GRAPH BISECTION ALGORITHMS

by

THANG NGUYEN BUI

B.S.E.E., Carnegie-Mellon University
(June 1980)
B.S. Mathematics, Carnegie-Mellon University
(June 1980)
S.M., Massachusetts Institute of Technology
(January 1983)

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the
Requirements for the
Degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1986

Signature of Author_____
Department of Electrical Engineering and Computer Science
January 24, 1986

Certified by_____
F. Thomson Leighton
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Departmental Graduate Committee

GRAPH BISECTION ALGORITHMS

by

Thang Nguyen Bui

Submitted to the Department of Electrical Engineering and Computer Science
on January 24, 1986 in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy in
Computer Science

ABSTRACT

In this thesis, we describe a polynomial time algorithm that, for every input graph, either outputs the minimum bisection of the graph or halts without output. More importantly, we show that the algorithm chooses the former course with high probability for many natural classes of graphs. In particular, for every fixed $d \geq 3$, all sufficiently large n and all $b = o(n^{1-1/\lfloor \frac{d+1}{2} \rfloor})$, the algorithm finds the minimum bisection for almost all d-regular labelled simple graphs with $2n$ nodes and bisection width b. For example, the algorithm succeeds for almost all 5-regular graphs with $2n$ nodes and bisection width $o(n^{2/3})$. The algorithm differs from other graph bisection heuristics (as well as from many heuristics for other NP-complete problems) in several respects. Most notably:

(i) the algorithm provides exactly the minimum bisection for almost all input graphs with the specified form, instead of only an approximation of the minimum bisection,
(ii) whenever the algorithm produces a bisection, it is guaranteed to be optimal (i.e., the algorithm also produces a proof that the bisection it outputs is an optimal bisection),
(iii) the algorithm works well both theoretically and experimentally,
(iv) the algorithm employs global methods such as network flow instead of local operations such as 2-changes, and
(v) the algorithm works well for graphs with small bisections (as opposed to graphs with large bisections, for which arbitrary bisections are nearly optimal).

We also show that with high probability the greedy algorithm will not be able to find the optimal bisection for almost every random regular graph with given bisection width.

In the last part of the thesis we describe a new algorithm which is found to perform well in practice, but we have no analysis for it. Finally, we describe a heuristic that when combined with other well-known algorithms such as Kernighan-Lin seems to improve the performance of these algorithms for small degree graphs.

Thesis Supervisor : F. Thomson Leighton
Title : Associate Professor of Applied Mathematics

To my parents

ACKNOWLEDGEMENTS

I would like to thank Tom Leighton who has helped me tremendously in the past few years. He has unselfishly contributed to this thesis. His quick insight and intuition have been very helpful to me, and I have learned a lot from working with him.

I also would like to thank the following individuals who have in one way or another influenced and helped me in my graduate study. Ron Rivest who guided me during my first few years at MIT and continues to be a source of support and encouragement. Charles Leiserson is always ready to listen to my problems, to offer advice, information and encouragement. Mike Sipser has always been helpful and ready to discuss my problems since my first days at MIT.

I thank Ron and Charles for having agreed to serve as my thesis readers, and for having to read the thesis in a very short span of time.

Part of Chapter 2 and Chapter 4 represent work done jointly with S. Chaudhuri, T. Leighton, and M. Sipser. This work appears in [Bu84] and [Bu86]. I would like to thank Soma Chaudhuri and Christopher Heigham for having programmed almost all of the algorithms tested in this thesis.

I would like to thank my friends in the MIT-VSA who have made my stay at MIT a more pleasant and memorable experience. In particular, I would like to thank Như-Ý for all the things that she has done for me.

I am grateful to my parents and my brothers for their constant love, support and encouragement.

TABLE OF CONTENTS

Chapter 1

Introduction

Let G be a $2n$-node undirected, simple graph. A *bisection* of G is a set of edges whose removal partitions G into two disjoint n-node subgraphs. A *minimum bisection* of G is a bisection with minimum cardinality. The cardinality of the minimum bisection is called the *bisection width* of the graph. The graph bisection problem is the problem of finding the minimum bisection of a graph.

The graph bisection problem is a special case of a more general problem, namely the graph partitioning problem. Given an undirected, simple graph G with a weight function on its edges and r a positive integer, the graph partitioning problem is the problem of finding a partition of the graph G into disjoint subsets each of size less than r such that the total weight of the edges having endpoints in different subsets of the partition is minimized. The graph partitioning problem serves as an abstraction for several problems such as program partitioning and printed circuit board layout in the natural way. It is, however, not easy to use this abstraction directly when the number of subsets is greater than two. The graph bisection problem on the other hand serves well as an abstraction because it fits better to the divide-and-conquer scheme. Perhaps the most visible application of graph bisection algorithms in recent years is in the VLSI placement and routing programs.

Engineering advances in recent years in the Very Large Scale Integration (VLSI) process have made possible the placing of hundreds of thousands of components on a single chip. Considering such a large number of components, it is essential to lay out these components and to route the wires connecting them efficiently. The main objective is to lay out these components in the smallest area subject to various constraints such as fabrication techniques and routability. The problem of minimizing the layout area is NP-complete even when there is no routing constraints [LaP80]. In practice a number of

efficient layout placement and routing heuristics are used. The typical programs for VLSI placement and routing ususally start by splitting a network in halves, recursively laying out each half, and then reinserting the wires connecting the two halves. It has been observed in practice that the quality of the final layout depends greatly on the number of wires that have to be reinserted in the last step. In fact, it has been proved recently that one can obtain a provably good layout algorithm if one has a provably good algorithm for splitting a network in halves [BL84]. It has also been observed that many divide-and-conquer based algorithms run much faster on graph with small bisection [Ba83][LT77].

Considering the wide-spread application of the graph bisection problem it is unfortunate that the problem is NP-complete [GJS76]. There are no known approximation algorithm for graph bisection, even for the case of planar graphs which always have bisection width $O(\sqrt{n})$ [LT79]. However, exact algorithms for graph bisection are known for special graphs such as trees and bounded width planar graphs.

Previous works on this problem have been focused on determining upper and lower bounds of the bisection width for various classes of graphs and on devising heuristics for bisecting graphs. No analysis of the behaviour of any of these well known heuristics are offered. Any hints of the performance of these heuristics are drawn from data collected in experiments or in practice. In this thesis we will try to overcome this deficiency by giving graph bisection algorithms which are provably good on the average for large classes of natural graphs. Perhaps the most important aspect of our work is the different approach that we take in devising the algorithms. In particular, we use global method instead of local optimization methods as in existing graph bisection algorithms. Our algorithms also differ from other graph bisection heuristics, as well as from many heuristics for other NP-complete problems, in several respects. Most notably:

(i) the algorithms provide exactly the minimum bisection for almost all input graphs with the specified form, instead of only an approximation of the minimum bisection,

(ii) whenever the algorithms produce a bisection, it is guaranteed to be optimal (i.e., the algorithms also produce a proof that the bisection they output is an optimal bisection),

(iii) the algorithms work well for graphs with small bisections (as opposed to graphs with large bisections, for which arbitrary bisections are nearly optimal).

(iv) the algorithms work well both theoretically and experimentally.

In addition we also analyze the performance of some well known heuristics on these

same classes of graphs. The analysis and data from our experiments indicate that our algorithms perform at least as well as or better than the well known heuristics.

The remainder of the thesis is divided as follows. In Chapter 2 we review the various models of random graphs that are commonly used in probabilistic analysis and we present a new model of random graphs that we argue to be better suited for the study of graph bisection. We then review the various well-known graph bisection heuristics in Chapter 3. In Chapter 4 we present our graph bisection algorithms which are based on the maxflow algorithm. Analysis of the performance of our algorithms will also be given. Chapter 5 provides the analysis of the performance of the greedy algorithm. We provide the data comparing the performances of these algorithms and also some new heuristics in Chapter 6. Chapter 6 also contains some remarks for the practitioner regarding those aspects of the thesis that might prove useful to them. The thesis concludes with our conclusions and the references.

Chapter 2

Models of Random Graphs

Even though there are several graph bisection algorithms which seem to perform well in practice, attempts at analyzing their performance on "any old graph" seems to be not useful in determining their true behavior and capability. It is, therefore, natural to restrict the analysis to graphs from special classes. Furthermore, even within a special class of graphs worst case analysis doesn't seem to be feasible either, at least with the known bisection algorithms. This leads to attempt at analyzing average behavior of graph bisection algorithms. To prove theorems about the average behavior of an algorithm we need a probability distribution over which the average is to be taken. For the case of graph bisection algorithms the natural choice is random graphs. However, there are several models of random graphs and not all of them are suitable for analyzing graph bisection algorithms. In this chapter we review the two most popular models and explain why they are not suitable to be used as inputs for analyzing and testing the behavior of graph bisection algorithms. We then present a new model of random graphs which we argue to be more suitable for testing and analyzing graph bisection algorithms. We also show that the graph bisection problem does not become easier when restricted to this class of graphs.

2.1. Model $\mathcal{G}(n,p)$

The study of random graphs was initiated by Erdös and Rényi in their seminal papers [ER59],[ER60]. Since then hundreds of papers have been written on the subject, and one of the most used models of random graphs is $\mathcal{G}(n,p)$. This class of graphs contains all simple graphs on n vertices, in which an edge between any two vertices is present with probability p independent of all other edges. The appeal of this model is that it is very

easy to work with due to the independence of the edges' existences and hence placing no restriction on the degree of the vertices in the graph. There are numerous results about this model, one of the most basic one is the following theorem which can be easily shown with the help of Chebyshev's inequality.

Theorem 2.1. *Given $\epsilon > 0$ and $p \in (0,1)$ fixed. Almost every graph in $\mathcal{G}(n,p)$ has at least $(p-\epsilon)n^2/2$ edges and at most $(p+\epsilon)n^2/2$ edges. If p is a function from N to $(0,1)$ such that $n^2p(n) \to \infty$ and $(1-p(n))n^2 \to \infty$ then we again have the same result.*

This theorem indicates that graphs in $\mathcal{G}(n,p)$ are usually very dense for fixed constant p. By our definition of bisection we can only consider graphs with an even number of vertices, and hence the following theorems will be stated with respect to graphs on an even number of vertices. The following theorem from [Bu83] showed that the bisection width of a graph in $\mathcal{G}(2n,p)$ is also very large and contain about half of the edges of the graph.

Theorem 2.2. [Bu83] *Let $f(n)$ be a function such that $f(n) = o(1)$ and $f(n) = \Omega(1/n)$. Let $p \in (0,1)$ be a fixed constant. Then almost every graph in $\mathcal{G}(2n,p)$ has bisection width greater than or equal to*

$$n^2p - n\sqrt{4npq\log 2 - 2pq\log n - 2pq\log f(n) + O(1)}$$

and less than or equal to

$$n^2p - \alpha n$$

for some $\alpha < \sqrt{pq}/2\pi$.

We note that the theorem is about graphs in $\mathcal{G}(2n,p)$, i.e., graphs on $2n$ vertices, not n vertices. For the case of $p = c/n$ for some $c > 1$, we have the following bounds on bisection width of graphs in $\mathcal{G}(2n,p)$.

Theorem 2.3. [Bu83] *Let $c > 9$ be a fixed constant, and $p = c/n$. Then almost every graph in $\mathcal{G}(2n,p)$ has bisection width greater than or equal to*

$$cn - n\sqrt{2c\log 2}\,(1+o(1))$$

and less than or equal to

$$cn - 2H(c)cn$$

where $H(c) \approx 0.238c^{-1/2}$.

It is not difficult to show that a random bisection of a graph in $\mathcal{G}(2n,p)$ will contain about half of the edges, and thus differs from optimal bisections in only low order terms. Therefore, this class of graphs may not serve well to distinguish between good heuristics and mediocre ones, either in practice or in analysis.

2.2. Model $\mathcal{G}(n,m)$

Another frequently used model of random graphs is $\mathcal{G}(n,m)$. This is a class of all graphs on n vertices having exactly m edges. This class of graph is turned into a probability space by assigning equal probability to each graph in the class. This class of graphs is closely related to the class $\mathcal{G}(n,p)$ for appropriate choice of p. In fact, results about graphs in the class $\mathcal{G}(n,p)$ can usually be translated into results for graphs in $\mathcal{G}(n,m)$ under appropriate conditions [Bo79]. As in the previous section we have the following bounds on the bisection width for graphs in $\mathcal{G}(2n,m)$.

Theorem 2.4. [Mac78] *Let $s \geq 9$ be fixed and $n \to \infty$. Almost every graph in $\mathcal{G}(2n,m)$ where $m = 2sn$ has bisection width greater than or equal to*

$$\left(\frac{1}{2} - \sqrt{\frac{\log 2}{2s}}\right) m$$

and less than or equal to

$$\left(\frac{1}{2} - H(s)\right) m$$

where $H(s) \approx 0.238 s^{-1/2}$ as $s \to \infty$.

Again as in the case of the $\mathcal{G}(2n,p)$ model the bisection width of a graph in $\mathcal{G}(2n,m)$ is about half the number of the edges in the graph, thus causing the same kind of problem for testing and analyzing the performance of graph bisection algorithms. In the next section we will present another model of random graphs which will prove to be more useful for our purpose.

2.3. Model $\mathcal{G}(n,d,b)$

Because graphs in $\mathcal{G}(n,p)$ and $\mathcal{G}(n,m)$ may not serve well to distinguish really good heuristics from mediocre or even horrible ones (e.g., heuristics that try to maximize the bisection), it is useful to examine graphs for which the minimum bisection is much smaller than the average bisections. Numerous papers (for the most part empirical studies) have attempted to do precisely this, but most end up constructing graphs according to a specified

procedure that (at best) imposes an upper bound on the bisection width of the constructed graphs. Unfortunately, it is usually not clear what relationship exists between the behavior of an algorithm on an average graph in such a class, and on an average graph with specified properties (such as fixed bisection width). Of course, it is the behavior of algorithms on graphs randomly selected from a class of the latter type that is of greatest interest. In this section we introduce a new model of random graphs which will overcome the problems of $\mathcal{G}(n,p)$ and $\mathcal{G}(n,m)$. We consider the class $\mathcal{G}(n,d,b)$ of labelled simple graphs that have $2n$ nodes, node degree d, and bisection width b for fixed $d \geq 3$ and $b = o(n^{1-1/\lfloor \frac{d+1}{2} \rfloor})$. In other words, we consider precisely the distribution of random $2n$-node, d-regular graphs conditioned on having minimum bisection b. Since every graph with dn edges has average bisection $dn/2$, the minimum bisection for these graphs is much smaller than the average bisection. Moreover, we will show that the graph bisection problem is NP-complete for $\mathcal{G}(n,d,b)$ whenever $b \geq n^{\epsilon}$ for any constant $\epsilon > 0$. Hence $\mathcal{G}(n,d,b)$ is a natural and suitable class of graphs for analysis.

2.3.1. Analyzing Random Graphs With Small Bisection Width

Methods of constructing d-regular graphs with uniform probability are well known [Bo80]. In what follows, we extend one such standard method to construct d-regular graphs with bisection width b with near uniform probability for $b = o(n^{1-1/\lfloor \frac{d+1}{2} \rfloor})$.

Step 1. Consider a set of $2n$ distinctly labelled nodes, and randomly designate half of them as left nodes , and half as right nodes. Then replace each node with d distinctly labelled points. (E.g., node 1 is replaced by points $1.1, 1.2, \ldots, 1.d$.)

Step 2. Randomly match b left points to b right points.

Step 3. Randomly match the remaining $dn - b$ left points among themselves and the remaining $dn - b$ right points among themselves.

Step 4. Coalesce each set of d points back into a node.

Step 5. Output the graph, maintaining the node and point labels.

Let $\mathcal{G}^*(n,d,b)$ be the collection of graphs (included according to multiplicity) that are constructed by the previous routine. At first glance it is not clear that $\mathcal{G}^*(n,d,b)$ has any relation at all to $\mathcal{G}(n,d,b)$. For example, $\mathcal{G}^*(n,d,b)$ contains graphs with multiple edges and loops as well as graphs with bisection width less than b. No such graphs are contained in $\mathcal{G}(n,d,b)$. Moreover, graphs in $\mathcal{G}(n,d,b)$ occur with varying frequencies in $\mathcal{G}^*(n,d,b)$, depending on the number of b-bisections in the graph and on the number of ways of labelling points.

Despite all of these obstacles, however, we prove a theorem later in this section that any condition that holds with probability $1 - o(1)$ for $\mathcal{G}^*(n,d,b)$ as $n \to \infty$ also holds with probability $1 - o(1)$ for $\mathcal{G}(n,d,b)$. This result is crucial to the thesis since it allows us to analyze the much simpler class $\mathcal{G}^*(n,d,b)$ in order to prove theorems about the more natural class $\mathcal{G}(n,d,b)$. Without such an indirect analysis, it is unlikely that we would be able to prove anything at all about graphs randomly selected from $\mathcal{G}(n,d,b)$. For example, no closed expression is known for the number of d-regular $2n$-node graphs (simple or otherwise), yet the number of graphs (counted according to multiplicity) contained in $\mathcal{G}^*(n,d,b)$ is easily calculated.

Before proving the main theorem of this section , however, we need several lemmas. These lemmas highlight some of the more interesting properties of graphs in $\mathcal{G}(n,d,b)$ and $\mathcal{G}^*(n,d,b)$, and will be used throughout the thesis. We start with a lemma concerning random pointwise-labelled d-regular graphs (possibly with multiple edges and loops). Such graphs are generated in much the same fashion as graphs in $\mathcal{G}^*(n,d,b)$. The term *pointwise-labelled* refers to the existence of labelled points at each node, as in Step 1 of the procedure for $\mathcal{G}^*(n,d,b)$.

