Approximation Algorithms for Finding Planar and Highly Connected Subgraphs

A THESIS
Presented to
The Academic Faculty

by

Cristina G. Fernandes

In Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Georgia Institute of Technology
September 1997

Approximation Algorithms for Finding Planar and Highly Connected Subgraphs

Approved:

Prof. Howard Karloff, Chairman

Prof. Dana Randall

Prof. Leonard Schulman

Prof. Robin Thomas

Prof. H. Venkateswaran

Date Approved by Chairman

Dedicated to

my father (in memoriam)

and my mother

Acknowledgments

I thank my advisor, Prof. Howard Karloff, for introducing me to the field of approximation algorithms, for his patience, enthusiasm and support. He gave me the opportunity to participate closely in all that was happening on my area of research. His availability and willingness to help were essential for this work to be completed. I really appreciate his careful reading of the dissertation, which improved a lot the write-up.

Prof. Robin Thomas and Prof. Vijay Vazirani were also very important. I am grateful to Prof. Robin Thomas for many helpful discussions, for his interest and encouragement. I worked with Prof. Vijay Vazirani during my advisor's leave for sabbatical, and the results from this work appear in the second part of my thesis (the part about the *Minimum k-Edge-Connected Spanning Subgraph* problem).

I thank the reading committee for their comments and suggestions. Also, I would like to thank the anonymous referees of SODA'96 for their comments and suggestions on one of the papers that contains results of the first part of my thesis (the part about the *Maximum Planar Subgraph problem*, which is joint work with Gruia Călinescu and my advisor). There was an independent submission, by Ulrich Finkler, to this same conference with a subset of our results, so the committee also suggested that we combined our paper and his.

Gruia Călinescu has helped me at many moments during these years. I thank him for his help and patience. Ion Mandoiu and Naven Garg listened to my ideas at some point, and I thank them for their insights and suggestions.

During these five years, I received financial support from several sources. I would not have come to Georgia Tech without the CoC support—a GTA position during my first two quarters here. I am grateful to Prof. Amir and Prof. Venkat for getting me that position, and to Prof. Wakabayashi (University of São Paulo) for asking this support. CNPq—Conselho Nacional de Desenvolvimento Científico e Technológico (Brazil)—supported me during four years. My advisor and CoC supported me during my last quarters in Georgia Tech. Also, I was able to attend several conferences and meetings thanks to the funds provided by my advisor, ACO, CoC and Georgia Tech.

But not only from school and money one lives! Friends are very important to give us strength and emotional support. For their importance in my life during these years, I thank Dilma, Luís Carlos, Mary, Tito, Lúcio, Sueli and David. Also friends far in distance, but close in thought: Wânia, Nilsa, Márcia, Carlinhos and Cláudia. Each of them is in part responsible for my accomplishment. Thanks also to Dinei for his effort and love, specially during the critical phase of writing this thesis.

Last but not least, I would like to thank my family... my mom and my brothers, Cláudio and Flávio, with their love and support, provided me with much of the energy I needed during these five years. I thank my father, for having set an example that I try to follow.

Contents

List of Figures

Summary

Planarity and connectivity are important concepts in graph theory. The focus of this thesis is two NP-hard problems related to these concepts: the Maximum Planar Subgraph problem—given a graph G, find a planar subgraph of G with the maximum number of edges—and the k-Edge-Connected Spanning Subgraph problem—given a positive integer k and a k-edge-connected graph G, find a k-edge-connected spanning subgraph of G with the minimum number of edges.

No previous approximation algorithm for the Maximum Planar Subgraph problem was known to achieve a performance ratio larger than 1/3, which is achieved simply by producing a spanning tree. In this thesis, the first efficient approximation algorithm for the problem with higher performance ratio, namely 4/9, is presented. The new algorithm is also applied to the Maximum Outerplanar Subgraph problem—given a graph G, find an outerplanar subgraph of G with the maximum number of edges—which is also NP-hard. The performance ratio of the algorithm is proved to be exactly 2/3 for this problem, which improves on the 1/2 achieved by producing a spanning tree. The Maximum Planar Subgraph problem and the Nonplanar Deletion problem—given $G = (V, E)$, produce a smallest subset L of E such that the graph $(V, E - L)$ is planar—are shown to be Max SNP-hard.

For the Minimum k-Edge-Connected Spanning Subgraph problem, a ratio of two was known for a long time. Only in 1995, Khuller and Raghavachari presented the first algorithm for the problem with a performance ratio smaller than a constant smaller than two, for all k. They proved an upper bound of 1.85 for the performance ratio of their algorithm. This was the best known for the problem, until recently, when Cheriyan and Thurimella

came up with a new algorithm with a performance ratio of $1 + 2/(k + 1)$. The work of Cheriyan and Thurimella was done concurrently with the work presented in this thesis on the Minimum k-Edge-Connected Spanning Subgraph problem. This thesis presents an improvement on the upper bound for the performance ratio of Khuller and Raghavachari's algorithm from 1.85 to 1.75. Furthermore, the concept of a tree-carving, introduced by Khuller and Vishkin, is generalized and used to prove that the performance ratio of Khuller and Raghavachari's algorithm is less than 1.7 for large enough k. Last, the Minimum 2-Edge-Connected Spanning Subgraph problem is proved to be Max SNP-hard.

Chapter 1

Introduction

Graph theory has applications in many areas: it can be used to model a vast class of practical problems. This simple mathematical object called a graph—a set of points and links between these points—can represent a computer or railroad network, the connections of a circuit, compatibility between pairs of elements, and many other real world situations.

This versatility justifies the amount of work developed toward efficiently solving many graph-theoretic problems. A formalization of the notion of efficiency seemed necessary. In a theoretical context, a solution for a problem is nothing but an algorithm that computes a solution for any instance of the problem. Roughly speaking, an algorithm for a problem is *efficient* if it takes time polynomial in the size of the instance to compute a corresponding solution.

As expected, several of the many interesting problems in graph theory seem to be really hard to solve. For many years, no efficient algorithm was found for several important problems, leading people to believe that for such problems there might not be any. Cook [13] introduced the concept of NP-complete problems: a class of problems not known to have an efficient solution, and equivalent, in the sense that if one in the class does have an efficient solution, all the others also do.

The proof that several well-studied problems, including several graph-theoretic problems, are in this class discouraged the search for efficient algorithms for these problems

[40, 28]. But we are still faced with these problems in real life, which means we cannot simply give up solving them.

Among the practical and hard problems modeled by graph theory, many are optimization problems: given a set of feasible solutions, we are looking for the best one according to some criterion. If finding a best solution is hard, it is natural to ask if an approximate solution would be enough, and if so if it is easier to find.

This natural relaxation of the optimization problems—finding approximate solutions instead of optimal ones—gave rise to the study of approximation algorithms [27, 37]. An efficient α-approximation algorithm is one that computes a solution whose value is within a factor of α of the optimum value ($\alpha \geq 1$ for minimization problems and $\alpha \leq 1$ for maximization problems).

For the problems where exact efficient algorithms seem hard to find, like NP-complete problems, a reasonable goal is to look for efficient α-approximation algorithms. The closer to one α is, the better. A polynomial-time approximation scheme (PAS) for a problem is a family of efficient α-approximation algorithms with α approaching one.

Unfortunately, once more it has been proved that there are problems that are hard even to approximate in the sense that finding a PAS seems hard. The concept of Max SNP-hardness was introduced as an extension of the concept of NP-completeness for optimization problems [51]. The class of Max SNP-hard problems is a class of problems no one of which is known to have a PAS, and equivalent to each other: if one in the class does, all the others do.

The focus of this thesis are problems related to two important concepts in graph theory: planarity and connectivity.

In order to formally define these concepts, we will introduce some notation. We summarize the basic definitions and well-known results in graph theory and complexity theory in the next section. After that, in Section 1.2, we state the two problems studied in this thesis, and present a discussion of them and other related problems. Section 1.3 describes some applications.

1.1 Background

This section contains basic definitions and some well-known results in graph theory and complexity theory.

A *graph* G is an ordered pair (V, E) where V is a finite set and E is a finite set of unordered pairs of distinct elements of V. Elements of V are called *vertices of* G, while elements of E are called *edges of* G. Sometimes the vertex set of G is denoted by $V(G)$, and its edge set by $E(G)$.

An edge $e = \{u, v\}$ is simply denoted by uv or vu. The vertices u and v are called *endpoints of* e, and are said to be *neighbors* or *adjacent*. An edge is said to be *incident to a vertex* w if w is one of its endpoints.

There is a natural way of graphically representing a graph: each vertex can be represented by a point, and each edge by a curve joining the two points which represent its two endpoints. (See Figure 1.)

Consider a graph $G = (V, E)$. A *subgraph of* G is a graph $G' = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$. Such a subgraph is said to be a *spanning subgraph of* G if $V' = V$. Let S be a set of vertices of G. The *subgraph of* G *induced by* S is the graph (S, E'), where $e \in E'$ if and only if $e \in E$ and both of its endpoints are in S.

A *path in* G is a sequence $v_0, v_1, \ldots, v_p$ of distinct vertices of G such that $v_i v_{i+1} \in E$ for all i, $0 \leq i < p$. Edges $v_i v_{i+1}$ are said to be the edges of the path. The *length of the path* is the number of edges in the path, that is, p. Vertices v_0 and v_p are the *endpoints of the path*. The other vertices are the *internal vertices of the path*. Such a path is also said to be

$$G = (V, E)$$

$$V = \{t, u, v, w, x, y, z\}$$

$$E = \{tu, tv, uv, uy, uz, vy, yz, vw, wx, xv\}$$

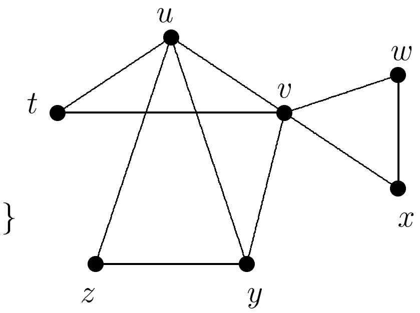

Figure 1: A graph and a graphic representation of it.

a $v_0 - v_p$ *path.*

Consider a sequence $v_0, v_1, \ldots, v_p$ of vertices of G such that $v_0 = v_p$, $v_0 v_1 \in E$, and $v_1, v_2, \ldots, v_p$ is a $v_1 - v_p$ path in G. The set C consisting of all the edges $v_i v_{i+1}$, $0 \leq i < p$, is a *cycle in* G. The *length of the cycle* is the number of edges in the cycle, that is, p. Sometimes the cycle is represented by the corresponding sequence $v_0, v_1, \ldots, v_p$. Note that several different sequences might represent the same cycle (any rotation of the original sequence is said to be the same cycle).

Two paths in G are *disjoint* if there is no vertex of G in both of them. Two paths are *internally disjoint* if any vertex of G that is in both paths is an endpoint of both of them. Two paths in G are *edge-disjoint* if there is no edge of G in both of them.

A *multigraph* is similar to a graph where the edge set is a *multiset* (repetitions are allowed) of unordered pairs of vertices. All the definitions given for graphs apply also to multigraphs.

A *directed graph* or *digraph* is the same as a graph, except that the edge set is a finite set of *ordered* pairs of distinct vertices. Sometimes the edges of a directed graph are called *arcs* and its vertices, *nodes*.

If $e = (u, v)$ is an edge in a digraph $D = (V, E)$, then e is said to be an edge from u to v. Also, e is said to be *leaving* u and *entering* v. All the definitions given for graphs apply to digraphs. Note however that the orientation of the edges is being considered. For example, if $v_0, v_1, \ldots, v_p$ is a path in D, then $v_i v_{i+1}$ is an edge in D, but not necessarily $v_{i+1} v_i$ (as opposed to the undirected case, where $v_i v_{i+1}$ and $v_{i+1} v_i$ denote the same edge).

The *underlying graph of a digraph* $D = (V, E)$ is the graph (V, E') where $E' = \{e = uv : (u, v) \in E\}$.

In a graph G, two vertices u and v are *connected* if there is a $u - v$ path in G. The graph G is *connected* if any pair of vertices u and v of G are connected. A *connected component of* G, sometimes simply called a *component of* G, is a maximal connected subgraph of G

In a digraph, two vertices u and v in a digraph are *strongly connected* if there are a $u \to v$ path and a $v \to u$ path in G. Two vertices u and v in a digraph are *weakly connected* if there is a $u - v$ path in the corresponding underlying graph. The concept of connected component for the digraph G is defined using the underlying graph instead of the digraph itself. That is, a *connected component of the digraph* G is a directed subgraph of G induced by the vertex set of a connected component of the underlying graph of G.

Let $G = (V, E)$ be a graph, and u be a vertex of G. The *graph obtained from* G *by removing* u, denoted $G - u$, is the graph $(V - \{u\}, E')$, where $E' = E - \{e \in E : e = uv$ for some $v \in V\}$. Let S be a set of *edges* of G. We denote by $G - S$ the graph $(V, E - S)$.

A graph (digraph) G is k*-vertex-connected*, or for short k*-connected*, if the graph obtained from G after the removal of any $k - 1$ vertices is still (strongly) connected. We say G is k*-edge-connected* if, for any set S of $k - 1$ edges, the graph $G - S$ is still (strongly) connected. A *2-connected component of a graph* G is a maximal 2-connected subgraph of G.

A *cut in* G is denoted by $(X, \overline{X})$, where $X \neq V$ is a nonempty subset of V, and $\overline{X}$ stands for $V - X$. A cut $(X, \overline{X})$ in G consists of the set of all edges of G which have one endpoint in X and the other in $\overline{X}$. A cut in G is *minimum* if, among all cuts in G, it contains the minimum number of edges.

A *Hamiltonian path in* G is a path in G containing all the vertices of G. A *Hamiltonian cycle in* G is a cycle in G containing all the vertices of G. If G has a Hamiltonian cycle, it is said to be *Hamiltonian*.

A *matching in* G is a set M of edges such that no two edges in M share an endpoint.

An *independent set in* G is a set of vertices such that no two in the set are adjacent in G.

A vertex set S is a *vertex cover in* G if every edge of G has at least one endpoint in S.

The *degree of a vertex* v *in* G is the number of edges of G incident to v. The *maximum degree of* G is the maximum degree among all of its vertices. A class of *bounded degree graphs* consists of all graphs with the following property. There is a constant d such that all the graphs in the class have d as an upper bound on their maximum degree. The *indegree of a vertex* v *in a digraph* D is the number of edges of D entering v. Similarly, the *outdegree of* v *in* D is the number of edges of D leaving v.

A *forest* is a graph with no cycles. Vertices of degree one in a forest are called *leaves.* A *tree* is a connected forest, that is, a connected graph with no cycles. If a forest or a tree is a subgraph of a graph G, then we say it is a *forest* or a *tree of* G. A *rooted tree* is a connected digraph with a distinguished vertex called *root*, and satisfying the following properties. The indegree of the root is zero, that is, no edge enters the root. The indegree of any vertex other than the root is one. Given an edge xy in a rooted tree, y is said to be a *child of* x, while x is the *parent of* y. A *rooted forest* is a digraph whose components are rooted trees. Vertices of outdegree zero in a rooted forest are called *leaves.* One can prove that the underlying graph of a rooted tree is in fact a tree.

1.1.1 Depth-First Search

Next, a special procedure for traversing a graph is described. A directed graph is used to facilitate the proof that the procedure produces a spanning forest of the input graph.

Depth-first search (DFS) is a way of searching a graph $G = (V, E)$ using a stack (LIFO) discipline to choose the next edge to explore. It produces a numbering of the vertices, called a *DFS numbering*, and a particular spanning tree, called a *DFS tree*, of each connected component of G. These trees together form a *DFS forest of* G. In fact, a rooted forest D is built, with vertex set V. The underlying graph of D is the desired DFS forest of G.

In the beginning, no vertex has a DFS number. Set a DFS counter c to zero. There is a phase for each component of G. A phase proceeds as follows.

Choose an arbitrary vertex u in V which has no DFS number. Assign it DFS number c and increase c by one. This vertex will be the root of one of the (rooted) trees. Push all the edges $uv \in E$ onto the stack. Now repeat the following activity until the stack becomes empty. Let xy be the top element of the stack. This is the next edge to explore. The vertex x has a DFS number already (this is an invariant of the loop). If y has no DFS number, then assign it DFS number c, increase c by one, pop the edge xy off the stack, push all edges $yz \in E$ onto the stack, make xy an edge of D and call it a *tree edge*. Otherwise, if y has a DFS number already, just pop xy off the stack.

The tree edges form a rooted tree with u as root, and whose underlying graph is a tree of the connected component containing u. This is true because tree edges only go from lower numbered vertices to higher numbered vertices and there is at most one edge entering each vertex.

If after the stack becomes empty, there still are vertices with no DFS number, start a new phase. Edges of G which are not tree edges are called *back edges*. (See Figure 2.)

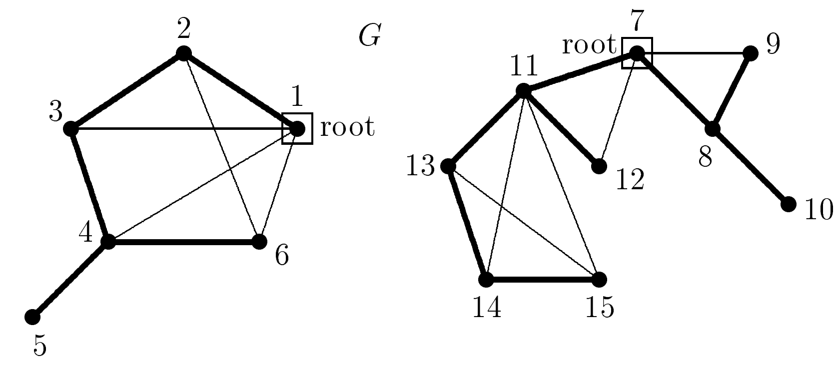

Figure 2: A graph G, a DFS forest in G and its corresponding DFS numbering.

1.1.2 Planar Graphs

A graph is *planar* if there is a graphic representation for it in the plane where curves corresponding to edges intersect only at their endpoints. A graphic representation in the plane of a planar graph G, formally defined next, is called a *plane embedding of G*. (See Figure 3.) Formally, a graph $G = (V, E)$ is *planar* if there is an injective function $\phi : V \to I\!R^2$ and, for each edge $e = uv$, a continuous 1-1 function $\psi_e : [0, 1] \to I\!R^2$ such that $\psi_e(0) = \phi(u)$, $\psi_e(1) = \phi(v)$ and, for any pair of edges e and f, $\psi_e(x) \neq \psi_f(y)$ for all x and y such that $0 < x < y < 1$. The functions ϕ and ψ_e, for all e, represent a plane embedding of G.

A planar graph with a fixed plane embedding is called a *plane graph*. Sometimes G will be used to refer to the fixed plane embedding of a plane graph G.

The following theorem [2, Cor. 9.5.2] gives an upper bound on the number of edges of a planar graph.

Theorem 1.1.1 *If $G = (V, E)$ is a planar graph with $|V| \geq 3$, then $|E| \leq 3|V| - 6$.*

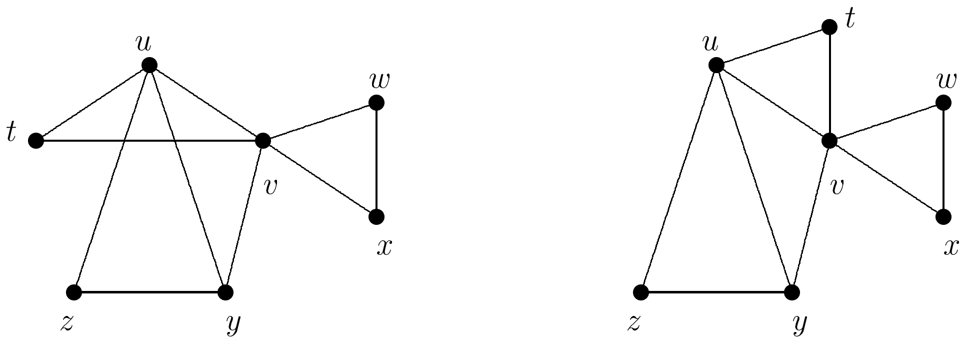

Figure 3: Two graphic representations of the same graph. Only the second one is a plane embedding.

A plane graph G partitions the rest of the plane into a number of connected regions. The closure of each of these regions is a *face of* G. Any plane embedding of G has exactly one unbounded face, called the *exterior face.* The other faces are called the *interior faces.*

A planar graph that has a plane embedding where all of its vertices are on the boundary of the exterior face is called an *outerplanar graph.*

A *walk in a graph* $G = (V, E)$ is a sequence of vertices $v_0, v_1, \ldots, v_p$ of G such that $v_i v_{i+1} \in E$ for all i, $0 \leq i < p$. Note that a path is a special kind of walk: one that does not repeat vertices. Edges $v_i v_{i+1}$ are said to be the edges of the walk. The *length of the walk* is p. A walk is *closed* if $v_0 = v_p$.

For connected plane graphs, the boundary of a face can be regarded as a closed walk. If all the faces of a connected plane graph have length three, then the graph is *triangulated.* A face whose length is three is called a *triangular face.*

Let $\Phi(G)$ denote the set of faces of a plane graph G. The following invariant, known as

Euler's formula [18], holds for any connected plane graph G,

$$|V(G)| - |E(G)| + |\Phi(G)| = 2.$$

From Theorem 1.1.1 and Euler's formula, one can conclude that for any plane graph G,

$$|\Phi(G)| \leq 2|V(G)| - 4. \tag{1}$$

Both Theorem 1.1.1 and Equation (1) are tight, in the sense that there are plane graphs G with at least three vertices, having $|E(G)| = 3|V(G)| - 6$ edges and $|\Phi(G)| = 2|V(G)| - 4$ faces. In fact, any triangulated plane graph on $n \geq 3$ vertices has $3n - 6$ edges and $2n - 4$ faces.

In the context of planarity, there are two very important graphs: K_5 and $K_{3,3}$. K_5 is the graph with five vertices and all possible edges, that is, $K_5 = (V, E)$ where $|V| = 5$ and $uv \in E$ for all $u, v \in V, u \neq v$. $K_{3,3}$ has two disjoint groups of three vertices and all edges with exactly one endpoint in one of the groups, that is, $K_{3,3} = (A \cup B, E)$ where $|A| = |B| = 3, A \cap B = \emptyset$, and $uv \in E$ if and only if $u \in A$ and $v \in B$, or $u \in B$ and $v \in A$. (See Figure 4.) It can be verified that K_5 and $K_{3,3}$ are nonplanar graphs, that is, for any graphic representation of K_5 or $K_{3,3}$ in the plane, there are edges crossing (i.e., edges whose curves intersect at some point other than their endpoints).

Consider a graph $G = (V, E)$ and an edge $e = uv \in E$. A new graph G' can be obtained from G by *subdividing edge* e: edge e is removed and replaced by a $u - v$ path of length two with a new internal vertex. That is, $G' = (V \cup \{w\}, (E \setminus \{e\}) \cup \{e_1, e_2\})$ where $w \notin V$, $e_1 = uw$ and $e_2 = wv$.

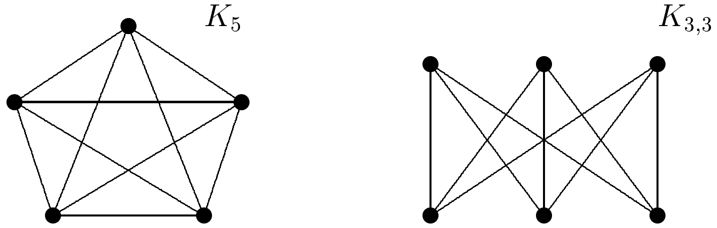

Figure 4: Graphs K_5 and $K_{3,3}$.

A *subdivision of* G is a graph obtained from G by a sequence of edge subdivisions. Note that, if a graph G is nonplanar, then any subdivision of G is also nonplanar. We say *a graph* H *contains a subdivision of* G if there is a subdivision of G which is a subgraph of H.

The next theorem, proved by Kuratowski in 1930 [44], shows why K_5 and $K_{3,3}$ play an important rule in the context of planarity.

Theorem 1.1.2 (Kuratowski) *A graph is planar if and only if it contains no subdivision of* K_5 *and* $K_{3,3}$.

Roughly speaking, K_5 and $K_{3,3}$ are the minimal obstructions that make a graph nonplanar.

1.1.3 Complexity Theory

Cook [13] in 1971 introduced the basic concepts of complexity theory. A summary of these concepts is presented here.

The *time complexity* of an algorithm is the relation between its running time and the length of its input.

It is reasonable to assume that the running time of an algorithm on a certain input is proportional to the number of basic operations (comparisons, additions, etc) executed by the algorithm while running on this input.

Consider an algorithm A for a problem B. The input for A is a representation of an instance I of B. Let l_I be the length of this representation, that is, its number of bits.

A is a *polynomial-time algorithm* if there is a polynomial $p(x)$ such that the running time of A is bounded by $p(l_I)$ for all possible instances I of B. It is said also that the running time of A is polynomial, or that A runs in polynomial time.

For graph algorithms, l_I is polynomially related to the number n of vertices and the number m of edges in the graph. So l_I can be substituted by $q(n,m)$ in the definition of polynomiality, where q is a polynomial. Thus, the running time of a graph algorithm A is polynomial if it is bounded by $p(n,m)$, where p is a polynomial. It is said that the running time of A is $O(p(n,m))$ if there is a constant c and a polynomial p such that the running time of A is bounded by $c \cdot p(n,m)$.

As an example, consider the procedure presented in Section 1.1.1 to construct a DFS forest in a graph G with n vertices and m edges. One can show that the running time of this procedure is $O(n+m)$. In this case, we say it runs in *linear time.*

A *decision problem* is a problem whose answer, for any instance, is either "yes" or "no." Cook defined two important classes of decision problems: *P* and *NP*. The class P consists of all decision problems solvable by a polynomial-time algorithm.

The class NP consists of all decision problems B that satisfy the following properties. Any instance I of B whose answer is "yes" can be associated to a *certificate* $C(I)$. Moreover, there is an algorithm A independent of I that answers "yes" if and only if its input is I followed by $C(I)$, for some instance I of B whose answer is "yes." That is, for any instance I whose answer is "no", there is no $C(I)$ such that A, given I followed by $C(I)$ as input, produces "yes" as answer. Besides, the running time of A is polynomial in l_I. In other words, A verifies, using $C(I)$, that I is a "yes" instance of B in time polynomial in the length of I.

Cook introduced the concept of a *polynomial-time reduction*, simply called *reduction*, from a decision problem to another. A *reduction from a decision problem A to a decision problem B* is a polynomial-time algorithm f that has as input instances I of A and as output instances $f(I)$ of B such that I is a "yes" instance of A if and only if $f(I)$ is a "yes" instance of B.

A problem B is said to be *NP-hard* if there is a reduction from all problems in NP to B. A problem B is said to be *NP-complete* if it is in NP and it is NP-hard. The first problem proved to be NP-complete is the so-called *Satisfiability problem* (see below), through an abstract reduction presented by Cook.

> **Satisfiability problem:** given a boolean formula in conjunctive normal form, determine whether it is satisfiable or not, that is, whether there is a truth assignment to its variables that makes the formula *TRUE*.

Karp [40] proved the NP-completeness of several well-known combinatorial problems. After that, many other problems were shown to be NP-complete. The famous book by Garey and Johnson [28], which is a basic reference on complexity theory, contains a long list of NP-complete problems.

Here are two other examples of NP-complete problems. They were chosen since a variation of each of them is used in the reductions presented later in this thesis.

> **Hamiltonian Cycle problem:** given a graph G, determine whether G is Hamiltonian or not, that is, whether there exists a Hamiltonian cycle in G or not.

Vertex Cover problem: given a graph G and an integer k, determine if there is a vertex cover in G with at most k vertices.

The first of the two problems is also NP-complete if we change cycle to path.

The most famous question in complexity theory is whether P=NP or not. An affirmative answer implies that hundreds of well-known hard problems, studied for decades (e.g., all the NP-complete problems) can be solved in polynomial time, which seems unlikely.

Papadimitriou and Yannakakis [51] defined a natural variant of NP for optimization problems: the complexity class Max SNP. Their definition is inspired by Fagin's characterization of the class NP [19], and required a generalization of the concept of reductions.

An *L-reduction* is a special kind of reduction that preserves approximability. Let A and B be two optimization problems. Problem A *L-reduces to* B if there are two polynomial-time algorithms f and g, and positive constants α and β, such that for each instance I of A,

1) Algorithm f produces an instance $I' = f(I)$ of B, such that the optima of I and I', of values denoted $Opt_A(I)$ and $Opt_B(I')$ respectively, satisfy $Opt_B(I') \leq \alpha \cdot Opt_A(I)$, and

2) Given any feasible solution of I' with cost c', algorithm g produces a solution of I with cost c such that $|c - Opt_A(I)| \leq \beta \cdot |c' - Opt_B(I')|$.

Fagin proved that all graph-theoretic problems in NP can be expressed in existential second-order logic. The class SNP, from *strict* NP, consists of all problems in NP expressible as

$$\exists S \forall x_1 \forall x_2 \ldots \forall x_k \Phi(S, G, x_1, x_2, \ldots, x_k), \tag{2}$$

where Φ is a quantifier-free first-order expression involving variables x_i and the structures G (the input) and S.

Equation (2) asks for a relation S such that all possible k-tuples $(x_1, x_2, \ldots, x_k)$ satisfy Φ. To define the class Max SNP, this requirement is relaxed: instead we seek a relation S such that Φ holds for as many k-tuples $(x_1, x_2, \ldots, x_k)$ as possible. Moreover, we generalize the input G to a collection of relations $G_1, G_2, \ldots, G_m$.

Formally, consider the following class of optimization problems: problem A is in this class if it can be defined as

$$\max_S |\{(x_1, x_2, \ldots, x_k) \in V^k : \Phi(S, G_1, G_2, \ldots, G_m, x_1, x_2, \ldots, x_k)\}|, \tag{3}$$

the input to problem A being the set of relations $G_1, G_2, \ldots, G_m$ over the finite universe V. The class Max SNP is the class of all optimization problems that are L-reducible to a problem in the class above. For any problem in Max SNP, there is an α-approximation algorithm, for some constant α.

A problem B is *Max SNP-hard* if all problems in Max SNP can be L-reduced to B. Papadimitriou and Yannakakis have shown that Max SNP contains several famous optimization problems. They proved also that, among others, the weighted optimization version of the Hamiltonian Cycle problem is Max SNP-hard. This problem is known as *Traveling Salesman problem* (below) and will be used in one of the reductions presented in this thesis.

> **Traveling Salesman problem:** given a graph G, a nonnegative cost $c(e)$ for each edge e of G, find a Hamiltonian cycle (called a *traveling salesman tour*) with the minimum cost.

Consider an optimization problem B, an approximation algorithm A for B and an instance I of B. Denote by $Opt(I)$ the optimum value of B for instance I, and by $A(I)$ the value of the output produced by algorithm A when I is the input. If B is a maximization problem, the *performance* or *approximation ratio of* A is the infimum, over all instances I, of $A(I)/Opt(I)$ (if $Opt(I) > 0$, and 1 otherwise). For minimization problems, we use supremum instead of infimum. In the literature, authors sometimes define their performance ratio as the reciprocal of ours.

Note that, according to our definition, if B is a maximization problem, $A(I) \leq Opt(I)$ for all I, which implies that the performance ratio is at most one. On the other hand, if B is a minimization problem, $A(I) \geq Opt(I)$ for all I, which implies that the performance ratio is at least one.

Arora et al. [1] proved that, for any maximization problem B which is Max SNP-hard, there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm for B with performance ratio at least $1 - \epsilon$ implies that $P = NP$. Analogously, for any minimization problem B which is Max SNP-hard, there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm for B with performance ratio at most $1 + \epsilon$ implies that $P = NP$. In other words, for a Max SNP-hard problem, there is no PAS unless P=NP.

1.2 The Problems

This thesis concentrates on the following two important graph-theoretic problems:

> **Maximum Planar Subgraph problem:** given a graph G, find a planar subgraph of G with the maximum number of edges.
>
> **Minimum k-Edge-Connected Spanning Subgraph problem:** given a positive integer k and a k-edge-connected graph G, find a k-edge-connected spanning subgraph of G with the minimum number of edges.

The Maximum Planar Subgraph problem is known to be NP-hard [45]. As shown in the next section, it has applications in facility layout and graph drawing [23, 54], which justify the existence of several approximation algorithms for it. None of the previous approximation algorithms was known to achieve a performance ratio larger than $1/3$. In this thesis, we present the first efficient approximation algorithm for the Maximum Planar Subgraph problem with higher performance ratio: $4/9$ instead of $1/3$ [5, 6]. We also apply our algorithm to another related problem: the Maximum Outerplanar Subgraph problem. Last, we show that the Maximum Planar Subgraph problem is Max SNP-hard.

A natural extension of the Maximum Planar Subgraph problem is its generalization to other surfaces. In [4], we apply the ideas of our new algorithm to obtain an approximation algorithm for the following problem. Given a graph G and a positive integer g, find a subgraph of G of maximum size and embeddable in a surface of genus g, where *size* is the number of edges.

Another obvious extension of the Maximum Planar Subgraph problem is its weighted version, that is, given a graph G and a weight $w(e)$ associated with each edge e of G, find a heaviest planar subgraph of G. This problem has applications in circuit layout and also in facility layout [16, 23]. The situation for this weighted version is very similar to its unweighted counterpart. Until very recently, no approximation algorithm was known to achieve a performance ratio larger than $1/3$. Besides, $1/3$ can be achieved by any algorithm that produces a maximum weight spanning tree. In [7], we present a new algorithm for this problem with a performance ratio of at least $1/3 + 1/72$.

The gap between the best known performance ratio for the Maximum Planar Subgraph problem and its weighted version is somehow intriguing. For most approximation algorithms proposed recently, the performance ratios for the weighted cases are the same as those for the unweighted cases [14, 53, 32, 20, 34, 35].

In the first part of this thesis, we focus on the Maximum Planar Subgraph problem. The new algorithm for the problem appears in Chapter 2. The discussion on the Maximum Outerplanar Subgraph problem is done in Chapter 3. Chapter 4 talks about the complexity of the Maximum Planar Subgraph problem and its complementary version, the Nonplanar Deletion problem.

The second family of problems are connectivity-related problems. This kind of problem arises naturally in the areas of network reliability and network design [23]. Several approximation algorithms have been developed for the problem of finding subgraphs satisfying certain connectivity constraints [8, 25, 29, 30, 31, 39, 43, 50]. We concentrate on the versions of the problem with uniform constraints.

There are several possible variations of the Minimum k-Edge-Connected Spanning Subgraph problem. The given graph might be directed. Vertex-connectivity might be the goal, instead of edge-connectivity. These choices give rise to four different NP-hard versions of the problem.

A performance ratio of two for all the four versions was known for a long time to be achievable. It is derived first from a trivial lower bound on the size of the optimum called the *degree lower bound*. In the undirected case, any k-edge-connected or k-vertex-connected spanning subgraph must have at least k edges incident to each vertex. This gives a total of at least $kn/2$ edges, where n is the number of vertices in the input graph. Similarly, in the directed case, any k-edge-connected or k-vertex-connected spanning subgraph must have at least k edges leaving each vertex. This implies a lower bound of kn on the size of the optimum in the directed case. What about upper bounds on the size of the optimum?

In the undirected case, every minimal k-edge-connected or k-vertex-connected spanning subgraph has at most kn edges [47, 48]. For the directed version, there are at most $2kn$ edges in any minimal k-edge-connected or k-vertex-connected spanning subgraph [17, 49]. Besides, it is easy to obtain, in polynomial time, a minimal k-edge-connected or k-vertex-connected spanning subgraph of a given graph (if one exists). For each edge in the graph, check if its removal still results in a k-edge-connected or k-vertex-connected spanning subgraph. If so, remove it. If not, leave it. The resulting subgraph is obviously minimal and satisfies the connectivity requirement. A performance ratio of two follows.

In fact, for the undirected case, there are new algorithms for finding k-edge-connected or k-vertex-connected spanning subgraph with at most kn edges [55, 50, 24, 8].

Consider the case of $k = 2$. Khuller and Vishkin [43] described an algorithm for the undirected version, whose performance ratio is 1.5 for the 2-edge-connected case and 1.66 for the 2-vertex-connected case. The latter performance ratio was improved to 1.5 by Garg et al. [29] using nontrivial lower bounds on the size of the optimum.

Karger [39] presented an elegant simple randomized algorithm with performance ratio $1+O(\sqrt{(\log n)/k})$ for the Minimum k-Edge-Connected Spanning Subgraph problem. This is smaller than two only when $k \gg \log n$. As observed in [41], by combining the biconnectivity algorithm in [43], and the sparse certificate algorithm in [8], one can easily obtain a ratio of $2 - 1/k$.

The first algorithm for the Minimum k-Edge-Connected Spanning Subgraph problem with a performance ratio smaller than a constant smaller than two, for all k, was presented by Khuller and Raghavachari [41] in 1995. They proved an upper bound of 1.85 for the performance ratio of their algorithm. In this thesis, we improve their analysis, proving that the performance ratio of their algorithm is smaller than 1.7 for large enough k, and that it is at most 1.75 for all k. Last, we show that the Minimum 2-Edge-Connected Spanning Subgraph problem is Max SNP-hard [21, 22].

Very recently, Cheriyan and Thurimella [9] came up with a new algorithm for the vertex-connectivity version of the problem. Their algorithm achieves a performance ratio of $1+1/k$.

They also present a variant of their algorithm for the edge-connectivity case, and prove an approximation ratio of $1+2/(k+1)$ in the undirected case (the Minimum k-Edge-Connected Spanning Subgraph problem), and of $1+4/\sqrt{k}$ in the directed case.

