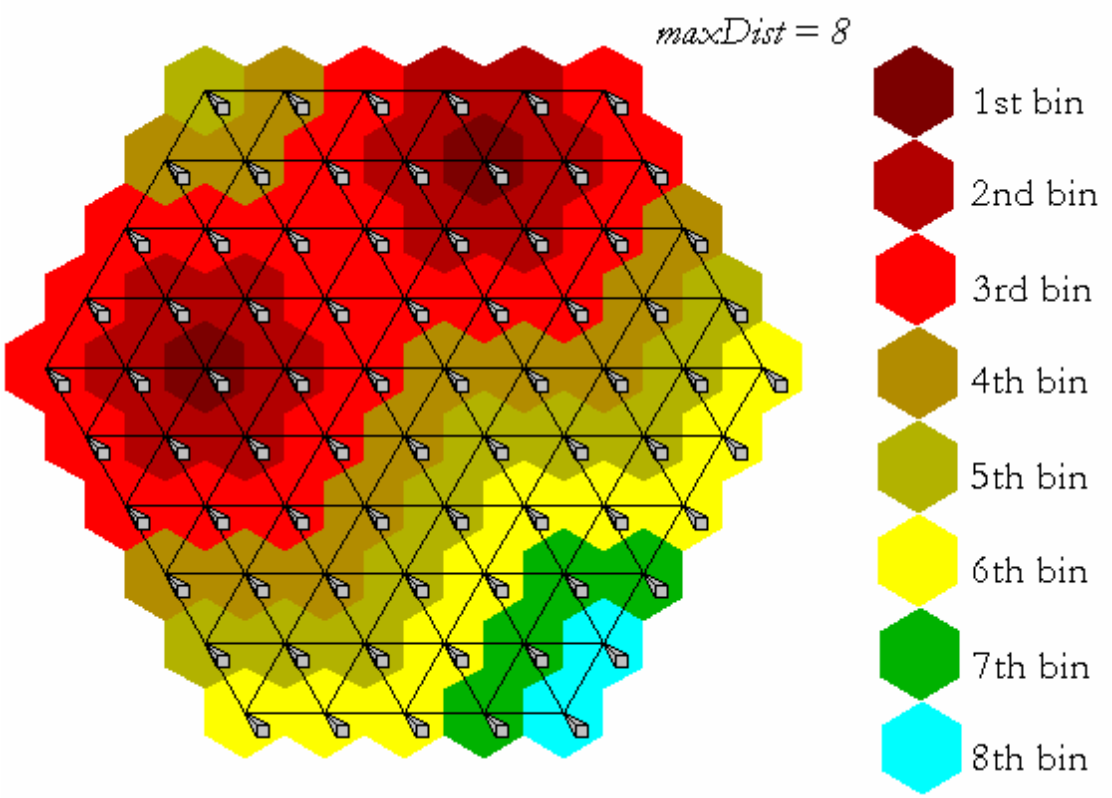

Figure 4. *maxDist* discovery and bin assignment example

In order to find this maximum distance, let it be called *maxDist*, a pseudo-parallel BFS is performed from each population center. Each population center is inserted to a corresponding FIFO and one neighbor node at a time is inserted. An edge's distance, in hops, from a population center is then considered to be the lowest of the distances of its attached nodes. Each node is inserted once in a FIFO. Since a node can be located at the border between two population centers, correction actions are performed when visiting such nodes, including re-insertion of these nodes in the current FIFO.

Once *maxDist* is computed, weight value bins must be created. Each edge is located a number of hops, '*i*', away from the closest population center (i.e, the population center's attached edges are at 1st hop) and its weight value will be drawn from the *i*-th bin which contains values which decrease as *i* increases. In Figure 4, an example of maxDist detection and bin assignment is presented for a graph with two population centers. Nodes centered on dark red hexagons are the population centers. The colored hexagons behind each node represent the distance in hops from its closest population center and therefore, the bin from which weight values will be drawn for each edge.

These bins must be created in such a way that for every node, the max and min weight of its attached edges must be at most *nodeRatio*. Simply separating the global weight range in a number of *maxDist* equal bins, can lead to violation

of this restriction. As nodes are located at a certain distance from one or more population centers, a node's edges can correspond to, at most two, consecutive bins. Thus, we must ensure that

$$w_i \le nodeRatio \cdot w_{i+2} \text{ for each } i = 0, 1, 2, \ldots, maxDist ,$$

for consecutive bins $[w_i, w_{i+1})$ and $[w_{i+1}, w_{i+2})$ where $w_i > w_{i+1}$.

This can be guaranteed by forming bin extremes, beginning from max population and ending to min population, using a factor *b*:

$$w_{i+1} = \frac{1}{b} \cdot w_i \Leftrightarrow w_i = b \cdot w_{i+1} \text{ such that } b \le nodeRatio^{1/2}$$

Proof:

$$w_{i+2} = w_{i+1} \cdot \frac{1}{b} \Rightarrow$$

$$w_{i+2} \ge w_{i+1} \cdot \frac{1}{nodeRatio^{1/2}} \Leftrightarrow$$

$$w_{i+2} \cdot nodeRatio^{1/2} \ge w_{i+1} \text{ (a)}$$

and since

$$w_{i+1} \ge w_i \cdot \frac{1}{nodeRatio^{1/2}} \Leftrightarrow$$

$$w_{i+1} \cdot nodeRatio^{1/2} \ge w_i \text{ (b),}$$

(a) and (b) lead to

$$w_{i+2} \cdot nodeRatio \ge w_i$$

which is the intended invariant.

Thus, for *k* bins (*k=maxDist*) we have:

1st bin: $\left[w_0, \quad w_0 \cdot \frac{1}{b^1} \right)$

2nd bin: $\left[w_0 \cdot \frac{1}{b^1}, \ w_0 \cdot \frac{1}{b^2} \right)$

3rd bin: $\left[w_0 \cdot \frac{1}{b^2}, \ w_0 \cdot \frac{1}{b^3} \right)$

...,

k-th bin: $\left[w_0 \cdot \frac{1}{b^{k-1}}, w_0 \cdot \frac{1}{b^k} \right)$

The resulting weight bins are unequal in width, in order to respect the *nodeRatio* invariant. This variance between bin widths is depicted in Figure 5. Bin width is decreasing as distance from the closest population center increases. For the depicted example, for a *maxWeight* = 1200 and $b = nodeRatio^{1/2} = 2$, the first three bins correspond to [1200, 600) [600, 300) and [300, 150).

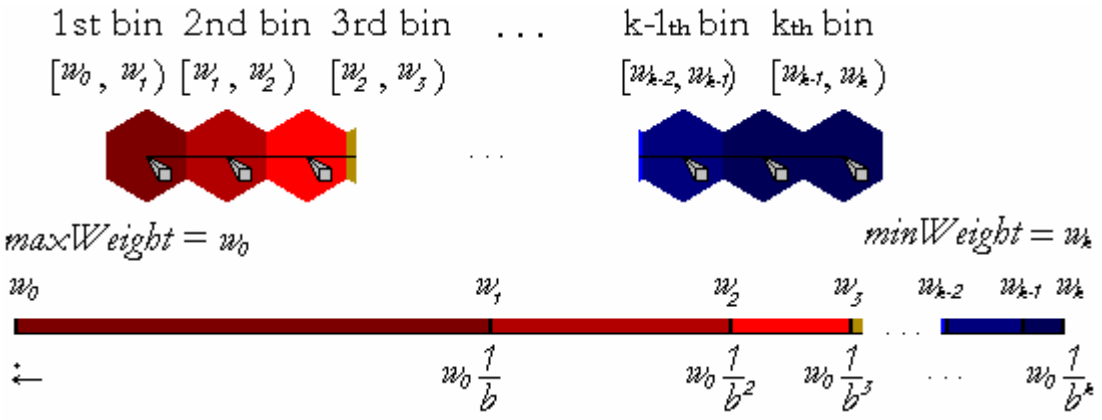

Figure 5. Weight bins respecting *nodeRatio* invariant

Given the weight global range [*maxWeight*, *minWeight*), the *maxDist* = *k* value (for clarity we use symbol *k* instead of *maxDist*) and that $w_{i+1} = w_i \cdot (1/b)$, and since $w_0 = maxWeight$, $w_k = minWeight$, the decreasing factor *b* is computed as follows:

$$w_k = w_0 \cdot \frac{1}{b^k} \Leftrightarrow$$

$$b = \left(\frac{w_k}{w_0}\right)^{1/k} \Leftrightarrow$$

$$b = \left(\frac{maxWeight}{minWeight}\right)^{1/k} \quad \text{(c)}$$

For example, for *maxWeight* = 1200, *minWeight* = 75, *nodeRatio* = 4 and for a graph were *maxDist* = 4, the decreasing factor $b = (1200/75)^{1/4} = 2$, and indeed $b \leq nodeRatio^{1/2}$.

Note that *b* depends on the *maxDist* value. In the case of a high *maxDist* value it is quite probable that $b < nodeRatio^{1/2}$, and thus $w_{i+2} < (1/nodeRatio) \cdot w_i$. Yet, we want w_{i+2} to be able to be up to $(1/nodeRatio) \cdot w_i$. In other words, we explicitly want the ratio between max and min weight of a node's attached edges to be able to reach the value of *nodeRatio* threshold. In the previous example, a *maxDist* = 8 would yield a decreasing factor $b = (1200/75)^{1/8} = 2^{1/2}$. Therefore $b << nodeRatio^{1/2}$ and $w_{i+2} << (1/nodeRatio) \cdot w_i$, for any *i*.

Therefore, we use overlapping bins by extending the lower extreme w_i of each *i*-th bin $[w_{i-1}, w_i)$, in order to form a ratio w_i / w_{i-2} equal to *nodeRatio*, with the higher extreme w_{i-2} of the previous bin $[w_{i-2}, w_{i-1})$. Thus, each *i*-th bin starts at $w_{i-1} = w_0 \cdot (1/b)^{i-1}$ interval, but it does not end at $w_i = w_0 \cdot (1/b)^i$, rather than at $w'_i = w_{i-2} \cdot (1/nodeRatio)$, and thus $w'_i / w_{i-2} = nodeRatio$. The lower extreme of the first bin needs a special calibration since there isn't any previous bin. We chose the value of $1/nodeRatio^{1/2}$ as the decreasing factor of w_1, since this is the maximum possible value of *b* that guarantees the *nodeRatio* invariant.

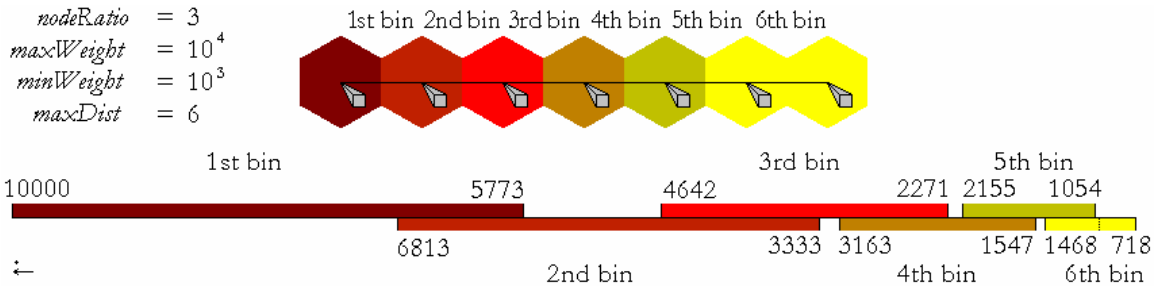

Figure 6. Overlapping weight bins example

This way, for $k = maxDist$, overlapping bins will have the following form:

$$\text{1st bin:} \quad \left[w_0, \quad w_0 \cdot \frac{1}{nodeRatio^{1/2}} \right)$$

$$\text{2nd bin:} \quad \left[w_0 \cdot \frac{1}{b^1}, \quad w_0 \cdot \frac{1}{b^0} \cdot \frac{1}{nodeRatio} \right)$$

$$\text{3rd bin:} \quad \left[w_0 \cdot \frac{1}{b^2}, \quad w_0 \cdot \frac{1}{b^1} \cdot \frac{1}{nodeRatio} \right)$$

...,

$$k\text{-th bin:} \quad \left[w_0 \cdot \frac{1}{b^{k-1}}, \quad w_0 \cdot \frac{1}{b^{k-2}} \cdot \frac{1}{nodeRatio} \right)$$

In Figure 6, overlapping weight bins are depicted for an example case of $maxWeight = 10^4$, $minWeight = 10^3$, $nodeRatio = 3$ and $maxDist = 6$. Note that $w_k < minWeight$, in which case the last bin is truncated to $[w_{k-1}, minWeight)$.

Apart from the bounds that are respected, bins formed this way provide greater diversity in the weight distribution, between population center neighborhoods. Non-overlapping bins result in weights whose value decreases monotonically with distance. Assuming such a strict pattern when modeling any kind of traffic (such as handover operations between cells) is not realistic. Overlapping bins allow the existence of additional local maxima of weight values, apart from the maxima that exist in the neighborhood of every population center.

Finally, note that since $b = (maxWeight/minWeight)^{1/k}$, b also depends on $maxWeight/minWeight$ ratio. In case of a low $maxDist$ value, it is not guaranteed that $b \leq 1/nodeRatio^{1/2}$. There are cases for which we cannot guarantee that max to min node weight ratio will be less or equal to $nodeRatio$, with the given population range and population distribution pattern. The $nodeRatio$ invariant is satisfied if

$$\frac{maxWeight}{minWeight} \leq \left(nodeRatio^{1/2}\right)^k, \quad k = maxDist$$

Thus, if the $minMaxRatio$ threshold is not satisfied, the $minWeight$ is set to the closest value that guarantees it, which is

$$minWeight' = \frac{maxWeight}{\left(nodeRatio^{1/2}\right)^k}$$

and the global weight range is adjusted to $[maxWeight, \ minWeight']$.

3..1.3. Testing the *nodeRatio* hypothesis

We tested whether setting *nodeRatio* to an empirical value of 3 could ensure that DGP would always find a partition, allowing a 20% tolerance with respect to the optimal section size. We generated a graph set consisting of about 21000 graphs of 200 nodes each, with global minimum weight of 10^3, global maximum weight 10^5, and the number of population centers varied from 5-25% of graph size. Here follow the partition results.

The algorithm didn't manage to find a bisection for approximately 1% of the created graphs, when the max allowed tolerance was 20% of the optimal section size. In Figure 7, the fraction of graphs which were successfully bisected is presented, for each tolerance value. Tolerance is defined as the number of nodes a section can differ from an optimal bisection section. In the plot tolerance is normalized by the optimal section size.

As the CCDF (Complementary Cumulative Distribution Function) plot of tolerance for which graphs were bisected shows in Figure 8, 70% of the graphs were bisected for a tolerance over 8% of the optimal section size, which in turn means that 30% of the created graphs were bisected with an tolerance of at most 8%. Note that a CDF plot of a value distribution of a metric, shows in the *y* axis the fraction of times this metric has a value less or equal to the *x* value. Consequently, a Complementary CDF plot of a value distribution of a metric, shows in the *y* axis the fraction of times this metric has a value over *x*.

We conclude that assuming a ratio of max to min weight equal to 3 at each node, does not guarantee bisectability, since 1% of graphs were not bisected.Additionally, four more sets of 500 graphs each, were created to verify whether the *nodeRatio* vaffects the partition result. The graphs consisted of 300 nodes. *minWeight* was set to 10^3 and *maxWeight* to 10^5. The *nodeRatio* was set to 2, 4, 5, and 6 respectively for each set. The number of population centers was set to 5% of graph size. The algorithm was run allowing a tolerance of 20% of the optimal partition size.

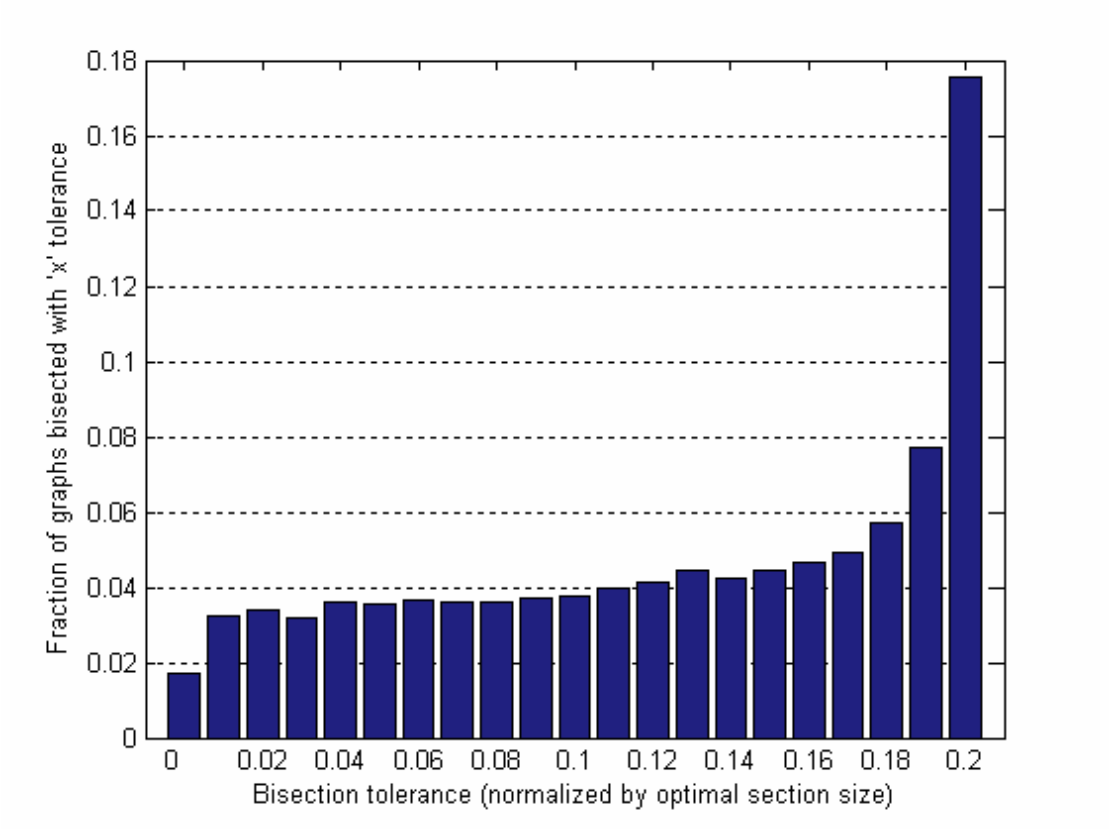

Figure 7. Partitioned graphs per tolerance

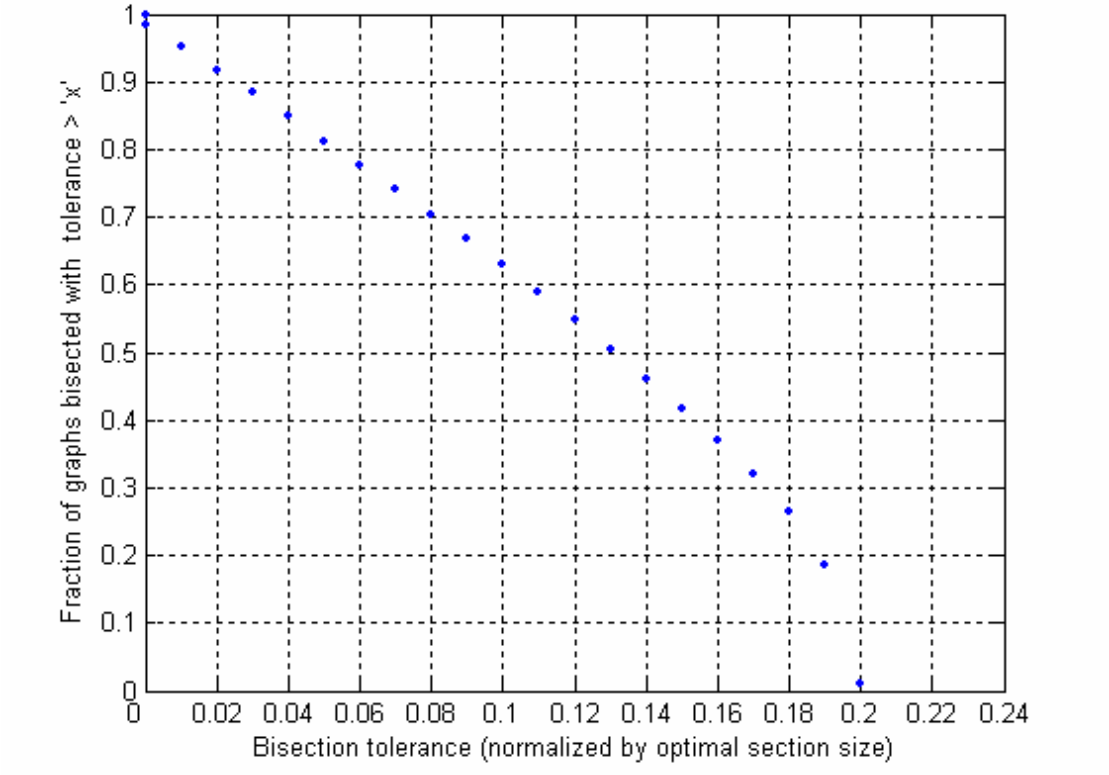

Figure 8. CCDF of partitioned graphs per tolerance

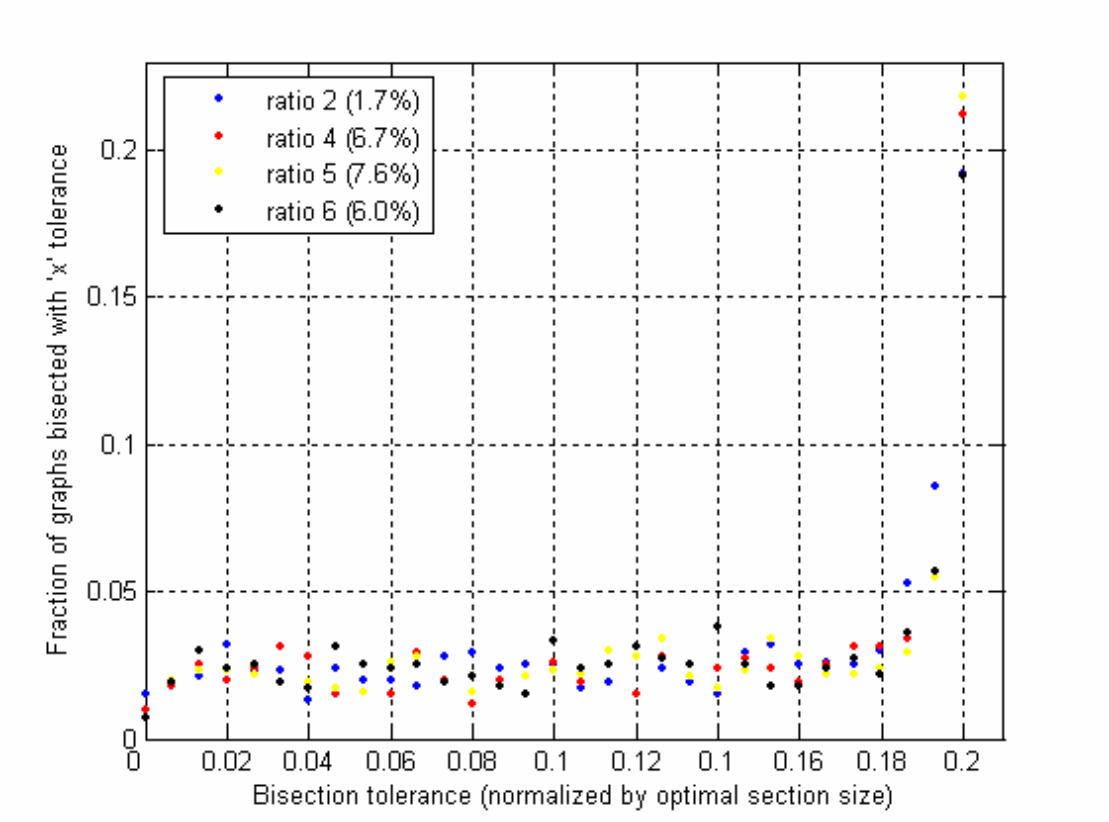

Figure 9. Partitioned graphs per tolerance

A lower percentage of graphs were unpartitioned for lower *nodeRatio* values. This can be related with the *minWeight* calibration at the weight assignment process, as it was described in section 3..1.2. It must hold that $nodeRatio^{1/2} \geq (maxWeight/minWeight)^{1/maxDist}$, a condition which can hold for greater *nodeRatio* values, but not for lower ones and whence, *minWeight* is increased. However, there is not a specific trend. As Figure 9 shows, a *nodeRatio* value of 2 resulted in 1.7% unbisected graph instances, whereas this percentage decreased to 1% for a ratio of 3. Similarly, although ratio 4 resulted in less unbisected graphs than ratio 5 (6.7% and 7.6% respectively), ratio 6 exhibited a lower percentage from both previous experiment sets (6%). Finally, we observe that the algorithm exhibits a similar performance for all sets, since the percentage of graphs bisected at max allowed tolerance is around 20% for all sets. The rest percentages are distributed at lower tolerance values.

In the context of wireless networks, we can conclude that we can assume that the ratio of max to min transitions of users between a node and its neighbors has an upper bound of 3, since there's no such value that guarantees a partition result.

3..2 Using PRAGMA Mobility Model to infer *nodeRatio*

3..2.1. PRAGMA Description and Setup

It was shown that no certain *nodeRatio* value can guarantee graph bisection, yet it can affect it. Instead of relying solely on an empirical value of *nodeRatio*, we tried to infer it using a mobility model. In the context of wireless networks, we considered the signaling communication cost between two wireless endpoints *a* and *b*, as the number of transitions of mobile clients from the coverage area of *a* to that of *b* and vice-versa. In order to simulate such transitions with a mobility model, a population of individuals is positioned randomly over a hexagonal grid with unassigned weights. Each individual is corresponded to its closer node, subject to a maximum correspondence range. As individuals move in space their corresponding node changes between neighboring nodes. Such an event increases the corresponding edge weight by a unity. After a specific amount of simulation time the graph can be considered populated with a representative distribution of weights.

We chose the PRAGMA mobility model presented by Borrel et al. [82], since it fitted best with our model of assigning weights in artificial graphs, which is based on the hypothesis that there are areas with more people than other places. Similarly, as described in [82], the PRAGMA mobility model consists of individuals, which move in space and attractors, which are fixed points of interest. Each individual '*I*' receives an attraction $A(I)$ by each attractor '*A*', which is inversely proportional to its distance and proportional to the sum of all individuals currently attracted by '*A*'. Each individual chooses an attractor as destination. The probability to choose an attractor is proportional to the attraction it gets by it. Then the individual moves towards this attractor at a speed, whose value is randomly chosen from a given range. When the individual arrives at its destination it pauses for an amount of time, randomly chosen from a given range. When the pause time elapses the individual chooses whether it will stay or disappear from the simulation space with a given 'stay probability'. In case it stays, it repeats the pre-described steps, called individual cycle. In Figure 10, an example of how PRAGMA is populating the weights of a graph is depicted.

Attractors have a lifetime randomly chosen from a given range. They disappear when their lifetime expires and in the meantime, other attractors appear following a Poisson process. This means that attractors arrive between exponentially distributed inter-arrival times. The mean of this distribution must be such to preserve the expected attractor size in the long run. In [82], authors leave the computation of attractor and individual lifetime, open. In practice, we used the average of maximum and minimum lifetime value as the distribution mean. We divided the produced inter-arrival times with the initial attractor size, since all initial attractors are expected to have disappeared after the maximum lifetime has elapsed.

Individuals also arrive between exponentially distributed inter-arrival times, however, the task of maintaining a desirable population size is not as straightforward. An individual's lifetime depends on the distance it has to cover until its destination, the speed at which it moves, its pause time, and the stay probability parameter, let it be called *stayProbability*. In order to compute the

maximum travel time, we considered the diagonal of the simulation space as the maximum possible distance. Then, we computed the maximum travel time using the min possible speed. We added the max possible pause time to produce an individual's max possible cycle time, *maxCycle*. We considered the individual mean lifetime to be equal to:

$$\frac{maxCycle \cdot (1 + stayProbability)}{2}$$

We divided the produced inter-arrival times by the initial individual population, since the pre-described computation concerned the lifetime of a single individual.

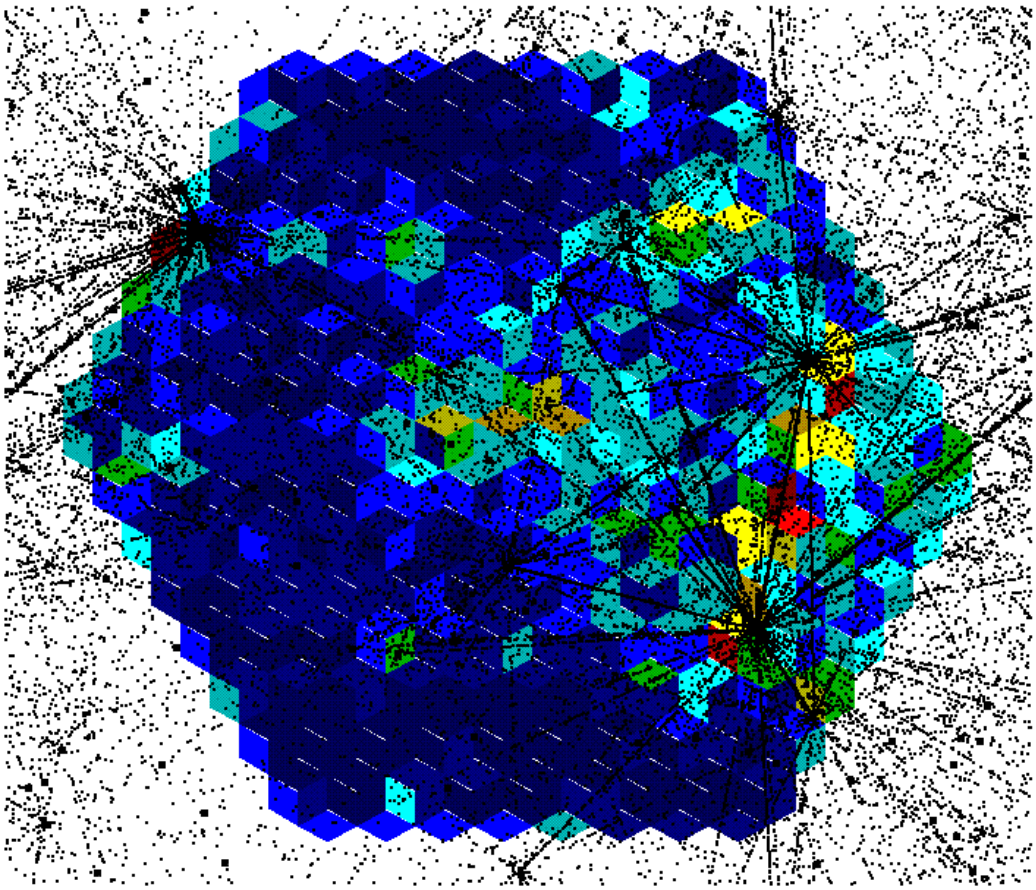

Figure 10. Instance of PRAGMA populating graph edges

3..2.2. Pragma Simulation Issues

Several issues arose while performing the simulation. Individuals tended to concentrate at the center of the bounded simulation space. This was solved by using a toroidal simulation space.

One of the most important issues concerned balancing individual population size. As it was shown, mean individual lifetime depended both on maximum possible speed and stay probability. Fine tuning these parameters was rather cumbersome and required a lot of simulation runs in order to successfully combine max speed with stay probability in a way that they wouldn't need to be tampered with, during the simulation. A small declination of stay probability

could result in, either a deserted simulation space, or a constantly growing individual population.

Another issue concerned the growth of attraction over time. In the long run, almost all individuals would be attracted by very few attractors. As these attractors became the destination of even more individuals over time, new individuals would always choose their destination among these attractors. It is true that PRAGMA model was designed to produce scale free topologies, however such an aggregation over time was undesirable for our purpose. A simple solution was to regulate attractor lifetime, yet it could only be reduced up to a point, being subject to used time-scale and its semantics. Moreover, this solution could not prevent the chain effect between attractors: when an attractor lived long enough to attract the majority of individuals, when it would disappear, the part of its population that chose to stay in the simulation would then migrate to one of the rest highly populated attractors. This is actually what it was supposed to happen, yet even for shorter attractor lifetimes the next attractor would always be at the imminent neighborhood. As a conclusion, an element which would provide some diversification in individual destination choice was missing from the model. One solution could be the addition of some mechanism of 'attraction aging' on individual's choice, that is, some attractors would become obsolete over time, although this could have a minor effect since individuals frequently disappear after they reach a destination. In order to control this undesirable effect, the simulation time length was restricted to shorter, yet meaningful periods (i.e instead of simulating a week we simulated a day, which is still a useful time period).

Last, there was an issue which mostly concerned the simulation conditions rather than the model. As we have previously explained, the created graphs had a circular-like shape. The PRAGMA simulation space was a rectangle containing the graph. Consequently, there was some space around the border of the graph where individuals could move being outside the maximum range of correspondence. When individuals moved along the border of the graphs, there would be only a couple of candidate nodes to correspond with, whereas in the interior of the graph there would be a greater probability that more nodes would correspond with the individual as it moves. This resulted in some border edges with higher weights than their immediate, internal neighbors.

3..2.3. PRAGMA Parameters and Simulation Results

We set the parameters of the simulation to represent human movement. We set distance unity to ten meters and time unity to one minute, which was also the simulation time step. We positioned nodes at a distance of 40 units (400 meters) and considered their maximum coverage range as 30 distance units (300 meters). We considered individuals to move at min and max walking speed of 0.5 m/sec and 1 m/sec respectively, thus, min and max individual speed were set to 3 and 6 distance units per time unit, respectively. We also considered an individual to pause at its destination for a time from 30 to 150 minutes (also time units). We desired a population of around 1000 individuals per node, as to produce a realistic simulation scenario. However, this limited the graph sizes between 60 and 200 nodes, due to the computational burden of updating info for up to 200000 individuals per time step.

We set attractor lifetime between 5 and 10 hours (300 - 600 time units), for most simulations. The min lifetime was altered to 60 and 150 minutes and the

max lifetime was set to 1500 time units (about a day), for some simulations. We set Population of attractors to correspond to three attractors per node in most sets. A ratio of 2 and 4 attractors per node was also tested in two graph sets. The resulting *nodeRatio* of each node, from 1000 created graphs, is presented in Figure 11, grouped by the attractor/node ratio.

We can observe that the empirical value of *nodeRatio* value at most 3 was not verified. However, results were not far from it for simulations that used a fair number of attractors per node. The majority of nodes (90%) had a *nodeRatio* value of at most 4. Thus we decided all tested graphs to be created as to have a similar distribution of *nodeRatio* values, with most such ratios to be at most 3 and allowing a small percentage to have greater values.

We also tested the algorithm performance. The percentage of unbisected graphs was somewhat higher from previous results, reaching 8%, using tolerance values up to 20% of the optimal partition size. The performance at each tolerance was also different, with most graphs to be partitioned with tolerance close to 20%: approximately 62% of the graphs were bisected at a tolerance over 16% of the optimal partition size. The corresponding PDF and CCDF plots of the fraction of partitioned graphs per tolerance are presented in Figures 12 and 13.