Lemma 2.5. *There is a constant $c > 0$ such that for all $d \geq 3$, $n \to \infty$ and almost every pointwise-labelled n-node d-regular graph G, every k-node subset S of G (for all $k \leq n/2$) is incident to at least cdk edges that connect nodes in S to nodes in $G - S$.*

Proof: Let $M(dn)$ denote the number of pointwise-labelled, n-node, d-regular graphs. It is easily seen that

$$M(dn) = \binom{dn}{\frac{dn}{2}} (dn/2)!\, 2^{-dn/2}.$$

The number of pointwise-labelled n-node d-regular graphs that have a k-node subset with exactly t connections to the rest of the graph is at most

$$\binom{n}{k}\binom{dk}{t}\binom{dn-dk}{t} t! M(dk-t) M(dn-dk-t).$$

Taking the ratio of the two formulas and simplifying, we find that the probability that such a graph has a k-node subset with only $t = cdk$ connections is

$$O\left(\left[c^c(1-c)^{(1-c)/2}\alpha^{(1-c)/2-1/d}\left(1-\frac{c}{\alpha}\right)^{(\alpha-c)/2}\left(1+\frac{1}{\alpha}\right)^{(1+\alpha)(1/2-1/d)}\right]^{-dk}\right)$$

where $\alpha = (n-k)/k \geq 1$. For $c \leq 1/4, d \geq 3$ and $\alpha \geq 1$, this is

$$O\left(\left[c^c(1-c)^{(1-c)/2}e^{-c/2}e^{1/6}\left(\frac{n-k}{k}\right)^{1/24}\right]^{-3k}\right).$$

Since $c^c(1-c)^{(1-c)/2}e^{-c/2} \to 1$ as $c \to 0$, we can conclude that

$$c^c(1-c)^{(1-c)/2}e^{-c/2}e^{1/6} \geq 1+\delta$$

for all sufficiently small c and some constant $\delta > 0$. Hence for small enough $c > 0$, the probability that a pointwise-labelled n-node d-regular graph has a k-node subset with less than cdk connections is

$$O\left(cdk\left[(1+\delta)\left(\frac{n-k}{k}\right)^{1/24}\right]^{-3k}\right).$$

It is easily checked that the preceding expression converges to 0 as $n \to \infty$ for all $1 \leq k \leq n/2$. In fact, the sum of these terms for $1 \leq k \leq n/2$ is $O(n^{-1/8})$ which also converges to 0. Thus the claim holds simultaneously for all k-node subsets in almost every graph.∎

Results such as Lemma 2.5 are common in probabilistic graph theory and have numerous useful applications. We include one such application in the following corollary. Although the result is only stated for pointwise-labelled graphs, possibly containing loops and multiple edges, it is easily extended to simple labelled d-regular graphs.

Corollary 2.6. *For all $d \geq 3$, $n \to \infty$ and almost every pointwise-labelled n-node d-regular graph G, every bisection of G has size between $(1-\epsilon)dn/4$ and $(1+\epsilon)dn/4$ where $\epsilon = O(1/\sqrt{d})$.*

Proof: Setting $k = n/2$ and $\alpha = 1$ in the proof of Lemma 2.5, we find that the probability that G has a bisection of size $cdn/2$ is

$$O\left(\left[c^c(1-c)^{1-c}2^{1-2/d}\right]^{-dn/2}\right).$$

This expression is maximized at $c = 1/2$ and thus the probability that G has a bisection of size less than $(1-\epsilon)dn/4$ or greater than $(1+\epsilon)dn/4$ is

$$O\left(n\left[\left(\frac{1+\epsilon}{2}\right)^{(1+\epsilon)/2}\left(\frac{1-\epsilon}{2}\right)^{(1-\epsilon)/2}2^{1-2/d}\right]^{-dn/2}\right).$$

Simplifying, we find that the preceding expression is

$$O\left(n\left[e^{\epsilon^2/2}2^{-2/d}\right]^{-dn/2}\right)$$

which tends to 0 for $\epsilon > 2/\sqrt{d\log e}$.∎

Of more immediate concern to us in this thesis is the following corollary to Lemma 2.5. In the corollary and throughout the rest of the thesis, the phrase *left or right half of a graph in* $\mathcal{G}^*(n,d,b)$ refers to the left or right, respectively, nodes created in Step 1 of the procedure for $\mathcal{G}^*(n,d,b)$ and to the edges inserted in Step 3, but does not include the bisection edges inserted in Step 2.

Corollary 2.7. *There is a constant $c > 0$ such that for all $d \geq 3$, $n \to \infty$ and almost every graph G that forms the left or right half of a graph in $\mathcal{G}^*(n,d,b)$, every m-node subset S of G that is incident to t bisection edges, is also incident to at least $cdm - t$ edges connecting S to $G - S$ for all $m \leq n/2$ and $t \leq b$.*

Proof: For simplicity, assume b is even and randomly connect the b bisection points in G with $b/2$ edges to form a new graph G'. It is easily observed that G' is a random d-regular pointwise-labelled graph, possibly containing loops or multiple edges. Hence, if G' is one of the $1 - o(1)$ portion of d-regular graphs satisfying Lemma 2.5, there will be at least cdm edges connecting S to $G' - S$. In that case, there clearly must be at least $cdm - t$ edges connecting S to $G - S$ in G.∎

The following lemmas will serve to further strengthen Corollary 2.7.

Lemma 2.8. *Given any $r \geq 2$, $d \geq 3, m = o(n^{1-1/r})$ and $n \to \infty$, if m items are chosen at random from n groups of d items each, then with probability $1 - o(1)$ fewer than r items will be selected from each group. Moreover, the same conclusion holds provided that each item is selected at random from some (possibly varying) subset of at least $n - m$ groups.*

Proof: Assume that each item is selected at random from some subset of at least $n - m$ groups. The probability that the ith item selected comes from the jth group is at most $d/[(n-m)d - m]$ since there are at least $n - m$ groups of d items to choose from and at most $i - 1 < m$ of the items have already been chosen. Hence, the probability that k items are chosen from the same group is at most

$$n\binom{m}{k}\frac{1}{(n - m - \frac{m}{d})^k}.$$

Simplifying, we find that this probability is $O\left(\frac{m^k}{n^{k-1}}\right)$ which for $m = o(n^{1-1/r})$ is $o(n^{1-k/r})$. Summing over $k \geq r$, we find the probability that fewer than r items are selected from each group is $1 - o(1)$.∎

Lemma 2.9. *For all fixed $d \geq 3$, all $b = o(n^{1-1/\lfloor\frac{d+1}{2}\rfloor})$, $n \to \infty$ and almost every graph G that forms the left or right half of a graph in $\mathcal{G}^*(n,d,b)$, every subset of G with at most $n/2$ nodes that is incident to k bisection edges for any $k \leq b$, is also incident to at least $k+1$ edges connecting nodes in S to nodes in $G - S$.*

Proof: Let G be the left half (without loss of generality) of a graph constructed according to the procedure for $\mathcal{G}^*(n,d,b)$. In what follows, we will show that the nodes of G which are incident to bisection edges have sufficiently bushy neighborhoods so that any set S incident to k bisection edges and at most k edges that connect nodes in S to nodes in $G - S$ must contain at least $4k/cd$ nodes, where c is the constant defined in Corollary 2.7. We will then use Corollary 2.7 to obtain a contradiction of the hypothesis that S is incident to at most k edges which link S to $G - S$.

Without loss of generality, we can assume that the edges created in Step 3 to form G were generated in order of increasing distance from the bisection edges. In particular, we are interested in the generation of edges within distance $d\log(1/c)$ of the bisection where c is the constant defined in Corollary 2.7. For fixed d, there are at most $m = O(b) = o(n^{1-1/\lfloor\frac{d+1}{2}\rfloor})$ such edges. Applying Lemma 2.8 to the node by node generation of edges to form G, it is easily shown that, with high probability each node of G within distance $d\log(1/c)$ is incident to fewer than $\lfloor\frac{d+1}{2}\rfloor$ previously generated edges (i.e., edges that are also incident to previously processed nodes).

Let S be a set of at most $n/2$ nodes of G that is incident to k bisection edges and at most k edges that connect S to $G - S$. Let e_i denote the number of edges in S that link two nodes which are of distance i from the bisection and $e_{i,i+1}$ denote the number of edges in S that link nodes which are of distance i and $i+1$ from the bisection. (Throughout this proof, *distance* means the length in edges of the shortest path totally contained in S to the bisection. Nodes incident to bisection edges are considered to be of distance 1 from the bisection.) By definition, $e_{0,1} = k$. Also define n_i to be the number of nodes in S at distance i from the bisection, and f_i to be the number of edges that link a distance i node of S to $G - S$. By assumption $f_0 = 0$ and $\sum_{i=0}^{\infty} f_i \leq k$.

Because the edges of S were generated in order of increasing distance from the bisection, and since each node is incident to at most $(d-1)/2$ previously generated edges,

we can deduce that

$$n_i \geq \frac{e_{i-1,i} + e_i}{\frac{d-1}{2}} \qquad (*)$$

and that

$$e_{i,i+1} \geq n_i \left(\frac{d+1}{2}\right) - e_i - f_i$$

for every $i \leq d\log(1/c)$. Combining the two inequalities, we find that

$$\begin{aligned} e_{i,i+1} &\geq \left(1 + \frac{2}{d-1}\right) e_{i-1,i} + \frac{2}{d-1} e_i - f_i \\ &\geq \left(1 + \frac{2}{d-1}\right) e_{i-1,i} - f_i. \end{aligned}$$

It is not difficult to show that $e_{i,i+1}$ is minimized for every $i \leq d\log(1/c)$ by setting $f_1 = k$ and $f_i = 0$ for $i \geq 2$. Then it is clear that

$$\begin{aligned} e_{i,i+1} &\geq \left(1 + \frac{2}{d-1}\right)^{i-1} \left(\left(1 + \frac{2}{d-1}\right) e_{0,1} - k\right) \\ &= \frac{2k}{d-1} \left(1 + \frac{2}{d-1}\right)^{i-1} \qquad (**). \end{aligned}$$

Hence for $i \leq d\log(1/c)$ using $(*)$ and $(**)$ we get $\sum_i n_{i+1} > 4k/(c(d-1))$ and hence S contains at least $4k/(c(d-1))$ nodes. By Corollary 2.7, however, this means that there are at least $cd\left(\frac{4k}{c(d-1)}\right) - k \geq 3k$ edges linking S to $G - S$. This provides the necessary contradiction and concludes the proof.∎

We are now able to prove the main result of this section.

Theorem 2.10. *For all fixed $d \geq 3$ and all $b = o(n^{1-1/\lfloor\frac{d+1}{2}\rfloor})$, any condition that holds with probability $1 - o(1)$ for $\mathcal{G}^*(n,d,b)$ as $n \to \infty$ also holds with probability $1 - o(1)$ for $\mathcal{G}(n,d,b)$ as $n \to \infty$.*

Proof: We first observe that every graph G in $\mathcal{G}(n,d,b)$ that has a unique minimum bisection is generated with the same frequency in $\mathcal{G}^*(n,d,b)$. This is because the b edges inserted in Step 2 of the procedure for $\mathcal{G}^*(n,d,b)$ must then be precisely the edges in the unique b-bisection of G. Since the left and right halves of G are connected, the halves and the edges they contain are also distinguished. Finally, since G contains no multiple edges or loops, there are exactly $d!$ ways to pointwise label the d edges incident to each node. Hence there are $(d!)^{2n}$ pointwise labellings for each labelled graph.

Graphs in $\mathcal{G}(n,d,b)$ with nonunique b-bisections appear proportionally more often in $\mathcal{G}^*(n,d,b)$ than do graphs with unique b-bisections. Moreover, $\mathcal{G}^*(n,d,b)$ also contains

graphs with multiple edges and loops, and with bisection width less than b. However, we will show in what follows that these bad graphs constitute at most a constant fraction of the graphs in $\mathcal{G}^*(n,d,b)$. We commence by showing that $\Omega\left(e^{-2(d-1)^2}\right)$ of the graphs generated for $\mathcal{G}^*(n,d,b)$ have no loops or multiple edges. For fixed d, this is a constant fraction.

The probability of generating a multiple edge during an insertion of a bisection edge in Step 2 is at most

$$b\frac{(d-1)^2}{(dn-b)^2} \leq \frac{b}{n^2} \leq \frac{1}{n}.$$

Hence the probability of avoiding multiple edges altogether during Step 2 is at least

$$\left(1-\frac{1}{n}\right)^b \geq 1-\frac{b}{n} = 1-o(1).$$

The probability of generating a multiple edge or loop at any fixed point of Step 3 (conditioned only on the knowledge that no multiple edges or loops were generated previously – not on the actual edge selections made) is at most $(d-1)^2/dn$. Hence the probability that none of the dn edge insertions create loops or multiple edges is at least

$$\left(1-\frac{(d-1)^2}{dn}\right)^{2dn} = \Omega\left(e^{-2(d-1)^2}\right).$$

We conclude the proof by showing that only a small portion of the graphs occurring in $\mathcal{G}^*(n,d,b)$ have bisections less than b or multiple bisections of size b. This fact is an immediate consequence of Lemma 2.9, since the existence of such a bisection for a graph G in $\mathcal{G}^*(n,d,b)$ would imply the existence of a subset S with at most $n/2$ nodes in the left or right half of G that is incident to k bisection edges for some $k \leq b$ but to k or fewer other edges that link S to $G-S$.

In conclusion, sampling graphs in $\mathcal{G}(n,d,b)$ is equivalent to sampling a constant portion $\Omega\left(e^{-2(d-1)^2}\right)$ of the graphs in $\mathcal{G}^*(n,d,b)$ and ignoring point labels. Hence, any condition that holds for $1-o(1)$ of the graphs in $\mathcal{G}^*(n,d,b)$ must also hold for $1-o\left(e^{-2(d-1)^2}\right) = 1-o(1)$ of the graphs in $\mathcal{G}(n,d,b)$.∎

2.3.2. NP-completeness Of Graph Bisection In $\mathcal{G}(n,d,b)$

In this section we will show that the problem of deciding whether or not a d-regular graph has a bisection of size b or less is NP-complete whenever $d \geq 3$ and $b = n^\epsilon$ for any fixed ϵ in the range $(0,1)$. We will reduce the general graph bisection problem to this problem. The proof will be done in two steps. Given a graph G and an integer b, we

transform G to a 3-regular graph G^* such that G has a bisection of size b or less if and only if G^* has a bisection of size b or less. We next transform G^* into a d-regular graph G' such that G^* has a bisection of size b or less if and only if G' has a bisection of size b' or less, where $b' = n^\epsilon$, where n is the size of G', for any fixed $\epsilon \in (0,1)$. We start with the following lemma.

Lemma 2.11. *Let H be an n-node honeycomb-like 3-regular graph as in Figure 1. Every s-node subset S of H, where $s \leq n/2$, is adjacent to at least $\sqrt{s/2}$ nodes not in S.*

The proof of this lemma is not difficult and we will omit it.

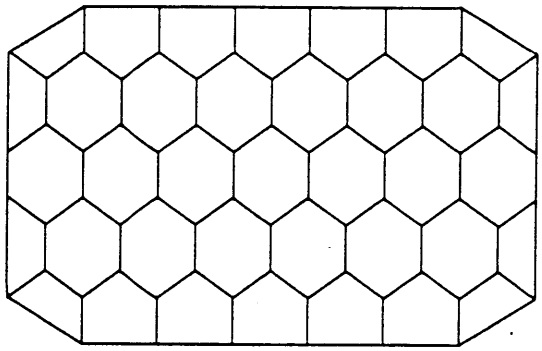

Figure 1. An example of a honeycomb-like graph.

Theorem 2.12. *The problem of deciding whether or not a d-regular graph has a bisection of size b or less is NP-complete, whenever $d \geq 3$ and $b = n^\epsilon$ for any fixed $\epsilon \in (0,1)$.*

Proof: Let G be an n-node graph and b an integer. We will construct a 3-regular graph G^* on $m = \Theta(n^6)$ nodes as follows. Replace each node of G with an n^5-node honeycomb-like graph H of Lemma 2.11. An edge between two nodes in G is replaced by an edge connecting two edges of the two corresponding graphs H in G^*, thus creating two new nodes of degree 3 (see Figure 2). Furthermore, edges coming into a graph H are dispersed widely so that any r-subset R of H which is incident to s incoming edges will also incident to at least $s + \sqrt{r/4}$ edges in $H - S$. This can be done using Lemma 2.11.

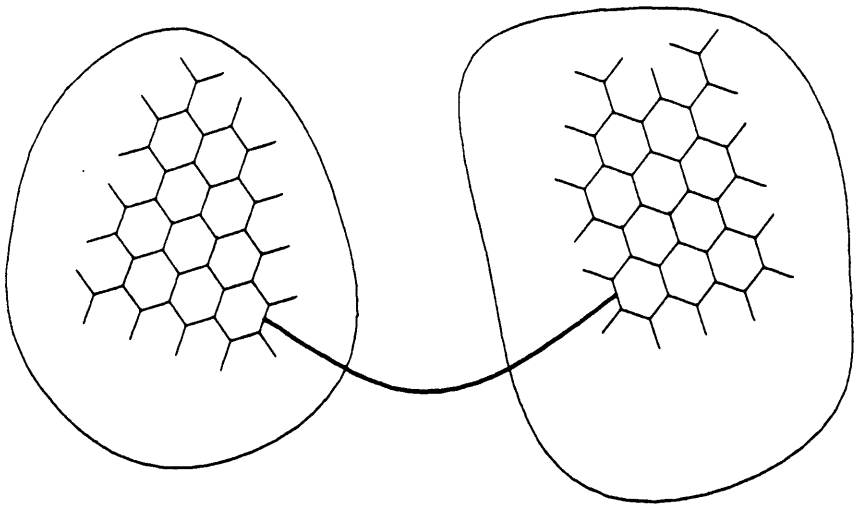

Figure 2. The connection of two H-graphs.

Let B be a minimum bisection of G^*, i.e., B is a minimum set of edges whose removal divides G^* into two subgraphs of equal size.

Claim 1: B induces a corresponding bisection of G, i.e., B contains only edges that correspond to the original edges of G.

Suppose not, then we can rearrange the bisection to obtain a new cut by moving each copy of H that is cut by B entirely to the side of the bisection containing the majority of its nodes. Suppose we have to move t nodes, then the new cut has at least $\sqrt{t/4}$ fewer edges than the original bisection. This new cut, however, might no longer be a bisection. To make it a bisection we have to move at most t/n^5 H-graphs. This will increase the size of the cut by at most t/n^4 edges. Since the size of any bisection of G is at most n^2, it can be easily seen that t is at most n^4. Thus the new bisection is smaller than the original one, a contradiction. This proves the claim.

Hence B is also a minimum bisection of G.

The next step is to transform G^* into a d-regular graph G' as follows. Replace each edge of G^* with k edges where k is such that $bk = (100mk)^\epsilon$ and $100k$ is a square. We then replace each node of G^* with a graph H' satisfying the following conditions:

(i) H' has $3k$ degree $d-1$ nodes and $97k$ degree d nodes, and

(ii) every subset of H' with $r \leq 50k$ nodes, s of which are degree $d-1$ nodes, is incident to at least $s + 1.3r/40$ nodes not in the subset.

Graphs like H' are easily constructed from expander graphs, even for $d = 3$.

Edges in G^* now can be replaced by edges connecting the $(d-1)$-degree nodes in the corresponding H' graphs. The resulting graph G' will then be d-regular.

Claim 2: G' has a bisection of size bk or less if and only if G^* has a bisection of size b or less.

