AN ABSTRACT OF THE DISSERTATION OF

Baigong Zheng for the degree of Doctor of Philosophy in Computer Science presented on November 7, 2018.

Title: Approximation Schemes in Planar Graphs

Abstract approved: _____

Glencora Borradaile

There are growing interests in designing polynomial-time approximation schemes (PTAS) for optimization problems in planar graphs. Many NP-hard problems are shown to admit PTAS in planar graphs in the last decade, including Steiner tree, Steiner forest, two-edge-connected subgraphs and so on. We follow this research line and study several NP-hard problems in planar graphs, including minimum three-vertex-connected spanning subgraph problem, minimum three-edge-connected spanning subgraph problem, relaxed minimum-weight subset three-edge-connected subgraph problem and minimum feedback vertex set problem. For the first three problems, we give the first PTAS results, and for the last problem, we give a PTAS result based on local search and a practical heuristic algorithm that provides a trade-off between running time and solution quality like a PTAS.

Approximation Schemes in Planar Graphs

by

Baigong Zheng

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented November 7, 2018
Commencement June 2019

Doctor of Philosophy dissertation of Baigong Zheng presented on November 7, 2018.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Baigong Zheng, Author

ACKNOWLEDGEMENTS

I would like to thank my parents, my advisor Dr. Glencora Borradaile and my colleague Dr. Hung Le.

CONTRIBUTION OF AUTHORS

Dr. Hung Le was involved with the design and analysis of the local search algorithm in Chapter 4, and assisted the implementation of the balanced separator part in Chapter 5.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF ALGORITHMS

Chapter 1: Introduction

1.1 Graphs

A *graph* in this thesis is a kind of mathematical structure, which consists of two parts: a set of *vertices* and a set of *edges*. Each edge connects two vertices; the two vertices may be identical, and in this case the edge is called a *loop*. The study of graphs began with Leonhard Euler's paper on the Seven Bridges of Königsberg in 1736 [13], and this branch of mathematics is called *graph theory*.

Graphs can be used to model different types of structures in the areas of computer science, linguistics, chemistry, sociology and biology. For example, in computer science we can represent the link structure of a website by a graph, where the vertices represent the web pages and each (directed) edge represents a link from one page to another; in computational linguistics we can represent the syntactic structure of a sentence by a special kind of graphs, called *trees*, where the vertices indicate the words in the sentence and the edges are defined by some context-free grammar; in chemistry we can model a molecule by a graph where vertices and edges correspond to atoms and bonds; in sociology graphs can be used to model people's relationships to understand some social issues; and, in biology we can model the physical interactions between an organism's proteins by a graph, where proteins will be the vertices.

1.2 Planar Graphs

A graph is *planar* if it can be drawn on the plane such that its edges intersect only at their endpoints. Kuratowski [92] provides a characterization of planar graphs in terms of *subdivision* (that is to insert vertices into edges):

Theorem 1.1 (Kuratowski's Theorem)**.** *A finite graph is planar if and only if it does not contain a subgraph as a subdivision of the complete graph K_5 or the complete bipartite graph $K_{3,3}$.*

After that, Wagner [117] gives a characterization of planar graphs in terms of *minor* (that is a graph results from edge deletion, edge contraction and vertex deletion from the original graph):

Theorem 1.2 (Wagner's Theorem)**.** *A finite graph is planar if and only if it does not have K_5 or $K_{3,3}$ as a minor.*

There is a long tradition of the research on optimization problems in planar graphs. People are interested in this theme for three main reasons. First, many optimization problems in planar graphs arise in distinct application areas. These include VLSI, image processing problems and geographic problems such as travelling time in a road map and network design problems. Second, people find that exploring the planarity of graphs can result in faster and more accurate algorithms. For example, the best algorithm for maximum s-t flow problem in general graphs runs in $O(mn)$ time by combining the algorithms of King et al. [85] and Orlin [104], where m and n are the numbers of edges and vertices in the graph, while in planar graphs there is an $O(n \log n)$ algorithm for this problem by Borradaile and Klein [21]. Third, the study on problems in planar graphs drives many interesting algorithmic techniques.