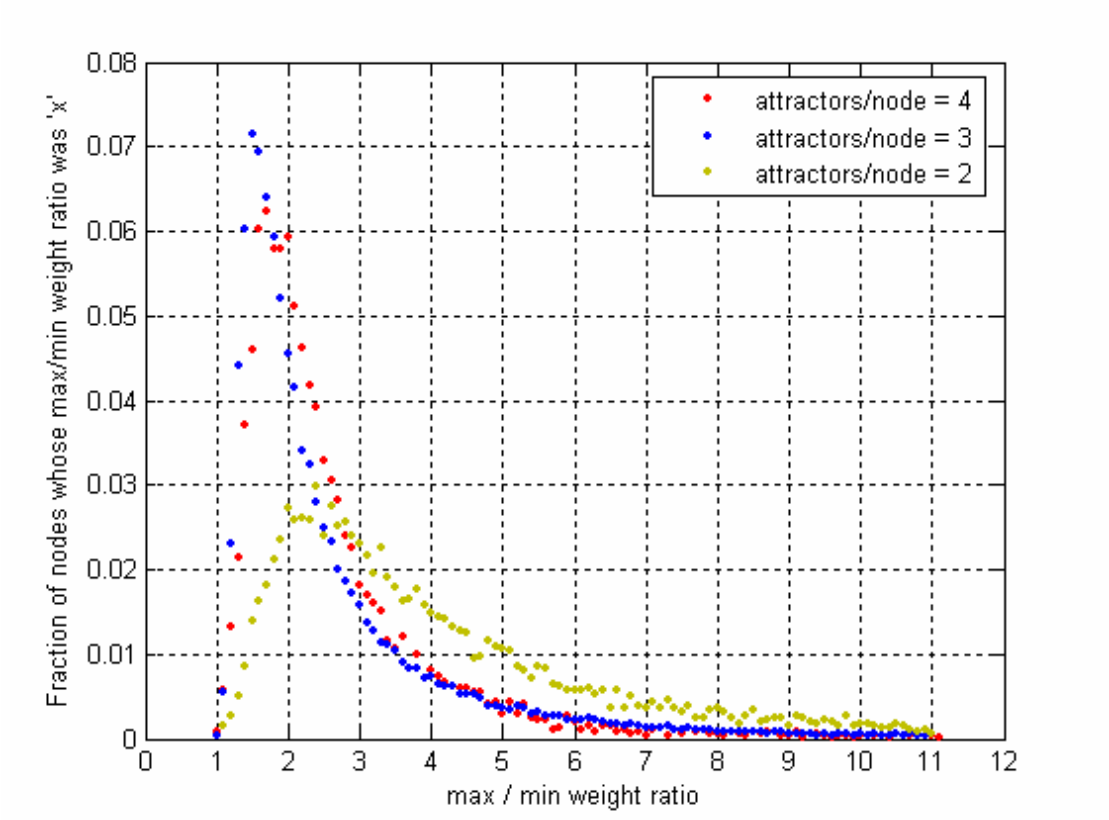

Figure 11. Node max / min edge weight ratio from PRAGMA generated graphs

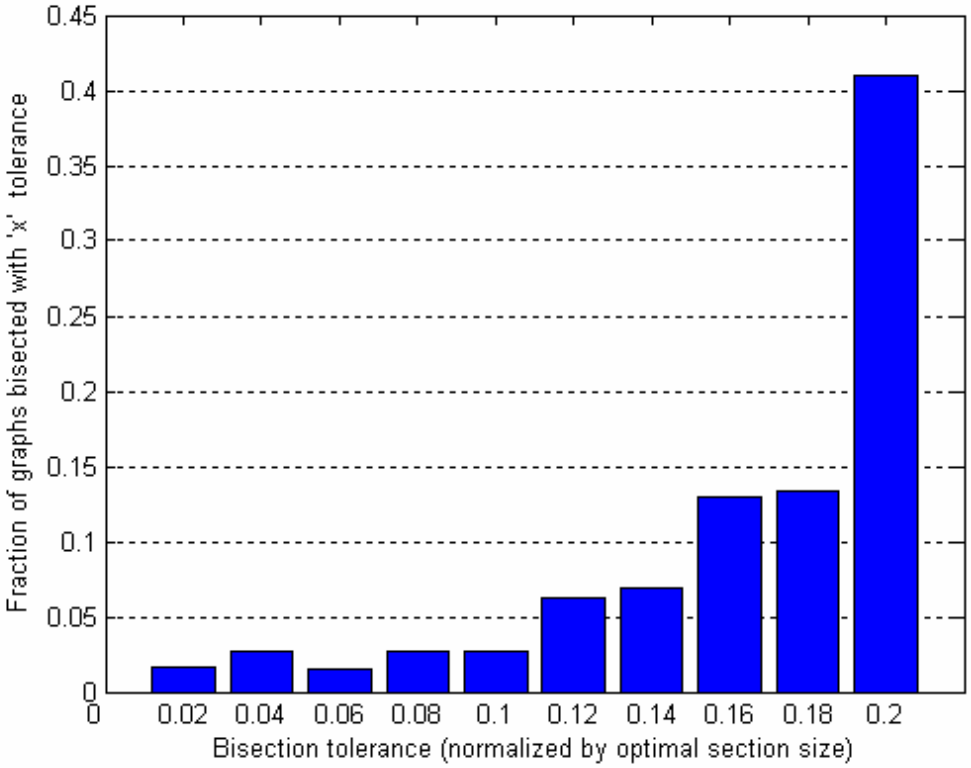

Figure 12. Partitioned graphs per tolerance for PRAGMA generated set

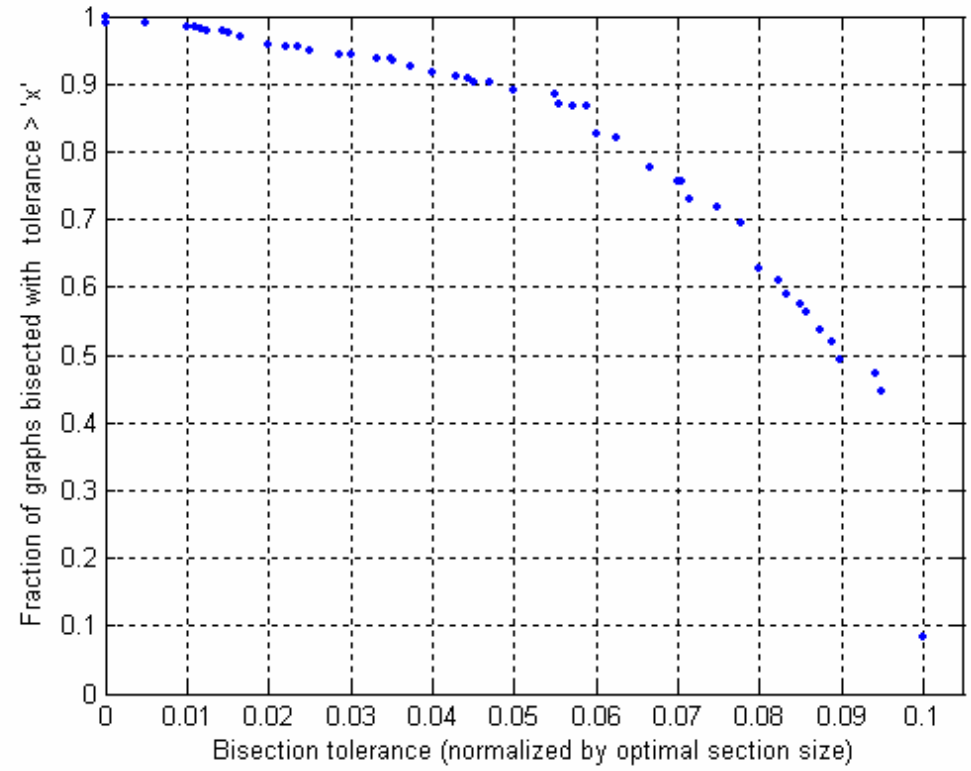

Figure 13. CCDF of partitioned graphs per tolerance for PRAGMA generated set

3..3 Graph sets focusing on graph characteristics and tolerance

We created various sets of graphs, each one focusing on a different characteristic. The sets are defined by the size of their graph, the global weight range [*maxWeight*, *minWeight*) and the number of population centers used. Weights were distributed with the procedure described earlier, allowing a small percentage of nodes to have a *nodeRatio* value over 3. The algorithm was run on each graph for a number of tolerance values.

3..3.1. Graph Size

The first two graph sets (set #1 and #2) included graphs with node sizes from 300 to 600 nodes. In the context of wireless networks, we would like to verify whether the algorithm can be applied in both large scale and smaller scale networks. The global weight range was [10^5, 10^3) and [10^5, 10^4) respectively for each set, and the weights were distributed using 5 population centers (around 0-2% of graph size). The graphs were bisected using tolerance values of 0%, 1%, 2%, 3%, and 4% of the optimal partition size.

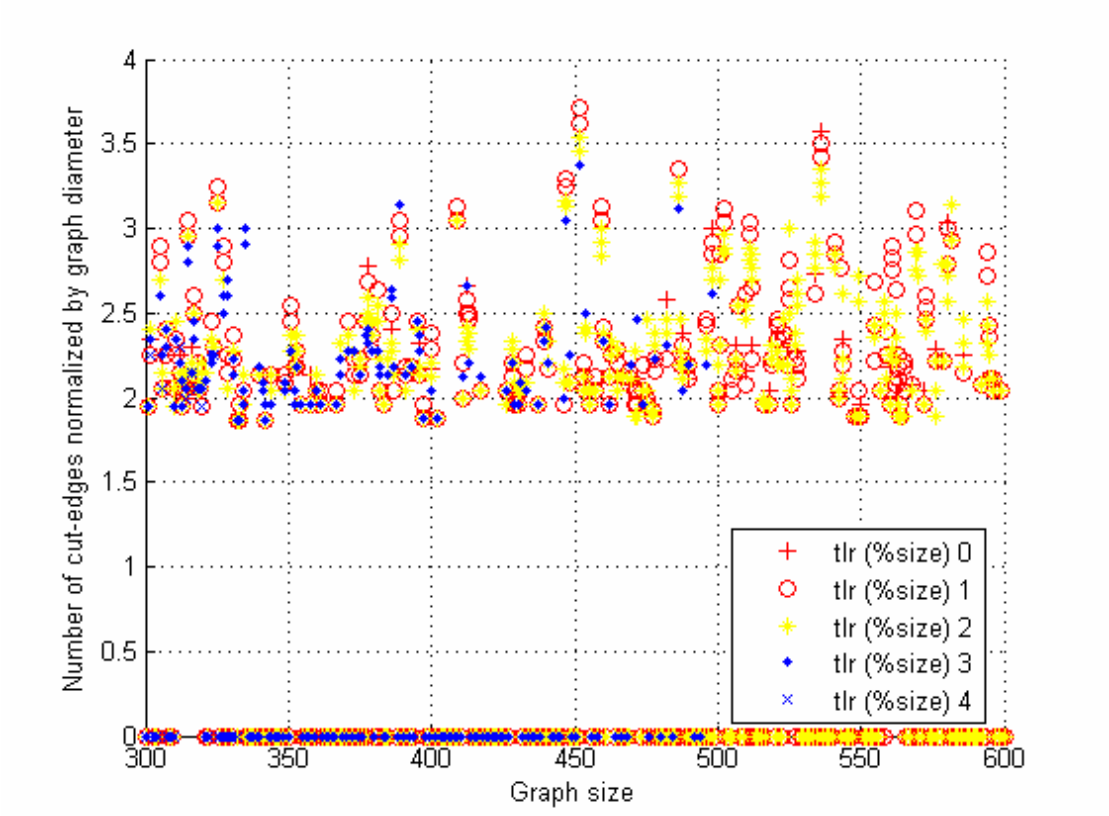

Figure 14. Bisecting path length of set #1

As it was expected the partition cost increased with graph size. In Figures 14 and 15, the partition length normalized by the graph diameter, both in hop counts, is shown for each graph size. Zero values represent unpartitioned cases. It is obvious that the partition length has a lower bound of around twice the graph diameter. This lower bound is not affected by the fact that global weight range is different between the two sets, however, it seems that a wider range

produced longer paths. The value of the lower bound can be explained from the fact that bisecting path length concerns paths in the dual graph and is normalized by the diameter of the original graph. Consider a path along the diameter at the original graph and the corresponding path in the dual graph, whose route passes from face nodes that surround original path nodes. For each edge in the original graph, correspond around two dual edges.

Concerning the wireless network case, we concluded that the algorithm scales well since the increase in size did not affected the fraction of partitioned graphs. However, we got the first indication that networks whose number of transitions between nodes can vary significantly between different areas disfavor DGP, as well as an indication that bisecting path lengths correspond mostly to graph structures that disfavor DGP.

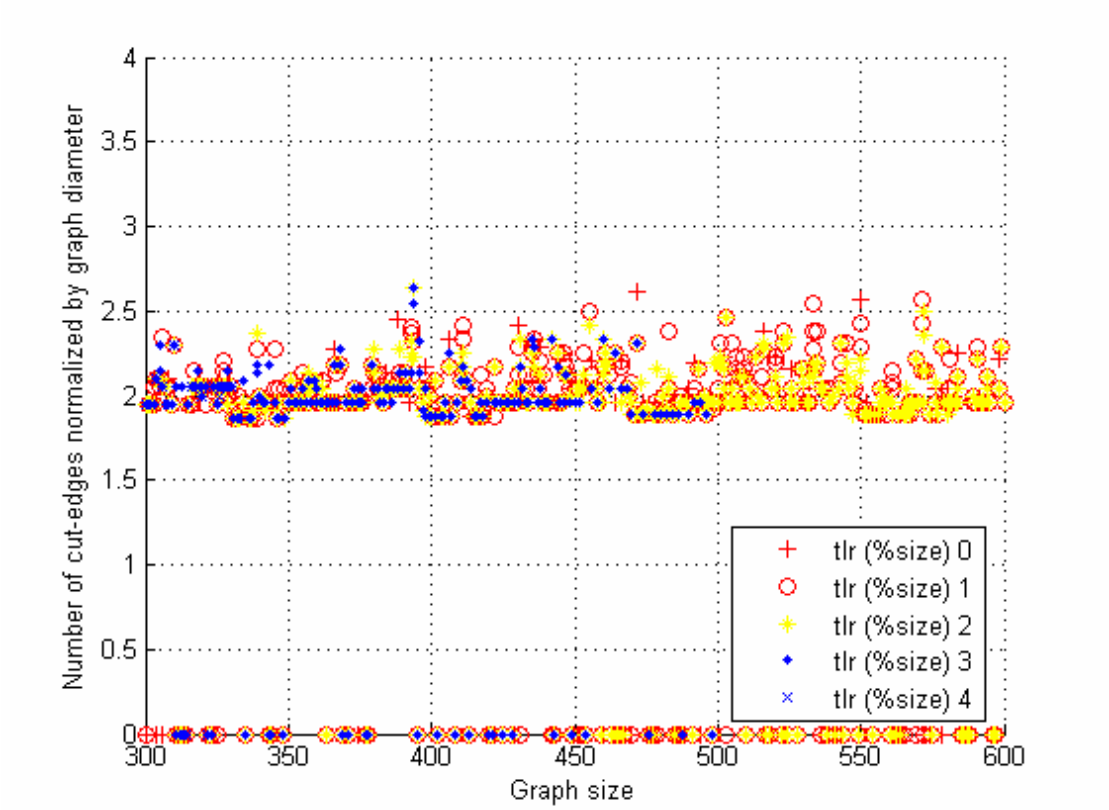

Figure 15. Bisecting path length of set #2

3..3.2. Graph Global Weight Range

The following set of graphs (set #3) was constructed to test the effect of the global weight range on the partition results. In the context of wireless networks, we wanted to verify our previous observations that networks with clients whose mobility differs greatly are less probable to be partitioned by DGP. The constructed graphs consisted of 500 nodes and their global min weight was set from 10^2 up to $9 \cdot 10^4$, with max weight fixed to 10^5 and, for each graph, the partition algorithm was run for tolerance values from 0 up to 6 nodes (approximately 2% of optimal section size). There were used 5 population centers (1% graph size). In Figure 16, we present the results of DGP with respect to the resulting partition number of cut-edges, grouped by tolerance. It was not straightforward to form a hypothesis on whether increasing min weight, and thus tightening the global weight range, would affect the partitioning result.

Results show that as the global edge weight range increases (*maxWeight* - *minWeight*), the resulting partition cost decreases, which was expected since there are more edges with relatively low values and some of them will eventually participate in the bisection cut. Also, higher tolerance values yield partitions with lower cost. This is also apparent from Figure 16, since higher tolerance values yield partitions with less cut-edges.

What is also noteworthy is that graphs with very wide global weight ranges are less likely to be partitioned, and when they are, the partitions have significantly more cut-edges, although the partition cost is lower. This is an indication that weight distributions with very large weight range produce graphs with great unevenness of weights in space. This in turn creates BB shortest paths in the dual graph, which avoid areas of high weights passing through areas with lower weights. This makes these paths long in terms of hops and probably unable to bisect the graph under a desired tolerance, since their route depends on the lower weight distribution in space. An insight in the number of population centers and their position in space will clarify this observation.

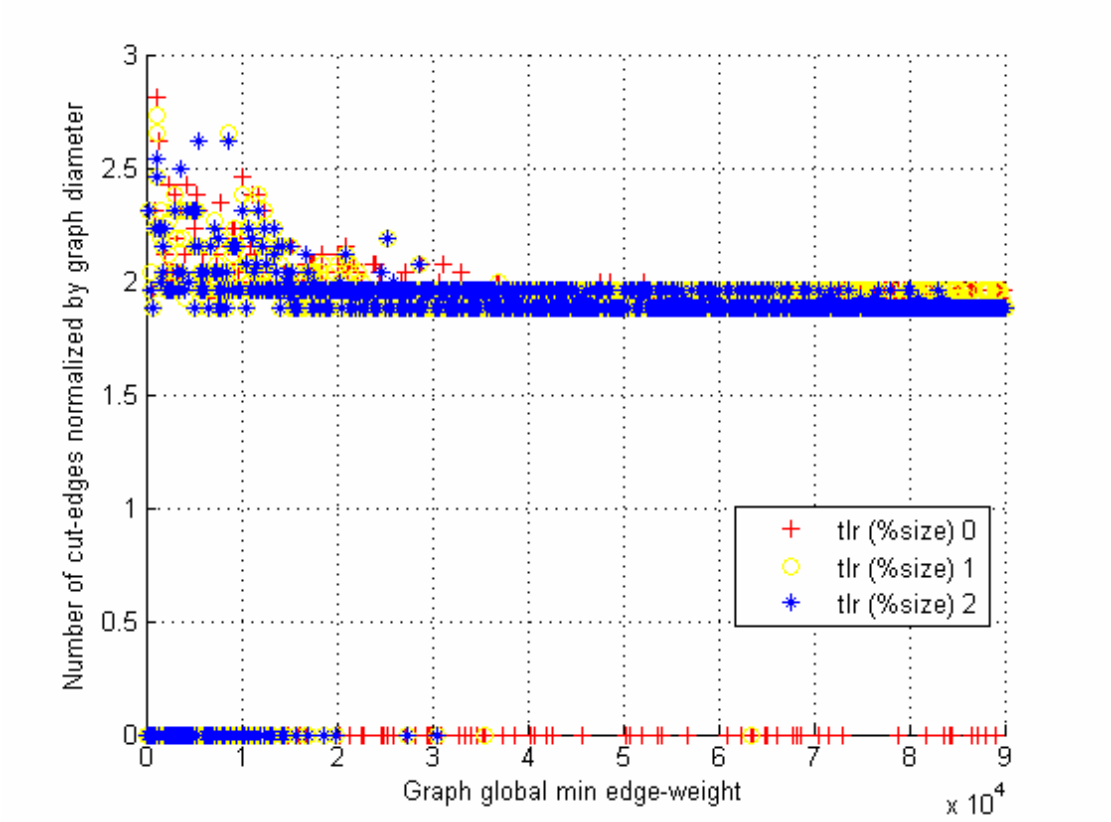

Figure 16. Bisecting path length for each graph of set #3

Our observation can be verified by presenting the number of partitioned and unpartitioned runs at each min weight value, in Figure 17. The results are grouped in bins of min weights [100, 1000], [1100, 2000], etc.

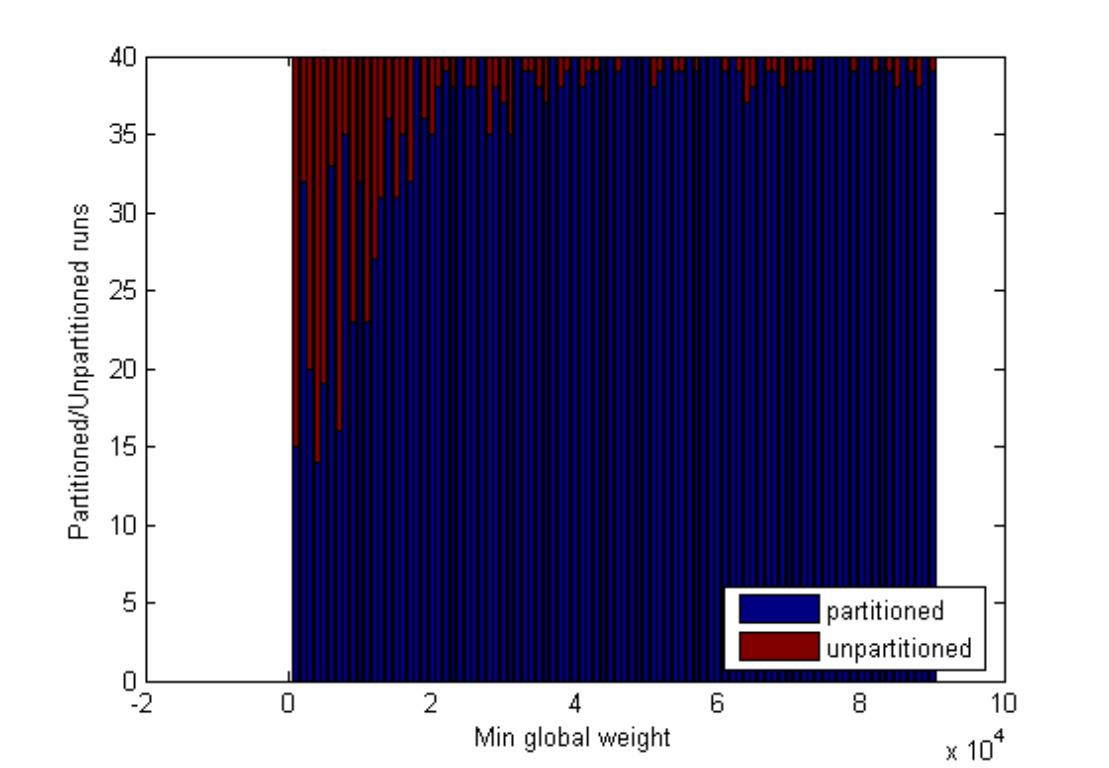

Figure 17. Partinoned and unpartitioned runs for graphs of set #3

Despite some variations between first bins, which may have incurred as an effect of the specific bin aggregation, it is clear that lower min global weight values produce graphs which are harder to bisect, that is, tolerance must be incremented to a certain value, for the algorithm to be able to find a partition. As global min weight increases and therefore, global weight range becomes tighter, the algorithm tends to find a partition for lower tolerance values and subsequent increment of tolerance contributes the corresponding bin with more partitioned runs. This, along with our previous observation, indicates that most graphs which were unbisectable for lower tolerance yielded long bisecting paths when they were indeed bisected with higher tolerance.

Concluding, in the context of wireless networks, we verified our previous observations that networks whose number of client transitions between cells take values from wider ranges are more probable to be unbisected by the algorithm and when they are bisected, the underline graph structure causes the bisecting paths to be long.

3..3.3. Value and Spatial distribution of weights

The purpose of the two following graph sets (set #4.1 and set #4.2) was to investigate how weight value and spatial distribution affects the simulation results. We analyzed this effect implicitly by the number and location of population centers, respectively. Many population centers correspond to many areas with high weights and hence, the weight value distribution will be shifted towards higher values, whereas less population centers correspond to weight value distributions shifted towards lower values. Obviously, the location of population centers reflects the spatial distribution of high weights. Thus, created graphs consisted of 500 nodes, the maximum weight value was 10^5, the minimum 10^3 and 10^4, for the first and second set respectively. The number of

population centers varied from 1% to 10% of graph size. The algorithm was run for each one of these graph instances for tolerance values 0, 1, 2, ..., 6 nodes (up to 2% of optimal partition size).

The number of population centers, in combination with the global weight range is expected to affect the algorithm greatly, since they primarily affect the way weights are distributed in the graph. Considering the edge weight assignment process, in case of many population centers, it is quite probable that *maxDist* will have a low value (edges will not be at great distances from population centers), resulting in relatively few weight bins. Moreover, if values are taken from a wide global weight range, each bin will span a significant range of values. Along with the fact that bins are overlapping, there will be variability in the weight values that can occur at each distance from a population center. On the other hand, few population centers can produce graphs where edges can be many hops away from the nearest population center (greater *maxDist*), with high probability. There will be more weight value bins and thus, there will be a more gradual decrease of weight values, from the first to the last bin, still subject to the fact that they overlap.

Apart from the number of population centers, which gives an indication of the probability to have more or less weight bins, their relative position in the graph affects the resulting bins as well. Highly concentrated population centers can yield high *maxDist* values. We expected that, as population centers define around them neighborhoods of weights greater than other places of the graph, it is possible that certain positioning of them would result in many long BB shortest paths in the dual graph, since their route would go around these highly-weighted neighborhoods. If most of the paths have sub-routes going through the same territory, in a way that will always leave out many nodes in one resulting section, the algorithm will not be able to find a partition with a desired tolerance.

In practice, the above speculations are summarized in the existence of many areas with highly mobile users and their distribution in space, from the wireless network point of view. Many population centers represent many areas of a wireless network with highly mobile users or many areas with a large client population. The distribution of population centers in space represents how the most mobile of wireless clients are distributed in space or how the population of wireless clients is higher and lower at different areas. The latter case assumes that a higher population in a transceiver cell corresponds to more transitions of clients, from and to neighboring transceiver cells.

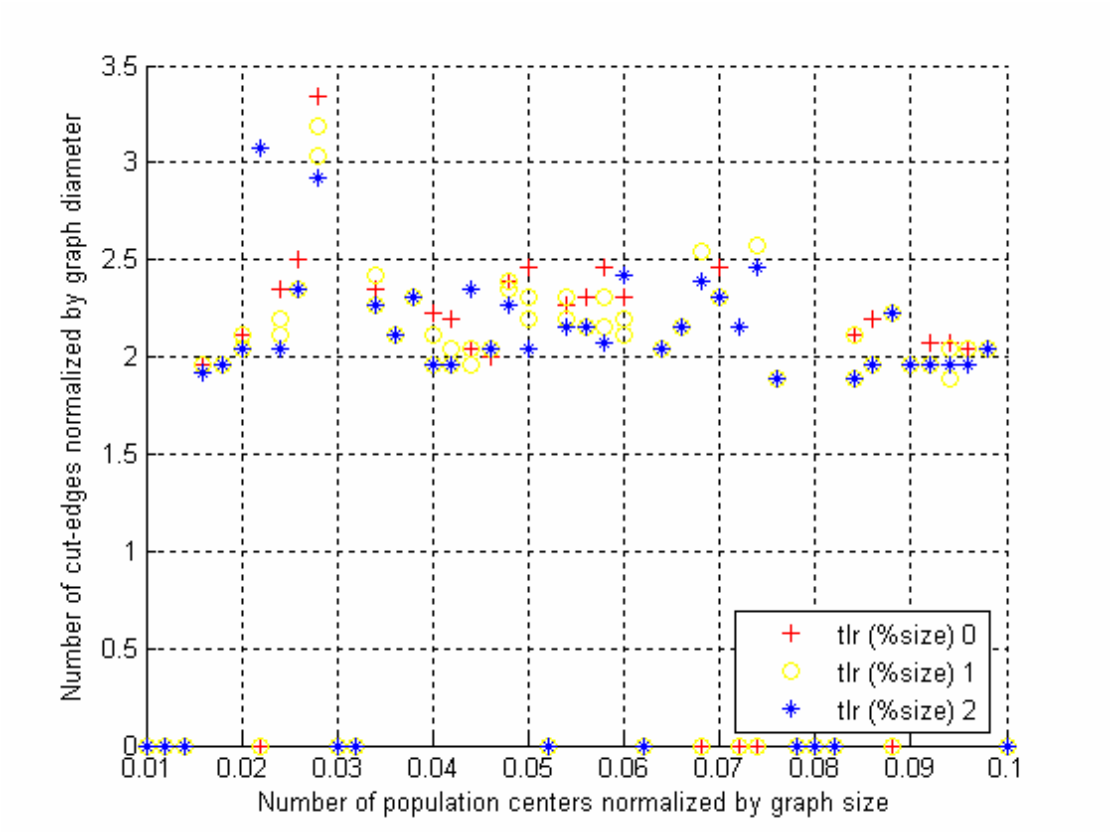

Figure 18. Bisecting path length for each graph of set #4.1

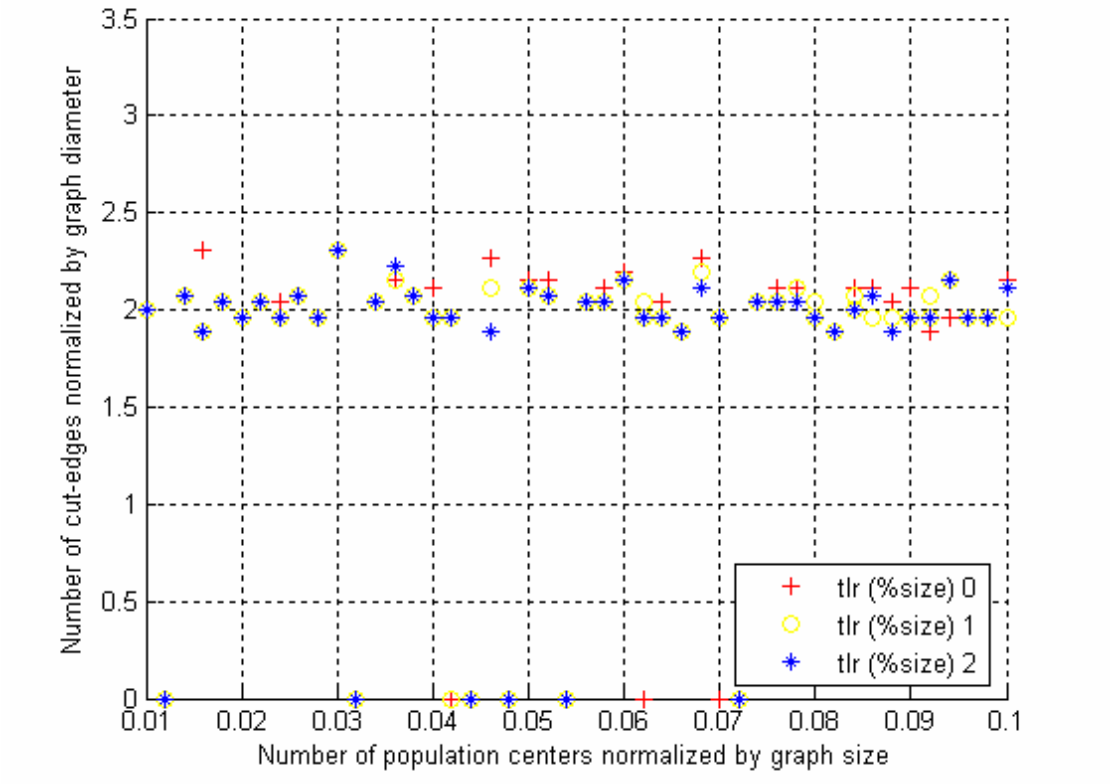

Figure 19. Bisecting path length for each graph of set #4.2

In Figures 18 and 19, the length of the bisecting BB shortest paths, normalized by the graph diameter, is plotted against the number of population centers used for weight distribution, normalized by graph size. Zero length represents unpartitioned graphs. As in previous runs, the BB shortest path length has a lower bound around twice the graph diameter and it's not affected by the difference in number of population centers between graphs. This is an indication that the positioning of population centers, rather than their number, primarily affects the partition result. This is also enhanced by the fact that unpartitioned graphs occurred for various numbers of population centers.

There are also some outlier cases where bisecting path length was much higher. In one of these cases, the algorithm couldn't partition the graph until tolerance was raised to a value of 2% of the optimal partition size. Then a long bisecting path occurred. As it was aforementioned, this was an expected case. Thus, we focus on the spatial weight distribution of this individual case to verify our hypothesis about its cause.

The weight bins and weight values of this graph instance are visually presented in Figure 20. In the first picture, colors represent node distance, and hence bin distance, from the closest population center. Distance is considered to increase from warmer to colder colors, as it was also depicted in Figures 4 to 6. Dark red represents population centers. Their attached edges are assigned values from the first bin. All neighbors are considered one hop away and the rest of their attached edges are assigned values from the 2nd bin, and so on. In the second picture, the resulting weight distribution is presented. The weight range was divided in 12 equal bins, corresponded to a different color. Colors represent weight values, with warmer colors corresponding to higher values and colder colors to lower values. Note that the range of weight assignment bins is not equal for each such bin, whereas colors were corresponded to equally sized bins. Hence, the rate at which weight values decrease as distance from population centers increases, is apparent. The edge cut is represented by the red dual nodes along the bisecting path.

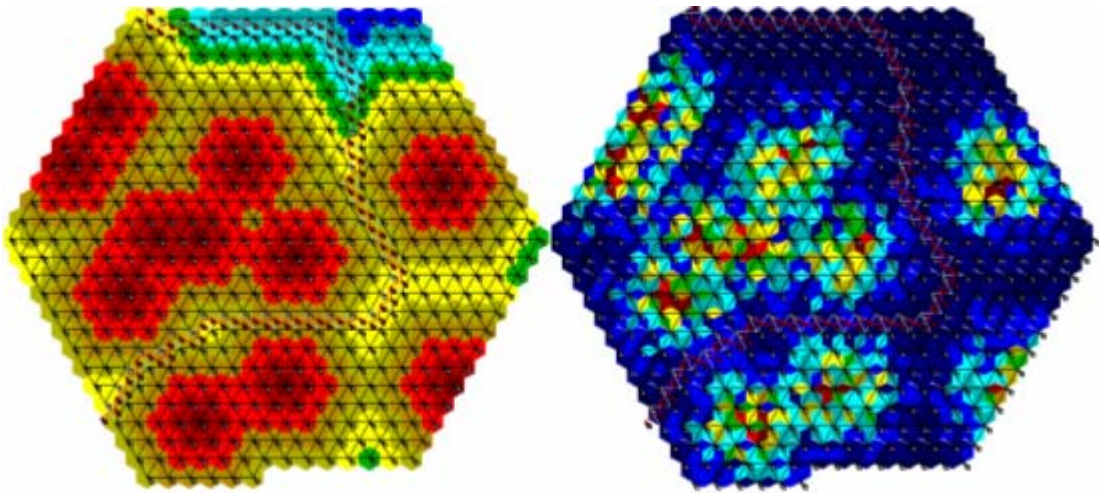

Figure 20. Weight bins and corresponding values, for a graph instance of set #4.1

From the first picture we can see the existence of a cluster of population centers in the north-west section of the graph. From the second plot we can verify the distribution of weights according to this positioning of population centers. The clustering of high weights is such that forces the bisecting path

route to go round this region, resulting in a long path. It is thus, apparent that the position of population centers, which implicitly represents the spatial distribution of high weights, primarily affects whether a graph can be partitioned by the algorithm. Such graph cases are "harder" to bisect, that is, tolerance must be increased for a bisecting path to be acceptable.

In order to establish our previous conclusion, a graph instance was created where population centers were positioned manually around the center of the graph. Maximum weight was 10^5. Minimum weight was set to 10^3 and the weights were distributed with *nodeRatio* 2, 3, 4, 5, and 6. The algorithm did not find an acceptable cut for a tolerance up to 20% of the optimal partition. Even for higher *nodeRatio* values, where lower weights were allowed near higher weights, the concentration of high weights was such that prevented these lower weights to constitute a part of a shortest BB path. Consequently, the graph was bisected for impractical tolerance values of 74.6%, 74%, 74%, 70.6%, and 70.6% of the optimal partition size, respectively for each *nodeRatio*. In Figure 21, an instance of these graphs is presented.

We increased the minimum weight by one order of magnitude, to 10^4, in order to decrease the unevenness between weights. There were no shortest BB paths passing through this central cluster of population centers, even for highest *nodeRatio* values. The algorithm found a bisecting path with tolerance 74%, 76%, 64.6%, 70.6%, and 50.6% of the optimal section size, respectively for each *nodeRatio*. Note that *nodeRatio* = 6 allowed many lower weights to be located between higher weights, which permitted shortest paths to pass through some regions, bisecting the graph with a lower tolerance.

Tightening even more the global weight range by setting the minimum weight to $2 \cdot 10^4$ yielded better results. Lower weights were increased to a level that routes going through the central cluster of population centers yielded shorter paths than routes going around it. The algorithm managed to bisect the graph for a tolerance up to 20% of the optimal partition, for all *nodeRatio* values, but 2. Such a low ratio did not allow significantly lower values to be attached at the same node with higher values and thus, the central region remained a cluster of high weights. As a conclusion, our speculations on the importance of the location of high weights along with global weight range, are validated.