The proof is similar to before. Given a bisection of G', form a new cut by moving each copy of H' entirely to the side of the bisection containing the majority of its nodes. If t nodes are moved in this step then the new cut contains at least $1.3t/40$ fewer edges than the original bisection. Although the new cut corresponds nicely to a cut of G^*, it is not necessarily a bisection. To make a bisection, move up to $t/(100k)$ copies of H' from one side to the other. This increases the cut by at most $3t/100$ edges which is less than the $1.3t/40$ edges decrease performed earlier.

Thus a bk-bisection of G' can be converted into a b-bisection of G. This proves Claim 2 and the theorem. ∎

Chapter 3

Classical Approaches

Previous works on the graph bisection problem have been focused on devising efficient heuristics for bisecting graphs. Some of these heuristics and their variations have been used widely in practice, even though no theoretical analysis of their performances are available. In this chapter we will review some of the most well-known graph bisection algorithms. The common feature of all these algorithms is that they are iterative improvement algorithms, where the improvements are achieved by local optimizing operations. Typically, such an algorithm starts with a random bisection of the graph. It then tries to improve upon this bisection by doing some local operation such as 2-change (the interchanging of a pair of vertices). The actual method for this step varies with each algorithm. This step is then repeated until no more improvement can be made. This constitutes one pass of the algorithm. The algorithm usually begins another pass with the starting bisection being the one found in the previous pass. This process repeats for a fixed number of passes or until no more improvement is made. Another approach is for the algorithm to start each pass, for a fixed number of passes, with a new random bisection and report the best solution found. In the following we will describe the greedy algorithm, the well-known Kernighan-Lin algorithm and the recently discovered simulated annealing algorithm.

To facilitate our discussion we will first define some terminologies. Let $G = (V, E)$ be a graph on $2n$ vertices and let (A, B) be a bisection of G. We denote the cardinality of the bisection (A, B) by $|(A, B)|$. With respect to that bisection we define the *gain* of each vertex in the graph as follows. For $a \in A$, the *gain* of a (denoted by g_a) is the difference between the number of edges connecting a to vertices in B and the number of edges connecting a to vertices in A, i.e.,

$$g_a = |\{ v \in B \mid (a, v) \in E \}| - |\{ v \in A \mid (a, v) \in E \}|$$

We extend this definition to pairs of vertices, one in A and one in B. More formally, define

$$g_{a,b} = |(A,B)| - |(A',B')|$$

where

$$A' = (A - \{a\}) \cup \{b\} \text{ and } B' = (B - \{b\}) \cup \{a\}$$

In other words, $g_{a,b}$ is the reduction in the size of the bisection when a and b are interchanged. Clearly,

$$g_{a,b} = g_a + g_b - 2\delta(a,b)$$

where

$$\delta(a,b) = \begin{cases} 1, & \text{if } (a,b) \in E; \\ 0, & \text{if } (a,b) \notin E. \end{cases}$$

3.1. The Greedy Algorithm

The first graph bisection algorithm that we will describe is the greedy algorithm, it is perhaps the simplest graph bisection algorithm. There are several versions of this algorithm, but we will consider only the following greedy algorithm. The algorithm has several passes, each pass tries to improve the result of the previous pass. The algorithm starts with a random bisection. At each step in one pass of the algorithm, two vertices will be chosen, one in each side of the bisection. These vertices are chosen in such a way that when they are interchanged they will yield the largest positive reduction in the size of the bisection. Ties are broken arbitrarily. If such a pair is found the vertices are then interchanged. Once a vertex has been chosen to be exchanged it will not be chosen again. A pass is finished when there is no pair that will give a positive reduction in the size of the bisection. That is, the algorithm keeps making downhill moves and it stops as soon as it has to make an uphill move. The algorithm can be run for a fixed number of passes or until no more improvement can be made.

The running time of this algorithm can be easily found to be $O(n^2 \log n)$. That is the worst case running time of the algorithm, on the average the greedy algorithm will stop much sooner. It is difficult, however, to determine the average running time of the algorithm without further assumption on the inputs. In practice, to save time each vertex in the pair is chosen independently, i.e., each vertex is the best choice in each half but as a pair they may not be the best choice over all pairs. By doing this we reduce the running time significantly, and experience shows that it does not affect the performance

begin
1. Compute g_a, g_b for each $a \in A, b \in B$.
2. $Q_A = \emptyset, Q_B = \emptyset$, STOP = FALSE.
3. **while** not STOP **do**
   begin
4. Choose $a' \in A - Q_A$ and $b' \in B - Q_B$ such that $g_{a',b'}$ is maximum over all choices of a and b.
5. **if** $g_{a',b'} \leq 0$, set STOP = TRUE
6. **else**
7. Set $Q_A = Q_A \cup \{a'\}, Q_B = Q_B \cup \{b'\}$
8. **for each** $a \in A - Q_A$ **do**
9. $g_a = g_a + 2\delta(a, a') - 2\delta(a, b')$
10. **for each** $b \in B - Q_B$ **do**
11. $g_b = g_b + 2\delta(b, b') - 2\delta(b, a')$
   end
12. Return new bisection $((A - Q_A) \cup Q_B, (B - Q_B) \cup Q_A)$.
end

Figure 3.1. One pass of the greedy graph bisection algorithm.

of the algorithm very much. Even for this simple algorithm no analysis of its performance is available.

3.2. The Kernighan-Lin Algorithm

Kernighan and Lin in [KL70] gave a heuristic for solving the graph bisection problem which seems to work well in practice. It and its variations are the most widely used graph bisection algorithms. Let a graph $G = (V, E), V = 2n$ be given. The main idea here is the same as before, that is to start with an arbitrary bisection, say (A, B), and improve upon it. The improvement is accomplished by interchanging subsets $X \subset A, Y \subset B$, and $|X| = |Y| < n$ so that the size of the bisection is decreased. Clearly, there exist equal sized subsets X and Y of A and B, respectively, such that when X and Y are interchanged we will obtain an optimal bisection, but there is no known efficient way of finding them short of exhaustive search. The Kernighan-Lin heuristic finds these subsets approximately by choosing elements of X and Y sequentially. This choosing process is done as follows. For each element $a \in A, b \in B$, let $g_{a,b}$ be defined as before. The algorithm first computes $g_{a,b}$ for all $a \in A, b \in B$. It then chooses $a_1 \in A, b_1 \in B$ such that

$$g_{a_1,b_1} = \max\{ g_{a,b} \mid a \in A,\ b \in B \}$$

begin

1. Compute g_a, g_b for each $a \in A, b \in B$.
2. $Q_A = \emptyset, Q_B = \emptyset$.
3. **for** $i = 2$ **to** n **do**
   begin
4. Choose $a_i \in A - Q_A$ and $b_i \in B - Q_B$ such that g_{a_i,b_i} is maximum over all choices of a and b.
5. Set $Q_A = Q_A \cup \{a_i\}, Q_B = Q_B \cup \{b_i\}$
6. **for each** $a \in A - Q_A$ **do**
7. $g_a = g_a + 2\delta(a, a_i) - 2\delta(a, b_i)$
8. **for each** $b \in B - Q_B$ **do**
9. $g_b = g_b + 2\delta(b, b_i) - 2\delta(b, a_i)$
   end
10. Choose $k \in \{1, \ldots, n\}$ to maximize $\sum_{i=1}^{k} g_{a_i,b_i}$.
11. Interchange the subsets $\{a_1, \ldots, a_k\}$ and $\{b_1, \ldots, b_k\}$ to get the new bisection.

end

Figure 3.2. One pass of the Kernighan-Lin graph bisection algorithm.

The algorithm now updates the gains of all vertices in V, except a_1 and b_1, with respect to the new bisection $((A - \{a_1\}) \cup \{b_1\}, (B - \{b_1\}) \cup \{a_1\})$. The algorithm next repeats the process for this new bisection and chooses a new pair of vertices to be exchanged, except that a_1 and b_1 will not be considered anymore in choosing the next pair that will give the maximum reduction. That is, once a vertex is chosen to be exchanged it will no longer be considered in later steps. The process is repeated untill all vertices have been considered. We now have a list of pairs $(a_1, b_1), \ldots, (a_n, b_n)$. Of course, if all these pairs are interchanged the total reduction is zero. The algorithm now chooses a $k < n$ such that the interchange of the subsets $\{a_1, \ldots, a_k\}$ and $\{b_1, \ldots, b_k\}$ will give a maximum reduction over all choices of $k < n$. This whole process makes up a pass of the algorithm. The algorithm can have several passes. Each pass, except the first one, starts with the bisection given as the result of the previous pass. The algorithm can have a fixed number of passes or it can run until no more improvement is possible. The main difference between this algorithm and the greedy algorithm is that Kernighan-Lin will accept uphill moves in the hope of finding a smaller bisection later on. As in the case of the greedy algorithm, the choices of a_i and b_i in Step 4 of Figure 3.2 are made independently to save time in practice. It is also observed that the performance of the algorithm is not greatly affected by doing that.

We now describe the algorithm formally. Let $G = (V, E)$ be a graph with $V = 2n$. Let (A, B) be a bisection of G. For each $a \in A$ $b \in B$ define $g_{a,b}$ as before. The heuristic is shown in Figure 3.2. Steps 7 and 9 of the algorithm can be easily checked that the values of the g_a and g_b are correctly updated with respect to the new bisection, that is after the sets $\{a_1, \ldots, a_i\}$ and $\{b_1, \ldots, b_i\}$ have been interchanged. It is also easily shown that the running time of the algorithm is $O(n^2 \log n)$.

Variations of the Kernighan-Lin's algorithm have been considered by Macgregor [Mac78], he also considered hybrid of Kernighan-Lin algorithm with other heuristics. A slight variation of the Kernighan-Lin heuristic has also been implemented by Fiduccia and Mattheyses [FM82] to run in linear time by using some clever data structures.

3.3. Simulated Annealing Graph Bisection Algorithm

In this section we present an algorithm proposed by Kirkpatrick, *et al.*, [KGV82], which makes an interesting connection between the annealing process and the iterative improvement process of the graph bisection heuristics. They actually presented a general scheme which mirrors the annealing process of materials and can be adapted to solve problems that are susceptible to an iterative improvement algorithm such as the graph bisection problem. In fact, experiments performed by Johnson, *et al.* on a variety of cominatorial optimization problems indicate that among the problems they considered, which include the traveling salesman problem, the graph coloring problem and the number partitioning problem, simulated annealing did well only on the graph bisection problem.

Consider a system consisting of a large number of atoms, such as a sample of liquid or solid matter. The aggregate behavior of the system can be observed by considering the average behavior taken over an ensemble of identical systems. We associate with each configuration of the system in the ensemble a Boltzmann's probability, $\exp(-E(\{r_i\})/k_B T)$, where $E(\{r_i\})$ is the energy of the configuration defined by the atomic positions $\{r_i\}$, k_B is the Boltzman's constant, and T is the temperature. One wishes to know what happens to the system in the limit of low temperature, for instance, whether atoms remain fluid or solidify. It is known that ground states and configurations having energy close to them are very rare, nonetheless, they dominate the behavior of the system at low temperature because as T is lowered the Boltzmann distribution collapses into the lowest energy state or states. To find the low temperature states of a system it is necessary to use an annealing process. That is to first melt the substance, then lower the temperature slowly, and spend a long time at the temperatures near the freezing point. Otherwise, the resulting

configuration will be metastable.

There is a simple algorithm given by Metropolis, *et al.* [M53] which simulates a collection of atoms in equilibrium at a given temperature. In each step of the algorithm, an atom is given a small random displacement, and the corresponding change in energy, ΔE, of the system is computed. If $\Delta E \leq 0$, then the displacment is accepted, and the configuration with the just displaced atom is used as the starting configuration for the next step. When $\Delta E > 0$, the configuration is accepted with probability $\mathbf{Pr}(\Delta E) = \exp(-\Delta E/k_B T)$. A random number is chosen uniformly in the interval $(0,1)$, and compared with $\mathbf{Pr}(\Delta E)$. If it is less than $\mathbf{Pr}(\Delta E)$ then the new configuration is accepted, if not, the original configuration is retained and we repeat the process.

It is observed in [KGV82] that the iterative improvement process in a combinatorial optimization problem such as the graph bisection problem is similar to the microscopic rearrangement processes modelled by statistical mechanics, where an appropriate cost function for the graph bisection problem will play the role of energy. Using this analogy we note that in the process of finding the solution if we only accept rearrangements that reduce the cost function, then this is like rapid quenching from high temperature to $T = 0$, thus the resulting solutions will often be local optima and metastable. By utilizing the Metropolis' algorithm described above, in which rearrangements that increase the cost function are sometimes accepted, we can expect to get better solutions as indicated by the observation made in actual physical processes.

We will now describe the simulated annealing algorithm for graph bisection that was used by Johnson in [J84]. The algorithm starts by creating a random partition of the vertex set into two sets that are not necessarily equal. The algorithm will then proceed through a "cooling" process. The rate of cooling is controlled by the variable TEMP-FACTOR, which is set to 0.95. At each temperature the algorithm will perform a number of steps determined by the product of the variables EPOCH-SCALE and N, where EPOCH-SCALE is set to be 16, and N is the size of the graph. For each step, the algorithm randomly picks a neighbor of the current partition of the graph, where a neighbor of a partition is defined to be another partition that is obtained from the original partition by moving one vertex from one side of the partition to the other. The algorithm then computes the change Δ in the cost of the two partitions, where the cost of a partition (V_1, V_2) is defined as

$$\text{Cut-Size}(V_1, V_2) + SCALE \times (|V_1, V_2|)^2,$$

where $SCALE = 0.1$. If $\Delta \leq 0$ the current partition is set to be the new partition,

otherwise the current partition is set to be the new partition with probability $e^{-\Delta/T}$. The cooling process stops when a reduction in the temperature does not yield an improvement. By then we have a partition of the graph which is not necessarily a bisection. A bisection can be obtained from this partition by performing a greedy heuristic, that is by balancing the partition in a greedy manner. In other words, we move vertices one by one from the larger side of the partition to the smaller side in such a way that the cut size increases the least.

begin
1. Randomly partition V into two not necessarily equal sets V_1 and V_2.
2. Set initial temperature $T = 1$.
3. Until a decrease in temperature yields no improvement do
4. **begin**
5. From 1 to $EPOCH\text{-}SCALE \times N$ do
6. **begin**
7. Randomly pick a neighbor $S' = (V_1', V_2')$ of the current partition $S = (V_1, V_2)$.
8. Compute Cost(S) = Cut-Size$(V_1, V_2) + SCALE \times (|V_1| - |V_2|)^2$
9. Compute Cost(S') = Cut-Size$(V_1', V_2') + SCALE \times (|V_1'| - |V_2'|)^2$
10. Compute $\Delta = Cost(S') - Cost(S)$
11. **if** $\Delta \leq 0$ **then** set $S = S'$
12. **if** $\Delta > 0$ **then** set $S = S'$ with probability $e^{-\Delta/T}$
13. **end**
14. Set $T = T \times TEMP\text{-}FACTOR$
15. **end**

end

Figure 3.3. Simulated Annealing Graph Bisection Algorithm.

Chapter 4

Maxflow-based Graph Bisection Algorithms

All the known algorithms share a basic property, namely they all use some local optimization method to improve the current bisection. In this chapter, we consider a totally different approach to the graph bisection problem by using global method such as network flow. In particular, we describe a graph bisection algorithm that finds the minimum bisection for almost every graph G in $\mathcal{G}(n,d,b)$ for fixed $d \geq 3$ and $b = o(n^{1-1/\lfloor\frac{d+1}{2}\rfloor})$. In addition, the algorithm is constructed so that every time a bisection is output, it is guaranteed to be optimal. The description of the algorithm and its analysis is divided into three sections. In Section 4.1, we present and analyze a simple algorithm for graphs with $o(\sqrt{n})$ bisections. The general algorithm is described and analyzed in Section 4.2. In Section 4.3, we bound the running time of the algorithms.

The idea of the algorithm is quite simple: we wish to convert G into an instance of the maxflow problem for which the mincut is the minimum bisection. Of course, it's hard to do this without knowing which edges comprise the minimum bisection, but we can come close. In fact, we will find that by replacing the neighborhoods around two nodes u and v with an infinite capacity source and sink, the resulting flow problem will often have a mincut close to a bisection. By exploiting this phenomenon, we are able to prove the desired result.

Throughout this chapter and for the rest of the thesis, we will state and prove "almost all"-type theorems for graphs in $\mathcal{G}^*(n,d,b)$. By Theorem 2.10, such results also hold for graphs in $\mathcal{G}(n,d,b)$. We start by proving one such result for the size of neighborhoods around nodes in the left and right halves of graphs in $\mathcal{G}^*(n,d,b)$.

Lemma 4.1. *For all fixed $d \geq 3$, all $b = o(n^{1-1/\lfloor \frac{d+1}{2} \rfloor})$, $n \to \infty$ and almost every graph G that forms the left or right half of a graph in $\mathcal{G}^*(n,d,b)$, every node of G is within distance $\log_{(d-1)} m + O(1)$ of at least m other nodes for every $m = o(n^{1-1/\lfloor \frac{d+1}{2} \rfloor})$.*

Proof: Let G be a graph that forms the left or right half of a graph in $\mathcal{G}^*(n,d,b)$ and let v be a fixed node of G. We will show that with probability $1 - o(1/n)$, there are m nodes within distance $\log_{(d-1)} m + O(1)$ of v for every $m = o(n^{1-1/\lfloor \frac{d+1}{2} \rfloor})$. Hence, with probability $1 - o(1)$, this condition will be true for every node of G. (Note that distance in G is measured by paths that are contained entirely within G. Artificial bisection edges inserted in Step 2 of the graph generating procedure are not allowed in such paths.)

Without loss of generality, we can select the edges of G in Step 3 of the procedure for $\mathcal{G}^*(n,d,b)$ in order of increasing distance from v. Initially, we try to select neighbors for v. Of course, some of the points comprising v may be incident to bisection edges (thus not having a neighbor in G), some might be incident to other points in v, and some might be incident to points in some other common node of G. Let n_1 denote the number of nodes in G found to be adjacent to v via a single edge. Similarly, define n_i to be the number of nodes selected only once to be adjacent to a node of distance $i-1$ from v. For each i, it is easily shown that $n_{i+1} \geq n_i(d-1) - 2r_i$, where r_i is the number of points at distance i from v that become incident to a bisection edge, to another point at distance i, or to a point in a node that already is known to be of distance $i+1$ from v. The probability that a point falls into one of these classes is at most

$$\frac{b + n_i d + n_i d}{nd - md^{O(1)}} = o(n^{-1/\lfloor \frac{d+1}{2} \rfloor})$$

since $n_i \leq m = o(n^{1-1/\lfloor \frac{d+1}{2} \rfloor})$ and d is fixed.