The discussion of the Minimum k-Edge-Connected Spanning Subgraph problem is done in Chapters 4, 5 and 6, the proof of the Max SNP-hardness of the problem being in Chapter 6. Chapter 4 contains a review of Khuller and Raghavachari's algorithm and the proof of the 1.75 upper bound on its performance ratio. In Chapter 5, the theory necessary for the proof that its performance ratio is at most 1.7, for large k, and the proof itself are presented.

The next section describes some real world problems and shows how they can be modeled as instances of either the Maximum Planar Subgraph problem or the Minimum k-Edge-Connected Spanning Subgraph problem.

1.3 Applications

A visualization of a graph can be used to give some insight into a problem, and is useful for several real-life situations. The problem of *graph drawing* [54] consists of attempting to draw a given graph in a way as nice as possible. For a planar graph G, it seems natural to think of a plane embedding of G as a nice drawing. In fact, it is always possible to draw a planar graph where the edges are straight lines. For a nonplanar graph G, a planar subgraph of G can be used as a start for obtaining a nice drawing of G. The more edges in the subgraph, the better. So algorithms for the Maximum Planar Subgraph problem have applications in graph drawing.

Consider the problem of deciding the layout of a set of facilities [23]. It is reasonable to assume that trips are saved if facilities are located adjacently. In this sense, a good layout is one that maximizes the number of adjacent pairs of facilities.

We can model this problem as an instance of the Maximum Planar Subgraph problem:

the set of vertices of the graph is the set of facilities, and there is an edge between two vertices if the corresponding facilities can be adjacent.

Now consider a communication network, where centers are connected by links. A possible measure of the reliability of this network is the minimum number of links whose breakdown disconnects a pair of centers [23]. When planning the construction of a network, one can have in mind that the resulting network must have a reliability of at least k, for some k. Obviously, the larger the number of links, the larger the cost of building the network. So, given a construction plan consisting of centers and possible links between centers, it would be nice to know what is the minimum number of links necessary to have a network with reliability at least k.

This problem can be modeled as an instance of the Minimum k-Edge-Connected Spanning Subgraph problem. The vertices of the graph represent the centers, while the links are represented by the edges, and the desired reliability is the parameter k.

If one measures the reliability in terms of the minimum number of *centers* whose breakdown disconnects the network, the problem can be modeled as the vertex-connectivity version of the Minimum k-Edge-Connected Spanning Subgraph problem.

Chapter 2

Finding Large Planar Subgraphs

The Maximum Planar Subgraph is the following problem: given a graph G, find a planar subgraph of G with the maximum number of edges. No previous approximation algorithm for this NP-Complete problem was known to achieve a performance ratio larger than $1/3$, which is achieved simply by producing a spanning tree of G. We present the first approximation algorithm for the Maximum Planar Subgraph problem with higher performance ratio: $4/9$ instead of $1/3$.

2.1 History of the Problem

The Maximum Planar Subgraph problem has applications in facility layout and graph drawing [23, 54], as discussed in Section 1.3.

In 1977, Liu and Geldmacher proved that Maximum Planar Subgraph is NP-Complete [45]. As a consequence, no polynomial-time algorithm for the problem exists, unless P=NP. The search for efficient approximation algorithms becomes a natural goal.

For a graph G, let $Opt(G)$ be the maximum size of a planar subgraph of G. Given an algorithm A that takes graphs G as input and outputs subgraphs of G, define $A(G)$ to be the size of the planar graph A produces when G is the input. According to the definition given in Section 1.1.3, the performance ratio of A, here denoted as $r(A)$, is the infimum, over all graphs G, of $A(G)/Opt(G)$ (if $Opt(G) > 0$, and 1 otherwise).

Numerous approximation algorithms for the Maximum Planar Subgraph problem appear in the literature, the simplest ones being the *Spanning Tree algorithm* and the *Maximal Planar Subgraph algorithm*.

The Spanning Tree algorithm output any spanning tree of G, assuming G is connected. It is known to have performance ratio 1/3 (see below).

The Maximal Planar Subgraph algorithm outputs any planar spanning subgraph to which the addition of any new edge would violate planarity. Dyer, Foulds and Frieze [16] proved that the Maximal Planar Subgraph algorithm has performance ratio 1/3.

Cimikowski [12] proved that a path embedding heuristic of Chiba, Nishioka and Shirakawa [10] and an edge embedding heuristic of Cai, Han and Tarjan [3] have performance ratios not exceeding 1/3. In the same paper, Cimikowski studied two other polynomial-time heuristics: the "vertex-addition heuristic" and the "cycle-packing heuristic." The performance ratio of the former, to the author's knowledge, is not known, whereas for the cycle-packing algorithm, it is zero. Dyer, Foulds and Frieze [16] studied two other algorithms and proved that each has performance ratio at most 2/9. Also see [38].

In short, to the author's knowledge, no previously proposed algorithm was known to have a performance ratio exceeding 1/3. What makes the problem more tantalizing is that achieving a performance ratio of 1/3 is trivial. In fact, the Spanning Tree algorithm has performance ratio 1/3, since every spanning tree of a connected graph on n vertices has $n-1$ edges and every planar graph on n vertices has at most $3n-3 = 3(n-1)$ edges (see Theorem 1.1.1). Moreover, there are planar graphs on n vertices with $3n-6$ edges, for all $n \geq 3$. No previous algorithm could beat the bound achieved by a trivial algorithm.

In this chapter, we present two new approximation algorithms for Maximum Planar Subgraph. Each achieves a performance ratio exceeding 1/3. The higher performance ratio is $4/9 = 0.444...$ and is achieved by an algorithm which invokes an algorithm for the

Graphic Matroid Parity problem (defined later) as a subroutine and which runs in time $O(m^{3/2} n \log^6 n)$, where m is the number of edges in the graph and n is the number of vertices. A greedy variant still has performance ratio $7/18 = 0.3888...$, and runs in linear time on graphs of bounded degree.

2.2 The New Approximation Algorithms

Let us give some motivation for our algorithm. As we said, given a (connected) graph G, an algorithm which outputs a spanning tree of G achieves a performance ratio of $1/3$. A graph whose cycles all have length three, i.e., are triangles, is planar, as it cannot contain a subdivision of K_5 or $K_{3,3}$. Moreover, note that a connected spanning subgraph of G whose cycles are triangles, besides being planar, has one more edge per triangle than a spanning tree of G has.

Our better algorithm produces a subgraph of G whose cycles are triangles and, among these subgraphs, has the maximum number of edges. It can be implemented in time $O(m^{3/2} n \log^6 n)$, where m is the number of edges in G and n is the number of vertices in G, using a graphic matroid parity algorithm, as we will see later. We first present a greedy version of the algorithm.

2.2.1 A Greedy Version of the Algorithm

Algorithm A, presented below, is a greedy version of our algorithm. It has a performance ratio of 7/18=0.3888.... After presenting the algorithm and proving its performance ratio is 7/18, we will show it can be implemented in linear time for graphs with bounded degree. We begin with some definitions.

A *triangular cactus* is a graph whose cycles (if any) are triangles and such that all edges appear in some cycle. A *triangular cactus in a graph* G is a subgraph of G which is a

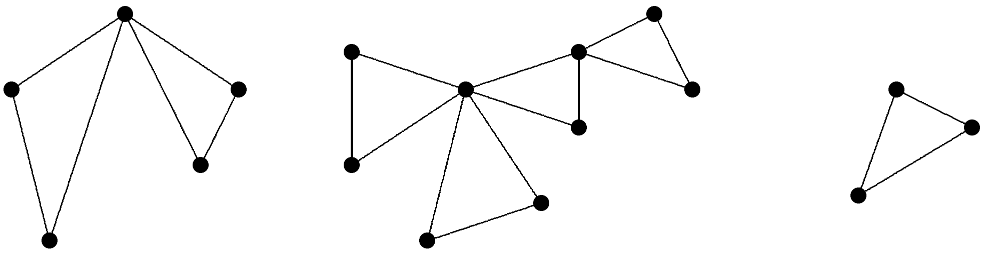

Figure 5: A triangular cactus.

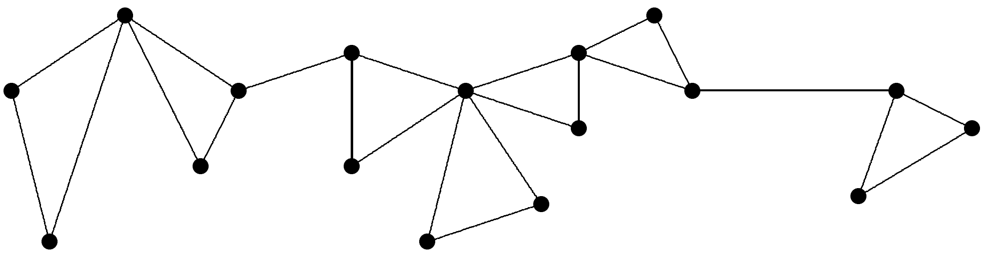

Figure 6: A triangular structure.

triangular cactus. (See Figure 5.)

A *triangular structure* is a graph whose cycles (if any) are triangles. A *triangular structure in a graph* G is a subgraph of G which is a triangular structure. Note that every triangular cactus is a triangular structure, but not *vice versa*. (See Figure 6.)

Algorithm A produces a triangular structure in the given graph G, and this will be our output for a planar subgraph of G. The algorithm consists of two phases. First, A greedily constructs a maximal triangular cactus S_1 in G. Second, A extends S_1 to a triangular structure S_2 in G by adding as many edges as possible to S_1 without forming any new cycles.

Given a graph $G = (V, E)$ and $E' \subseteq E$, we denote by $G[E']$ the *spanning* subgraph of G induced by E', that is, the graph (V, E'). (Note that this is not the usual definition of subgraph induced by an edge set, since we require the subgraph to be spanning.)

Algorithm A

Starting with $E_1 = \emptyset$, repeatedly (as long as possible) find a triangle T whose vertices are in different components of $G[E_1]$, and add the edges of T to E_1.
Let $S_1 := G[E_1]$.

Starting with $E_2 = E_1$, repeatedly (as long as possible) find an edge e in G whose endpoints are in different components of $G[E_2]$, and add e to E_2.
Let $S_2 := G[E_2]$.

Output S_2.

Note that the output S_2 is indeed a triangular structure in G. As we mentioned before, S_2 is planar since it does not contain cycles of length greater than three.

Theorem 2.2.1 *The performance ratio of Algorithm A is* $\frac{7}{18}$.

Proof. First let us show that the performance ratio is at least 7/18. Without loss of generality, we may assume G is connected, and has at least three vertices. Observe that the number of edges in S_2 is the number of edges in a spanning tree of G plus the number of triangles in S_1. So it suffices to count the number of triangles in S_1.

Let H be a plane embedding of a maximum planar spanning subgraph of G. Let $n \geq 3$ be the number of vertices in G, and $t \geq 0$ be such that $3n - 6 - t$ is the number of edges in H. We can think of t as the number of edges missing for H to be a triangulated plane graph. The number of triangular faces in H is at least $2n - 4 - 2t$. (This is a lower bound on the number of triangular faces of H since if H were triangulated, it would have $2n - 4$

triangular faces, and each missing edge can destroy at most two of these triangular faces.)

Let k be the number of components of S_1 each with at least one triangle, and let $p_1, p_2, \ldots, p_k$ be the number of triangles in each of these components. Let $p = \sum_{i=1}^{k} p_i$. We will prove that p, the number of triangles in S_1, is at least a constant fraction of $n - 2 - t$. Note that if a triangle cannot be added to S_1, it is because two of its vertices are in the same component of S_1. Hence one of its edges has its two endpoints in the same component of S_1. This means that at the end of the first phase, every triangle in G must have some two vertices in the same component of S_1. In other words, every triangular face in H must have some two vertices in the same component of S_1, and therefore one of its three edges must be in the subgraph of H induced by the vertices in a component of S_1. Thus we can associate with each triangular face F in H an edge e in F whose endpoints are in the same component of S_1. But any edge e in H lies in at most two triangular faces of H, so e could have been chosen by at most two triangular faces of H. It follows that the number of triangular faces in H is at most twice the number of edges in H whose endpoints are in the same component of S_1.

Let H' be the subgraph of H induced by the edges of H whose endpoints are in the same component of S_1. Note that $p_i \geq 1$, for all i, and that the number of vertices in the i^{th} component of S_1 is $2p_i + 1 \geq 3$. Since H' is planar, H' has at most $\sum_{i=1}^{k}(3(2p_i + 1) - 6) = 6p - 3k$ edges. By the observation at the end of the previous paragraph, $2(6p - 3k) \geq 2|E(H')| \geq$ (number of triangular faces in H) $\geq 2n - 4 - 2t$. From this, we have

$$p \geq \frac{n - 2 - t + 3k}{6} \geq \frac{n - 2 - t}{6}.$$

Therefore the number of triangles in S_1 is at least $\frac{n-2-t}{6}$, and the ratio between the number of edges in S_2 and the number of edges in H is at least

$$\frac{n-1+\frac{n-2-t}{6}}{3n-6-t} = \frac{7n-8-t}{18n-36-6t} \geq \frac{7}{18},$$

since $t \geq 0$. This completes the proof that the performance ratio of Algorithm A is at least 7/18.

Now, we will prove that the performance ratio is at most 7/18. Let S be any connected triangular cactus with $p > 0$ triangles. S has $2p+1 \geq 3$ vertices. Let S' be any triangulated plane supergraph of S on the same set of vertices (S' can be obtained from S by adding edges to S until it becomes triangulated). Since S' is triangulated, S' has $2(2p+1)-4 = 4p-2$ (triangular) faces. For each face of S', add a new vertex in the face and adjacent to all vertices on the boundary of that face. Let G be the new graph. Observe that G is a triangulated plane graph and has $(2p+1)+(4p-2) = 6p-1$ vertices. This means that G has $3(6p-1)-6 = 18p-9$ edges. With G as input for Algorithm A, in the first phase it can produce $S_1 = S$, and S_2 can be S plus one edge for each of the new vertices (the vertices in G not in S). The number of edges in S is $3p$. Hence S_2 can have $3p+(4p-2) = 7p-2$ edges, while G has $18p-9$ edges. Thus the ratio between the number of edges in S_2 and the number of edges in G is

$$\frac{7p-2}{18p-9}.$$

By choosing p as large as we wish, we get a ratio as close to 7/18 as we want. ■

2.2.1.1 Linear Time for Bounded-Degree Graphs

In case G has bounded degree d, we can implement Algorithm A in linear time. First we describe the implementation of the first phase. At any time, the vertices of the graph are partitioned into three sets: new, active and used. At the beginning, all the vertices are new and $E_1 = \emptyset$. The following process is repeated indefinitely. Test if there are any new vertices. If there are none, halt. Test if there are any active vertices. If there are none, choose a new vertex and make it active. Choose an active vertex x and "use" it: that is, while there exist two new vertices y and z adjacent to x and to each other, choose such a pair, add the edges of the triangle xyz to E_1 and make y and z active. When there are no two new vertices adjacent to x and adjacent to each other, mark x used.

We will show in detail how "using" one vertex takes time $O(d^2)$. All the other operations—keeping lists of new and active vertices, initializing an array with the status of each vertex, initializing an auxiliary array and maintaining the list of edges in E_1—can easily be implemented in $O(n)$ time. This means the running time of this implementation is $O(nd^2)$.

Before and after "using" a vertex x, the auxiliary array has a 0 for each vertex. Going through the adjacency list of x, we change to 1 the entry of the auxiliary array for each new vertex encountered. Then, going again through the adjacency list of x, for each new vertex u encountered, go through the adjacency list of u, checking if some neighbor of u is marked 1 in the auxiliary array. This takes time $O(d)$ for each neighbor of x, for a total of $O(d^2)$ time. In this way we find, one after the other, all triangles formed by x and two new vertices. Each time such a triangle is found, mark as active the two new vertices in the triangle and place a 0 for each in the auxiliary array. At the end of this process, go through the adjacency list of x and change back to zero all remaining entries which are one in the auxiliary array.

First we note that E_1 is a triangular cactus. Indeed, at any time when we add a triangle to E_1, two of the vertices of the triangle are of type new, and since a new vertex is always an isolated vertex in $G[E_1]$, adding the three edges of the triangle keeps $G[E_1]$ a triangular cactus.

To prove that E_1 is maximal, we maintain the following invariant. All vertices which are active at a given time are in the same connected component of $G[E_1]$ at that time. To see this, note that a node u is made active either because there are no active nodes left, in which case u is the only node active, or is made active by an active vertex x, a neighbor in G of u which is being used, and in this case the edge ux is included in $G[E_1]$.

Now, let us prove that E_1 at the end of the process is maximal, in that no triangles can be added to it. Consider a triangle xyz not in E_1. The triangle xyz has at least two vertices in the same component of $G[E_1]$ for the following reason. When the process stops, it is because there are no new vertices. Thus either x, y and z are active, or at least one of them is used. If x, y and z are active, by the invariant, the three of them are in the same component of $G[E_1]$. In the other case, assume x is the first among x, y and z to be used. While x was being used, the triangle xyz was processed. It was not added to E_1 because at least one of y and z, say y, was active. Since x was also active at that moment, x and y were (and still are) in the same component of $G[E_1]$, by the invariant.

In this paragraph we describe how to implement in linear time the second phase of Algorithm A. The time is $O(|V(G)|+|E(G)|)$ and does not depend on the graph's having bounded degree. Recall that at the end of the first phase we have a set E_1 of edges. First compute the connected components of $G[E_1]$. Let $C(v)$ be the connected component of $G[E_1]$ containing vertex $v \in V(G)$. Then construct a multigraph H in the following way: $V(H)$ is the set of connected components of $G[E_1]$, and for each edge $uv \in E(G)$, include in $E(H)$ one edge (labeled uv) with endpoints $C(u)$ and $C(v)$ if $C(u) \neq C(v)$. Run a graph transversal algorithm (for example DFS) on H and obtain a maximum spanning forest F (which we view as a subset of $E(H)$) of H. Output $E_2 = \{uv | \text{there is an edge in } F \text{ labeled } uv\}$.

2.2.2 A Better Algorithm

The new algorithm, Algorithm B below, finds a *maximum* triangular structure (one with the maximum number of edges) in a given graph G. Algorithm B has performance ratio 4/9, and can be implemented in time $O(m^{3/2} n \log^6 n)$. Now let us present the algorithm and the analysis of its performance ratio.

Algorithm B also has two phases. In the first one, B constructs a maximum triangular cactus S_1 in G. We will show later how to use a graphic matroid parity algorithm to construct S_1. In the second phase, B extends S_1 to a triangular structure S_2 in G, as before, by adding to S_1 as many edges as possible which do not form new cycles.

Algorithm B

Let S_1 be a *maximum* triangular cactus in G.

Starting with $E_2 = E(S_1)$, repeatedly (as long as possible) find an edge e in G whose endpoints are in different components of $G[E_2]$, and add e to E_2.

Let $S_2 := G[E_2]$.

Output S_2.

Observe that S_2 is a triangular structure in G, and therefore is planar. To analyze the algorithm, we need a definition. In any graph L, let $mts(L)$ denote the number of edges in a maximum triangular structure in L. Define $\rho(L) = mts(L)/|E(L)|$ if $E(L) \neq \emptyset$, and $\rho(L) = 1$ if $E(L) = \emptyset$.

The main result of this section is Corollary 2.2.11, which states that the performance ratio of Algorithm B is 4/9. As we will see, the proof of Corollary 2.2.11 is not difficult, given the next two theorems.

Let $G = (V, E)$ be a graph on n vertices. Let $\mathcal{P} = \{V_1, \ldots, V_k\}$ be a partition of the vertices of G and $\mathcal{Q} = \{E_1, \ldots, E_m\}$ be a partition of the edges of G. For $1 \leq i \leq m$, let

u_i denote the number of classes (sometimes referred to as *color classes*) V_j of $\mathcal{P}$ met by E_i. We call the ordered pair $(\mathcal{P}, \mathcal{Q})$ *valid for* G if every triangle of G has either at least two vertices in the same part of $\mathcal{P}$ or all three edges in the same part of $\mathcal{Q}$. Set

$$\Phi(\mathcal{P}, \mathcal{Q}) = n - k + \sum_{i=1}^{m} \lfloor \tfrac{u_i - 1}{2} \rfloor. \tag{4}$$

Note that $\Phi(\mathcal{P}, \mathcal{Q}) \geq 0$, and that there is always a valid pair $(\mathcal{P}, \mathcal{Q})$ for G (e.g., $\mathcal{P} = \{V(G)\}, \mathcal{Q} = \{E(G)\}$).

According to Lovász and Plummer [46], we have the following theorem:

Theorem 2.2.2 *The number of triangles in a maximum triangular cactus in a graph G is equal to the minimum of $\Phi(\mathcal{P}, \mathcal{Q})$ taken over all valid pairs $(\mathcal{P}, \mathcal{Q})$ for G.* [1]

Let G be a planar graph with n vertices. Embed G in the plane without crossing edges, obtaining a plane graph. Let t be the number of edges missing for this embedding to be triangulated. A triangulated plane graph has $3n-6$ edges, if $n \geq 3$. So $t = (3n-6) - |E(G)|$; t does not depend on the embedding.

We will prove the following theorem.

Theorem 2.2.3 *Let G be a connected planar graph with $n \geq 3$ vertices. Let t be the number of missing edges, defined as above. Then*

$$\Phi(\mathcal{P}, \mathcal{Q}) \geq \frac{1}{3}(n - 2 - t), \tag{5}$$

[1] Lovász was contacted and agreed that the formula given here, which differs slightly from that in [46], is correct.

for all valid pairs $(\mathcal{P}, \mathcal{Q})$ *for* G.

As shown in Theorem 2.2.10, it follows easily from Theorems 2.2.2 and 2.2.3 that $\rho(H) \geq 4/9$ for all planar graphs H.

The proof of Theorem 2.2.3 is technical and long, so let us start with a general description. The proof is by induction on the number of edges in G, the basis being simple to prove, as we will see. The induction step starts with a connected plane graph G, and a valid pair $(\mathcal{P}, \mathcal{Q})$, where $\mathcal{P} = \{V_1, \ldots, V_k\}$ and $\mathcal{Q} = \{E_1, \ldots, E_m\}$. We concentrate on *singletons*, which are vertices v of G such that $\{v\} \in \mathcal{P}$.

The singletons are partitioned into three sets A, B and C as follows. Let G' be a triangulated plane supergraph of G with the same set of vertices. A is the set of singletons all of whose neighbors in G' have the same color (all neighbors in G' are in the same V_j for some j). A *facial triangle* is a triangle in G' which is the boundary of some face of G'. B is the set of singletons v not in A such that there is a facial triangle containing v with all edges in one edge class E_i for some i. We say singleton $v \in B$ *chooses* some such edge class E_i. C contains all remaining singletons.

Each singleton in B will choose an edge class E_i. For $1 \leq i \leq m$, let u_i be the number of classes V_j of $\mathcal{P}$ met by E_i, and let s_i be the number of singletons in B that have chosen E_i.

The induction step is divided into two cases. The simpler case is the one in which there is an i such that $u_i = s_i = 4$. In this case, induction is applied to a smaller graph constructed from G, and after some accounting, the theorem follows.

The hard case is the one in which there is no i such that $u_i = s_i = 4$. In this case, the result will be proved directly—induction will not be applied. Recall that t is the number of edges missing for G to be triangulated (i.e., the number of edges in G' not in G). Let s be the number of singletons, p be the number of non-singleton vertices, that is, $p = n - s$

(where n is the number of vertices of G), and q be the number of non-singleton colors, that is, $q = k - s$. Then the following upper bounds on $|A|, |B|$ and $|C|$ will be proved.

- $|A| \leq 2p - 3q - c'$ (Lemma 2.2.4)
- $|B| \leq 3\sum_{i=1}^{m} \lfloor \frac{u_i - 1}{2} \rfloor$ (Lemma 2.2.5)
- $|C| \leq t + c'$ (Lemma 2.2.6)

for some carefully chosen c'.

The upper bounds on $|A|$ and $|B|$ are not hard. The hard one is the upper bound on $|C|$. The theorem follows easily from these upper bounds and some accounting.

Proof of Theorem 2.2.3. By induction on the number of edges of G. Let us denote by E_G the edge set of G and by V_G its vertex set.

Basis: $|E_G| \leq 2n - 4$.

Then $t \geq (3n-6) - (2n-4) = n - 2$ and therefore $\Phi(\mathcal{P}, \mathcal{Q}) \geq 0 \geq \frac{1}{3}(n - 2 - t)$ for all valid pairs $(\mathcal{P}, \mathcal{Q})$ for G.

Inductive step: $|E_G| > 2n - 4$.

Suppose that for any connected planar graph H on n vertices such that $|E_H| < |E_G|$, $\Phi(\mathcal{P}, \mathcal{Q}) \geq (1/3)(n - 2 - t_H)$ for all valid pairs $(\mathcal{P}, \mathcal{Q})$ for H, where t_H denotes the number of edges missing for H to be triangulated, Let us prove that $\Phi(\mathcal{P}, \mathcal{Q}) \geq (1/3)(n - 2 - t)$ for all valid pairs $(\mathcal{P}, \mathcal{Q})$ for G.

We begin by embedding G in the plane without crossings, obtaining a plane graph. Next, we augment G to get a triangulated plane graph G'. The edges we add are called *missing edges.* The number of missing edges in G' is t.

Let $(\mathcal{P}, \mathcal{Q})$ be a valid pair for G, with $\mathcal{P} = \{V_1, \ldots, V_k\}$ and $\mathcal{Q} = \{E_1, \ldots, E_m\}$. As before, for $1 \leq i \leq m$, let u_i denote the number of color classes V_j of $\mathcal{P}$ met by E_i. We may assume that

$$u_i \leq 2 \Rightarrow |E_i| = 1. \tag{6}$$

If $u_i \leq 2$ and $|E_i| > 1$, then we can split E_i into individual edges, obtaining a new edge partition $\mathcal{Q}'$. Any triangle in G with three edges in the old E_i also has at least two vertices of the same color, because $u_i \leq 2$. Therefore $(\mathcal{P}, \mathcal{Q}')$ is also a valid pair for G. And, moreover, $\Phi(\mathcal{P}, \mathcal{Q}') = \Phi(\mathcal{P}, \mathcal{Q})$, because $\lfloor \frac{u-1}{2} \rfloor = 0$ when $1 \leq u \leq 2$.

Assume that for $1 \leq j \leq q$, $|V_j| = p_j \geq 2$ and for $q < j \leq k$, $|V_j| = 1$. Let us call a vertex x a *singleton* if $\{x\} \in \mathcal{P}$; thus the last $k - q$ color classes are singletons. Let $s = k - q$ be the number of singletons and $p = \sum_{i=1}^{q} p_i$ be the number of vertices which are not singletons. So

$$n = p + s \quad \text{and} \quad k = q + s. \tag{7}$$

Observe that $\mathcal{P}$ is a partition of the vertices of G' (since G and G' have the same vertex set), $\mathcal{Q}$ is a partition of the edges of G (not of G'), and for any triangle of G' (not of G), facial or not, at least one of the following three conditions holds.

(1) At least two of its vertices are of the same color. We say that the triangle *is covered by a color.*

(2) All three edges are in the same class E_i of $\mathcal{Q}$. We say that the triangle *is covered by an edge class.*

(3) One or more of its edges is a missing edge. This corresponds to the fact that this triangle does not exist in G. We say that the triangle *is covered by a missing edge.*

A *facial triangle* is a triangle in G' which is the boundary of some face of G'. (Note that a facial triangle is not necessarily a triangle in G, but it is a triangle in G'.) A facial triangle T is a *facial triangle neighboring vertex v* if it contains v.

Let us partition the set of singletons into three sets A, B, C, according to how the facial triangles neighboring these singletons are covered.

A: The set of singletons all of whose neighbors in G' are of the same color.

B: The set of singletons x not in A such that there is an edge class which covers one of the facial triangles neighboring x.

C: The set of all singletons not in $A \cup B$.

Notice that all the facial triangles neighboring a singleton in A (even the ones containing missing edges) are covered by a color. Also, not all facial triangles neighboring a singleton in C are covered by a color class (otherwise, all of its neighbors in G' would have the same color and this singleton would be in A). And no facial triangle neighboring a singleton in C is covered by an edge class (otherwise, this singleton would be put in set B, if it's not already in A).

Let a, b and c be the sizes of A, B and C, respectively. Observe that s, the number of singletons, satisfies

$$s = a + b + c. \tag{8}$$

For each singleton x in B, we choose an i such that E_i covers a facial triangle neighboring x. We denote by s_i the number of singletons x in B which have chosen E_i. Observe that E_i meets all singletons which have chosen E_i; therefore

$$s_i \leq u_i. \tag{9}$$

Moreover, if $|E_i| = 1$, then $s_i = 0$, because E_i cannot cover any triangle.

Case 1: There is an i such that $u_i = s_i = 4$.

This is the simpler case, in which we apply induction.

In this case, there are four distinct singletons x, y, w, z in B which are the only vertices E_i meets. Each of x, y, w, z has chosen E_i, implying that E_i must cover a facial triangle

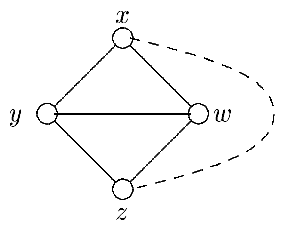

Figure 7: Edges in E_i. After removing yw and, if it exists, xz, no triangles are left.

neighboring x. This facial triangle meets only a subset of these four vertices. Without loss of generality, we may assume this facial triangle is xyw. E_i also covers a facial triangle neighboring z. Also without loss of generality, we may assume it is ywz. Then $\{xy, yw, xw, yz, wz\} \subseteq E_i \subseteq \{xy, yw, xw, yz, wz, xz\}$ (see Figure 7).

Let G_1 be G after we remove the edges yw and xz (if edge xz exists). From the partition $\mathcal{Q}$ of $E(G)$ we obtain a partition $\mathcal{Q}_1$ of $E(G_1)$ in the following way: all parts of $\mathcal{Q}_1$ are the same as the corresponding parts of $\mathcal{Q}$, except that E_i is replaced by four parts. Each of these four parts contains exactly one edge from $E_i - \{yw, xz\}$. Note that indeed $\mathcal{Q}_1$ is a partition of $E(G_1)$. Also, the pair $(\mathcal{P}, \mathcal{Q}_1)$ is valid for G_1: each triangle in G_1 is covered exactly as it was covered in G (the triangles covered by class E_i do not exist in G_1).

Observe that G_1 is connected, and so it is a connected planar graph. Moreover, it has fewer edges than G. Thus we can apply induction, and conclude that $\Phi(\mathcal{P}, \mathcal{Q}_1) \geq \frac{1}{3}(n - 2 - t_1)$, where t_1 is the number of missing edges for G_1 to be triangulated. Because we removed at most two edges from G to get G_1, $t_1 \leq t + 2$.

The new classes in $\mathcal{Q}_1$ have zero contribution to $\Phi(\mathcal{P}, \mathcal{Q}_1)$ (since any edge class of size one contributes zero), and, since the contribution of E_i to $\Phi(\mathcal{P}, \mathcal{Q})$ is $\lfloor \frac{4-1}{2} \rfloor = 1$, it follows that $\Phi(\mathcal{P}, \mathcal{Q}_1) = \Phi(\mathcal{P}, \mathcal{Q}) - 1$.

Putting all this together, we have

$$\Phi(\mathcal{P}, \mathcal{Q}) = \Phi(\mathcal{P}, \mathcal{Q}_1) + 1 \geq \frac{1}{3}(n - 2 - t_1) + 1 \geq \frac{1}{3}(n - 2 - (t + 2)) + 1 > \frac{1}{3}(n - 2 - t).$$

Case 2: There is no i such that $u_i = s_i = 4$.

This is the hard case, whose proof requires several pages. We start by defining c', used in the upper bounds on a and c.

Consider pairs (j, F), where $1 \leq j \leq q$ and F is a face of $G'[V_j]$, the plane subgraph of G' induced by the vertex set V_j. Let c' be the number of such pairs (j, F) such that some vertex not in A is embedded in F. Call each of these c' pairs *special.*

Recall that p is the number of vertices which are not singletons, q is the number of color classes with more than one vertex, t is the number of missing edges in G', and a, b, c are the numbers of singletons in A, B, C, respectively.

The next lemmas give upper bounds on a, b and c, which will be needed to complete the proof of Theorem 2.2.3.

Lemma 2.2.4 *$a \leq 2p - 3q - c'$.*

Proof. Each singleton x in A has all neighbors in G' of some color $j \leq q$; hence $p_j \geq 2$. Clearly x is embedded in a face of $G'[V_G - \{x\}]$. Because all neighbors of x in G' are in V_j, this face F is also a face of $G'[V_j]$, and there cannot be any other vertex embedded in F. Therefore the size a of A is at most the number of pairs (j, F) which are not special.

The maximum number of pairs (j, F), where F is a face of $G'[V_j]$, is $2p_j - 3$ (if $p_j = 2$ then it has only one face; if $p_j > 2$ then it has the maximum number of faces when it is triangulated, and in this case, it has $2p_j - 4$ faces). Therefore there are at most $\sum_{j=1}^{q}(2p_j - 3) = 2p - 3q$ pairs (j, F), where $j \leq q$ and F is a face of $G'[V_j]$. From these pairs (j, F), c' of them are special (have a vertex which is not in A embedded in F). Thus the number of pairs (j, F) which are not special is at most $2p - 3q - c'$, and therefore $a \leq 2p - 3q - c'$. ■

Lemma 2.2.5 $b \leq 3\sum_{i=1}^{m} \lfloor \frac{u_i - 1}{2} \rfloor$.

Proof. If we prove that $s_i \leq 3\lfloor \frac{u_i-1}{2} \rfloor$, then by summing $s_i \leq 3\lfloor \frac{u_i-1}{2} \rfloor$ over $i \in \{1, 2, ..., m\}$, the lemma follows, since $b = \sum_{i=1}^{m} s_i$.

First, by (6), if $u_i \leq 2$, then $|E_i| = 1$, which implies that $s_i = 0$. And, so $s_i = 0 = 3\lfloor \frac{u_i-1}{2} \rfloor$. If $u_i > 2$, then we use the fact that inequality (9) implies that $s_i \leq u_i$. As a consequence, $s_i \leq 3\lfloor \frac{u_i-1}{2} \rfloor$ can only be false if $u_i = s_i = 4$. But this does not happen because we are given that for no i does $u_i = s_i = 4$. ■

Lemma 2.2.6 is the difficult upper bound. After this, we do some accounting, and the theorem follows.

Lemma 2.2.6 $c \leq t + c'$.

Proof. Since there is nothing to prove if $c = 0$, so we may assume $c \neq 0$, that is,

$$C \neq \emptyset. \tag{10}$$

A missing edge covers at most two facial triangles. Therefore it suffices to prove the existence of at least $2c - 2c'$ facial triangles which must be covered by missing edges.

Let us associate a set of facial triangles with each component of $G'[C]$. More specifically, for a component of $G'[C]$ whose vertex set is D, let us associate a set of either $2|D|$ or $2|D|-2$ facial triangles, each with at least one vertex in D, such that each such facial triangle must be covered by a missing edge. We will make sure that each of these facial triangles will have its three vertices in different color classes and at least one of them is in C. (The triangle is not covered by a color class and, because of the vertex in C, it is not covered by an edge class. So it has to be covered by a missing edge.)

Observe that the sets of facial triangles corresponding to two different components of $G'[C]$ are disjoint (since there is no edge between two different components of $G'[C]$, there cannot be a triangle with vertices in two different components of $G'[C]$). If at most c' of the components were associated with a set with $2|D|-2$ facial triangles, then in total we would have at least $2c-2c'$ facial triangles which would have to be covered by missing edges (since $\sum_D |D| = c$). This would complete the proof of Lemma 2.2.6.

Let D be the vertex set of a component of $G'[C]$. A *relevant facial triangle for* D is a facial triangle of G' with at least two vertices in D. Let f_D be the number of faces of $G'[D]$ (which is a connected graph), f_3 be the number of faces of $G'[D]$ which are also faces of G', and e_D be the number of edges in $G'[D]$.

Let us start by proving that $C \neq V_G$. Because $|E_G| > 2n-4$, there are at most $(3n-6)-(2n-3) = n-3$ missing edges. A missing edge covers at most two facial triangles. So there are at most $2n-6$ facial triangles covered by missing edges. But in G' there are $2n-4$ facial triangles (all the faces of G'). Thus there are facial triangles covered by colors or by edge classes. If there are facial triangles covered by colors, then at least two vertices have the same color. This means that there are vertices which are not singletons, implying that $C \neq V_G$. If all vertices are singletons and a facial triangle is covered by an edge class, then the singletons neighboring this facial triangle are in B. It follows that in either case

$$C \neq V_G. \tag{11}$$

Embedded in the faces of $G'[D]$ are all the vertices of $V_G - D$. Now $V_G - D \supseteq V_G - C \neq \emptyset$.

Thus there is at least one face of $G'[D]$ which contains some vertex of $V_G - D \neq \emptyset$. This face of $G'[D]$ is not a face of G', since no face of G' contains any vertex. Thus $f_3 < f_D$.

Claim 2.2.7 *If $f_3 = f_D - 1$ then there are at least $2|D| - 2$ relevant facial triangles for D. If $f_3 \leq f_D - 2$ then there are at least $2|D|$ relevant facial triangles for D.*

Proof. A relevant facial triangle for D has at least two vertices in D, which means it contains an edge e in $G'[D]$. But e is contained in exactly two facial triangles of G'. These are the only two relevant facial triangles for D that contain e. This would give us $2e_D$ of the desired relevant facial triangles for D, except that not all of them are distinct. If a facial triangle is counted by two edges, then all three of its vertices are in D, which implies it is a face of $G'[D]$ (and, being a facial triangle, also of G'). Moreover, it was counted exactly three times (once for each of its edges). Therefore we have $2e_D - 2f_3$ relevant facial triangles for D. Now, if $f_3 = f_D - 1$, applying Euler's formula, we get $2e_D - 2f_3 = 2e_D - 2(f_D - 1) = 2(|D| - 2) + 2 = 2|D| - 2$ relevant facial triangles. If $f_3 \leq f_D - 2$, we get $2e_D - 2f_3 \geq 2e_D - 2(f_D - 2) = 2(|D| - 2) + 4 = 2|D|$. ■

Include these $2|D|$ (if possible) or $2|D| - 2$ relevant facial triangles in the set of facial triangles corresponding to $G'[D]$.