1.3 Polynomial-time Approximation Schemes

A polynomial-time approximation scheme (PTAS) is an algorithm that for any given constant $\epsilon > 0$, finds a $(1+\epsilon)$-approximation for the minimization problem (or a $(1-\epsilon)$-approximation for maximization problem) in polynomial time. Papadimitriou and Yannakakis [105] proved that there are problems that do not have PTAS unless $P = NP$, one of which is the maximum independent set. However, Lipton and Tarjan [96] gave a PTAS for this problem in planar graphs, and this is also the first PTAS for an NP-hard problem in planar graphs. After that, more problems have been shown to admit PTAS in planar graphs, including minimum vertex cover [6], travelling salesman problem (TSP) [4, 88], Steiner tree [23], and two-edge-connected spanning subgraph [12]. Further, with the development of algorithmic techniques, more efficient PTASes have appeared. An approximation scheme is *efficient* if its running time is a polynomial whose degree is fixed and independent of the error parameter ϵ. This kind of improvement goes back to the work of Baker [6], who introduced the shifting technique, now known as Baker's

technique.

In the following sections, we briefly review the commonly used approaches to design PTASes for NP-hard problems in planar graphs, including the balanced-separator technique, Baker's shifting technique, the bidimensionality theory, the local search method, and Klein's spanner framework. To illustrate these techniques, we will use maximum independent set as an example problem (except for the spanner framework). For completeness, we summarize many planar PTAS results based on these approaches in Table 1.1 (where the work of this thesis is also highlighted).

1.3.1 Balanced Separator

A separator is a set of vertices whose removal separates the remaining vertices into two disjoint parts such that there is no edge between these two parts. We say a separator is *balanced* if every part has size at most a constant fraction of the size of the original graph. This can be generalized to the setting where vertices are weighted. Lipton and Tarjan [95] first proved there is a balanced separator of size $O(\sqrt{n})$ for any n-vertex planar graph, and this can be found in linear time. See Figure 1.1.

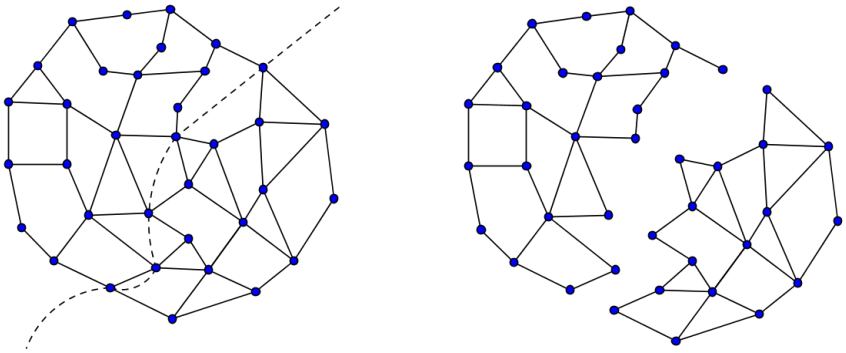

Figure 1.1: A balanced separator. Left: the dashed curve passes through a separator. Right: the separator divides the graph into two parts.

Planar separators are a powerful tool that has ample algorithmic applications, including data structures for dynamic graph algorithms [50], faster algorithms for shortest path in planar graphs [71], and construction of nearest neighbor graphs [60]. The first PTAS for maximum independent set in planar graphs [96], TSP in edge-weighted planar

Table 1.1: Summary of PTAS results in planar graphs. P stands for PTAS and E stands for EPTAS. U stands for "unweighted version of the problem" and W stands for "weighted version of the problem". Our work is highlighted.