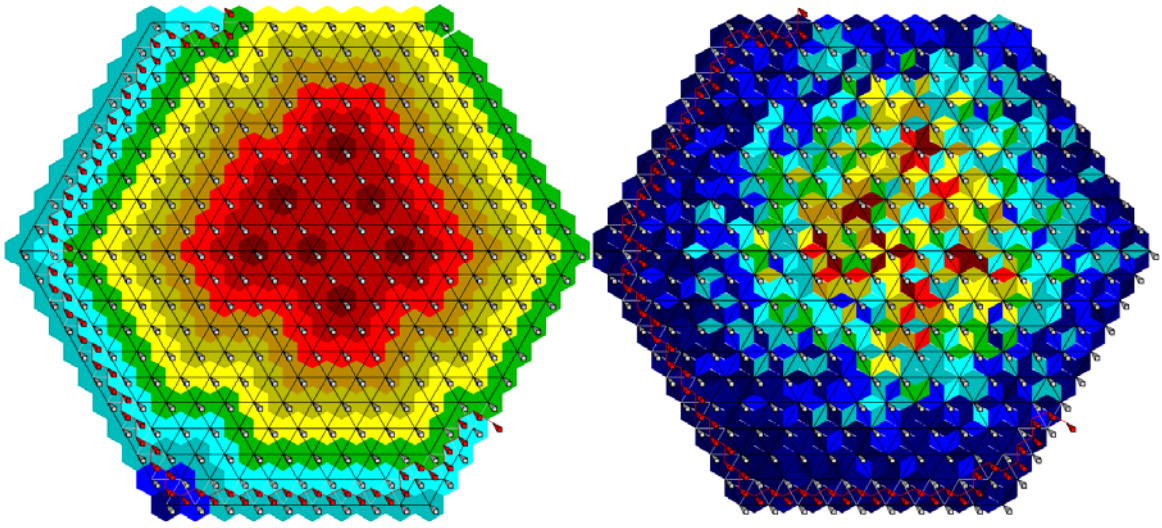

Figure 21. Example of the population center distribution of an unbisectable graph

3..3.4. Tolerance

The last graph sets (set #5 and #6) focused on the effect of tolerance. Since tolerance is an algorithm parameter, set #5 was also used to cross partition results with those of previous sets. Thus, for set #5, graph size was between 325 and 425 (step of 50), *maxWeight* was set to 10^5, *minWeight* varied from 10^3 to 10^4 (step of 10^3), and the number of population centers varied from 2% to 7% of graph size.

We bisected the graphs for tolerance values up to 60% of optimal section size and we tested the effect on the partition cost, the bisecting path length, and the fraction of partitioned graphs for each tolerance. We grouped results by *minWeight* and number of population centers in order to speculate their effect and cross-check the corresponding conclusions from previous sets. Extended results for all such parameters are presented in Appendix B, in Figures 39 to 45.

Results validated our conclusion on the effect of global weight range since graphs with lower *minWeight* were less frequently bisected for low tolerance values. Moreover, our conclusion that bisecting path length (in hops) is an indication of how "hard" is for DGP to bisect a graph was enhanced, since the majority of graphs which were bisected with very long paths had lower number of population centers and in the same time, the graphs with lower number of population centers were the ones less frequently bisected. Interestingly enough, these graphs also had the lowest of *minWeight* values, which implies that a combination of these cases affects DGP performance.

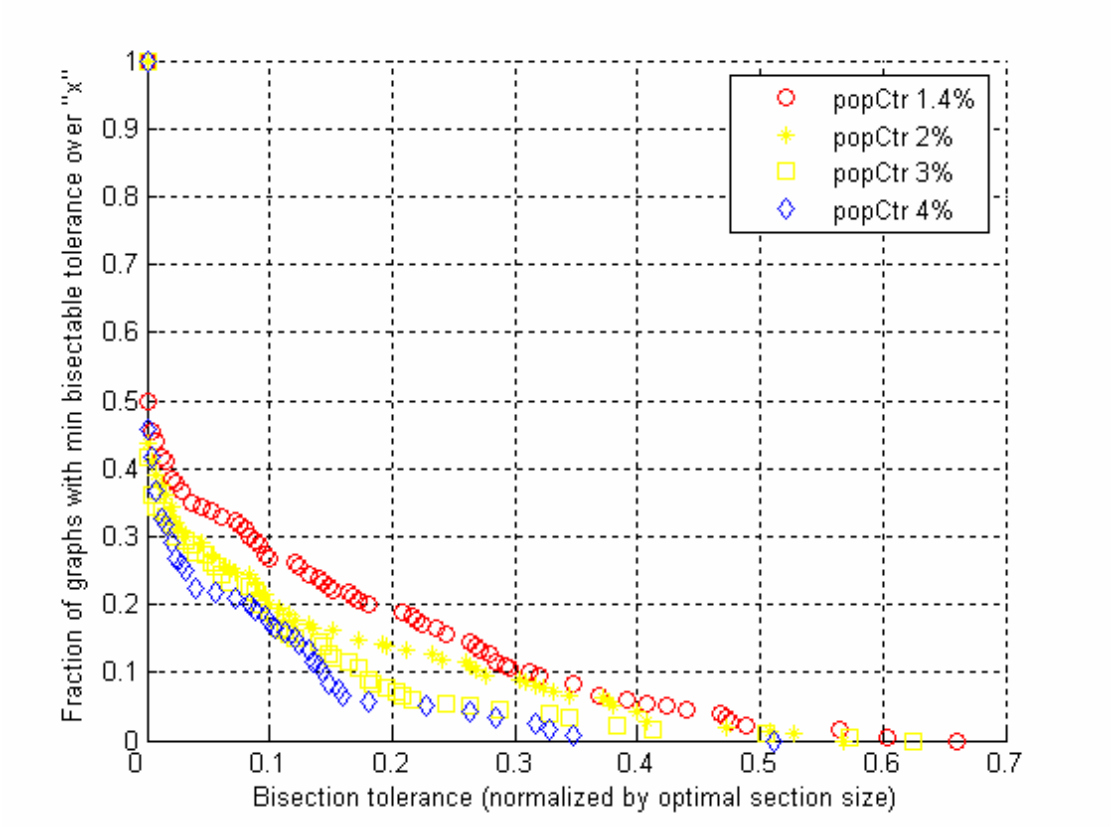

Figure 22. CCDF of min bisectable tolerance for set #6

Graph set #6 was tested solely in terms of partition succession. The graphs consisted of 500 nodes, *maxWeight* was set to 10^5 and *minWeight* was set to 10^3, in order to create less favorable instances. The number of population centers was

set to 7, 10, 15 and 20 (1.4, 2, 3, and 4% of the graph's size). The minimum tolerance at which a graph was successfully bisected was found for each graph, by using different tolerance values in a binary search-wise order. The CCDF of the minimum tolerance at which a graph was successfully bisected versus the fraction of graphs bisected with this minimum tolerance, is presented in Figure 22. The stochastic order between graphs with different number of population centers is apparent. In the worst case (fewer population centers), the algorithm finds a partition for all graphs, for a tolerance around 64% of the optimal partition section and around 34% in the best case (most population centers). Considering 5% of the highest tolerance values as outliers, these percentages decrease to 40% and 20% respectively.

3..3.5. Inferring Weight Spatial Distribution

3.3.5.1. Detection of Population Centers

Due to its primary importance, we focused on the spatial distribution of weights at generated graphs. We searched for a simple way to quantify this spatial distribution. In the generated graphs it is modeled implicitly by population centers, since their location defined the essence of this distribution. However, the locations of population centers are already known for these graphs and it would be inaccurate to take this information for granted. In order to generalize our study, we constructed a simple heuristic with which population centers are defined and located in a graph. In a wireless network, this corresponds to a detection of endpoints, within their coverage are located many mobile clients, assuming that the more clients there are, the higher the probability for a transition to occur from and to neighboring endpoints.

The heuristic is based on the fact that “higher” weights are considered to be attached to population center nodes. The boundary that makes weights “high” must be found. What is most certain is that the highest weight edge will definitely be attached to a population center. Its attached nodes are located and their distance, in hops, to the most distant border node is computed. The average of their distances is considered as a candidate *maxDist*.

The graph's maximum and minimum weight are corresponded to *maxWeight* and *minWeight*. As it was described in the weight assignment procedure in section 3..1.2, these parameters allow us to compute a decreasing factor *b* and define the bounds of a number of '*maxDist*' weight value bins. Nevertheless, we only need to define the first bin from which, edges attached to population centers are assigned weight values. The candidate *maxDist* will be used to define the bound of the first bin. Thus, every edge whose weight is the interval $[maxWeight,\ maxWeight\ /(maxWeight/minWeight)^{1/maxDist}]$, let it be called “heavy edge”, is considered to be attached to a population center and a set of candidate population centers is defined.

Note that each such edge is attached to two candidate nodes and each of them might also share another such edge with another node and so on. However, only a subset of them should be considered as population centers and the rest as neighbors of population centers. All candidate population center nodes are stored in a priority queue based on the number of heavy edges attached to them. The node with the highest priority is removed and considered as a population center. Its attached edges are removed from the corresponding count of the rest candidate population centers that shared them and their priority is re-evaluated.

Iteratively, nodes with the heaviest edges are removed first and considered as population centers, since they're obviously the most probable to be ones. During this process, the count of attached heavy edges might decrease to zero, for some nodes in the priority queue, which means that all their heavy edges are considered attached to already defined population centers. These nodes are not defined as population centers, when they're removed, last, from the priority queue.

It is not guaranteed that the minimal subset of population centers is defined, since the choice between two equally sized candidates affects the heavy edge count of their neighboring candidates, if any, and thus, all subsequent removals from the priority queue. However, a representative set of population centers has been located. From that point, the actual *maxDist* can be computed, if needed.

3.3.5.6.1. The *maxDist* parameter

Our first approach to infer the spatial distribution of a graph in a simple way concerns studying the *maxDist* value. *maxDist* expresses the furthest, in hops, a node can be located in the graph from its nearest population center. For a reasonable number of population centers this is an indication of how they re distributed in space. A high *maxDist* value implies that some population centers are located close one to each other, forming a cluster of relatively high weights, and as a consequence there is another cluster of relatively low weights where population centers are not located. Such topological formations primarily affect the algorithm since they affect the routes of the shortest paths between all border nodes in the dual graph. The formation of these routes is what finally makes them useful or not, since if they partition the graph in two sections which comply with a given tolerance, each such route represents a desired edge-cut.

From a wireless network point of view, analyzing the effect of *maxDist* corresponds to testing whether the distance between areas with most mobile clients and those with less mobile clients, or the distance between areas with most clients and those with less clients, affects DGP. The latter case, assumes that in areas with more wireless clients, the probability of a client transition between neighboring transceivers is higher.

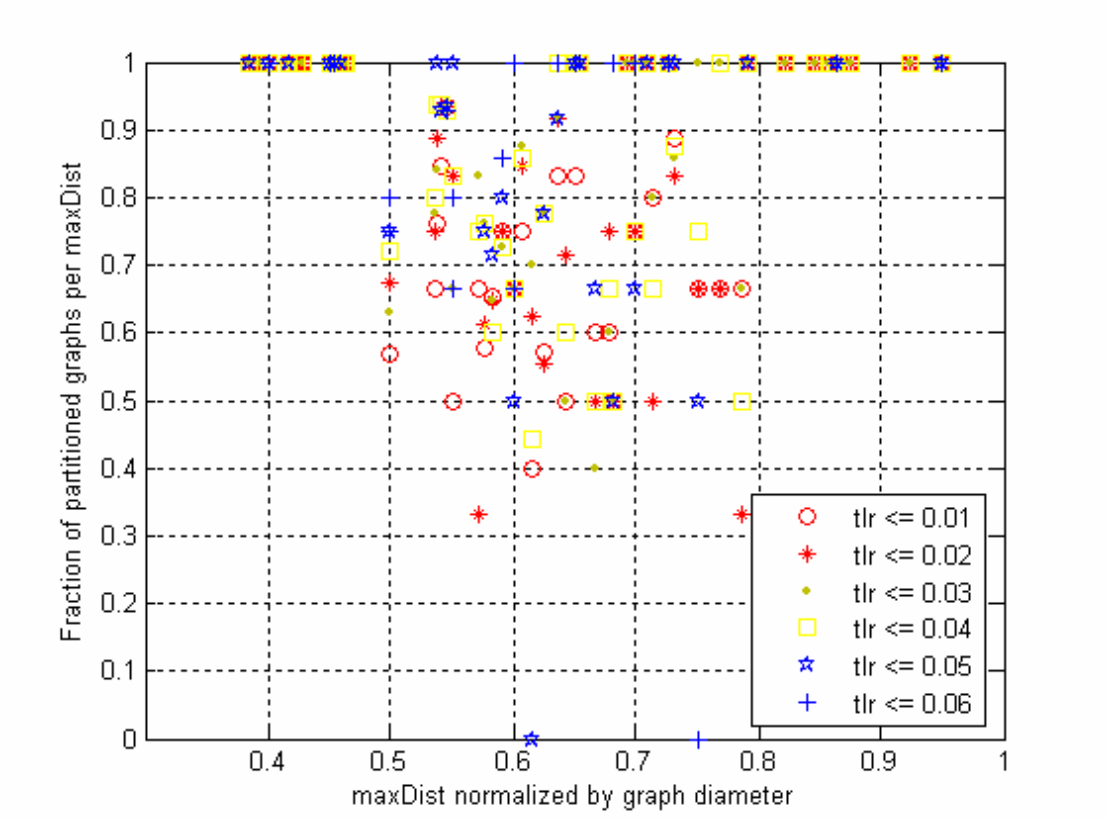

Figure 23. Partitioned graphs per *maxDist*, grouped by tolerane bins of 0.01 for set #2

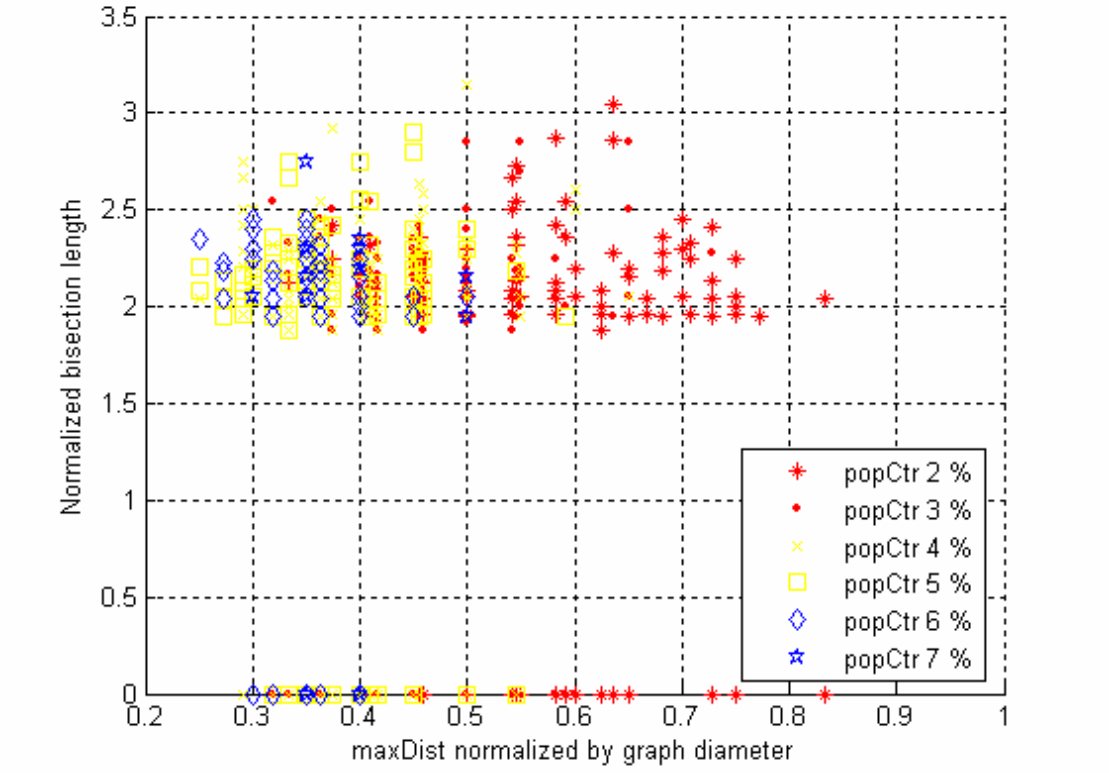

Figure 24. Bisection length per *maxDist*, grouped by number of population centers for set #5

Studying the bisection results in relation with *maxDist* revealed that graphs with extreme maxDist values, i.e., either very low or very high *maxDist*, are

more probable to be bisected by DGP. A representative result is shown for data from set #2, in Figure 23. This is also supported by results concerning bisecting path lengths, where extreme *maxDist* values corresponded mostly to lower path lengths. This is clearly depicted in Figure 24, for data of set #5, where we can also infer the relation of *maxDist* with the number of population centers. Very few population centers let a big area to be valued with lower weights, resulting in short bisecting paths. Too many population centers saturate graph space spread high and low weights in a nearly uniform manner; hence bisecting paths find a short way through the graph. On the other hand, intermediate *maxDist* values correspond to a fair number of population centers, which either form a cluster that prevents a successful bisection, or they can be spread at the perimeter of the graph allowing paths to route through the middle and bisect the graph. A detailed presentation of the most characteristic results from all graph sets is presented in Appendix C.

The *maxDist* parameter cannot be evaluated as an adequate measure of the spatial distribution of weights. Although extreme *maxDist* values clearly correspond to spatial distributions that favor the algorithm, intermediate values can correspond to variable cases which either favor or disfavor BB partitioning. However, this analysis helped us locate the most essential element that makes such distributions easier or harder to bisect. The existence of higher weights in the middle of the graph prevent shortest paths to route through, or differently, the existence of lower weights in the middle of the graph allow BB shortest paths to route through, successfully bisecting the graph. This attribute is analyzed in the following section

3.3.5.6.2. Population center to Border node Distance

In order to infer the aforementioned attribute, the distance between every population center and its closest border node was calculated for graph sets #1, #2, #3, #4.1, #4.2, #5 and the PRAGMA generated graphs. We expected that graphs having most of their population centers near border nodes would be more frequently partitioned. If most of the population centers are far from the border, this consequently means that they re in the middle of the graph, thus disfavoring a successful partition. In the context of wireless networks, we try to validate whether, high inter-cell client mobility clustered and located in the middle core of the graph corresponds to most unsuccessful runs of DGP. Assuming once more, that areas with more wireless clients exhibit more inter-cell transitions of clients, the above case also corresponds to testing whether networks with most wireless clients aggregated in their core disfavor DGP. Nevertheless, the above characteristic is always subject to the embedding of the graph, which DGP is applied to.

In order to test our hypothesis, we plot all population center-border node shortest distances, as well as the average of these distances of each graph, separately for bisected and unbisected graphs, with a tolerance of 10% of the optimal partition size. The distance between a population center and a border node, let it be called D_{bp}, is normalized by graph radius. Thus, the maximum distance can be at most 1 and the minimum 0. Distances are aggregated in bins of 0.1. The results for sets #1 - #5 are shown in Figures 25 to 28.

The results validate our speculations. From the plots of all D_{bp} values, we can observe that bisected graphs exhibit more population centers near the border of the graph in contrast with unbisected ones, which exhibit many more, higher D_{bp}

values. Our visual observation was verified statistically since the Kolmogorov-Smirnov test rejected the null hypothesis that the two sampled distributions are drawn from the same distribution at the 5% significance level, and the Wilcoxon ranksum test (also called Mann-Whitney U test) rejected the hypothesis that the sample distributions come from distributions with equal medians, again at the 5% significance level.

This inference is enhanced by average D_{bp} plots, where it is apparent that bisected graphs exhibit a Normal-like distribution of average distances, whose center is around 0.35 of the graph radius, whereas unbisected graphs exhibit a similar distribution of weights, yet centered at a higher average distance value, around 0.5 of graph's radius.

In order to verify statistically this observation, we performed a two sample T-test. The size of these two populations was large, thus, we could relax the normality constraint. We performed a two-sided test of the null hypothesis h_0 that the distributions of these two populations (bisected and unbisected data) have the same mean ($\mu_b = \mu_u$) over a specific alternative hypothesis that the distribution of bisected data have a different mean ($\mu_b \neq \mu_u$). Furthermore, we performed two one-sided tests of h_0, over the alternative hypothesis ($\mu_b < \mu_u$) as well as the alternative hypothesis ($\mu_b > \mu_u$). Our inferences were verified at the 5% significance level, since h_0 was rejected over the alternative hypotheses that ($\mu_b \neq \mu_u$) and ($\mu_b < \mu_u$), whereas it could not be rejected over the alternative hypothesis that unbisected data come from a distribution with a lower mean.

It is noteworthy that all graph sets exhibited a similar behavior, with the exception of mean distances of PRAGMA generated graphs. In order to explain this case correctly, we must take into account that the way PRAGMA was simulated affected population center locations as to be frequently close to the border. As it was noted in the PRAGMA simulation issues, graphs were relatively small and simulation space was extended outside the border of the graph, at a distance of about a graph hop. Nodes at the border tended to be accounted for more individual transitions from one node to another, because there was no neighbor at the exterior side. Taking all this under consideration, we can see in Figures 29 and 30, that bisected graphs exhibit more nodes at distances between 0-0.2 and 0.2-0.4 of their radius, whereas unbisected graphs exhibit more nodes at distances 0.4-0.6, 0.6-0.8, 0.8-1.0 times their radius. This in accordance with our previous observations.

We can conclude that D_{bp} distribution is an adequate measure of the spatial distribution of weights. It thus can be used to infer the likelihood of a graph or a set of graphs, to be partitioned by DGP.

From the wireless network perspective, results showed that the algorithm is more likely to find a result in networks were most roaming (or handover) operations are not clustered or they occur near the perimeter of the graph. Of course, their effect on the algorithm is always subject to the network graph's embedding. A network graph whose most mobile activity is performed in its core can have other embeddings which relocates this effect near its outer border.

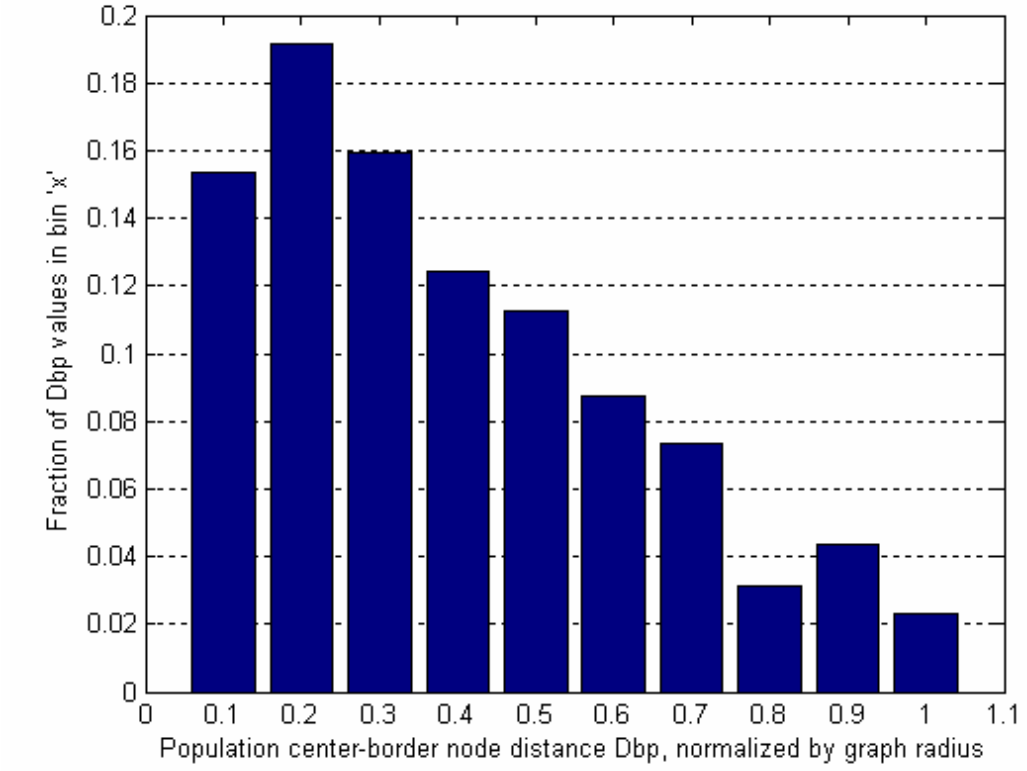

Figure 25. D_{bp} values of each bisected graph from sets #1-#5, grouped in bins of 0.1

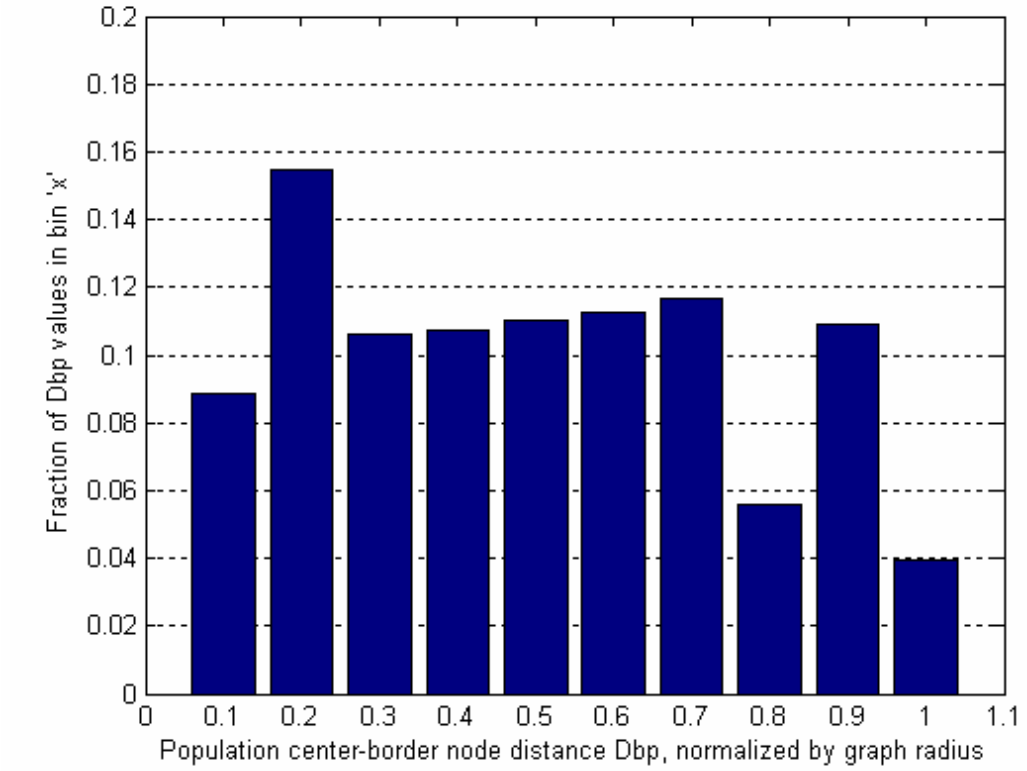

Figure 26. D_{bp} values of each unbisected graph from sets #1-#5, grouped in bins of 0.1

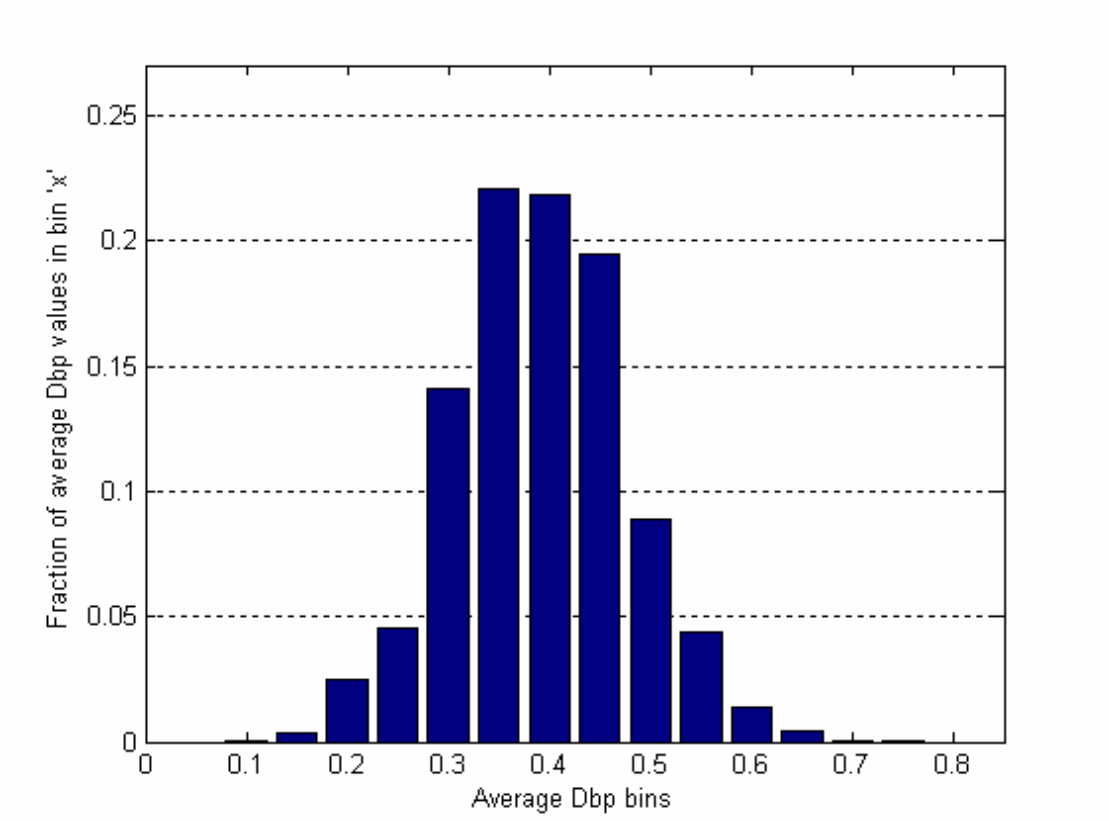

Figure 27. Average D_{bp} of each bisected graph from sets #1-#5, grouped in bins of 0.1

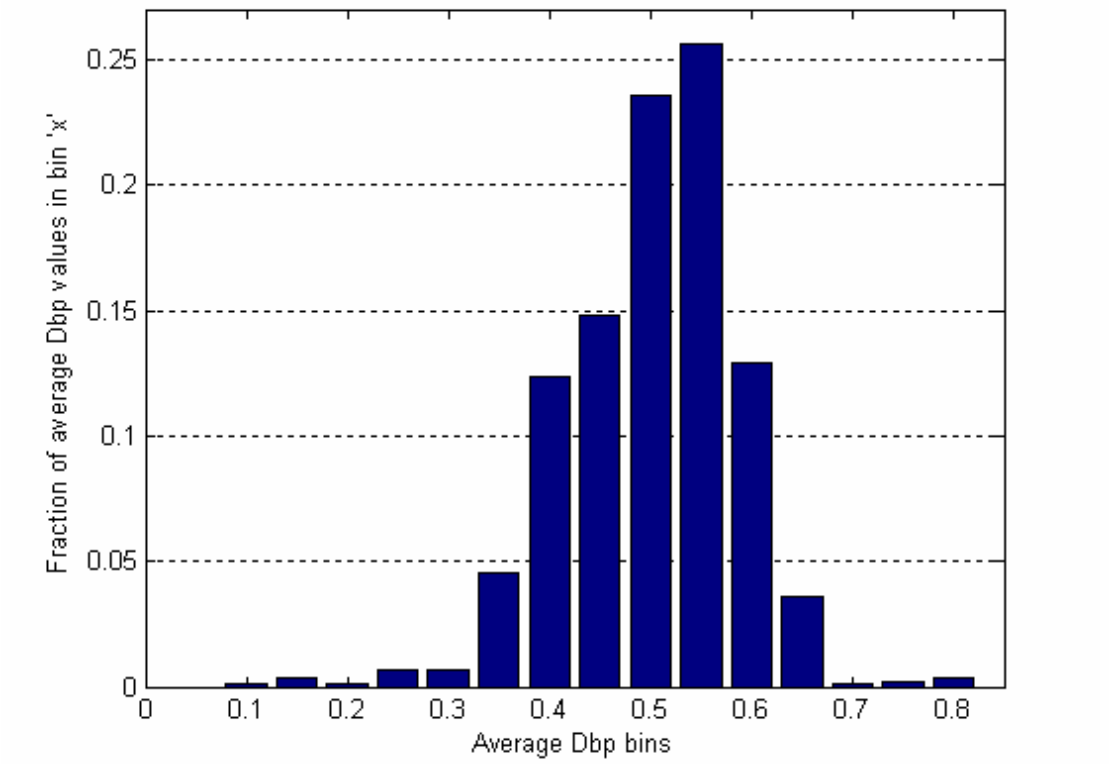

Figure 28. Average D_{bp} of each unbisected graph from sets #1-#5, grouped in bins of 0.1

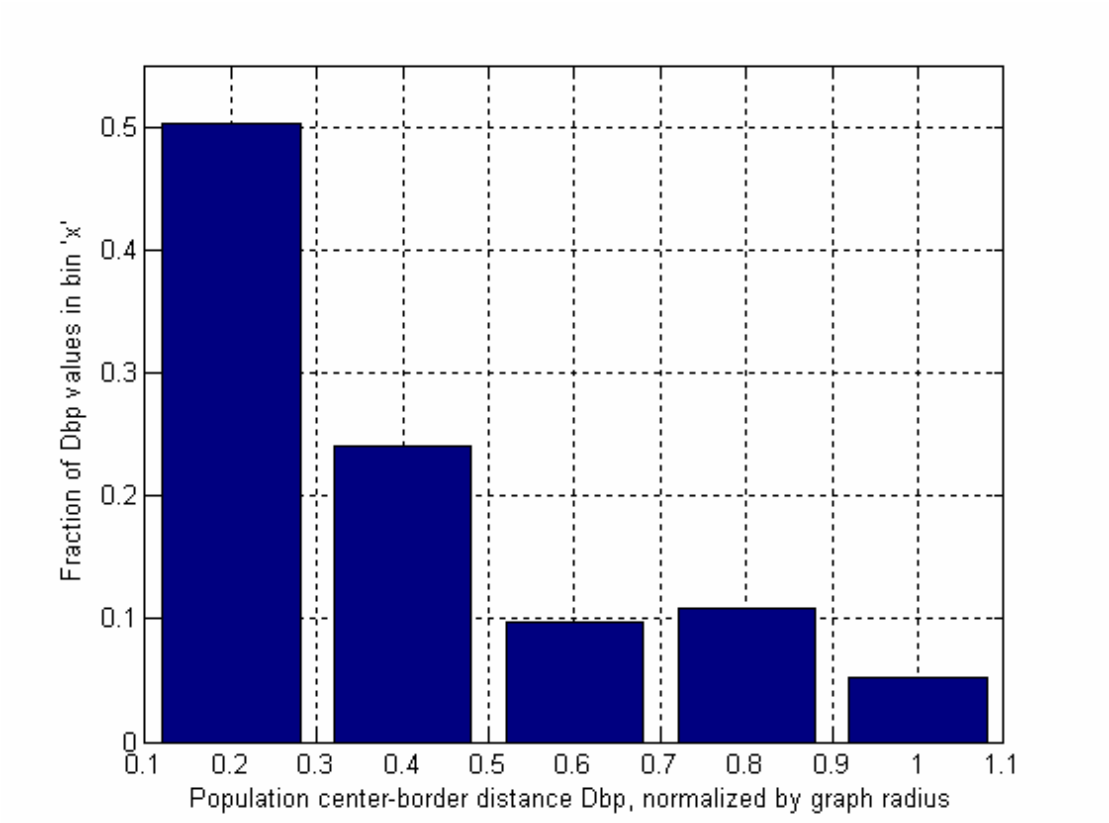

Figure 29. D_{bp} values of each bisected PRAGMA generated graph, grouped in bins of 0.2

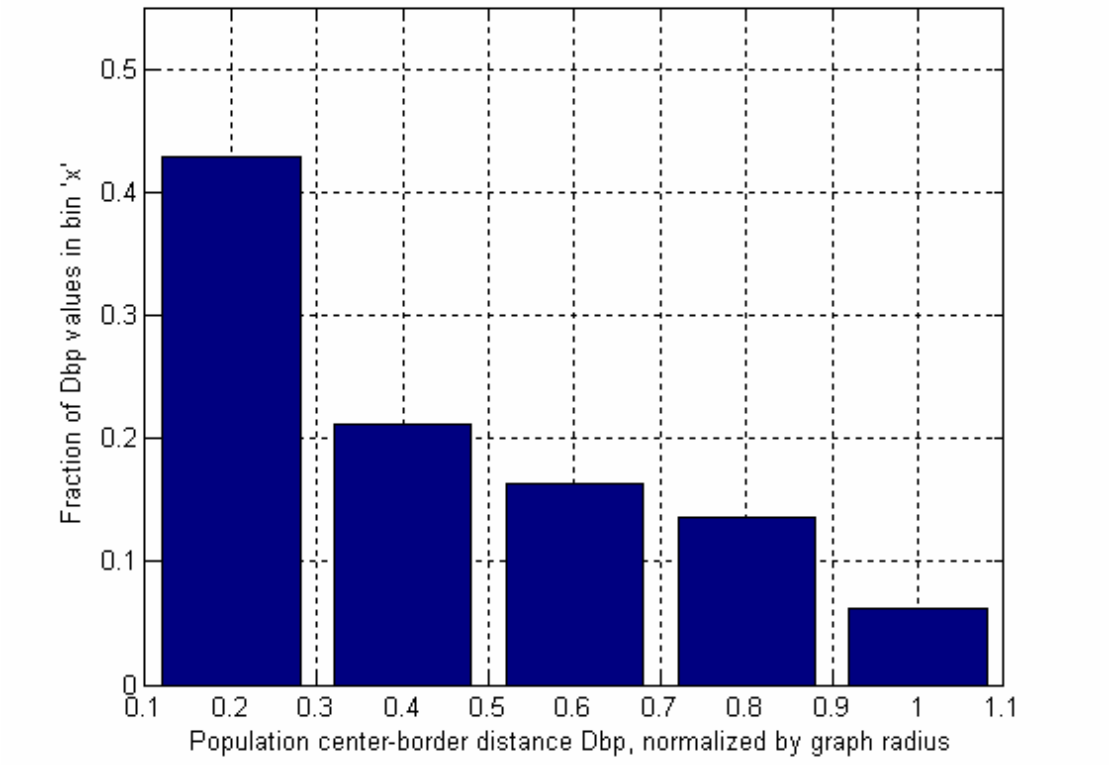

Figure 30. Average D_{bp} values of each unbisected PRAGMA generated graph, grouped in bins of 0.2

3..3.6. Random Graphs

Sets of random graphs were created as an opposite example of the characteristics that were analyzed. Random graphs exhibit no particular structure in how weights are distributed in space. Therefore, this case corresponds to a wireless network example where wireless clients can be arbitrarily mobile from place to place. Each edge is assigned a weight value randomly from a global range [*minWeight*, *maxWeight*). The Mersenne Twister Random generator was used for weight assignment. It has been proven that it produces graph topologies closer to the theoretic random graph than other random functions [59].