To guarantee that at most c' of the components of $G'[C]$ have only $2|D| - 2$ facial triangles in their corresponding sets, we will need to add two more facial triangles to some of the sets that currently have only $2|D| - 2$. Now let D be the vertex set of a component of $G'[C]$ such that all the faces of $G'[D]$ but one are faces of G', that is, $f_3 = f_D - 1$. (Any component induced by a set D' with $f_3 \leq f_{D'} - 2$ already has $2|D'|$ associated facial triangles.)

Since $f_3 = f_D - 1$, all the faces of $G'[D]$ but one are faces of G'. It follows that all the vertices in $V_G - D$ must be embedded in the same face of $G'[D]$ (that face is the only face which is not a face of G'). Applying Claim 2.2.8 (presented below) to G' and D (as M and S, respectively), we conclude that $G'[V_G - D]$ is connected.

We use the terminology given in [2] for a facial walk and for the length of a face.

Claim 2.2.8 *Let $M = (V, E)$ be a connected plane multigraph (possibly with loops) whose faces all have length at most three. Let S be a non-empty set of vertices of M such that the subgraph $M[S]$ of M induced by S is connected, and all vertices in $V - S$ are embedded in the same face of $M[S]$. Then $M[V - S]$ is connected.*

Proof. We may assume that $V - S \neq \emptyset$. By induction on the number of edges in $M[S]$.

Basis: $M[S]$ has no edge.

$M[S]$ consists of a unique vertex u. Suppose $M[V - S]$ is not connected. Because M is connected, all the components of $M[V - S]$ are adjacent to some vertex in S, i.e., to u. The plane embedding of M determines a cyclic ordering of the edges of M incident with u. Then there exist two consecutive edges in the cyclic ordering, say e_1 and e_2, such that their other endpoints belong to different components of $M[V - S]$. But this is impossible, because then the length of the face of M lying between e_1 and e_2 would be at least four, a contradiction.

Inductive step: There is an edge in $M[S]$.

If there is an e in $M[S]$ with two distinct endpoints u and v, contract e obtaining a new connected plane multigraph H with a new vertex w corresponding to u and v. (H's vertex set V' is $(V - \{u, v\}) \cup \{w\}$.) Let $S' = (S - \{u, v\}) \cup \{w\}$. The hypotheses of the claim hold for H and S': H is connected, $H[S']$ is connected ($H[S']$ is $M[S]$ after contracting e), all the faces of H have length at most three (since contracting edges cannot increase the length of a face), $S' \neq \emptyset$, and all vertices in $V' - S' = V - S$ are embedded in the same face of $H[S']$. (There is a natural correspondence between faces of a graph and faces of the graph obtained after contracting an edge of the graph.) Furthermore, $H[S']$ has fewer edges than $M[S]$. Hence by the inductive hypotheses, $H[V' - S']$ is connected, and hence $M[V - S] = H[V' - S']$ is connected.

If every edge in $M[S]$ is a loop, then let e be such a loop, and call its unique endpoint u. Note that because $M[S]$ is connected, $S = \{u\}$. Then e divides the plane (minus e and u) into two open regions R and R'. Because all vertices of $V - S$ are embedded in the same face of $M[S]$, there is no vertex in $V - S$ embedded in one of these two regions; without loss of generality, let it be R. So there is no vertex embedded in R. This implies that any edge embedded in R is a loop, with u as its endpoint.

We consider two cases. The first case is the one in which there is some edge embedded in R. Then let H be the graph obtained from M by removing all edges of M embedded in R (they are all loops). Note that H has fewer edges than M. We will prove that the hypotheses of the claim hold for H and S. H and $H[S]$ are connected since we removed only loops. The only face of H which is not a face of M is R, and the boundary of R consists of only e: R has length one. Therefore all the faces of H have length at most three. So by the inductive hypothesis, $H[V - S]$ is connected, and hence $M[V - S] = H[V - S]$ is connected.

In the second case, there is no edge embedded in R. Hence R is a face of M (with only one edge on the boundary). Let H be the graph obtained from M by removing e. Again, H has fewer edges than M. Let F be the other face of M having e in its boundary. Removing e creates only one new face F' in H (which is the union of R, F and e). The boundary of the new face F' is the union of the boundaries of R and F minus e. The boundary of R consists of only e. Thus the boundary of F' has one edge (edge e) fewer than the boundary of F, implying that the length of F' is one less than the length of F. Therefore all the faces of H have length at most three. So by the inductive hypotheses, $H[V - S]$ is connected, and therefore $M[V - S] = H[V - S]$ is connected. ■

We proceed with the proof of Lemma 2.2.6.

Because $G'[D]$ is connected, all vertices of D are embedded in the same face of $G'[V_G - D]$. Call this face F'.

Claim 2.2.9 *One of the following holds:*

(a) *All the vertices on the boundary of F' are of the same color $j \leq q$. In this case, F' is a face of $G'[V_j]$.*

(b) *There are two distinct facial triangles of G', each with exactly one vertex in D and the other two vertices of different colors.*

Proof. The boundary of F' is a closed walk in $G'[V_G - D]$. Moreover, $V_G - D$ cannot be a set containing only one vertex x. This is true because of the following reasoning. Otherwise this vertex would have to be a singleton (because vertices in D are singletons, none can have the same color as x), more specifically, a singleton in A or in B (it could not be in C, because it is adjacent to at least one vertex of D, and hence would be in the same component of $G'[C]$ as defined by D), with all neighbors in (D and hence) C. Any vertex of G' has at least two neighbors, and a singleton x in A has all the neighbors of the same color, and therefore none of its neighbors can be a singleton. It follows that no neighbor of x can be in C; hence $x \notin A$. A singleton x in B has a neighboring facial triangle (with vertices x, y and z) covered by an edge class. But then y cannot be in C (since it has a neighboring facial triangle covered by an edge class); hence $x \notin B$.

Therefore $|V_G - D| \geq 2$. Hence the boundary of F' cannot consist of only one vertex. (If there is only one vertex on the boundary of a face of a simple plane graph, then the graph consists of only this vertex, and this does not happen because $|V_G - D| \geq 2$.)

If all vertices on the boundary of F' have the same color j, then clearly $j \leq q$ (there are at least two vertices on the boundary of F' and hence in V_j, so $j \leq q$). Also, because the only vertices embedded in F' are the vertices of D, which are not of color j, F' is also a face of $G'[V_j]$. This is case (a).

If, on the other hand, not all the vertices on the boundary of F' have the same color, then, traversing the boundary of F' (which is a closed walk), we will find at least two "oriented" edges whose endpoints have different colors. Each edge on the boundary of F' is on the boundary of exactly two triangular faces of G'. At least one of these triangular faces is contained in F'. More specifically, if an edge on the boundary of F' is traversed twice, then the two triangular faces containing this edge on the boundary are contained in F'.

Let us show that any of these triangular faces contained in F' has a vertex of D on its boundary. Suppose not. A triangular face with the three vertices on its boundary in $V_G - D$ is a face of $G'[V_G - D]$. So this triangular face would be a face of $G'[V_G - D]$ contained in F', which is a face of $G'[V_G - D]$. Because faces of a graph are disjoint, this can only happen if F' coincides with this triangular face of G'. But this is impossible, because F' is not a face of G' (all vertices in D are embedded in F').

Traversing the boundary of F', for each "oriented" edge we traverse, we find a (distinct) facial triangle of G' contained in F', containing the (unoriented version of the) oriented edge and a vertex of D on its boundary (if the two "oriented" edges are not the same unoriented edge, then each appears on the boundary of a different triangle, otherwise, the one unoriented edge appears in two distinct triangles). The facial triangles corresponding to the two "oriented" edges we picked satisfy the requirements of (b). ■

If (b) holds, add the two extra facial triangles to the set of facial triangles corresponding to $G'[D]$. Note that they are different from the ones already in the set because they contain exactly one vertex of D and all the others contain at least two vertices of D. Each set D for which (b) holds is now associated with a set of at least $2|D|$ facial triangles.

For how many components of $G'[C]$ can (a) hold? Let us show that each component of $G'[C]$ for which (a) holds corresponds to one of the c' special pairs, and that different components correspond to different special pairs.

Let D be the vertex set of a component of $G'[C]$ for which (a) holds. Let us prove that there is a special pair (j, F') such that the only vertices of C embedded in F' are the vertices in D. Let F' be the face of $G'[V_G - D]$ in which the vertices of D are embedded. Recall that $G'[V_G - D]$ is connected. Because (a) holds for this component, F' is a face of $G'[V_j]$. Since the vertices of D are embedded in F', and vertices of D are not in A, some vertex not in A is embedded in F'; this makes the pair (j, F') special. The only vertices of G' embedded in F' are in D (because F' is a face of $G'[V_G - D]$). Since the only vertices of C embedded in F' are the vertices in D, no other component of $G'[C]$ (with a different set D) will correspond to this pair (j, F').

So the number of components such that (a) holds is at most the number c' of special pairs. Thus there are at most c' components of $G'[C]$ such that (a) holds, which completes the proof of Lemma 2.2.6, by the discussion in the third paragraph of the proof. ■

Using (4) and the upper bounds given by the previous lemmas, we conclude the proof of Theorem 2.2.3 as follows:

$$\begin{aligned}
\Phi(\mathcal{P}, \mathcal{Q}) &= n - k + \sum_{i=1}^{m} \lfloor \tfrac{u_i - 1}{2} \rfloor, \text{ by (4)}, \\
&= p - q + \sum_{i=1}^{m} \lfloor \tfrac{u_i - 1}{2} \rfloor, \text{ by (7)}, \\
&\geq p - q + \frac{1}{3} b, \text{ by Lemma 2.2.5}, \\
&= p - q + \frac{1}{3}(s - a - c), \text{ by Equation (8)}, \\
&\geq p - q + \frac{1}{3}(s - (2p - 3q - c') - (t + c')), \text{ by Lemmas 2.2.4 and 2.2.6},
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{3}(p+s) - \frac{1}{3}t \\
&= \frac{1}{3}n - \frac{1}{3}t, \text{ by the first equation in (7),} \\
&> \frac{1}{3}(n-2-t).
\end{aligned}
$$

This completes the proof of Theorem 2.2.3. ■

Recall that a triangular structure is a graph all of whose cycles are triangles. Also that $mts(L)$ denotes the number of edges in a maximum triangular structure in a graph L. And $\rho(L) = mts(L)/|E(L)|$ if $E(L) \neq \emptyset$, and $\rho(L) = 1$ if $E(L) = \emptyset$.

Theorem 2.2.10 *If H is a planar graph, then $\rho(H) \geq 4/9$.*

Proof. We may assume H is connected and has at least three vertices. Let t be the number of missing edges for H to be triangulated. By Euler's formula, $|E(H)| = 3n - 6 - t$, where $n \geq 3$ is the number of vertices of H.

A maximum triangular structure in H can be obtained by extending a maximum triangular cactus to a connected graph (by adding edges without forming any new cycles). Also, a maximum triangular structure in H has one more edge per triangle than a spanning tree of H.

From Theorems 2.2.2 and 2.2.3, the number of triangles in a maximum triangular cactus is at least $\frac{1}{3}(n-2-t)$. From this, we conclude that $mts(H) \geq n - 1 + \frac{1}{3}(n-2-t)$, and then

$$\rho(H) \geq \frac{n - 1 + \frac{1}{3}(n-2-t)}{3n-6-t} = \frac{4n-5-t}{9n-18-3t} \geq \frac{4}{9},$$

for all $t \geq 0$. ■

Corollary 2.2.11 *The performance ratio of Algorithm B is exactly* $\frac{4}{9}$.

Proof. Let G be a graph and H be a maximum planar subgraph of G. Clearly $mts(G) \geq mts(H)$. Now $Opt(G) = |E(H)|$ implies that $\mathrm{B}(G)/Opt(G) = mts(G)/|E(H)| \geq mts(H)/|E(H)| = \rho(H)$. By Theorem 2.2.10, $\rho(H) \geq 4/9$ for any planar graph H. And from this we infer that the performance ratio of B is at least 4/9.

Next we prove that the performance ratio of Algorithm B is at most 4/9. Let G' be any triangulated plane graph on n' vertices. Call V' the vertex set of G'. Since G' is triangulated, G' has $2n' - 4$ (triangular) faces. For each face of G', add a new vertex in the face and adjacent to all three vertices on the boundary of that face. Let G be the new graph and let V be the vertex set of G. Observe that G is a triangulated plane graph, and has $n' + (2n' - 4) = 3n' - 4$ vertices. Therefore G has $3(3n' - 4) - 6 = 9n' - 18$ edges. Let S be a maximum triangular structure in G.

Any edge in G has at least one endpoint in V'. Moreover, $|V'| = n'$. Therefore a maximum matching in G has at most n' edges (each with at least one distinct endpoint in V'). The following lemma is observed in [46, p. 440].

Lemma 2.2.12 *If S is a triangular structure with t triangles in a given graph G, then there is a matching in G of size t.*

Using Lemma 2.2.12, we conclude that S has at most n' triangles. Recall that S, being a triangular structure, is a spanning tree of G plus one edge per triangle in S, which implies that S has at most $(3n' - 5) + n' = 4n' - 5$ edges. Furthermore, G has $9n' - 18$ edges. Therefore the ratio between the number of edges in S and the number of edges in G is

$$\frac{4n' - 5}{9n' - 18}.$$

By choosing n' as large as we wish, we get a ratio as close to $4/9$ as we want. ■

How can one find a maximum triangular cactus quickly? A graphic matroid parity algorithm can be used to construct a maximum triangular cactus in a given graph [46]. The problem solved by a graphic matroid parity algorithm is Graphic Matroid Parity (GMP): given a multigraph $G' = (V', E')$ and a partition of the edge set E' into pairs of distinct edges $\{f, f'\}$, find a (simple) forest F with the maximum number of edges, such that $f \in F$ if and only if $f' \in F$, for all $f \in E'$.

Let us show how to reduce the problem of finding a maximum triangular cactus in a given graph $G = (V, E)$ to GMP. This is done by describing a multigraph $G' = (V', E')$ and a partition $\mathcal{P}$ of E' into pairs of distinct edges of E', such that, from a solution to GMP for G' and $\mathcal{P}$, we can construct a maximum triangular cactus in G.

First, let $V' = V$. Now, let us describe E' and the partition $\mathcal{P}$. Initially, $E' = \emptyset$ and $\mathcal{P} = \emptyset$. For each triangle in G with edge set T, let e, e' be any pair of distinct edges in T. Add two new edges f and f' to E', f with the same endpoints as e, and f' with the same endpoints as e'. We say that T *corresponds to* $\{f, f'\}$. Insert $\{f, f'\}$ into $\mathcal{P}$. The construction of E' and $\mathcal{P}$ is complete.

We say a forest F in G' is *valid* if $f \in F$ if and only if $f' \in F$, for all f in E'. Observe that any valid forest has an even number of edges. The following lemma states a relation between valid forests in G' and triangular cacti in G. Let m and n be the number of edges and vertices, respectively, in G.

Lemma 2.2.13 *There is a valid forest F in G' with $2p$ edges if and only if there is a triangular cactus S in G with p triangles. Moreover, S can be obtained from F in time $O(n)$.*

Proof. Assume that S is a triangular cactus in G with p triangles. Note that an edge in

S appears in exactly one cycle; otherwise that edge is a chord in a cycle of length at least four. So if in each triangle we replace the three edges by the two edges f, f', we are left with a valid forest in G', with $2p$ edges.

Assume now that F is a valid forest in G' with $2p$ edges. Note that $2p \leq n-1$. For each pair $\{f, f'\}$ in F, substitute f, f' by e, e' and add the third edge of the corresponding triangle. The substitution can be done in $O(1)$ time per pair, in a total time of $O(p)$, which is $O(n)$.

Call the resulting graph S. Next we prove that S is a triangular cactus with p triangles. Note that S is a subgraph of G. Each edge in S appears in a cycle: the corresponding triangle. In order to show that S is a cactus, we need to prove that each cycle has length three. We can substitute f, f' by e, e' and add the third edge for each pair $\{f, f'\}$ one at a time, and we maintain the invariant that each cycle consists of $\{e, e'\}$, corresponding to a matched pair $\{f, f'\}$ of F, and the third edge of the corresponding triangle. This property is true before substituting the first pair and adding any third edge, since F is a forest. Let us prove that we maintain the invariant after substituting f, f' by e, e' and adding the third edge e'' of a triangle whose edges are $\{e, e', e''\}$. We know that before substituting f, f' by e, e' and adding e'', f and f' did not appear in any cycle, by the invariant. We need to show that substituting f, f' by e, e' and adding e'' only creates the cycle e, e', e''. Because e is parallel to f and e' is parallel to f', any cycle created by the substitution of f, f' by e, e' and the addition of e'' has to contain e'' (otherwise f and f' would appear in a cycle).

Assume by contradiction that there is a cycle C which is not e, e', e''.

Assume first that C contains neither e nor e'. By replacing e'' by f and f' in C, we either get a cycle with f and f' and without e, e', e'', a contradiction, or a union of two cycles (this is the case in which C goes through the common endpoint of f and f'). But one of these cycles contains f and does not contain e, e', e'', a contradiction. If C contains

e and e'' but not e', then by replacing e and e'' by f' in C we get a cycle containing f' and not e, e', e'', contradicting the invariant. The case in which C contains e' and e'' but not e is similar. It follows that S is a cactus.

For each two edges in F we get a triangle in S, for a total of p triangles. This completes the proof. ■

As described by Chiba and Nishizeki [11], we can explicitly list all the triangles in a graph G with m edges in time $O(m^{3/2})$. So $|E'|$ is $O(m^{3/2})$.

Gabow and Stallmann [26] describe an algorithm for GMP, which runs in time $O(m'n' \log^6 n')$, where m' and n' are the number of edges and vertices, respectively, in the input graph. In our case, $n' = n$ and $m' = |E'|$, which is $O(m^{3/2})$. This gives a time bound of $O(m^{3/2} n \log^6 n)$ for this phase.

From the output of the Gabow-Stallmann algorithm, it is easy to find a maximum triangular cactus in time $O(n)$ (Lemma 2.2.13). Therefore the total time is $O(m^{3/2} n \log^6 n)$.

Chapter 3

Finding Large Outerplanar Subgraphs

The new algorithm presented in Chapter 2 produces outerplanar graphs. So it is an approximation algorithm for the Maximum Outerplanar Subgraph problem, which is the following NP-Complete problem: given a graph G, find an outerplanar subgraph of G of maximum size. The algorithm has performance ratio exactly $2/3$, which surpasses the bound of $1/2$ that is trivially obtained by producing a spanning tree, as we will see later.

3.1 Triangular Structures in Outerplanar Graphs

Recall that an outerplanar graph is a planar graph which has an embedding where all vertices are on the boundary of the exterior face. (From now on, all the embeddings are planar, unless otherwise specified.)

An outerplanar graph G is a *maximal outerplanar graph* if no edge can be added without losing outerplanarity. As mentioned in [33, p. 106], every maximal outerplanar graph G with at least three vertices is a triangulation of a polygon (i.e., the boundary of the exterior face is a Hamiltonian cycle and each interior face is triangular). By [33, Cor. 11.9], G must have a vertex of degree two and $2|V(G)|-3$ edges (this last statement is also true for $|V(G)|=2$).

Lemma 3.1.1 *Let H be an embedding of a maximal outerplanar graph as the triangulation* *of a polygon. If H has an odd number $n = 2p + 1$ of vertices, then there is a triangular cactus in H with p triangles. If H has an even number $n = 2p$ of vertices and xy is an edge on the boundary of the exterior face, then there is a triangular cactus S in H with $p - 1$ triangles such that in S, x and y are not connected.*

Notice that we obtain the maximum number of triangles possible. In the former case all vertices are in the same component of the cactus, while in the latter, the cactus has two components.

Proof. The proof is by induction on n, the number of vertices of H. The case $n = 1$ is trivial. If $n = 2$ (in this case there is only one edge and $p = 1$), the theorem is true.

We inductively construct a triangular cactus of the given size.

Let $n = 2p + 1$. Let u be a vertex of degree two. Let x and y be its neighbors. They are adjacent, since interior faces are triangles. The graph $H - \{u\}$ is maximal outerplanar (since it has $(2n - 3) - 2 = 2(n - 1) - 3$ edges) and has an even number of vertices. It is easy to check that if a triangular cactus S' in this smaller graph has the property that x and y are not connected in S', we can add the triangle xyu to get a triangular cactus in H. The size of this cactus is $p - 1$, by induction, plus one, for a total of p.

Let $n = 2p$ and let the edge xy be on the boundary of the exterior face. This edge is on the boundary of a triangular face xyv on the inside. Walking along the Hamiltonian cycle which is the boundary of the exterior face, starting at v and in the direction that visits x just before y, let D_1 be the set of vertices visited between v and x, inclusive, and let $n_1 = |D_1|$. Walking along the Hamiltonian cycle in the opposite direction again starting at v, let D_2 be the set of vertices visited between v and y, inclusive, and let $n_2 = |D_2|$; $D_1 \cap D_2 = \{v\}$ and $D_1 \cup D_2 = V(H)$. The only edge in H between $D_1 - \{v\}$ and $D_2 - \{v\}$ is the edge xy.

Let H_1 be the subgraph of H induced by vertex set D_1, with, say, e_1 edges, and let H_2

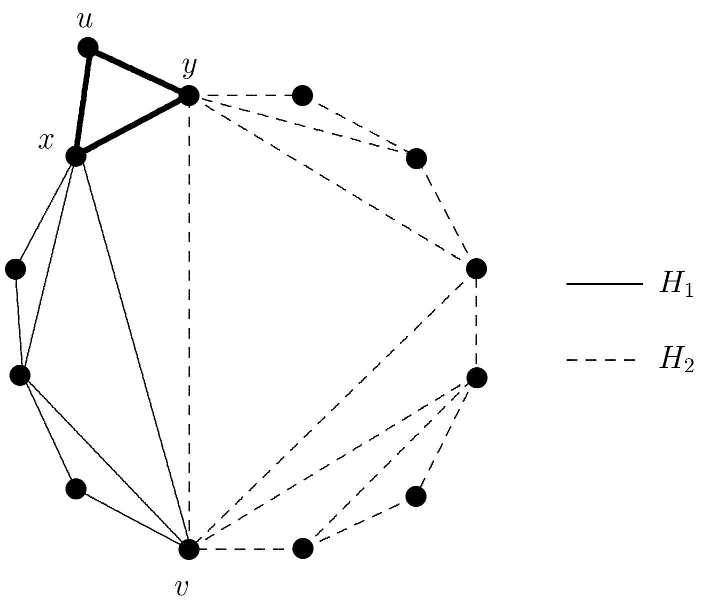

Figure 8: Graphs H_1 and H_2, built from H.

be the subgraph of H induced by vertex set D_2, with, say, e_2 edges (see Figure 8).

We have n_1+n_2 is odd, since v is counted twice. Let us say without loss of generality that $n_1 = 2p_1+1$ is odd and $n_2 = 2p_2$ is even. Then $n = 2(p_1+p_2)$. We have $e_1+e_2 = (2n-3)-1$, as from H only the edge xy is not an edge of either H_1 or H_2. Since $e_1 \leq 2n_1 - 3$ and $e_2 \leq 2n_2 - 3$, we infer that $e_1 + e_2 \leq 2(n_1 + n_2) - 6 = 2(n+1) - 6 = 2n - 4$. Since, in fact, $e_1 + e_2 = 2n - 4$, we infer that $e_1 = 2n_1 - 3$ and $e_2 = 2n_2 - 3$. Thus both H_1 and H_2 have to be maximal outerplanar, as they have the maximum allowable number of edges.

Then by the inductive hypothesis we can construct in H_1 a cactus S_1 with p_1 triangles. If we apply the inductive hypotheses to H_2 with vy being the edge on the exterior face, we obtain a triangular cactus S_2 with $p_2 - 1$ triangles in which y and v are not connected. Then putting together the edges of S_1 and S_2 we get S, a cactus in H. In the new cactus

S, any possible $x-y$-path must visit v, since neither S_1 nor S_2 has edge xy. But in S_2, y and v are not connected. It follows that x and y are not connected in S, so S is the desired cactus. S has $p_1 + (p_2 - 1)$ triangles, which is exactly the number we wanted. ■

In conclusion, for a maximal outerplanar graph with n vertices, we can find a triangular structure with $\lfloor \frac{n-1}{2} \rfloor$ triangles.

3.2 The New Algorithm for Outerplanar Graphs

Note that a triangular structure is an outerplanar graph. So, algorithm B presented in Chapter 2 produces outerplanar graphs.

The Maximum Outerplanar Subgraph problem—given G, find an outerplanar subgraph of G of maximum size—is known to be NP-complete [28, p. 197], motivating the interest in approximation algorithms for it.

Any algorithm which produces a spanning tree has performance ratio at least 1/2, because any outerplanar graph on $n \geq 2$ vertices has at most $2n-3$ edges (see below). A careful analysis shows that the performance ratio of algorithm B when used for the Maximum Outerplanar Subgraph problem is at least 2/3. This is an easy consequence of the following theorem, which gives a lower bound on $\rho(H)$. (Recall $\rho(H) = mts(H)/|E(H)|$, where $mts(H)$ denotes the number of edges in a maximum triangular structure in H.)

Theorem 3.2.1 *If H is outerplanar, then $\rho(H) \geq 2/3$.*

Proof. Let H be any 2-connected outerplanar graph. We add t edges to obtain a maximal outerplanar plane graph H'. Note that H' has $2n-3$ edges and a triangular structure S with exactly $\lfloor \frac{n-1}{2} \rfloor$ triangles.

However, the t missing edges can destroy at most t of these triangles in S, because S is a cactus. If $t \geq n/2$, we infer that

$$\rho(H) \geq \frac{n-1}{2n-3-n/2} > \frac{2}{3}.$$

Assume to the contrary that $t \leq \lfloor \frac{n-1}{2} \rfloor$. Then the number of edges in the triangular structure S is at least $n - 1 + (\lfloor \frac{n-1}{2} \rfloor - t)$. Thus

$$\rho(H) \geq \frac{n-1+\lfloor \frac{n-1}{2} \rfloor - t}{2n-3-t}.$$

The worst case is achieved when $t = \lfloor \frac{n-1}{2} \rfloor$ and is $2/3$.

If H is not 2-connected, we can do the above analysis for each of the 2-connected components (an edge appears in exactly one 2-connected component) and infer that a maximum triangular structure has $2/3$ of the edges in H. ■

The theorem above is tight in the sense that there are outerplanar graphs H for which $\rho(H)$ is arbitrarily close to $2/3$. In fact, there are outerplanar graphs H_i with $2i$ vertices and $3i-2$ edges which do not have any triangles. For such an outerplanar graph H_i, a maximum triangular structure is simply a tree, that is, $mts(H_i) = 2i-1$, and so

$$\rho(H_i) = \frac{2i-1}{3i-2},$$

which can be made as close to $2/3$ as we wish.

In Figure 9, we show how H_6 could look like.

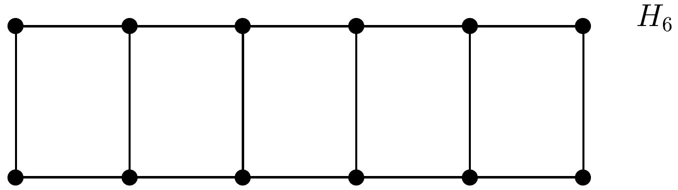

Figure 9: An outerplanar graph with 12 vertices, 16 edges, and no triangles.

Corollary 3.2.2 *Algorithm B has performance ratio exactly* 2/3 *for the Maximum Outerplanar Subgraph problem.*

Chapter 4

The Complexity of the Maximum Planar Subgraph Problem

We show that the Maximum Planar Subgraph (MPS) problem is Max SNP-hard, implying that there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at least $1 - \epsilon$ would imply that $P = NP$. In addition, we show that the complementary problem, called Nonplanar Deletion—given $G = (V, E)$, produce a smallest subset $L \subseteq E$ such that $(V, E - L)$ is planar—is also Max SNP-hard.

4.1 MPS is Max SNP-hard

The complexity class Max SNP was introduced by Papadimitriou and Yannakakis [51] and is an extension of the class NP. To prove that a problem is Max SNP-hard, we use the concept of *L-reduction*, given in Section 1.1.3.

Denote by $TSP_4(1, 2)$ the following variant of the Traveling Salesman problem: given a complete graph, a pair of distinct vertices x, y, and costs one or two for each edge, such that the graph induced by the edges of cost one has maximum degree at most four, find a Hamiltonian path from x to y of minimum cost. Papadimitriou and Yannakakis [52] showed that $TSP_4(1, 2)$ is Max SNP-hard.

We shall prove that $TSP_4(1, 2)$ L-reduces to MPS. The basic idea of the reduction comes from Liu and Geldmacher [45], where the decision version of MPS is proved to be

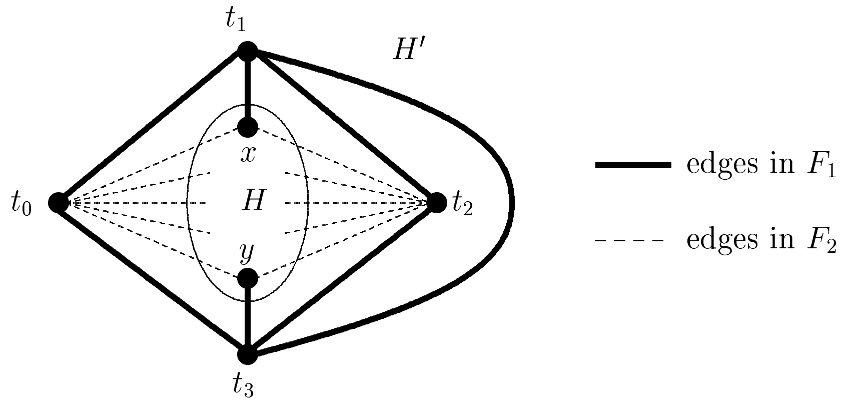

Figure 10: Graph H' constructed from H.

NP-complete.

Theorem 4.1.1 *The Maximum Planar Subgraph problem is Max SNP-hard.*

Proof. The first part of the L-reduction is the polynomial-time algorithm f and the constant α. Given any instance I of $TSP_4(1,2)$, f produces an instance G of MPS such that the cost of the optimum of G in MPS, denoted $Opt_{MPS}(G)$, is at most α times the cost of the optimum of I in $TSP_4(1,2)$, denoted by $Opt_{TSP_4(1,2)}(I)$, i.e., $Opt_{MPS}(G) \leq \alpha \cdot Opt_{TSP_4(1,2)}(I)$.

Consider an instance I of $TSP_4(1,2)$. I is a complete graph $K = (V,E)$, a pair of distinct vertices x, y of V and a subset E_1 of E consisting of the edges of cost one. Let $H = (V, E_1)$ and $H' = (V \cup T, E_1 \cup F_1 \cup F_2)$, where $T = \{t_0, t_1, t_2, t_3\}$, $T \cap V = \emptyset$, $F_1 = \{t_0t_1, t_0t_3, t_1t_2, t_1t_3, t_1x, t_2t_3, t_3y\}$ and $F_2 = \cup_{z \in V}\{t_0z, t_2z\}$ (see Figure 10).

Denoting by n the number of vertices of H, let G be the graph obtained from H' by (1)

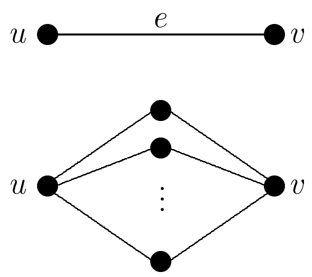

Figure 11: Edge e and its substitution by internally-disjoint paths of length two.

replacing each edge e in F_1 by $2n$ parallel internally-disjoint paths of length two (having new internal vertices) between the endpoints of e and (2) replacing each edge e in F_2 by eight parallel internally-disjoint paths of length two (having new internal vertices) between the endpoints of e (see Figure 11).

Clearly G can be obtained from I in time polynomial in the size of I.

Lemma 4.1.2 $Opt_{MPS}(G) \leq 124 \cdot Opt_{TSP_4(1,2)}(I)$.

Proof. Observe that $Opt_{TSP_4(1,2)}(I) \geq n-1$. A clear upper bound for $Opt_{MPS}(G)$ is the number of edges of G. To compute this, note first that H has maximum degree at most four by the definition of $TSP_4(1,2)$, and so H has at most $2n$ edges. Let us call the edges in F_1 *2n-edges* and the edges in F_2 *8-edges*. There are seven $2n$-edges: F_1 contains seven edges, each of them corresponding to $4n$ edges in G. There are $2n$ 8-edges: F_2 contains $2n$ edges, each of them corresponding to 16 edges in G. Hence the number of edges in G outside of H is $7 \cdot 4n + 16 \cdot 2n = 60n$. The total number of edges in G is therefore at most

$2n + 60n = 62n \leq 124(n-1)$. Therefore,

$$Opt_{MPS}(G) \leq 124(n-1) \leq 124 \cdot Opt_{TSP_4(1,2)}(I). \blacksquare$$

This finishes the first part of the L-reduction, since we can take $\alpha = 124$.

The second and hard part of the L-reduction is the constant β and the algorithm g. Given a planar subgraph of G with m edges, g produces in polynomial time a Hamiltonian path from x to y of cost t in K such that $|t - Opt_{TSP_4(1,2)}(I)| \leq \beta |m - Opt_{MPS}(G)|$. We shall see that $\beta = 1$ suffices.

First, given a planar subgraph P of G, let us describe another planar subgraph P' of G with at least as many edges as P. Moreover, P' shall contain all edges of G not in H.

Let e be a $2n$-edge or an 8-edge of H'. We say *e appears in P* if P contains both edges of *all* the paths of length two corresponding to e. In this case, we also say that the endpoints of *e are adjacent in P by the $2n$-edge* or *8-edge e*. We say *e is missing in P* if P contains both edges of *none* of the paths (of length two) corresponding to e. (In this case, if e is an 8-edge, then P is missing at least eight of its 16 edges in G corresponding to e.) It is possible that a $2n$-edge or an 8-edge of H' neither appears in P nor is missing in P.

Let us modify P so that any $2n$-edge or 8-edge of H' either appears in P or is missing in P. This is done as follows. If a $2n$-edge or an 8-edge e of H' neither appears nor is missing in P, then we insert in P all edges of G in the paths (of length two) corresponding to e. Note that P remains planar. Clearly, the number of edges in P cannot decrease by this operation. The new graph is also called P.

Now we can describe P'. We have three cases: (1) if some $2n$-edge e does not appear in P, then define P' to be the graph induced by all edges of G not in H; (2) if all the $2n$-edges and the 8-edges of H' appear in P, then let P' be the same as P; and (3) if all the $2n$-edges of H' appear in P, but not all the 8-edges, then we modify P to obtain P', as described in the next two paragraphs.

The idea is to remove from P some edges of H and add to P edges of H' not in H so that all the 8-edges of H' appear in the modified graph, and it remains planar and has at least as many edges as the original P.

Let U be the set of vertices v of H such that at least one of the two 8-edges incident to v in H' is missing in P. Observe that $|U| \geq 1$, as case (2) considered $|U| = 0$. For each vertex v in U, remove from P all edges of H incident to v in P (at most four edges are removed per vertex) and add to P all the edges outside of H so that the two 8-edges incident to v appear in P (at least eight edges are added, corresponding to the 8-edges incident to v missing in P). To guarantee that the graph obtained this way is planar, we must make room to embed the modified 8-edges. This is done by also removing from P all edges of H incident to y (if they were not already removed). Let P' be the graph obtained after all these modifications.

Lemma 4.1.3 *P' is planar and has at least as many edges as P.*

Proof. In case (1), we include in P' at least $2n$ edges that do not appear in P (at least one in each of the $2n$ paths corresponding to e), and we remove at most $2n$ edges, the maximum number of edges in H. So P' has at least as many edges as P. Moreover P' is planar.

There is nothing to be proved in case (2).

Case (3) is the complicated one. First note that P' has at least as many edges as G, since, for each vertex in U, we remove at most four edges and add at least eight. Furthermore, we remove at most four edges incident to y. Hence, we gain at least $(8-4)|U|-4 = 4|U|-4 \geq 0$ edges, since $|U| \geq 1$.

Now, let us show that P' is planar. We can think of the $2n$-edges and 8-edges as single

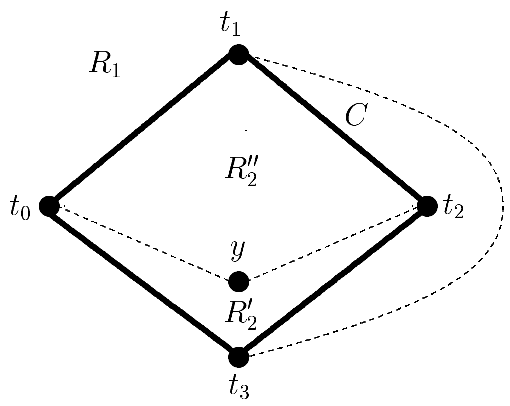

Figure 12: Cycle C, regions R_1 and $R_2 = R_2' \cup R_2''$.

edges, as they are in H' (since if we can embed a single edge, we can embed a $2n$-edge or an 8-edge as well). We will modify a given embedding of P into an embedding for P'.