Hence the probability that r_i of the $n_i(d-1)$ points fall into this bad class is at most

$$\begin{aligned}\binom{n_i(d-1)}{r_i} o\left(n^{-r_i/\lfloor \frac{d+1}{2} \rfloor}\right) &= o\left(\left[\frac{n_i(d-1)e}{r_i n^{1/\lfloor \frac{d+1}{2} \rfloor}}\right]^{r_i}\right) \\ &= o\left(\left[\frac{n_i de}{r_i n^{2/(d+1)}}\right]^{r_i}\right).\end{aligned}$$

For $n_i \leq n^{1/(d+1)}$, choosing $r_i = 4d$ is more than sufficient to make this probability $o(1/n^2)$. Otherwise, it is sufficient to make $\frac{n_i de}{r_i n^{2/(d+1)}} \leq \frac{1}{2}$ and $r_i \geq 2\log n$. For $n_i \geq n^{1/(d+1)}$, this can be accomplished by setting $r_i = 2n_i \log n / n^{1/(d+1)}$.

Thus for any $m = o(n^{1-1/\lfloor \frac{d+1}{2} \rfloor})$, we can conclude that with probability $1 - o(1/n)$,

$$n_{i+1} \geq n_i(d-1) - 8d$$

for all i such that $n_i \leq n^{1/(d+1)}$, and

$$n_{i+1} \geq n_i \left(d - 1 - \frac{2 \log n}{n^{1/(d+1)}} \right)$$

for all i such that $n_i \leq m$. Provided that n_f is greater than $8d/(d-2)$ for some constant f, the first recurrence can be solved to find that $n_i = \Theta\left((d-1)^{i-f}\right)$. The second recurrence extends this result to large i. Hence with probability $1 - o(1/n)$, v has m neighbors within distance $\log_{d-1} m + O(1)$. Although we have omitted the proof that n_f is greater than $8d/(d-2)$ for some constant f (with probability $1 - o(1/n)$), the details are not difficult to work out.∎

4.1. Bisecting Graphs With $o(\sqrt{n})$ Bisection Width

Let G be a d-regular graph. For each node v in G, define the *neighborhood* $N(v)$ of v to be the set of all nodes within distance $\log_{d-1} \sqrt{n} - 2$ of v. For each pair of nodes u and v, the algorithm finds the mincut $c(u,v)$ in G using the maxflow-mincut algorithm when $N(u)$ is replaced by an infinite capacity source and $N(v)$ is replaced by an infinite capacity sink. More precisely, the edges linking $N(u)$ and $N(v)$ to $G - N(u) - N(v)$ are replaced by edges linking $G - N(u) - N(v)$ directly to the source and sink, respectively. Edges of G linking nodes contained in $N(u)$ to nodes contained in $N(v)$ are replaced by edges linking the source and sink directly. If a cut with the minimum cardinality is a bisection, then the algorithm outputs that cut. Otherwise, the algorithm halts without output. We call this procedure Algorithm 1.

We first show that Algorithm 1 never outputs a suboptimal bisection.

Theorem 4.2. *Whenever Algorithm 1 outputs a bisection for a d-regular graph, it is guaranteed to be the minimum bisection.*

Proof: Suppose a graph G has a bisection of size b' that is less than the bisection of size b output by the algorithm. Since the sources and sinks are grown to a distance of $\log_{d-1} \sqrt{n} - 2$, it is easily shown that every mincut has size at most $\sqrt{n}$, and thus $b' < \sqrt{n}$.

Given that G is d-regular for some d, the number of nodes within distance r of the b'-bisection in each half of the $2n$-node graph is at most $2b'(d-1)^r$. Hence, at least half of the nodes in each half of G (with respect to the b'-bisection) have distance greater than $\log_{d-1}(n/4b') \geq \log_{d-1}(\sqrt{n}/4)$ from the b'-bisection. Hence, the algorithm finds at least one such pair of nodes on opposite sides of the bisection. Because the sources and sinks grown out from these nodes extend for distance at most $\log_{d-1} \sqrt{n} - 2$, neither will cross the

b'-bisection. Hence, the maxflow between the two can be at most b'. This is a contradiction since the algorithm would not have output a b-bisection had there been a mincut of size $b' < b$. ∎

From the preceding analysis, it is clear that Algorithm 1 never outputs bisections for graphs with bisection width greater than $\sqrt{n}$. For almost all graphs with $o(\sqrt{n})$ bisection width, however, the smallest mincut found in Algorithm 1 is precisely the minimum bisection.

Theorem 4.3. *For all $d \geq 3$, all $b = o(\sqrt{n})$, $n \to \infty$ and almost every graph G in $\mathcal{G}^*(n, d, b)$, Algorithm 1 outputs the minimum bisection of G.*

Proof: We will show that for almost all G, the smallest of the mincuts (over all pairs of sources and sinks) is precisely the bisection artificially inserted into G during Step 2 of the procedure for $\mathcal{G}^*(n, d, b)$. The fact that this bisection is optimal then follows from either Theorem 2.10 or Theorem 4.2.

There are two cases to consider depending on whether the source and sink originate on the same or different sides of the bisection. In either case, they encompass at least $3b/cd$ nodes on their respective sides (by Lemma 4.1), where c is the constant defined in Corollary 2.7. If they are on the same side, then by Corollary 2.7, the mincut separating them must contain at least $cd(3b/cd) - b \geq 2b$ edges of G (including edges incident to the source and/or sink). Such large cuts have no impact on the output.

If the source and sink originate on opposite sides of the bisection, there are again two cases to consider depending on whether or not either includes one or more edges of the bisection. By the arguments in the proof of Theorem 4.2, at least 1/4 of such source-sink pairs will not reach the bisection. In this case, Corollary 2.7 and Lemma 2.9 are easily combined to show that the mincut between the source and sink is precisely the bisection. Were another cut of smaller or equal size to exist, then there would be a cut with k or fewer edges separating the source from k of the bisection edges in (without loss of generality) the left half of G for some k. If the smaller piece of this cut contains the source, Lemma 4.1 and Corollary 2.7 provide a contradiction as before. Otherwise, Lemma 2.9 applies to provide the contradiction.

If the source and sink originate on opposite sides of the bisection, but one or both includes one or more edges of the bisection, then the mincut must be greater than b. This is because both the source and sink still encompass at least $3b/cd$ nodes on their respective sides. By the argument in the preceding paragraph, however, the implanted bisection is

the only cut of size b or smaller separating the two. Since at least one edge of the bisection is included inside the source or sink, that bisection no longer separates them. Hence, the mincut that does separate them must be larger.

In conclusion, at least 1/8 of the source-sink pairs will produce a unique mincut that is the bisection. The remainder will produce larger cuts.∎

4.2. Bisecting Graphs With Larger Bisection Width

Algorithm 1 does not work for graphs with bisection width $b = \Omega(\sqrt{n})$ since the neighborhoods required for such graphs must be grown to a depth of $\log_{d-1} b + \Theta(1)$ and, as a consequence, will almost always contain part of the minimum bisection. Hence the minimum bisection is not likely to appear as a mincut for any source-sink pair.

However, it is possible to prove that for almost all graphs in $\mathcal{G}^*(n,d,b)$ with $b = o(n^{1-1/\lfloor\frac{d+1}{2}\rfloor})$, many of the mincuts will contain all the bisection edges not absorbed by the source and sink and, otherwise, only edges that are incident to the source and/or sink. Hence, by summing the number of times each edge appears in a mincut $c(u,v)$ over all pairs u and v, it is possible to readily distinguish the edges in the minimum bisection of such graphs (since they are guaranteed to appear in many more mincuts than edges not in the bisection). This process is the first phase of Algorithm 2. Phase II is designed to verify that bisections found in Phase I are, in fact, optimal. A more detailed description of Algorithm 2 follows.

Algorithm 2

(Do both phases for $q = 2,4,8,\ldots,o(n^{1-1/\lfloor\frac{d+1}{2}\rfloor})$ and then halt.)

Phase I Initial computation of bisection.

Step 1. For each node v in G define the neighborhood $N(v)$ of v to be the set of all nodes within distance $\log_{d-1} q$ of v.

Step 2. For each pair of nodes u and v in G, compute the mincut $c(u,v)$ in G using the maxflow-mincut algorithm when $N(u)$ is replaced by an infinite capacity source and $N(v)$ is replaced by an infinite capacity sink. (As in Algorithm 1, the edges linking $N(u)$ or $N(v)$ to $G-N(u)-N(v)$ are replaced by edges linking the source or sink to $G-N(u)-N(v)$, respectively. Edges linking nodes contained in $N(u)$ to nodes contained in $N(v)$ are replaced by edges linking the source and sink.)

Step 3. Let B be the set of b edges of G contained in at least $n^2/2$ of the $\binom{2n}{2}$ mincuts. If B is a bisection then proceed to Phase II. Otherwise, proceed with the next value of q.

Phase II Verification that B is the minimum bisection.

Step 4. Repeat Steps 1 and 2 above for all u and v on opposite sides of B, except replace each edge of B with an edge of capacity $1+1/d$ and restrict the construction of the sources and sinks so that they do not cross from one side of B to the other.

Step 5. Check that the maxflows computed in Step 4 all have size $b(1+1/d)$. If this is the case, then output B and halt. Otherwise, proceed with the next value of q.

In Theorems 4.5 and 4.6 we will show that Algorithm 2 never outputs a suboptimal bisection, and almost always finds the optimal bisection. Both theorems make use of the following simple lemma.

Lemma 4.4. *For every $2n$-node graph G with node degree at most d, and every s-edge subset S of G,*

$$\sum_{v \in G} \rho(v,r) \leq 4sr(d-1)^{r-1}$$

for all r, where $\rho(v,r)$ is the number of nodes reachable by a path of length r or less originating from v and traveling through S.

Proof: The claim is proved by bounding the number of paths of length r or less that pass through one of the s edges of S. The number of such paths is clearly at most

$$4s\sum_{i=0}^{r-1}(d-1)^i(d-1)^{r-i-1}$$

since at most $2(d-1)^i$ nodes are within distance i of the one side of an edge of S and at most $2(d-1)^{r-1-i}$ are within distance $r-1-i$ of the other side for any i. Simplifying the preceding expression then gives the desired bound.∎

Theorem 4.5. *For sufficiently large n, whenever Algorithm 2 outputs a bisection, it is guaranteed to be the minimum bisection.*

Proof: Suppose a $2n$-node d-regular graph G has a bisection B' of size b' which is less than the size b of the bisection B output by Algorithm 2. In what follows, we will show that this implies that a substantial portion of the source-sink pairs computed in Step 4 have flow less than $b(1+1/d)$, thus establishing a contradiction.

A simple counting argument reveals that at least half of the source-sink pairs on opposite sides of B originated with a pair of nodes u and v that are also on opposite sides

of B'. The maximum flow for such a pair is at most

$$b' + \frac{1}{d}(b' - s) + m \leq b\left(1 + \frac{1}{d}\right) + m - \frac{s}{d}$$

where s is the number of nodes in $S = B' - B$ and m is the number of nodes included in $N(u)$ or $N(v)$ but that are across B' from u or v, respectively.

Since the source and sink cannot cross or include edges in B, only nodes that have short paths through S to u or v can be opposite B' from u or v and still be included in the source or sink, respectively. Hence, by Lemma 4.4 we can deduce that

$$m \leq \frac{16sr(d-1)^{r-1}}{n}$$

for at least $1/4$ of the source-sink pairs that are on opposite sides of B and of B', where $r = \log_{d-1} q$ is the radius of the sources and sinks defined in Algorithm 2. Since $q \leq n^{1-1/\lfloor\frac{d+1}{2}\rfloor}$, m is much less than s/d for large values of n, thus giving the contradiction.∎

By Theorem 4.5, we know that bisections output by Algorithm 2 are optimal whenever n satisfies

$$16d(1-1/\lfloor\frac{d+1}{2}\rfloor)\log_{d-1} n < (d-1)n^{1/\lfloor\frac{d+1}{2}\rfloor}.$$

For small values of n, the inequality is not satisfied but the result is probably still true. We are currently working through a more careful analysis that will provide lower bound proofs for most small graphs.

Theorem 4.6. *For all $d \geq 3$, all $b = o(n^{1-1/\lfloor\frac{d+1}{2}\rfloor})$, $n \to \infty$ and almost every graph G in $\mathcal{G}^*(n,d,b)$, Algorithm 2 outputs the minimum bisection of G.*

Proof: We consider a pass of Algorithm 2 when q is much larger than b, but is still much smaller than $n^{1-1/\lfloor\frac{d+1}{2}\rfloor}$. For such q, we show that with probability $1 - o(1)$, all of the artificially inserted bisection edges of G are included in at least $(1 - o(1))n^2$ of the $\binom{2n}{2}$ mincuts, and that all other edges are included in only $o(n^2)$ mincuts. Hence, precisely the artificial bisection B is identified at the end of Phase I for almost all G in $\mathcal{G}^*(n,d,b)$. We conclude by showing that B almost always satisfies the conditions checked in Phase II, thus completing the proof.

We first make the following observation. From Lemma 4.4 it can be deduced that no more than $O(q \log n)$ sources or sinks which start from one side of B and include more than $o(b)$ edges of B or nodes and edges on the opposite side of B. Thus $1 - O(q \log n/n) = 1 - o(1)$ of all the sources and sinks will stay (for the most part) on the side of B from which they start from.

We divide the analysis into 2 cases: (i) the source-sink pair starts from the same side of B, and (ii) the source-sink pair starts from opposite sides of B.

Case (i): By the above observation, all but $o(1)$ of the $n^2 - n$ source-sink pairs that start on the same side of B stay (for the most part) on that same side. We first observe that the mincut returned by the maxflow algorithm must be close to, i.e., within constant distance of, the frontier of the source or sink. Otherwise, by Corollary 2.7 and the fact that $b \ll q \ll n^{1-1/\lfloor\frac{d+1}{2}\rfloor}$, the mincut will be of size much larger than a cut at the bisection and around the source or sink, which is $\Theta(q+b)$. Now by using a similar argument to that of Lemma 2.9, we can easily show that the neighborhoods around a source or sink are sufficiently bushy that the mincut has to occur right on the boundary of the source or sink.

Since an edge can be on the frontier of a source or sink for at most $O(qn) = o(n^2)$ source-sink pairs, the preceding analysis means that edges not in B are included in at most $o(n^2)$ mincuts during Phase I of the algorithm.

Case (ii): This is similar to the above case, in particular we can show that the mincut is precisely the edges of B not in a source or sink along with the edges incident to a source or sink but on the opposite side of B from the origin of the source or sink, respectively. This is the case since by the observation at the beginning of the proof, all but $o(1)$ of the n^2 source-sink pairs that start from opposite sides of B stay (for the most part) on the sides of their origin. This fact, together with the observation that every bisection edge is in at most $O(b) = o(n)$ sources or sinks, implies that each edge of B is included in at least $n^2 - o(n^2)$ mincuts. Hence B is distinguished for almost all G at the end of Phase I.

The analysis of Phase II is easier than that of Phase I since the sources and sinks are not allowed to cross B. We need only mimic the proof of Theorem 4.3, substituting higher capacity edges at the bisection in the proof of Lemma 2.9. Thus, with probability $1 - o(1)$, all Phase II source-sinkpairs have mincuts at B with size $b(1 + 1/d)$.∎

4.3. Running Time Analysis

As stated, each pass of Algorithm 2 solves $O(n^2)$ flow problems on graphs with $2n$ nodes and dn edges. At most $O(dq)$ augmenting paths (each carrying $1/d$ unit of flow) need to be found for each flow problem, and each requires at most $O(dn)$ steps in the worst case. Hence Algorithm 2 can always be made to run in $O(d^2 n^{4-1/\lfloor\frac{d+1}{2}\rfloor})$ steps. However, by modifying the algorithm slightly, this bound can be substantially improved. For example, by modifying the algorithm to check that most of the mincuts have $\Theta(q)$ edges at each

pass before proceeding to the next value of q, the worst case running time can be improved to $O(d^2bn^3)$ where b is the bisection width of the graph. (This can be proved by showing that for $q \leq b$, this condition is almost always satisfied, but for $q \gg b$ the condition is never satisfied.)

A more substantial improvement can be achieved by finding the mincuts for only a small random sample of the source-sink pairs. A more careful look at the proof of Theorems 4.5 and 4.6 reveals that only $\log n$ source-sink pairs are needed to insure the results with probability $1 - o(1/n)$ for any graph. For the upper bounds on bisection, this probability can be incorporated into the $1 - o(1)$ term in Theorem 4.6. The randomization has a more serious impact on the lower bound, however, since lower bound proofs would then only be correct with probability $1 - o(1/n)$. In any case, the running time for the probabilistic version of the algorithm is $O(d^2bn \log n)$. If we only require correct answers with probability $1 - o(1)$, then the $\log n$ term can be replaced by any increasing function. If we remove the lower bound portion of the algorithm entirely, then the expected time is $O(d^2bn)$.

Savings can also be made in the flow algorithm itself. By restricting ourselves to unit size flows in all but the bisection edges, one of the d factors can be removed. Although we do not yet have a proof, it is quite possible that even greater savings can be obtained by using the properties of random graphs to show that the augmenting paths are usually found in far fewer than $\Theta(dn)$ steps. In fact, it might be the case that the ith augmenting path can be found in $O(n/(b - i))$ steps. If so, this would replace the dbn term in the preceding expressions with an $n \log b$ term. Hence, the expected time to find the upper bound might be as fast as $O(n \log b)$.

In practice, the probabilistic version of the algorithm runs very quickly. When computing the data in Chapter 6, the algorithm appeared to be much faster than both the Kernighan-Lin algorithm and simulated annealing.

Chapter 5

Analysis of The Greedy Algorithm

In this chapter we analyze the performance of the greedy algorithm on the class $\mathcal{G}(n,3,b)$. In particular, we will show that with probability $1-o(1)$, the greedy algorithm will not find the optimal bisection for graphs in $\mathcal{G}(n,3,b)$, with $b = o(\sqrt{n})$. The greedy algorithm that we consider in this chapter is a minor modification of the greedy algorithm described in Chapter 3. Particularly, the greedy algorithm here does not mark a vertex once the vertex has been chosen.

We will start with some definitions. Let $G = (V,E)$ be a d-regular graph. let $S \subset V$, $|S| = k$. A vertex $v \in S$ is *bad* if

$$|\{ w \in S \mid (v,w) \in E \}| < |\{ w \in V - S \mid (v,w) \in E \}|,$$

i.e., if v is more strongly connected to $V - S$ than to S. A set S is *stable* if it does not have any bad vertices, otherwise it is *unstable*.