The first random graph set consisted of 1000 graphs. Size was set to 500 nodes and the weight range was $[10^3, 10^5)$. The bisection tolerance was set to 5% of the optimal partition size. In Figure 31, the tolerance at which the best partition was found is plotted against the fraction of graphs to which it corresponded. More graphs were bisected with higher tolerance values, yet there was not a big difference between higher and lower tolerance fractions. Two graphs were successfully bisected after increasing tolerance to 13 nodes (since 5% of 250 is 12.5) and only two out of 1000 graphs were not bisected (0.2%). This is a negligible percentage in comparison with those of previous sets.

Two more random graph sets were created with edge weights assigned from a tighter value range. The graph size was 300 and 500 respectively for each set. *minWeight* was set to 10^3 and *maxWeight* varied from $3 \cdot 10^3$ to 10^4. Only one graph was not bisected from each set and corresponded to the widest of weight value ranges used. Overall, this number represented the 0.3% of graphs from both sets.

Results show that random graphs are “easier” to partition. This is also indicated by resulting path lengths which are all very close to twice the graph diameter. Note that this was the lower bound for previous sets and that path length indicates how "hard" is for the algorithm to bisect a graph. Moreover, weight range doesn’t seem to be as important, concerning the performance of the algorithm, due to the lack of clustering of weights. Yet, it is evident that in the case of a random local clustering of weights, a wider range is more probable to cause a problem to the algorithm. We can also conclude that graphs degenerate to random ones for an excessive number of population centers and a *nodeRatio* value equal to *maxWeight*/*minWeight*.

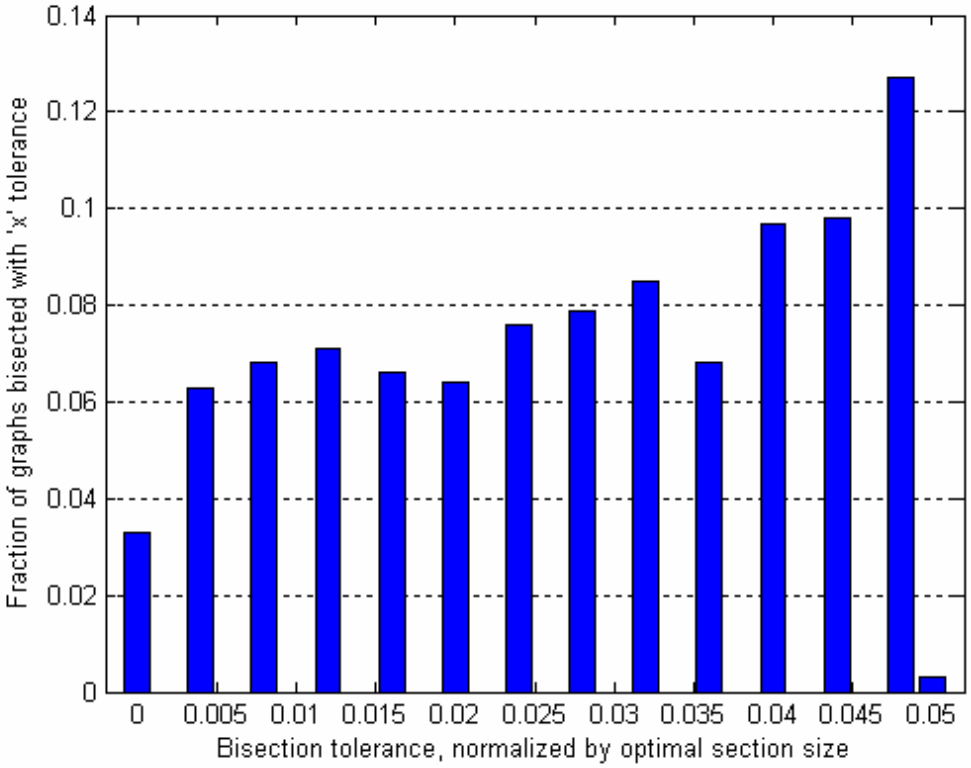

Figure 31. Partitioned graphs per tolerance, of random graph set

3.3..7 Variation of DGP

The variation of DGP was tested as a complement that guarantees a partition result in a regular grid graph. As it was described in section 2.4, instead of creating bisections using border to border node shortest paths in the dual graph (let this be called border to border (BB) partitioning), we create a bisection using pairs of shortest paths that begin from an interior dual node and end at an outer border node (let this be called path pair (PP) partitioning). The intermediate region consists of subsequent neighboring nodes along two vertical lines.

In section 2.4.2, we stated that these lines must cross in the center of the graph as to ensure that all BB results would be checked. A small variation of this procedure was used to ensure that a result would be found. If the lines defining the intermediate region nodes were to cross in the center of the graph, there could be a certain population center positioning that would prevent any pair of paths to partition the graph with a desired tolerance. Thus, it is desired that the crossing point of these lines should be as closer to the center of a cluster of high weights as possible, forcing routes to cut through an area where original BB shortest paths would not. Thus, the center of mass of weights is computed and the first line of the intermediate region is defined as to start from a random border node and pass from this point. Nodes along this line are set as intermediate ones. Then the second line is defined as to pass from the center of mass and be vertical to the first. Then, intermediate nodes are set, starting from a border node closer to this line.

We performed a PP bisection for graph sets #1, #2, #3, #4.1, and #4.2. We studied whether the cost was lower or equal than the one previously found, the ratio between PP and BB cost and whether the graph was bisected in cases that BB shortest paths did not yield a bisection.

As it was expected, the PP technique found a result for all graphs that were not bisected with the original version of the algorithm. It also improved the result for some of the graphs for which there existed a BB partition. In Table 1, the percentage of partitioned graphs, for which the PP cost was lower than the BB cost, is presented for each set. The variation of the algorithm yielded a lower partition cost for 38% of graphs, in average.

The comparison between PP and BB tolerance at which graphs were eventually bisected is presented in Table 2. For each set, the first column denotes the percentage of graphs for which the PP tolerance was lower, equal, or greater than the BB tolerance, respectively for each row. For each such value, the percentage of graphs that yielded a lower cost is presented at the second column. For example, a value of 100% in the last row indicates that all graphs of the corresponding set, whose PP partition tolerance was higher than the BB tolerance, yielded a lower cost.

We can observe that the variation of the algorithm found lower cost partitions, even with lower tolerance, with respect to BB partitions. For an average of 10% of graphs the PP partition was more balanced than the BB partition. For all such cases the PP cost was lower than BB cost. The PP partitioning tolerance was greater for 21% of graphs, in average, which apparently improved the partition cost. Partition tolerance was the same between the two techniques, for the rest of graphs, however, for an average of 11% of them, the PP partition cost was still lower, which means that they corresponded to different partitions than those found by BB partitioning.

Graph set	Set #1	Set #2	Set #3	Set #4.1	Set #4.2
% PP < BB cost	39	50	46	33	24

Table 1. Percentage of graphs whose PP cost was lower than BB cost

Graph set	Set #1		Set #2		Set #3		Set #4.1		Set #4.2	
% PP < BB tlr	13	100	13	100	8	100	9	100	7	100
% PP = BB tlr	73	17	57	12	60	9	77	12	80	6
% PP > BB tlr	14	100	30	100	32	100	14	100	13	100

Table 2. Comparison of PP and BB bisection tolerance and yielded cost

In Figures 32 to 36, the CDF plot of PP to BB partition cost ratio is presented for each graph set. In set #1, it is noteworthy that there were cases, where the partition cost was reduced to 40% of the bb cost. 10% of graphs were bisected with cost reduced to a value at least 25% lower. In set #2, for 5% of the bisected graphs the cost was reduced by 13%-27%. For another 5% of bisected graphs the cost was reduced by a value between 8%-13%. In set #3, we can also see some cases were the cost was greatly improved (more than 40%). However, about 97% of costs were reduced by up to 10% and only a percentage of about 3% of costs was reduced by 10%-45%. In set #4.1, the improvement of cost was

between 8-24%, for 10% of the graphs and in set #4, 10% of costs were reduced by a value between 8%-12%.

We can conclude that the variation of the algorithm can guarantee that a partition will be found and that its cost will be less or equal to that of any existing BB shortest path.

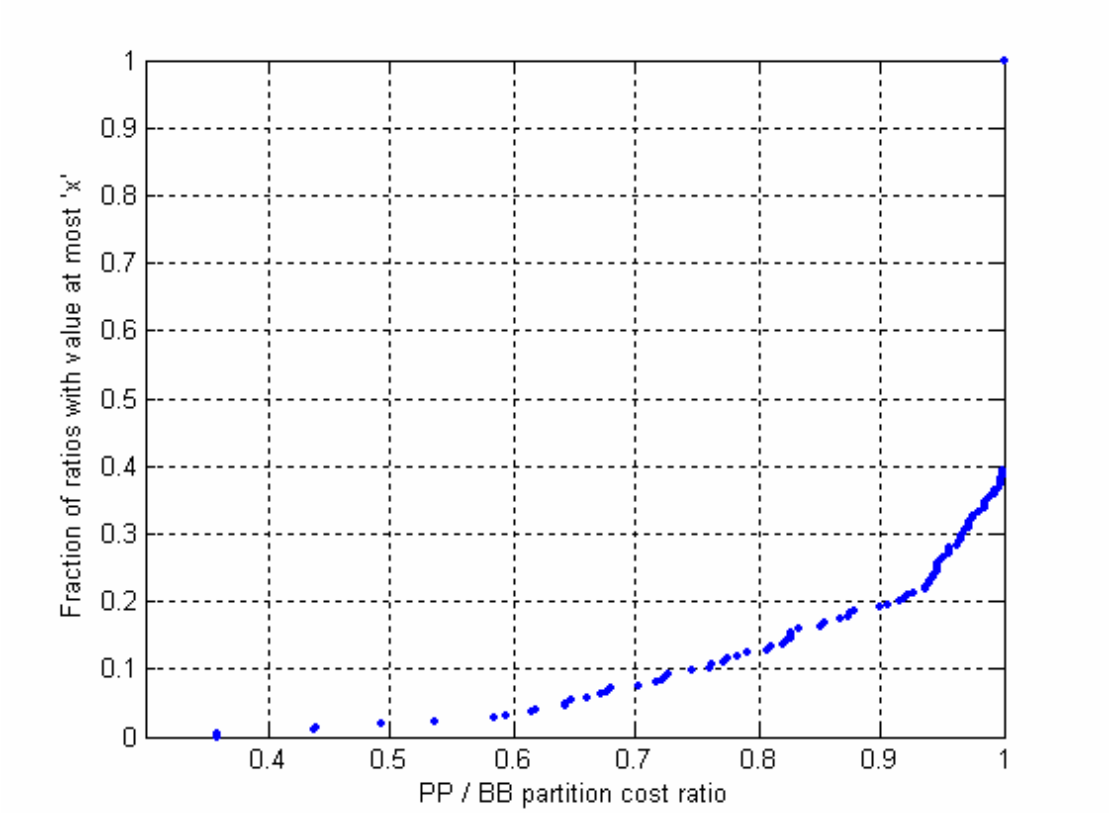

Figure 32. CDF of PP / BB partition cost ratio for set #1

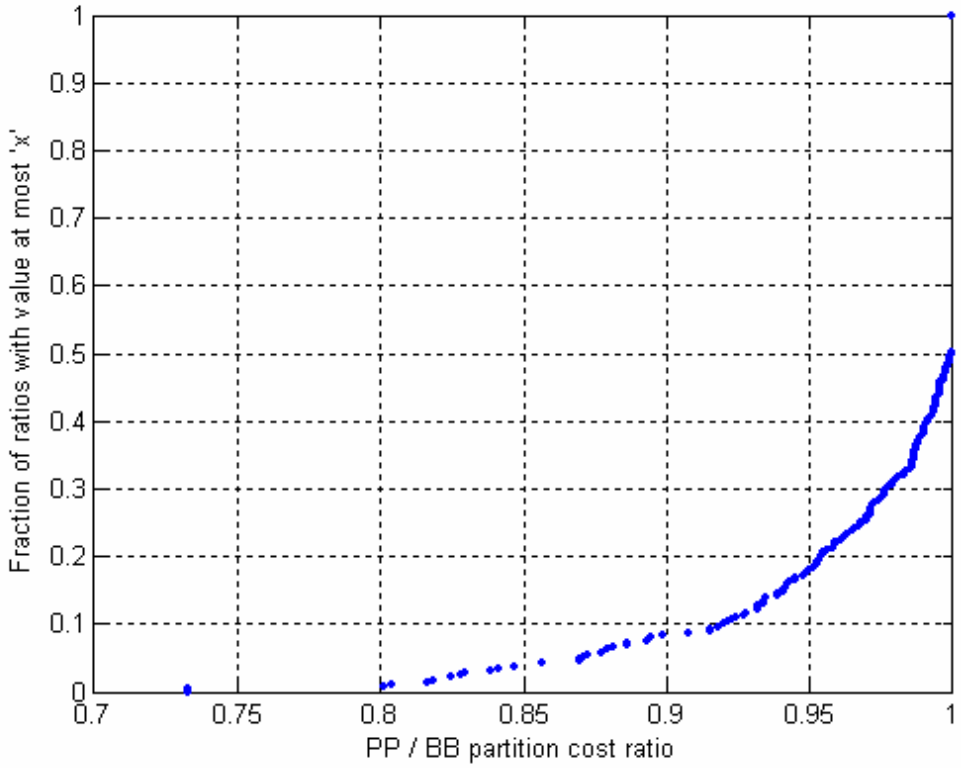

Figure 33. CDF of PP / BB partition cost ratio for set #2

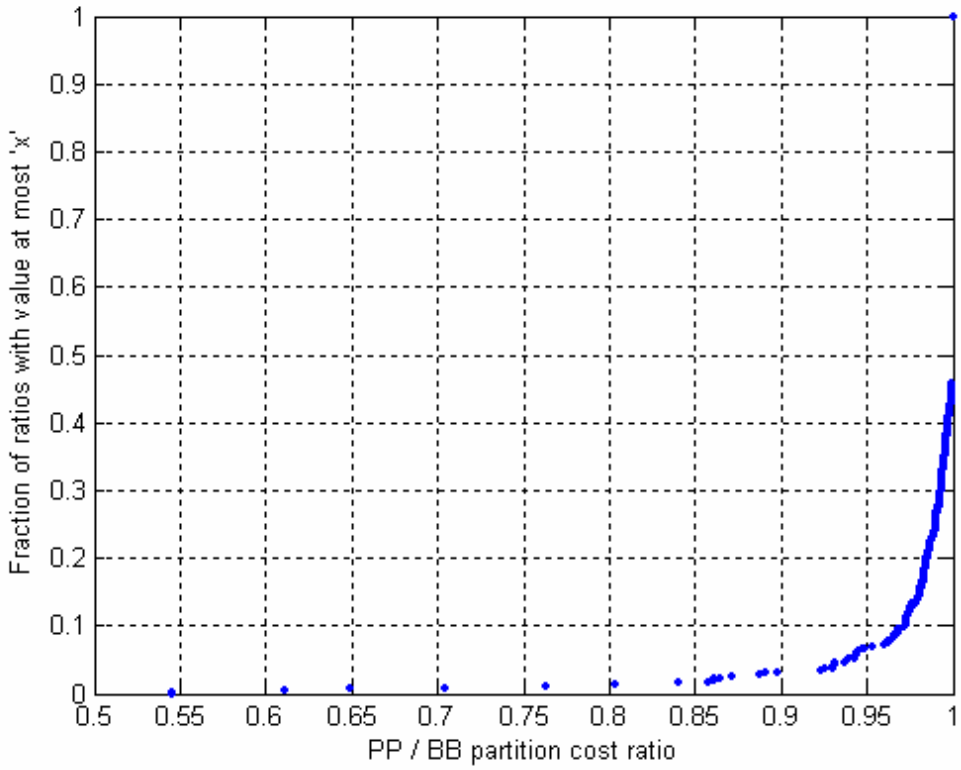

Figure 34. CDF of PP / BB partition cost ratio for set #3

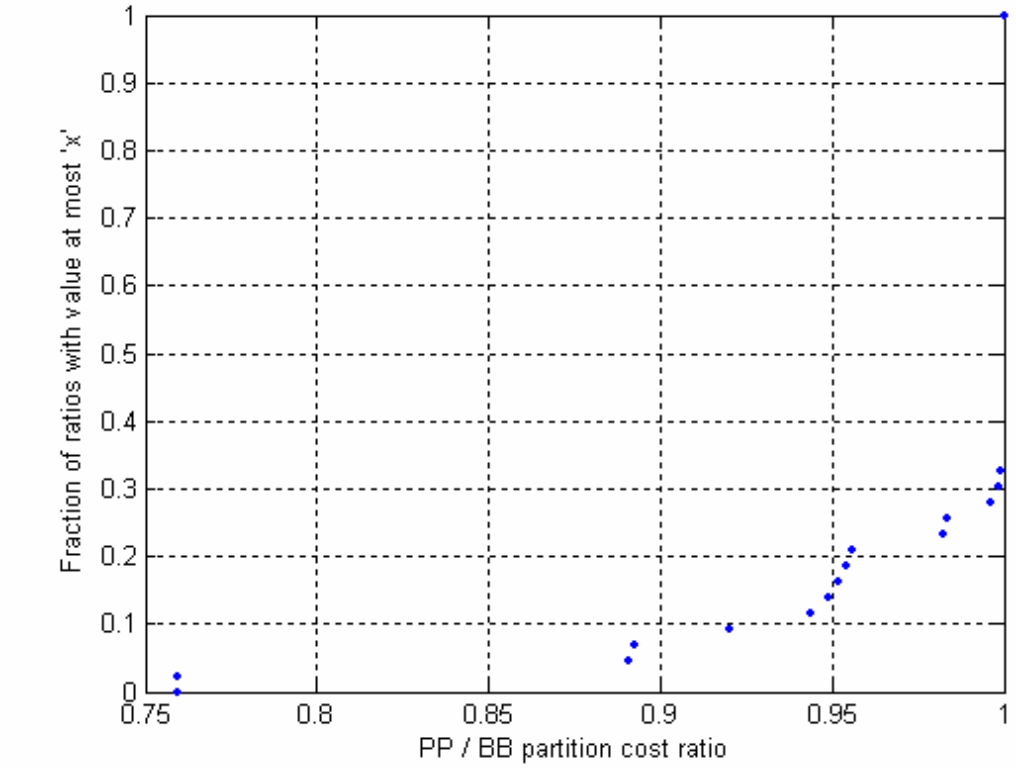

Figure 35. CDF of PP / BB partition cost ratio for set #4.1

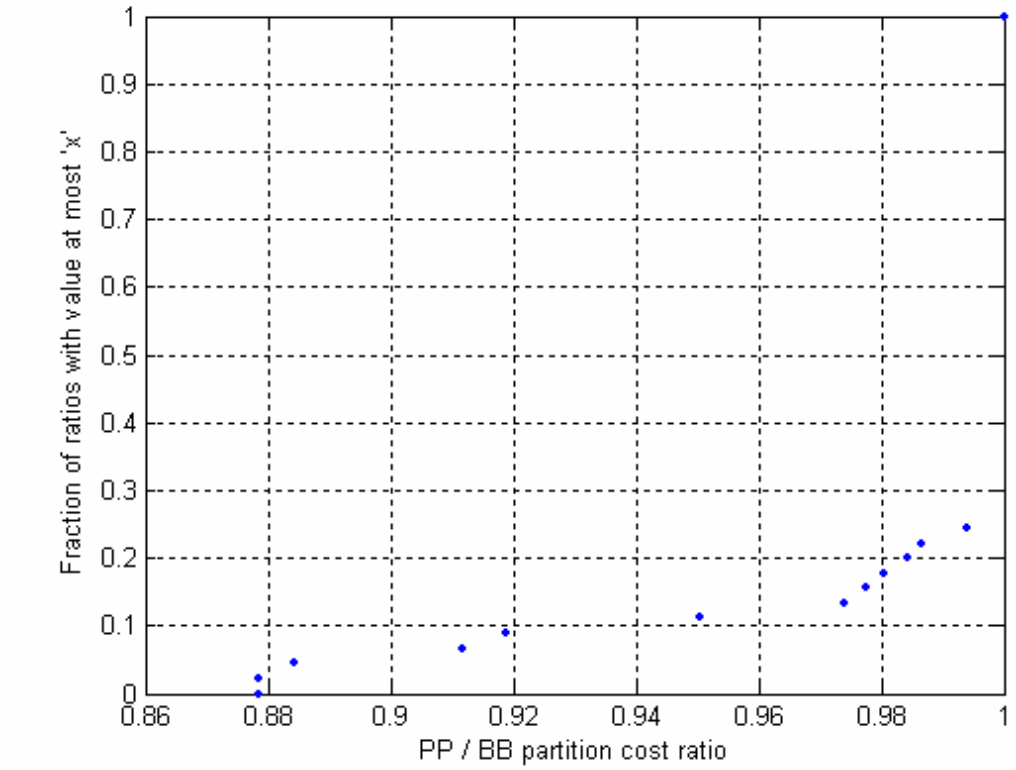

Figure 36. CDF of PP / BB partition cost ratio for set #4.2

Chapter 4

Comparison with Other Techniques

4..1 Purpose and Resources

In Chapter 2, the complexity of the algorithm was analyzed. The run time of the algorithm is bounded and can be estimated from the number of outer border nodes and the number of intermediate nodes, in the case of PP partitioning. However, most techniques concerning the partitioning problem perform iterative search in the solution space, based on generic techniques and other heuristics adjusted to the specific situation. Thus, DGP cannot be compared in terms of time complexity rather than in terms of result quality.

We know a priori that DGP neither performs an exhaustive search in the solution space nor it can produce all kinds of partition formations and hence, it yields sub-optimal results which will probably be of lower quality than those of iterative techniques. However, better results are produced with the cost of computational time. Therefore, when run time is an important factor, it is interesting to know what would be the expected trade-off between the time gained and the decrease of quality.

In [79], a set of test graphs was created, which have been subsequently used in other partitioning research works, and constitutes a benchmark reference on which techniques have found the best partitions so far. These graphs have unary weights and are publicly available at [101], where results for k-way partitioning are presented, for k = 2, 4, 8, 16, 32, and 64 and for tolerance values 0%, 1%, 2%, 3%, and 5% of the optimal partition size. We ran DGP on several planar instances, which differ in size and regularity. For $k > 2$ recursive bisection was performed on each resulted sub-graph, thus the resulting cost for each k was always subject to the cost found for $k/2$, a fact which is also noted in [40].

4..2 Bisection Issues

Several issues arose when performing a *k*-way partitioning, concerning the DGP's characteristics, the way partitions will be formed and the tolerance used at each recursive step. The archive results included partitions whose nodes were not contiguous, that is, a section could consist of two disconnected sub-graphs. As it was described in section 2.54.2, such a variation is possible using DGP, however, this would require subsequent comparisons between different formations for each sub-partitioning, which would eventually increase the time complexity. Thus, it was set that a single BB path would form the edge cut of a partitioning.

The next issue concerns what tolerance value would be given to subsequent recursive partitions, up to the final partition. Considering the recursion tree, where the initial graph would be located at root and the *k*-sections at leaves, the tolerance of each sub-partition should ensure that the final sections would comply with given tolerance. This can result in different choices of tolerance, with two extremes being either performing all sub-partitions but last, with zero tolerance and then allow the final *k*/2 bisections to partition graphs with the desired tolerance, or allowing tolerance to be used earlier, with the potential effect that at some level of recursion, the size of a sub-graph could require all subsequent bisections to have zero tolerance. In the latter case tolerance must be calibrated at each recursion level based on the value used in the previous level.

Consider the example of a 800 node graph to be partitioned in 8 sections, with tolerance 5% of the optimal section size, i.e., 5 nodes. Considering that two leaves that come from bisection of a common subgraph can have a size of at most 105 nodes, their parent can have a size of at most 210 nodes. Similarly, its parent can have a size of at most 420. This means that at first bisection the max allowed tolerance is 5% of the resulting sub section size. Nevertheless, if it is bisected with the maximum tolerance, all subsequent bisections should be done with zero tolerance, as to ensure that final section will consist of at most 105 nodes. This restricts all following partitions to a more limited solution space and therefore, what it was gained in the first sub-partition will be probably lost in all subsequent ones.

After some testing, it was observed that best results occurred more frequently for a tolerance calibration, in such a way that given tolerance would increase with recursion depth. This allows partitioning to search some better solutions both earlier and later in the recursion tree. In practice what it was done was that for a demanded tolerance for a k-way partitioning, the results of all lower or equal tolerance values, for 2·k-way partitioning were compared in terms of cost and partition imbalance. Results which were more balanced concerning final partitions and whose cost was not much greater than less balanced ones, usually yielded the best results. In the general case of size N and k-way partitioning with tolerance t as a fraction of the optimal partition size, the tolerance of each sub partitioning, in nodes, could be calibrated in the following manner: $N/k \cdot t$ for final partition, $(N/(k/2)) \cdot (t/2)$, for its parent, then $(N/(k/4)) \cdot (t/4)$, and so on (for a $k = 4 \cdot l$, $l = 1,2,3...$).

The last issues concern inherent characteristics of DGP. As it was stated in Chapter 2, the Dijkstra SSSP algorithm returns the first path among equals that

is found between the source and a node. However, other paths of same length might yield a better or a wanted ratio between sections, in contrast with the returned one. We implemented the DGP algorithm as to test all equal paths. In a weighted graph the possibility to find equal paths is such that allows the incorporation of this additional check. However, in a unary weighted graph, there's at least another equally weighted path at each hop from a source to a destination node, which makes algorithm run-time exponential. In order to avoid this, we added a number of lower magnitude to each weight. The magnitude of this number is such that does not affect the result in hops, although the partition result in terms of section balance is apparently subject to this random variation in number between weights.

Finally, as convenient as it may seem, unary weights do not guarantee a partition result. Graph structure can be such that renders the graph unbisectable using the BB technique. In fact, there were many graph instances with irregular topology, where the density of nodes, and hence edges, was different from place to place, thus affecting the procedure of shortest path search. The PP technique was used in these cases.

4..3 Comparison Results

Tables 3 to 6, contain the *k*-way partition results for different tolerance values, in percentage of optimal partition size. In each table, the first row contains *k* value and the first column contains the names of the partitioned graph instances. Results for each *k* are shown in subsequent columns. For each *k*, there correspond two sub-columns, where the result of DGP is followed by the best result which is found so far, as it is stated at the graph partitioning archive. A cost value of -1 denotes that no result could be found either with BB or PP partitioning. The irregularity of some graph's structure prevented the intermediate region process to find a useful set of intermediate nodes. Thus, intermediate nodes were chosen manually in some cases.

The results were worse by 0% to 5% for low *k* values, concerning graphs 'crack' and 'whitaker3' which have a more regular structure. These percentages increased up to 15% for higher *k* values. The corresponding percentages varied from 0%-20% for the majority of less regular graphs bisections. Graphs '3elt' and '4elt' exhibited a graph structure with a dense core with holes. The low density of border nodes led to BB shortest paths, whose routes were located in the perimeter of the graph. The holes in the core complicated the choice of intermediate region nodes. It seems that for the specific structure and embedding, there was no single node from which paths to border nodes would result in an optimal or near optimal bisection.

Concluding, graphs with a more regular structure were partitioned with results close to the current best found so far, for low *k* values and as recursion depth increased the partition quality worsened. In contrast, graph with less regular structure yielded results far from current best, yet the results improved as *k* increased and bisection was performed on smaller subgraphs with more regular structure.

graph \ *k*	2		4		8		16		32		64	
uk	25	20	54	44	106	89	181	159	295	280	504	438
3elt	-1	90	-1	201	-1	348	-1	581	-1	969	-1	1564
4elt	-1	139	-1	327	-1	553	-1	960	-1	1600	-1	2671
fe_4elt2	130	130	413	349	734	611	1191	1028	1874	1662	2912	2537
crack	185	184	386	368	812	687	1242	1108	1975	1734	2998	2648
whitaker3	128	127	390	382	769	661	1237	1108	1938	1718	2966	2569

Table 3. Comparison of partition results for 0% tolerance

graph \ *k*	2		4		8		16		32		64	
uk	25	19	51	42	104	84	174	152	285	258	471	438
3elt	-1	89	-1	199	-1	342	-1	569	-1	969	-1	1564
4elt	-1	138	-1	321	-1	535	-1	952	-1	1587	-1	2636
fe_4elt2	130	130	400	349	631	607	1097	1020	1810	1662	2783	2537
crack	183	183	369	362	767	678	1233	1092	1924	1707	2905	2571
whitaker3	126	126	381	380	689	656	1196	1093	1878	1717	2863	2567

Table 4. Comparison of partition results for 1% tolerance

graph \ *k*	2		4		8		16		32		64	
uk	22	18	47	40	94	81	165	148	273	251	457	414
3elt	133	87	236	198	400	336	661	565	1117	958	1766	1542
4elt	-1	137	-1	319	-1	527	-1	916	-1	1537	-1	2581
fe_4elt2	130	130	349	343	625	601	1074	1007	1761	1651	2729	2536
crack	182	182	361	360	740	676	1191	1082	1883	1679	2819	2569
whitaker3	126	126	379	378	684	655	1182	1092	1810	1686	2793	2535

Table 5. Comparison of partition results for 3% tolerance

graph \ *k*	2		4		8		16		32		64	
uk	22	18	45	40	89	78	159	139	266	246	450	410
3elt	133	87	236	197	389	330	653	560	1100	950	1719	1539
4elt	-1	137	-1	315	-1	520	-1	909	-1	1537	-1	2581
fe_4elt2	130	130	349	335	623	588	1061	1004	1747	1645	2706	2516
crack	182	182	361	360	714	667	1163	1080	1843	1679	2718	2569
whitaker3	126	126	378	378	680	650	1177	1084	1796	1686	2756	2535

Table 6. Comparison of partition results for 5% tolerance

Chapter 5

Conclusion and Application Suggestions

5..1 Conclusion

We presented the Dual Graph Partitioning algorithm (DGP). We showed that its main advantage was that it performs quite well for an average deterministic polynomial time of $O(V^{3/2} \cdot \log(V))$. We investigated its weaknesses and devised a heuristic to determine whether a graph is likely to be partitioned by DGP.

We defined the graph partitioning problem and analyzed the DGP bisection algorithm. The method is based on exploiting characteristics of the dual graph and thus, it can only be applied in planar graphs. We ran the algorithm for a series of graph sets, each one focusing on a different characteristic, in order to find the graph properties that primarily affect partition results. The graph structure used was a circle shaped hexagonal grid. We modeled weight value and spatial distribution with the notion of population center nodes. Higher weights are located around these nodes, and weight values decrease with edge distance from its closest population center.

In order to infer the ratio of weight value decrement, we ran a number of simulations using the PRAGMA user mobility model. PRAGMA fitted best with our model of population centers since it assumes individual movement towards attractor points.

All graph set partitioning results led to the conclusion that the algorithm returns a valid partition under certain topological conditions, concerning weight distribution in space. We concluded that it is important that border to border node shortest paths in the dual graph be able to pass through or near the central region of the graph. Thus, the partition succession was correlated with population center node location.

We managed to measure this relation through the distribution of distances between population centers and border nodes. Results confirmed that partitioned graphs had more population center nodes closer to their border than unpartitioned. Distributions of mean such distances of each graph, exhibited a normal like shape, centered at a lower distance value for partitioned graphs than unpartitioned. We statistically verified this observation by performing a two-sample T-test.

As a counter-scenario we ran the algorithm in random graphs, where there is not a particular scheme of weight value and spatial distribution. The partition succession was almost absolute.

We completed the analysis of the algorithm by presenting a variation which guarantees a result with the exchange of higher computational cost. This variation can be used when partitions cannot be found with the original form of the algorithm or solely, when more run-time can be tolerated, since it was proven that it can compute all results of the original version and it was shown that it yields better or equal results.

The algorithm was compared with other well known techniques, in terms of performance. A number of graphs were used from an online graph partitioning archive, where current best partition results using other techniques are available. These techniques concern iterative search in the solution space and thus, their results are better with the cost of run time. The partitioning results concerned multi-way partitioning apart from bisection. Recursive bisection with DGP was used in the case of k-way partitioning for k = 4, 8, ..., 32. The quality of the partition was not very far from the best found so far and for certain graphs they even matched for specific partition parameters. For more regular graphs, results for higher number of partitions were worse than those for two or four sections, since the disadvantage between other techniques and bisection was aggregated at each recursion. Moreover, the more regular the structure of a graph was, the closer were the results to the current best. However, some graph structures were unbisectable even with the variation of the algorithm, due to the difficulty to locate intermediate nodes, or possibly due to the lack of path pairs that would bisect the graph, for any intermediate node.

5..2 Future Suggestions and Applications

Certain transformation of the graph structure could be proposed as to eliminate structure-related problems. An example of such transformation is presented in Figures 37 and 38. In the first figure the original graph and its dual are shown. The graph structure is such that all shortest paths between border nodes of the dual graph route via its perimeter. In its core there are many more nodes and thus paths via these nodes are longer. In the next figure, a symmetrical inversion of graph nodes transformed the inner hole as to constitute the external face and consequently, the former outer face constituted a new internal hole. Note that the border now consists of three times as many nodes than the original dual graph. On the contrary, there are fewer nodes in the core. As a result, there are more candidate paths and there is one passing through the central region that bisected the graph optimally, represented by red dual nodes.

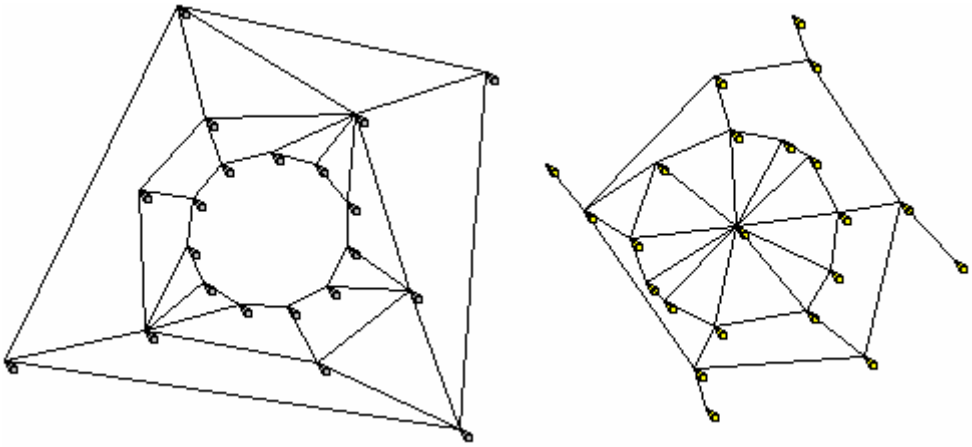

Figure 37. Original graph and its Dual

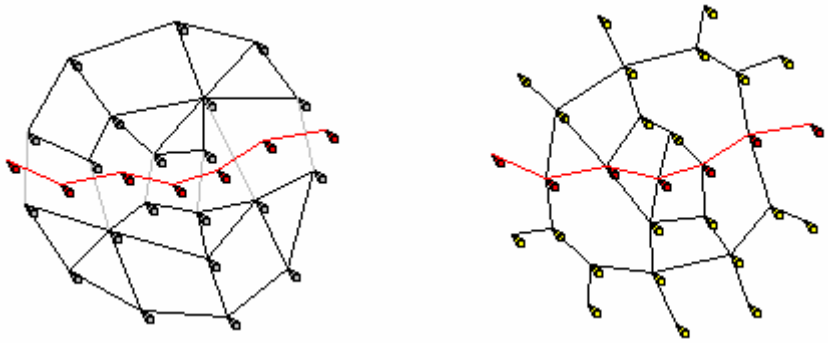

Figure 38. Transformed graph and its Dual

We showed that the algorithm ensures an average deterministic polynomial time, while performing quite well. In partitioning applications were time is essential DGP can be used for practically good results. For a further computational cost the variation can be used for even better results and a partition guarantee. Note that other techniques of defining the intermediate nodes, apart from these described here, can be used as to exploit possible info related to the problem, as long as they bound the computational time within a practical limit.