Let C be the cycle (using four $2n$-edges) $t_0t_1t_2t_3t_0$. Observe that the $2n$-edges in C appear in P, since we are in case (3). Given an embedding for P, cycle C divides the plane into two regions, R_1, containing the $2n$-edge t_1t_3, and R_2 (see Figure 12). The $2n$-edge t_1t_3 separates t_0 from t_2 in R_1. Moreover, each vertex in $V - U$ (the vertices of H not in U) is adjacent in P by 8-edges to t_0 and t_2. Because t_0 and t_2 are separated in R_1, none of these vertices can be embedded in R_1, which implies they must be embedded in R_2. Keep these vertices (t_0, t_1, t_2, t_3 and the vertices in $V - U$) embedded as they are.

Now, observe that y is adjacent in P' only to t_0, t_2 (by 8-edges) and t_3 (by a $2n$-edge). Furthermore, y is the only vertex in H which is adjacent to t_3. This means, before we embed y, the vertices t_0, t_2 and t_3 are not separated in R_2. Therefore, y can be embedded in R_2 with the $2n$-edge t_3y and the 8-edges t_0y and yt_2 "next to" the $2n$-edges t_0t_3 and t_3t_2. The edges t_0y and yt_2 together split region R_2 into two regions R_2' containing edge

t_3y, and R_2'' containing vertex t_1. Observe that t_0 and t_2 are not separated in R_2'', since t_3 is the only vertex, besides t_0 and t_2, which is adjacent to y (by a $2n$-edge).

All the vertices in $U-\{x\}$ are adjacent in P' only to t_0 and t_2 (by 8-edges), and therefore they all can be embedded in R_2'' with their two 8-edges "next to" the 8-edges t_0y and yt_2. If $x \in U$ then observe that x is the only vertex in H which is adjacent in P' to t_1 (by a $2n$-edge). This means, before we embed x, the vertices t_0, t_1 and t_2 are not separated in R_2''. Therefore, x can be embedded in R_2'' with the $2n$-edge xt_1, and the 8-edges t_0x and xt_2 "next to" the $2n$-edges t_0t_1 and t_1t_2. If $x \notin U$ then it does not need to be moved in the embedding. The embedding obtained this way is a plane embedding of P', completing the proof that P' is planar. ■

Observe that P' contains all the edges of G not in H. Let F be the set of edges of H appearing in P'.

Lemma 4.1.4 *The graph $G_F = (V, F)$ is a collection of vertex-disjoint paths which can be extended in K (the complete graph on V) to a Hamiltonian path from x to y, in polynomial time.*

Proof. Let us prove that G_F satisfies the following four conditions: (1) There is no vertex of degree greater than two in G_F. (2) Vertices x and y have degree at most one in G_F. (3) There is no cycle in G_F. (4) If x and y are in the same component of G_F, then this component spans all vertices in V. We will prove each of these conditions holds by contradiction.

Suppose (1) does not hold. Let z_0 be a vertex in V of degree at least 3 in G_F. Let z_1, z_2, z_3 be three of its neighbors in G_F. (Notice that z_0, z_1, z_2, z_3 are distinct vertices of H, so they are distinct of t_0, t_2.) Then each one of t_0, t_2, z_0 is adjacent in P' to each one of z_1, z_2, z_3 (some of them are adjacent in P' by 8-edges). Therefore, $t_0, t_2, z_0; z_1, z_2, z_3$ define a subdivision of $K_{3,3}$ in P', a contradiction, because P' is planar. Thus, (1) holds.

Suppose (2) does not hold. If x has degree more than one in G_F, let z_1 and z_2 be two of its neighbors in G_F. (Notice that z_1 and z_2 are distinct vertices of H, distinct of x, so they are distinct of t_0, t_2.) Then each one of t_0, t_2, x is adjacent in P' to each one of t_1, z_1, z_2 (some of them are adjacent in P' by $2n$-edges or 8-edges). Therefore, $t_0, t_2, x; t_1, z_1, z_2$ define a subdivision of $K_{3,3}$ in P', a contradiction, because P' is planar. Analogously, we have a contradiction if y has degree more than one in G_F. Thus, (2) holds.

Suppose (3) does not hold. Let z_1, z_2, z_3 be three vertices in a cycle of G_F. (Since z_1, z_2, z_3 must have degree at least two, they are not x or y by condition (2), and they are not t_0 or t_2 since they are vertices of H.) Then z_1, z_2, z_3, t_0, t_2 are pairwise linked by internally vertex-disjoint paths (the path between t_0 and t_2 uses the two 8-edges incident to x, while the others use one $2n$-edge or 8-edge). Therefore, t_0, t_2, z_1, z_2, z_3 define a subdivision of K_5 in P', a contradiction, because P' is planar. Hence, (3) holds.

Suppose (4) does not hold. Let z_0 be a vertex in V which is not in the component having x and y in G_F. In this case, t_0, t_1, t_2, x, y are pairwise linked by internally vertex-disjoint paths (the path between t_0 and t_2 uses z_0, the path between t_1 and y uses t_3, the path between x and y is in G_F, and the others use one $2n$-edge or 8-edge). Therefore, t_0, t_1, t_2, x, y define a subdivision of K_5 in P', a contradiction, because P' is planar. Hence, (4) holds.

Therefore, the conditions hold. From (1) and (2), we conclude that G_F is a collection of paths. From (2) and (3), these paths can be extended in K (the complete graph on V) to a Hamiltonian path from x to y. Furthermore, note that this can be done in polynomial time. ■

Let HP be a Hamiltonian path from x to y containing all edges in F and some edges (in K) of cost two. HP exists by Lemma 4.1.4. Denote by m' the number of edges of P'

and by t the cost of HP.

Now the following lemma states that β exists, and specifically, $\beta = 1$. This will complete the proof of the theorem.

Lemma 4.1.5

$$t - Opt_{TSP_4(1,2)}(I) = Opt_{MPS}(G) - m',$$

and hence

$$|t - Opt_{TSP_4(1,2)}(I)| = |m' - Opt_{MPS}(G)|.$$

Proof. As in the proof of Lemma 4.1.2, the number of edges in G outside of H is $60n$. All these edges are in P'. Therefore, the number of edges in F is $m' - 60n$. And the cost of HP is

$$t = 2(n-1) - (m' - 60n). \tag{12}$$

Let Q be an optimal solution of MPS for G. Using the same argument for Q that we used for P, Lemma 4.1.4 and the argument above imply the existence of a Hamiltonian path of cost $2(n-1) - (Opt_{MPS}(G) - 60n)$. Therefore

$$Opt_{TSP_4(1,2)}(I) \leq 2(n-1) - (Opt_{MPS}(G) - 60n). \tag{13}$$

Given an optimum solution HP^* of $TSP_4(1,2)$ for I, we can construct a solution of MPS for G by selecting all the edges outside of H plus the edges of cost one in HP^*. Observe that these edges really determine a planar subgraph of G. Let z be the number of these edges. Since we have $60n$ edges in G outside of H, HP^* has $z - 60n$ edges of cost one and the remaining ones (of its $n-1$ edges) have cost two. This means that

$$\begin{aligned} Opt_{TSP_4(1,2)}(I) &= 2(n-1) - (z - 60n) \\ &\geq 2(n-1) - (Opt_{MPS}(G) - 60n), \end{aligned}$$

since $Opt_{MPS}(G) \geq z$.

Therefore, from (13) and (14), we have

$$Opt_{TSP_4(1,2)}(I) = 2(n-1) - (Opt_{MPS}(G) - 60n).$$

And this together with (12) means

$$t - Opt_{TSP_4(1,2)}(I) = Opt_{MPS}(G) - m'.$$

Hence $|t - Opt_{TSP_4(1,2)}(I)| = |m' - Opt_{MPS}(G)|$. ■

From $m \leq m'$, it follows that

$$t - Opt_{TSP_4(1,2)}(I) \leq Opt_{MPS}(G) - m$$

and

$$|t - Opt_{TSP_4(1,2)}(I)| \leq 1 \cdot |m - Opt_{MPS}(G)|.$$

This completes the proof of Theorem 4.1.1. ■

4.2 Extension for the Complementary Problem

Let us denote the complementary version of the Maximum Planar Subgraph problem by Nonplanar Deletion or NPD: given a graph G, find a smallest set of edges of G whose removal results in a planar graph.

A slight modification of the L-reduction presented above proves the following.

Theorem 4.2.1 *The Nonplanar Deletion problem is Max SNP-hard.*

Proof. The first part of the L-reduction is almost the same. From an instance I of $TSP_4(1,2)$, we construct G in exactly the same way. As before, $Opt_{TSP_4(1,2)}(I) \geq n-1$. As in the proof of Lemma 4.1.2, the maximum number of edges of G is $62n$. Thus, the optimum of G in NPD, denoted as $Opt_{NPD}(G)$, is at most $62n$. And then $Opt_{NPD}(G) \leq 62n \leq 124(n-1) \leq 124 \cdot Opt_{TSP_4(1,2)}(I)$. We can take $\alpha = 124$, as before.

In the second part, given an instance I of $TSP_4(1,2)$, let G be constructed from I as in the previous reduction. Let D be a subset of the edges of G whose removal results in a planar subgraph of G. We shall find in polynomial time a Hamiltonian path HP such that $t - Opt_{TSP_4(1,2)}(I) \leq d - Opt_{NPD}(G)$, where t is the number of edges in HP and $d = |D|$. Just as we took a planar subgraph P of G and found a planar subgraph P' which contains all edges of G not in H, and is at least as large as P, from D we can find a set D' of edges of G containing *none* of the edges of G not in H, which is at least as *small* as D. Applying Lemma 4.1.4, we can obtain a Hamiltonian path HP, as before, in polynomial time.

Now, let us prove that β exists. Let m' be the number of edges in P'. Note that $d' + m' = |E(G)|$. Moreover, $Opt_{MPS}(G) + Opt_{NPD}(G) = |E(G)|$. Therefore, $d' - Opt_{NPD}(G) = Opt_{MPS}(G) - m'$. Applying Lemma 4.1.5, we conclude that $t - Opt_{TSP_4(1,2)}(I) = d' - Opt_{NPD}(G)$, which, together with $d' \leq d$, implies that we can take $\beta = 1$. ∎

As a consequence, by results of Arora et al. [1], there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm for the Nonplanar Deletion with performance ratio at least $1 + \epsilon$ implies that $P = NP$.

In contrast with the Maximum Planar Subgraph problem, no α-approximation algorithm is known for the Nonplanar Deletion problem, where α is a constant.

The problem happens when the input graph is almost planar. For example, when the removal of $O(\log n)$ edges, where n is the number of vertices in the graph, is enough to make the graph planar. Our algorithm in this case can still remove $O(n)$ edges, resulting in a more than constant factor.

Chapter 5

Finding k-Edge-Connected Subgraphs: a Better Analysis

The Minimum k-Edge-Connected Spanning Subgraph is the following NP-hard problem: given a positive integer k and a k-edge-connected graph G, find a k-edge-connected spanning subgraph of G with the minimum number of edges. Khuller and Raghavachari presented the first algorithm with a performance ratio smaller than a constant smaller than two for all k. They proved an upper bound of 1.85 for the performance ratio of their algorithm. We improve their analysis, proving that the performance ratio of their algorithm is at most 1.75 for all k.

5.1 History of the Problem

As discussed in Section 1.3, the study of connectivity in graph theory has important applications in the areas of network reliability and network design. We concentrate on the Minimum k-Edge-Connected Spanning Subgraph problem.

This problem is known to be NP-complete [28], even for $k = 2$: a graph G is Hamiltonian if and only if a 2-edge-connected spanning subgraph of G with the minimum number of edges has n edges. So the goal is to look for good polynomial-time approximation algorithms for the problem.

Recall that the approximation ratio of an algorithm for a minimization problem is the supremum, over all possible inputs, of the ratio between the size of the output of the algorithm and the size of the optimum.

For a long time, two was the best approximation ratio achieved for all k. Just to illustrate, let us quickly describe how to get a ratio of two. Assume we have a c-edge-connected spanning subgraph H of G, for some $c < k$. Observe that if we add to H the edges of a maximum forest (i.e., one with the maximum number of edges) in $G - E(H)$, the resulting spanning subgraph of G is $(c+1)$-edge-connected. Using this fact, one can easily derive a procedure to construct a k-edge-connected spanning subgraph of G. Start with $S = \emptyset$, and repeatedly add to S the edges of a maximum forest in $G - S$. After k iterations, S has at most $k(n-1)$ edges, where n is the number of vertices in G, and it is the edge set of a k-edge-connected spanning subgraph of G. Now note that any k-edge-connected spanning subgraph of G must have at least $kn/2$ edges, since each vertex must have degree at least k. Thus this procedure leads to a polynomial-time algorithm with approximation ratio at most two. In fact, there are examples which prove that the performance ratio is exactly two.

Khuller and Vishkin [43] introduced the following concept: a *tree-carving in a graph* $G = (V, E)$ is a partition of the vertex set V into nonempty subsets $V_1, \ldots, V_t$ with the following properties. Each subset constitutes a node of a tree Γ. For every edge $e = uv$ of G, there are i and j such that $u \in V_i$, $v \in V_j$ and either $i = j$ or V_i is adjacent to V_j in Γ. In Figure 13, we give an example of a tree-carving in a graph G. Khuller and Vishkin used a tree-carving to prove the 1.5 bound on the approximation ratio of an algorithm they developed for $k = 2$.

Khuller and Raghavachari [41] presented the first algorithm with a performance ratio smaller than a constant smaller than two for all k. They proved an upper bound of 1.85 for the performance ratio of their algorithm. In this chapter, we improve their analysis, proving that the performance ratio of their algorithm is at most 1.75 for all k [21, 22].

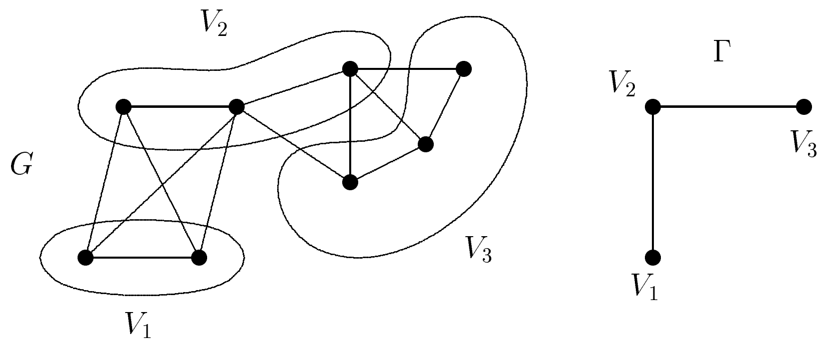

Figure 13: A graph G and a tree-carving in G.

The currently best known performance ratio for the Minimum k-Edge-Connected Spanning Subgraph problem is $1 + 2/(k + 1)$. It is achieved by a new algorithm developed by Cheriyan and Thurimella [9] concurrently with the research presented in this chapter.

5.2 Khuller and Raghavachari's Algorithm

We start by reviewing Khuller and Raghavachari's algorithm [41] and their upper bound of 1.85 on its performance ratio. Their bound is actually better than 1.85 for small values of k: it is 1.5 for $k = 2$, 1.666... for $k = 3$, 1.75 for $k = 4$, 1.733... for $k = 5$, etc., being tight for $k = 2$ and 3.

Before describing the algorithm, some preliminaries are needed.

Consider a graph G, a DFS forest F in G, and the corresponding DFS numbering of the vertices of G. For each edge e in G, call *top (bottom)* the endpoint of e with smaller (bigger) DFS number. Recall that any edge of G which is not in F is a back edge. In fact, this means that its top endpoint is an ancestor in F of its bottom endpoint. For each edge

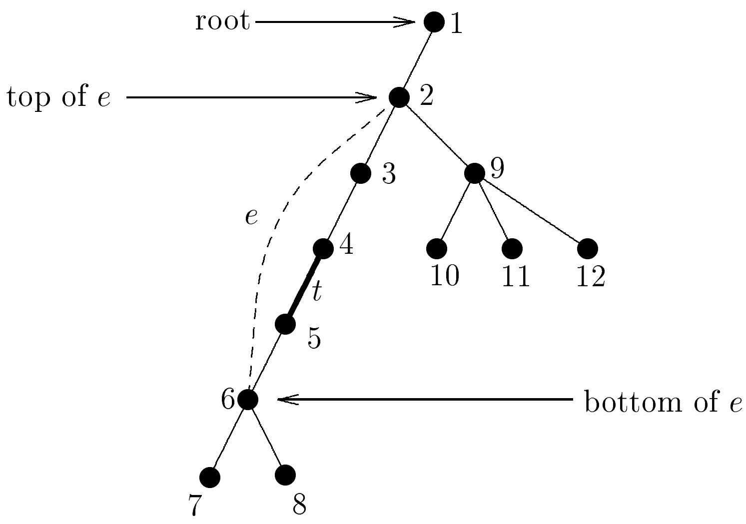

Figure 14: A DFS forest F, plus an edge e which is a back edge of the edge t in F.

t in F, we call an edge e of G which is not in F a *back edge of* t if t lies on the unique path in (V, F) between the two endpoints of e. (See Figure 14.)

The following lemma is a restatement of Lemmas 3.1, 3.2, 3.3 and 3.7 in [41]. We present it here, since it permits a better understanding of the algorithm and thus of our analysis.

Lemma 5.2.1 *Let k be a positive integer, $G = (V, E)$ be a k-edge-connected graph, and i be an integer, $0 \leq i < k - 1$. Let S and F be two disjoint subsets of E such that (V, S) is an i-edge-connected spanning subgraph of G and (V, F) is a DFS forest in $(V, E - S)$. Let $R = E - S - F$. Then the following hold:*

1) $(V, S \cup F)$ is $(i + 1)$-edge-connected.

2) Any cut in G with exactly $(i + 1)$ edges in $S \cup F$ contains exactly one edge of F and i edges of S.

3) Consider a cut $(X, \overline{X})$ *in* G *with exactly* $(i+1)$ *edges in* $S \cup F$. *Let* t *be the edge of* F *in this cut and* A_t *be the set of edges in* R *which are back edges of* t. *The set of all edges in the cut* $(X, \overline{X})$ *which are in* $E - S = F \cup R$ *is* $\{t\} \cup A_t$, *that is,* $(E - S) \cap (X, \overline{X}) = \{t\} \cup A_t$.

In Figure 15 (a), we show how a cut $(X, \overline{X})$ as in 3) looks. In Figure 15 (b), we present an example: a graph $G = (V, E)$ and edge sets S and F which satisfy the statement of Lemma 5.2.1 for $k = 3$ and $i = 2$: G is 3-edge-connected, (V, S) is 2-edge-connected and (V, F) is a DFS forest in $(V, E - S)$ (the numbering in Figure 15 (b) is a DFS numbering for F). The dashed edges are back edges with respect to F, in fact, all are back edges of the edge t indicated in Figure 15 (b). The cut $(X, \overline{X})$ shown is a cut for which 2) and 3) apply.

Proof of Lemma 5.2.1. Let us prove that there are at least $i+1$ edges of $S \cup F$ in any cut in G. Let $(X, \overline{X})$ be a cut in G. Since (V, S) is i-edge-connected, $(X, \overline{X})$ has at least i edges of S. If there are more than i edges of S in $(X, \overline{X})$, then there is nothing to prove. Assume there are exactly i edges of S in $(X, \overline{X})$. Because G is k-edge-connected and $i < k$, there is at least one edge of $E - S$ in $(X, \overline{X})$. But (V, F) is a maximum forest in $(V, E - S)$. (Any DFS forest is maximum.) This means that there must be an edge of F in $(X, \overline{X})$, and since this edge is in $E - S$, this implies that there are at least $i+1$ edges of $S \cup F$ in $(X, \overline{X})$, completing the proof of 1).

To prove 2), let $(X, \overline{X})$ be a cut in G with exactly $i+1$ edges in $S \cup F$. As above, $(X, \overline{X})$ must contain at least i edges in S. So either $(X, \overline{X})$ has exactly i edges in S and exactly one in F, or $(X, \overline{X})$ has exactly $i+1$ edges in S and none in F. In the first case, there is nothing to prove. Let us prove that the latter cannot happen. Suppose, for a contradiction, that $(X, \overline{X})$ has exactly $i+1$ edges in S and none in F. Because G is k-edge-connected

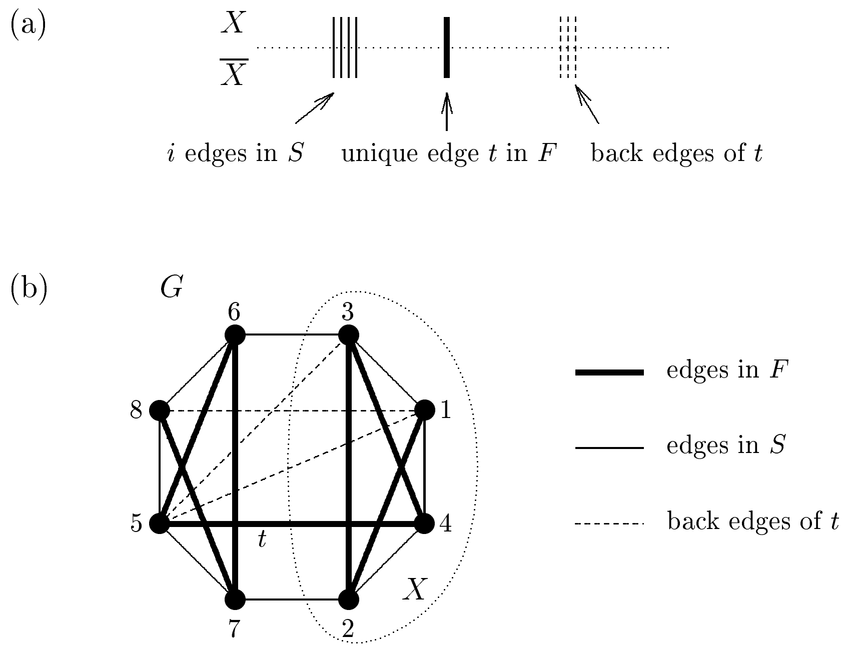

Figure 15: (a) Edges in a cut $(X, \overline{X})$ as in 3) (b) A 3-edge-connected graph G, edge sets S and F, and a cut $(X, \overline{X})$ as in 3). The numbering is a DFS numbering for F.

and $i+1<k$, there must be at least one edge of $E-S$ in $(X,\overline{X})$. But F has no edges in $(X,\overline{X})$, contradicting the fact that (V,F) is a maximum forest in $(V,E-S)$.

Now, let us prove 3). Let $(X,\overline{X})$ be a cut in G with exactly $i+1$ edges in $S\cup F$. Let t be the edge of F in $(X,\overline{X})$, and $x\in X, y\in\overline{X}$ be the endpoints of t. We first prove that $(E-S)\cap(X,\overline{X})\subseteq\{t\}\cup A_t$. Let e be an edge of $(X,\overline{X})$ in $E-S$. If $e=t$ then there is nothing to prove. Suppose $e\neq t$. Since t is the only edge of F in $(X,\overline{X}), e\in E-S-F$. Because F is a DFS forest in $(V,E-S)$, e must be a back edge with respect to F and its two endpoints must be in the same component of F. The path in F between the two endpoints of e plus e forms a cycle. Since any cycle intersects a cut an even number of times, and because $e\in(X,\overline{X})$, there must be at least one more edge of this cycle in $(X,\overline{X})$. But all other edges in the cycle are in F, and t is the only edge of F in $(X,\overline{X})$. Therefore, the path in F between the endpoints x and y of e must contain t, that is, e is a back edge of t and thus $e\in A_t$. This completes the proof that $(E-S)\cap(X,\overline{X})\subseteq\{t\}\cup A_t$.

Now let us prove the reverse inclusion. Clearly $t\in(E-S)\cap(X,\overline{X})$. Let $e\in A_t\subseteq E-S-F$. By the definition of A_t, the path P in F between the endpoints x and y of e contains t. The cycle formed by e and P intersects $(X,\overline{X})$ at least twice (it intersects $(X,\overline{X})$ at least once, because of t). But no edge in P other than t belongs to $(X,\overline{X})$. Hence e is the only other edge in the cycle which can belong to $(X,\overline{X})$, that is, $e\in(X,\overline{X})$, implying that $(E-S)\cap(X,\overline{X})\supseteq\{t\}\cup A_t$. ■

5.2.1 The Algorithm

The input for Khuller and Raghavachari's algorithm is a positive integer k and a k-edge-connected graph $G=(V,E)$. Their algorithm works in phases. It consists of $\lfloor k/2\rfloor$ phases, plus an extra final phase when k is odd. For now, let us assume that k is even. Later we comment on the final phase for odd k.

After phase i, the algorithm will have selected a set S_i of edges of G which induces a $2i$-edge-connected spanning subgraph of G. Hence at the end of $k/2$ iterations, the current set $S_{k/2}$ induces a k-edge-connected spanning subgraph of G which is the output of their algorithm.

How does each phase work? Let $S_0 := \emptyset$. In phase $i \geq 1$, the algorithm chooses two disjoint edge sets F_i and B_i, both disjoint from S_{i-1}. The set F_i is simply the edge set of a DFS forest in the graph $(V, E - S_{i-1})$. The edge set B_i is built as follows. Initially $B_i = \emptyset$. Then the edges in F_i are scanned in the reverse of the DFS order (i.e., in the reverse of the order they were included in F_i). When edge $t \in F_i$ is scanned, the algorithm finds a minimum cut between the endpoints of edge t in the graph $(V, S_{i-1} \cup F_i \cup B_i)$. Denoting by X one of the sides of this minimum cut, let $(X, \overline{X})$ be the corresponding cut in G, that is, $(X, \overline{X}) = \{h \in E : h \text{ has exactly one endpoint in } X\}$. If the cut $(X, \overline{X})$ in G contains exactly $2(i-1)+1 = 2i-1$ edges in $S_{i-1} \cup F_i \cup B_i$, then let $A := (E - (S_{i-1} \cup F_i \cup B_i)) \cap (X, \overline{X})$. The algorithm adds to B_i the edge $e \in A$ which has the smallest DFS-numbered top endpoint. Set $S_i := S_{i-1} \cup F_i \cup B_i$ at the end of phase i. This concludes the definition of the algorithm.

Consider B_i at the time $t \in F_i$ is scanned. Assume the algorithm found a cut $(X, \overline{X})$ separating the endpoints of t with exactly $2i-1$ edges in $S_{i-1} \cup F_i \cup B_i$. Let us prove that $A \neq \emptyset$ and $A = A_t$, where A_t is the set of edges of $E - S_{i-1} - F_i$ which are back edges of t with respect to F_i (this will guarantee the existence of edge e). Clearly $t \in (X, \overline{X})$ and $t \in F_i$. Since $(X, \overline{X})$ contains at least $2i-2$ edges of S_{i-1} (because S_{i-1} is $(2i-2)$-edge-connected) and $2i-1$ edges in $S_{i-1} \cup F_i \cup B_i$, $(X, \overline{X})$ must have zero edges in B_i (that is, $(X, \overline{X})$ has $2i-2$ edges in S_{i-1}, one edge (edge t) in F_i and no edges in B_i). By part 3) of Lemma 5.2.1, $(E - S_{i-1}) \cap (X, \overline{X}) = \{t\} \cup A_t$. Thus $A = (E - (S_{i-1} \cup F_i \cup B_i)) \cap (X, \overline{X}) = A_t$. Also, because $k > 2i-1$ and G is k-edge-connected, there must be some edge of $(X, \overline{X})$

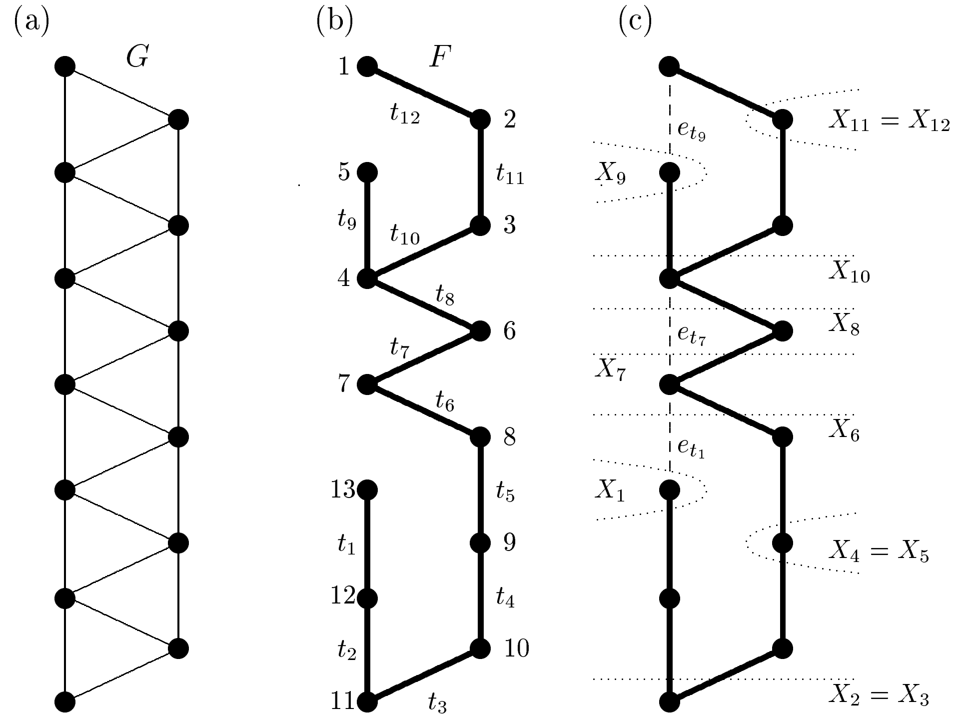

Figure 16: (a) A 2-edge-connected graph G. (b) A DFS forest F and the corresponding DFS numbering. (c) Possible cuts found by the algorithm if it chooses $F_1 = F$. The dashed edges are the edges inserted in B_1.

which is not in $S_{i-1} \cup F_i \cup B_i$, that is,

$$\emptyset \neq (X, \overline{X}) - (S_{i-1} \cup F_i \cup B_i) = ((E - S_{i-1}) \cap (X, \overline{X})) - \{t\} = A_t.$$

In Figure 16 (a), we have a 2-edge-connected graph G. In (b), the dark edges form a DFS tree F. The numbers on the vertices are a DFS numbering according to F. If we run the algorithm with G as input, it can choose $F_1 := E(F)$. After choosing F_1, the algorithm scans its edges in the order given in Figure 16 (b): $t_1, t_2, \ldots, t_{12}$. In (c), we show the cuts found by the algorithm and the edges $e_{t_1}, e_{t_7}, e_{t_9}$ inserted in B_1.

Now, let us present the analysis made by Khuller and Raghavachari, which will serve as background for our improvement. The next lemma was proved in [41] and is basically a corollary of Lemma 5.2.1. The correctness of the algorithm is a direct consequence of it.

Lemma 5.2.2 *For all i, $0 \le i \le k/2$, S_i is a $2i$-edge-connected spanning subgraph of G.*

Proof. By induction on i. For $i = 0$, the statement is trivially true. Let $i \ge 1$, and assume that S_{i-1} is $2(i-1)$-edge-connected. Let $(X, \overline{X})$ be a cut in G. We must prove that there are at least $2i$ edges of S_i in $(X, \overline{X})$. By 1) of Lemma 5.2.1, $(V, S_{i-1} \cup F_i)$ is $(2(i-1)+1)$-edge-connected. This means that there are at least $2(i-1)+1 = 2i-1$ edges of $S_{i-1} \cup F_i$ in $(X, \overline{X})$. Note that $S_{i-1} \cup F_i \subseteq S_i$. So, if there are at least $2i$ edges of $S_{i-1} \cup F_i$ in $(X, \overline{X})$, we are done. Suppose there are exactly $2i-1$ edges of $S_{i-1} \cup F_i$ in $(X, \overline{X})$. By 2) of Lemma 5.2.1, there is exactly one edge t of F_i in $(X, \overline{X})$ and exactly $2i-2$ edges in S_{i-1}. By 3) of Lemma 5.2.1, $(E - S_{i-1}) \cap (X, \overline{X}) = \{t\} \cup A_t$. Denote by B_i' the set B_i at the moment edge t was scanned by the algorithm. At that moment, either $S_{i-1} \cup F_i \cup B_i'$ had at least $2i$ edges in $(X, \overline{X})$, or it had exactly $2i-1$ edges in $(X, \overline{X})$ and the algorithm inserted into B_i a new edge $e \in A_t = A \subseteq E - (S_{i-1} \cup F_i \cup B_i')$. Since $e \in A_t$, we have $e \in (X, \overline{X})$. Therefore, $S_i = S_{i-1} \cup F_i \cup B_i$ contains at least $2i$ edges of $(X, \overline{X})$. ■

We need some extra notation for the analysis. Denote by OPT a k-edge-connected spanning subgraph of G with the minimum number of edges, and by *opt* the number of edges in OPT. Given an edge e in B_i, let t_e be the edge in F_i which caused e to be included in B_i. Let $(X_e, \overline{X_e})$ be a cut in $(V, S_{i-1} \cup F_i \cup B_i^e)$ of size $2i-1$ separating the endpoints of t_e, where B_i^e is B_i just before e was included in B_i by the algorithm. So $(X_e, \overline{X_e})$ contains t_e and at least $2i-2$ edges of S_{i-1} (because S_{i-1} is $(2i-2)$-edge-connected). But, $(X_e, \overline{X_e})$

has exactly $2i-1$ edges in $S_{i-1} \cup F_i \cup B_i^e$, implying that it has exactly $2i-2$ edges in S_{i-1} (one edge (edge t_e) in F_i and none in B_i^e).

Let P_e be the set of edges in $(X_e, \overline{X_e})$ which are in $E - S_{i-1}$, that is,

$$P_e = (E - S_{i-1}) \cap (X_e, \overline{X_e}) = \{t_e\} \cup A_{t_e}, \tag{14}$$

the last equality holding by 3) of Lemma 5.2.1. In Figure 17 (a) and (b), we show the same graph G of Figure 16, the forest F_1 and the set B_1 obtained by the algorithm in that example. In Figure 17 (c), we show the sets P_e for each edge e in B_1. Since $S_0 = \emptyset$, $P_e = (X_e, \overline{X_e})$.

Because OPT is k-edge-connected, OPT must contain at least k edges of $(X_e, \overline{X_e})$, and hence at least $k - |S_{i-1} \cap (X_e, \overline{X_e})| = k - 2(i-1)$ edges in P_e, that is,

$$|E(OPT\ ) \cap P_e| \geq k - 2(i-1), \text{ for all } e \text{ in } B_i. \tag{15}$$

Observe that this inequality indeed holds for the example in Figure 17 when OPT is the Hamiltonian cycle indicated by the dark edges in (a).

The next lemma states a key fact, proved in [41], essential for the analysis. (Note that it holds for the example in Figure 17.)

Lemma 5.2.3 *For any distinct edges e and f in B_i, the edge sets P_e and P_f are disjoint.*

Proof. Choose $e \neq f$, both in B_i; $e \neq f$ implies that $t_e \neq t_f$. By (14), $P_e = \{t_e\} \cup A_{t_e}$ and $P_f = \{t_f\} \cup A_{t_f}$. If $t_f \in P_e$, then P_e would contain more than one edge in F_i, contradicting 2) of Lemma 5.2.1. So $t_f \notin P_e$. Similarly, $t_e \notin P_f$, implying that $P_e \cap P_f = A_{t_e} \cap A_{t_f}$.

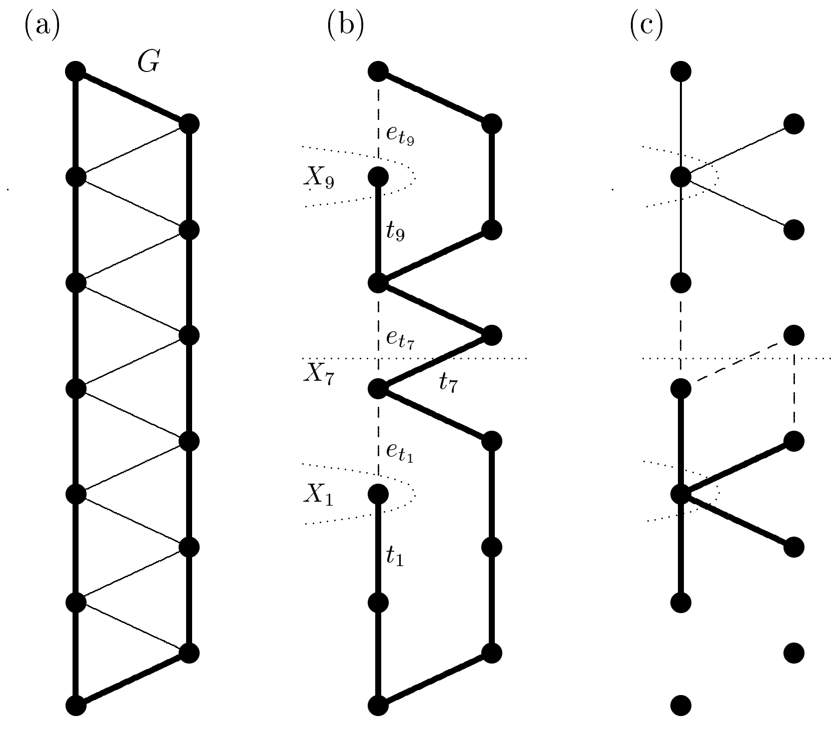

Figure 17: (a) Graph G and a 2-edge-connected subgraph of G with minimum number of edges indicated by the dark edges. (b) The DFS forest F_1 and the corresponding set $B_1 = \{e_{t_1}, e_{t_7}, e_{t_9}\}$. (c) The dark edges indicate set $P_{e_{t_1}}$, the dashed edges indicate $P_{e_{t_7}}$, and the solid edges indicate $P_{e_{t_9}}$.