Problem	Approaches				
	Separator	Shifting	Spanner	Local Search	Bidimensionality
Independent Set	EU [96]	EW [6]		PU [29]	EU [55]
Vertex Cover		EW [6]		PU [29]	EU [55]
Connected Vertex Cover	PW [34]				EU [55]
Dominating Set		EW [6]		PU [29]	EU [55]
Connected Dominating Set	PW [34]				EU [55]
Feedback Vertex Set	PW [34] EU [25]	EU [25]		PU [94]	EU [55]
TSP	PW [4]		EW [88]		
Steiner Tree			EW [23]		
Steiner Forest			EW [8, 49]		
Planar Group Steiner Tree			EW [7]		
Two-Edge-Connected Spanning Subgraph	PU [37]		PW [12]		
Relaxed Two-Edge-Connected Subset Subgraph			EW [22]		
Three-Edge-Connected Spanning Subgraph		EU [122]			
Relaxed Three-Edge-Connected Subset Subgraph			EW [26]		
k-Means				P [35]	
k-Median				P [35]	
Uncapacitated Facility Location				PU [35]	

graphs [4] and minimum-weight connected dominating set [34] are all based on balanced separators. In these algorithms, balanced separator is used in a recursive way to decompose the original graph into small pieces in which the target problem can be computed efficiently. The error in these algorithms is usually caused by the separators, so if we can bound that by an ϵ-fraction of an optimal solution, we can obtain a PTAS for the target problem.

Consider the maximum independent set problem in a planar graph G. We can obtain a set of small pieces by recursively applying balanced separators. The total size of the separators P can be bounded by an $\frac{\epsilon}{4}$-fraction of the size of G if we bound the size of each piece by a function of ϵ. We can solve the problem in each piece by exhaustive search and obtain a union S of those solutions, which will be the output of the algorithm. This union S is an independent set for graph G, so we only need to show its size is at least $(1-\epsilon)$-fraction of an optimal solution $OPT(G)$. Since S is the union of optimal solutions on each piece, we know that $|S| \geq |OPT(G) \setminus P| \geq |OPT(G)| - |P|$. By the four-color theorem [110], the size of a maximum independent set in G is at least one fourth of the size of G. So the total size of the separator can be bounded by an ϵ-fraction of $|OPT(G)|$, implying that $|S| \geq (1-\epsilon)|OPT(G)|$.

1.3.2 Shifting Technique

Shifting technique is introduced by Baker [6], so it is also known as Baker's technique. By this technique, Baker gives EPTAS for a set of problems, including maximum independent set, minimum vertex cover, minimum dominating set and so on. This technique first partition the vertices into levels according to their distances to the outer boundary, then group constant number (depending on ϵ) adjacent levels into a piece. See Figure 1.2 for an example. Baker proved that the *treewidth* of each piece can be bounded by a function of the number of its levels, which only depends on ϵ. Since many NP-hard problems can be solved optimally in polynomial time in a graph of bounded treewidth, we can compute the optimal solution in each piece efficiently.

When grouping the levels into pieces, two pieces can share one or two levels depending on the specific problems, and this introduces error into the solution. By the pigeonhole principle, there exists a way to group levels such that the shared levels only contain an ϵ fraction of the optimal solution. Since there are at most constant number of ways to

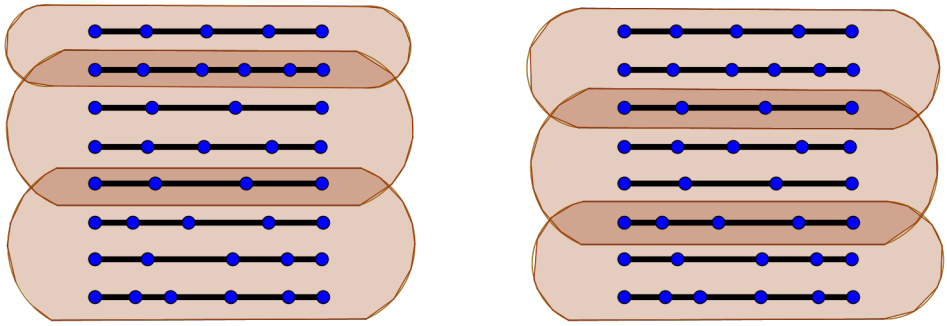

Figure 1.2: Example for shifting technique. The horizontal lines represent levels and the shaded regions represent the pieces. Each piece has at most 4 levels. Two adjacent pieces share one level. The pieces in the right figure can be obtained by shifting the pieces in the left figure once.

group those levels, we can try all the possible ways and output the best solution among them. By grouping more levels into a piece, the error will decrease but the running time will increase, resulting in the trade-off between the precision and running time.