We will first consider graphs in the class $\mathcal{G}(n,3,0)$, i.e., random cubic graphs on $2n$ vertices with bisection width 0. We will show that starting with a random bisection of a graph $G' \in \mathcal{G}(n,3,0)$, with probability $1-o(1)$, the greedy algorithm will not be able to find the optimal bisection. This will be accomplished by showing that for any random bisection of G', there are stable sets in each of the 4 parts of G' created by the bisection (note that G' consists of 2 disjoint cubic graphs on n vertices each.) Since the greedy algorithm will stop when it encounters a stable set, we will have shown that the greedy algorithm fails to find the optimal solution. The stable sets that we are looking for will be cycles. We will then show that the same result holds for graphs in $\mathcal{G}(n,3,b)$, with $b = o(\sqrt{n})$.

For our purpose it is sufficient to consider a cubic graph $G = (V,E)$ on n vertices

which makes up the left or right part of a graph in $\mathcal{G}^*(n,3,0)$. A *cycle* in G is a set of distinct ordered vertices $\{v_1,\ldots,v_k\}$ such that $(v_j,v_{j+1}) \in E$ for $j = 1,\ldots,k-1$ and $(v_k,v_1) \in E$. We shall call a cycle of size i an *i-cycle*, and we will consider only cycles of size $o(\sqrt{n})$. Let n_i be the number of potential pointwise-labelled i-cycles in G. Then

$$\begin{aligned} n_i &= \binom{n}{i}(i-1)!\binom{3}{2}^i 2^i \\ &= \frac{n!\,6^i}{i\,(n-i)!} \end{aligned}$$

by using Stirling's formula we get

$$\frac{(6n)^i}{i}\,e^{-i^2/n} \le n_i \le \frac{(6n)^i}{i}\,e^{-i^2/4n} \tag{5.1}$$

for $i = o(\sqrt{n})$. For $j = 1,\ldots,n_i$ define

$$X_j^i = \begin{cases} 1, & \text{if the } j\text{th } i\text{-cycle is in } G; \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

Then it is not difficult to see that

$$Pr\{X_j^i = 1\} = \frac{M(3n-2i)}{M(3n)}, \tag{5.3}$$

where as before $M(x)$ denotes the number of ways to perfect match x points, and

$$\sqrt{2}\,x^{x/2}\,e^{-x/2-1/(9x)} \le M(x) \le \sqrt{2}\,e^{-x/2}\,x^{x/2}.$$

Thus substituting into the above equation we get

$$(3n)^{-i}\,e^{i^2/4n} \le Pr\{X_j^i = 1\} \le (3n)^{-i}\,e^{i^2/n} \tag{5.4}$$

for $i = o(\sqrt{n})$, and for sufficiently large n.

Consider a random bisection of G', it partitions G into 2 disjoint subsets A and B not necessarily of equal size. We define the following indicators Z_j^i,

$$Z_j^i = \begin{cases} 1, & \text{if the } j\text{th } i\text{-cycle lies entirely in } A; \\ 0, & \text{otherwise.} \end{cases} \tag{5.5}$$

Clearly,

$$Pr\{Z_j^i = 1\} = \binom{2n-i}{n-i} \Big/ \binom{2n}{n} \tag{5.6}$$

which yields

$$2^{-i}\,e^{-i^2/3n} \le Pr\{Z_j^i = 1\} \le 2^{-i}\,e^{-i^2/12n} \tag{5.7}$$

for $i = o(\sqrt{n})$ and for large enough n. We further let $U_j^i = X_j^i Z_j^i$. Then $U_j^i = 1$ if the jth i-cycle appears in G and lies entirely within A. Since X_j^i and Z_j^i are independent we have

$$\begin{aligned} Pr\left\{U_j^i = 1\right\} &= Pr\left\{X_j^i = 1 \text{ and } Z_j^i = 1\right\} \\ &= Pr\left\{X_j^i = 1\right\}\ Pr\left\{Z_j^i = 1\right\} \end{aligned} \tag{5.8}$$

By equations (5.4) and (5.7) we get the following inequalities for $i = o(\sqrt{n})$ and for large enough n

$$(6n)^{-i}\, e^{-i^2/n} \leq Pr\left\{U_j^i = 1\right\} \leq (6n)^{-i}\, e^{i^2/n}. \tag{5.11}$$

Let W_i denote the number of i-cycles that are in A. Then

$$W_i = \sum_{j=1}^{n_i} U_j^i, \tag{5.12}$$

and let

$$W = \sum_{i=3}^{\sqrt{\log n}} W_i \tag{5.13}$$

be the total number of cycles in A of size at most $\sqrt{\log n}$ and at least 3. To show our main result of this chapter it suffices to show that

$$Pr\{W \geq 1\} \to 1 \tag{5.14}$$

as $n \to \infty$.

We will start with some lemmas. The first lemma is the crux of what is usually referred to as the second moment method.

Lemma 5.1.

$$Pr\{W = 0\} \leq \frac{E(W^2)}{E(W)^2} - 1.$$

Proof: For any $t > 0$, Chebyshev's inequality (see for example [Fe68]) gives

$$Pr\{|W - E(W)| \geq t\} \leq \frac{Var(W)}{t^2} \tag{5.1.1}$$

If $W = 0$ then (5.1.1) will be satisfied when $t = |E(W)|$. Thus

$$\begin{aligned} Pr\{W = 0\} &\leq \frac{Var(W)}{E(W)^2} \\ &= \frac{E(W^2) - E(W)^2}{E(W)^2} \end{aligned}$$

$$= \frac{E(W^2)}{E(W)^2} - 1. \blacksquare$$

Lemma 5.2. *For $i = o(\sqrt{n})$ and for sufficiently large n,*

$$\frac{1}{i}\, e^{-2i^2/n} \leq E(W_i) \leq \frac{1}{i}\, e^{i^2/n}.$$

Proof: Since $W_i = \sum_{j=1}^{n_i} U_j^i$, we have

$$\begin{aligned} E(W_i) &= \sum_{j=1}^{n_i} E(U_j^i) \\ &= \sum_{j=1}^{n_i} Pr\left\{U_j^i = 1\right\} \\ &\leq \frac{(6n)^i}{i}\, e^{-i^2/4n}\, (6n)^{-i}\, e^{i^2/n} \\ &= \frac{1}{i}\, e^{i^2/n} \end{aligned}$$

for $i = o(\sqrt{n})$, by (5.1) and (5.11). By using the lower bounds of these equations we also get

$$E(W_i) \geq \frac{1}{i}\, e^{-2i^2/n}. \blacksquare$$

The next lemma gives an upper bound for the value of $E(W_i^2)$.

Lemma 5.3. *For $i = o(\sqrt{n})$ and for large enough n,*

$$E(W_i^2) \leq \frac{e^{i^2/n}}{i} + \frac{e^{3i^2/n}}{i^2} + \frac{2^{2i}}{n}\, e^{4i^2/n}\, i!.$$

Proof: Since $W_i = \sum_{j=1}^{n_i} U_j^i$, we have

$$\begin{aligned} E(W_i^2) &= E\Big(\sum_{j=1}^{n_i} U_j^i\Big)^2 \\ &= \sum_{j=1}^{n_i} E\Big(\left(U_j^i\right)^2\Big) + \sum_{j \neq k} E\left(U_j^i\, U_k^i\right) \\ &= \sum_{j=1}^{n_i} E\left(U_j^i\right) + \sum_{j \neq k} E\left(U_j^i\, U_k^i\right) \\ &= E\left(W_i\right) + \sum_{j \neq k} Pr\left\{U_j^i = 1 \text{ and } U_k^i = 1\right\} \end{aligned} \qquad (5.3.1)$$

Claim:

$$\sum_{j \neq k} Pr\left\{U_j^i = 1 \text{ and } U_k^i = 1\right\} \leq \frac{e^{3i^2/n}}{i^2} + \frac{2^{2i}}{n} e^{4i^2/n}\, i!\,. \tag{5.3.2}$$

With this claim and Lemma 5.2 we obtain the lemma. It remains, therefore, to prove the claim. We have

$$\begin{aligned}\sum_{j \neq k} Pr\left\{U_j^i = 1 \text{ and } U_k^i = 1\right\} &= \sum_{j=1}^{n_i} \sum_{\substack{k=1 \\ k \neq j}}^{n_i} Pr\left\{U_j^i = 1 \text{ and } U_k^i = 1\right\} \\ &= \sum_{j=1}^{n_i} Pr\left\{U_j^i = 1\right\} \sum_{\substack{k=1 \\ k \neq j}}^{n_i} Pr\left\{U_k^i = 1 \mid U_j^i = 1\right\}. \end{aligned} \tag{5.3.3}$$

Consider the sum

$$\sum_{\substack{k=1 \\ k \neq j}}^{n_i} Pr\left\{U_k^i = 1 \mid U_j^i = 1\right\}. \tag{5.3.4}$$

Another way to look at it is to classify the potential i-cycles by the size of their intersections with the jth i-cycle.

- If the intersection of the kth i-cycle with the jth i-cycle is empty then

$$Pr\left\{U_k^i = 1 \mid U_j^i = 1\right\} \leq \frac{M(3n-4i)}{M(3n-2i)}\, 2^{-i} \leq (6n)^{-i}\, e^{3i^2/n} \tag{5.3.5}$$

and the number of i-cycles that do not intersect with the j-th i-cycle is

$$\binom{n-i}{i} (i-1)! \binom{3}{2}^i 2^i \leq \frac{(6n)^i}{i}\, e^{-i^2/n}, \tag{5.3.6}$$

for $i = o(\sqrt{n})$.

- On the other hand, if there are $s < i$ edges in common between the jth i-cycle and the kth i-cycle, then there are at most

$$\binom{i}{s} \binom{n-i}{i-s-1} (i-1)! \binom{3}{2}^{i-s} 2^{i-s} \leq \frac{2^i}{n} (6n)^{i-s}\, i! \tag{5.3.7}$$

such i-cycles, for $i = o(\sqrt{n})$, and the probability $Pr\left\{U_k^i = 1 \mid U_j^i = 1\right\}$ is at most

$$\frac{M(3n-4i+2s)}{M(3n-2i)}\, 2^{-i+2s} \leq (6n)^{-i+s}\, 2^s\, e^{3i^2/n} \tag{5.3.8}$$

for $i = o(\sqrt{n})$.

Substituting equations (5.3.5), (5.3.6), (5.3.7), and (5.3.8) into equation (5.3.4) we get

$$\sum_{\substack{k=1\\k\neq j}}^{n_i} Pr\left\{U_k^i = 1 \mid U_j^i = 1\right\} \leq (6n)^{-i}\, e^{3i^2/n}\, \frac{(6n)^i}{i}\, e^{-i^2/n} + \sum_{s=1}^{i-1} \frac{2^i}{n}\, (6n)^{i-s}\, i!\, (6n)^{-i+s}\, 2^s\, e^{3i^2/n}$$

$$\leq \frac{e^{2i^2/n}}{i} + \frac{2^{2i}}{n}\, i!\, e^{3i^2/n} \tag{5.3.9}$$

Using (5.3.9) the sum (5.3.3) can now be written as

$$\sum_{j\neq k} Pr\left\{U_j^i = 1 \text{ and } U_k^i = 1\right\} = \sum_{j=1}^{n_i} Pr\left\{U_j^i = 1\right\} \sum_{\substack{k=1\\k\neq j}}^{n_i} Pr\left\{U_k^i = 1 \mid U_j^i = 1\right\}$$

$$\leq \sum_{j=1}^{n_i} Pr\left\{U_j^i = 1\right\} \left[\frac{e^{2i^2/n}}{i} + \frac{2^{2i}}{n}\, e^{3i^2/n}\, i!\right]$$

$$\leq n_i\, (6n)^{-i}\, e^{i^2/n} \left[\frac{e^{2i^2/n}}{i} + \frac{2^{2i}}{n}\, e^{3i^2/n}\, i!\right]$$

$$\leq \frac{(6n)^i}{i}\, e^{-i^2/4n}\, (6n)^{-i}\, e^{i^2/n} \left[\frac{e^{2i^2/n}}{i} + \frac{2^{2i}}{n}\, e^{3i^2/n}\, i!\right]$$

$$\leq \frac{e^{3i^2/n}}{i^2} + \frac{2^{2i}}{n}\, e^{4i^2/n}\, i!\,. \tag{5.3.10}$$

This proves the claim and completes the proof of the lemma.∎

The previous lemma and the next lemma will help us compute an upper bound for $E(W^2)$.

Lemma 5.4. *For $i < j = o(\sqrt{n})$ and for sufficiently large n,*

$$E(W_i W_j) \leq \frac{e^{3j^2/n}}{ij} + \frac{2^{2i}}{n}\, e^{4j^2/n}\, j!\,.$$

Proof: By definition of W_i we have

$$E(W_i W_j) = E\left[\left(\sum_{k=1}^{n_i} U_k^i\right)\left(\sum_{l=1}^{n_j} U_l^j\right)\right]$$

$$= \sum_{k=1}^{n_i} \sum_{l=1}^{n_j} E(U_k^i\, U_l^j)$$

$$= \sum_{k=1}^{n_i} \sum_{l=1}^{n_j} Pr\left\{U_k^i = 1 \text{ and } U_l^j = 1\right\}$$

$$= \sum_{k=1}^{n_i} Pr\left\{U_k^i = 1\right\} \sum_{l=1}^{n_j} Pr\left\{U_l^j = 1 \mid U_k^i = 1\right\}. \tag{5.4.1}$$

Claim:

$$\sum_{l=1}^{n_j} Pr\left\{U_l^j = 1 \mid U_k^i = 1\right\} \leq \frac{e^{2j^2/n}}{j} + \frac{2^{2i}}{n} e^{3j^2/n} j!. \tag{5.4.2}$$

Again we arrange the terms of the above sum according to the size of their intersection with the kth i-cycle. We proceed as in the proof of the previous lemma.

- When the lth j-cycle and the kth i-cycle have an empty intersection then

$$Pr\left\{U_l^j = 1 \mid U_k^i = 1\right\} \leq \frac{M(3n-2i-2j)}{M(3n-2i)} 2^{-j} \leq (6n)^{-j} e^{2j^2/n} \tag{5.4.3}$$

and there are

$$\binom{n-i}{j} (j-1)! \binom{3}{2}^j 2^j \leq \frac{(6n)^j}{j} \tag{5.4.4}$$

such j-cycles.

- When the lth j-cycle and the kth i-cycle have $s < i$ edges in common, then the number of such j-cycles is at most

$$\binom{i}{s} \binom{n-i}{j-s-1} (j-1)! \binom{3}{2}^{j-s} 2^{j-s} \leq \frac{2^i}{n} (6n)^{j-s} j! \tag{5.4.5}$$

for $i < j = o(\sqrt{n})$. In that case the probability $Pr\{ U_l^j = 1 \mid U_k^i = 1 \}$ is at most

$$\frac{M(3n-2i-2j+2s)}{M(3n-2i)} 2^{-j+2s} \leq (6n)^{-j+s} 2^s e^{3j^2/n}. \tag{5.4.6}$$

Combining these terms we now have

$$\begin{aligned} \sum_{l=1}^{n_j} Pr\left\{U_l^j = 1 \mid U_k^i = 1\right\} &\leq (6n)^{-j} e^{2j^2/n} \frac{(6n)^j}{j} + \sum_{s=1}^{i-1} \left[\frac{2^i}{n} (6n)^{j-s} j! (6n)^{-j+s} 2^s e^{3j^2/n}\right] \\ &\leq \frac{e^{2j^2/n}}{j} + \frac{2^{2i}}{n} e^{3j^2/n} j! \end{aligned} \tag{5.4.7}$$

This completes the proof of the claim. We now use this claim to compute the value of $E(W_i W_j)$. From (5.4.1) and the claim we now have

$$\begin{aligned} E(W_i W_j) &\leq \sum_{k=1}^{n_i} Pr\left\{U_k^i = 1\right\} \sum_{l=1}^{n_j} Pr\left\{U_l^j = 1 \mid U_k^i = 1\right\} \\ &\leq n_i (6n)^{-i} e^{i^2/n} \left[\frac{e^{2j^2/n}}{j} + \frac{2^{2i}}{n} e^{3j^2/n} j!\right] \end{aligned}$$

$$\leq \frac{(6n)^i}{i}\, e^{-i^2/4n}\,(6n)^{-i}\, e^{i^2/n}\left[\frac{e^{2j^2/n}}{j} + \frac{2^{2i}}{n}\, e^{3j^2/n}\, j!\right]$$

$$\leq \frac{e^{3j^2/n}}{ij} + \frac{2^{2i}}{n}\, e^{4j^2/n}\, j!$$

This completes the proof of the lemma.∎

We can now compute an upper bound for the second moment of W.

Lemma 5.5. *For $m = \sqrt{\log n}$, let $H(m)$ denote $\sum_{i=3}^{m}(1/i)$. Then for large enough n we have*

$$E(W^2) \leq e^{m^2/n} H(m) + e^{3m^2/n} H^2(m) + o(1).$$

Proof: Let $m = \sqrt{\log n}$, then by Lemmas 5.5 and 5.4 we have

$$E(W^2) = E\Big[\Big(\sum_{i=3}^{m} W_i\Big)^2\Big]$$

$$= \sum_{i=3}^{m} E(W_i^2) + 2\sum_{i<j} E(W_i W_j)$$

$$\leq \sum_{i=3}^{m}\Big(\frac{e^{i^2/n}}{i} + \frac{e^{3i^2/n}}{i^2} + \frac{2^{2i}}{n}\, e^{4i^2/n}\, i!\Big)$$

$$+ 2\sum_{i<j}\Big(\frac{e^{3j^2/n}}{ij} + \frac{2^{2i}}{n}\, e^{4j^2/n}\, j!\Big)$$

$$\leq e^{m^2/n}\, H(m) + e^{3m^2/n} \sum_{i=3}^{m} \frac{1}{i^2} + \frac{2^{2m}}{n}\, m\, e^{4m^2/n}\, m!$$

$$+ 2e^{3m^2/n} \sum_{i<j} \frac{1}{ij} + \frac{2^{2m+1}}{n}\, m\, e^{4m^2/n}\, m! \qquad (5.5.1)$$

Since $m = \sqrt{\log n}$,

$$\frac{2^{2m+1}}{n}\, m\, e^{4m^2/n}\, m! = o(1),$$

We also have

$$e^{3m^2/n} \sum_{i=3}^{m} \frac{1}{i^2} + 2e^{3m^2/n} \sum_{i<j} \frac{1}{ij} = e^{3m^2/n}\Big[\sum_{i=3}^{m} \frac{1}{i}\Big]^2 = e^{3m^2/n}\, H^2(m),$$

thus (5.5.1) now becomes

$$E(W^2) \leq e^{m^2/n}\, H(m) + e^{3m^2/n}\, H^2(m) + o(1). \qquad (5.5.2)$$

This completes the proof of the lemma.∎

With these lemmas, we are now ready to show that with high probability there exists a cycle in A.