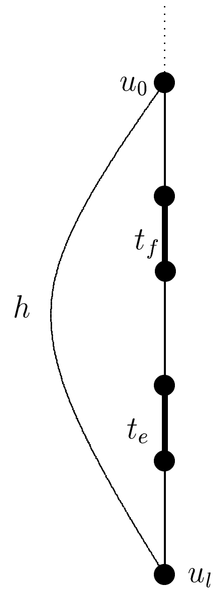

Figure 18: Edges t_e, t_f and $h \in P_e \cap P_f$.

Suppose, for a contradiction, that there is an edge h in $A_{t_e} \cap A_{t_f}$. By 3) of Lemma 5.2.1, because $h \in A_{t_e} \cap A_{t_f}$, t_e and t_f must lie on the path in F_i between the two endpoints of h. Let $P = u_0, u_1, \ldots, u_l$ be this path, where u_0 is the top endpoint of h and u_l is the bottom endpoint of h. Note that, because F_i is a DFS forest, the DFS numbers of $u_0, u_1, \ldots, u_l$ are in increasing order. We may assume, without loss of generality, that t_e was scanned before t_f (that is, the top endpoint of t_f has a smaller DFS number). See Figure 18.

Edge $e \in E - S_{i-1} - F_i$ was chosen as a back edge of t_e with smallest possible DFS-numbered top endpoint. Because h is a back edge of t_e and $h \in E - S_{i-1} - F_i$, the DFS number of the top endpoint of e is at least as small as the DFS number of the top endpoint of h (possibly $h = e$). Let us argue that when t_f was scanned, the algorithm would not add edge f to B_i.

Let B_i^f be B_i just before f was included in B_i by the algorithm. Note that $e \in B_i^f$ (because t_e was scanned before t_f). Consider the unique cycle C in $F_i \cup \{e\} \subseteq F_i \cup B_i^f$. This cycle C contains the subpath P' of P starting at u_0 and ending at the bottom endpoint of t_e. Edge t_f is in P', so C contains t_f. Let $(X, \overline{X})$ be any cut in G between the endpoints of t_f with the minimum number of edges in $S_{i-1} \cup F_i \cup B_i^f$. Because S_{i-1} is $(2i-2)$-edge-connected, $(X, \overline{X})$ has at least $2i-2$ edges in S_{i-1}. Also, $(X, \overline{X})$ has at least two edges in C (because $(X, \overline{X})$ has an even number of edges in any cycle, and t_f is in C and in $(X, \overline{X})$). Thus, since $C \subseteq F_i \cup B_i^f$, $(X, \overline{X})$ has at least $(2i-2)+2 = 2i$ edges in $S_{i-1} \cup F_i \cup B_i^f$, and the algorithm would not have added f into B_i, a contradiction (for edge f to be added into B_i, $(X, \overline{X})$ should have exactly $2i-1$ edges in $S_{i-1} \cup F_i \cup B_i^f$). ■

From (15) and Lemma 5.2.3, we get

$$(k - 2(i-1))|B_i| \le opt, \quad \text{for all } i, 1 \le i \le k/2. \tag{16}$$

We also have $opt \ge nk/2$, $|F_i| \le n$, and $|B_i| \le n$. From all this, Khuller and Raghavachari's upper bound on the performance ratio for even k follows:

$$\begin{aligned}
\frac{|S_{k/2}|}{opt} &= \frac{\sum_{i=1}^{k/2}(|F_i| + |B_i|)}{opt} \\
&= \frac{\sum_{i=1}^{k/2}|F_i| + \sum_{i=1}^{\lceil k/4\rceil}|B_i| + \sum_{i=\lceil k/4\rceil+1}^{k/2}|B_i|}{opt} \\
&\le \frac{\frac{k}{2}n}{\frac{kn}{2}} + \sum_{i=1}^{\lceil k/4\rceil}\frac{1}{k-2(i-1)} + \frac{\lfloor\frac{k}{4}\rfloor n}{\frac{kn}{2}} \\
&= 1 + \frac{\lfloor\frac{k}{4}\rfloor}{\frac{k}{2}} + \left(\sum_{i=1}^{\lceil k/4\rceil}\frac{1}{k-2(i-1)}\right)
\end{aligned}$$

$$\leq \frac{3}{2} + \frac{\ln 2}{2} \leq 1.85.$$

This completes the review of Khuller and Raghavachari's algorithm and analysis for even k. For odd k, the algorithm runs $\lfloor k/2 \rfloor$ phases, and in the end, it runs a "half" phase, computing $F_{\lceil k/2 \rceil}$ only (not $B_{\lceil k/2 \rceil}$). The final $S_{\lceil k/2 \rceil}$ is simply $S_{\lfloor k/2 \rfloor} \cup F_{\lceil k/2 \rceil}$. The analysis for odd k is similar, so we omit it.

For fixed k, the above analysis gives bounds smaller than 1.85. More specifically, it gives 1.5 for $k = 2$, 1.666... for $k = 3$, 1.75 for $k = 4$ and 1.733... for $k = 5$.

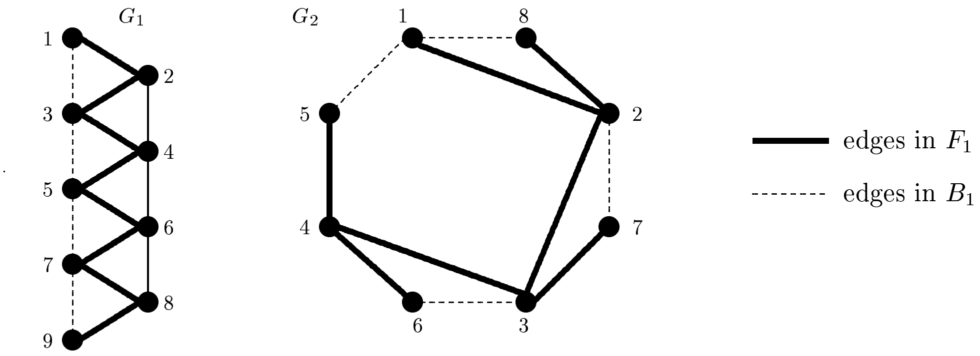

Figure 19: Graphs G_1 and G_2, with the DFS numbering corresponding to F_1.

These bounds are tight for $k = 2$ and 3. In Figure 19, we present our tight examples for $k = 2$. Our results originated from attempts of generalizing these examples for $k > 3$, and we think they can give some intuition. We have two essentially different types of examples for $k = 2$: G_1 and G_2. Denoting by n the number of vertices in each of these graphs, in both cases we have $opt = n$ (G_1 and G_2 are Hamiltonian). Also, if the algorithm chooses as F_1 the dark edges, then the set B_1 can have $\lfloor n/2 \rfloor$ edges in both cases, as shown in Figure 19. Therefore the output of the algorithm can have $n + \lfloor n/2 \rfloor$ edges (in both cases). Note that both these examples can be extended for large values of n, and as n grows, the ratio between the size of the output and *opt* approaches 1.5.

Graph G_3 is our tight example for $k = 3$. It is defined for all even $n \geq 10$. The set of edges of G_3 is the union of the sets of edges of four graphs: H, S_u, S_v and P. Graph H (Figure 20 (a)) is a cycle C on n vertices plus the chords from each vertex in C to the opposite vertex in C. Graph S_u is a star with a vertex u of C as center and spanning all other vertices of C (Figure 20 (b)). Graph S_v is similar to S_u, but its center is a vertex $v \neq u$ adjacent to a vertex w adjacent to u in the cycle C. Figure 20 (c) shows S_v and the vertices u, w and v in C. Graph P is a path of length $n/2$ starting at u and ending at v, containing alternate vertices of C. Figure 20 (d) indicates path P. Note that this can be extended to any even $n \geq 10$.

H is 3-edge-connected and has the minimum number of edges: $n + n/2$ edges. So, $opt = n + n/2$. When we run the algorithm on G_3, the first phase can choose a "bad" forest F_1: the path P plus the edges in S_v necessary to extend P to a tree. The solid edges in Figure 21 (a) represent F_1 ($n - 1$ edges). A DFS numbering for F_1 is indicated on the vertices. We can choose B_1 for this choice of F_1 to be the set of dashed edges in Figure 21 (a), and it has $n/2$ edges. The graph remaining after we remove the edges in F_1 and B_1 from G_3 is still connected (Figure 21 (b) shows a tree in this graph). Therefore, $|F_2| = n-1$, and the output of the algorithm has $2(n-1) + n/2$ edges. The ratio between this and *opt* approaches $5/3 = 1.666...$ as n grows.

The intuition that we would like to get from these examples comes from the following reasoning. For the previous bound to be tight, *opt* must be approximately $kn/2$, and the first $\lceil k/4 \rceil$ B_i must have approximately $opt/(k - 2(i-1))$ edges. As we will see, for large k, the only way to expect that $|B_i| \approx opt/(k - 2(i-1))$ for the initial B_i's is if F_i has many leaves. However, we will see that this cannot happen if $opt \approx kn/2$.

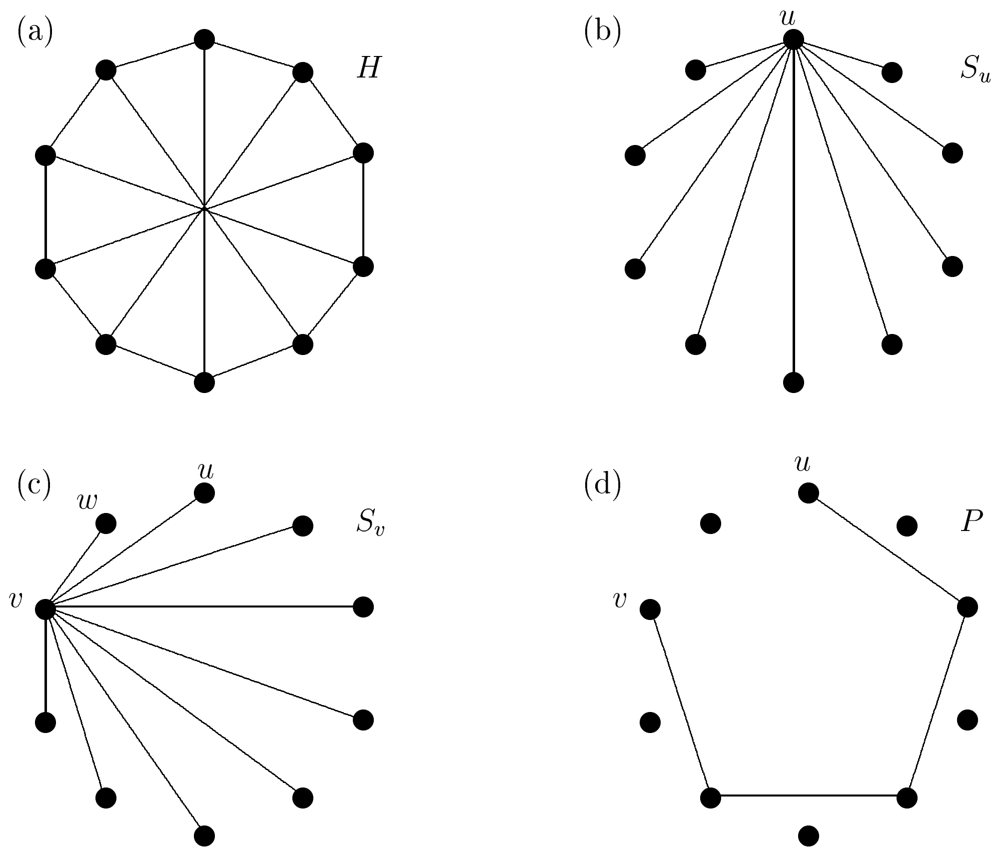

Figure 20: (a) Graph H: the outer cycle is called C. (b) Star S_u. (c) Star S_v and vertices u, w and v. (d) Path P.

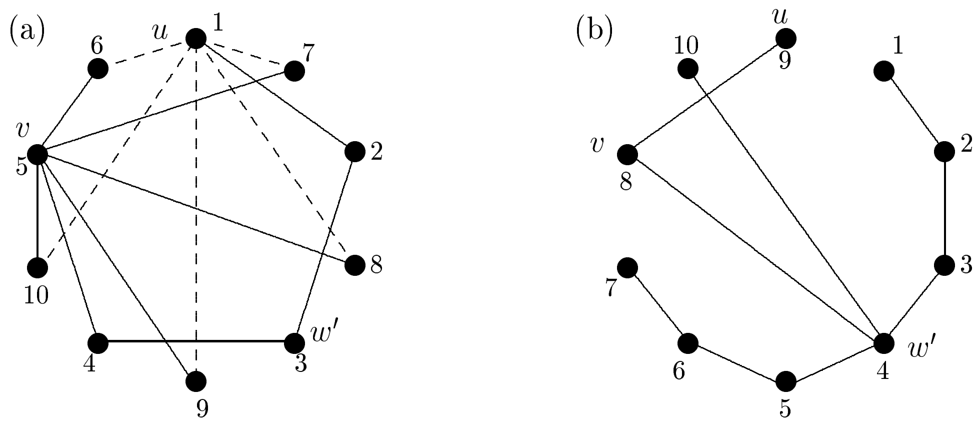

Figure 21: (a) The solid edges represent forest F_1 and the dashed edges, set B_1. (b) The forest F_2.

5.3 The 1.75 Bound on the Performance Ratio

In this section, we present a better analysis of Khuller and Raghavachari's algorithm and prove a new upper bound of 1.75 for all k on its performance ratio. We do this by proving a better upper bound on $|B_i|$.

Our analysis actually improves Khuller and Raghavachari's bound for any fixed $k \geq 4$. In particular, we get 1.65 for $k = 4$, and 1.68 for $k = 5$, the bound on $k = 5$ being tight.

For each e in B_i, recall that t_e is the edge in F_i which caused e to be included in B_i. Denote by b_e the bottom endpoint of t_e. (Note that $e \neq f$ implies that $t_e \neq t_f$ and therefore $b_e \neq b_f$.) For each i, $1 \leq i \leq \lfloor k/2 \rfloor$, let l_i be the number of b_e's which are leaves of F_i.

Our improvement is first based on the following stronger version of (16):

Lemma 5.3.1 $(k - 2(i-1))|B_i| + (i-1)l_i \leq \text{opt}$ *for all* i, $1 \leq i \leq \lfloor k/2 \rfloor$.

Proof. Call a pair (u, h) *special* if u is a vertex of G and h is an edge of OPT incident to u. The number of special pairs (u, h) is $2opt$: two for each edge h of OPT. Next, we compute a lower bound for the number of special pairs: we prove that the number of special pairs is at least $2(k-2(i-1))(|B_i|-l_i)+((k-2(i-1))+k)l_i = 2(k-2(i-1))|B_i|+2(i-1)l_i$. From this lower bound, the lemma follows trivially.

For the lower bound, we associate a set of special pairs with each edge e in B_i. If b_e is a leaf of F_i, then e will be associated with at least $(k-2(i-1))+k$ special pairs. If b_e is not a leaf, then e will be associated with at least $2(k-2(i-1)) \leq (k-2(i-1))+k$ special pairs. No special pair will be associated with two different edges in B_i.

By (15), OPT has at least $k-2(i-1)$ edges h in P_e, for each edge e in B_i. Suppose b_e is not a leaf of F_i. Associate with e two special pairs for each of these at least $k-2(i-1)$ edges h: (u, h) with u being the top endpoint of h and (u', h) with u' being the bottom endpoint of h. This scheme associates at least $2(k-2(i-1))$ special pairs with e.

Now, suppose b_e is a leaf of F_i. In a first phase, associate with e the special pairs (u, h) where h is an edge of OPT in P_e and u is the top endpoint of h. In this phase, at least $k-2(i-1)$ pairs are associated with e. Then, note that OPT has at least k edges incident to b_e (because OPT is k-edge-connected). Associate also with e the special pairs (b_e, h), for all h in OPT incident to b_e. In total, we have associated with e at least $(k-2(i-1))+k$ special pairs, because b_e is a leaf of F_i and thus is not the top endpoint of any edge in P_e.

Before showing that no two distinct edges e and f in B_i have an associated special pair in common, let us state a property of the pairs associated with an edge e in B_i.

Fact 5.3.2 *Any pair (u, h) associated with $e \in B_i$ is such that either $h \in P_e$ or $u = b_e$. Moreover, if $h \notin P_e$, then $u = b_e$ is a leaf of F_i.*

Proof. If b_e is not a leaf of F_i, this is clearly true. If b_e is a leaf of F_i, then in the first phase, all pairs (u, h) associated with e are such that $h \in P_e$. Besides these pairs, we associate with e pairs (u, h) with $u = b_e$. So any pair (u, h) associated with e is such that either $h \in P_e$ or $u = b_e$ is a leaf of F_i. ■

Now, let us show that no two distinct edges e and f in B_i have an associated special pair in common. Suppose, for a contradiction, that they do. Let (u, h) be such a pair. Since $b_e \neq b_f$ (because $e \neq f$), either $u \neq b_e$ or $u \neq b_f$. Without loss of generality, we may assume $u \neq b_e$. This means, by Fact 5.3.2, that $h \in P_e$. As was said in Lemma 5.2.3, the sets P_e and P_f are disjoint, so $h \notin P_f$. But then, again by Fact 5.3.2, b_f is a leaf of F_i and $u = b_f$. Because b_f is a leaf of F_i, any edge l incident to b_f is either (1) in S_{i-1} or (2) goes to an ancestor of b_f in F_i (because F_i is a DFS forest in $(V, E - S_{i-1})$) so that l is in $\{t_f\} \cup A_{t_f} = P_f$. So, since $u = b_f$ (so that h is incident to b_f) and $h \notin P_f$, we must be in case (1), contradicting the fact that $h \in P_e \subseteq E - S_{i-1}$. Thus, no two distinct edges e and f in B_i have an associated pair in common.

Therefore, we must have at least $2(k - 2(i-1))(|B_i| - l_i) + ((k - 2(i-1)) + k)l_i = 2(k - 2(i-1))|B_i| + 2(i-1)l_i$ special pairs. Joining this to the fact that there are exactly $2opt$ special pairs, we conclude the proof of Lemma 5.3.1. ■

We will present a lower bound on l_i, for $1 \leq i \leq \lfloor k/2 \rfloor$, which will imply an improved upper bound on $|B_i|$. Let us fix i such that $1 \leq i \leq \lfloor k/2 \rfloor$. For such an i, by (15), $|P_e| \geq k - 2(i-1) \geq 2$ for any e in B_i. Denote by B the set $\{b_e : e \in B_i\}$. The following lemma will be used in Lemma 5.3.4.

Lemma 5.3.3 *If $e, f \in B_i$ are such that b_f is a child in F_i of b_e, then the following hold.*

1) Any edge $h \in A_{t_f}$ has b_e as top endpoint and a proper descendant of b_f as bottom endpoint.

2) For any edge $h \in A_{t_f}$, consider the path in F_i from b_f to the bottom endpoint of h.

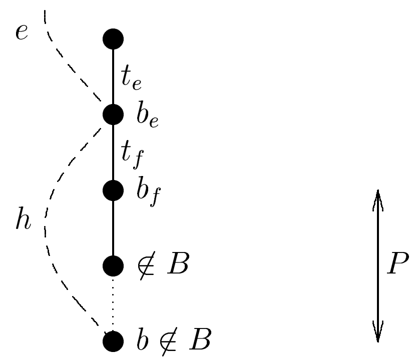

Figure 22: Edges e and f in B_i such that b_f is a child of b_e.

Vertex b_f is the only vertex of this path which is in B.

3) b_f has at least one child which is not in B.

Proof. Consider $e, f \in B_i$ such that b_f is a child of b_e, as in Figure 22. Let $h \in A_{t_f}$.

Let us prove 1). Since h is a back edge of t_f, $h \in P_f$ and has as top endpoint an ancestor of b_e (which is the top endpoint of t_f). (We consider a vertex to be an ancestor of itself.) If the top endpoint of h were not b_e, h would be a back edge of t_e as well, that is, $h \in A_{t_e} \subseteq P_e$, contradicting the fact that $P_e \cap P_f = \emptyset$. Therefore b_e is the top endpoint of h. Now, let us prove that h has as bottom endpoint a proper descendant of b_f. The bottom endpoint of h must be a descendant of b_f. Since t_f has b_e and b_f as endpoints and $h \neq t_f$ (because h is a back edge of t_f), h cannot have b_f as bottom endpoint, that is, h has as bottom endpoint a proper descendant of b_f. This completes the proof of 1).

For 2), let b be the bottom endpoint of h and P be the path in F_i from b_f to b. By 1), b is a proper descendant of b_f. Thus, for all $u \neq b_f$ in P, the edge whose bottom endpoint is u is in P. Let u be a vertex other than b_f in P. We want to prove that $u \notin B$. Suppose, for a contradiction, that $u \in B$, that is, $u = b_g$ for some edge $g \in B_i$. Thus t_g is in P, implying that $h \in A_{t_g}$. But then $h \in A_{t_g} \cap A_{t_f} \subseteq P_g \cap P_f$. Because t_f is not in P, $t_g \neq t_f$ and hence $g \neq f$, contradicting the fact that $P_g \cap P_f \neq \emptyset$.

Now, let us prove 3). Note that $f \in A_{t_f}$. Thus, by 1), f has as bottom endpoint a proper descendant b of b_f. Let u be the child of b_f on the path in T_i from b_f to b. By 2), $u \notin B$, completing the proof of 3). ■

The lower bound on l_i is given by the following lemma.

Lemma 5.3.4 *$l_i \geq 2|B_i| - n$, for any i, $1 \leq i < k/2$, and for even k, $l_{k/2} \geq 3|B_{k/2}| - 2n$.*

Proof. First, let us consider i such that $1 \leq i < k/2$. The case where $i = k/2$ for even k is treated later. Denote by N the set of b_e's in B which are not leaves in F_i. We will associate with each b_e in N a distinct vertex z_e in $V - B$. The association is made in two phases (Phase 1 executed before Phase 2):

Phase 1. For each b_e in N which has a child in $V - B$:

Let z_e be a child of b_e which is in $V - B$.

Phase 2. For each b_e in N whose children are all in B:

Let $f \in B_i$ be such that b_f is one of b_e's children. By 3) of Lemma 5.3.3, b_f has a child which is not in B. Hence b_f is in N. So b_f satisfies the statement of Phase 1, which means that z_f was assigned in Phase 1.

Case 1: b_f has two or more children which are not in B.

Let z_e be any child of b_f which is not in B and is distinct from z_f (which was assigned previously in Phase 1).

Case 2: b_f has exactly one child which is not in B.

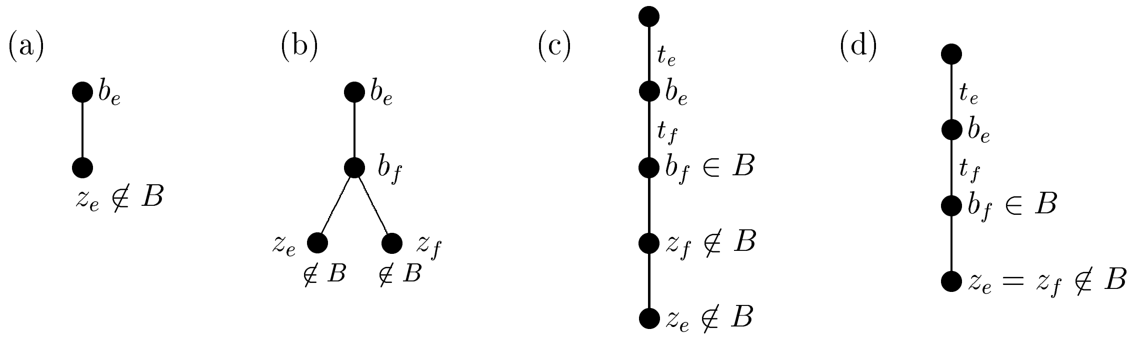

Figure 23: (a) Situation in Phase 1. (b) Situation in Phase 2, Case 1. (c) Situation in Phase 2, Case 2. (d) Situation in the new Phase 2, Case 2 (used later).

Because z_f was assigned previously in Phase 1, and b_f has only one child which is not in B, this child is in fact z_f. Let z_e be a child of z_f which is not in B. (Later we show that z_f has at least one child which is not in B.)

The three different situations are depicted in Figure 23. Note that z_e is always either a child, or a grandchild, or a great-grandchild of b_e. Moreover, z_e is a child of b_e if and only if z_e was assigned in Phase 1; z_e is a grandchild of b_e if and only if z_e was assigned in Phase 2, Case 1; and z_e is a great-grandchild of b_e if and only if z_e was assigned in Phase 2, Case 2.

Let us prove that in Phase 2, Case 2, z_f in fact has at least one child z which is not in B. By 1) of Lemma 5.3.3, because z_f is the only child of b_f which not in B, all the edges in A_{t_f} have as bottom endpoint distinct descendants of z_f. Equation (15) and $i \leq (k-1)/2$ imply that $|P_f| \geq k - 2(i-1) \geq 3$, that is, $|A_{t_f}| \geq 2$. So one of these at least two edges in A_{t_f}, say edge g, must have a bottom endpoint distinct from z_f. Let z be the child of z_f on the path from z_f to the bottom endpoint of g. By 2) of Lemma 5.3.3, $z \notin B$.

Now, let us verify that if $b_e \neq b_g$ then $z_e \neq z_g$. If both z_e and z_g were assigned in

Phase 1, then b_e is the parent of z_e and b_g is the parent of z_g. Because $b_e \neq b_g$, we have $z_e \neq z_g$. If both z_e and z_g were both assigned in Phase 2, Case 1, the same argument goes through with grandparent in place of parent. If z_e and z_g were assigned in Phase 2, Case 2, the same goes through again with great-grandparent in place of parent.

Suppose z_g was assigned in Phase 1 and z_e was assigned in Phase 2. Let f be as in Phase 2 when z_e was assigned. Because z_g was assigned in Phase 1, b_g is the parent of z_g. If z_e was assigned in Phase 2, Case 1, then z_e is a child of b_f which is not in B and is distinct from z_f, and b_e is the grandparent of z_e. The only way to have $z_e = z_g$ is if their parents are the same, that is, $b_f = b_g$. But this implies that $f = g$ and then $z_f = z_g = z_e$, a contradiction. If z_e was assigned in Phase 2, Case 2, then b_e is the parent of b_f, b_f is the parent of $z_f \notin B$ and z_f is the parent of z_e. For $z_e = z_g$, we must have $b_g = z_f$ (because b_g is the parent of z_g and z_f is the parent of z_e). But this contradicts $z_f \notin B$ (since $b_g \in B$).

Last, suppose z_g was assigned in Phase 2, Case 1 and z_e was assigned in Phase 2, Case 2. Let f be as in Phase 2 when z_e was assigned. Recall that z_f was assigned in Phase 1. Because z_g was assigned in Phase 2, Case 1, b_g is the grandparent of z_g. Because z_e was assigned in Phase 2, Case 2, b_e is the parent of b_f, b_f is the parent of $z_f \notin B$ and z_f is the parent of z_e. We only have $z_e = z_g$ if $b_g = b_f$ (because b_g is the grandparent of z_g and b_f is the grandparent of z_e). But this means $f = g$, which contradicts the fact that $z_f = z_g$ was assigned in Phase 1 (since z_g, by assumption, was assigned in Phase 2, Case 1). This completes the proof that if $b_e \neq b_g$ then $z_e \neq z_g$, that is, one can assign a different vertex in $V - B$ to each vertex in N.

Denoting by Z the set $\{z_e : e \in B_i \text{ and } b_e \in N\}$, we have $|Z| = |N|$. Also, $|B| = |B_i|$, $N \subseteq B$ and $Z \subseteq V - B$. Thus $l_i = |B - N| = |B| - |N| = |B| - |Z| \geq |B| - |V - B| = |B| - (n - |B|) = 2|B| - n = 2|B_i| - n$, completing the proof of the first part of Lemma 5.3.4.

The proof of that $l_{k/2} \geq 3|B_{k/2}| - 2n$ differs from the above proof only at Phase 2, Case 2. We still execute Phase 1 and Phase 2, but in Phase 2, Case 2, we simply select $z_e = z_f$. (See Figure 23 (d).) We do this because for $i = k/2$ (even k), we just have $|P_f| \geq 2$, and thus z_f might not have a child to be selected as z_e.

With this change, we can prove that, given z, $z = z_e$ for at most two distinct $b_e \in N$. Consider $b_e, b_g \in N$ such that $b_e \neq b_g$ but $z_e = z_g$. Since Phase 1 and Phase 2, Case 1 are the same, we know that if z_e and z_g were assigned in Phase 1 or Phase 2, Case 1, then $z_e \neq z_g$. So either z_e or z_g was assigned in (the new) Phase 2, Case 2. We may assume, without loss of generality, that z_e was assigned in Phase 2, Case 2. Suppose there is a $b_h \neq b_g$, with $z_h = z_e = z_g$. We want to prove that $b_h = b_e$. Again, since Phase 1 and Phase 2, Case 1 are the same, if z_g and z_h were assigned in Phase 1 or Phase 2, Case 1, then $z_g \neq z_h$. So either z_g or z_h was assigned in (the new) Phase 2, Case 2. If both z_e and z_g were assigned in (the new) Phase 2, Case 2, then b_e is the grandparent of z_e and b_g is the grandparent of z_g. Because $b_e \neq b_g$, we have $z_e \neq z_g$, a contradiction. So z_e and z_h were assigned in Phase 2, Case 2 and, since $z_e = z_h$, the same reasoning shows that $b_e = b_h$, as we wanted to show. Therefore, $z = z_e$ for at most two distinct $b_e \in N$.

Again, let $Z = \{z_e : e \in B_i \text{ and } b_e \in N\} \subseteq V - B$. From the previous paragraph, we conclude that $|N| \leq 2|Z|$. And, thus, $l_{k/2} = |B| - |N| \geq |B| - 2|Z| \geq |B| - 2(n - |B|) = 3|B| - 2n = 3|B_{k/2}| - 2n$, completing the proof of the lemma. ■

Next we present a new upper bound on $|B_i|$.

Corollary 5.3.5

$$
\begin{aligned}
|B_i| &\leq \left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right) opt, \quad \textit{for any } i \textit{ such that } 1 \leq i < k/2. \\
|B_{k/2}| &\leq \frac{2(3k-4)}{k(3k-2)}\, opt.
\end{aligned}
$$

Proof. Consider $1 \le i < k/2$. From Lemma 5.3.1,

$$\begin{aligned} opt &\ge (k-2(i-1))|B_i| + (i-1)l_i \\ &\ge (k-2(i-1))|B_i| + (i-1)(2|B_i|-n), \text{ by Lemma 5.3.4} \\ &= k|B_i| - n(i-1) \\ &\ge k|B_i| - \frac{2opt(i-1)}{k}, \text{ because } opt \ge nk/2. \end{aligned}$$

Therefore

$$|B_i| \le \frac{1}{k}\left(1+\frac{2(i-1)}{k}\right) opt = \left(\frac{1}{k}+\frac{2(i-1)}{k^2}\right) opt.$$

The case $i = k/2$ is similar, so we omit it. ■

This bound is not the best we can get using this technique, as we will show in the next section. But it implies the better upper bound of 1.75 on the performance ratio of the algorithm, as stated below.

Theorem 5.3.6 *The performance ratio of Khuller and Raghavachari's algorithm is less than 1.75 for all k. More specifically, it is at most*

$$1.75 - \frac{3k^2-6k+8}{2k^2(3k-2)}, \textit{ for even } k, \textit{ and } 1.75 - \frac{2k-3}{4k^2}, \textit{ for odd } k > 1.$$

Proof. For $k = 1$, the algorithm is optimal, i.e., the ratio is one. For odd $k > 1$,

$$\frac{|S_{\frac{k+1}{2}}|}{opt} = \frac{\sum_{i=1}^{(k+1)/2}|F_i| + \sum_{i=1}^{(k-1)/2}|B_i|}{opt}$$

$$
\begin{aligned}
&\leq \frac{\frac{k+1}{2}n}{\frac{kn}{2}} + \sum_{i=1}^{(k-1)/2}\left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right) = \left(1 + \frac{1}{k}\right) + \frac{k-1}{2k} + \frac{2}{k^2}\sum_{i=1}^{(k-1)/2}(i-1) \\
&= \frac{3}{2} + \frac{1}{2k} + \frac{(k-1)(k-3)}{4k^2} = \frac{3}{2} + \frac{k^2-2k+3}{4k^2} = \frac{7}{4} - \frac{2k-3}{4k^2}.
\end{aligned}
$$

Observe that $2k - 3 > 0$ for all $k \geq 3$ and $4k^2 > 0$ for all positive k.

The accounting is similar for even k.

$$
\begin{aligned}
\frac{|S_{k/2}|}{opt} &= \frac{\sum_{i=1}^{k/2}|F_i| + \sum_{i=1}^{k/2}|B_i|}{opt} \leq \frac{\frac{k}{2}n}{\frac{kn}{2}} + \sum_{i=1}^{k/2-1}\left(\frac{1}{k} + \frac{2(i-1)}{k^2}\right) + \frac{2(3k-4)}{k(3k-2)} \\
&= 1 + \left(\frac{1}{2} - \frac{1}{k}\right) + \frac{2}{k^2}\sum_{i=1}^{k/2-1}(i-1) + \frac{2(3k-4)}{k(3k-2)} \\
&= \frac{3}{2} + \frac{k^2-10k+8}{4k^2} + \frac{2(3k-4)}{k(3k-2)} \\
&= \frac{3}{2} + \frac{1}{4} - \frac{5k-4}{2k^2} + \frac{2(3k-4)}{k(3k-2)} = \frac{7}{4} - \frac{3k^2-6k+8}{2k^2(3k-2)}.
\end{aligned}
$$

In this case, $3k^2 - 6k + 8 > 0$ always, and $2k^2(3k-2) > 0$ for all positive k. Hence the ratio is in fact always smaller than 1.75 for all values of k. ∎

5.3.1 Analysis for Small Values of k

The analysis presented above results in upper bounds for Khuller and Raghavachari's algorithm better than 1.75 for small values of k. For example, by plugging $k = 4$ and $k = 5$ into the bounds given by Theorem 5.3.6, one gets 1.65 for $k = 4$ and 1.68 for $k = 5$. For these cases, the previous bounds given in [41] are 1.75 and 1.733..., respectively.

Let us present examples that show a lower bound on the performance ratio of 1.625 for

$k = 4$ and 1.68 for $k = 5$, meaning that our bound is tight for $k = 5$. The examples are a generalization of our tight examples for $k = 2$ and $k = 3$ presented as G_2 and G_3 on page 84.

Graph G_4 is our example for $k = 4$. Denote by $v_0, v_1, \ldots, v_{n-1}$ its vertices. G_4 is defined only when n is a multiple of 16. The edge set of G_4 is the union of four edge sets (subscripts are taken module n):

$$
\begin{aligned}
H &= \{v_i v_{i+1} : 0 \leq i < n\} \cup \{v_i v_{i+3} : 0 \leq i < n\} \\
P_1 &= \{v_i v_{i+8} : i \equiv 0 \bmod 8,\ i < n - 8\} \cup \{v_{n-8} v_2\} \\
&\quad \cup \{v_i v_{i+4} : i \equiv 2 \bmod 4,\ i < n - 2\} \cup \{v_{n-2} v_4\} \\
&\quad \cup \{v_i v_{i+8} : i \equiv 4 \bmod 8,\ i < n - 4\} \\
P_2 &= \{v_i v_{i+8} : i \equiv 6 \bmod 8,\ i < n - 2\} \cup \{v_{n-2} v_2\} \\
&\quad \cup \{v_i v_{i+8} : i \equiv 2 \bmod 8,\ i < n - 6\} \\
R &= \{v_i v_{i+2} : i \equiv 0 \bmod 2\}.
\end{aligned}
$$

Figures 24, 25, 26 and 27 show the sets H, P_1, P_2 and R when $n = 16$.

Note that the set of vertices of G_4 with odd subscript is an independent set. In a first phase, the algorithm can choose

$$F_1 = P_1 \cup \{v_i v_{i-3}, v_i v_{i-1}, v_i v_{i+1}, v_i v_{i+3} : i \equiv 4 \bmod 8\} \quad \text{and}$$

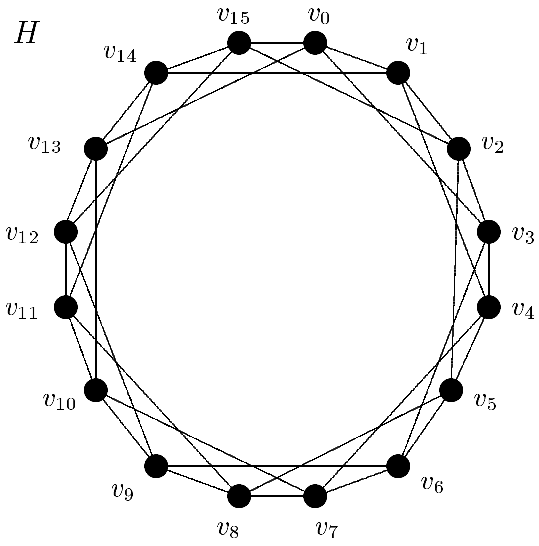

Figure 24: Edge set H for G_4 when $n = 16$.

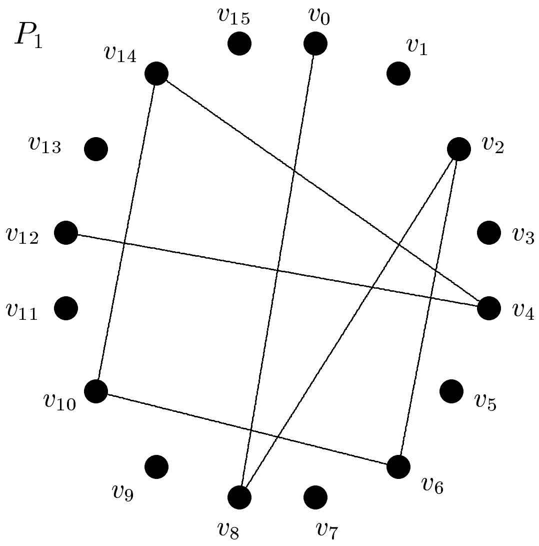

Figure 25: Edge set P_1 for G_4 when $n = 16$.