On the other hand, when partition quality is of primary importance, the variation of the algorithm can be extended as to widen the search in the solution space. Pairs of intermediate nodes can be used and the path between these intermediate nodes can be then combined with paths from border nodes to these intermediate nodes. Note that for a proper number of intermediate nodes the computational time could still be bounded with a relatively low computational cost. This particular variation could bisect some graphs of the online archive that were not bisected by the algorithm. Moreover, more time-consuming variations could be proposed, such as using paths between all possible intermediate graph nodes, or even combinations of all previous propositions. Other variations could concern recursive bisection. At each level of recursion, different partition schemes can be performed iteratively. These schemes can concern choice between partitions using connected or disconnected subgraphs, and/or the distribution of tolerance along the recursion tree.

For applications where the iterative partitioning algorithms are traditionally preferred and used, the algorithm can be incorporated as a diversification technique. Different results that are found by the algorithm during its execution or results using different parameters can be used in the diversification phase of generic techniques such as Tabu search. In such techniques, when results cease to improve after some iterations, a diversion in the result space is needed and different starting search points are used. DGP could provide different sub-optimal starting points depending on the version of the algorithm or tolerance that will be used.

Finally, it could be worthwhile to attempt to overcome the restriction that the algorithm can only be applied to planar graphs. Graph planarization techniques can be used towards this direction. Heuristic algorithms that try to minimize edge crossing could be combined with a simple planarization scheme of replacing each cross with a pseudo-node. The resulting planar graph would be then partitioned and pseudo nodes would not be counted in the section definition process. Additionally, more elaborate techniques could involve setting their weight to zero, during the Dijkstra SSSP procedure, when another edge which is created from the same original edge is already included in the current shortest path.

Chapter 6
References

[1] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, “Equations of state calculations by fast computing machines”, J. Chem. Phys. 21 (1953) 1087-1092

[2] Kernighan, B. W. 1969. “Optimal segmentation points for programs”. In Proceedings of the Second Symposium on Operating Systems Principles (Princeton, New Jersey, October 20 - 22, 1969). SOSP '69. ACM, New York, NY, 47-53.

[3] B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs." The Bell Systems Tech. J., vol. 49, pp. 291-307, 1970

[4] W.L.G. Koontz, K. Fukunaga, “A nonparametric valley-seeking technique for cluster analysis”. IEEE Trans. Comput. 21 (1972) 171-178

[5] N. Biggs, E. Lloyd, and R. Wilson, “Graph Theory 1736-1936”, Clarendon Press Oxford, 1976 (ISBN 0-19-853901-0)

[6] Chandy, K.M., and Misra, J., “Distributed Simulation: A Case Study in Design and Verification of Distributed Programs”, IEEE Trans. on Software Engineering, SE-5, Sept. 1979, pp 440-452

[7] C. M. Fiduccia , R. M. Mattheyses, “A linear-time heuristic for improving network partitions”. Proceedings of the 19th conference on Design automation, p.175-181, January 1982

[8] S. Kirkpatrick and C. D. Gelatt and M. P. Vecchi, “Optimization by Simulated Annealing”, Science, Vol 220, Number 4598, pages 671-680, 1983

[9] M. Goldberg and M. Burstein, “Heuristic improvement technique for bisection of VLSI networks,” Proc. Int’l Conf. Computer Design: VLSI in Computers, pp. 122-125, 1983

[10] B. Krishnamurthy, “An improved min-cut algorithm for partitioning VLSI networks,” IEEE Trans. Computers, vol. 33, no. 5, May 1984

[11] Matula, D. W. 1985. “Concurrent flow and concurrent connectivity on graphs”. In Graph theory with Applications To Algorithms and Computer Science, Y. Alavi, G. Chartrand, D. R. Lick, C. E. Wall, and L. Lesniak, Eds. John Wiley & Sons, New York, NY, 543-559

[12] Thompson, B., Matula D. W., “A Flow Rerouting Algorithm for the Maximum Concurrent Flow Problem with Variable Capacities and Demands, and its Applications to Cluster Analysis”. Technical Report 86-CSE-12, Department of Computer Science and Engineering, SMU, 1986

[13] Biswas, J. and Matula, D. W. 1986. “Two flow routing algorithms for the maximum concurrent flow problem”. In Proceedings of 1986 ACM Fall Joint Computer Conference (Dallas, Texas, United States). IEEE Computer Society Press, Los Alamitos, CA, 629-636

[14] Hong, Y. C., Payne, T. H., and Ferguson, L. B. 1986. “Graph allocation in static dataflow systems”. SIGARCH Comput. Archit. News 14, 2 (May. 1986), 55-64

[15] Koutsougeras, C., Papachristou, C. A., and Vemuri, R. R. 1986. Data flow graph partitioning to reduce communication cost. SIGMICRO Newsl. 17, 4 (Dec. 1986), 82-91

[16] Natour, I. A. 1986, “Control Flow and Data Flow Analysis for the Translation of Control Graphs into Data Graphs” (Parallel Processing). Doctoral Thesis. UMI Order Number: AAI8706192., Wayne State University

[17] Gilbert, J. R. and Zmijewski, E. 1988. “A Parallel Graph Partitioning Algorithm for a Message-Passing Multiprocessor”. In Proceedings of the 1st international Conference on Supercomputing (June 08 - 12, 1987). E. N. Houstis, T. S. Papatheodorou, and C. D. Polychronopoulos, Eds. Lecture Notes In Computer Science, vol. 297. Springer-Verlag, London, 498-513

[18] Shahrokhi, F. and Matula, D. W. 1987. “On solving large maximum concurrent flow problems”. In Proceedings of the 15th Annual Conference on Computer Science (St. Louis, Missouri, United States). CSC '87. ACM, New York, NY, 205-209

[19] Dey, S. and Srimani, P. K. 1988. “Parallel VLSI computation of all shortest paths in a graph”. In Proceedings of the 1988 ACM Sixteenth Annual Conference on Computer Science (Atlanta, Georgia, United States). CSC '88. ACM, New York, NY, 373-379

[20] Patil, S., Banerjee, P., and Polychronopoulos, C. 1989. “Efficient circuit partitioning algorithms for parallel logic simulation”. In Proceedings of the 1989 ACM/IEEE Conference

on Supercomputing (Reno, Nevada, United States, November 12 - 17, 1989). Supercomputing '89. ACM, New York, NY, 361-370
[21] Liu, J. W. 1989. “A graph partitioning algorithm by node separators”. ACM Trans. Math. Softw. 15, 3 (Sep. 1989), 198-219
[22] Savage, J. E. and Wloka, M. G. 1989 Heuristics for Parallel Graph-Partitioning. Technical Report. UMI Order Number: CS-89-41., Brown University
[23] Jill J. Hallenbeck , James R. Cybrynski , Nick Kanopoulos , Tassos Markas , Nagesh Vasanthavada, “The Test Engineer's Assistant: A Support Environment for Hardware Design for Testability”, Computer, v.22 n.4, p.59-68, April 1989
[24] J. J. Hallenbeck, N. Kanopoulos, J. R. Cybrinski, “The Test Engineer's Assistant: A design environment for testable and diagnostable systems” IEEE Trans. on Industrial Electronics, Special issue on Electronic testing, May 1989
[25] Tragoudas, S., Farrell, R., and Makedon, F. 1991. “Circuit partitioning into small sets: a tool to support testing with further applications”. In Proceedings of the Conference on European Design Automation (Amsterdam, The Netherlands, February 25 - 28, 1991). European Design Automation Conference. IEEE Computer Society Press, Los Alamitos, CA, 518-522
[26] Liu, S. F. and Soffa, M. L. 1992. “Parallel Task Assignment by Graph Partitioning”. In Proceedings of the 4th international PARLE Conference on Parallel Architectures and Languages Europe (June 15 - 18, 1992). D. Etiemble and J. Syre, Eds. Lecture Notes In Computer Science, vol. 605. Springer-Verlag, London, 965-966
[27] Liu, J. 1992, “Graph Embedding and Data Communication in a Large Family of Network Topologies”. Doctoral Thesis. UMI Order Number: UMI Order No. GAX93-03018., Michigan State University
[28] Silc, J. and Robic, B. 1993 “Program Partitioning for a Control/Data Driven Computer”. Technical Report. UMI Order Number: CSD-TR-93-7., Jozef Stefan Institute
[29] Nandy, B. and Loucks, W. M. 1993. “On a parallel partitioning technique for use with conservative parallel simulation”. In Proceedings of the Seventh Workshop on Parallel and Distributed Simulation (San Diego, California, United States, May 16 - 19, 1993). R. Bagrodia and D. Jefferson, Eds. PADS '93. ACM, New York, NY, 43-51
[30] Heath, L. S. and Lavinus, J. W. 1993 “Optimal and Random Partitions of Random Graphs”. Technical Report. UMI Order Number: TR-93-24., Virginia Polytechnic Institute & State University
[31] Martin, O. C. and Otto, S. W. 1993 “Combining Simulated Annealing with Local Search Heuristics”. Technical Report. UMI Order Number: CSE-94-016., Oregon Graduate Institute School of Science & Engineering
[32] Hsu, T., Ramachandran, V., and Dean, N. 1993 “Implementation of Parallel Graph Algorithms on a Massively Parallel Simdcomputer with Virtual Processing”. Technical Report. UMI Order Number: CS-TR-93-14., University of Texas at Austin
[33] Glover F., and Laguna M. “Tabu Search” (1993)
[34] Philip Klein , Serge A. Plotkin , Satish Rao, “Excluded minors, network decomposition, and multicommodity flow”, Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, p.682-690, May 16-18, 1993, San Diego, California, United States [10.1145/167088.167261]
[35] Martin, O. C. and Otto, S. W. 1994 “Partioning of Unstructed Meshes for Load Balancing”. Technical Report. UMI Order Number: CSE-94-017., Oregon Graduate Institute School of Science & Engineering
[36] Boukerche, A. and Tropper, C. 1994. “A static partitioning and mapping algorithm for conservative parallel simulations”. In Proceedings of the Eighth Workshop on Parallel and Distributed Simulation (Edinburgh, Scotland, United Kingdom, July 06 - 08, 1994). D. K. Arvind, R. Bagrodia, and J. Y. Lin, Eds. PADS '94. ACM, New York, NY, 164-172
[37] Maini, H., Mehrotra, K., Mohan, C., and Ranka, S. 1994. “Genetic algorithms for graph partitioning and incremental graph partitioning”. In Proceedings of the 1994 ACM/IEEE Conference on Supercomputing (Washington, D.C., November 14 - 18, 1994). Supercomputing '94. ACM, New York, NY, 449-457
[38] Ou, C. and Ranka, S. 1994. “Parallel incremental graph partitioning using linear programming”. In Proceedings of the 1994 Conference on Supercomputing (Washington, D.C., United States). IEEE Computer Society Press, Los Alamitos, CA, 458-467
[39] Misic, J. and Jovanovic, Z. 1994. “Communication Aspects of the Star Graph Interconnection Network”. IEEE Trans. Parallel Distrib. Syst. 5, 7 (Jul. 1994), 678-687
[40] Horst D. Simon and Shang-Hua Teng, “How good is recursive bisection?” (1995)

[41] Buch, P., Sanghavi, J., and Sangiovanni-Vincentelli, A. 1995. “A parallel graph partitioner on a distributed memory multiprocessor”. In Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Computation (Frontiers'95) (February 06 - 09, 1995). FRONTIERS. IEEE Computer Society, Washington, DC, 360

[42] Kapp, K. L., Hartrum, T. C., and Wailes, T. S. 1995. “An improved cost function for static partitioning of parallel circuit simulations using a conservative synchronization protocol”. In Proceedings of the Ninth Workshop on Parallel and Distributed Simulation (Lake Placid, New York, United States, June 13 - 16, 1995). C. IEEE Computer Society Press, Ed. Workshop on Parallel and Distributed Simulation. IEEE Computer Society, Washington, DC, 78-85

[43] Saab, Y. G. 1995. “A Fast and Robust Network Bisection Algorithm”. IEEE Trans. Comput. 44, 7 (Jul. 1995), 903-913

[44] Hendrickson, B. and Leland, R. 1995. “A multilevel algorithm for partitioning graphs”. In Proceedings of the 1995 ACM/IEEE Conference on Supercomputing (Cdrom) (San Diego, California, United States, December 04 - 08, 1995). Supercomputing '95. ACM, New York, NY, 28

[45] Jonathan Berry, W. and Goldberg, M. K. 1995 “Path Optimization for Graph Partitioning Problems”. Technical Report. UMI Order Number: 95-34., Center for Discrete Mathematics & Theoretical Computer Science

[46] Karypis, G. and Kumar, V. 1995. “Analysis of multilevel graph partitioning”. In Proceedings of the 1995 ACM/IEEE Conference on Supercomputing (Cdrom) (San Diego, California, United States, December 04 - 08, 1995). Supercomputing '95. ACM, New York, NY, 29

[47] G. Even , J. Naor , S. Rao , B. Schieber, “Divide-and-conquer approximation algorithms via spreading metrics”. Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95), p.62, October 23-25, 1995

[48] Oommen, B. J. and de St. Croix, E. V. 1996. “Graph Partitioning Using Learning Automata”. IEEE Trans. Comput. 45, 2 (Feb. 1996), 195-208

[49] Aarts, E. H., Essink, G., and de Kock, E. A. 1996. “Recursive Bipartitioning of Signal Flow Graphs for Programmable Video Signal Processors”. In Proceedings of the 1996 European Conference on Design and Test (March 11 - 14, 1996). European Design and Test Conference. IEEE Computer Society, Washington, DC, 460

[50] Bui, T. N. and Moon, B. R. 1996. “Genetic Algorithm and Graph Partitioning”. IEEE Trans. Comput. 45, 7 (Jul. 1996), 841-855

[51] Karypis, G. and Kumar, V. 1996. “Parallel multilevel k-way partitioning scheme for irregular graphs”. In Proceedings of the 1996 ACM/IEEE Conference on Supercomputing (Cdrom) (Pittsburgh, Pennsylvania, United States, January 01 - 01, 1996). Conference on High Performance Networking and Computing. IEEE Computer Society, Washington, DC, 35

[52] Shantanu Dutt , Wenyong Deng, “VLSI circuit partitioning by cluster-removal using iterative improvement techniques”, Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design, p.194-200, November 10-14, 1996, San Jose, California, United States

[53] I. G. Tollis. “Optimal partitioning of cellular networks”. In IEEE International Conference on Communications, ICC'96, volume 3, pages 1377 -1381, 1996

[54] Lim, K. and Lee, Y. 1997. “Optimal Partitioning of Heterogeneous Traffic Sources in Mobile Communications Networks”. IEEE Trans. Comput. 46, 3 (Mar. 1997), 312-325

[55] Seong, K. S. and Kyung, C. M. 1997. “Two-way partitioning based on direction vector”. In Proceedings of the 1997 European Conference on Design and Test (March 17 - 20, 1997). European Design and Test Conference. IEEE Computer Society, Washington, DC, 306

[56] Chan, T. F., Ciarlet, P., and Szeto, W. K. 1997. “On the Optimality of the Median Cut Spectral Bisection Graph Partitioning Method”. SIAM J. Sci. Comput. 18, 3 (May. 1997), 943-948

[57] Kuo, M. and Cheng, C. 1997. “A network flow approach for hierarchical tree partitioning”. In Proceedings of the 34th Annual Conference on Design Automation (Anaheim, California, United States, June 09 - 13, 1997). DAC '97. ACM, New York, NY, 512-517

[58] Shazely, S., Baraka, H., and Abdel-Wahab, A. 1998. “Solving Graph Partitioning Problem Using Genetic Algorithms”. In Proceedings of the 1998 Midwest Symposium on Systems and Circuits (August 09 - 12, 1998). MWSCAS. IEEE Computer Society, Washington, DC, 302

[59] M. Matsumoto and T. Nishimura, “Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator”, ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 (1998)

[60] Battiti, R. and Bertossi, A. A. 1999. “Greedy, Prohibition, and Reactive Heuristics for Graph Partitioning”. IEEE Trans. Comput. 48, 4 (Apr. 1999), 361-385
[61] Mirko Ferracioli, Roberto Verdone, “A General Methodology Based on the Handover Rate for Network Planning of Cellular Radio Networks Based on ATM”. (1999)
[62] Hilderman, R. J., Hamilton, H. J., and Cercone, N. 1999. Data Mining in Large Databases Using Domain Generalization Graphs. J. Intell. Inf. Syst. 13, 3 (Nov. 1999), 195-234.
[63] C. Walshaw and M. Cross. “Mesh Partitioning: a Multilevel Balancing and Refinement Algorithm”. SIAM J. Sci. Comput., 22(1):63-80, 2000
[64] Saha, D., Mukherjee, A., and Bhattacharya, P. S. 2000. “A Simple Heuristic for Assignment of Cells to Switches in a PCS Network”. Wirel. Pers. Commun. 12, 3 (Mar. 2000), 209-223
[65] Kobler, D. and Rotics, U. 2001. “Polynomial algorithms for partitioning problems on graphs with fixed clique-width (extended abstract)”. In Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (Washington, D.C., United States, January 07 - 09, 2001). Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, 468-476
[66] Schweitz, E. A. and Agrawal, D. P. 2002. “A Parallelization Domain Oriented Multilevel Graph Partitioner”. IEEE Trans. Comput. 51, 12 (Dec. 2002), 1435-1441
[67] Ricci, R., Alfeld, C., and Lepreau, J. 2003. “A solver for the network testbed mapping problem”. SIGCOMM Comput. Commun. Rev. 33, 2 (Apr. 2003), 65-81
[68] Amir, E., Krauthgamer, R., and Rao, S. 2003. “Constant factor approximation of vertex-cuts in planar graphs”. In Proceedings of the Thirty-Fifth Annual ACM Symposium on theory of Computing (San Diego, CA, USA, June 09 - 11, 2003). STOC '03. ACM, New York, NY, 90-99
[69] Drechsler, R., Günther, W., Eschbach, T., Linhard, L., and Angst, G. 2002. “Recursive Bi-Partitioning of Netlists for Large Number of Partitions”. In Proceedings of the Euromicro Symposium on Digital Systems Design (September 04 - 06, 2002). DSD. IEEE Computer Society, Washington, DC, 38
[70] Yigal Begerano, Nicole Immorlica, Joseph (Seffi) Naor, Mark Smith, “Efficient Location Area Planning for Personal Communication Systems” (2003)
[71] Ilker Dermirkol, Cem Ersoy, M. Ufuk Caglayan, Hakan Delic, “Location Area Planning and Cell-to-Switch Assignment in Cellular Networks” (2003)
[72] Karypis, G. 2003. “Multi-Constraint Mesh Partitioning for Contact/Impact Computations”. In Proceedings of the 2003 ACM/IEEE Conference on Supercomputing (November 15 - 21, 2003). Conference on High Performance Networking and Computing. IEEE Computer Society, Washington, DC, 56
[73] Washio, Motoda, H. “State of the art of graph-based data mining”. SIGKDD Explor. Newsl. 5, 1 (Jul. 2003), 59-68
[74] Kim, Y. and Moon, B. 2004. “Investigation of the Fitness Landscapes in Graph Bipartitioning: An Empirical Study”. Journal of Heuristics 10, 2 (Mar. 2004), 111-133
[75] Saab, Y. G. 2004. “An Effective Multilevel Algorithm for Bisecting Graphs and Hypergraphs”. IEEE Trans. Comput. 53, 6 (Jun. 2004), 641-652
[76] Spielman, D. A. and Teng, S. 2004. “Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems”. In Proceedings of the Thirty-Sixth Annual ACM Symposium on theory of Computing (Chicago, IL, USA, June 13 - 16, 2004). STOC '04. ACM, New York, NY, 81-90
[77] Andreev, K. and Räcke, H. 2004. “Balanced graph partitioning”. In Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (Barcelona, Spain, June 27 - 30, 2004). SPAA '04. ACM, New York, NY, 120-124
[78] Martínez, A. M., Mittrapiyanuruk, P., and Kak, A. C. 2004. “On combining graph-partitioning with non-parametric clustering for image segmentation”. Comput. Vis. Image Underst. 95, 1 (Jul. 2004), 72-85
[79] A.J. Soper, C. Walshaw and M. Cross, “A Combined Evolutionary Search and Multilevel Optimisation Aproach to Graph-Partitioning” J. Global Optimization, 29(2):225-241, 2004
[80] Aringhieri, R. and Dell'Amico, M. 2005. “Comparing Metaheuristic Algorithms for Sonet Network Design Problems”. Journal of Heuristics 11, 1 (Jan. 2005), 35-57
[81] Chen, J., Wu, B., Delap, M., Knutsson, B., Lu, H., and Amza, C. 2005. “Locality aware dynamic load management for massively multiplayer games”. In Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (Chicago, IL, USA, June 15 - 17, 2005). PPoPP '05. ACM, New York, NY, 289-300

[82] V. Borrel, M. D. de Amorim, and S. Fdida. “A preferential attachment gathering mobility model”. IEEE Communications Letters, 9(10):900--902, 2005
[83] Langham, E. 2005. “Emergent restructuring of resources in ant colonies: a swarm-based approach to partitioning”. In Proceedings of the 18th international Conference on innovations in Applied Artificial intelligence (Bari, Italy, June 22 - 24, 2005). M. Ali and F. Esposito, Eds. Lecture Notes In Computer Science. Springer-Verlag, London, 638-647
[84] Chamberland, S. and Pierre, S. 2005. “On the design problem of cellular wireless networks”. Wirel. Netw. 11, 4 (Jul. 2005), 489-496
[85] Dhillon, I., Guan, Y., and Kulis, B. 2005. “A fast kernel-based multilevel algorithm for graph clustering”. In Proceedings of the Eleventh ACM SIGKDD international Conference on Knowledge Discovery in Data Mining (Chicago, Illinois, USA, August 21 - 24, 2005). KDD '05. ACM, New York, NY, 629-634
[86] Ketkar, N. S., Holder, L. B., and Cook, D. J. 2005. “Qualitative comparison of graph-based and logic-based multi-relational data mining: a case study”. In Proceedings of the 4th international Workshop on Multi-Relational Mining (Chicago, Illinois, August 21 - 21, 2005). MRDM '05. ACM, New York, NY, 25-32
[87] Feige, U. and Krauthgamer, R. 2006. “A Polylogarithmic Approximation of the Minimum Bisection”. SIAM Rev. 48, 1 (Jan. 2006), 99-130
[88] Meals, B. 2006. “Hierarchical decomposition algorithm for hardware/software partitioning”. In Proceedings of the 44th Annual Southeast Regional Conference (Melbourne, Florida, March 10 - 12, 2006). ACM-SE 44. ACM, New York, NY, 18-23
[89] Member-Baris Sumengen and Fellow-B. S. Manjunath 2006. “Graph Partitioning Active Contours (GPAC) for Image Segmentation”. IEEE Trans. Pattern Anal. Mach. Intell. 28, 4 (Apr. 2006), 509
[90] Khandekar, R., Rao, S., and Vazirani, U. 2006. “Graph partitioning using single commodity flows”. In Proceedings of the Thirty-Eighth Annual ACM Symposium on theory of Computing (Seattle, WA, USA, May 21 - 23, 2006). STOC '06. ACM, New York, NY, 385-390
[91] Hwang, I., Kim, Y., and Moon, B. 2006. “Multi-attractor gene reordering for graph bisection”. In Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (Seattle, Washington, USA, July 08 - 12, 2006). GECCO '06. ACM, New York, NY, 1209-1216
[92] Dereniowski, D. and Kubale, M. 2006. “Efficient Parallel Query Processing by Graph Raning”. Fundam. Inf. 69, 3 (Aug. 2006), 273-285
[93] Loguinov, D., Casas, J., and Wang, X. “Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience”. IEEE/ACM Trans. Netw. 13, 5 (Oct. 2005), 1107-1120. DOI= http://dx.doi.org/10.1109/TNET.2005.857072
[94] C. Gil, R. Banos, M. G. Montoya, J. Gomez, “Performance of Simulated Annealing, Tabu Search, and Evolutionary Algorithms for Multi-objective Network Partitioning”. Algorithmic Operations Research, Vol 1, No 1 (2006)
[95] Povh, J. and Rendl, F. 2007. “A Copositive Programming Approach to Graph Partitioning”. SIAM J. on Optimization 18, 1 (Feb. 2007), 223-241
[96] Aleksandrov, L., Djidjev, H., Guo, H., and Maheshwari, A. 2007. “Partitioning planar graphs with costs and weights”. J. Exp. Algorithmics 11 (Feb. 2007), 1.5
[97] James, T., Vroblefski, M., and Nottingham, Q. 2007. A hybrid grouping genetic algorithm for the registration area planning problem. Comput. Commun. 30, 10 (Jul. 2007), 2180-2190
[98] Dhillon, I. S., Guan, Y., and Kulis, B. 2007. “Weighted Graph Cuts without Eigenvectors A Multilevel Approach”. IEEE Trans. Pattern Anal. Mach. Intell. 29, 11 (Nov. 2007), 1944-1957
[99] Nguyen, S. N., Orlowska, M. E., and Li, X. 2007. “Graph mining based on a data partitioning approach”. In Proceedings of the Nineteenth Conference on Australasian Database - Volume 75 (Gold Coast, Australia, December 03 - 04, 2007). ACM International Conference Proceeding Series, vol. 313. Australian Computer Society, Darlinghurst, Australia, 31-37
[100] Angles, R. and Gutierrez, C. 2008. “Survey of graph database models”. ACM Comput. Surv. 40, 1 (Feb. 2008), 1-39
[101] http://staffweb.cms.gre.ac.uk/~wc06/partition/

*Appendix A*Related Work

A.1 Preface

There has been extensive work on graph partitioning algorithms with applications on various contexts of science, using different approaches and techniques. There will follow an elaborate presentation of the most important techniques along with their related work as well as some interesting heuristics, concerning graph partitioning.

A.2 Subsequent Maximization Algorithms

This category of algorithms is based on the fact that the problem of minimizing the external cost (that is, the cost between partitions) is equivalent with the problem of maximizing the internal cost. Given that a partition must have at most p nodes (and thus, $\lceil n/p \rceil$ partitions will be created, for a graph with n vertices), a Subsequent Maximization Algorithm (SMA) chooses a starting node and creates the partition by adding each time a node based on a defined metric, whose implementation differs in the literature and depends on the context. This is repeated until all n/p partitions are created. The choice of the starting nodes of each partition can be implemented with various ways: randomly or by using again a metric with respect to already created partitions.

In [61], Ferracioli et al. propose some sub-optimal heuristic partitioning algorithms. The algorithms use the "handover matrix", which is the matrix containing the cost of each edge and the weight of the nodes (in the diagonal). These algorithms are based on the notion of "proximity", between a node v and a set of nodes W, which is the sum of the costs of the edges connecting the node v and the nodes of the set W. In their SMA implementation, the authors choose to add in the currently formed partition the node which has the maximum proximity to the current partition, Given a graph with n vertices, their choice of the starting nodes of each partition of size p can be implemented randomly, by choosing the node with the minimum proximity with respect to the created partitions, by choosing the node with the maximum proximity with respect to the created partitions, and finally, by choosing all the remaining un-partitioned nodes as the starting node (and choosing the best solution), which has the worst complexity of $n \cdot (n-p) \cdot (n-2p) \cdot \ldots \cdot 2p \cdot 1$.

A.3 Iterative Refinement Algorithms

This family of algorithms takes a non-optimal partition and iteratively tries to decrease its cost by exchanging nodes which belong to different partitions, until no improvement can be achieved. The choice of the sequence of the nodes to be exchanged varies. It can be implemented using the preexisting pre-ordering, a random pre-ordering, and a topological pre-ordering. The Kernighan-Lin algorithm [3] is one of the most characteristic techniques that

represent Iterative Refinement Algorithms (IRA). It is often used as a basic means of comparison with new proposed techniques.

In [17], Gilbert et al. develop a parallel algorithm for partitioning the vertices of a graph into $k \geq 2$ sets in such a way that few edges connect vertices in different sets. The algorithm is intended for a message-passing multiprocessor system, such as the hypercube, and is based on the Kernighan-Lin algorithm for finding small edge separators on a single processor. The authors use this parallel partitioning algorithm to find orderings for factoring large sparse symmetric positive definite matrices. These orderings not only reduce fill, but also result in good processor utilization and low communication overhead during the factorization. A complexity analysis of the algorithm, as well as some numerical results from an Intel hypercube and a hypercube simulator, are provided.

In [18], the maximum concurrent flow problem (MCFP) is readily illustrated by problems such as traffic flow in road networks and message transfer in packet switched networks. The MCFP was introduced in [11] and can be solved using either linear programming techniques or using the flow routing algorithms [12, 13]. Application of these techniques has typically been limited to problems of small size (40 vertices or less). The authors present a refinement of the algorithm in [13], which operates successfully on problems three times larger than those solved by the previous techniques. Empirical polynomially bounded storage and time complexity is demonstrated and applications in packet switched networks are investigated.

In [25], Tragoudas et al. consider the general partitioning problem, namely, how to partition the elements of a circuit into sets of size less than a small constant, so that the number of nets which connect elements in different sets is minimized. One application is in the design for testability of VLSI chips and printed circuit boards [23, 24]. The authors consider two different versions of a bottom-up iterative improvement approach. In the first version they present an efficient heuristic. In an alternative version, they use the heuristic as a subroutine to an approximation (provably good) algorithm, to result in comparably good solutions. Both approaches are compared with the familiar top-down approach which uses a well known bisection heuristic [7] as a subroutine. The authors' solutions outperform the top-down partitioning approach.

In [43], Saab addresses the network partitioning problem with the use of node clustering. Node clustering or compaction has been shown to enhance the performance of iterative partitioning algorithms by several authors. However, clustering has been mainly used as a preprocessing step before partitioning in existing methods. Saab describes a technique to extract clusters using information collected during a pass of an iterative exchange algorithm. Alternative approaches for the implementation of this new clustering technique are discussed, and one such approach is chosen to be incorporated in a modified Fiduccia-Mattheyses algorithm based on a tradeoff between run time and performance. The resulting algorithm, BISECT, performs well in comparison with variants of the Kernighan-Lin algorithm including the Fiduccia-Mattheyses algorithm [7], local approaches, and simulated annealing (which will be described later on) on a wide variety of real and randomly generated

benchmarks. BISECT is also used to find small vertex separators, and its results are compared with previous methods on several benchmarks. The empirical results show that BISECT is stable and is not very sensitive to the initial partition. Under suitably mild assumptions, BISECT can be shown to run in linear time. The empirical results confirm the speed of BISECT which can partition very large graphs (12,598 nodes and 91,961 edges) in less than six minutes of CPU time on a Sun Sparc 1+ workstation.

In the context of VLSI circuit partitioning, Dutt and Deng [52] propose new iterative-improvement methods that select cells to move with a view to moving clusters that straddle the two subsets of a partition into one of the subsets. The new algorithms significantly improve partition quality while preserving the advantage of time efficiency. Experimental results on 25 medium to large size ACM/SIGDA benchmark circuits show up to 70% improvement over FM in cutsize, with an average of per-circuit percent improvements of about 25%, and a total cut improvement of about 35%. They also outperform the recent placement-based partitioning tool Paraboli and the spectral partitioner MELO by about 17% and 23%, respectively, with less CPU time. This demonstrates the potential of iterative improvement algorithms in dealing with the increasing complexity of modern VLSI circuitry.

Given n heterogeneous traffic sources which generate multiple types of traffic among themselves, Lim and Lee [54] consider the problem of finding a set of disjoint clusters to cover n traffic sources. The objective is to minimize the total communication cost for the entire system in the context that the intracluster communication is less expensive than the intercluster communication. Different from the general graph partitioning problem, their work takes into account the physical topology constraints of the linear arrangement of physical cells in highway cellular systems and the hexagonal mesh arrangement of physical cells in cellular systems. In their partitioning schemes, the optimal partitioning problem is transformed into an equivalent problem with a relative cost function, which generates the communication cost differences between the intracluster communications and the intercluster communications. For highway cellular systems, the authors have designed an efficient optimal partitioning algorithm of $O(kn^2)$ by dynamic programming, where k is the number of clusters of n base stations. The algorithm also finds all the valid partitions in the same polynomial time, given the size constraint on a cluster and the total allowable communication cost for the entire system. For hexagonal cellular systems, they have developed four heuristics for the optimal partitioning based on the techniques of moving or interchanging boundary nodes between adjacent clusters. The heuristics have been evaluated and compared through experimental testing and analysis.

In [61], the authors focus on the topological pre-ordering concerning the nodes exchanged in their IRA implementation. The first nodes to be exchanged should be the ones with the minimum proximity to the rest nodes of their partition (in case of a tie, the one with the maximum proximity to the rest of the partitions is chosen). The complexity of one iteration is of an $O(n^2)$. The overall complexity of the algorithm depends on the number of iterations needed before the algorithm finishes, which in real cases can be considered constant. (Note that in the results of Ferracioli et al., a maximum number of 10 iterations was sufficient).

Finally, the authors propose a General Partitioning Algorithm, which consists of a combination of the two families of SMA and IRA algorithms. Since in practice it has been shown that the SMA produce sub-optimal results, these results can be given to the IRA to refine them into better partitioning results.

In [70], Begerano et al. propose a Location Area Planning technique which guarantees a bounded quality of the solution in the worst case. Their analysis focus on the "update cost", that is the inter-partition cost and the "paging cost", which is related with the number of cells inside a partition. They form an objective function which is the minimization of the sum of these costs, under some constraints, concerning the maximum number of cells in a partition and the maximum sum of weights in a partition. These constraints may be imposed in practice, from the devices used. The authors prove that the problem is NP-Hard and thus, they construct a polynomial-time approximation algorithm, whose cost is an approximation factor 'a' times the cost of the optimal solution. As a special case, they offer an optimal solution when the graph to be partitioned is a line. Using dynamic programming they prove they offer the optimal solution in $O(V^2)$.

In order to construct their approximation algorithms, they express the problem and the objective function in terms of linear programming. By relaxing the integrality constraints they obtain a linear program which contains $O(V^2)$ variables and $O(V^3)$ constraints. This program returns an optimal fractional solution which is a lower bound (in terms of cost) on the value of the optimal integral solution. The authors then round the fractional solution to a near-optimal integral solution.

The rounding algorithm uses the technique of "region/ball growing". Its main notions are:

The distance metric $d(i, j)$, which is a variable defined in the integer programming formulation of the problem and expresses whether two nodes should be in the same partition, or differently, how close they are.

The ball(i,r), of radius r around node i, which consists of all nodes j whose $d(i, j) \leq r$ and the edges between them, as well as the fraction of the edges (j, k) with only one endpoint (j) in the ball, equal to $(r - d(i, j)) / (d(i, \text{k}) - d(i, j))$.

The volume of a ball $vol(\ b(i, r)\ )$, which is the weighted distance of the edges (or differently, the fraction of the edges) belonging to the ball.

The algorithm iteratively grows balls around arbitrary nodes of the graph. Starting from a $b(i,0)$, it subsequently grows r until it finds a ball such that the weight of the cut defined by the ball is at most $c \cdot \ln(|V|+1)$ times the volume of the ball (i.e, until $\text{cutweight}(\ b(i, r)\ ) \leq c \cdot \ln(|V|+1) \cdot \text{vol}(\ b(i, r)\ )$ holds). The authors prove that the approximation factor of their algorithm has an $O(\log(|V|))$ for general graphs. Specifically, their algorithm guarantees a solution with maximum update cost $c \cdot \ln(|V|+1) \cdot OPT$ and maximum paging cost $c/(c\text{-}2) \cdot OPT$, for maximal region radius $1 / c$ for any given $c > 2$, with OPT being the cost of the optimum solution. Finally, in the case of planar graphs, using the technique of "weak diameter" by, Klein, Plotkin and Rao [34], the region growing algorithm has a constant approximation factor. Nevertheless, the authors note that this constant may be very large.