Theorem 5.6.

$$Pr\{W \geq 1\} \rightarrow 1 \qquad as\ n \rightarrow \infty.$$

Proof: By Lemma 5.1, Lemma 5.2, and Lemma 5.5 we have

$$\begin{aligned} Pr\{W = 0\} &\leq \frac{E(W^2)}{E(W)^2} - 1 \\ &\leq \frac{e^{m^2/n} H(m) + e^{3m^2/n} H^2(m) + o(1)}{e^{-4m^4/n} H^2(m)} - 1 \\ &= \frac{e^{m^2/n+4m^4/n}}{H(m)} + e^{3m^2/n+4m^4/n} + o(1) - 1 \end{aligned}$$

Since

$$\frac{e^{m^2/n+4m^4/n}}{H(m)} = O\left(\frac{e^{m^2/n+4m^4/n}}{\log m}\right) = O\left(\frac{e^{5\log^2 n/n}}{\log\log n}\right) = o(1)$$

and

$$e^{3m^2/n+4m^4/n} = 1 + o(1)$$

for $m = \sqrt{\log n}$, we have

$$Pr\{W = 0\} \leq o(1),$$

thus completing the proof of the theorem.∎

Theorem 5.7. *For almost every graph in $\mathcal{G}(n,3,0)$, starting with a random bisection the greedy algorithm will not be able to find the optimal solution, namely the bisection of size 0.*

Proof: Let G' be a graph in $\mathcal{G}^*(n,3,0)$, and let G be a cubic graph that makes up the left hand side of G'. A random bisection of G' will partition G into two disjoint subsets of not necessarily equal size. Then by theorem 5.6 we know that with probability $1 - O(e^{5\log^2 n/n}/\log\log n)$ there is a cycle of size at least 3 and no more than $O(\sqrt{\log n})$ in both parts of this partition of G. By symmetry the same is true of the right hand side of G'. Furthermore, by the arguments in the proof of Theorem 2.10 and Lemma 2.9 we know that the optimal bisection is unique. Therefore, the greedy algorithm will not be able to

find the optimal solution for almost every graph in $\mathcal{G}^*(n,3,0)$. Thus by Theorem 2.10 we have the theorem.∎

We now extend the above theorem to graphs with bisection width greater than 0. In particular we will show that the same result holds true for graphs in $\mathcal{G}(n,3,b)$, with $b = o(\sqrt{n})$.

Theorem 5.8. *For almost every graph in $\mathcal{G}(n,3,b)$, $b = o(\sqrt{n})$, starting with a random bisection the greedy algorithm will not be able to find the optimal solution.*

Proof: Let H be a graph in $\mathcal{G}^*(n,3,b)$ and let G be the graph that makes up the left hand side of H. Randomly connect the b bisection points to make a new graph G'. By Theorem 5.7, in any two parts of G' created by a random bisection of H, there will be a cycle of size at most $\sqrt{\log n}$ with probability $1 - O(e^{5\log^2 n/n}/\log\log n)$. Let C be such a cycle. Then C is also a stable set in that partition with respect to H if none of the edges in the cycles has both endpoints being bisection points. Let $a = \binom{3n}{2}$ and m be the size of C, then the probability that C is also a stable set with respect to H is

$$\binom{a-m}{b/2} \Big/ \binom{a}{b/2} \leq e^{-mb/a} \tag{5.8.1}$$

for $m = O(\sqrt{\log n})$ and $b = o(\sqrt{n})$. Therefore, the probability that there exists a stable set of size at most $\sqrt{\log n}$ in any part of a partition of G created by a random bisection of H is at least

$$\left(1 - O(e^{5\log^2 n/n}/\log\log n)\right)\ e^{-mb/a} = 1 - o(1).$$

By symmetry the same holds true for the right hand side of H. Combining this with the fact that the optimal bisection is unique (by the arguments in the proof of Theorem 2.10 and Lemma 2.9,) and Theorem 2.10 we complete the proof of this theorem.∎

The result in the above theorem, however, is not strong enough to say that the algorithm still fails after a polynomial number of trials. We believe, nonetheless, that this is indeed the case. To show this, however, we think that new techniques have to be used since it is not difficult to see that $Prob\{W_i \geq 1\} \leq E(W_i) \approx 1/i$. Thus starting with a random bisection we have $Prob\{W_i = 0\} \geq 1 - 1/i$. Hence if all the W_i's are independent then the probability that there are no cycles is at least

$$\prod_{i=3}^{n}(1 - \frac{1}{i}) = \frac{2}{n}.$$

We believe that even if the random bisection made at the beginning of the greedy algorithm does not contain any cycle, the greedy algorithm still gets into trouble as the algorithm progresses, i.e., that cycles are created along the way. We also conjecture that algorithms which utilize a local optimization technique, e.g., the Kernighan-Lin and simulated annealing algorithms, will fail for almost every graph in $\mathcal{G}(n, 3, b)$, with very high probability. The conjecture is based on informal intuition not presented here and the data in Chapter 6. An observation worth noting is that for large degree graphs, when one vertex is moved from one side of the bisection to the other it is likely that a large number of vertices, which were stable before, will now become unstable. This observation together with the data collected in Chapter 6 prompts us to make the conjecture that for large enough degree d, algorithms such as greedy and Kernighan-Lin will be able to find the optimal bisection for almost every graph in $\mathcal{G}(n, d, b)$.

Chapter 6

Experimental Data and New Heuristics

This chapter consists of three sections. In the first section we present the data that we have collected in comparing the algorithms described in Chapter 3 and our algorithms in Chapter 4. This set of data also contains the results of some new algorithms that we suggest in the second section of this chapter. Finally, in the third section we give some guidelines on how one could use our algorithms efficiently in practice.

6.1. Experimental Data

This section contains the data comparing the performance of the algorithms described in Chapter 3 and our algorithms in Chapter 4. We have also tested two new algorithms on the same set of graphs. These new algorithms will be described in the next section. Overall we have generated over 100 graphs in $\mathcal{G}(n,3,b)$, $\mathcal{G}^*(n,4,b)$ and $\mathcal{G}^*(n,5,b)$. Each graph was generated in the manner described in Chapter 2, and associated with each graph is a seed used by the random number generator in the graph generating program. This seed allows us to regenerate the exact same graph again to be tested on different algorithms. These seeds are also recorded in the tables of data. Due to the difficulty in generating graphs without loops and multiple edges for $d > 3$, we have generated graphs in $\mathcal{G}^*(n,4,b)$ and $\mathcal{G}^*(n,5,b)$ instead of graphs in $\mathcal{G}(n,4,b)$ and $\mathcal{G}(n,5,b)$. Our experience with $\mathcal{G}^*(n,3,b)$ and $\mathcal{G}(n,3,b)$ showed that the results are the same in either case.

For each graph generated, we run the greedy algorithm (GD), the Kernighan-Lin algorithm (KL), the Kernighan-Lin algorithm with contraction (CKL, this algorithm will be described in the next section), simulated annealing (SA), our maxflow-based algorithm described in Chapter 4 (MF), and finally another maxflow-based algorithm (AMF) which will be described in the next section.

We used a probabilistic version of our algorithm described in Chapter 4 in collecting the data. That is, instead of running the algorithm with all possible pairs of vertices as sources and sinks, we only ran it on a number of random pairs. The actual number of pairs chosen varied between 30 and 200, with the larger number of pairs used for larger graphs. Except for rare cases when a node was incident to more than $\lfloor \frac{d-1}{2} \rfloor$ bisection edges or when an edge linked two nodes that were each incident to $\frac{d-1}{2}$ bisection edges, the algorithms always identified the correct bisection. Even in the few cases when a bisection was not precisely identified, a quick look at vertices near the cut revealed the irregularity and the optimal bisection. Such cases are identified by letters in the tables of data.

As can be seen from the data, none of the algorithms KL, SA, and GD performed very well for degree 3 graphs, although the Kernighan-Lin and simulated annealing algorithms performed dramatically better as the degree was increased. Experiment with graphs of higher degree showed that the performance of the greedy algorithm also increased but at a slower rate. Since the Kernighan-Lin and greedy algorithms were discovered to be sensitive to the choice of the initial bisection, we ran these algorithms several times for each graph (each time starting with a different random bisection). For each random starting bisection KL and GD were run until there was no more improvements. The data for these algorithms represent the bisections found for the three initial bisections tried for each graph. To reduce the running time in our implementations of KL and GD, we have used the shortcut approach described in Chapter 3 in choosing the pairs to be exchanged. That is, the algorithms chose each vertex in the pair independently. This did not affect the performance of these algorithms very much as observed in [KL70].

For the simulated annealing algorithm we used the version of [J84] with the parameters given in Chapter 3. For large graphs, however, SA seemed to run much too slow comparing to the others, almost 3 times as slow as KL, and thus we have reduced the running time by changing the parameter EPOCH-SCALE from 16 to 4 for some graphs. Those results that were obtained with this reduced value of EPOCH-SCALE are marked by †.

The values of the bisection width b were usually chosen between 2 and the largest values for which MF still works. Our algorithm breaks down for large values of b primarily because the probabilistic lemmas proved in Chapter 2 and 4 do not hold for large b.

The performance of CKL and AMF will be discussed in the next section after we have described them. All the algorithms are written in Zeta Lisp, a dialect of Lisp, and run

on the Symbolics 3600 Lisp Machines, release version 6.1. Since we have not implemented the fastest known version of the Kernighan-Lin algorithm, i.e., the one used in [FM82], we can't rigorously compare the running time of all these algorithms. However, we can make the following general observation about the running time. First, the simulated annealing algorithm seems to run very slow compared to our version of the Kernighan-Lin algorithm and the algorithms presented in this thesis. The maxflow based algorithms, Algorithm 2 and AMF, seem to run faster than our implementation of the Kernighan-Lin algorithm, even though in the worst case the Kernighan-Lin algorithm has a better time complexity than the maxflow-based algorithms. The reason might be that the maxflow algorithm runs faster on random graphs. It might be a good research topic to investigate the expected running time of the maxflow-mincut algorithm on random graphs. Also we don't know if in practice the fast implementation of the Kernighan-Lin algorithm will run faster than the maxflow-based algorithms.

Data for degree 3, random regular graphs with given bisection width b.

TABLE 3.1. 3-regular graphs on 100 vertices with bisection width b

b	2			6		
Seed	5785	11790	79654	98123	81925	25132
KL	(18,2,22)	(2,2,2)	(22,2,2)	(10,14,18)	(12,6,6)	(18,8,8)
CKL	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(6,6,6) (6,8,6) (6,6,6)	(6,6,10) (6,6,6) (6,6,6)	(6,6,8) (6,6,8) (6,8,6)
GD	(18,24,42)	(46,48,24)	(14,32,38)	(40,32,18)	(32,24,26)	(26,24,18)
SA	2	2	2	6	6	6
MF	2	2	2	6	6	6
AMF	2	2	2	6	6	6

TABLE 3.2. 3-regular graphs on 200 vertices with bisection width b

b	2			10		
Seed	92153	45242	55660	89811	2312	10990
KL	(2,36,34)	(36,2,10)	(2,14,44)	(38,34,24)	(18,38,28)	(32,34,20)
CKL	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(14,12,12) (10,12,10) (10,10,12)	(10,10,10) (10,12,12) (10,10,10)	(10,10,10) (10,10,10) (10,10,10)
GD	(88,48,76)	(84,66,68)	(88,50,40)	(80,50,38)	(48,42,58)	(52,68,54)
SA	20	2	14	38	10	10
MF	2	2	2	10	10	10
AMF	2	2	2	12	10	12

TABLE 3.3. 3-regular graphs on 400 vertices with bisection width b

b	2			10	
Seed	99388	34545	2656	93553	9329
KL	(2,74,74)	(2,28,14)	(2,82,84)	(74,68,24)	(78,64,10)
CKL	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(10,10,10) (10,10,10) (10,10,10)	(10,10,10) (10,10,10) (10,10,10)
GD	(102,168,110)	(94,156,138)	(130,128,164)	(82,96,168)	(94,180,90)
SA	2	2	42	26	50
MF	2	2	2	10	10
AMF	2	2	2	10	10

TABLE 3.4. 3-regular graphs on 400 vertices with bisection width b

b	14		16		18	
Seed	95549	32321	24243	66542	86310	60629
KL	(74,80,44)	(80,46,78)	(72,42,24)	(70,32,82)	(74,42,58)	(46,72,60)
CKL	(14,14,14) (18,14,14) (14,14,14)	(14,14,14) (14,16,14) (14,14,14)	(20,16,22) (16,20,16) (24,16,20)	(18,18,18) (16,16,20) (16,18,16)	(16,22,16) (16,18,16) (16,16,16)	(20,20,18) (18,20,20) (20,22,24)
GD	(92,168,86)	(94,90,98)	(180,82,86)	(90,102,84)	(170,98,96)	(100,148,124)
SA	60	76	70	44	74	54
MF	14	14	16	16	18	18
AMF	14	14	16	16	18	18

TABLE 3.5. 3-regular graphs on 800 vertices with bisection width b

b	2		10		20	
Seed	31768	64969	75198	69456	37730	21231
KL	(94,2,26)	(78,8,138)	(142,22,102)	(146,10,158)	(22,50,46)	(152,144,98)
CKL	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(10,10,10) (10,10,10) (10,10,10)	(10,10,10) (10,10,10) (10,10,10)	(20,20,18) (18,18,20)	(24,20,24) (20,20,20)
GD	(286,186,314)	(346,186,332)	(314,174,190)	(346,190,194)	(188,198,174)	(202,180,180)
SA	104	104	116	102	126	138
MF	2	2	10	10	20	20
AMF	2	2	10	10	20	32

TABLE 3.6. 3-regular graphs on 800 vertices with bisection width b

b	24		28	
Seed	87930	27934	2769	75407
KL	(112,90,156)	(78,150,54)	(144,144,144)	(136,166,170)
CKL	(26,24,26) (24,24,24)	(26,26,28) (28,30,34)	(26,26,30) (28,30,30)	(28,30,28) (30,30,28)
GD	(174,312,176)	(236,240,186)	(298,326,188)	(188,196,170)
SA	50	126	328	230
MF	24	24	28	28
AMF	24	24	26	32

TABLE 3.7. 3-regular graphs on 1000 vertices with bisection width b

b	10	20	30	32	34
Seed	35303	75386	38338	99915	70177
KL	(36,168,180)	(48,196,222)	(192,186,172)	(166,54,200)	(194,184,174)
CKL	(14,10,10) (10,10,10)	(20,24,24) (20,24,20)	(30,32,34) (30,32,30)	(34,32,32) (34,34,34)	(34,36,34) (34,36,34)
GD	(216,226,400)	(242,216,226)	(414,414,210)	(230,502,438)	(466,242,254)
SA	372	388	344	318†	380†
MF	10	20	a	b	c
AMF	10	20	30	32	34

$a = 28, b = 32, c = 34$, these values were obtained with the help of a brief visual inspection as explained in the text on page 50.
† data obtained with different parameter values. See discussion on page 50.

Data for degree 4, random regular graphs with given bisection width b.

TABLE 4.1. 4-regular graphs on 100 vertices with bisection width b

b	2			6		
Seed	76869	27319	47952	6018	4632	1702
KL	(2,2,2)	(2,2,2)	(2,2,2)	(6,6,6)	(6,6,6)	(6,6,6)
CKL	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(6,6,6) (6,6,6) (6,6,6)	(6,6,6) (6,6,6) (10,6,6)	(6,6,6) (6,6,6) (6,6,6)
GD	(50,44,44)	(48,50,54)	(42,40,46)	(36,40,36)	(42,46,62)	(46,26,48)
SA	2	2	2	6	6	6
MF	2	2	2	6	6	6
AMF	2	2	2	6	6	6

TABLE 4.2. 4-regular graphs on 100 vertices with bisection width b

b	8			12		
Seed	76936	30014	99651	907	353	66410
KL	(8,8,8)	(8,8,8)	(8,8,8)	(12,12,12)	(12,12,12)	(12,12,12)
CKL	(8,8,8) (8,8,8) (8,8,8)	(8,8,8) (8,8,8) (8,8,8)	(8,10,8) (10,8,12) (8,8,8)	(12,12,12) (12,12,12) (12,12,12)	(12,12,12) (12,12,12) (12,12,12)	(12,12,12) (12,12,12) (12,12,12)
GD	(40,44,50)	(54,46,48)	(48,46,46)	(52,50,50)	(68,42,44)	(46,46,48)
SA	8	8	8	40†	20†	32†
MF	8	8	8	12	12	12
AMF	8	8	8	14	28	20

† data obtained with different parameter values. See discussion on page 50.

TABLE 4.3. 4-regular graphs on 200 vertices with bisection width b

b	2			10		
Seed	80932	12199	92472	70782	19498	85738
KL	(2,2,2)	(2,2,2)	(2,2,2)	(10,10,10)	(10,10,12)	(10,10,10)
CKL	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(10,10,10) (10,10,10) (10,10,10)	(10,10,10) (10,10,10) (10,10,10)	(10,10,10) (10,10,10) (10,10,10)
GD	(86,106,74)	(78,102,86)	(100,86,94)	(82,88,100)	(70,88,94)	(102,102,106)
SA	2	2	2	10	10	10
MF	2	2	2	10	10	10
AMF	2	2	2	10	10	10

TABLE 4.4. 4-regular graphs on 200 vertices with bisection width b

b	14			20		
Seed	842	57597	6877	12950	16298	42255
KL	(14,14,14)	(14,14,14)	(14,14,14)	(20,20,20)	(20,20,20)	(20,20,20)
CKL	(14,14,14) (14,14,14) (14,14,14)	(14,14,14) (14,14,14) (14,14,14)	(14,14,14) (14,14,14) (14,14,14)	(20,22,20) (20,20,20) (20,20,22)	(20,20,20) (20,20,20) (20,22,22)	(20,20,20) (20,20,20) (20,20,20)
GD	(74,88,108)	(86,106,106)	(114,100,72)	(90,106,110)	(118,98,88)	(104,98,84)
SA	14	14	14	20†	34†	28†
MF	14	14	14	20	20	20
AMF	14	14	14	20	22	32

† data obtained with different parameter values. See discussion on page 50.