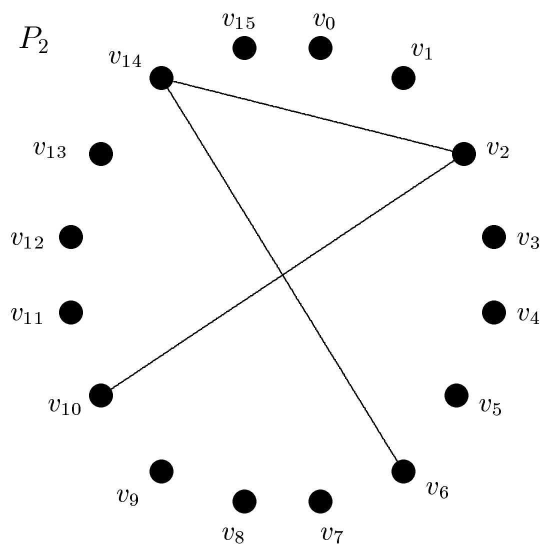

Figure 26: Edge set P_2 for G_4 when $n = 16$.

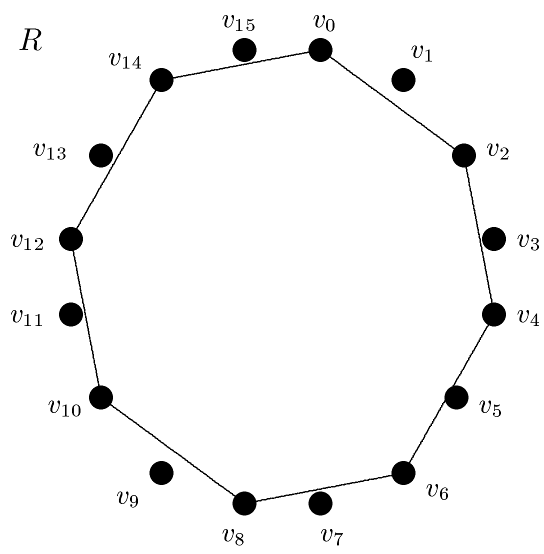

Figure 27: Edge set R for G_4 when $n = 16$.

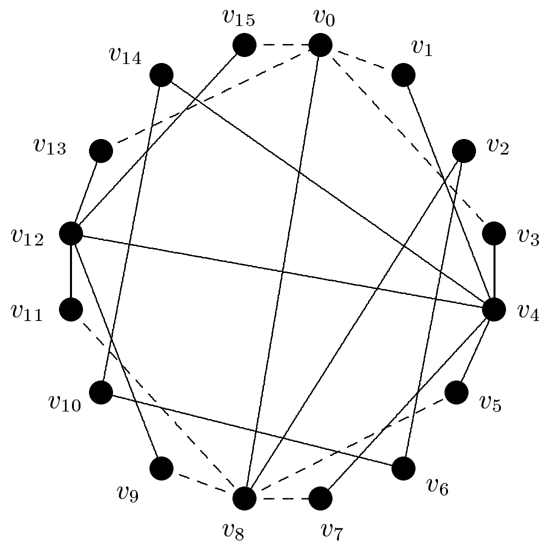

Figure 28: The solid edges are the edges in F_1, while the dashed edges are the edges in B_1.

$$B_1 \quad = \quad \{v_iv_{i-3}, v_iv_{i-1}, v_iv_{i+1}, v_iv_{i+3} : i \equiv 0 \bmod 8\}.$$

Note that F_1 is in fact a DFS forest: F_1 visits all even subscripted vertices, choosing the edges in P_1, and then the odd subscripted vertices all become leaves of F_1. In Figure 28, we show F_1 when $n = 16$.

In the second phase, the algorithm can choose

$$F_2 \quad = \quad P_2 \cup \{v_iv_{i-3}, v_iv_{i-2}, v_iv_{i-1}, v_iv_{i+1}, v_iv_{i+2}, v_iv_{i+3} : i \equiv 6 \bmod 8\} \quad \text{and}$$

$$B_2 \quad = \quad \{v_iv_{i-3}, v_iv_{i-2}, v_iv_{i-1}, v_iv_{i+1}, v_iv_{i+2}, v_iv_{i+3} : i \equiv 2 \bmod 8\}.$$

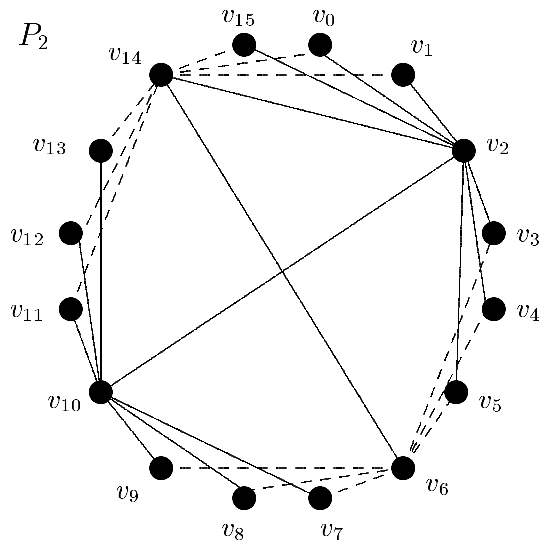

Figure 29: The solid edges are the edges in F_2, while the dashed edges are the edges in B_2.

Again F_2 is a DFS forest: F_2 chooses the edges in P_2 first, then all other vertices become leaves of F_2. In Figure 29, we show F_2 when $n = 16$.

We have $|F_1| = |F_2| = n - 1, |B_1| = n/2$ and $|B_2| = n/2 + n/4$. This implies that the output of the algorithm for these choices of F_1 and F_2 is

$$2(n-1) + n + \frac{n}{4} = \frac{13}{4}n - 2.$$

On the other hand, H is the edge set of a 4-edge-connected graph, and it is a minimum set with this property. So, $opt = |H| = 2n$. Therefore the performance ratio of the algorithm is at least

$$\frac{13}{8} - \frac{1}{n} = 1.625 - \frac{1}{n},$$

which approaches 1.625 as n grows. This gives a lower bound of 1.625 on the performance ratio of Khuller and Raghavachari's algorithm for $k = 4$.

This example can be generalized for $k = 5$. Let us describe G_5, a graph on which the algorithm can achieve a ratio for $k = 5$ as close to 1.68 as we want. Denote the vertices of G_5 by $v_0, v_1, \ldots, v_{n-1}$. Graph G_5 is defined only when n is a multiple of 20. The edge set of G_5 is partitioned into five edge sets (subscripts are taken module n):

$$
\begin{aligned}
H &= \{v_i v_{i+1} : 0 \le i < n\} \cup \{v_i v_{i+3} : 0 \le i < n\} \cup \{v_i v_{i+5} : i \equiv 1 \bmod 2\} \\
P_1 &= \{v_i v_{i+10} : i \equiv 0 \bmod 10,\ i < n-10\} \cup \{v_{n-10} v_8\} \\
&\quad \cup \{v_{i+8} v_{i+6}, v_{i+6} v_{i+12}, v_{i+12} v_{i+16}, v_{i+16} v_{i+2}, v_{i+2} v_{i+18} : i \equiv 0 \bmod 20\} \\
&\quad \cup \{v_{i+18} v_{i+28} : i \equiv 0 \bmod 20,\ i + 28 < n\} \cup \{v_{n-2} v_4\} \\
&\quad \cup \{v_i v_{i+10} : i \equiv 4 \bmod 10,\ i < n-6\} \\
P_2 &= \{v_i v_{i+10} : i \equiv 2 \bmod 10, i < n-8\} \cup \{v_{n-8} v_8\} \\
&\quad \cup \{v_i v_{i+10} : i \equiv 8 \bmod 10,\ i < n-2\} \cup \{v_{n-2} v_{16}\} \\
&\quad \cup \{v_i v_{i+10} : i \equiv 6 \bmod 20,\ i < n-2\} \cup \{v_i v_{i+36} : i \equiv 6 \bmod 20,\ i + 36 < n\} \\
P_3 &= \{v_i v_{i-8}, v_i v_{i-6}, v_i v_{i-4}, v_i v_{i+8} : i \equiv 8 \bmod 20\} \\
&\quad \cup \{v_i v_{i-12}, v_i v_{i-10}, v_i v_{i-6}, v_i v_{i-4} : i \equiv 18 \bmod 20\} \cup \{v_0 v_{18}\} \\
&\quad \cup \{v_i v_{i+20} : i \equiv 0 \bmod 20,\ i + 20 < n\} \cup \{v_i v_{i+20} : i \equiv 18 \bmod 20,\ i + 20 < n\}
\end{aligned}
$$

$$R = \{v_{i+2}v_i, v_{i+2}v_{i+4} : i \equiv 0 \bmod 10\} \cup \{v_{i+6}v_{i+4}, v_{i+6}v_{i+10} : i \equiv 0 \bmod 10\}.$$

Figures 30, 31 and 32 show the sets H, P_1, P_2, P_3 and R for $n = 20$.

Note that the set of vertices of G_5 with odd subscript is an independent set. The algorithm can choose

$$F_1 = P_1 \cup \{v_iv_{i-5}, v_iv_{i-3}, v_iv_{i-1}, v_iv_{i+1}, v_iv_{i+3} : i \equiv 4 \bmod 10\}$$

$$B_1 = \{v_iv_{i-5}, v_iv_{i-3}, v_iv_{i-1}, v_iv_{i+1}, v_iv_{i+3} : i \equiv 0 \bmod 10\}$$

$$F_2 = P_2 \cup \{v_iv_{i-5}, v_iv_{i-4}, v_iv_{i-3}, v_iv_{i-2}, v_iv_{i-1}, v_iv_{i+1}, v_iv_{i+2}, v_iv_{i+3} : i \equiv 6 \bmod 10\}$$

$$B_2 = \{v_iv_{i-5}, v_iv_{i-4}, v_iv_{i-3}, v_iv_{i-2}, v_iv_{i-1}, v_iv_{i+1}, v_iv_{i+2}, v_iv_{i+3} : i \equiv 2 \bmod 10\}$$

$$F_3 = P_3 \cup \{v_iv_{i-5}, v_iv_{i-3}, v_iv_{i-1}, v_iv_{i+1}, v_iv_{i+3} : i \equiv 8 \bmod 10\}$$

One can verify that F_1, F_2 and F_3 are DFS-forests. Since $|F_1| = |F_2| = |F_3| = n - 1, |B_1| = n/2$ and $|B_2| = n/2 + n/5$, the output of the algorithm for these choices of F_1 and F_2 is

$$3(n-1) + n + \frac{n}{5} = \frac{21}{5}n - 3.$$

H is the edge set of a 5-edge-connected graph, and it is a minimum set with this property. So, $opt = |H| = 5n/2$. Therefore the performance ratio of the algorithm is at least

$$\frac{42}{25} - \frac{6}{5n} = 1.68 - \frac{6}{5n},$$

which approaches 1.68 as n grows. This gives a lower bound of 1.68 on the performance

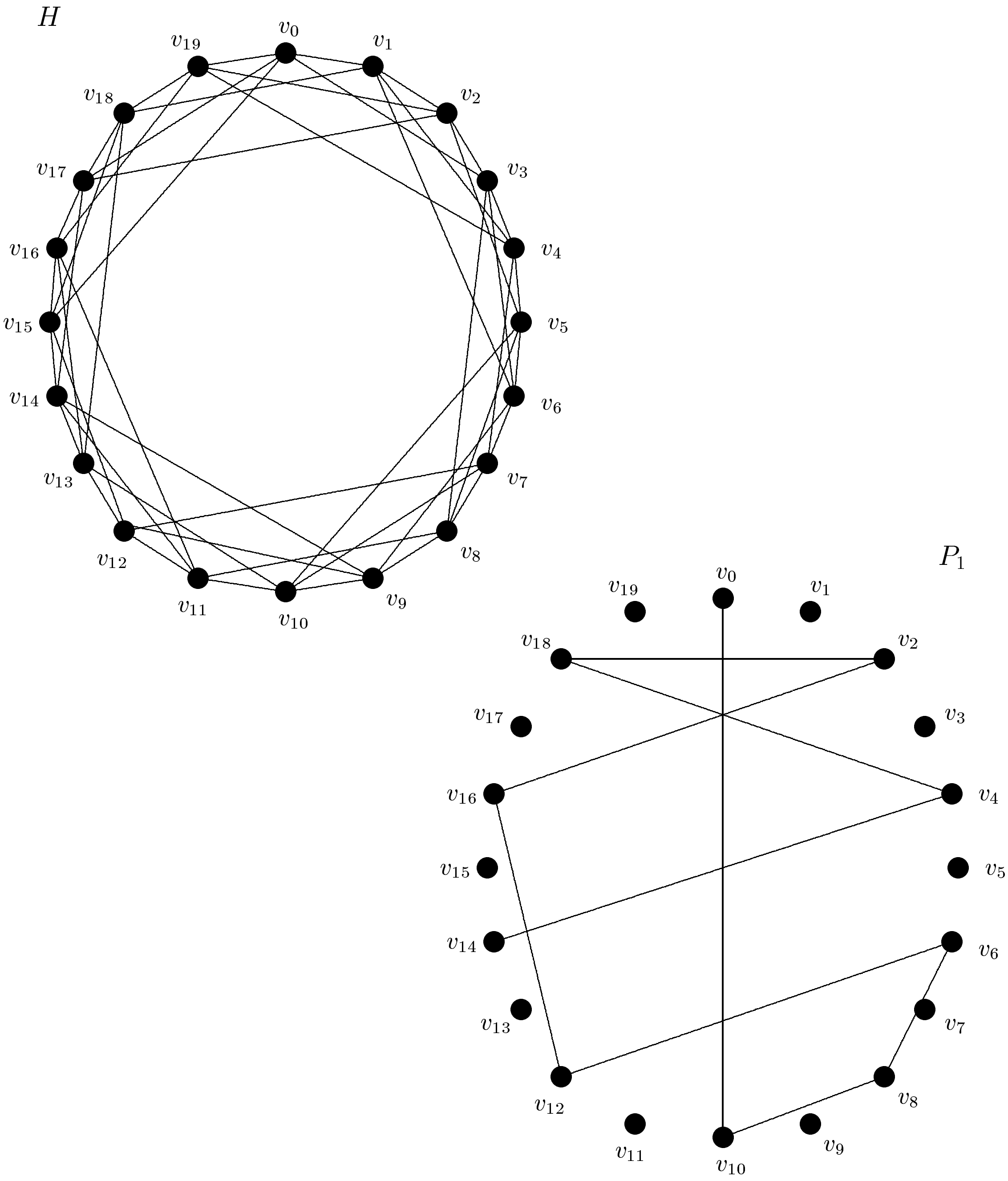

Figure 30: Edge sets H and P_1 for G_5 when $n = 20$.

ratio of Khuller and Raghavachari's algorithm for $k = 5$, proving that the bound given in Theorem 5.3.6 is tight for $k = 5$.

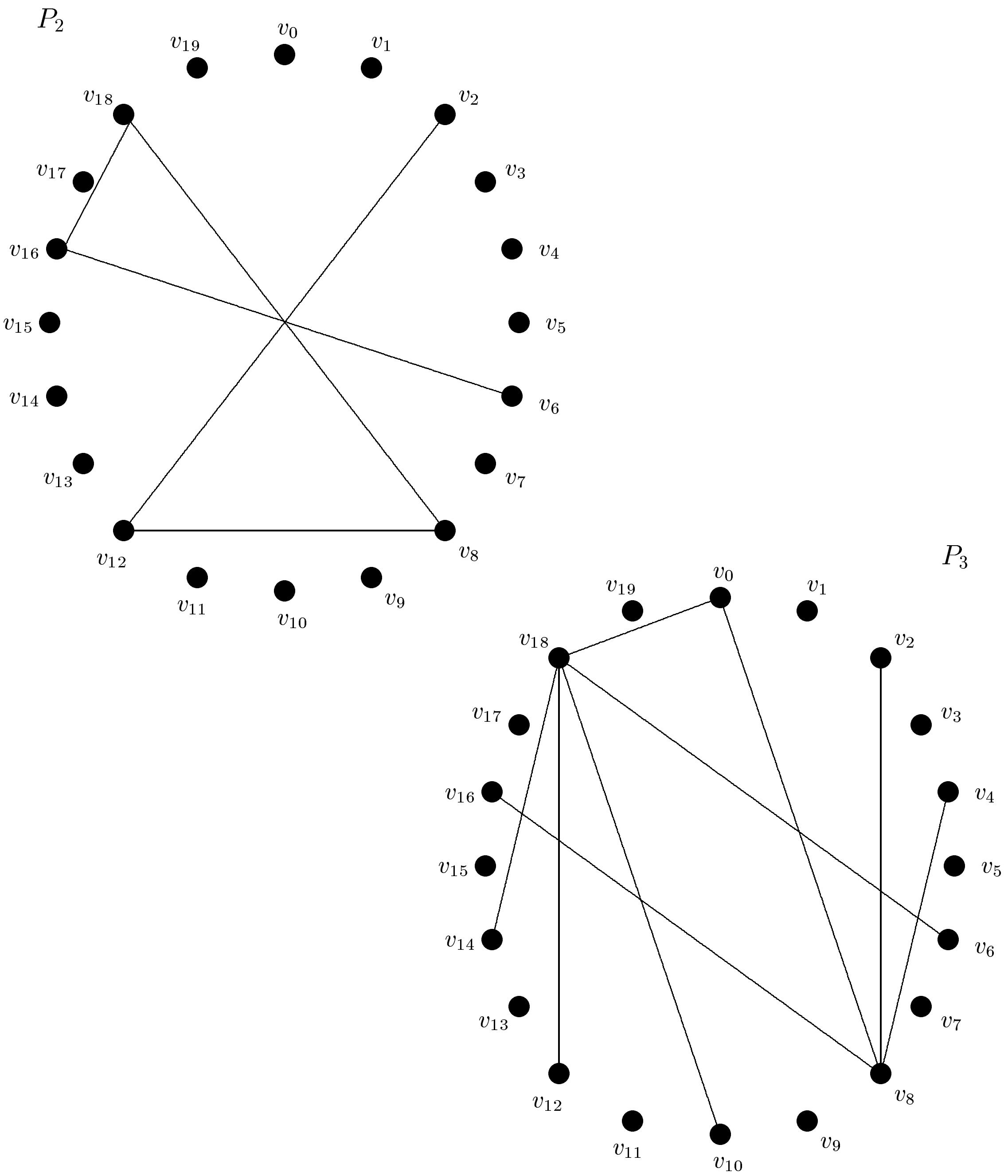

Figure 31: Edge sets P_2 and P_3 for G_5 when $n = 20$.

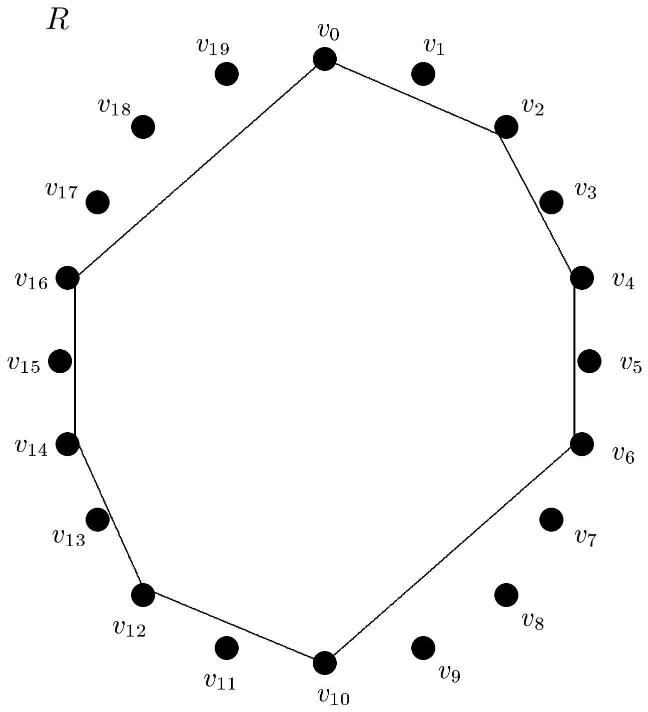

Figure 32: Edge set R for G_5 when $n = 20$.

Chapter 6

Generalized Tree-Carvings

Khuller and Vishkin introduced the concept of a tree-carving. We generalize this concept and use our generalization to prove an upper bound of 1.7 on the performance ratio of Khuller and Raghavachari's algorithm for the Minimum k-Edge-Connected Spanning Subgraph problem, for large values of k.

6.1 Tree-Carvings and our Generalization

Recall that a tree-carving in a graph $G = (V, E)$ is a partition of the vertex set V into subsets $V_1, V_2, \ldots, V_t$ with the following properties. Each subset constitutes a node of a tree Γ. For every edge $e = uv$ of G, the unique x and y such that $u \in V_x$ and $v \in V_y$ satisfy either $x = y$ or V_x is adjacent to V_y in Γ.

To give our generalization of a tree-carving, consider a positive integer k, a k-edge-connected graph $G = (V, E)$, a nonnegative integer $c < k$, and a subset S of E such that (V, S) is a c-edge-connected spanning subgraph of G.

A *c-tree-carving in G with respect to S* is a partition of the vertex set V into subsets $V_1, V_2, \ldots, V_t$ with the following properties.

1) Each subset consists of a node of a *rooted forest* Γ.

2) For every edge $e = uv$ of $(V, E - S)$, the unique x and y such that $u \in V_x$ and $v \in V_y$ satisfy either $x = y$ or V_x is adjacent to V_y in Γ.

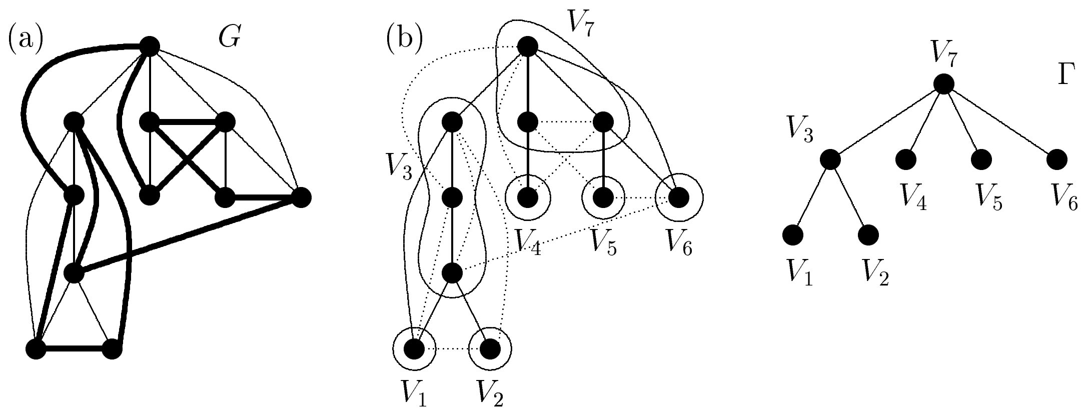

Figure 33: (a) A graph G and a 2-edge-connected spanning subgraph S of G, indicated by the dark edges. (b) A 2-tree-carving in G with respect to S.

3) For each arc e in Γ, there is a cut $(Y_e, \overline{Y_e}) = \{e\}$ in Γ such that, for $X_e = \cup\{V_j : V_j \in Y_e\}$, the cut $(X_e, \overline{X_e})$ in G contains exactly c edges of S. Denote by P_e the set of edges in $(X_e, \overline{X_e})$ not in S.

Let us make some observations about this definition. Note that $P_e \subseteq E(G)$. For any $e = (V_i, V_j)$ in Γ, observe that any edge in P_e has one endpoint in V_i and the other in V_j. Also, if Γ is a tree, there are exactly two choices for Y_e (the vertex set of each one the two connected components resulting from the removal of e from Γ). If Γ is not a tree, then there might be more than two choices for Y_e, depending on in which side of the cut $(Y_e, \overline{Y_e})$ each other component of Γ is. When $c = 0$, $S = \emptyset$ and Γ is a tree, this definition is the same as the tree-carving definition. In Figure 33 (a), we show a graph G and a subset S of the edges of G (the dark edges) such that (V, S) is a 2-edge-connected spanning subgraph of G. In (b), we show a 2-tree-carving for G with respect to S.

In this chapter, the concept of a c-tree-carving is used to extend some of the results from the previous chapter. In particular, we present stronger versions of both Lemmas 5.3.1 and 5.3.4. From these, one can get an upper bound on the $|B_i|$'s which is better than the ones given by Corollary 5.3.5, and, as a consequence, conclude that the performance ratio of the algorithm is smaller than 1.7 for large enough k.

6.2 Relation with Khuller and Raghavachari's Algorithm

Khuller and Raghavachari's algorithm can be modified so that, at each phase, a $2(i-1)$-tree-carving in G with respect to S_{i-1} is computed. This is done as follows. At the end of phase i, let $T_i = \{t_e : e \in B_i\}$. Note that $T_i \subseteq F_i$ and $|T_i| = |B_i|$. Let Γ_i be the rooted forest obtained from F_i by contracting all the edges of $F_i - T_i$. Clearly, Γ_i has exactly $|T_i| = |B_i|$ arcs. Each node of Γ_i corresponds to a subset of the original graph $G = (V, E)$ that was collapsed into this node. So, the nodes of Γ_i are subsets $V_1, V_2, \ldots, V_t$ of V which define a partition of V where $t = |B_i| + 1$. Hence 1) holds. Note that each set V_j is the vertex set of a connected component of $(V, F_i - T_i)$.

Now, let us verify 2). Let $g = uv$ be an edge in $E - S_{i-1}$, and let x and y be such that $u \in V_x$ and $v \in V_y$. We want to prove that either $x = y$ or V_x is adjacent to V_y in Γ. Consider the unique $u-v$ path P in F_i (there is a $u-v$ path in F_i since F_i is a DFS forest in $(V, E - S_{i-1})$). If P has no edge in T_i, then u and v are in the same connected component of $(V, F_i - T_i)$, implying that $x = y$. If P has exactly one edge t in T_i, then u is connected to one of the endpoints of t, say a, in $(V, F_i - T_i)$, and v to the other endpoint, say $b \neq a$. So $a \in V_x$, $b \in V_y$ and V_x is adjacent to V_y in Γ (edge t is the edge connecting them in Γ).

Thus, to complete the proof of 2), it is enough to show that P has at most one edge in T_i. If $g \in F_i$, then P has exactly one edge (edge g itself). Therefore, certainly P has at most one edge in T_i. If $g \notin F_i$, then $g \in E - S_{i-1} - F_i$, implying that g is a back edge with respect to F_i. If there is a $t_e \in T_i$ in P, then g is a back edge of t_e (because t_e lies on the path in F_i between the endpoints of g), that is, $g \in A_{t_e}$. Recall that $A_{t_e} \cap A_{t_f} \subseteq P_e \cap P_f = \emptyset$, for all $f \in B_i, f \neq e$ (by (14) and Lemma 5.2.3, on page 80). If there were another edge $t_f \in T_i$, $f \neq e$, in P, then $g \in A_{t_f}$, contradicting the fact that $A_{t_e} \cap A_{t_f} \neq \emptyset$. Thus, there is no other edge $t_f \in T_i$, $f \neq e$, in P. This means there is at most one edge of T_i in P.

For 3), recall that for any edge $e \in B_i$, there is a cut $(X_e, \overline{X_e})$ in $(V, S_{i-1} \cup F_i \cup B_i^e)$ of size $2i - 1$ separating the endpoints of t_e, where B_i^e is B_i just before e was included in B_i by the algorithm. Such a cut contains exactly $2i - 2 = 2(i - 1)$ edges of S_{i-1}, exactly one edge (edge t_e) of F_i and no edges of B_i^e. Because $(X_e, \overline{X_e})$ has exactly one edge, edge t_e, in F_i and $t_e \in T_i \subseteq F_i$, it has no edge in $F_i - T_i$. So, if we contract the edges in $F_i - T_i$, both endpoints of each edge we contract are in the same side of the cut $(X_e, \overline{X_e})$. This means that each node V_j of Γ_i (which, by definition, is the forest we obtain if we contract the edges in $F_i - T_i$) is a subset of either X_e or $\overline{X_e}$. In other words, for each j, either $V_j \subseteq X_e$ or $V_j \subseteq \overline{X_e}$. Thus we can define $Y_e = \{V_j : V_j \subseteq X_e\}$, and $\overline{Y_e} = \{V_j : V_j \subseteq \overline{X_e}\}$. Note that in fact $Y_e \cup \overline{Y_e} = \{V_1, V_2, \ldots, V_t\}$, hence $(Y_e, \overline{Y_e})$ is a cut in Γ_i. Moreover, $(Y_e, \overline{Y_e}) = \{t_e\}$, since t_e is the unique edge of T_i in $(X_e, \overline{X_e})$. The set X_e defined in 3) coincides with the set X_e (or with $\overline{X_e}$) of the algorithm, which contains exactly $2i - 2 = 2(i - 1)$ edges of S_{i-1}, completing the proof of 3).

In Figure 34, we show the 0-tree-carving that the algorithm outputs when running on the example given in Figure 16 (page 78). The cut $(X_{f_i}, \overline{X_{f_i}})$ associated with the arc f_i in Γ is exactly the cut $(X_i, \overline{X_i})$ shown in Figure 16 (page 78), and $P_{f_i} = P_{e_i}$.

6.3 Properties of c-Tree-Carvings

Consider a c-tree-carving $V_1, \ldots, V_t$ of G with respect to S. And let Γ, $(X_e, \overline{X_e})$ and P_e be as in the definition of c-tree-carving. The next fact is the analogue of (15) (page 80).

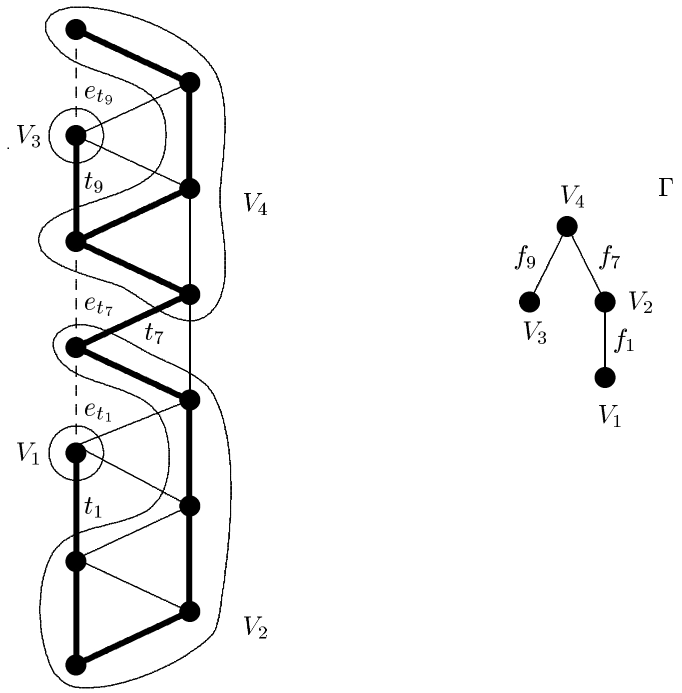

Figure 34: The 0-tree-carving output by the algorithm with graph G as input, F_1 as the set of dark edges and B_1 as the set of dashed edges.

Fact 6.3.1 *Any k-edge-connected spanning subgraph of G has at least $k-c$ edges in each P_e.*

Proof. Let H be a k-edge-connected spanning subgraph of G. For each arc e in Γ, there must be at least k edges of H in $(X_e, \overline{X_e})$. Exactly c edges of $(X_e, \overline{X_e})$ are in S. Hence there must be at least $k-c$ edges of H in $(X_e, \overline{X_e})$ not in S, that is, in P_e. ■

The next fact corresponds to Lemma 5.2.3 (page 80).

Fact 6.3.2 *For any distinct arcs e and f in Γ, the edge sets $P_e \subseteq E(G)$ and $P_f \subseteq E(G)$ are disjoint.*

Proof. Let $e = (V_i, V_j)$ and $f = (V_x, V_y)$ be two arcs in Γ. Suppose that $P_e \cap P_f \neq \emptyset$. Let us prove that $e = f$. Let $h \in P_e \cap P_f$. By one of the observations just after the definition of c-tree-carving, h has one endpoint in V_i and the other in V_j. Also, for the same reason, h has one endpoint in V_x and the other in V_y. But, since $V_1, V_2, \ldots, V_t$ is a partition of V, we must have ($V_i = V_x$ and $V_j = V_y$) or ($V_i = V_y$ and $V_j = V_x$). In both cases, $e = f$. ■

Denote the number of leaves of Γ by l. The following result is an extension of Lemma 5.3.1 (page 87). Its proof mimics closely the proof of Lemma 5.3.1.

Lemma 6.3.3 *Any k-edge-connected spanning subgraph H of G has at least*

$$(k-c)(t-1) + cl/2$$

edges.

Proof. Call a pair (V_i, h) *special* if h is an edge of H in $(V_i, \overline{V_i})$. Denote by m the number of edges in H. The number of special pairs (V_i, h) is at most $2m$: at most two for each edge

h of H. Next, we compute a lower bound for the number of special pairs: we prove that the number of special pairs is at least $2(k-c)(t-1-l)+((k-c)+k)l = 2(k-c)(t-1)+cl$. From this lower bound, the lemma follows trivially.

For the lower bound, we associate a set of special pairs with each arc $e = (V_x, V_y)$ in Γ. We may assume, without loss of generality, that V_x is the parent of V_y. If V_y is a leaf of Γ, then e will be associated with at least $(k-c)+k$ special pairs. If V_y is not a leaf, then e will be associated with at least $2(k-c) \leq (k-c)+k$ special pairs. No special pair will be associated with two different arcs in Γ.

By Fact 6.3.1, H has at least $k-c$ edges h in P_e. Suppose V_y is not a leaf of Γ. Associate with e two special pairs for each of these at least $k-c$ edges h in P_e: (V_x, h) and (V_y, h). This scheme associates at least $2(k-c)$ special pairs with e.

Now, suppose V_y is a leaf of Γ. In a first phase, associate with e the special pair (V_x, h) where h is an edge of H in P_e. In this phase, at least $k-c$ pairs are associated with e. Then, note that $|E(H) \cap (V_y, \overline{V_y})| \geq k$ (because H is k-edge-connected). Associate also with e the special pairs (V_y, h), for all h in $E(H) \cap (V_y, \overline{V_y})$. In total, we have associated with e at least $(k-c)+k$ special pairs, because V_y is a leaf of Γ and thus is not the parent of any other V_z, $z \neq x$.

Before showing that no two distinct arcs e and f in Γ have an associated special pair in common, let us state a property of the pairs associated with an arc e in Γ.

Fact 6.3.4 *Let $e = (V_x, V_y)$ be an arc in Γ, where V_x is the parent of V_y. Any pair (V_z, h) associated with arc e is such that either $h \in P_e$ or $z = y$. Moreover, if $h \notin P_e$, then $V_z = V_y$ is a leaf of Γ.*

Proof. If V_y is not a leaf of Γ, this is clearly true. If V_y is a leaf of Γ, then in the first phase, we associate pairs (V_x, h) with e such that $h \in P_e$. Besides these pairs, we associate with e pairs (V_y, h). So any pair (V_i, h) associated with e is such that either $h \in P_e$ or $V_i = V_y$ is a leaf of Γ. ■

Let $e = (V_x, V_y)$ and $f = (V_i, V_j)$ in Γ be two distinct arcs in Γ. We may assume, without loss of generality, that V_x is the parent of V_y and V_i is the parent of V_j. Since $e \neq f$, $j \neq y$.

Note that, for any cut $(Y_f, \overline{Y_f}) = \{f\}$ in Γ, we may assume that $V_i \subseteq Y_f$ and $V_j \subseteq \overline{Y_f}$. So $(V_i, V_j) \subseteq (X_f, \overline{X_f})$ (defined as in 3) of the c-tree-carving definition). Thus,

$$(V_i, V_j) \cap (E - S_{i-1}) \subseteq P_f. \tag{17}$$

Let us prove that e and f do not have an associated special pair in common. Suppose, for a contradiction, that they do. Let (V_z, h) be such a pair. Since $j \neq y$, either $z \neq j$ or $z \neq y$. Without loss of generality, we may assume that $z \neq y$. This means, by Fact 6.3.4, that $h \in P_e \subseteq E - S_{i-1}$ (implying, in particular, that $h \notin S_{i-1}$). As was said in Fact 6.3.2, because $e \neq f$, the sets P_e and P_f are disjoint, so $h \notin P_f$. Thus $h \notin P_f$ and $h \notin S_{i-1}$.

To complete the proof, it is enough to prove that $h \in (V_i, V_j)$, because this will imply, by (17) and since $h \notin P_f$, that $h \in S_{i-1}$, a contradiction.

So let us prove that $h \in (V_i, V_j)$. Applying Fact 6.3.4 to $f = (V_i, V_j)$, since (V_z, h) is a special pair associated with f and, we infer that V_j is a leaf of Γ and $z = j$ (because $h \notin P_f$). Recall that $h \in (V_z, \overline{V_z})$, that is, $h \in (V_j, \overline{V_j})$. Hence h has one endpoint in V_j and the other in some $V_x \in \overline{V_j}$. Since $h \notin S_{i-1}$, by 2) of the c-tree-carving definition, the endpoint of h in $\overline{V_j}$ must be in V_i (which is the only adjacent of V_j in Γ), that is, $h \in (V_i, V_j)$. This completes the proof that no two distinct arcs e and f in Γ have an associated pair in common.

Therefore, we must have at least $2(k-c)(t-1-l) + ((k-c)+k)l = 2(k-c)(t-1) + cl$ special pairs. Joining this to the fact that there are exactly $2m$ special pairs, we conclude the proof of the Lemma 6.3.3. ■

The generalization of Lemma 5.3.4 we want to present gives a lower bound on l. But before stating the generalization, we need to prove one more fact.