Additionally, they propose a heuristic similar to the family of IRA described in [61], which tries to improve the cost of the algorithm by iteratively

exchanging nodes between partitions. The authors compare their algorithm to two other greedy algorithms of the literature which also try to minimize the update and paging cost under some constraints. They do not compare their results with other algorithms, such as simulated annealing because they do not have polynomial run time and the quality of the result depends on the duration of the optimization.

In contrast with the work of Ferracioli et al, the authors give a guarantee of the quality of their algorithm, while providing a polynomial run-time algorithm. However, their work is intuitively quite similar since both algorithms construct a partition by selecting starting nodes and then adding nodes to the partition until a certain criterion is met. The work of Ferracioli et al, proposes a better scheme for the selection of the starting nodes, comparing to the random choice of the region growing technique. They both then propose the iterative heuristic technique of exchanging nodes between partitions in order to improve the cost of an existing partitioning. Ferracioli et al, propose a specific sequence of nodes to be exchanged in order to achieve a better result before the algorithm finishes, trapped in a local minimum, whereas the latter work leave this choice open.

In [88], Meals proposes a novel fast technique for partitioning a system into hardware and software components and scheduling the resulting components for execution. The technique maps hierarchical task graphs onto a heterogeneous architecture containing a single sequential processing engine for executing software components and programmable logic for implementing hardware components. The technique uses an iterative improvement algorithm to evaluate placement of components in the hardware and software partitions. It also uses a decomposition phase to alter the number and granularity of tasks evaluated. By doing so, the algorithm can perform early evaluations on fewer, coarse grained tasks and only evaluate smaller granularities as needed to achieve the desired execution time of the task system. The suitability of this technique in finding a solution which meets a given timing constraint on the system is evaluated by comparing it to a iterative improvement algorithm on a typical non-hierarchical task graph where the task granularities are fixed. To show robustness of the approach, the comparison is done for a large number of synthetic task graphs and the results are normalized so that statistical analysis can be performed on the large sample set.

A.4 Simulated Annealing

As described by S. Kirkpatrick et al. in [8], the analysis of the movement of atoms of matter at given temperatures and the search of this system's ground state can be corresponded to combinatorial optimization problems. The initial energy of the system at a high temperature can be corresponded to the initial state of a combinatorial problem having a certain cost. In the simulation of the displacement of atoms under a certain temperature, at each step, an atom's displacement results in a change of the system's energy DE. If the energy is reduced, the new system configuration is accepted. However in the case of $DE > 0$ the new configuration is accepted with probability $P(DE) = \exp(-DE/(k \cdot T))$, where k is the Boltzmann's constant and T the temperature. This can be implemented by taking a random number between 0 and 1 from a uniform distribution and if it's less than $P(DE)$ the new configuration is accepted. Using

the cost function in place of the energy and defining configurations by a set of parameters, the Metropolis procedure generates a population of configurations of a given optimization problem at some effective temperature (Metropolis et al. in [1], present an algorithm for the approximate numerical simulation of the behavior of many-body systems at a finite temperature). In the experiments of such systems, the system must be first melted and then carefully annealed by slowly quenching it until the system freezes, otherwise a rapid freezing of the system results in a highly metastable final state. Respectively, the simulation annealing procedure first "melts" the system being optimized at a high effective temperature and then the temperature is lowered by slow stages until no changes occur (the system "freezes"). The sequence of temperatures and the number of rearrangements attempted to reach equilibrium at each temperature are considered an annealing schedule.

The advantage of Simulated Annealing (SA) is that it does not necessarily stick in local minima, since it accepts worse solutions and transitions out of a local minimum are possible as long as $T > 0$. Such transitions are more possible at high temperatures and less possible at low temperatures. The latter fact gives an adaptive divide and conquer characteristic of this technique, since at higher temperatures, more gross features are optimized, whereas in lower temperatures, more microscopic details are optimized. In order to use simulated annealing to solve a combinatorial optimization problem, it is important to have a concise description of a configuration of the system, a random generator of "moves" /rearrangements of the elements in a configuration, a quantitative objective function containing the trade-offs that have to be made (such as those between paging and update cost, for the NP problem), and an annealing schedule of the temperatures and length of times for which the system is to be evolved.

Clearly, this technique is quite different from the aforementioned heuristic techniques. As Begerano et al, have pointed out, the quality of the result depends on the run time of the optimization procedure. Once again the notion of exchanges/rearrangements, modeled through a theory of atom displacements at a given temperature, is presented and used in order to search possibly better solutions of the optimization problem. The main difference and strength of the algorithm is the acceptance of worse solutions as well, more frequently at early steps of the procedure (i.e., at high temperatures of the simulation), which allows the escape from local minima through the way. On the other hand, there is no run-time guarantee and a time-quality trade-off governs the algorithm.

Savage et al. in [22], show that the Kernighan-Lin and simulated annealing heuristics are logspace complete for the classes P and BPP, respectively, which means that they are hard to parallelize. They also describe a new parallel heuristic that on the 32K-processor CM-2 Connection Machine handles graphs with more than two million edges and gives in nine minutes partitions that are within 2 best ever found.

In [31], the authors introduce a meta-heuristic to combine simulated annealing with local search methods for CO problems. This new class of Markov chains leads to significantly more powerful optimization methods that wither simulated annealing or local search. The main idea is to embed deterministic local search techniques into simulated annealing so that the chain explores only local optima. It makes large, global changes, even at low temperatures, thus overcoming large barriers in configuration space. The

authors have tested this meta-heuristic for the traveling salesman and graph partitioning problems. Tests on instances from public libraries and random ensembles quantify the power of the method. The proposed algorithm is able to solve large instances to optimality, improving upon state of the art local search methods very significantly. For the traveling sales man problem with randomly distributed cities in a square, the procedure improves on 3-opt by 1.6% and on Lin-Kernighan local search by 1.3%. For the partitioning of sparse random graphs of average degree equal to 5, the improvement over Kernighan-Lin local searches 8.9%. In [35], the authors address the problem of partitioning an unstructured mesh of memory parallel processors so that the load balance is maximized and inter-processor communication time is minimized. This can be approximated by the problem of partitioning a graph so as to obtain a minimum cut, a well-studied combinatorial optimization problem. They once again show that a general procedure enables one to combine simulating annealing with Kerninghan-Lin. The resulting algorithm is both very fast and extremely effective.

In the same context, Boukerche et al. [36] consider the problem of partitioning a conservative parallel simulation for execution on a multi-computer, assuming that the synchronization protocol makes use of null messages [6]. They also propose the use of a simulated annealing algorithm with an adaptive search schedule to find good (sub-optimal) partitions. They discuss the algorithm, its implementation and report on the performance results of simulations of a partitioned FCFS queueing network model, executed on iPSC/860 hypercube. The results obtained are compared with a random partitioning. They show that a partitioning which makes use of their simulated annealing algorithm results in a reduction of 25-35% of the running time of the simulations when compared to the running time of a random partition of the model.

In [67], Ricci and Alfeld explore the network testbed mapping problem. Network experiments of many types, especially emulation, require the ability to map virtual resources requested by an experimenter onto available physical resources. These resources include hosts, routers, switches, and the links that connect them. Experimenter requests, such as nodes with special hardware or software, must be satisfied, and bottleneck links and other scarce resources in the physical topology should be conserved when physical resources are shared. In the face of these constraints, this mapping becomes an NP-hard problem. Yet, in order to prevent mapping time from becoming a serious hindrance to experimentation, this process cannot consume an excessive amount of time. The authors describe the interesting challenges that characterize this problem, and explore its applications to emulation and other spaces, such as distributed simulation. They present the design, implementation, and evaluation of a solver for this problem, which is in production use on the Netbed shared network testbed. This solver builds on simulated annealing to find very good solutions in a few seconds for the authors' historical workload, and scales gracefully on large well-connected synthetic topologies.

Yet another work where simulated annealing is used in order to solve the network partitioning problem in the context of Location Area planning (cell-LA assignment) and cell to switch assignment in cellular networks, is analyzed in [71]. The main difference with the previously described techniques is that

instead of a multi-objective approach or including both the paging and registration cost in the objective function, they focus on reducing the registration cost in the objective function whereas the paging cost is included in the problem formulation as a constraint. The reason lies in the fact that the formulation of a linear objective function which incorporates both costs would include assumptions concerning relative values between the two types of costs, which would be constant throughout the network, although this varies from cell to cell. Moreover, these costs do not have compatible units since different control channels are used for paging and registration. Thus, they form a number of constraints which concern the number of BS (Base stations/cells) in a BSC (Base Station Controller) the number of BSC in a MSC (Mobile Switching Center) the number of BSs in an LA and the relations between BSs, concerning the BSCs, LAs and MSCs which they belong to.

The annealing schedule includes moves from an initial partition solution at an initial temperature, to neighboring solutions. Since both cell-LA and cell-switch assignment is performed, a move may consist of changing a BS-BSC assignment, changing a BSC-MSC assignment or changing a BS-LA assignment. If the resulting solution, *NewSolution* is feasible concerning the formulated constraints a move is considered to have been done and if it results in a partition with cost *Cost*(*NewSolution*) which is better than the current solution, *CurrentSolution*, it is accepted, otherwise it is accepted with probability $e^{(-|Cost(NewSolution) - Cost(CurrentSolution)| / T)}$. After accepting a certain number of new solutions the temperature parameter is decreased. The simulation stops after a certain number of steps where no improvement is observed, or if *T* falls under a certain temperature.

In order to be able to evaluate the results of SA, the authors consider two algorithms to compare with. A greedy algorithm, which typically accepts only better solutions (or differently T=0) and a heuristic algorithm. The heuristic consists of sorting of cell pairs by hand-over rate. Then, starting from cell pairs with higher handover rate, it assigns unassigned cells to a feasible BSC and subsequently to a feasible LA in that BSC. Finally, any remaining disconnected cells are added to feasible BSCs and LAs. In order to assure that the comparison results will not be site specific, the authors compared the three algorithms over real data traces of different sites as well as over randomly generated network data, representing high or low mobility scenarios under the assumption of exponentially distributed load.

The experiments on the real network data showed that simulation annealing greatly outperformed the most close competitor, the greedy algorithm, by producing results of 50% lower costs. A number of randomly generated partitions were compared as well to give a wider sense of the solution space and corresponding quality. As far as the random data sets are concerned, the SA produced better results, by 11% in the case of load generated by a certain distribution, which led to neighboring solutions of similar cost. The performance was compared in terms of average costs, however the authors show that SA solutions have a tight range of 15% divergence from the average.

A.5 Tabu Search

This technique introduced by Glover and Laguna in [33], incorporates various structures and strategies in order to escape from local optima while

searching for a solution. The main concepts of the technique are: *a)* the move, which is an operation which takes us from a current solution to a neighboring solution. *b)* The memory structures, which include information across four dimensions, recency, frequency, quality and influence of a move. Influence reflects the impact of a choice, quality involves considerations beyond those reflected solely by the value of an objective function and recency and frequency participate in characterizing a move as "Tabu". Moves which are characterized as "Tabu" cannot be done or undone for a certain period called a Tabu tenure. For example, in the NP problem an exchange of nodes between partitions may not be undone for the subsequent "k" moves, or differently, the nodes which participated in an exchange may not participate in another move during their Tabu tenure. Another way of expressing a move as "Tabu active" is to refer to additions and removals of edges to the inter-partition frontier; In this sense a removed edge - or one of its nodes - could not be added again until its Tabu tenure expires, as well as, an added edge could not be removed during its Tabu tenure, where the aforementioned two tenures do not have to be of same length.

The above description of the permitted moves at a certain stages shows that recency-based memory modifies the neighborhood $N(x)$ of the current solution x, to a $N^*(x)$ formed by the Tabu restrictions upon the moves. Here come the concepts of intensification and diversification strategies with which TS attempts to find better solutions.

Intensification strategies focus on examining neighbors of solutions which have been found so far to be the best (elite solutions). In order to successfully do this, these strategies modify choice rules For example, in the NP problem, although an edge might include a "Tabu active" node, this restriction could be overridden in order to visit an unvisited solution. Such modifications are based on strategies called aspiration criteria, which make sure that good neighboring solutions will be visited (Here is where the quality and influence - based memory is used).

On the other hand, diversification strategies examine unvisited regions and generate solutions that differ in various significant ways from those seen before. They're employed during restarting procedures: if the rate of finding new best solutions falls below a threshold, the method is restarted by a process designated to generate a new sequence of solutions. A diversification strategy is often based on critical events, which could be all previous starting solutions, as well as, all previous local optima (solutions whose objective function value is better from its preceding and subsequent ones); a new restarting point should be far from all the aforementioned solutions in order to explore new solution regions. With the help of frequency based memory, (some of) the elements of these solutions could be excluded from the new starting points For example, in the NP problem the most frequent edges/nodes to belong to the inter-partition frontier of previous starting points and local optima visited, would be excluded from a new starting point.

The Tabu search (TS) technique returns multiple local optimal solutions, rather than being trapped to one based solely on the objective function value as a quality metric. While simulated annealing accepts worse solutions probabilistically, the Tabu search avoids exhaustive searches by employing strategies which alter the possible moves that can be made, marking as tabu, elements which are removed (probably considered low-quality) or added (which might have proved to lead to good solutions) to the solution, or provide new

diversified starting points in order to explore different solutions, by excluding elements that were frequent in past solutions and starting points. The notion of solution "quality" is liberated and can rely on various objectives and on the selective history of the search; the notion of move is also liberated and for example it can express a rearrangement as described in simulation annealing.

The technique is obviously not a polynomial time algorithm since it returns some local optima values and many re-runs may be needed in order to create a frontier of "best" solutions which may refer to more than one objectives In the case of one objective each restart returns a best solution which might be accepted or not, based on the current best.

The relationship between prohibition-based diversification (Tabu Search) and the variable-depth Kernighan-Lin algorithm is discussed in [60]. A greedy construction scheme with an appropriate tie-breaking rule (Min-Max-Greedy) produces initial assignments in a very fast time. For some classes of graphs, independent repetitions of Min-Max-Greedy are sufficient to reproduce solutions found by more complex techniques. When the method is not competitive, the initial assignments are used as starting points for a prohibition-based scheme, where the prohibition is chosen in a randomized and reactive way, with a bias towards more successful choices in the previous part of the run. Detailed experimental results are presented on benchmark suites used in the previous literature, consisting of graphs derived from parametric models (random graphs, geometric graphs, etc.) and of real-world graphs of large size. On the first series of graphs, a better performance for equivalent or smaller computing times is obtained, while, on the large real-world instances, significantly better results than those of multilevel algorithms are obtained, but for a much larger computational effort.

Saab [75] introduces a new approach to partition graphs and hypergraphs. This new approach combines local and global sampling, clustering, and Tabu Search in a multilevel partitioning algorithm (TPART). TPART was implemented in a C program and compared to many state-of-the-art partitioning algorithms using a wide variety of benchmarks. TPART consistently performs well on the various benchmarks used and in comparison with other partitioning algorithms. TPART has a reasonably fast running time and it can produce a high quality partition of a graph of 262,144 nodes and 524,286 edges in less than 2 minutes CPU times on a Compaq Alpha DS20E 67/667 MHZ machine with 1GB of main memory.

In [80] Aringhieri et al. consider two problems that arise in the design of optical telecommunication networks when a ring-based topology is adopted, namely the SONET Ring Assignment Problem and the Intraring Synchronous Optical Network Design Problem. The authors show that these two network topology problems correspond to graph partitioning problems with capacity constraints: the first is a vertex partitioning problem, while the latter is an edge partitioning problem. Solution methods are considered for both problems, based on metaheuristic algorithms. The authors first describe variable objective functions that depend on the transition from one solution to a neighboring one, then they apply several diversification and intensification techniques including Path Relinking, eXploring Tabu Search and Scatter Search. Finally the authors propose a diversification method based on the use of multiple neighborhoods. A

set of extensive computational results is used to compare the behaviour of the proposed methods and objective functions.

In [84], Chamberland and Pierre deal with the problem of how to design cellular networks in a cost-effective way. They first propose an optimization model that deals with selecting the location of the base station controllers (BSCs) and mobile service switching centers (MSCs), selecting their types, designing the network topology and selecting the link types. In order to find a "good" solution, The authors propose a tabu search algorithm. Numerical results show that the tabu search algorithm produces solutions close to a proposed lower bound.

A.6 Evolutionary Search

This technique belongs to the category of **genetic algorithms**, since as A.J. Soper, C. Walshaw and M. Cross elaborate in [79], it is a stochastic search technique that generates new points in a search space using information from a finite population of already evaluated points. This is achieved by choosing the fittest individuals of the population, in terms of metrics concerning solution quality (e.g., in the case of network partitioning, the partition quality), and applying on them the operations of crossover and mutation. The first operation allows for a transition of previous parent generations' predominant characteristics to the new generation and the mutation function allows the further diversification among individuals.

Maini et al. [37] present a genetic algorithm for the problem of partitioning in parallel computing. They propose new crossover operators (KNUX, DKNUX) that lead to orders of magnitude improvement over traditional genetic operators in solution quality and speed. Their method can improve on good solutions previously obtained by using other algorithms or graph theoretic heuristics in minimizing the total communication cost or the worst case cost of communication for a single processor. They also extend their algorithm to Incremental Graph Partitioning problems, in which the graph structure or system properties changes with time.

In [50], hybrid genetic algorithms (GAs) for the graph partitioning problem are described. The algorithms include a fast local improvement heuristic. One of the novel features of these algorithms is the schema preprocessing phase that improves GAs' space searching capability, which in turn improves the performance of GAs. Experimental tests on graph problems with published solutions showed that the new genetic algorithms performed comparable to or better than the multi-start Kernighan-Lin algorithm and the simulated annealing algorithm. Analyses of some special classes of graphs are also provided showing the usefulness of schema preprocessing and supporting the experimental results.

In [58], Shazely et al. present new techniques for discovering more than one solution to the partitioning problem using genetic algorithms. The techniques used are based upon applying niching methods to obtain multiple good solutions instead of only one solution. They also present in detail a comparison between the results of a traditional method; Simple Genetic Algorithm (SGA), and two

niching methods; fitness sharing and deterministic crowding when applied to the graph partitioning problem.

Hwoang et al. [91] propose a new gene reordering scheme for the graph bisection problem: Gene reordering starts with two or more vertices to capture the clustering structure of graphs effectively. The authors devised a chromosome repairing method for hybrid genetic search, which helps exploit clusters when combined with gene reordering. Experimental tests showed that the suggested reordering scheme significantly improves the performance of genetic algorithms over previous reordering methods.

James et al. [97] consider the registration area planning (RAP) problem. This problem examines the grouping of cells comprising a personal communication services (PCS) network into contiguous blocks in an effort to reduce the cost of managing the location of the devices operating on the network, in terms of bandwidth. Their study introduces a hybridized grouping genetic algorithm (HGGA) to obtain cell formations for the RAP problem. The hybridization is accomplished by adding a tabu search-based improvement operator to a traditional grouping genetic algorithm (GGA). Results indicate that significant performance gains can be realized by hybridizing the algorithm, especially for larger problem instances. The HGGA is shown to consistently outperform the traditional GGA on problems of size greater than 19 cells.

In the context of the network partitioning problem, Soper, Walshaw and Cross combine a multilevel graph partitioner with an evolutionary algorithm in [79]. The multilevel partitioning is performed by creating the maximal (not necessarily optimal) set of disjoint edges of the graph and then collapsing each one of these edges. A new node is created, whose weight is the sum of the weights of the edge's nodes. A new graph is formed (next level graph) and this procedure is repeated until the number of nodes of the resulting graph equals the number of desired partitions. "Heaviest" edges are collapsed first. Implicitly, an initial partition (generation) has been constructed, since a final node represents two collapsed nodes of the previous level, and so on. In second phase, rearrangements are performed between the nodes of each partition in order to improve the partition solution. The nodes are grouped by gain (improvement in the cut-weight) using bucket sort and those with highest gains are first chosen. There is a limited ability of escaping from local minima by making grouped rearrangements which improve the solution, since they can include rearrangements of nodes with negative gains. As is common in such procedures, rearrangements stop when there's no further improvement. This procedure is performed for every level of the graph, starting from the one where the number of nodes equals the number of partitions, up to the initial graph level.

The evolutionary part of the algorithm lies in the fact that the nodes' weights are subject to some added biases, which change from generation to generation of solutions, using the crossover and mutation operators. Specifically, during the rearrangement phase, gains are calculated using the biased edge weights. An edge's weight is the sum of the attached nodes' biases plus a dependent weight of unity. Moreover, before the creation of a new generation/solution, the biases of the nodes are assigned by performing either the crossover or the mutation operators (in some analogy) to the nodes of the previous generation. The crossover operator is applied to a number of previous parent generations. It assigns a uniformly random bias from the range [0, 0.01] which is added to the

node's weight, plus a bias value of 0.1 if this node was not a border node (i.e., attached to a cut-edge). The mutation function adds a bias from the range [0, 0.01] to the node's weight, plus a bias value of 2.0 if the node was not a border node or a neighbor of a border node or a neighbor of a neighbor of a border node, in a number of previous parent generations (two or more). Note that these values are in a scale assuming a unity value for the weight of an edge.

The authors used the commercial multilevel partitioner JOSTLE. As they acknowledge, the algorithm has a considerably large run-time and is suitable in planning partitions which are to be permanent and whose quality is very important. Moreover, contrary to SA and TS, evolutionary search does not incorporate any particular technique in order to avoid local optima and this can be partially attempted with the acceptance of grouped rearrangements of nodes which give a better solution, including individual rearrangements with negative gains. The quality of the result lies in the mechanism of the algorithm which makes more likely that nodes with low biases will remain border nodes or near the borders. Thus, nodes which contribute positively in a low cost partition will be candidates to be border nodes.

A.7 Spectral Clustering

In the spectral method, the vertices in a graph can be mapped into the vectors in *d*-dimensional space, thus the vectors are partitioned instead of vertices to obtain graph partitioning. Spectral bisection is a heuristic technique for finding a minimum cut graph bisection. To use this method the second eigenvector of the Laplacian of the graph is computed and from it a bisection is obtained. The most common method is to use the median of the components of the second eigenvector to induce a bisection.

Incremental graph partitioning is also examined by Ou and Ranka in [38]. Its application concerns a large class of irregular and adaptive data parallel applications (such as adaptive meshes), where the computational structure changes from one phase to another in an incremental fashion. In incremental graph-partitioning problems the partitioning of the graph needs to be updated as the graph changes over time; a small number of nodes or edges may be added or deleted at any given instant. The authors use a linear programming-based method to solve the incremental graph partitioning problem. All the steps used by their method are inherently parallel and hence their approach can be easily parallelized. By using an initial solution for the graph partitions derived from recursive spectral bisection-based methods, their methods can achieve repartitioning at considerably lower cost than can be obtained by applying recursive spectral bisection from scratch. Further, the quality of the partitioning achieved is comparable to that achieved by applying recursive spectral bisection to the incremental graphs from scratch.

Hendrickson and Leland in [44] present a multilevel algorithm for graph partitioning in which the graph is approximated by a sequence of increasingly smaller graphs. The smallest graph is then partitioned using a spectral method, and this partition is propagated back through the hierarchy of graphs. A variant of the Kernighan-Lin algorithm is applied periodically to refine the partition. The entire algorithm can be implemented to execute in time proportional to the

size of the original graph. Experiments indicate that, relative to other advanced methods, the multilevel algorithm produces high quality partitions at low cost.

In [55], the authors show a method to obtain optimal two-way vector partitioning based on an optimal direction vector. As the problem to find the optimal direction vector is NP-problem, we propose an efficient heuristic to obtain high quality direction vector. As we approximate a given netlist into the graph and only use ten eigenvectors in practice, there is a chance to improve the solution quality by local optimization. Fiduccia-Mattheyses algorithm is employed as a post processing. Compared with FM and MELO, the proposed algorithm PDV reduces cutsize on the average 40% and 20.5%, respectively.

In the same context, recursive spectral bisection (RSB) is a heuristic technique for finding a minimum cut graph bisection. As it was mentioned before, the most common method is to use the median of the components of the second eigenvector to induce a bisection. In [56], Chan et al. prove that this median cut method is optimal in the sense that the partition vector induced by it is the closest partition vector, in any l_s norm, for $s \geq 1$, to the second eigenvector. Moreover, they prove that the same result also holds for any m-partition, that is, a partition into m and $(n\text{-}m)$ vertices, when using the m-th largest or smallest components of the second eigenvector.

A.8 Heuristic Algorithms and Case Studies

Apart from the techniques described above, there's an extensive work in the literature concerning heuristic algorithms for solving the partitioning problem, which are adjusted to the context of the problem and they often incorporate and combine various techniques. Moreover, there exist much theoretic work which analyzes the potential of the described techniques and other heuristics by proving bounds on their run-time performance and result quality with respect to the optimal solution. On the other hand, many practical comparisons between different techniques have been performed and results are presented towards the discovery of the most profitable technique depending on the problem context. Here follows the related work in chronological order.

In [2], a program is modeled as a directed graph on n nodes, where the nodes are instructions, or data items, or contiguous groups of these. The problem of partitioning such sets of nodes into pages to minimize the number of transitions between pages during execution of the program, is discussed. The nodes are assumed to have a given ordering which may not be changed. The author requires that nodes on any page must be contiguous, so the only degree of freedom is in selecting "break points" between the pages. It is shown that if the expected number of transitions between each node of the program graph and its successors is known, then there is an algorithm for selecting the optima break points. The algorithm requires an execution time which grows linearly with the number of nodes in almost all cases.

From the context of dataflow multiprocessors, the algorithm by which the nodes of a program graph are allocated for execution to its processors is considered in [14]. In the case of the static type of architecture one must consider pipelining as well as spatial concurrency. The authors use a graph

partitioning scheme to aid in the design of such algorithms and selection of the associated interconnection topology. The scheme first refines the usual dataflow program graph and then partitions it into "trees." Their study shows that the hypercube interconnection accommodates these trees in a way that allows them to develop an algorithm that places producer nodes (of a dataflow graph) nears their consumers to keep the message path very short. The algorithm achieves concurrency in simultaneous execution of subgraphs that involve parallel operations, e.g., array operations, and subgraphs that are highly iterative or recursive. An analytical model is given to evaluate the performance of the algorithm. In [15], a cost-effective scheme for partitioning large data flow graphs is presented. Standard data flow machine architectures are assumed. The objective is to reduce the overhead due to token transfers through the communication network of the machine. When this scheme is employed on large graphs, the load distribution on the rings of the data flow machine is also improved. A canonical form of a data flow graph is introduced to establish the relationship between the communication overhead and the size reduction of the partition cut-set. General lower estimates on the overhead are derived in terms of processing and transmission delay parameters of the machine. The method uses heuristics and an evaluation function to guide the partition algorithm. Some implications of the proposed method on the organization of the data flow machines are discussed.

In [20], Patil et al. address logic simulation on parallel machines by exploiting the concurrency in the circuit being simulated (called data parallelism) as opposed to exploiting parallelism inherent in the simulation algorithm itself (called functional parallelism). The most crucial step in obtaining the maximum parallelism using data parallelism is the partitioning of circuit elements. They introduce a cost function which tries to model the simulation of a logic circuit in a parallel environment. The cost function tries to estimate the parallel run time for logic simulation given the processor assignment and the underlying multiprocessor architecture. The authors present different heuristic algorithms to partition the circuit and evaluate the efficiency of these algorithms using the proposed cost function. Partitioning algorithms for both event-driven and compiled code simulation are given.

In [21], a heuristic graph partitioning scheme is presented to determine effective node separators for undirected graphs. An initial separator is first obtained from the minimum degree ordering, an algorithm designed originally to produce fill-reducing orderings for sparse matrices. The separator is then improved by an iterative strategy based on some known results from bipartite graph matching. This gives an overall practical scheme in partitioning graphs. Experimental results are provided to demonstrate the effectiveness of this heuristic algorithm on graphs arising from sparse matrix applications.

In [26], a technique is described that performs the mapping of tasks to processors in parallel. The technique first partitions a task graph into subgraphs and then applies mapping algorithms on the individual subgraphs in parallel. The assignments of all subgraphs are combined in the final phase to form the assignment for the entire task graph. This technique has a number of important advantages including (1) enabling the use of more expensive algorithms on the subgraphs, (2) allowing different mapping techniques to be applied to different parts of the task graph, (3) reducing the space requirements during mapping and

(4) accommodating modifications to a program without remapping of the entire task graph. Simulation results demonstrate that parallelism can be found in the mapping process and the scheduling performance of this technique using scheduling heuristics is either close or better than when the same mapping algorithm is applied to the entire graph.

The problem of dataflow graph partitioning is examined by Silc et al., in [28]. Such partitioning aims to improve the efficiency of macro-dataflow computing on a hybrid control/data driven architecture. The partitioning consists of dataflow graph synchronization and scheduling of the synchronous graph. A new scheduling algorithm, called Global Arc Minimization (GAM), is introduced. The performance of the GAM algorithm is evaluated relative to some other known heuristic methods for static scheduling. When interprocessor communication delays are taken into account, the GAM algorithm achieves better performance on the simulated hybrid architecture.

In [29], Nandy et al. develop an implementation of a parallel partitioning algorithm which is suitable for use in a conservatively synchronized Parallel Discrete Event Simulation (PDES) environment. Effective partitioning is essential for performance and capacity consideration, for any PDES problem. The performance of the partitioning algorithm is very important, to the overall simulation performance. There are two possible approaches to improve performance for the partitioning step: algorithm modifications; and parallelize the partitioning algorithm [7] is developed. The basic algorithm has been modified, first for parallel execution with a similar quality of final partition; and then further modified to increase the parallelism of the algorithm, at the expense of partition quality.

The behavior of random graphs with respect to graph partitioning is considered in [30]. Heath et al. identify the conditions under which random graphs cannot be partitioned well, i.e., a random partition is likely to be almost as good as an optimal partition.

Recursive bisection cannot produce optimal results even with the introduction of tolerance. Simon and Hua [40] prove that if the cost of an optimal (p/e)-way partition of graph G is C, then the modified recursive partitioning that uses an optimal (1/2 + 1/s)-bisection algorithm finds a (1+e, p)-way partition of cost at most O(C·log$p$) (A (b, p)-way partition decomposes G into disjoint $G_1...G_b$ such that $|G_i| \leq b \cdot |G|/p$ for all $1 \leq i \leq p$). The modified recursive partitioning scheme that uses a (1/2 + 1/s)-bisection algorithm, with an approximation factor a, finds a (1+e, p)-way partition of cost at most O(a·C·log$p$). Therefore, there exists a polynomial time algorithm that finds a (1+e, p)-way partition of cost O(C·log$V$·log$p$). These results are quite similar to those of Y. Begerano et al, whose heuristic algorithm had a O(log|V|) approximation factor for general graphs and a large constant approximation factor for planar graphs.

Another work on parallelization speed-up via graph partitioning is that of Buch et al. in [41]. They present a method for graph partitioning that is suitable for parallel implementation and scales well with the number of processors and the problem size. Their algorithm uses hierarchical partitioning. It exploits the parallel resources to minimize the dependence on the starting point with multiple starts at the higher levels of the hierarchy. These decrease at the lower

levels as it zeroes in on the final partitioning. This is followed by a last-gasp phase that randomly collapses partitions and repartitions to further improve the quality of the final solution. Each individual 2-way partitioning step can be performed by any standard partitioning algorithm. Results are presented on a set of benchmarks representing connectivity graphs of device and circuit simulation problems.

In the context of hardware design, distributing computation among multiple processors is one approach to reducing simulation time for large VLSI circuit designs. However, parallel simulation introduces the problem of how to partition the logic gates and system behaviors of the circuit among the available processors in order to obtain maximum speedup. A complicating factor that is often ignored is the effect of the time-synchronization protocol. Inherent in the partitioning problem is the question of how to effectively measure the relative quality of a partition. In [42], Kapp et al. describe an objective cost function for measuring the relative quality of a task partition that includes a synchronization factor for a conservative NULL-message protocol. A graph-based partitioning tool based on this cost function is used to perform the static task allocation for parallel simulation of a structural VHDL circuit. Results for two 1000 – 4000 gate circuits demonstrate that the additional consideration of the synchronization protocol in the cost function generates partitions that exhibit improved speedup.

In [45], Berry et al., present a new heuristic for graph partitioning called Path Optimization PO, and the results of an extensive set of empirical comparisons of the new algorithm with two very well known algorithms for partitioning: The Kernighan-Lin algorithm and simulated annealing. The experiments are described in detail, and the results are presented in such a way as to reveal performance trends based on several variables. Sufficient trials are run to obtain 99% confidence intervals small enough to lead to a statistical ranking of the implementations for various circumstances. The results for geometric graphs, which have become a frequently-used benchmark in the evaluation of partitioning algorithms, show that PO holds an advantage over the others. In addition to the main test suite described above, comparisons of PO to more recent partitioning approaches are also given. The authors present the results of comparisons of PO with a parallelized implementation of Goemans' and Williamson's 0.878 approximation algorithm, a flow-based heuristic due to Lang and Rao, and the multilevel algorithm of Hendrickson and Leland [44].

In [46], Karypis and Kumar present a theoretical analysis that could explain the ability of multilevel algorithms to produce good partitions. They show under certain reasonable assumptions that even if no refinement is used in the uncoarsening phase, a good bisection of the coarser graph is worse than a good bisection of the finer graph by at most a small factor. They also show that for planar graphs, the size of a good vertex-separator of the coarse graph projected to the finer graph (without performing refinement in the uncoarsening phase) is higher than the size of a good vertex-separator of the finer graph by at most a small factor.

A pioneer approach is presented in [48]. The authors propose a learning-automaton based solution to the partition problem. They compare this new solution to various reported schemes such as the Kernighan-Lin's algorithm, and

two heuristic methods proposed by Rolland et al., an extended local search algorithm and a genetic algorithm. The automaton-based algorithm outperforms all the other schemes. This solution can also be adapted for the GPP in which the edge costs are not constant but random variables whose distributions are unknown.

Graph partitioning is also applied in the very diverse context of video signal processors. Aarts et al. [49] consider the problem of partitioning video algorithms over an arbitrary network of high-performance video signal processors. The partitioning problem under consideration is very hard due to the many constraints that need to be satisfied. They present a solution strategy based on a recursive bipartitioning approach, which effectively handles the routing of the data flows through the network under time and resource constraints. The bipartitions are generated using a variable-depth search algorithm. The authors present results for industrially relevant video algorithms.

In [51], Karypis and Kumar present a parallel formulation of a multilevel k-way graph partitioning algorithm. The multilevel *k*-way partitioning algorithm reduces the size of the graph by collapsing vertices and edges (coarsening phase), finds a *k*-way partition of the smaller graph, and then it constructs a *k*-way partition for the original graph by projecting and refining the partition to successively finer graphs (uncoarsening phase). A key feature of their parallel formulation is that it utilizes graph coloring to effectively parallelize both the coarsenin and the refinement during the uncoarsening phase. Their algorithm is able to achieve a high degree of concurrency, while maintaining the high quality partitions produced by the serial algorithm. The authors test their scheme on a large number of graphs from finite element methods, and transportation domains. Their parallel formulation on Cray T3D, produces high quality 128-way partitions on 128 processors in a little over two seconds, for graphs with a million vertices. Thus, their parallel algorithm makes it possible to perform dynamic graph partition in adaptive computations without compromising quality.

The algorithm whose performance characteristics are analyzed in this work is described in [53] by Tollis. A graph theory approach is used to solve the network partitioning problem in hexagonal grids. The partitioning problem is transferred to the dual graph where each node corresponds to an internal face of the initial graph and there's an edge between any two neighboring faces; some additional "border" nodes are added to the external face (one for each edge of the initial graph, which separates an internal face with the external face) along with their corresponding edges. Initial and resulting graphs are planar.

A 2-way partition scheme is first analyzed. The basic idea of the algorithm proposed is the search for a minimum path between two border nodes, which divides the graph into two equally sized subgraphs. The author notes that there can be a tolerance parameter *t*, which will allow partitions of size between $V/2$-t and $V/2$+t, resulting in better bipartitions. The procedure of creating all such shortest paths and choosing the best among them has a cost of $O(V^{3/2} \cdot \log V)$.