We can illustrate this technique by the maximum independent set problem. If we want to obtain a $(1-\epsilon)$-approximation solution for this problem in graph G, then we will group $k = \lceil 1/\epsilon \rceil + 1$ adjacent levels into a piece such that two adjacent pieces share one level. Since there are only k levels in each piece, we can obtain $k-1$ distinct ways of grouping. For each way of grouping, we will remove all the shared levels and then solve the problem in each remaining piece by dynamic programming efficiently (since the treewidth of such a piece is at most a function of k), which gives us a union of those solutions. The output will be the minimum union from different groupings. Since the error of this algorithm is caused by those removed levels, we only need to show there is one way of grouping such that the shared levels contain at most an ϵ-fraction of an optimal solution, implying that the corresponding union of solutions is at least $(1-\epsilon)$-fraction of the optimal solution. Note that there are $k-1$ ways to group the pieces, and each way defines a distinct set of shared levels. So all the $k-1 = \lceil 1/\epsilon \rceil$ different sets of shared levels define a partition of graph G. By the pigeonhole principle, there exists one set of shared levels that contains at most an ϵ-fraction of an optimal solution in G.

1.3.3 Bidimensionality

Bidimensionality strengthens the separator technique and shifting technique, and it can be applied in more general families of graphs such as H-minor free graphs. It was first introduced by Demaine et al. [40] to design sub-exponential algorithms for problems in H-minor free graphs. Demaine and Hajiaghayi [42] extended this theory to design PTASes for bidimensional problems, including connected dominating set and feedback vertex set. Later, Fomin et al. [55] redesigned the framework to obtain EPTASes. There are two kinds of bidimensional problems: minor-bidimensional problems (such as vertex cover and feedback vertex set) and contraction-bidimensional problems (such as dominating set and independent set).

A graph is a *minor* of a graph G if it can be obtained from G by a sequence of edge contractions, edge deletions and vertex deletions. A problem is *minor-bidimensional* if it satisfies the following:

(1) For any minor H of G, the size of an optimal solution in H cannot be larger than that in G.

(2) There is a constant c such that the size of an optimal solution in grid g_t is at least ct^2.

A graph is a *contraction* of a graph G if it can be obtained fro G by a sequence of edge contractions. To define the contraction-bidimensionality, we need to define a triangulated grid Γ_t based on grid g_t. Let Γ_t be the graph obtained from g_t by first triangulating all internal faces of g_t such that all internal vertices have degree 6 and all non-corner boundary vertices have degree 4, and then one corner vertex of degree two is connected to all boundary vertices. One example Γ_4 is given in Figure 1.3. A problem is *contraction-bidimensional* if it satisfies the following:

(1) For any contraction H of G, the size of an optimal solution in H cannot be larger than that in G.

(2) There is a constant c such that the size of an optimal solution in graph Γ_t is at least ct^2.

To obtain PTAS for a bidimensional problem, we need one additional property: the *separation property*. Assume we can partition a graph G into three parts L, R and S

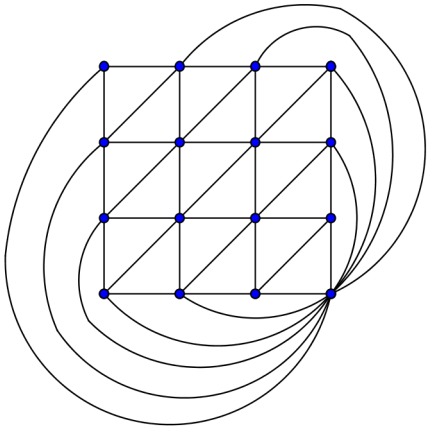

Figure 1.3: Graph Γ_4.

such that there is no edge between L and R. Then the separation property says the optimal solution in L (or R) cannot be larger than the remaining part of the global optimal solution in L (or R) plus the size of S. Intuitively, this property means when we partition the graph, the sum of optimal solutions in all parts can be bounded by the global optimal solution plus constant times of the separator set.