Fact 6.3.5 *For each arc $e = (V_i, V_j)$ in Γ, $|V_i| + |V_j| \geq 2\sqrt{k-c}$.*

Proof. Consider the cut $(X_e, \overline{X_e})$ in G corresponding to e. Since G is k-edge-connected, and $(X_e, \overline{X_e})$ contains exactly c edges of S, there must be at least $k-c$ edges of $E-S$ in $(X_e, \overline{X_e})$. These edges must have one endpoint in V_i and other in V_j. The maximum number of edges with one endpoint in V_i and the other in V_j is $|V_i| \cdot |V_j|$. Thus $|V_i| \cdot |V_j| \geq k-c$. This implies that $|V_i| + |V_j| \geq 2\sqrt{k-c}$. ■

Now we can prove the following lower bound on l, the number of leaves of the forest Γ.

Lemma 6.3.6 $$l \geq \left(1 + \frac{1}{\lfloor\sqrt{k-c}\rfloor - 1}\right) t - \frac{1}{(\lfloor\sqrt{k-c}\rfloor - 1)}\, n.$$

Proof. The proof of this lemma uses basically the same technique used in Lemma 5.3.4. We will associate a set Z_i of vertices of V with each V_i, where $Z_i \subseteq V_i \cup V_j$ for some V_j which is adjacent to V_i in Γ. The sets Z_i will be pairwise disjoint and will contain exactly $\lfloor\sqrt{k-c}\rfloor$ vertices if V_i is not a leaf of Γ, and exactly one vertex if V_i is a leaf of Γ. The association is done in two phases. (Phase 1 is executed before Phase 2.)

Phase 1. For each V_i which is a leaf of Γ or such that $|V_i| \geq \lfloor\sqrt{k-c}\rfloor$:

Let Z_i be a subset of V_i with exactly $\lfloor\sqrt{k-c}\rfloor$ vertices if V_i is not a leaf of Γ, and with one vertex if V_i is a leaf. (Note that $|V_i| \geq 1$ always.)

Phase 2. For each V_i which is not a leaf of Γ and such that $|V_i| < \lfloor\sqrt{k-c}\rfloor$:

Let $e = (V_i, V_j)$ be an arc of Γ joining V_i to one of its children V_j. By Fact 6.3.5, $|V_i| + |V_j| \geq 2\sqrt{k-c}$. Thus we must have $|V_j| \geq 2\sqrt{k-c} - |V_i| \geq \sqrt{k-c}$, and so $|V_j| \geq \lfloor\sqrt{k-c}\rfloor$, which implies that V_j had its set Z_j assigned previously in

Phase 1. Choose Z_i to be a subset of $V_i \cup (V_j - Z_j)$ with exactly $\lfloor\sqrt{k-c}\rfloor$ vertices. (There is such a subset since $|V_i \cup (V_j - Z_j)| = |V_i| + |V_j| - \lfloor\sqrt{k-c}\rfloor \geq \lfloor\sqrt{k-c}\rfloor$.)

In Phase 1, clearly the assigned sets are pairwise disjoint. It is also clear that a set assigned in Phase 2 does not intersect a set assigned in Phase 1. By Fact 6.3.5, two sets assigned in Phase 2 cannot be associated with adjacent sets V_y. So they are disjoint.

Since the union of the sets Z_i is a subset of V, we must have $(t-l)\lfloor\sqrt{k-c}\rfloor + l \leq n$, where $t = |V(\Gamma)|$. This means

$$l \geq \frac{\lfloor\sqrt{k-c}\rfloor}{\lfloor\sqrt{k-c}\rfloor - 1} t - \frac{1}{\lfloor\sqrt{k-c}\rfloor - 1}\, n = \left(1 + \frac{1}{\lfloor\sqrt{k-c}\rfloor - 1}\right) t - \frac{1}{(\lfloor\sqrt{k-c}\rfloor - 1)}\, n,$$

concluding the proof of the lemma. ■

6.4 Back to Khuller and Raghavachari's Algorithm

In Section 6.2, we showed how to modify the algorithm so that it outputs at each phase a $2(i-1)$-tree-carving, denoted simply by Γ_i, of G with respect to S_{i-1} and having $|B_i|$ arcs.

Recall that l_i is the number of b_e's which are leaves of F_i. Recall that b_e is the bottom endpoint of $t_e \in T_i$. If b_e is a leaf of F_i then t_e is the only edge of F_i incident to b_e, which means that $\{b_e\}$ is the vertex set of a connected component of $(V, F_i - T_i)$. In other words, there is an x such that $V_x = \{b_e\}$. Finally, note that V_x is a leaf of Γ_i. All this implies that Γ_i has at least l_i leaves.

Applying Lemma 6.3.6 to Γ_i, we get the following inequality:

$$l_i \geq \left(1 + \frac{1}{\lfloor\sqrt{k-2(i-1)}\rfloor - 1}\right) |B_i| - \frac{1}{(\lfloor\sqrt{k-2(i-1)}\rfloor - 1)}\, n, \qquad (18)$$

since $t = |V(\Gamma_i)| > |B_i|$.

We also have, for $1 \leq i \leq j$,

$$
\begin{aligned}
l_i &\geq \left(1 + \frac{1}{\lfloor\sqrt{k-2(i-1)}\rfloor - 1}\right)|B_i| - \frac{1}{(\lfloor\sqrt{k-2(i-1)}\rfloor - 1)}\, n \\
&= |B_i| + \frac{1}{(\lfloor\sqrt{k-2(i-1)}\rfloor - 1)}(n - |B_i|) \\
&\geq |B_i| + \frac{1}{(\lfloor\sqrt{k-2(j-1)}\rfloor - 1)}(n - |B_i|),\ \text{since } n \geq |B_i| \text{ and } 1 \leq i \leq j,
\end{aligned}
$$

which means

$$
l_i \geq \left(1 + \frac{1}{\lfloor\sqrt{k-2(j-1)}\rfloor - 1}\right)|B_i| - \frac{1}{(\lfloor\sqrt{k-2(j-1)}\rfloor - 1)}\, n, \tag{19}
$$

for any $1 \leq i \leq j$.

The next corollary gives the new upper bounds on $|B_i|$.

Corollary 6.4.1

$$
|B_i| \leq \frac{k(\lfloor\sqrt{k-2(i-1)}\rfloor - 1) + 2(i-1)}{k((k-(i-1))(\lfloor\sqrt{k-2(i-1)}\rfloor - 1) + (i-1))}\, opt, \tag{20}
$$

and, for any $1 \leq i \leq j$,

$$
|B_i| \leq \frac{k(\lfloor\sqrt{k-2(j-1)}\rfloor - 1) + 2(i-1)}{k((k-(i-1))(\lfloor\sqrt{k-2(j-1)}\rfloor - 1) + (i-1))}\, opt. \tag{21}
$$

Proof. Let us start proving (20). From Lemma 6.3.3 applied to Γ_i, which is a $2(i-1)$-tree-carving with $t > |B_i|$ nodes, we infer that

$$\begin{aligned}
opt &\geq (k-2(i-1))|B_i| + (i-1)l_i \\
&\geq (k-2(i-1))|B_i| \\
&\quad +(i-1)\left(1+\frac{1}{\lfloor\sqrt{k-2(i-1)}\rfloor-1}\right)|B_i| - (i-1)\frac{1}{(\lfloor\sqrt{k-2(i-1)}\rfloor-1)}\, n, \text{ by (18)} \\
&= \left(k-(i-1)+\frac{(i-1)}{\lfloor\sqrt{k-2(i-1)}\rfloor-1}\right)|B_i| - \frac{(i-1)}{(\lfloor\sqrt{k-2(i-1)}\rfloor-1)} n \\
&\geq \left(k-(i-1)+\frac{(i-1)}{\lfloor\sqrt{k-2(i-1)}\rfloor-1}\right)|B_i| \\
&\quad -\frac{2(i-1)}{k(\lfloor\sqrt{k-2(i-1)}\rfloor-1)} opt, \text{ because } opt \geq kn/2 \\
&= \frac{(k-(i-1))(\lfloor\sqrt{k-2(i-1)}\rfloor-1)+(i-1)}{\lfloor\sqrt{k-2(i-1)}\rfloor-1}|B_i| - \frac{2(i-1)}{k(\lceil\sqrt{k-2(i-1)}\rceil-1)} opt.
\end{aligned}$$

Therefore,

$$|B_i| \leq \frac{k(\lfloor\sqrt{k-2(i-1)}\rfloor-1)+2(i-1)}{k((k-(i-1))(\lfloor\sqrt{k-2(i-1)}\rfloor-1)+(i-1))}\, opt.$$

The proof of (21) is similar, so we omit it. ■

Using these new upper bounds, we derive our final result.

Theorem 6.4.2 *The performance ratio of Khuller and Raghavachari's algorithm, for large enough k, is at most 1.7.*

Proof. We will give the proof for even k. The proof for odd k is similar.

We start by using, for some fixed $r \geq 2$, bound (20) for $B_{\frac{k}{2}-r+1}, \ldots, B_{\frac{k}{2}}$. For the remaining B_i's, we use bound (21) with $j = \frac{k}{2} - r$. From this, we infer, for even k, that

$$\begin{aligned}
\frac{|S_{k/2}|}{opt} &= \frac{\sum_{i=1}^{k/2}(|F_i| + |B_i|)}{opt} \leq 1 + \sum_{i=1}^{j} \frac{|B_i|}{opt} + \sum_{i=j+1}^{k/2} \frac{|B_i|}{opt} \\
&\leq 1 + \sum_{i=j+1}^{k/2} \frac{k(\lfloor\sqrt{k-2(i-1)}\rfloor - 1) + 2(i-1)}{k((k-(i-1))(\lfloor\sqrt{k-2(i-1)}\rfloor - 1) + (i-1))} \\
&\quad + \sum_{i=1}^{j} \frac{k(\lfloor\sqrt{k-2(j-1)}\rfloor - 1) + 2(i-1)}{k((k-(i-1))(\lfloor\sqrt{k-2(j-1)}\rfloor - 1) + (i-1))}.
\end{aligned}$$

Now, for the numerator of the first summand, note that, for $\frac{k}{2} - r + 1 = j + 1 \leq i \leq \frac{k}{2}$, $k(\lfloor\sqrt{k-2(i-1)}\rfloor - 1) + 2(i-1) \leq k(\lfloor\sqrt{2r}\rfloor - 1) + 2(\frac{k}{2} - 1) = k\lfloor\sqrt{2r}\rfloor - 2$. For the second sum, because $j = \frac{k}{2} - r$, we have that $\sqrt{k-2(j-1)} = \sqrt{2(r+1)}$. Hence

$$\begin{aligned}
\frac{|S_{k/2}|}{opt} &\leq 1 + \sum_{i=j+1}^{k/2} \frac{k\lfloor\sqrt{2r}\rfloor - 2}{k((k-(i-1))(\lfloor\sqrt{k-2(i-1)}\rfloor - 1) + (i-1))} \\
&\quad + \sum_{i=1}^{j} \frac{k(\lfloor\sqrt{2(r+1)}\rfloor - 1) + 2(i-1)}{k((k-(i-1))(\lfloor\sqrt{2(r+1)}\rfloor - 1) + (i-1))}.
\end{aligned}$$

Note that for $\frac{k}{2} - r + 1 = j + 1 \leq i \leq \frac{k}{2}$, the denominator of the first summand

$$\begin{aligned}
&k((k-(i-1))(\lfloor\sqrt{k-2(i-1)}\rfloor - 1) + (i-1)) \geq \\
&\quad k((k-(\frac{k}{2}-1))(\lfloor\sqrt{k-2(\tfrac{k}{2}-1)}\rfloor - 1) + ((\frac{k}{2}-r+1)-1)) = \frac{k}{2} - r.
\end{aligned}$$

Thus, substituting the denominator of the first summand, we get

$$\frac{|S_{k/2}|}{opt} \leq 1 + \sum_{i=j+1}^{k/2} \frac{2(k\lfloor\sqrt{2r}\rfloor - 2)}{k(k-2r)}$$
$$+\sum_{i=1}^{j} \frac{k(\lfloor\sqrt{2(r+1)}\rfloor - 1) + 2(i-1)}{k((k-(i-1))(\lfloor\sqrt{2(r+1)}\rfloor - 1) + (i-1))}.$$

Denoting $\lfloor\sqrt{2(r+1)}\rfloor$ by C_r, just to simplify the formulas, we have

$$\begin{aligned}\frac{|S_{k/2}|}{opt} &\leq 1 + r\frac{2(k\lfloor\sqrt{2r}\rfloor - 2)}{k(k-2r)} + \sum_{i=1}^{j} \frac{k(C_r - 1) + 2(i-1)}{k((k-(i-1))(C_r - 1) + (i-1))} \\ &\leq 1 + \frac{2r(k\lfloor\sqrt{2r}\rfloor - 2)}{k(k-2r)} - \frac{2j}{k(C_r - 2)} \\ &\quad + \sum_{i=1}^{j}\left(\frac{2}{k(C_r-2)} + \frac{k(C_r - 1) + 2(i-1)}{k((k-(i-1))(C_r - 1) + (i-1))}\right).\end{aligned}$$

(Note that indeed $C_r = \lfloor\sqrt{2(r+1)}\rfloor \geq \lfloor\sqrt{6}\rfloor > 2$, because $r \geq 2$).

Next, we will manipulate the last term of the equation above.

$$\begin{aligned}&\sum_{i=1}^{j}\left(\frac{2}{k(C_r-2)} + \frac{k(C_r - 1) + 2(i-1)}{k((k-(i-1))(C_r - 1) + (i-1))}\right) \\ &= \sum_{i=1}^{j} \frac{2((k-(i-1))(C_r-1) + (i-1)) + k(C_r-2)(C_r-1) + 2(C_r-2)(i-1)}{k(C_r-2)((k-(i-1))(C_r-1) + (i-1))},\end{aligned}$$

and the numerator

$$2((k-(i-1))(C_r-1) + (i-1)) + k(C_r-2)(C_r-1) + 2(C_r-2)(i-1)$$

$$
\begin{aligned}
&= 2k(C_r-1) - 2(i-1)(C_r-1) + 2(i-1) + k(C_r-2)(C_r-1) + 2(C_r-2)(i-1) \\
&= k[2+(C_r-2)](C_r-1) + 2(i-1)[-(C_r-1)+1+(C_r-2)] \\
&= kC_r(C_r-1).
\end{aligned}
$$

So

$$
\begin{aligned}
&\sum_{i=1}^{j}\left(\frac{2}{k(C_r-2)} + \frac{k(C_r-1)+2(i-1)}{k((k-(i-1))(C_r-1)+(i-1))}\right) \\
&= \sum_{i=1}^{j}\frac{kC_r(C_r-1)}{k(C_r-2)((k-(i-1))(C_r-1)+(i-1))} \\
&= \frac{C_r(C_r-1)}{C_r-2}\sum_{i=1}^{j}\frac{1}{(C_r-1)k-(C_r-2)(i-1)} \\
&= \frac{C_r(C_r-1)}{(C_r-2)^2}\sum_{i=1}^{j}\frac{1}{\left(\frac{C_r-1}{C_r-2}\right)k-(i-1)} \\
&\leq \frac{C_r(C_r-1)}{(C_r-2)^2}\left(H_{\lfloor\left(\frac{C_r-1}{C_r-2}\right)k\rfloor} - H_{\lfloor\left(\frac{C_r-1}{C_r-2}\right)k\rfloor-j}\right),
\end{aligned}
$$

where $H_q = \sum_{p=1}^{q}\frac{1}{p}$ is the so-called q^{th} *Harmonic number*.

We know that, as q goes to infinity, $H_q - \ln q$ converges to a constant γ. Hence, for any $\epsilon > 0$, we can choose q_0 large enough that $\gamma - \epsilon < H_q - \ln q < \gamma + \epsilon$, for all $q > q_0$.

Set $q = \lfloor\left(\frac{C_r-1}{C_r-2}\right)k\rfloor$. Note that, as k goes to infinity, q goes to infinity (because r is fixed). Also, $q - j = \lfloor\left(\frac{C_r-1}{C_r-2}\right)k\rfloor - j = \lfloor\left(\frac{C_r-1}{C_r-2}\right)k\rfloor - \frac{k}{2} + r > \left(\frac{C_r-1}{C_r-2}\right)k - 1 - \frac{k}{2} + r = \left(\frac{C_r-1}{C_r-2} - \frac{1}{2}\right)k + r - 1 > \frac{k}{2} + r - 1$, because $\frac{C_r-1}{C_r-2} > 1$ for any $r \geq 2$. So, as k goes to infinity, $q - j$ goes to infinity as well.

Thus, for any $\epsilon > 0$, we can choose k_0 large enough that $H_q - \ln q < \gamma + \epsilon/2$ and $H_{q-j} - ln(q-j) > \gamma - \epsilon/2$, for all $k > k_0$. This implies that

$$H_q - H_{q-j} < \ln q - \ln(q-j) + \epsilon = \ln\frac{q}{q-j} + \epsilon = \ln\left(1 + \frac{j}{q-j}\right) + \epsilon \leq \ln\left(1 + \frac{j}{q'-1-j}\right) + \epsilon,$$

where $q' = \left(\frac{C_r - 1}{C_r - 2}\right) k$ (and $q' - 1 < q$).

So, for large enough k, we have

$$\begin{aligned}
&\sum_{i=1}^{j}\left(\frac{2}{k(C_r-2)} + \frac{k(C_r-1)+2(i-1)}{k((k-(i-1))(C_r-1)+(i-1))}\right) \\
&\leq \frac{C_r(C_r-1)}{(C_r-2)^2}\left(\ln\left(1+\frac{j}{q'-1-j}\right)+\epsilon\right) \\
&= \frac{C_r(C_r-1)}{(C_r-2)^2}\left(\ln\frac{q'-1}{q'-1-j}+\epsilon\right),
\end{aligned}$$

and we can infer that, for any $\epsilon > 0$,

$$\begin{aligned}
\frac{|S_{k/2}|}{opt} &\leq 1 + \frac{2r(k\lfloor\sqrt{2r}\rfloor - 2)}{k(k-2r)} - \frac{2j}{k(C_r-2)} + \frac{C_r(C_r-1)}{(C_r-2)^2}\left(\ln\frac{q'-1}{q'-1-j}+\epsilon\right) \\
&= 1 + \frac{2r(k\lfloor\sqrt{2r}\rfloor - 2)}{k(k-2r)} - \frac{k-2r}{k(C_r-2)} + \frac{C_r(C_r-1)}{(C_r-2)^2}\left(\ln\frac{q'-1}{q'-1-\frac{k}{2}+r}+\epsilon\right),
\end{aligned}$$

because $j = \frac{k}{2} - r$.

Note that C_r is function solely of r. Recall that $q' = \left(\frac{C_r-1}{C_r-2}\right) k$. So, as k goes to infinity, the bound on the ratio converges to

$$1-\frac{1}{C_r-2}+\frac{C_r(C_r-1)}{(C_r-2)^2}\ln\frac{\frac{C_r-1}{C_r-2}}{\frac{C_r-1}{C_r-2}-\frac{1}{2}}=1-\frac{1}{C_r-2}+\frac{C_r(C_r-1)}{(C_r-2)^2}\ln\frac{2(C_r-1)}{C_r}.$$

Now, let us make r goes to infinity, which implies that C_r goes to infinity as well. The above expression converges to $1+\ln 2<1.7$. From this, we finally conclude that, for large enough k, the performance ratio of the algorithm is at most 1.7. $\blacksquare$

Chapter 7

The Complexity of the Minimum 2-Edge-Connected Spanning Subgraph Problem

We show that the Minimum 2-Edge-Connected Spanning Subgraph (2-ECSS) problem is Max SNP-hard, which implies that there is a constant $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at most $1+\epsilon$ implies that $P = NP$, by results of Arora et al. [1].

7.1 The Reduction

We use the concept of L-reduction, presented in Section 1.1.3.

Denote by VC_7 the Vertex Cover problem restricted to graphs with maximum degree at most seven. Papadimitriou and Yannakakis [51] showed that VC_7 is Max SNP-complete. We prove that VC_7 L-reduces to 2-ECSS. The idea comes from [42], where a directed version of 2-ECSS is proved to be Max SNP-hard.

Theorem 7.1.1 *The Minimum 2-Edge-Connected Spanning Subgraph problem is Max SNP-hard.*

Proof. The first part of the L-reduction is a polynomial-time algorithm f and a constant α. Given any instance G of VC_7, f produces an instance H of 2-ECSS such that the minimum

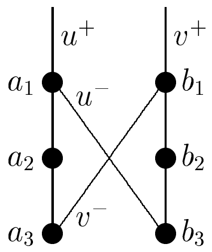

Figure 35: The gadget for edge uv.

number of edges in a 2-edge-connected spanning subgraph of H, denoted by $opt_{2-ECSS}(H)$, is at most α times the minimum size of a vertex cover in G, denoted by $opt_{VC_7}(G)$. In other words, $opt_{2-ECSS}(H) \leq \alpha \cdot opt_{VC_7}(G)$.

Let us describe algorithm f. Consider an instance G of VC_7. G is a graph of maximum degree at most seven. Here is a procedure to construct an instance H of 2-ECSS. As in [42], start with a special vertex, the root. Each vertex in G will have a "current vertex" in H, initially the root. All the vertices in H but the root will be labeled with one of the following six labels: a_1, a_2, a_3, b_1, b_2 or b_3. Sequentially, in some arbitrary order, for each edge uv in G, add a "cover-testing gadget" to H, as illustrated in Figure 35. Specifically, add six new vertices and label each of them with one of the labels a_1, a_2, a_3, b_1, b_2 and b_3. We refer to the new vertices by their labels in the remainder of this paragraph. Vertex a_2 is adjacent only to a_1 and a_3. Analogously, b_2 is adjacent only to b_1 and b_3. There is an edge labeled u^- between a_1 and b_3, and an edge labeled v^- between b_1 and a_3. There is one more edge incident to a_1: an edge labeled u^+ between a_1 and the current vertex of u; and one more edge incident to b_1: an edge labeled v^+ between b_1 and the current vertex of v. Make b_3 the new current vertex of u, and a_3 the new current vertex of v. In Figure 36,

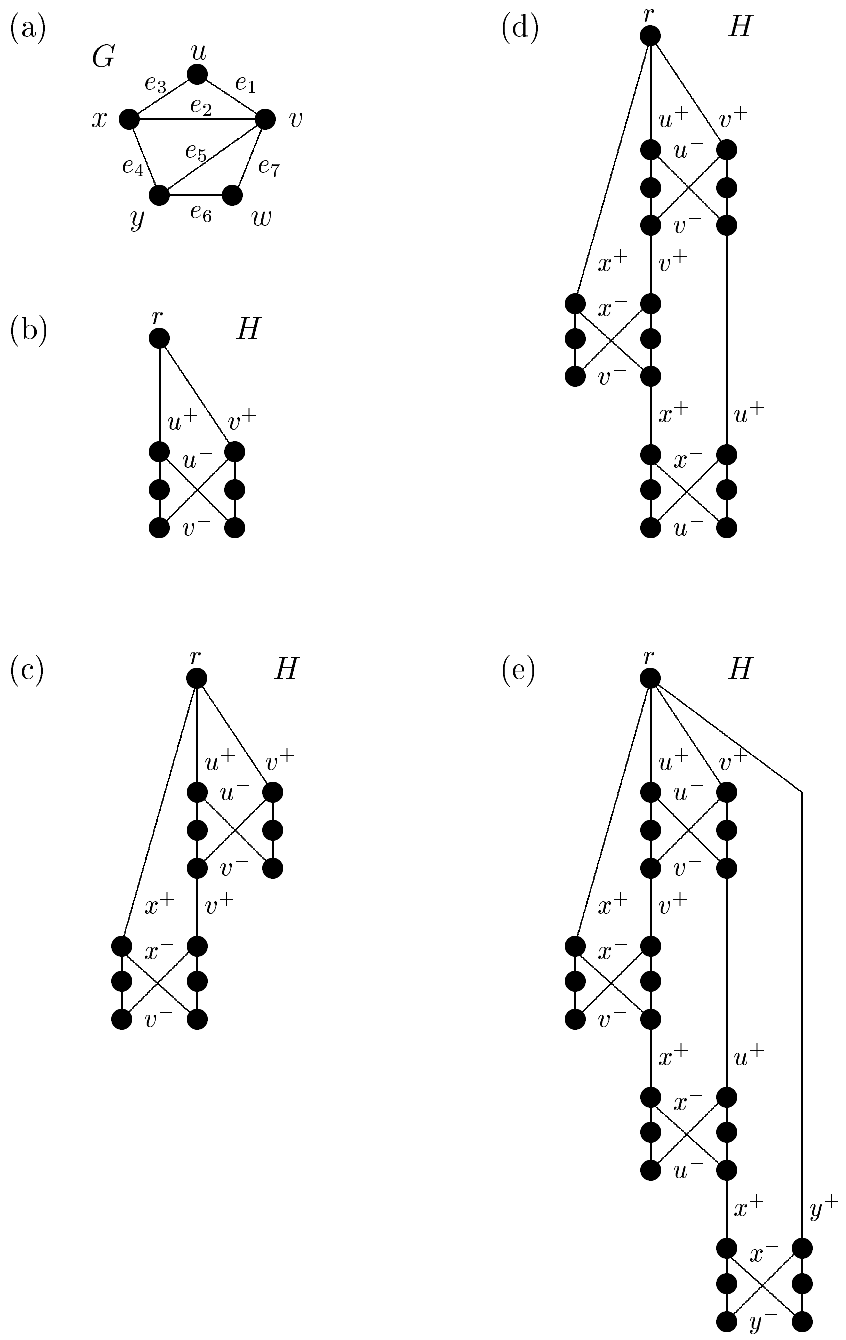

Figure 36: (a) Graph G. (b) Graph H after the gadget for edge $e_1 = uv$ of G is added. (c), (d) and (e) The same for edges $e_2 = xv, e_3 = xu$ and $e_4 = xy$, respectively.

we show the first steps of the construction of H for a given graph G.

Finally, after all the edges of G have been processed, for each vertex v in G, add an edge labeled v^+ between its final current vertex and the root. In Figures 37 and 38, we continue with the construction of H for G shown in Figure 36 (a).

The gadgets are implicitly numbered in the order we added them. Clearly H can be obtained in polynomial time in the size of G. This completes the description of f.

Let us prove a property of H that will be used later.

Fact 7.1.2 *If H' is a spanning subgraph of H and the following hold:*

1) All vertices in H' but the root have degree two.

2) For any edge uv in G, H' does not contain both the edge labeled u^- and the edge labeled v^- in the gadget for uv.

Then H' is a 2-edge-connected spanning subgraph of H.

Proof. Let us prove that H' is a collection of cycles intersecting only at the root and spanning all vertices of H. This implies that H' is a 2-edge-connected spanning subgraph of H.

First, let us prove that all cycles in H' contain the root. Consider a cycle C in H'. Assume, for a contradiction, that C does not contain the root. Since H' is spanning, each vertex labeled a_2 or b_2 in each gadget is in H'. These vertices have degree two in H. So both edges incident to each such vertex are in H'. Consider the smallest numbered gadget that has vertices of C. Say this is the gadget for edge uv of G. Let z be a vertex in C in the chosen gadget whose label (a_1, b_1, a_2, b_2, a_3 or b_3) has a subscript as small as possible. Let us prove that z has label a_1 or b_1. If z has label a_2, then the vertex labeled a_1 in this same gadget is also in C (because there are exactly two edges incident to vertices labeled

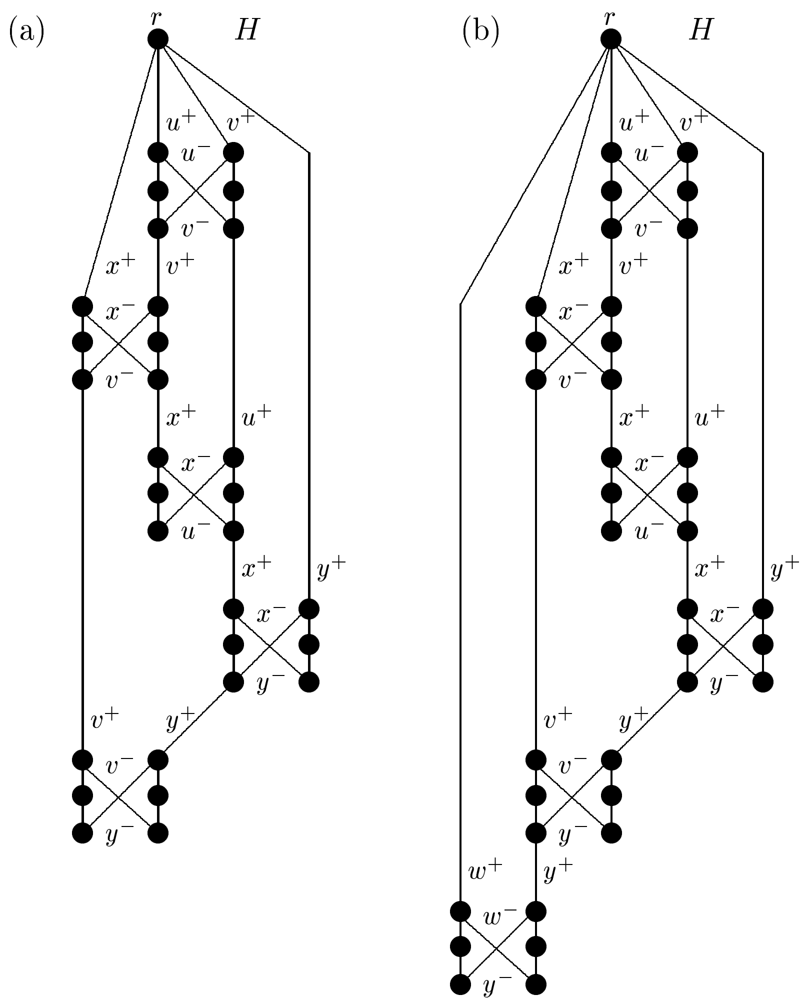

Figure 37: (a) Graph H after the gadget for edge $e_5 = vy$ of G is added. (b) The same for edge $e_6 = wy$.

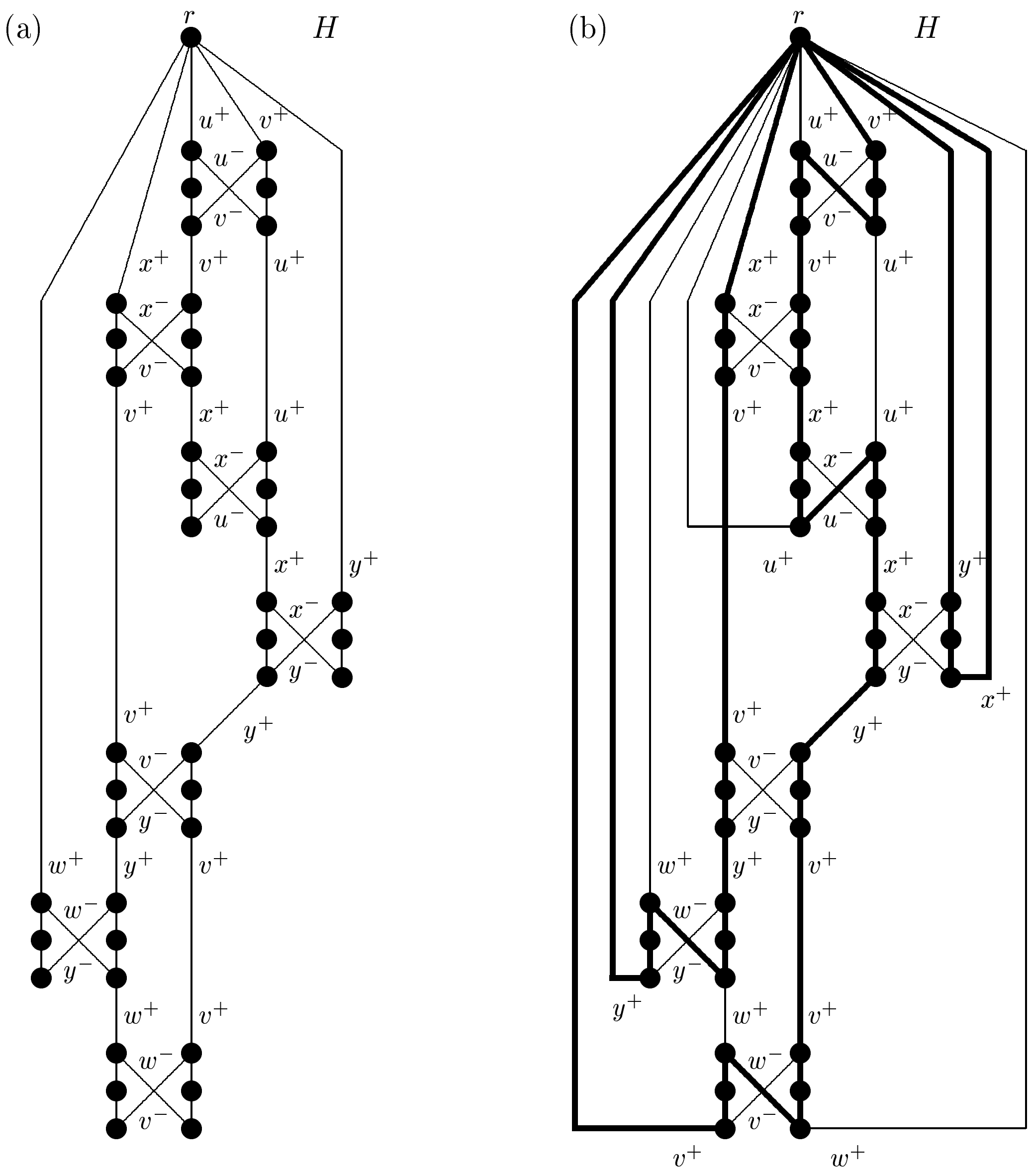

Figure 38: (a) Graph H after the gadget for edge $e_7 = wv$ of G is added. (b) The final graph H built from G. Why some edges are dark will be explained later.

a_2—one going to the vertex labeled a_1 and the other going to the vertex labeled a_3—and both these edges are in C). So z cannot have label a_2. Using the same argument, we infer that z cannot have label b_2. If z has label a_3, then since the edge from z to the vertex labeled a_2 in this same gadget is in H' and all vertices other than the root have degree two in H', this edge is also in C, meaning that the vertex labeled a_2 is in C. So z cannot have label a_3. Similarly, z cannot have label b_3. Thus, z must have either label a_1 or label b_1. Without loss of generality, we may assume z has label a_1. But because this gadget (gadget for edge uv) is the smallest numbered gadget containing vertices of C, the edge incident to z labeled u^-, whose other endpoint has label b_3, must be in C and so in H'. Because in H' all the vertices in this gadget have degree two, C contains the path z, b_3, b_2, b_1, where we are representing the vertices of the gadget by their labels. Since this gadget is the smallest numbered gadget containing vertices of C, the edge in this gadget incident to b_1 labeled v^- is in C and so in H'. But then both the edge labeled u^- and the edge labeled v^- are in H', contradicting 2). Therefore every cycle in H' includes the root.

Because all vertices but the root have degree two, if we remove the root, we get paths and cycles. But we proved that there is no cycle without the root. Thus H' is a collection of cycles intersecting at the root. Furthermore, every vertex lies in some cycle. This makes H' a 2-edge-connected spanning subgraph of H. ■

The next lemma will be used in the proof of the existence of a constant α. Let m be the number of edges in G, and s be a positive integer.

Lemma 7.1.3 *If G has a vertex cover with at most s vertices, then H has a 2-edge-connected spanning subgraph with at most $6m + s$ edges.*

Proof. Suppose G has a vertex cover S with at most s vertices. Consider the spanning subgraph H' of H with all edges incident to vertices labeled a_2 or b_2, all edges labeled u^+ for

$u \in S$, and all edges labeled u^- for $u \notin S$. For example, for the vertex cover $S = \{x, y, v\}$ of the graph G given in Figure 36 (a), the dark edges in Figure 38 (b) are the edges of the graph H'.

Note that H' is a subgraph of H satisfying 1) and 2) of Fact 7.1.2, that is, H' is spanning, all vertices of H' but the root have degree two and H' does not contain, for any edge uv of G, both the edge labeled u^- and the edge labeled v^- in the gadget for uv (because uv is an edge of G, the vertex cover must contain u or v). Thus, H' is a 2-edge-connected spanning subgraph of H.

To count how many edges there are in H', let us prove that H' has exactly six edges per gadget plus one extra edge per vertex in S. Consider a gadget, say, for edge uv of G. By construction, H' has the four edges (with no label) incident to vertices labeled a_2 or b_2 in the gadget. Also, H' has one labeled edge for u in the gadget (either the edge labeled u^+ or the edge u^-) and one labeled edge for v in the gadget (either the edge labeled v^+ or the edge v^-), for a total of exactly six edges in the gadget for uv. Now, consider the edges of H which are not in any gadget. These edges were inserted in the final phase of the construction of H. There is one of these edges for each vertex u of G, from the final current vertex of u in H to the root, with a label u^+. Since H' has all edges labeled u^+ for $u \in S$, H' has exactly $|S| \leq s$ edges that do not appear in gadgets. So H' has at most $6m + s$ edges. ■

The following is an easy consequence of Lemma 7.1.3.

Corollary 7.1.4 $opt_{2-ECSS}(H) \leq 6m + opt_{VC_7}(G)$.

Completing the first part of the L-reduction, the next lemma tells we can set $\alpha = 43$.

Lemma 7.1.5 $opt_{2-ECSS}(H) \leq 43 \cdot opt_{VC_7}(G)$.