The quality of the partition can be improved by altering the algorithm to search for a minimum path from any internal node to any border node and then search, for all intermediate nodes, the minimum path from a border node to another border node, via the internal node, which produces acceptable bisections. The quality was found to be improved by 15% for a run-time cost of

$O(V^3)$. The author proposes a technique which can improve this runtime by defining an "*X*" region from which intermediate nodes will be chosen. This resulted in partitions of same quality, whereas the runtime was about the one of the initial algorithm.

The extension of this algorithm to *k*-way partitioning of a graph may require bisections with explicit size ratios. For example, the trisection scheme which is proposed is a bisection with a $2/3|V| : 1/3|V|$ analogy in the resulting subgraph size and then a bisection at the $2/3|V|$-sized subgraph. A critical point is that the best $2/3|V| : 1/3|V|$ bisection does not result in the best trisection and thus, a lower quality solution has to be chosen. The proposed strategy of checking each possible such bisection which result to the best trisection, is time consuming, but yields good results. Generally *k*-way partitioning for odd *k* values is implemented by bisecting the graph to two sections with a size analogy

$$\frac{k/2}{k}|V| \quad : \quad \frac{(k/2)+1}{k}|V|$$

The author poses two questions concerning the partitioning problem and whether the algorithm can solve it:

-What is an acceptable partition?
-Is there always a path that gives such a partition?

The author answers the first condition with the tolerance parameter *t*. However, for the second question the author claims that there's a connection between the degree of nodes of the initial graph, the min and max traffic costs and the tolerance, in a way that the correct choice of tolerance guarantees that there will always be such a shortest path to provide an acceptable partition. The exact relation between the pre-described elements is not presented in the paper.

Another unanswered question is related to the "*X*" region through which one have to prove that every such path must go in order to allow the algorithm to run with a lower run time (by reducing the search space).

Finally, a question which comes up is whether this recursive bisection scheme has the ability to produce optimal *k*-way solutions. As Horst D. Simon and Shang-Hua Teng prove in [40], ideal recursive bisections have worst case approximation ratio of $T((|V|/k)^{1/2})$ for planar graphs and $T((|V|/k)^{1-1/d})$ for well shaped meshes in *d*-dimensions. The ratio is considered between the produced and optimal solutions, *k* and $|V|$.

In [57], Kuo and Cheng present a new network flow approach for partitioning circuits into tree hierarchies. They formulate a linear program for the hierarchical tree partitioning problem by spreading metrics proposed in [47]. The size constraints in partitioning can be formulated directly as linear constraints. Motivated by the duality between the linear programs for partitioning and network flow problems, the authors devise a heuristic algorithm based on network flow and spreading metric computations. Experimental results demonstrate that their algorithm can generate better solutions for MCNC benchmarks.

Multilevel algorithms are a successful class of optimization techniques which addresses the mesh partitioning problem. They usually combine a graph contraction algorithm together with a local optimization method which refines the partition at each graph level. In [63], Walshaw and Cross present an enhancement of the technique which uses imbalance to achieve higher quality

partitions. They also present a formulation of the Kernighan--Lin partition optimization algorithm which incorporates load-balancing. The resulting algorithm is tested against a different but related state-of-the-art partitioner and shown to provide improved results.

In [64], Saha et al. deal with a design problem for a network of Personal Communication Services (PCS). The goal is to assign cells to switches in a PCS Network (PCSN) in an optimal manner so as to minimize the total cost which includes two types of cost, namely handoff cost between two adjacent cells, and cable cost between cells and switches. The design is to be optimized subject to the constraint that the call volume of each switch must not exceed its call handling capacity. In the literature, this problem has been conventionally formulated as an integer programming problem. However, because of the time complexity of the problem, the solution procedures are usually heuristic when the number of cells and switches are more. The authors proposed an assignment heuristic which is faster and much simpler than the existing algorithms. Despite its simplicity, experimental results show that it performs equally well in terms of solution quality, and, at the same time, it is faster than its predecessors. They present the algorithm as well as comparative results to justify their claim.

Kobler and Rotics [65] consider three graph partitioning problems, both from the vertices and the edges point of view. These problems are dominating set, list-q-coloring with costs (fixed number of colors q) and coloring with non-fixed number of colors. They are all known to be NP-hard in general. The authors show that all these problems (except edge-coloring) can be solved in polynomial time on graphs with clique-width bounded by some constant k, if the k-expression of the input graph is also given. In particular, they present the first polynomial algorithms (on these classes) for chromatic number, edge-dominating set and list-q-coloring with costs (fixed number of colors q, both vertex and edge versions). Since these classes of graphs include classes like P4-sparse graphs, distance hereditary graphs and graphs with bounded treewidth, their algorithms also apply to these graphs.

In [66], Schweitz and Agrawal present a novel multilevel graph partitioning algorithm, KACE, which uses knowledge about the domain and employs several graph transformation techniques. Both functional and structural parallelism in the sequential code are explored to improve the quality of parallel tasks. Statistical information about communication times between nodes as a function of message size and/or other factors are used to have a better estimate of balancing factors, code replication, and synchronization penalties. This enables the use of a task cohesion algorithm to obtain a coarse version of the partitioned graph. Many of KACE's parameters are shown to have definite impact on the parallelized program code.

Amir et al. [68] devise the first constant factor approximation algorithm for minimum quotient vertex-cuts in planar graphs. Their algorithm achieves approximation ratio $1+4/3(1+\varepsilon)$ with running time $O(W \bullet n^{3+2/\varepsilon})$, where W is the total weight of the vertices. The approximation ratio improves to $4/3(1+\varepsilon+O(1))$ if there is an optimal quotient vertex-cut (A^*,B^*,C^*) where the weight of C^* is of low order compared to those of A^* and B^*; this holds, for example, when the input graph has uniform weights and costs. The ratio further improves to $1+\varepsilon+O(1)$ if, in addition, $\min[w(A^*),w(B^*)] \leq 1/3\ W$. The algorithm is used for

quotient vertex-cuts to achieve the first constant-factor pseudo-approximation for vertex separators in planar graphs. The authors' technical contribution is two-fold. First, they prove a structural theorem for planar graphs, showing the existence of a near-optimal quotient vertex-cut whose high-level structure is that of a bounded-depth tree. Second, they develop an algorithm that optimizes over such complex structures in running time that depends (exponentially) not on the size of the structure, but rather only on its depth. These techniques may be applicable in other problems.

In the context of VLSI CAD, Drechsler et al. [69] present a new recursive bi-partitioning algorithm that is especially applicable, if a large number of final partitions, e.g. more than 1000, has to be computed. The algorithm consists of two steps. Based on recursive splits the problem is divided into several sub-problems, but with increasing recursion depth more run time is invested. By this an initial solution is determined very fast. The core of the method is a second step, where a very powerful greedy algorithm is applied to refine the partitions. Experimental results are given that compare the new approach to state-of-the-art tools. The experiments show that the new approach outperforms the standard techniques with respect to run time and quality. Furthermore, the memory usage is very low and is reduced in comparison to other methods by more than a factor of four.

Kim and Moon [74] perform an empirical study on the local-optimum space in graph bipartitioning. The authors examine some statistical features of the fitness landscape and the local properties of the landscape. These features include the distributions of local optima, their cost-distance correlations, their attraction powers, the properties around the central area of local optima, etc. The study reveals some new notable results about the properties of the fitness landscape. For example, the central area yielded good quality in local-optimum space, the local-optimum space had the self-similar structure of global convexity, local optima showed clusters in more than one place, etc. The authors also provide a simple experiment on whether it is worth to exploit the area around the Euclidean center of the problem space.

In [76], Spielman et al. present algorithms for solving symmetric, diagonally-dominant linear systems to accuracy ε in time linear in their number of non-zeros and log ($\kappa f(A)$ ε), where $\kappa f(A)$ is the condition number of the matrix defining the linear system. Their algorithm applies the preconditioned Chebyshev iteration with preconditioners designed using nearly-linear time algorithms for graph sparsification and graph partitioning.

Andreev et al. [77] consider the problem of (k, υ)-balanced graph partitioning - dividing the vertices of a graph into k almost equal size components (each of size less than $\upsilon \bullet n/k$) so that the capacity of edges between different components is minimized. This problem is a natural generalization of several other problems such as minimum bisection, which is the (2,1)-balanced partitioning problem. The authors present a bicriteria polynomial time approximation algorithm with an $O(\log^2 n)$-approximation for any constant $\upsilon > 1$. For $\upsilon = 1$ we show that no polytime approximation algorithm can guarantee a finite approximation ratio unless P=NP. Previous work has only considered the (k, υ)-balanced partitioning problem for $\upsilon \geq 2$.

In the context of image segmentation [78] Martinez et al. suggest an alternative implementation of the *k*-way Ncut approach. The authors believe that this implementation alleviates a problem associated with the Ncut algorithm for some types of images: its tendency to partition regions that are nearly uniform with respect to the segmentation parameter. Previous implementations have used the *k*-means algorithm to cluster the data in the eigenspace of the affinity matrix. In the *k*-means based implementations, the number of clusters is estimated by minimizing a function that represents the quality of the results produced by each possible value of *k*. The proposed approach uses the clustering algorithm of Koontz and Fukunaga [4] in which *k* is automatically selected as clusters are formed (in a single iteration). The authors show comparison results obtained with the two different approaches to non-parametric clustering. The Ncut generated oversegmentations are further suppressed by a grouping stage - also Ncut based -in their implementation. The affinity matrix for the grouping stage uses similarity based on the mean values of the segments.

Chen et al. [81] consider a different application of graph partitioning. Most massively multiplayer game servers employ static partitioning of their game world into distinct mini-worlds that are hosted on separate servers. This limits cross-server interactions between players, and exposes the division of the world to players. The authors have designed and implemented an architecture in which the partitioning of game regions across servers is transparent to players and interactions are not limited to objects in a single region or server. This allows a finer grain partitioning, which combined with a dynamic load management algorithm enables a better handling of transient crowding by adaptively dispersing or aggregating regions from servers in response to quality of service violations. The authors' load balancing algorithm is aware of the spatial locality in the virtual game world. Based on localized information, the algorithm balances the load and reduces the cross server communication, while avoiding frequent reassignment of regions. Results show that locality aware load balancing reduces the average user response time by up to a factor of 6 compared to a global algorithm that does not consider spatial locality and by up to a factor of 8 compared to static partitioning.

In [83] partitioning of finite element meshes is tackled using colonies of artificial ant-like agents. These agents must restructure the resources in their environment in a manner which corresponds to a good solution of the underlying problem. Standard approaches to these problems use recursive methods in which the final solution is dependent on solutions found at higher levels. For example partitioning into *k* sets is done using recursive bisection which can often provide a partition which is far from optimal [40]. The inherently parallel, distributed nature of the swarm-based paradigm allows for simultaneously partitioning into *k* sets. Results show that this approach can be superior in quality when compared to standard methods. Whilst it is marginally slower, the reduced communication cost will greatly reduce the much longer simulation phase of the finite element method. Hence this will outweigh the initial cost of making the partition.

Graph clustering with a general cut objective has been shown to be mathematically equivalent to an appropriate weighted kernel *k*-means objective function. In [85], Dhillon et al. exploit this equivalence to develop a very fast

multilevel algorithm for graph clustering. Multilevel approaches involve coarsening, initial partitioning and refinement phases, all of which may be specialized to different graph clustering objectives. Unlike existing multilevel clustering approaches, such as METIS, The authors' algorithm does not constrain the cluster sizes to be nearly equal. Their approach gives a theoretical guarantee that the refinement step decreases the graph cut objective under consideration. Experiments show that they achieve better final objective function values as compared to a state-of-the-art spectral clustering algorithm: on a series of benchmark test graphs with up to thirty thousand nodes and one million edges, their algorithm achieves lower normalized cut values in 67% of the experiments and higher ratio association values in 100% of the experiments. Furthermore, on large graphs, their algorithm is significantly faster than spectral methods. Finally, the algorithm requires far less memory than spectral methods. The authors cluster a 1.2 million node movie network into 5000 clusters, which due to memory requirements cannot be done directly with spectral methods.

In [87], Feige et al. present an algorithm that finds a bisection whose cost is within a factor of $O(\log^{1.5} V)$ from the minimum. For graphs excluding any fixed graph as a minor (e.g., planar graphs) the authors obtain an improved approximation ratio of $O(\log V)$. The previously known approximation ratio for bisection was roughly $|V|^{1/2}$.

In [89], the authors introduce new types of variational segmentation cost functions and associated active contour methods that are based on pairwise similarities or dissimilarities of the pixels. As a solution to a minimization problem, they introduce a new curve evolution framework, the graph partitioning active contours (GPAC). Using global features, their curve evolution is able to produce results close to the ideal minimization of such cost functions. New and efficient implementation techniques are also introduced. Experiments show that GPAC solution is effective on natural images and computationally efficient. Experiments on gray-scale, color, and texture images show promising segmentation results.

Khandekar et al. [90] show that the sparsest cut in graphs can be approximated within $O(\log^2 V)$ factor in $\tilde{O}(V^{3/2})$ time using polylogarithmic single commodity max-flow computations. Previous algorithms are based on multicommodity flows which take time $\tilde{O}(V^2)$. The authors' algorithm iteratively employs max-flow computations to embed an expander flow, thus providing a certificate of expansion. This technique can also be extended to yield an $O(\log^2 V)$ (pseudo) approximation algorithm for the edge-separator problem with a similar running time.

In [94], the authors compare five Multi-Objective Meta-Heuristic (MOMH) algorithms by solving the network partitioning problem. MOMH are defined as master strategies that guide and modify other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. The algorithms they present are based on three optimization techniques: SA TS GA.

.Since C. Gil et al., compare the above three techniques from a multi-objective viewpoint, they compare each technique's resulting set of indifferent solutions. These sets include solutions which do not dominate one another, that is, there are no solutions which are better or at least equal in all the objectives,

comparing to all the other solutions of the set. The comparison between these sets is made in terms of

-the coverage (C) of two sets, which expresses whether the solutions of one set are dominating or indifferent to the solutions of another set

-the average size of the space covered S, which in the bi-dimensional case corresponds to the average rectangle space defined by the (0.0) (f_1, f_2) for all solutions f of a set and expresses the approximation to the unknown Pareto-optimal front of solutions.

The striking observation was that in all cases a MOMH based on simulated annealing would outperform a MOHM based on Tabu search and evolutionary methods. This indicates that simulated annealing must be considered as the most reliable choice, provided that the parameters are fine-tuned with experimentation.

Povh and Rendl [95] consider 3-partitioning the vertices of a graph into sets S_1, S_2, and S_3 of specified cardinalities, such that the total weight of all edges joining S_1 and S_2 is minimized. This problem is closely related to several NP-hard problems like determining the bandwidth or finding a vertex separator in a graph. The authors show that this problem can be formulated as a linear program over the cone of completely positive matrices, leading in a natural way to semidefinite relaxations of the problem. They show in particular that the spectral relaxation introduced by Helmberg et al. can equivalently be formulated as a semidefinite program. Finally, the authors propose a tightened version of this semidefinite program and show on some small instances that this new bound is a significant improvement over the spectral bound.

In [96], Aleksandrov et al. describe new algorithms for computing separators in planar graphs as well as techniques that can be used to speed up the implementation of graph partitioning algorithms and improve the partition quality. In particular, they consider planar graphs with costs and weights on the vertices, where weights are used to estimate the sizes of the partitions and costs are used to estimate the size of the separator. The authors show that in these graphs one can always find a small cost separator (consisting of vertices or edges) that partitions the graph into components of bounded weight. They describe implementations of the partitioning algorithms and discuss results of their experiments.

Dhillon et al. [98] discuss the equivalence between the objective functions used in spectral clustering and kernel k-means. In particular, a general weighted kernel k-means objective is mathematically equivalent to a weighted graph clustering objective. The authors exploit this equivalence to develop a fast, high-quality multilevel algorithm that directly optimizes various weighted graph clustering objectives, such as the popular ratio cut, normalized cut, and ratio association criteria. This eliminates the need for any eigenvector computation for graph clustering problems, which can be prohibitive for very large graphs. Previous multilevel graph partitioning methods, such as Metis, have suffered from the restriction of equal-sized clusters; the authors' multilevel algorithm removes this restriction by using kernel k-means to optimize weighted graph cuts. Experimental results show that this multilevel algorithm outperforms a state-of-the-art spectral clustering algorithm in terms of speed, memory usage, and quality. The authors demonstrate that their algorithm is applicable to large-

scale clustering tasks such as image segmentation, social network analysis and gene network analysis.

*Appendix B*Experiments on Tolerance

B.1 Detailed Results

Set #5 was used to cross partition results with those of previous sets. Thus, graph size was between 325 and 425 (step of 50), *maxWeight* was set to 10^5, *minWeight* varied from 10^3 to 10^4 (step of 10^3), and the number of population centers varied from 2% to 7% of graph size.

The partition cost is plotted against the tolerance (in percentage of optimal partition size) in Figures 39 and 40. Comparing the two extreme cases of *minWeight* equal to 10^3 and 10^4, with respect to partition cost, there is more gain when increasing tolerance for the former case. Interestingly, cost does not decrease linearly with tolerance. Cost decreases as tolerance increases up to a value of 10%, for most cases. From that point, it stays relatively stable until tolerance reaches a considerably high value, which obviously allows for shorter, in hops, bisecting paths, resulting in further reduction of bisection cost.

Graphs of set #6, whose *minWeight* was higher, had a lower percentage of unbisected runs than those of set #5, which is in accordance with previous results.

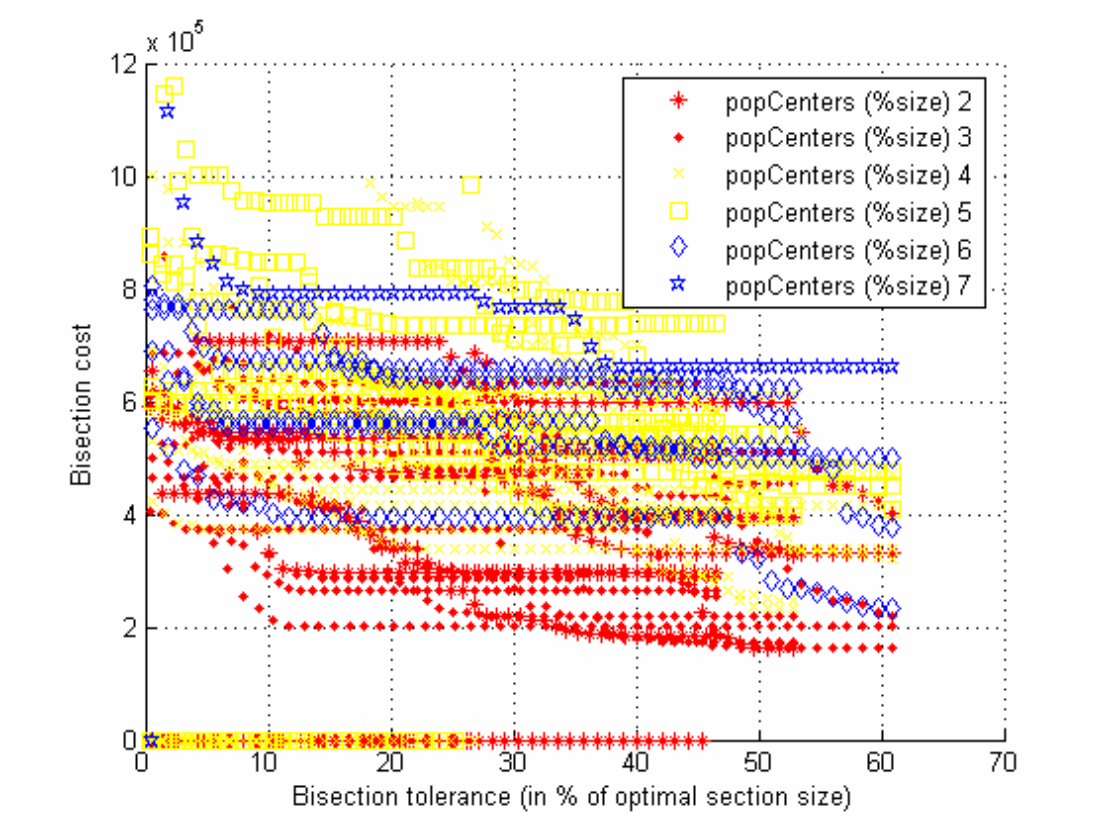

Figure 39. Partition cost per tolerance for graphs with *minWeight* 10^3, set #5

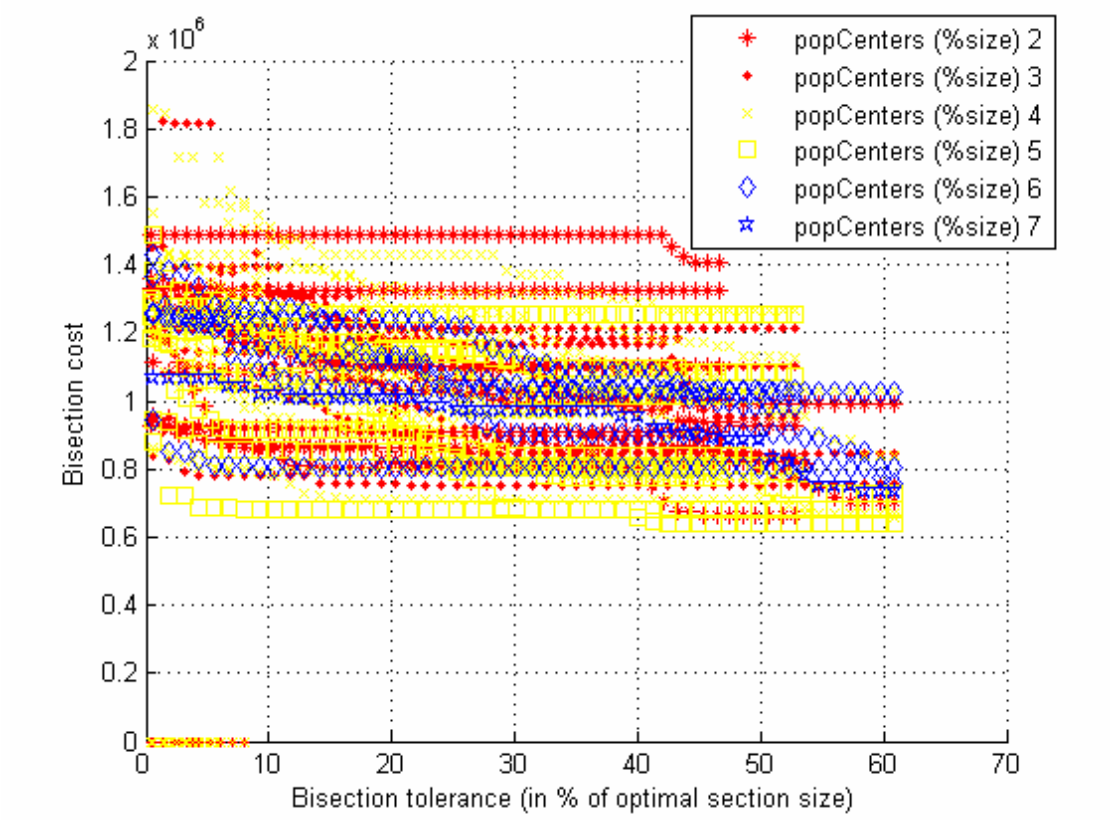

Figure 40. Partition cost per tolerance for graphs with *minWeight* 10^4, set #5

In section 3..3.2 we concluded that a tighter global weight range favors DGP and in section 3..3.3 we speculated that the bisecting path length indicates how "hard" is for DGP to bisect a graph. These observations are verified by Figure 41, where the length of bisecting paths, grouped by *minWeight*, is plotted against tolerance. Graph instances with higher *minWeight* were bisected with shorter bisecting paths. The observation that the lower bound of bisection length is around twice the graph's diameter length is also established, for relatively low tolerance values. As it was expected, as tolerance increases to higher values, bisection length decreases. This was also reflected in the bisection cost, which followed a similar trend. For tolerance values over 20% of the optimal section size, the bisection length decreases. It reaches a lower bound of 1.5 times the size of the graph's diameter for the significant tolerance value of 50-60% of the optimal partition size. There are no unpartitioned graph instances for tolerance values over 52%.

The results are also grouped by the percentage of graph nodes which are population centers. In order to decouple the effect of *minWeight* value, results from the two extreme cases of 10^3 and 10^4 are shown in Figures 42 and 43.

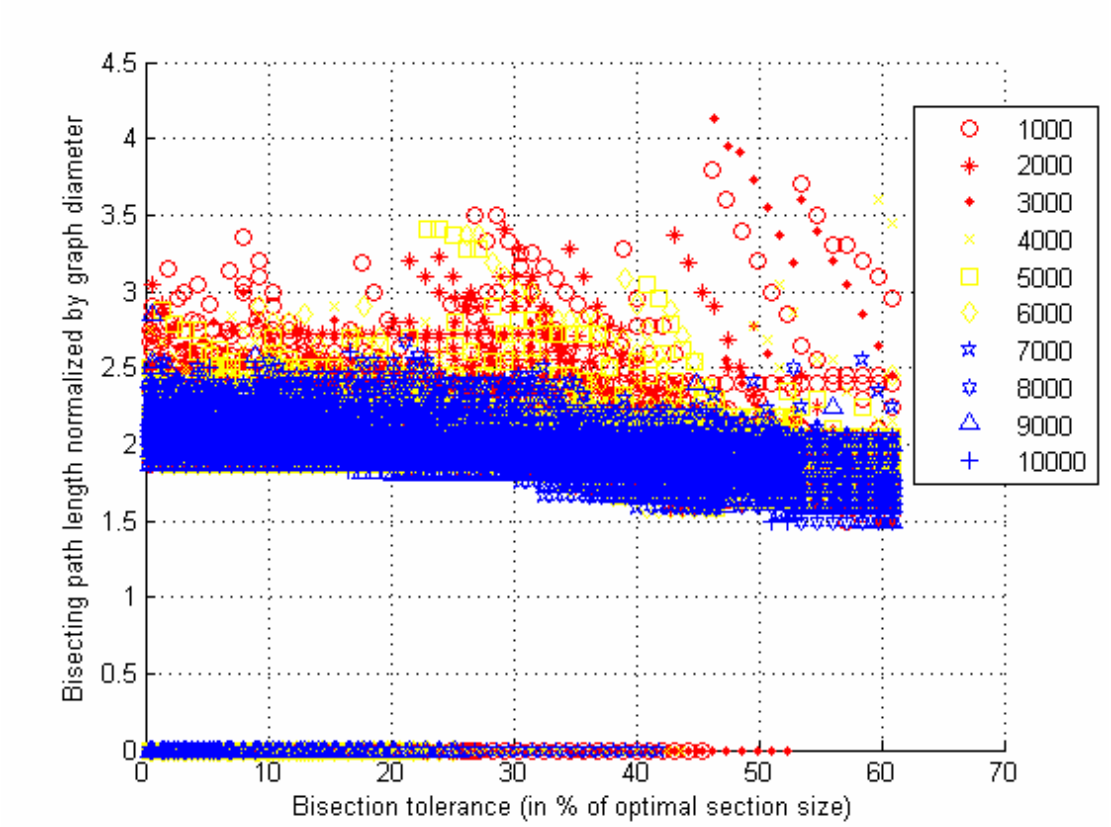

Figure 41. Bisection length per tolerance, grouped by *minWeight* for set #5

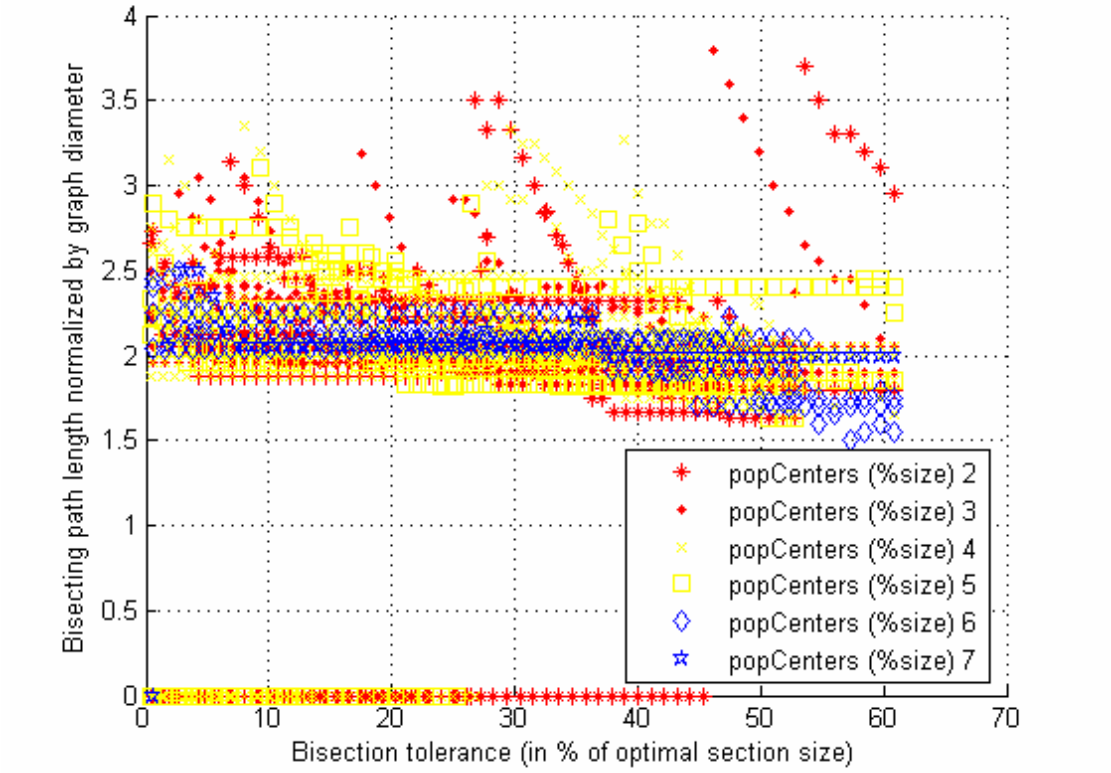

Figure 42. Bisection length per tolerance, for graphs with *minWeight* 10^3, grouped by number of population centers for set #5

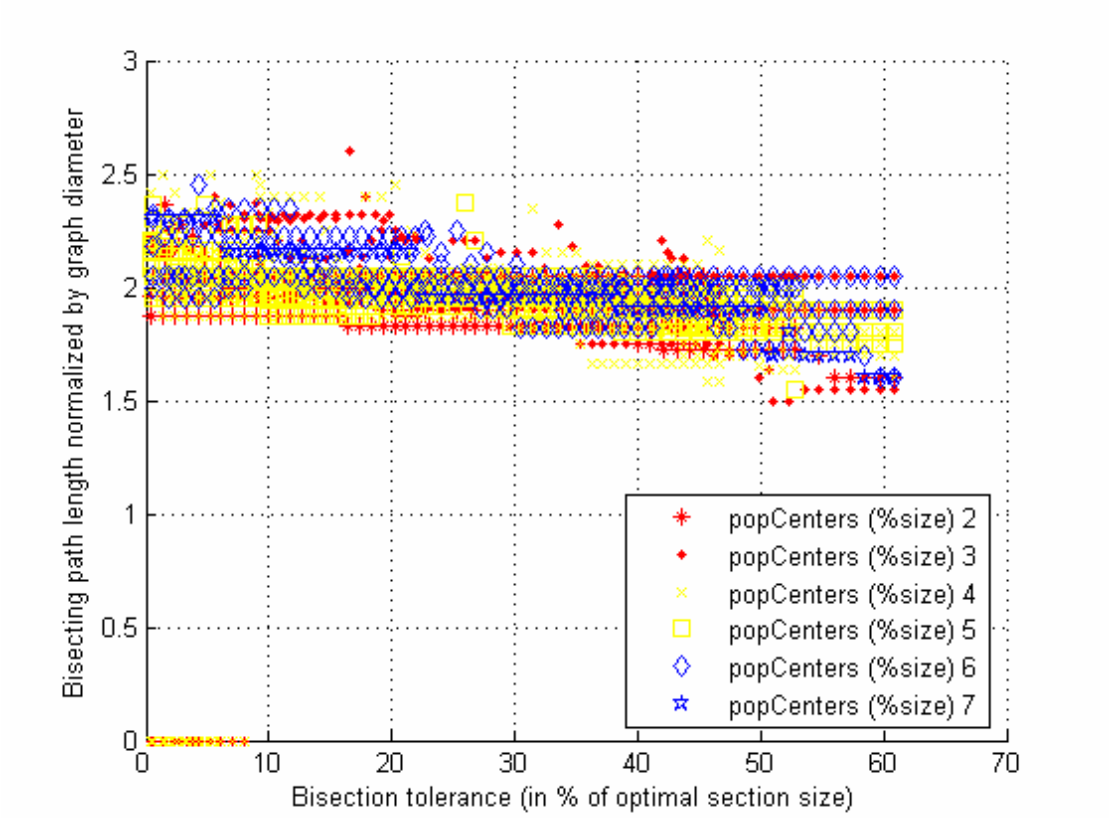

Figure 43. Bisection length per tolerance, for graphs with *minWeight* 10^4, grouped by number of population centers for set #5

Most variable bisection lengths result from graph instances with the widest global weight range, whereas instances with a tighter global weight range result in bisection lengths close to lower bound. The former case exhibits graph instances which can't be bisected until a certain tolerance value, which implies that the topology is such that affects the results. Note that it seems that these unpartitioned cases come mostly from graph instances of lower population centers, which can have more variable spatial weight distributions. In such cases, it seems that when a graph can't be partitioned due to weight distribution, eventual partitioning at higher values concern BB shortest paths, whose route twist through regions of high weights. On the other hand, many population centers can saturate the graph space, dividing it in many small clusters of high and low weights. In the absence of large distinct clusters of high weights, there exist BB shortest paths that go through the central area of the graph, bisecting it with a desired tolerance.

Finally, the percentage of graphs which were successfully bisected at each tolerance is presented in Figures 44 and 45. It is once more clear that tighter global weight ranges exhibit higher percentages of partitioned graph instances. On the other hand, there is not a clear trend concerning frequently partitioned graphs and their percentage of population centers. What is apparent though is that graphs with the lowest percentages of population centers were less frequently partitioned by the algorithm.

We can conclude that apart from minWeight, which is an easily measured parameter, the algorithm result depends primarily on the spatial distribution of weights.