TABLE 4.5. 4-regular graphs on 400 vertices with bisection width b

b	2	10	14	20	22	24
Seed	82292	74588	55222	14499	74289	65298
KL	(2,2,2)	(10,10,10)	(14,14,14)	(20,20,20)	(22,22,22)	(24,24,24)
CKL	(2,2,2) (2,2,2) (2,2,2)	(10,10,10) (10,10,10) (10,10,10)	(14,14,14) (14,14,14) (14,14,14)	(20,20,20) (20,20,20) (20,20,20)	(22,22,22) (22,22,22) (22,22,22)	(24,24,24) (24,24,24) (24,24,24)
GD	(400,378,366)	(196,192,202)	(208,184,190)	(196,174,184)	(206,178,140)	(200,218,192)
SA	2	10	14	20†	26†	58†
MF	2	10	14	20	22	24
AMF	2	10	14	20	22	26

† data obtained with different parameter values. See discussion on page 50.

TABLE 4.6. 4-regular graphs on 800 vertices with bisection width b

b	2	14	20	28
Seed	83080	20467	52320	97303
KL	(2,2,2)	(14,14,14)	(20,20,20)	(28,28,28)
CKL	(2,2,2) (2,2,2) (2,2,2)	(14,14,14) (14,14,14) (14,14,14)	(20,20,20) (20,20,20) (20,20,20)	(28,28,28) (28,28,28) (28,28,28)
GD	(400,378,366)	(384,404,374)	(394,386,368)	(404,362,390)
SA	30†	14†	54†	28
MF	2	14	20	28
AMF	2	14	20	28

† data obtained with different parameter values. See discussion on page 50.

TABLE 4.7. 4-regular graphs on 800 vertices with bisection width b

b	34	42	50
Seed	83107	3051	52224
KL	(34,34,34)	(42,42,42)	(50,50,50)
CKL	(34,34,34) (34,34,34) (34,34,34)	(42,42,42) (42,42,42) (42,42,42)	(50,50,50) (50,50,50) (50,50,50)
GD	(358,396,424)	(390,354,402)	(404,354,384)
SA	74†	58†	80†
MF	34	42	50
AMF	36	44	50

† data obtained with different parameter values. See discussion on page 50.

TABLE 4.8. 4-regular graphs on 1000 vertices with bisection width b

b	10	20	30	40	50
Seed	84932	36824	22071	51220	9816
KL	(10,10,10)	(20,20,20)	(30,30,30)	(40,40,44)	(50,50,50)
CKL	(10,10,10)	(20,20,20)	(30,30,30)	(40,40,40)	(50,50,50)
GD	(518,490,492)	(476,492,512)	(526,476,468)	(504,502,462)	(468,496,478)
SA	58†	44†	30†	402†	152†
MF	10	20	30	a	b
AMF	10	20	30	40	50

$a = 40, b = 50$, these values were obtained with the help of a brief visual inspection as explained in the text on page 50.
† data obtained with different parameter values. See discussion on page 50.

Data for degree 5, random regular graphs with given bisection width b.

TABLE 5.1. 5-regular graphs on 100 vertices with bisection width b

b	2		10		14	
Seed	37552	98456	71564	15052	4885	43587
KL	(2,2,2)	(2,2,2)	(10,10,10)	(10,10,10)	(14,14,14)	(14,14,14)
CKL	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(10,10,10) (10,10,10) (10,10,10)	(10,10,10) (10,10,10) (10,10,10)	(14,14,14) (14,14,14) (14,14,14)	(14,14,14) (14,14,14) (14,14,14)
GD	(2,40,2)	(2,56,58)	(52,10,56)	(10,46,10)	(50,76,64)	(66,64,28)
SA	2	2†	10†	10†	14	14
MF	2	2	10	10	14	14
AMF	2	2	10	10	14	14

† data obtained with different parameter values. See discussion on page 50.

TABLE 5.2. 5-regular graphs on 100 vertices with bisection width b

b	18		22	
Seed	39835	22833	88519	18175
KL	(18,18,18)	(18,18,18)	(22,22,22)	(22,22,22)
CKL	(18,18,18) (18,18,18) (18,18,18)	(18,18,18) (18,18,18) (18,18,18)	(22,22,22) (22,22,22) (22,24,22)	(22,22,22) (22,22,22) (22,22,22)
GD	(38,50,58)	(38,34,60)	(62,74,64)	(56,82,18)
SA	62	20	26	22
MF	18	18	22	22
AMF	18	18	26	26

TABLE 5.3. 5-regular graphs on 200 vertices with bisection width b

b	2		20	
Seed	45562	59744	92082	58228
KL	(2,2,2)	(2,2,2)	(20,20,20)	(20,20,20)
CKL	(2,2,2) (2,2,2) (2,2,2)	(2,2,2) (2,2,2) (2,2,2)	(20,20,20) (20,20,20) (20,20,20)	(20,20,20) (20,20,20) (20,20,20)
GD	(124,158,140)	(2,30,106)	(120,116,142)	(126,40,115)
SA	2	2	20	20
MF	2	2	20	20
AMF	2	2	20	20

TABLE 5.4. 5-regular graphs on 200 vertices with bisection width b

b	26		32	
Seed	43479	68838	16980	95669
KL	(26,26,26)	(26,26,26)	(32,32,32)	(32,32,32)
CKL	(26,26,26) (26,26,26) (26,26,26)	(26,26,26) (26,26,26) (26,26,26)	(32,32,32) (32,32,32) (32,32,32)	(32,32,32) (32,32,38) (32,32,32)
GD	(26,126,26)	(98,136,108)	(152,80,116)	(164,104,114)
SA	32†	30†	32	32†
MF	26	26	32	32
AMF	26	28	32	32

† data obtained with different parameter values. See discussion on page 50.

TABLE 5.5. 5-regular graphs on 400 vertices with bisection width b					
b	2	14	20	30	36
Seed	94008	54921	6403	40405	59897
KL	(2,2,2)	(14,14,14)	(20,20,20)	(30,30,30)	(36,36,36)
CKL	(2,2,2) (2,2,2) (2,2,2)	(14,14,14) (14,14,14) (14,14,14)	(20,20,20) (20,20,20) (20,20,20)	(30,30,30) (30,30,30) (30,30,30)	(36,36,36) (36,36,36) (36,36,36)
GD	(272,2,224)	(14,298,14)	(308,236,20)	(266,234,314)	(214,180,218)
SA	2	14	20	30	40
MF	2	14	20	30	36
AMF	2	14	20	30	40

TABLE 5.6. 5-regular graphs on 800 vertices with bisection width b			
b	2	20	40
Seed	12457	19301	42694
KL	(2,2,2)	(20,20,20)	(40,40,40)
CKL	(2,2,2) (2,2,2) (2,2,2)	(20,20,20) (20,20,20) (20,20,20)	(40,40,40) (40,40,40) (40,40,40)
GD	(430,2,2)	(580,450,448)	(454,440,512)
SA	32	26	44†
MF	2	20	40
AMF	2	20	40

† data obtained with different parameter values. See discussion on page 50.

TABLE 5.7. 5-regular graphs on 800 vertices with bisection width b			
b	50	70	94
Seed	54273	77409	84494
KL	(50,50,50)	(70,70,70)	(94,94,94)
CKL	(50,50,50) (50,50,50) (50,50,50)	(70,70,70) (70,70,70) (70,70,70)	(94,94,94) (94,94,94) (94,94,94)
GD	(634,622,492)	(604,444,460)	(532,456,378)
SA	110	78	138
MF	50	70	94
AMF	52	80	100

TABLE 5.8. 5-regular graphs on 1000 vertices with bisection width b				
b	10	30	50	70
Seed	26386	226	2373	24974
KL	(10,10,10)	(30,30,30)	(50,50,50)	(70,70,70)
CKL	(10,10,10)	(30,30,30)	(50,50,50)	(70,70,70)
GD	(520,586,556)	(534,548,736)	(550,550,58)	(550,476,436)
SA	18†	30†	70†	84†
MF	10	30	50	70
AMF	10	30	50	72

† data obtained with different parameter values. See discussion on page 50.

TABLE 5.9. 5-regular graphs on 1000 vertices with bisection width b			
b	90	110	130
Seed	52305	10643	32841
KL	(90,90,90)	(110,110,110)	(128,128,128)
CKL	(90,90,90)	(110,110,110)	(128,132,128)
GD	(540,536,524)	(550,542,518)	(536,542,782)
SA	116†	126†	142†
MF	90	110	128
AMF	106	120	138

† data obtained with different parameter values. See discussion on page 50.

6.2. New Graph Bisection Heuristics

The data in the previous section on the performance of KL, SA, and GD seems to indicate that these algorithms work better for graphs with high degree. This observation suggests an interesting variation of the algorithms for small degree graphs, namely to: 1) randomly contract edges out of the graph to increase the average node degree, 2) run the algorithm to find a bisection of the contracted graph, 3) uncontract the edges to obtain the original graph (the bisection of the contracted graph found in step 2 might not be a bisection of the original graph, in that case randomly adjust that bisection to obtain a bisection for the original graph) and 4) run the algorithm again, starting with this new bisection.

We have implemented such a modified algorithm for the KL algorithm and it is called the contracted Kernighan-Lin algorithm (CKL). The actual CKL algorithm is as follows. Find a random maximal matching of the given graph. Form a new graph by contracting the edges in the random matching, i.e., coalesce the two endpoints of an edge in the random matching to form a new vertex. This new vertex now has degree $2d - 1$. To keep the new graph from having an odd number of vertices we might have to leave one edge of the random matching uncontracted. We then run KL on this contracted graph to obtain a bisection. The graph is then uncontracted and if necessary the found bisection is randomly adjusted to obtain a bisection for the original graph. This bisection is now used as a starting bisection to run KL again. For small graphs, we usually tried 3 random contractions for each graph.

As can be seen from the data in the previous section, this algorithm seemed to perform remarkably well. Even for degree 3 graphs, CKL almost always found the optimal solution. (A related strategy is also used in a heuristic by Goldberg and Burstein in [GB84] to upgrade the performance of bisection algorithms, although less dramatic results are observed.)

It is also worth noting that even though the KL algorithm is used twice in CKL, CKL in fact appears to run faster than KL alone on the same graph. The reason seems to be that starting with a random bisection KL usually takes many passes before it reaches a local minimum, whereas when KL starts with a bisection from the contracted graph in the second stage of CKL, it converges much more quickly (in 2 or 3 passes). Furthermore, the time KL spent on the contracted graph is also small since for a random matching the contracted graph is usually about half the size of the original graph. Also the random

matching does not take very much time either.

Thus CKL seems to perform better than KL for small degree graphs in both efficiency and quality of the solutions. For larger degree graphs, there is essentially no difference in the quality of the solution, however, CKL again seems to run faster.

We now describe a new algorithm which is also based on the maxflow-mincut algorithm. The algorithm, which will be denoted by AMF, works as follows. For each pair of vertices the algorithm performs the following routine to obtain a bisection. It will return the smallest bisection found.

For each pair of vertices, let one be the source and the other be the sink. Run maxflow-mincut algorithm to determine a mincut separating the source and the sink. If this mincut exactly bisects the graph, then we are done with this pair of vertices. Consider the next pair of vertices. Otherwise, there is one side of the cut with fewer than $n/2$ vertices, say the side containing the source. Make that side into a super-source, i.e., coalesce every vertex in that side into one big vertex, and every edge incident to that side will now be incident to that new super-vertex. Then run maxflow again between the super-source and the old sink. Repeat the above process until the difference between the two sides is no more than a predetermined number of vertices, say 10. Then we will use a greedy type procedure to balance out the partition to obtain a bisection.

This algorithm seems to work well for small bisection width graphs since these small bisections are usually the bottleneck when the source and sink are on opposite sides of an optimal bisection. We have tested a probabilistic version of this algorithm. The algorithm just chose 10 pairs of vertices at random and returned the best bisection found. The data in the previous section indicated that for small degree graphs and also for graphs with small bisection width AMF performed as well as or better than KL and SA. Certainly it seemed to perform much better than GD. This algorithm also ran much faster than KL, SA, and GD as well as MF. The reason seems to be that it converged rather quickly, usually taking no more than 3 or 4 passes, i.e., maxflow-mincut runs.

6.3. Guide to Pratical Use of Our Algorithms

Since the major part of this thesis is a theoretical study of the graph bisection problem, not all the algorithms presented in this thesis can be used directly for all graphs in practice. The purpose of this section is to give the practitioner some of the main ideas of this thesis that can be used in devising practical graph bisection algorithms.

The algorithms that we have given in Chapter 4 are found to perform well in ex-

periments with graphs from $\mathcal{G}(n,d,b)$, for $b = o(n^{1-1/\lfloor\frac{d+1}{2}\rfloor})$ as expected from the analysis. For the case of graphs with larger bisection width, it is not clear that Algorithm 2 can be used directly without any modification. The reason is that, intuitively, graphs with large bisection width tend to have more than one optimal bisection, whereas Algorithm 2 does rely on the fact that graphs in $\mathcal{G}(n,d,b)$ almost always have a small and unique optimal bisection. Nonetheless, we believe the idea of using the maxflow technique will still be of help here. One such application of the maxflow technique is the AMF algorithm described in the previous section.

To use our algorithms for graphs with small bisection width in practice, we offer the following suggestions which we have found to be useful in our experiments. First, we observe that the two algorithms in Chapter 4 and algorithm AMF described in the previous section of this chapter all run some main procedure on all pairs of vertices. In practice a substantial amount of running time can be saved by running the algorithms on a number of randomly chosen pairs of vertices. In that case only Step 3 of Algorithm 2 needs to be modified slightly to work correctly.

Another modification can be made to Algorithm 2 as follows. If the set of edges picked out by the algorithm does not make up a bisection but is close to a bisection then we can use a greedy heuristic to obtain a bisection. Alternatively, we can remove this set of edges and run the algorithm again.

Another important idea that can be used in designing graph bisection algorithms is that of contraction. The contraction heuristic seems to work well for graphs with small degree and for any bisection width. This heuristic is very useful when combined with algorithms which work well for large degree graphs but fail to work well for small degree graphs. In this thesis we have tested the contraction heuristic with the Kernighan-Lin algorithm as described in the previous section. The contraction heuristic not only improved the performance of the Kernighan-Lin algorithm on small degree graphs, but it also improved the running time. The idea of contraction can be carried further by doing repeated contraction. Although it is not clear if this will further improve the performance of the algorithms involved, it will require a greater programming effort.

Chapter 7

Conclusion

In this thesis we have introduced the model $\mathcal{G}(n,d,b)$ of random regular graphs with bisection width b, where $b = o(n^{1-1/\lfloor\frac{d+1}{2}\rfloor})$. We have argued that this model is more suitable for the testing and analyzing of graph bisection algorithms. We then presented two graph bisection algorithms, one for small bisection width graphs and one for larger bisection width graphs, with provably good average performance on $\mathcal{G}(n,d,b)$. Particularly, we showed that for fixed $d \geq 3$, for almost every graph in $\mathcal{G}(n,d,b)$ our algorithms find the optimal bisection. Furthermore, we showed that whenever the algorithms produce a bisection it is guaranteed to be optimal.

Our experiments also showed that our algorithms performed as well as expected. We also tested other algorithms such as greedy, Kernighan-Lin and simulated annealing. We found them to perform badly on cubic graphs but their performance improves as the degree gets larger. In fact, we showed that for cubic graphs, starting with a random bisection, the greedy algorithm will not be able to find the optimal bisection with probability $1 - o(1)$.

Finally, based on our observation of the data we suggested a heuristic which can be combined with the various algorithms to improve their performance for graphs with small degree. One such modification was implemented for the Kernighan-Lin algorithm and the results were very good.

We also introduced a new heuristic which used the maxflow idea of our main algorithms. Experiments showed that the performance of this algorithm is very good and is comparable to or better than the Kernighan-Lin and simulated annealing algorithms.

The main open question that remains to be resolved is the construction of an approximate algorithm for the graph bisection problem. We suspect that maxflow-based techniques will be of some help in doing that. It would also be nice to show that the

greedy algorithm will be able to find the optimal bisection for graphs in $\mathcal{G}(n, d, b)$ with high probability when d is large enough.

References

[Ba83] Baker, B., "Approximation Algorithms for NP-complete Problems on Planar Graphs," FOCS 1983, pp. 265–273.

[Bar81] Barnes, E. R., "An Algorithm for Partitioning the Nodes of a Graph," IBM Techical Report RC8690, 1981.

[BH] Barnes, E. R. and A. J. Hoffman, "Partitioning, Spectra and Linear Programming," IBM Research Report, (unknown date).

[BL84] Bhatt, S. N. and F. T. Leighton, "A Framework for Solving VLSI Graph Layout Problems," Journal of Computer and System Sciences, Vol. 28, No. 2, 1984.

[Bo79] Bollobás, B., Graph Theory : An Introductory Course, Springer Verlag, New York, (1979).

[Bo80] Bollobás, B., "A Probabilistic Proof of an Asymptotic Formula for the Number of Labelled Regular Graphs," European J. Combinatorics, 1(1980), pp. 311–316.

[BF82] Bollobás, B. and W. Fernandez de la Vega, "The Diameter of Random Regular Graphs," Combinatorica, 2 (1982), pp. 125–134.

[Br77] Breuer, M. A., "Min-cut Placement," J. Design Aut. and Fault Tol. Comp., Vol. 1, No. 4, Oct. 1977, pp. 343–362.

[Bu83] Bui, T. N., *On Bisecting Random Graphs*, S.M. Thesis, Dept. of Electrical Engineering and Computer Science, (1983). Also appears as M.I.T. LCS Technical Report 287.

[Bu84] Bui, T., S. Chaudhuri, T. Leighton, and M. Sipser, "Graph Bisection Algorithms With Good Average Case Behavior," Proceedings of the 25th Symposium on the Foundation of Computer Science, 1984, pp. 181–192.

[Bu86] Bui, T., S. Chaudhuri, T. Leighton, and M. Sipser, "Graph Bisection Algorithms With Good Average Case Behavior," to appear in Combinatorica, 1986.

[DH73] Donath, W. E., and A. J. Hoffman, "Lower Bounds for the Partitioning of Graphs," IBM J. Res. Develop., Vol. 17, Sept. 1973, pp. 420–425.

[ER59] Erdős, P. and A. Rényi, "On Random Graphs, I," Publicationes Mathematicae, 6, 1959, pp. 290–297.

[ER60] Erdős, P. and A. Rényi, "On the Evolution of Random Graphs," Publications of the Mathematical Institute of the Hungarian Academy of Sciences, 5(1960), pp. 17–60.

[Fe68] Feller, W., *An Introduction to Probability Theory and Its Applications*, Volume 1, third Edition, John Wiley and Sons, 1968.

[FM82] Fiduccia, C. M., and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," Proceedings of the 19th Design Automation Conference, January 1982, pp. 175–181, IEEE Computer Society Press.