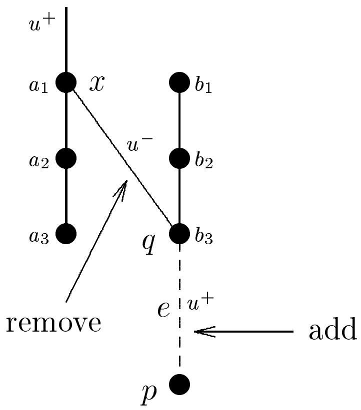

Figure 39: The modification of the gadget for edge uv applied to vertex x.

Proof. Because G has maximum degree at most seven, $opt_{VC_7}(G) \geq m/7$. From this and Corollary 7.1.4, we get

$$opt_{2-ECSS}(H) \leq 6m + opt_{VC_7}(G) \leq (6 \cdot 7 + 1)opt_{VC_7}(G) = 43 \cdot opt_{VC_7}(G). \blacksquare$$

The second part of the L-reduction is the constant β and algorithm g. We may assume that $m \geq 1$. Note that H has $6m + 1$ vertices; therefore any 2-edge-connected spanning subgraph of H must have at least $6m + 1$ edges. Consider a 2-edge-connected spanning subgraph H' of H with $6m + s$ edges, where s is some positive integer. Algorithm g will produce in polynomial time a vertex cover of G of size at most s from H'. As a consequence, $opt_{VC_7}(G) \leq opt_{2-ECSS}(H) - 6m$. Thus, by Corollary 7.1.4, $opt_{2-ECSS}(H) = 6m + opt_{VC_7}(G)$, implying that $\beta = 1$ suffices.

So, let us describe algorithm g, and show that in fact it produces in polynomial time a vertex cover of G of size at most s from H'. In a first phase, g produces another 2-edge-connected spanning subgraph H'' of H with at most as many edges as H', and such that in H'' all vertices but the root have degree two.

To get H'', the gadgets are sequentially checked (in the order specified below) for vertices of degree three. Let x be a vertex other than the root having degree three in H. Note that x has a label in $\{a_1, b_1, a_3, b_3\}$ (because vertices labeled a_2 or b_2 have degree two in H). Vertex x is in the gadget for some edge, say uv of G, where, we may assume, the edge labeled u^- in the gadget is incident to x. If x has degree three in H', then both the edges labeled u^+ and u^- incident to x appear in H'. Remove the edge labeled u^- incident to x whose other endpoint is, say, vertex q. Add (if it is not already there) the edge labeled u^+ incident to q. (See Figure 39.) H'' is the graph obtained after applying this modification to H' while there are vertices other than the root of degree three. Figures 40, 41 and 42 show an example. The dark edges in Figure 40 (a) represent a 2-edge-connected spanning subgraph H' of H.

In a first phase, sequentially apply the modification to all the gadgets in increasing order by gadget number to the vertices labeled a_1 or b_1 (or both). In our example of Figure 40 (a), we mark with a square the first vertex on which the modification is applied. Figure 40 (b) shows the graph resulting after the modification on that vertex. Also in (b), the next vertex on which the modification is applied is shown. Figure 41 (a) shows the resulting graph and the next vertex on which the modification is applied. Figure 41 (b) shows the resulting graph after the modification, and this ends the first phase.

For the second phase, start all over and sequentially apply the modification to all the gadgets in decreasing order by gadget number to the vertices labeled a_3 or b_3 (or both). In Figure 42 (a) we see the first (and last) vertex on which the modification is applied in this phase. In (b) the dark edges indicate the resulting graph H''.

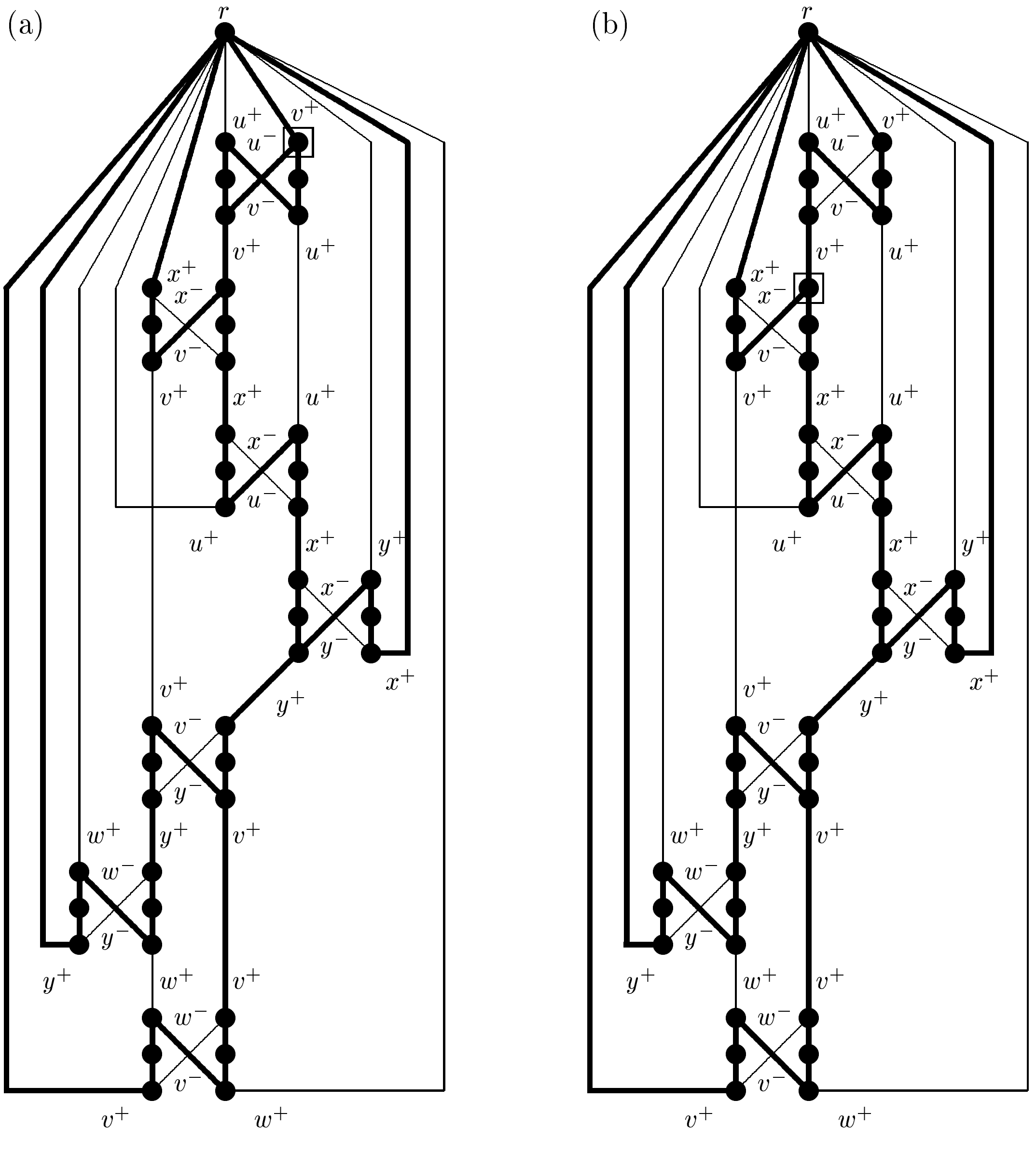

Figure 40: (a) The dark edges show a 2-edge-connected spanning subgraph H' of H with a marked vertex of degree three in H'. (b) The same after the modification is applied to the marked vertex in (a). In this case, no new edge was added, since it was already present.

This procedure takes polynomial time: the modification can be done in polynomial time in the size of G, and after each modification the number of edges whose label has a superscript "$-$" decreases. Clearly H'' has at most as many edges as H'.

Next we prove that H'' is spanning (meaning a spanning subgraph of H) and in H'' all vertices but the root have degree two. We do this in two steps. We start by proving that at the end of the first phase, H' is spanning and all vertices have degree at least two and all vertices labeled a_1 or b_1 have degree exactly two in the current H'. Then, assuming this, we prove that at the end of the second phase, H' is spanning and all vertices but the root have degree exactly two in the current H', which is H''.

Suppose we are in the first phase, just after applying the modification to a vertex in the gadget numbered i. Let us prove, by induction on i, that H' is spanning and all vertices have degree at least two and all vertices labeled a_1 or b_1 in gadgets with a number i or less have degree exactly two in the current H'. The statement is true for $i = 0$ (the first gadget is number one), because, since H' is a 2-edge-connected spanning subgraph of H, all vertices have degree at least two in H'.

Now, for the induction step, consider $i > 0$ and assume the statement is true for $i - 1$. Let us prove the statement holds for i. Consider the moment just before the modification is applied to the gadget (numbered) i. Let uv be the edge of G whose gadget is gadget i. For simplicity, denote the vertices in gadget i by their labels. (Note that all vertices in H but the root have degree at most three, so in H' any vertex other than the root has degree either two or three.) If both vertices a_1 and b_1 have degree exactly two in the current H', then there is nothing to prove. Otherwise, assume a_1 has degree three. The other case (when b_1 has degree three) is symmetric. Let e be the edge incident to b_3 labeled u^+, and let p be the endpoint of e other than b_3 (see Figure 39). Note that p is either the root, or is in a gadget with a number greater than i. When we apply the modification to a_1, the

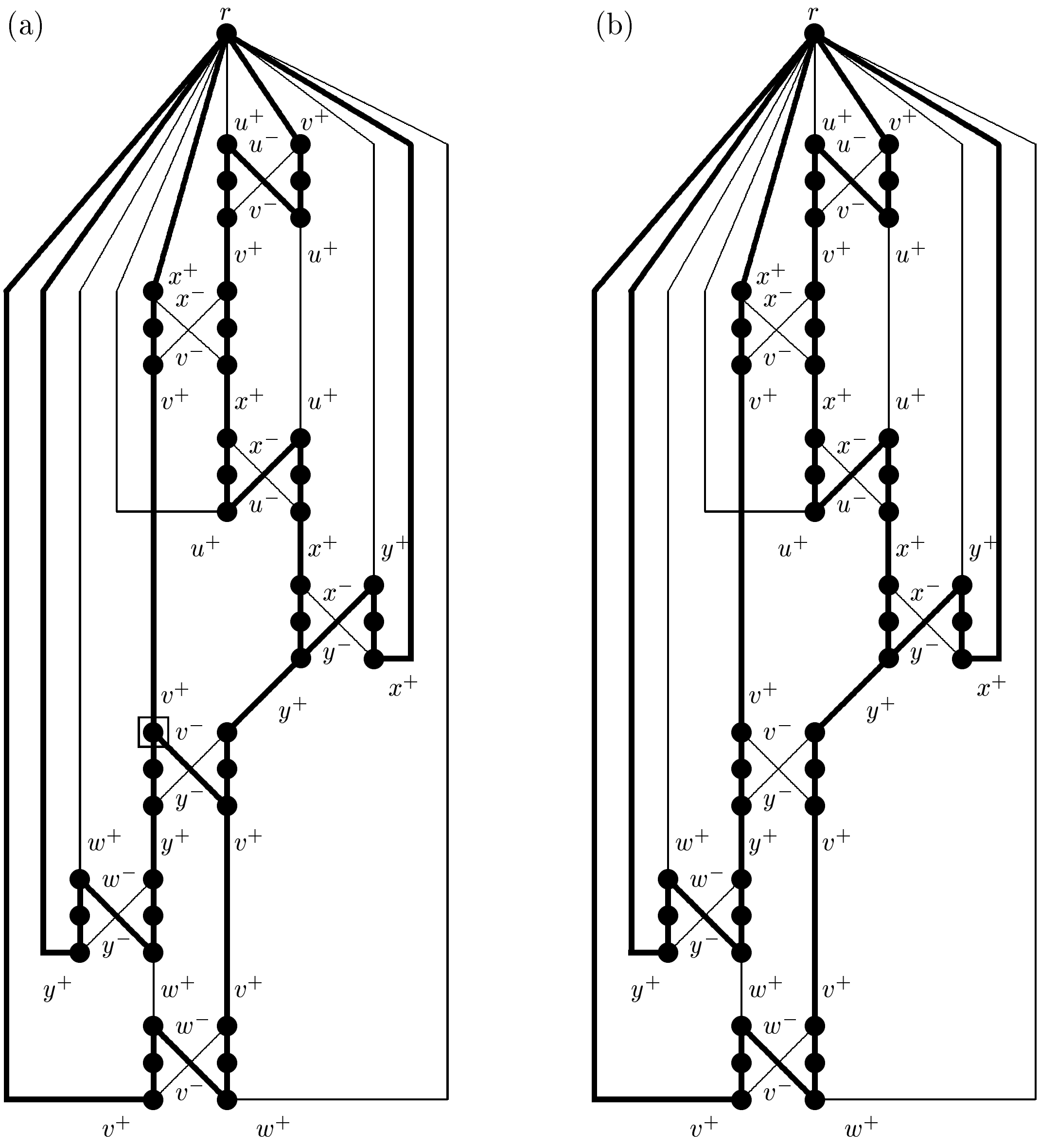

Figure 41: (a) and (b) The dark edges represent graph H' after the modification is applied to the previously marked vertex.

degree of a_1 changes from three to two (because we removed the edge labeled u^- incident to a_1). If e was not in H', then it is added, the degree of p increases by one and the degree of b_3 does not change (it was two, because the edges a_1b_3 and b_3b_2, but not e). If e was already in H', no edge is added, the degree of p does not change and the degree of b_3 goes from three (because the edges a_1b_3, e and b_3b_2 were in H') to two (because we removed e). Therefore, the statement holds after the modification is applied to vertex a_1.

As we said, the case in which b_1 has degree three is symmetric, so we omit its proof. Also, if the modification is applied twice in gadget i, once to a_1 and once to b_1, still the statement holds, because the modification applied to a_1 does not affect the degree of b_1 and vice-versa. The statement for $i = |E(G)|$ (the number of gadgets) says that all vertices have degree at least two and all vertices labeled a_1 or b_1 have degree exactly two in H' at the end of the first phase.

Assuming this, the proof that at the end of the second phase, H' is spanning and all vertices but the root have degree exactly two in $H' = H''$ is similar, so we omit it. From this, we conclude that H'' satisfies 1) of Fact 7.1.2.

Let us verify that H'' also satisfies 2) of Fact 7.1.2. Consider the gadget for edge uv of G and the original H' (before the modifications are applied). Such an H', by assumption, is a 2-edge-connected spanning subgraph of H. So, there must be edges in H' incident to the gadget for uv whose label has a superscript "+", otherwise this gadget would not be connected to any other gadget or to the root, contradicting the fact that H' is connected. No edge of H' whose label has a superscript "+" was removed, so the same holds true for H''. Therefore, either an edge labeled u^+ or an edge labeled v^+ incident to the gadget for uv appears in H''. But in H'' all vertices but the root have degree two. This implies that if an edge labeled u^+ incident to the gadget for uv is in H'' then the edge labeled u^- in this gadget is not. Similarly, if an edge labeled v^+ incident to the gadget for uv is in H''

then the edge labeled v^- in this gadget is not. Since either an edge labeled u^+ or an edge labeled v^+ incident to the gadget for uv is in H'', not both the edge labeled u^- and the edge labeled v^- in the gadget for uv are in H'', that is, 2) of Fact 7.1.2 holds, and so H'' is 2-edge-connected.

Since all vertices in H'' other than the root have degree two, for each vertex u in G, either all edges labeled u^+ and no edge labeled u^- appear in H'', or all edges labeled u^- and no edge labeled u^+ appear in H''. Finally, a vertex cover of size at most s is computed: let S be the set of all vertices u in G such that all edges labeled u^+ (and no edge labeled u^-) appear in H''. (The set S for the example given in Figure 42 is the set $\{x, y, v\}$.) Note that S can be computed from H'' in polynomial time in the size of H'', which is polynomial in the size of G.

Let us prove that S is in fact a vertex cover in G. Recall that H'' satisfies 2) of Fact 7.1.2, that is, for any edge uv in G, H'' does not contain both the edge labeled u^- and the edge labeled v^- in the gadget for uv. If H'' does not contain the edge labeled u^- in the gadget for uv, then $u \in S$. If H'' does not contain the edge labeled v^- in the gadget for uv, then $v \in S$. This proves that S is a vertex cover in G.

Now, let us compute the size of S. Recall that for each vertex u of G, there is exactly one edge in H with a label u^+ not appearing in any gadget. Also, for each vertex u in G, we know that either all edges labeled u^+ and no edge labeled u^- appear in H'', or all edges labeled u^- and no edge labeled u^+ appear in H''. So, to compute the size of S, it is enough to count how many edges not appearing in gadgets there are in H''. For this, recall that H' had $6m + s$ edges. Hence H'' has at most $6m + s$ edges. Since in H'' all vertices but the root have degree two, H'' has exactly six edges in each gadget. Therefore, H'' has at most s edges that are not in gadgets, implying that $|S| \leq s$. This finishes the description of algorithm g, and the proof of Theorem 7.1.1. ∎

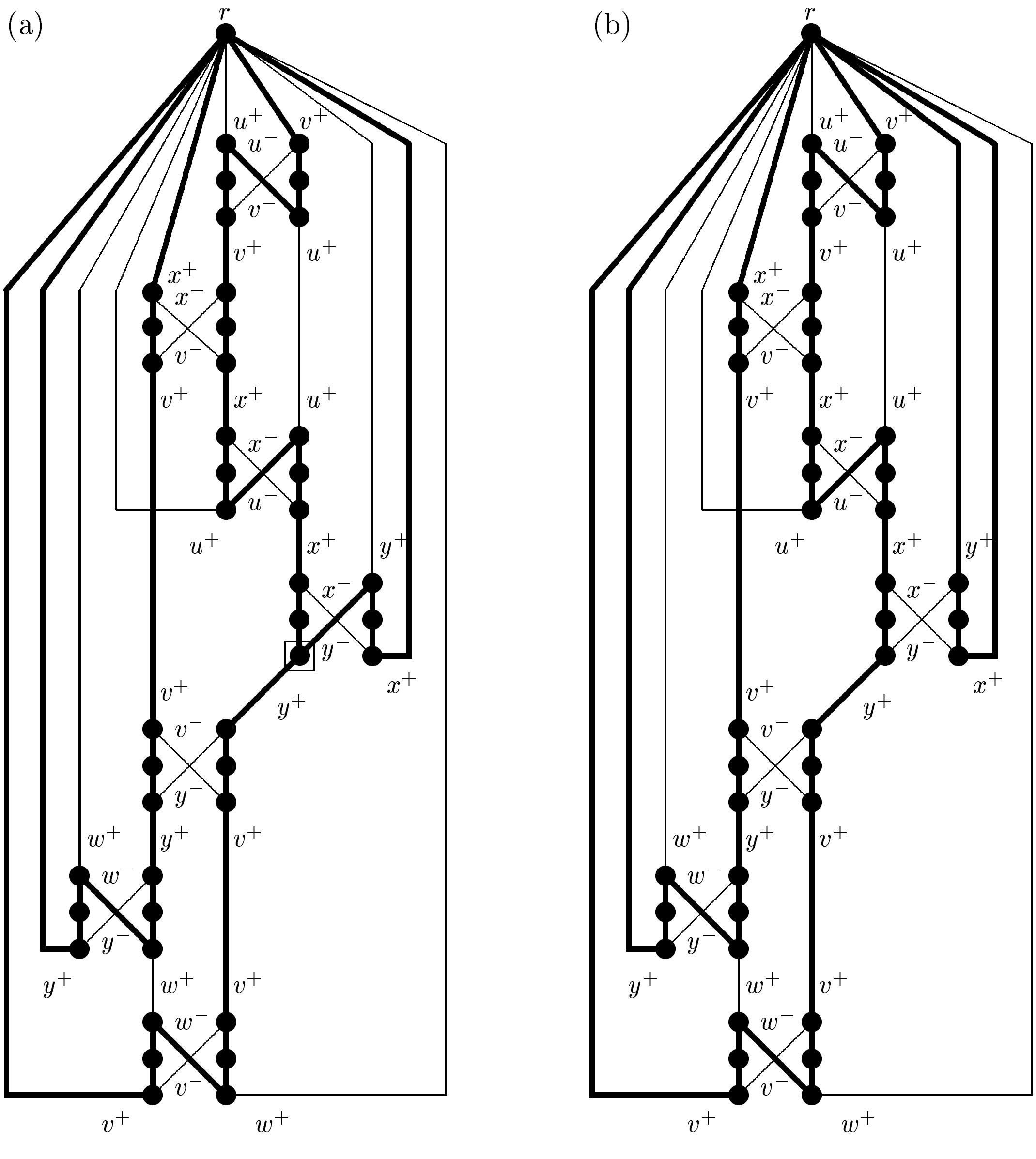

Figure 42: (a) The dark edges represent graph H' after the first phase and the marked vertex is the first (and last) vertex on which the modification is applied in the second phase. (b) The resulting graph H''.

Chapter 8

Conclusions

The focus of this thesis was problems related to two important concepts in graph theory: planarity and connectivity. Specifically, we first discussed the Maximum Planar Subgraph problem, which has applications in facility layout and graph drawing. Second, we examined the k-Edge-Connected Spanning Subgraph problem, which has applications in network reliability and network design.

Both of these problems are known to be NP-complete, which means that an efficient algorithm for them is unlikely to exist. A natural goal, then, is to look for efficient approximation algorithms.

None of the previous approximation algorithms for the Maximum Planar Subgraph problem was known to achieve a performance ratio larger than $1/3$, which is achieved simply by producing a spanning tree. We presented the first efficient approximation algorithm for the problem with higher performance ratio, namely $4/9$.

We applied our algorithm to the Maximum Outerplanar Subgraph problem, achieving a performance ratio of $2/3$ for this problem, which improves on the $1/2$ achieved by producing a spanning tree.

In terms of complexity, we showed that the Maximum Planar Subgraph problem and the Nonplanar Deletion problem are Max SNP-hard. This guarantees that there is a small $\epsilon > 0$ such that the existence of a polynomial-time approximation algorithm with performance ratio at least $1 - \epsilon$ for the first, or at most $1 + \epsilon$ for the second one, implies P=NP.

In [4], we apply the ideas of our algorithm to obtain an approximation algorithm for the following problem: given a graph G and a positive integer g, find a subgraph of G of maximum size and embeddable in a surface of genus g.

In the second part of this thesis, we investigated the Minimum k-Edge-Connected Spanning Subgraph problem. Khuller and Raghavachari [41] presented the first algorithm for the Minimum k-Edge-Connected Spanning Subgraph problem with a performance ratio smaller than a constant smaller than two, for all k. They proved an upper bound of 1.85 for the performance ratio of their algorithm.

This thesis has two main contributions with respect to this problem. First, the improvement on the upper bound for the performance ratio of Khuller and Raghavachari's algorithm from 1.85 to 1.75. Second, the problem was proved to be Max SNP-hard for $k = 2$.

We also generalized the concept of a tree-carving, and used it to prove that the performance ratio of Khuller and Raghavachari's algorithm is less than 1.7 for large enough k.

Our analysis actually improves Khuller and Raghavachari's bound for any fixed $k \geq 4$. In particular, we get 1.65 for $k = 4$, and 1.68 for $k = 5$, the bound on $k = 5$ being tight.

8.1 Open Problems

Several interesting questions arose while working on the Maximum Planar Subgraph problem. The ϵ given by the Max SNP-hardness proof is very small, so there is a huge gap between the best conceivably achievable performance ratio and the best known. Can one close this gap? A new algorithm with a better performance ratio would at least decrease this gap.

Consider the the following *randomized algorithm* for the Maximum Planar Subgraph problem: start with the edge set $S =$. Randomly choose a new edge of the input graph $G = (V, E)$ and add it to S if (V, S) remains planar. Repeat until all edges of G have been chosen. Output the final graph (V, S). Is the performance ratio of this randomized algorithm greater than $1/3$? (We know it is at least $1/3$.)

A *maximal* planar subgraph can be found in linear time [36, 15], which means a ratio of $1/3$ can be achieved in linear time. We exhibited in this thesis a greedy version of our algorithm that can be implemented in linear time on bounded degree graphs. It would be interesting to develop a linear-time algorithm whose performance ratio is $1/3 + \epsilon$, for some $\epsilon > 0$.

In contrast with the Maximum Planar Subgraph problem, no α-approximation algorithm is known for the Nonplanar Deletion problem, where α is a constant. An obvious open problem is whether such an algorithm exists or not.

In [7], we present a new algorithm for the weighted version of the Maximum Planar Subgraph problem with a performance ratio of at least $1/3 + 1/72$. This is the best known performance ratio for the weighted case, so there is a gap between the performance ratio for the weighted and the unweighted cases. Can one develop an algorithm with a better performance ratio for the weighted case, closing this gap? For finding heavy outerplanar subgraphs, no ratio better than $1/2$ is known, to the author's knowledge. A better ratio here also would be interesting. The algorithm in [7] builds a heavy triangular structure in the input graph. It is not known whether finding a maximum weight triangular structure is NP-hard. A more general question is if the Weighted Graphic Matroid Parity problem is NP-hard or not.

Still related to planarity is the Thickness problem: given a graph G, find a partition of the edges of G into as few planar subgraphs as possible. This problem has applications in circuit layout and is also known to be NP-complete, so good approximations would be of interest. To achieve a ratio of three here is trivial, via arboricity. Can one, maybe using the ideas of our algorithm, develop an algorithm with a better performance ratio for this problem?

The Minimum k-Edge-Connected Spanning Subgraph problem has four natural versions, by choosing between the directed or the undirected case, and the vertex-connectivity and the edge-connectivity cases. Cheriyan and Thurimella [9] presented a new algorithm for the vertex-connectivity version of the problem. Their algorithm achieves a performance ratio of $1 + 1/k$. They also exhibit a variant of their algorithm for the edge-connectivity case, and prove an approximation ratio of $1+2/(k+1)$ in the undirected case (the Minimum k-Edge-Connected Spanning Subgraph problem), and of $1 + 4/\sqrt{k}$ in the directed case. These are the best known performance ratios for each of these problems in general. Can one do better in any of these cases? We believe that our analysis significantly contributes to a better understanding of the structure of the output of Khuller and Raghavachari's algorithm. Hopefully, this will help in the development of new algorithms with better performance ratios for the Minimum k-Edge-Connected Spanning Subgraph problem, or even one of its variants.

One can be interested in the best performance ratio for particular values of k. In this case, the state of the art is as follows. For the Minimum k-Edge-Connected Spanning Subgraph problem, the best known is 1.5 for $k = 2$ and Cheriyan and Thurimella's bound $1 + 2/(k + 1)$, for $k \geq 3$ [9]. For the vertex-connectivity case, the best known performance ratio is $1 + 1/k$ [9, 29]: 1.5 for $k = 2$, $4/3 = 1.33...$ for $k = 3$, 1.25 for $k = 4$, etc. In the directed case, the best known ratio for $k = 1$ is $\phi^2/6 - 1/36 \approx 1.617$ [42]. For $k > 2$ and vertex-connectivity case, the best known ratio is $1+1/k$ [9]. For the edge-connectivity directed case and $2 \leq k \leq 16$, the best known performance ratio is still two! For $k \geq 17$, Cheriyan and Thurimella's bound is the best known: $1 + 4/\sqrt{k}$ [9].

Bibliography

[1] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof Verification and Hardness of Approximation Problems," *Proc. 33^{rd} IEEE Symposium on Foundations of Computer Science*, 14–23, 1992.

[2] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, MacMillan Press, 1976.

[3] J. Cai, X. Han and R. E. Tarjan, "An $O(m \log n)$-time Algorithm for the Maximal Planar Subgraph Problem," *SIAM Journal on Computing*, 22:1142–1162, 1993.

[4] G. Călinescu and C. G. Fernandes, "Finding Large Planar Subgraphs and Large Subgraphs of a Given Genus," *Proc. 2nd International Computing and Combinatorics Conference*, 152–161, 1996.

[5] G. Călinescu, C. G. Fernandes, U. Finkler and H. Karloff, "A Better Approximation Algorithm for Finding Planar Subgraphs," *Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, 16–25, 1996.

[6] G. Călinescu, C. G. Fernandes, U. Finkler and H. Karloff, "A Better Approximation Algorithm for Finding Planar Subgraphs," submitted to the *Journal of Algorithms*, 1996.

[7] G. Călinescu, C. G. Fernandes, H. Karloff and A. Zelikovsky, "A New Approximation Algorithm for Finding Heavy Planar Subgraphs," submitted to the *Symposium on Foundations of Computer Science*, 1997.

[8] J. Cheriyan, M. -Y. Kao and R. Thurimella, "Algorithms for Parallel k-vertex Connectivity and Sparse Certificates," *SIAM Journal of Computing*, 22 (1), 157–174, 1993.

[9] J. Cheriyan and R. Thurimella, "Approximating Minimum-size k-Connected Spanning Subgraphs via Matching," *Proc. 37rd IEEE Symposium on Foundations of Computer Science*, October 1996.

[10] T. Chiba, I. Nishioka and I. Shirakawa. "An Algorithm for Maximum Planarization of Graphs," *Proc. IEEE Symp. on Circuits and Systems*, 649–652, 1979.

[11] N. Chiba and T. Nishizeki, "Arboricity and Subgraph Listing Algorithms," *SIAM Journal of Computing*, 14:210–223, 1985.

[12] R. Cimikowski, "An Analysis of Some Heuristics for the Maximum Planar Subgraph Problem," *Proc. 6th Annual ACM-SIAM Symp. on Discrete Algorithms*, 322–331, 1995.

[13] S. A. Cook, "The Complexity of Theorem-Proving Procedures," *Proc. 3rd Annual ACM Symp. on Theory of Computing*, 151–158, 1971.

[14] P. Crescenzi, R. Silvestri and L. Trevisan, "To Weight or Not to Weight: Where is the Question?", *Proc. 4th Israel Symposium on Theory of Computing and Systems*, 68–77, 1996.

[15] H. N. Djidjev, "A Linear Algorithm for the Maximal Planar Subgraph Problem," *Proc. 4th International Workshop on Algorithms and Data Structures* (WADS '95), 369–380, 1995.

[16] M. E. Dyer, L. R. Foulds and A. M. Frieze, "Analysis of Heuristics for Finding a Maximum Weight Planar Subgraph," *European Journal of Operational Research*, 20:102–114, 1985.

[17] J. Edmonds, "Edge-Disjoint Branchings," in *Combinatorial Algorithms*, R. Rustin (ed.), Algorithmic Press, New York, 91–96, 1972.

[18] L. Euler, "Demonstratio Nonnullarum Insignium Proprietatum Quibus Solida Hedris Planis Inclusa Sund Praedita," *Nove Comm. Acad. Sci. Imp. Petropol.* 4, 140–160, Opera Omnia (1), 1752-3, published 1758.

[19] R. Fagin, "Generalized First-Order Spectra and Polynomial-Time Recognizable Sets," in *Complexity of Computer Computations*, R. Karp (ed.), 43–77, 1974.

[20] U. Feige and M. X. Goemans, "Approximating the Value of Two-Prover Proof Systems, with Applications to MAX 2SAT and MAX DICUT," *Proc. 3rd Israel Symposium on the Theory of Computing and Systems*, 182–189, 1995.

[21] C. G. Fernandes, "A Better Approximation Ratio for the Minimum k-edge-connected Spanning Subgraph Problem," *Proc.* 8^{th} *Annual ACM-SIAM Symp. on Discrete Algorithms*, 629–638, 1997.

[22] C. G. Fernandes, "Finding k-Edge-Connected Spanning Subgraphs: a Better Analysis," submitted to the *Journal of Algorithms*, 1996.

[23] L. R. Foulds, *Graph Theory Applications*, Springer-Verlag, New York, 1992.

[24] A. Frank, T. Ibaraki and H. Nagamochi, "On Sparse Subgraphs Preserving Connectivity Properties," *Journal of Graph Theory*, 17:275–281, 1993.

[25] H. N. Gabow, M. X. Goemans and D. P. Williamson, "An Efficient Approximation Algorithm for the Survivable Network Design Problem," *Proc.* 3^{rd} *Integer Programming and Combinatorial Optimization Conference*, 57–74, 1993.

[26] H. N. Gabow and M. Stallmann, "Efficient Algorithms for Graphic Matroid Intersection and Parity," *Automata, Language and Programming:* 12^{th} *Colloq., Lecture Notes in Computer Science*, Vol. 194, 210–220, 1985.

[27] M. R. Garey, R. L. Graham and J. D. Ullman, "Worst Case Analysis of Memory Allocation Algorithms," *Proc.* 4^{th} *Annual ACM Symposium on the Theory of Computing*, 143–150, 1972.

[28] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Co., 1979.

[29] N. Garg, V. Santosh and A. Singla, "Improved Approximation Algorithms for Biconnected Subgraphs via Better Lower Bounding Techniques," *Proc. 4^{th} Annual ACM-SIAM Symposium on Discrete Algorithms*, 103–111, 1993.

[30] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. Shmoys, E. Tardos and D. P. Williamson, "Improved Approximation Algorithms for Network Design Problems," *Proc. 5^{th} Annual ACM-SIAM Symposium on Discrete Algorithms*, 223–232, 1994.

[31] M. X. Goemans and D. P. Williamson, "A General Approximation Technique for Constrained Forest Problems," *SIAM Journal of Computing*, 24 (2), 296–317, 1995.

[32] M. X. Goemans and D. P. Williamson, "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming," *Journal of the Association for Computing Machinery* 42:6, 1115-1145, 1995.

[33] F. Harary, *Graph Theory*, Addison-Wesley Publishing Company, 1972.

[34] D. Hochbaum, "Approximation Algorithms for the Set Covering and Vertex Covering Problems," *SIAM Journal of Computing* 11:555–556, 1982.

[35] D. Hochbaum, *Approximation Algorithms for NP-Hard Problems*, PWS Publications, 1997.

[36] W. Hsu, "A Linear Time Algorithm for Finding Maximal Planar Subgraphs," *Proc. 6^{th} Annual International Symposium on Algorithms and Computation* (ISAAC95), 1995.

[37] D. S. Johnson, "Approximation Algorithms for Combinatorial Problems," *Journal of Computer and System Sciences*, 9:256–278, 1974.

[38] M. Jünger and P. Mutzel, "Maximum Planar Subgraph and Nice Embeddings: Practical Layout Tools," *Universität zu Köln, Report No. 93.145*, 1993, to appear in Special Issue of *Algorithmica* on Graph Drawing.

[39] D. Karger, "Random Sampling in Cut, Flow, and Network Design Problems," *Proc. 26^{th} Annual ACM Symposium on the Theory of Computing*, 648–657, 1994.

[40] R. M. Karp, "Reducibility Among Combinatorial Problems," *Complexity of Computer Computations*, 85–103, 1972.

[41] S. Khuller and B. Raghavachari, "Improved Approximation Algorithms for Uniform Connectivity Problems," *Journal of Algorithms*, 21, 434–450, 1996.

[42] S. Khuller, B. Raghavachari and N. Young, "Approximating the Minimum Equivalent Digraph," *SIAM Journal of Computing*, 24 (4), 859–872, 1995.

[43] S. Khuller and U. Vishkin, "Biconnectivity Approximations and Graph Carvings," *J. Assoc. Comput. Mach.*, 41 (2), 214–235, 1994.

[44] C. Kuratowski, "Sur le Problème des Courbes Gauches en Topologie," *Fund. Math.*, 15:271-283, 1930.

[45] P. C. Liu and R. C. Geldmacher, "On the Deletion of Nonplanar Edges of a Graph," *Proc.* 10^{th} *Southeastern Conference on Combinatorics, Graph Theory, and Computing*, 727–738, 1977.

[46] L. Lovász and M. D. Plummer, *Matching Theory*, Elsevier Science, Amsterdam, 1986.

[47] W. Mader, "Minimale n-Fach Kantenzusammenhängende Graphen," *Math. Ann.*, 21–28, 1971.

[48] W. Mader, "Ecken vom Grad n in Minimalen n-Fach Zusammenhängenden Graphen," *Archive der Mathematik*, 23:219–224, 1972.

[49] W. Mader, "Minimal n-Fach Zusammenhängenden Digraphen," *Journal of Combinatorial Theory Series B*, 38:102–224, 1985.

[50] H. Nagamochi and T. Ibaraki, "Linear-time Algorithm for Finding a Sparse k-connected Spanning Subgraph of a k-connected Graph," *Algorithmica*, 7 (5-6), 583–596, 1992.

[51] C. H. Papadimitriou and M. Yannakakis. "Optimization, Approximation, and Complexity Classes," *Journal of Computer and System Sciences*, 43:425–440, 1991.

[52] C. H. Papadimitriou and M. Yannakakis, "The Traveling Salesman Problem with Distances One and Two," *Mathematics of Operations Research*, 18 (1), 1–11, 1993.

[53] D. B. Shmoys, "Computing Near-Optimal Solutions to Combinatorial Optimization Problems," in *Combinatorial Optimization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science (20), W. Cook and L. Lovász (eds.), AMS Publications, 355–397, 1995.

[54] R. Tamassia, G. Di Battista and C. Batini, "Automatic Graph Drawing and Readability of Diagrams," *IEEE Transactions on Systems, Man and Cybernetics*, 18:61–79, 1988.

[55] R. Thurimella, "Techniques for the Design of Parallel Graph Algorithms," Ph.D. Thesis, The University of Texas at Austin, August 1989.

Vita

Cristina Gomes Fernandes was born in February 26, 1966, in Santos, a nice city on the coast of Brazil. She went to school in Santos, and graduated from high school in 1983.

In 1984, she moved to São Paulo, Brazil, to go to the University of São Paulo. She graduated with a Bachelor's in Computer Science in 1987. From 1988 to 1992, she worked as a lecturer for the University of São Paulo. In 1992, she got her Master's degree from the University of São Paulo.

In September 1992, she came to Atlanta, to join the Algorithms, Combinatorics and Optimization PhD program at the Georgia Institute of Technology. Since then, she has lived in Atlanta.

In August 1997, she will move back to Brazil and reassume her position in the Department of Computer Science at the University of São Paulo, Brazil.