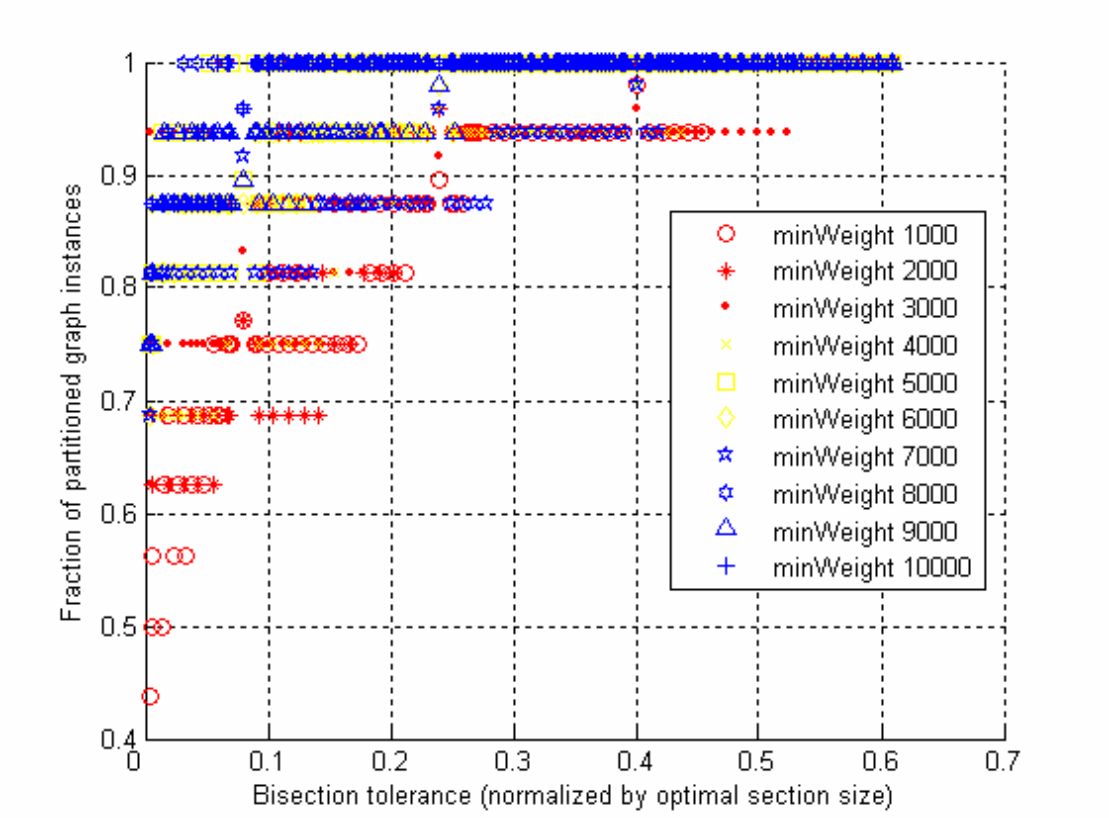

Figure 44. Partitioned graphs per tolerance, grouped by *minWeight* for set #5

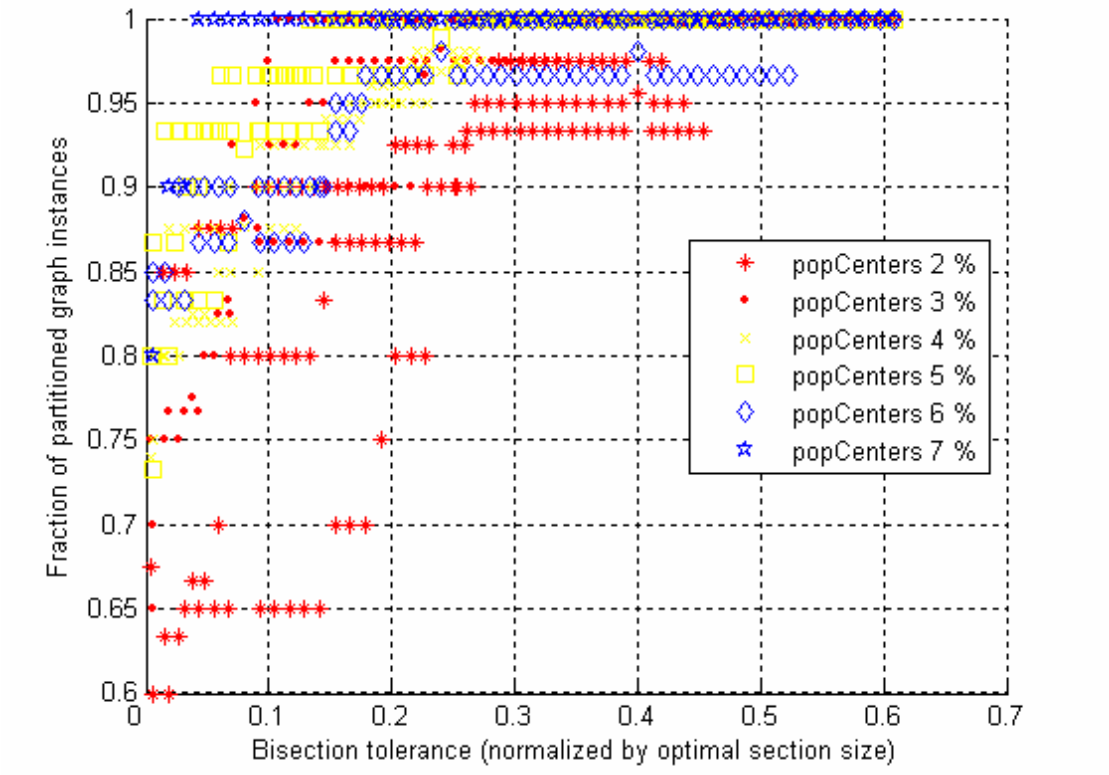

Figure 45. Partitioned graphs per tolerance, grouped by number of population centers for set #5

*Appendix C*The *maxDist* parameter

C.1 Graph characteristics and Partition results vs. *maxDist*

We analyzed the relation between *maxDist* and graph characteristics as well as partition results, for most generated graph sets. *maxDist* was normalized over the graph's diameter in order to make results from different sets comparable. The fraction of partitioned graphs sharing the same *maxDist* is presented, grouped by the characteristic that each set focused on. The most representative results are shown in Figures 46 to 49, from almost all graph sets. Graphs with extreme *maxDist* values were more frequently bisected, whereas those with intermediate *maxDist* values had various succession rates. Most unsuccessful and low fractions of successful partitioning correspond to intermediate *maxDist* values.

The length in hops of bisecting paths, for graphs sharing the same *maxDist*, complements these results. As we have concluded in previous results, the length of a partition is an indication of how "easy" was for the algorithm to find a result. In Figures 50 to 54, the distinct path lengths that occurred for each *maxDist* are shown. Note that each path length is normalized by the corresponding graph diameter. Indeed, extreme *maxDist* values correspond only to lower lengths, whereas intermediate values have variable lengths including long paths and most unpartitioned cases, which are represented by zero values. The effect of *minWeight* is notable for plots concerning set #3 as well as the correspondence between extreme *maxDist* values and the number of population centers, in plots of set #4 and #5.

Although the distinct path lengths are presented, there is no info concerning the frequency of their occurrence. All sets followed the same pattern concerning the latter information, which is shown in a representative plot of set #5, in Figure 55. Most bisecting paths lengths are around two times the graph's diameter. Moreover, most of the occurred *maxDist* values are centered around half the graph diameter and their number decreases while moving towards extreme *maxDist* values.

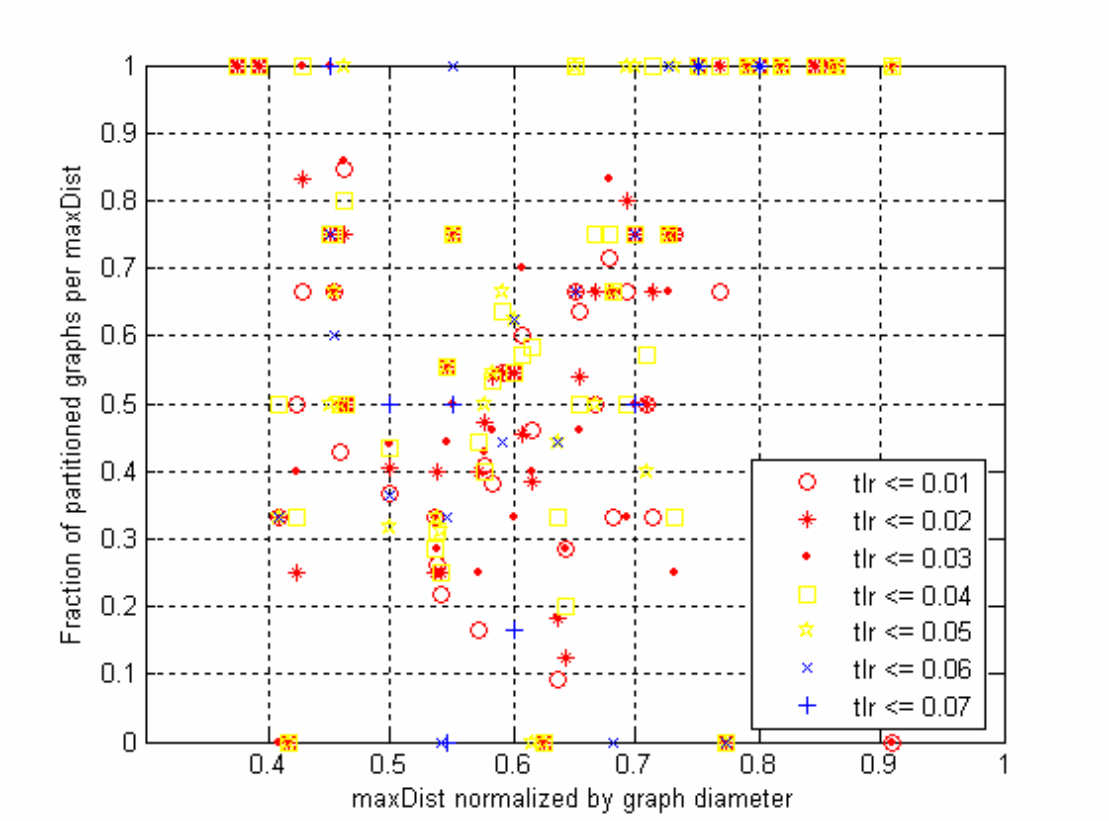

Figure 46. Partitioned graphs per *maxDist*, grouped by tolerance bins of 0.01 for set #1

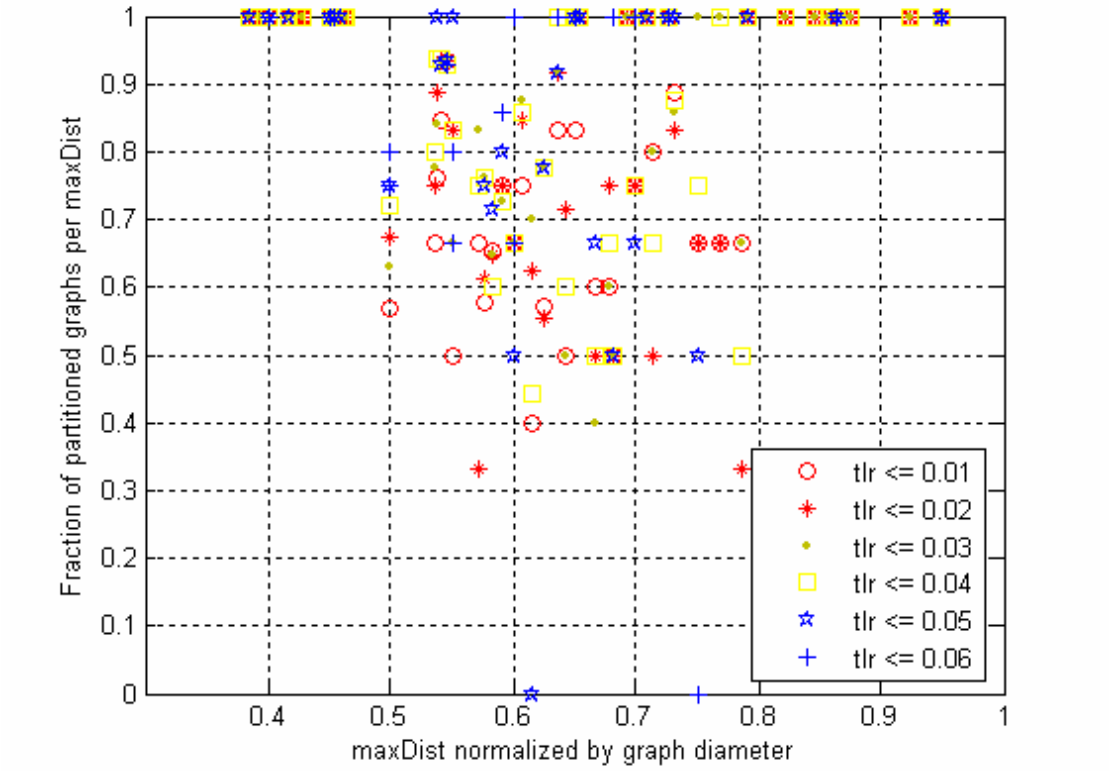

Figure 47. Partitioned graphs per *maxDist*, grouped by tolerane bins of 0.01 for set #2

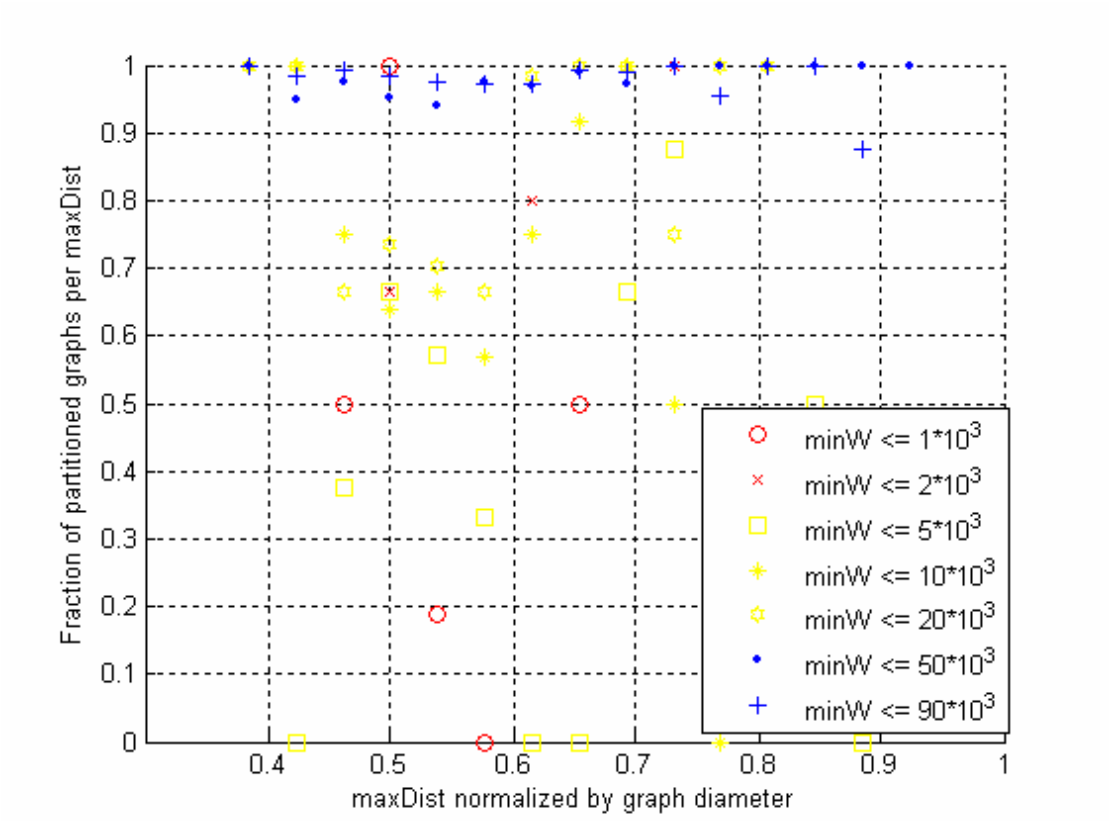

Figure 48. Partitioned graphs per *maxDist*, grouped by *minWeight* for set #3

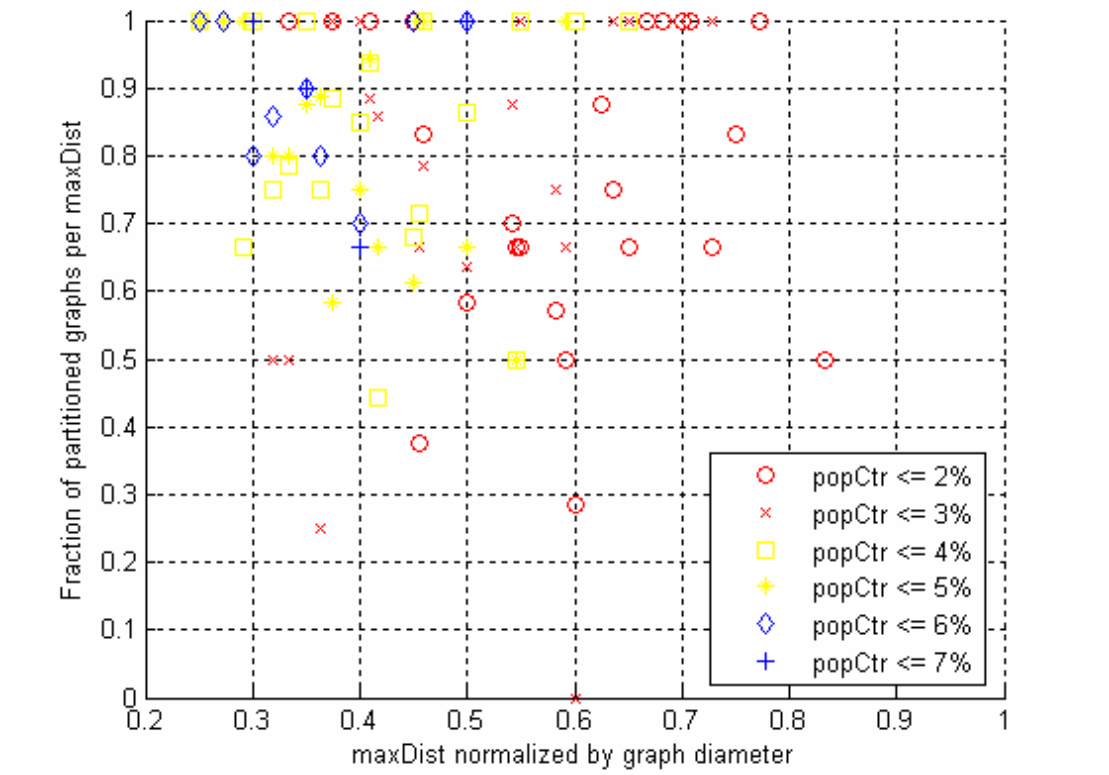

Figure 49. Partitioned graphs per *maxDist*, grouped by number of population centers for set #5

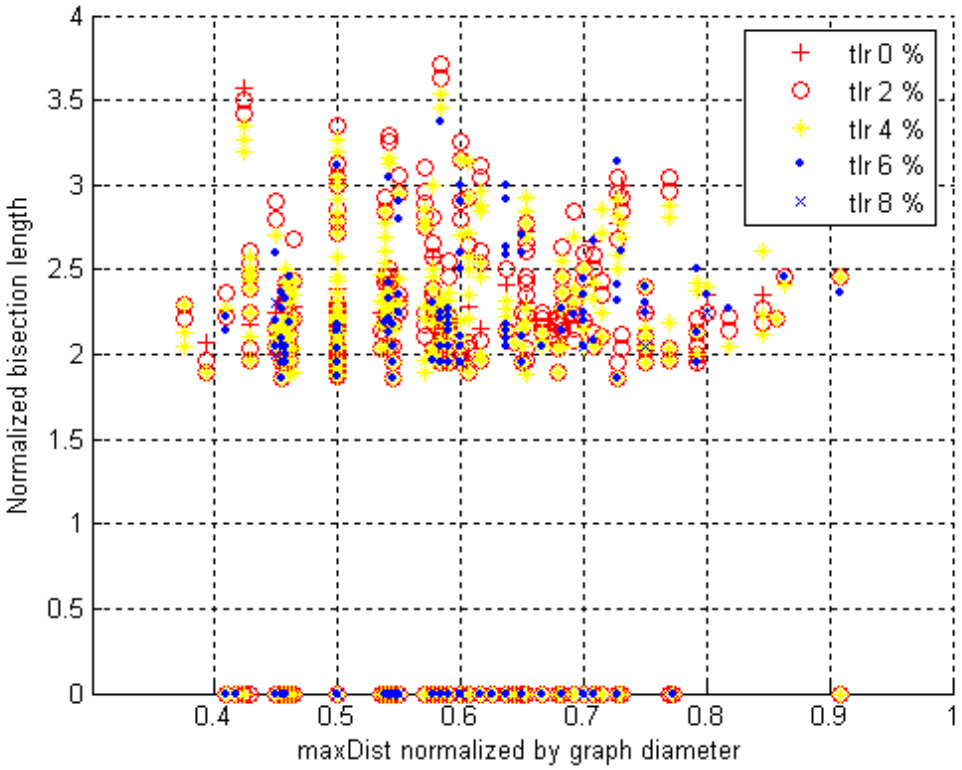

Figure 50. Bisection length per *maxDist*, grouped by tolerance bins of 0.02 for set #1

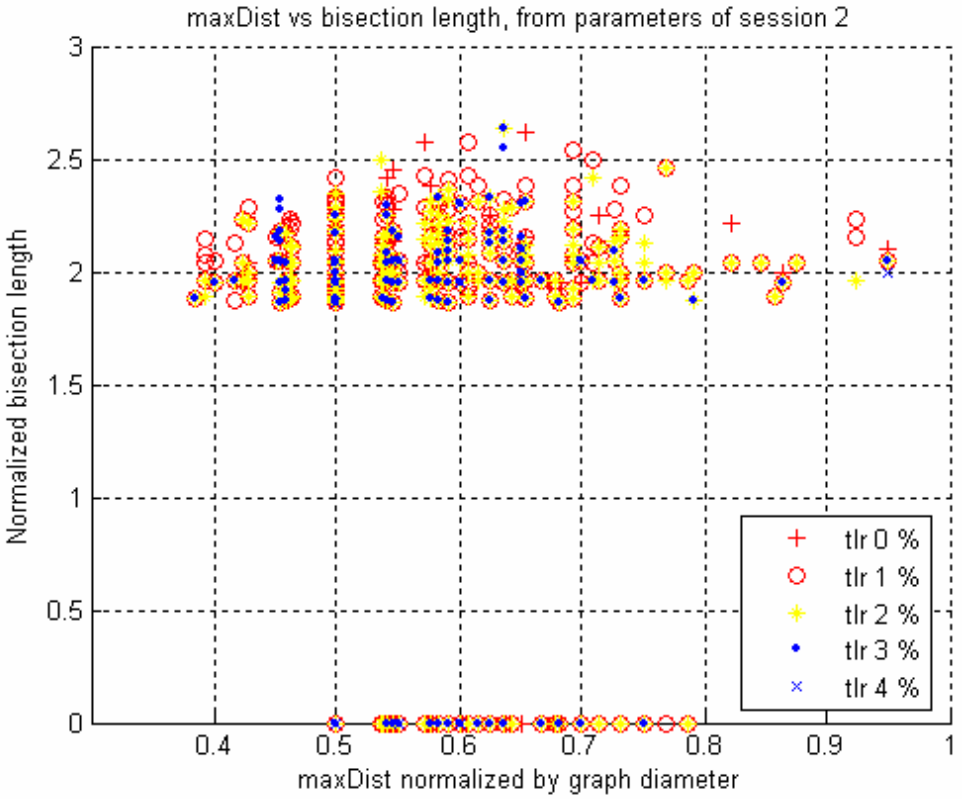

Figure 51. Bisection length per *maxDist*, grouped by tolerance bins of 0.02 for set #2

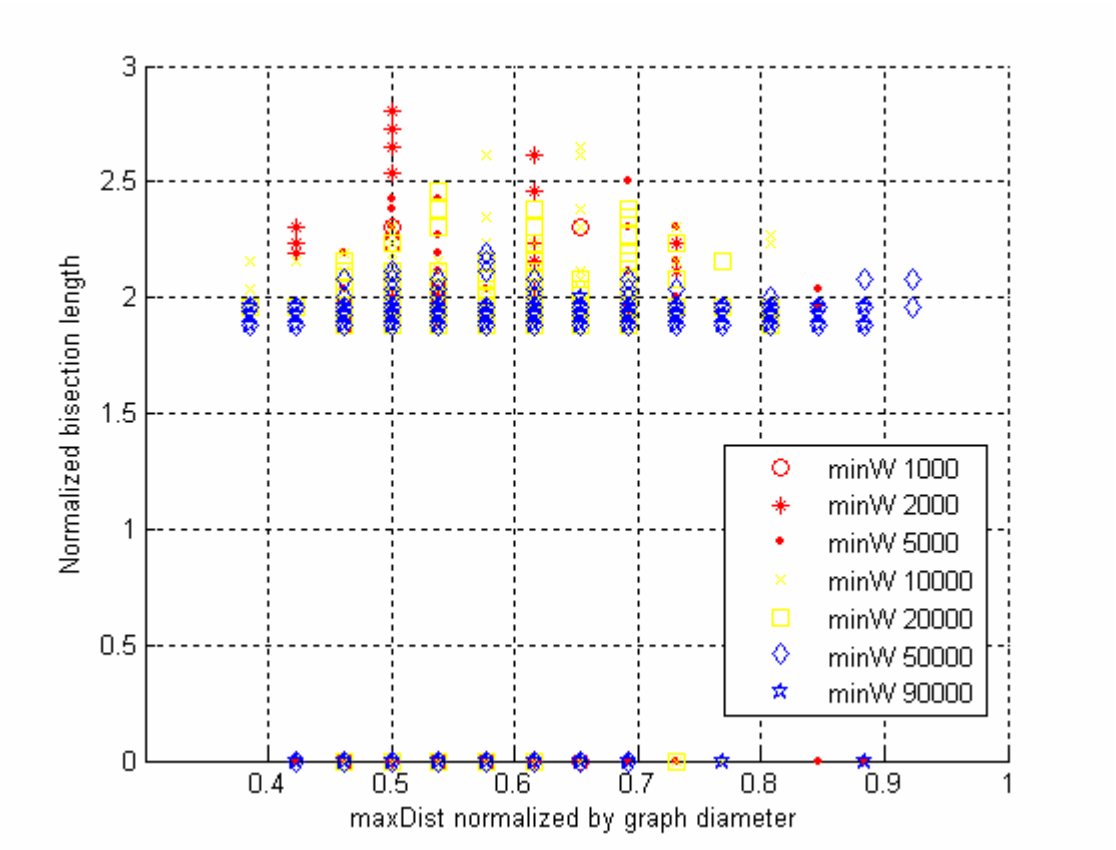

Figure 52. Bisection length per *maxDist*, grouped by *minWeight* for set #3

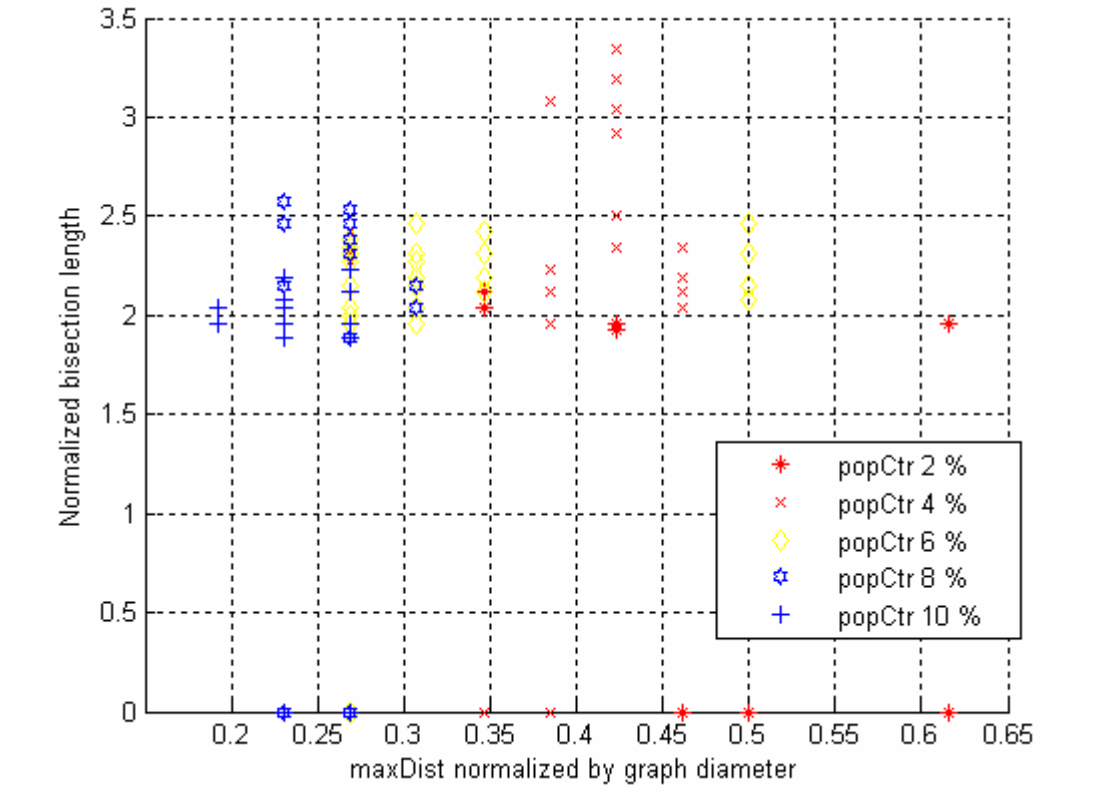

Figure 53. Bisection length per *maxDist*, grouped by number of population centers for set #4.1

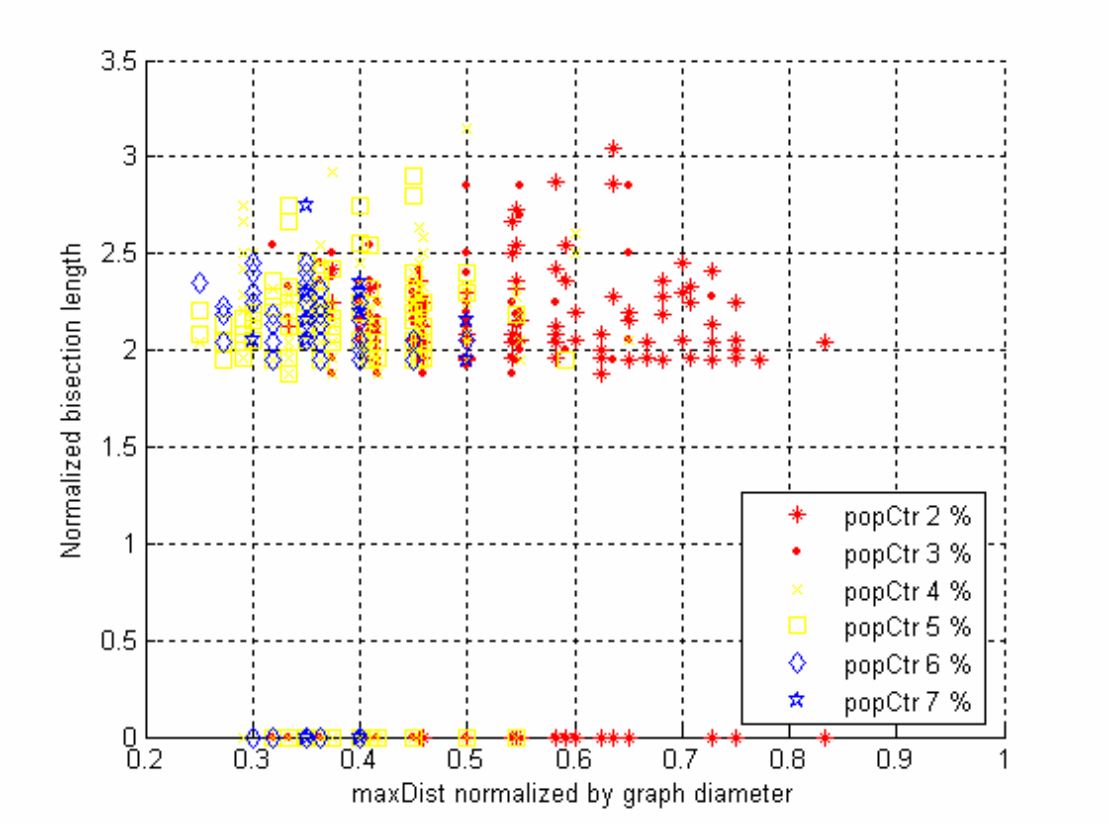

Figure 54. Bisection length per *maxDist*, grouped by number of population centers for set #5

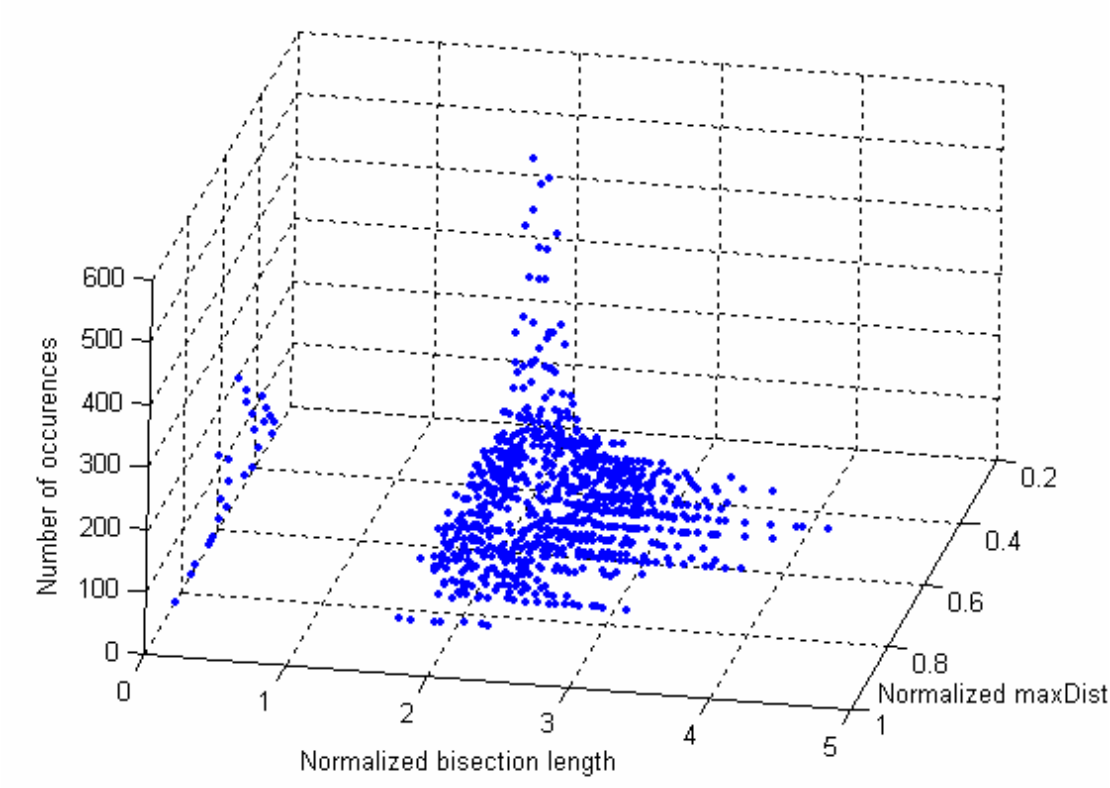

Figure 55. Bisection length vs *maxDist* vs number of occurences, for set #5

C.2 Intuition behind Results

Results verify our hypothesis about *maxDist* and spatial distribution. **Higher** *maxDist* values mostly belong to graphs with **fewer** population centers. From these graphs, those with average *maxDist* values exhibit the highest percentages of unpartitioned nodes. However, highest *maxDist* does not correspond to these cases. Extremely high *maxDist* tend to be easily bisected. The reason lies in the definition of *maxDist*: if a node can be in such a distance from a population center, that is close to the graph diameter (e.g. 0.7-0.8 times the graph diameter), it is apparent that there are also many nodes in its neighborhood which share analogous distances, which form a great region with low values. Such a big distance from a population center means that most population centers are near the border of the graph (*maxDist* is significantly greater than the graphs radius and thus, they cannot be near the center).

Depending on their number and exact location, population centers can either be close one another, or spread at the perimeter of the graph. In the former case, they create a cluster of population centers and hence, a small and dense cluster of high weights. A large fraction of the graph consists of average and lower edge weights. Thus, there's a high probability that a BB shortest path, in the dual graph, will bisect the graph under the demanded tolerance, since routes through the central region of the graph are not restricted by the heaviest weights. In the latter case, highest weights located throughout the graph border do not negatively affect the algorithm, since a bisecting path has to begin from a border node, in any case.

We can conclude that, its the interior of the graph where the distribution of high and low weights can yield quite twisted and long shortest paths between border dual nodes, such that would partition the graph in an undesired section size ratio. Hence, a bisecting path is more likely to be found for extremely high *maxDist* values.

On the other extreme, **many** population centers clearly have a great probability of corresponding to **very low** *maxDist* values. This is indicated by Figures 53 and 54. Too many population centers will more likely expand in the whole space of the graph thus providing a blend of higher and lower weights. Besides, too concentrated population centers could be considered as degenerating this graph to an instance with fewer population centers located uniformly in graph space. Therefore, a significant percentage of population centers results in the occurrence of high weights as well as low weights, between these population centers, which are probably relatively close one to each other and spread throughout the whole graph. Consequently, low weights will be located in many graph regions, including the graph central area, forming a candidate sub-route for all BB shortest paths, in the dual graph.

It is thus, the intermediate *maxDist* values which refer to graph instances which might not be bisected by the algorithm. Consider the most frequent case of *maxDist* equal to a value around the radius of the graph. This could correspond to a graph whose population centers are located either near its outer border, or they form a cluster in the middle of the graph. As we have seen in section 3..3.3, the latter case clearly affects the BB shortest paths in such a way, that they would go around this region. If this region is big enough, it can be sure

than no shortest path will have a route that will partition the graph under a desired tolerance.

This very specific case is not the only one. Many variations of this phenomenon can result locally in areas around the center of the graph. This notation, concerning the cluster and its size which can render the graph unbisectable, can be generalized. A cluster of population centers at any location in a graph can render shortest paths undesirable if it embodies a number of nodes close to half the size of the graph. It is then certain that no path between border nodes, which do not belong to this cluster, will result in an optimal partition. Especially, in the case that this cluster does not have many border nodes (i.e., only few shortest paths will begin from a node of this cluster), it is unlikely that any shortest BB path will bisect the path, for a practical tolerance value.

Finally, it is once more ascertained, that graphs with tight global weight ranges decrease the unevenness of weights and the aforementioned phenomena are alleviated, thus resulting in higher percentages of bisected graphs.