[GJ79] Garey, M. R., and D. S. Johnson, Computers and Intractibility : A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[GJS76] Garey, M. R., D. S. Johnson, and L. Stockmeyer, "Some Simplified NP-complete Graph Problems," Theoretical Computer Science, 1(1976), pp.237–267.

[GB84] Goldberg, M. K. and M. Burstein, "Heuristic Improvement Techniques for Bisection of VLSI Networks," Unpublished Manuscript, 1984, Clarkson University.

[GG84] Goldberg, M. K. and R. Gardner, "On the Minimal Cut Problem," Progress in Graph Theory, Edited by J. A. Bondy and U. S. R. Murty, Academic Press, 1984, pp. 295–305.

[J84] Johnson, D. S., Personal communication.

[KL70] Kernighan, B. W., and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," The Bell System Tech. J., Vol. 49, No. 2, Feb. 1970, pp. 291–307.

[KGV83] Kirkpatrick, S., C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," IBM Research Report RC 9355, April 1982, Yorktown Heights, New York.

[LT77] Lipton, R. J. and R. E. Tarjan, "Applications of a Planar Separator Theorem," Proceedings of the 18th Symposium on the Foundation of Computer Science, 1977, pp. 162–170.

[LT79] Lipton, R. J. and R. E. Tarjan, "A Separator Theorem for Planar Graphs," SIAM J. Computing, Vol. 36, No. 2, April 1979, pp. 177–189.

[Mac78] Macgregor, R. M., "On Partitioning a Graph : A Theoretical and Empirical Study," Ph.D. Thesis, University of California, Berkeley, 1978, also appeared as Technical Report UCB/ERL M78/14.

[Ri82] Rivest, R. L., "The "PI" (Placement and Interconnect) System," Proceedings of the 19th Annual Design Automation Conference, 1982, pp. 475–481, IEEE Computer Society Press.

Appendix

In this appendix we give a listing of the program that we used to generate the graphs tested in Chapter 6. The program is written in Zetalisp and run on the Symbolics 3600 Lisp Machine, release version 6.1.

```
(eval-when (compile)
  (special *half-size* *bisection* *deg* *counter* *not-legal* *points1* *points2* n
           *bis1* *bis2* *bisection-edge-list* *int11* *int12* *int21* *int22*
           *same-headvertex* *same-tailvertex* *vertices* *perm-vertices*
           *vertex-array* *old-vertex-array* *temp-vertex-array* *max-times* max-times
           *random-number* *large-num* *number-graphs* *seed*
           *multiplier* *increment* *modulus* *continue*))

;; creating arrays for storing points
;; initializing random seed

(defun grg-setup ()
  (setq *points1* (make-array (list (* *half-size* *deg*)) ':type 'art-q))
  (setq *points2* (make-array (list (* *half-size* *deg*)) ':type 'art-q))
  (cond ((= *bisection* 0) nil)
        (t
         (setq *bis1* (make-array (list *bisection*) ':type 'art-q))
         (setq *bis2* (make-array (list *bisection*) ':type 'art-q))))
  (setq *int11* (make-array (list (// (- (* *half-size* *deg*) *bisection*) 2)) ':type 'art-q))
  (setq *int12* (make-array (list (// (- (* *half-size* *deg*) *bisection*) 2)) ':type 'art-q))
  (setq *int21* (make-array (list (// (- (* *half-size* *deg*) *bisection*) 2)) ':type 'art-q))
  (setq *int22* (make-array (list (// (- (* *half-size* *deg*) *bisection*) 2)) ':type 'art-q))
  (setq *same-headvertex* (make-array (list *half-size*) ':type 'art-q))
  (setq *same-tailvertex* (make-array (list *half-size*) ':type 'art-q))
  (setq *vertices* (make-array (list (* 2 *half-size* *deg*)) ':type 'art-q))
  (setq *vertex-array* (make-array (list (* 2 *half-size*) *deg*) ':type 'art-q))
  (setq *old-vertex-array* (make-array (list (* 2 *half-size*) *deg*) ':type 'art-q))
  (setq *temp-vertex-array* (make-array (list (* 2 *half-size*) *deg*) ':type 'art-q))
  (setq *perm-vertices* (make-array (list (* 2 *half-size*)) ':type 'art-q)))


;; initializing some constants

(defun grg-set-constants (seed)
  (setq max-times 0)
  (setq *max-times* 10)
  (setq *large-num* seed)
  (setq *random-number* 0)
  (setq *increment* 76543)
  (setq *multiplier* 895432)
  (setq *modulus* 43896721))
```

```
;; generating graphs of size 2n, with bisection width b and degree d

(defun grg-gen-graph (size bisection deg)
  (format t "~2% A random ~3d -regular graph on ~6d vertices
and bisection ~6d is being generated ~1%" deg (* 2 size) bisection)
  (setq *not-legal* t *half-size* size *bisection* bisection *deg* deg
        *number-graphs* 500 *counter* 0)
  (cond ((oddp (- (* *half-size* *deg*) *bisection*))
         (print "WRONG VALUE OF ARGUMENTS"))
        (t
         (grg-setup)
         (loop while (and *not-legal* (> *number-graphs* *counter*))
               do
           (setq *seed* (random 100000))
           (grg-set-constants *seed*)
           (grg-gen-graph-aux)
           (grg-check-legal))))
  (cond (*not-legal*
         (setq *continue* nil)
         (format t "~1% Fail to construct a graph after ~4d trials.  Try again; you
          may want to increase the number of trials (*number-graphs*)" *number-graphs*))
        (t
         (setq *continue* t)
         (format t "~1% Success on trial number ~4d" *counter*)
         (format t "~1% The seed is ~6d" *seed*))))


;;; this is the same as the above function except that the *seed* is
;;; given

(defun grg-gen-graph-seed (size bisection deg seed)
  (format t "~2% A random ~3d -regular graph on ~6d vertices
and bisection ~6d is being generated ~1%" deg (* 2 size) bisection)
  (setq *half-size* size *bisection* bisection *deg* deg
        *counter* 0 *seed* seed *not-legal* t)
  (cond ((oddp (- (* *half-size* *deg*) *bisection*))
         (print "WRONG VALUE OF ARGUMENTS"))
        (t
         (grg-setup)
         (grg-set-constants *seed*)
         (grg-gen-graph-aux)
         (setq *not-legal* nil)))
  (cond (*not-legal*
         (setq *continue* nil)
         (format t "~1%  Can't construct a graph"))
        (t
         (setq *continue* t)
         (format t "~1% Success on trial number 1")
         (format t "~1% The seed is ~6d" *seed*))))


;; generates a random deg graph without eliminating multiple edges and loops

(defun grg-gen-graph-and-loops (size bisection deg)
  (format t "~2% A random ~3d -regular graph on ~6d vertices
and bisection ~6d is being generated ~1%" deg (* 2 size) bisection)
  (setq  *half-size* size *bisection* bisection *deg* deg
         *counter* 0)
  (cond ((oddp (- (* *half-size* *deg*) *bisection*))
         (print "WRONG VALUE OF ARGUMENTS"))
        (t
         (grg-setup)
         (setq *seed* (random 100000))
         (grg-set-constants *seed*)
         (grg-gen-graph-aux)))
  (setq *continue* t)
  (format t "~1% Success on trial number ~4d" *counter*)
  (format t "~1% The seed is ~6d" *seed*))
```

```
;; an auxiliary procedure which increments the counter.  Additional comments will
;; be provided for each function.

(defun grg-gen-graph-aux ()
  (setq *counter* (1+ *counter*))
  (grg-fill-*points1-and-2*)
  (grg-choose-points)
  (grg-create-*vertices*)
  (grg-create-*vertex-array*))


;; This procedure fills the arrays *points1* and *points2* with arbitrary numbers
;; since the function grg-create-*vertices* creates a variable points from entries of
;; *points1* and *points2*.  Points must be a numeric argument, so if either array
;; contains any NILs an error may occur.

(defun grg-fill-*points1-and-2* ()
  (loop for i from 0 to (1- (* *deg* *half-size*))
        do (aset i *points1* i)
           (aset i *points2* i)))


;; creates a set of points for the bisection edges if a bisection is called for, then
;; creates a set of points for the interior edges.

(defun grg-choose-points ()
  (if (not (= *bisection* 0))
      (grg-choose-bisection-points))
  (grg-choose-interior-points))

(defun grg-choose-bisection-points ()
  (grg-create-*bis1*)
  (grg-create-*bis2*))

(defun grg-choose-interior-points ()
  (grg-create-*int11*)
  (grg-create-*int12*)
  (grg-create-*int21*)
  (grg-create-*int22*))


;; Randomly selects points which will be the boundaries of the bisection edges
;; (hence the bisection edges will also be random).  grg-create-*bis1* selects
;; points for one half of the graph; grg-create-*bis2* selects points for the other.

(defun grg-create-*bis1* ()
  (loop for i from 0 to (1- *bisection*)
        do (let* ((j (random-between i (1- (* *half-size* *deg*))))
                  (i-val (aref *points1* i))
                  (j-val (aref *points1* j)))
             (aset j-val *points1* i)
             (aset i-val *points1* j)
             (aset j-val *bis1* i))))


(defun grg-create-*bis2* ()
  (loop for i from 0 to (1- *bisection*)
        do (let* ((j (random-between i (1- (* *half-size* *deg*))))
                  (i-val (aref *points2* i))
                  (j-val (aref *points2* j)))
             (aset j-val *points2* i)
             (aset i-val *points2* j)
             (aset j-val *bis2* i))))
```

```
;; Randomly selects points which will be the boundaries of the interior edges
;; of the graph (hence the interior edges will also be random).  grg-create-*int11*
;; selects the points which will be one boundary of the interior edges in one side
;; of the graph; grg-create-*int12* selects the points which will be the other
;; boundaries of the edges on that side.  grg-create-*int21* and grg-create-*int22*
;; do the same for the other half of the graph.

(defun grg-create-*int11* ()
  (loop for i from 0 to (1- (// (- (* *half-size* *deg*) *bisection*) 2))
        do (let* ((j (random-between (+ i *bisection*) (1- (* *half-size* *deg*))))
                  (i-val (aref *points1* (+ i *bisection*)))
                  (j-val (aref *points1* j)))
             (aset j-val *points1* (+ i *bisection*))
             (aset i-val *points1* j)
             (aset j-val *int11* i))))

(defun grg-create-*int12* ()
  (loop for i from 0 to (1- (// (- (* *half-size* *deg*) *bisection*) 2))
        do (let* ((val (aref *points1* (+ i (// (+ (* *half-size* *deg*)
                                                   *bisection*) 2)))))
             (aset val *int12* i))))

(defun grg-create-*int21* ()
  (loop for i from 0 to (1- (// (- (* *half-size* *deg*) *bisection*) 2))
        do (let* ((j (random-between (+ i *bisection*) (1- (* *half-size* *deg*))))
                  (i-val (aref *points2* (+ i *bisection*)))
                  (j-val (aref *points2* j)))
             (aset j-val *points2* (+ i *bisection*))
             (aset i-val *points2* j)
             (aset j-val *int21* i))))

(defun grg-create-*int22* ()
  (loop for i from 0 to (1- (// (- (* *half-size* *deg*) *bisection*) 2))
        do (let* ((val (aref *points2* (+ i (// (+ (* *half-size* *deg*)
                                                   *bisection*) 2)))))
             (aset val *int22* i))))


;; grg-update-*points1* and grg-update-*points2* update the points in *points1* and
;; *points2* after multiple edges and loops are removed

(defun grg-update-*points1* ()
  (loop for i from *bisection* to (1- (// (+ (* *half-size* *deg*) *bisection*) 2))
        do (aset (aref *int11* (- i *bisection*)) *points1* i))

  (loop for i from (// (+ (* *half-size* *deg*) *bisection*) 2) to (1- (* *half-size* *deg*))
        do (aset (aref *int12* (- i (// (+ (* *half-size* *deg*) *bisection*) 2)))
                 *points1* i))
  (if (not (= *bisection* 0))
      (loop for i from 0 to (1- *bisection*)
            do (aset (aref *bis1* i) *points1* i))))


(defun grg-update-*points2* ()
  (loop for i from *bisection* to (1- (// (+ (* *half-size* *deg*) *bisection*) 2))
        do (aset (aref *int21* (- i *bisection*)) *points2* i))
  (loop for i from (// (+ (* *half-size* *deg*) *bisection*) 2) to (1- (* *half-size* *deg*))
        do (aset (aref *int22* (- i (// (+ (* *half-size* *deg*) *bisection*) 2)))
                 *points2* i))
  (if (not (= *bisection* 0))
      (loop for i from 0 to (1- *bisection*)
            do (aset (aref *bis2* i) *points2* i))))
```

```
;; grg-create-*vertex-array* stores, at the index of vertex in *vertex-array*, the
;; vertices connected to vertex

(defun grg-create-*vertex-array* ()
  (loop for vertex from 0 to (1- (* 2 *half-size*))     do
    (let ((num 0))
      (cond ((< vertex *half-size*)
             (loop for i from 0 to (1- *bisection*) do          ;; points in *bis1*
               (if (< num *deg*)
                   (cond ((= (aref *vertices* i) vertex)
                          (aset (aref *vertices* (+ i (* *half-size* *deg*)))
                                *vertex-array* vertex num)
                          (setq num (1+ num)))
                         (t nil))))
             ;; points in *int11*
             (loop for i from *bisection* to (1- (// (+ (* *half-size* *deg*) *bisection*) 2))
                   do
               (if (< num *deg*)
                   (cond ((= (aref *vertices* i) vertex)
                          (aset (aref *vertices*
                                      (+ i (// (- (* *half-size* *deg*) *bisection*) 2)))
                                *vertex-array* vertex num)
                          (setq num (1+ num)))
                         (t nil))))
             ;; points in *int12*
             (loop for i from (// (+ (* *half-size* *deg*) *bisection*) 2)
                         to (1- (* *half-size* *deg*)) do
               (if (< num *deg*)
                   (cond ((= (aref *vertices* i) vertex)
                          (aset (aref *vertices*
                                      (- i (// (- (* *half-size* *deg*) *bisection*) 2)))
                                *vertex-array* vertex num)
                          (setq num (1+ num)))
                         (t nil)))))
                                        ; points in *bis2*
            (t (loop for i from (* *half-size* *deg*) to (1- (+ (* *half-size* *deg*)
                                                               *bisection*)) do
                 (if (< num *deg*)
                     (cond ((= (aref *vertices* i) vertex)
                            (aset (aref *vertices* (- i (* *half-size* *deg*)))
                                  *vertex-array* vertex num)
                            (setq num (1+ num)))
                           (t nil))))
                                        ; points in *int21*
               (loop for i from (+ (* *half-size* *deg*) *bisection*)
                         to (1- (+ (* *half-size* *deg*) (// (+ (* *half-size* *deg*)
                                                                *bisection*) 2))) do
                 (if (< num *deg*)
                     (cond ((= (aref *vertices* i) vertex)
                            (aset (aref *vertices*
                                        (+ i (// (- (* *half-size* *deg*) *bisection*) 2)))
                                  *vertex-array* vertex num)
                            (setq num (1+ num)))
                           (t nil))))
                                        ; points in *int22*
               (loop for i
                     from (+ (* *half-size* *deg*)
                             (// (+ (* *half-size* *deg*) *bisection*) 2))
                     to (1- (* 2 (* *half-size* *deg*))) do
                 (if (< num *deg*)
                     (cond ((= (aref *vertices* i) vertex)
                            (aset (aref *vertices*
                                        (- i (// (- (* *half-size* *deg*) *bisection*) 2)))
                                  *vertex-array* vertex num)
                            (setq num (1+ num)))
                           (t nil)))))))))
```

```
;; grg-create-*vertices* stores the vertices of the graph

(defun grg-create-*vertices* ()
  (loop for i from 0 to (1- (* *half-size* *deg*))
        do (let* ((point (aref *points1* i)))
             (aset (grg-find-vertex point) *vertices* i)))

  (loop for i from (* *half-size* *deg*) to (1- (* 2 (* *half-size* *deg*)))
        do (let* ((point (aref *points2* (- i (* *half-size* *deg*)))))
             (aset (+ *half-size* (grg-find-vertex point)) *vertices* i))))

;; grg-change-ordering randomly permutes the ordering of the vertices of one half
;; of the bisected graph and creates the new vertex-array

(defun grg-change-ordering ()
  (grg-create-*perm-vertices*)
  (copy-array *old-vertex-array* *vertex-array*)
  (loop for vertex from 0 to (1- (* 2 *half-size*))
        do (let* ((new-vertex (aref *perm-vertices* vertex)))
             (loop for num from 0 to (1- *deg*)
                   do (let* ((change-vertex (aref *old-vertex-array* vertex num)))
                        (loop for num1 from 0 to (1- *deg*)
                              do (if (= (aref *old-vertex-array* change-vertex num1)
                                        vertex)
                                     (aset new-vertex *vertex-array* change-vertex num1)))))))
  (copy-array *old-vertex-array* *vertex-array*)
  (loop for vertex from 0 to (1- (* 2 *half-size*))
        do (let* ((new-vertex (aref *perm-vertices* vertex)))
             (loop for num from 0 to (1- *deg*)
                   do (aset (aref *old-vertex-array* vertex num)
                            *vertex-array* new-vertex num)))))

;; grg-create-*perm-vertices* fills *perm-vertices* with numbers from 1 to
;; (1- (* 2 *half-size)), then randomly permutes these numbers to form the
;; final *perm-vertices*

(defun grg-create-*perm-vertices* ()
  (loop for i from 0 to (1- (* 2 *half-size*))
        do (aset i *perm-vertices* i))
  (loop for i from (1- (* 2 *half-size*)) downto 1
        do (let* ((j (random i))
                  (i-val (aref *perm-vertices* i))
                  (j-val (aref *perm-vertices* j)))
             (aset j-val *perm-vertices* i)
             (aset i-val *perm-vertices* j))))


;; grg-check-legal checks the graph for multiple edges and loops, and it sets
;; *not-legal* to t if any occur

(defun grg-check-legal ()
  (setq *not-legal* nil)
  (loop for vertex from 0 to (1- (* 2 *half-size*)) until *not-legal*
        do (loop for num from 1 to (1- *deg*) until *not-legal*
                 do (loop for num1 from 0 to (1- num) until *not-legal*
                          do (cond ((= (aref *vertex-array* vertex num)
                                       (aref *vertex-array* vertex num1))
                                    (setq *not-legal* t))
                                   (t nil))))))


;; since each vertex is represented by deg points, grg-find-vertex returns the
;; vertex represented by a given point.

(defun grg-find-vertex (point)
  (fix (// point *deg*)))
```