If a problem is bidimensional and has the separation property, then it can be shown that given a graph G and an $\epsilon > 0$, there is a set of vertices X such that (1) removing X from G results in a graph with treewidth bounded by a function of ϵ and (2) the size of X is bounded by an ϵ fraction of the size of an optimal solution. Further, such a set X can be computed in polynomial time (based on a constant approximation solution of the problem or a sublinear treewidth bound). The algorithm starts by computing such vertex set X, and then removes it to obtain a bounded treewidth graph. After solving the problem in the resulting graph by dynamic programming, the algorithm combine the solution and the set X to obtain its final solution. By removing more vertices, the error, which is the size of set X, will increase but the treewidth will decrease, resulting in a trade-off between the precision and running time.

Consider the maximum independent set problem in a planar graph G as an example. The algorithm first computes a set of vertices S, whose size is at most a constant times of the size of an optimal solution, such that removing S results in a graph with constant treewidth. Based on this S, the algorithm can find the set X which can be bounded

by a small fraction of S. Then this X will be removed from G to obtain a graph with bounded treewidth, in which we can solve the problem efficiently. The output will be the solution obtained from this new graph. The error is caused by the set X, which can be bounded by a small fraction of the size of S and then an ϵ-fraction of the size of an optimal solution.

1.3.4 Local Search

Local search was first used to obtain PTASes for geometric problems (e.g. Chan and Har-Peled [30], and Mustafa and Ray [102]). Cabello and Gajser [29] gave the first PTAS by local search for optimization problems in H-minor-free graphs, including maximum independent set, minimum vertex cover and minimum dominating set. Cohen-Addad, Klein and Mathieu [35] showed that local search yields PTAS for k-means, k-median and uniform uncapacitated facility location in H-minor-free graphs.

Algorithm 1 PTAS by Local Search

Input: an instance and a constant c
Let S be an arbitrary solution.
while there is a solution S' such that $|S \setminus S'| \leq c$, $|S' \setminus S| \leq c$ and $cost(S') < cost(S)$ **do**
    $S = S'$
return S

Different from previous approaches, the PTASes by local search are often simple to implement and do not rely on planarity. See Algorithm 1 for a prototypical local search algorithm. Given a constant $c = f(\epsilon)$ depending on ϵ, the algorithm starts with an arbitrary solution, and tries to improve this solution by changing at most c vertices or edges. If it cannot improve the solution in this way, then the current solution is output. Since the algorithm needs to check all the combinations of the c vertices or edges, its running time is $O(n^{f(\epsilon)})$, so not efficient.

The underlying tool for analyzing the PTAS by local search is an *r-division*, which is closely related to balanced separators and in fact obtained by applying balanced separators recursively. Given a constant r, an r-division decomposes the graph into a set of regions such that each region has at most r vertices, each region has at most $O(\sqrt{r})$ boundary vertices, and the number of boundary vertices, summing over all regions in

an r-division, is bounded by $O(n/\sqrt{r})$, where n is the number of vertices in the graph. This concept is due to Frederickson [54] in the context of planar graphs, but it is easy to extend it to any family of graphs that has sublinear-size balanced separators. The analysis for the PTAS by local search starts from constructing a graph, called *exchange graph*, which is usually a minor of the original graph, consisting of vertices in the two solutions: the optimal solution and the solution by local search. With an r-division of the exchange graph, it can be shown that the difference between the two solutions is bounded by a constant times of the total size of the boundary vertices. By carefully setting r as a function of ϵ, the size of boundary vertices can be bounded by an ϵ fraction of the size of the exchange graph, and this is used as the error bound for the solution of local search.

We can illustrate this kind of analysis by the maximum independent set problem in a planar graph G. For this problem, the exchange graph X is the subgraph induced by the union of an optimal solution O and the solution L obtained by local search algorithm. Now we find an r-division of X, such that each region has at most $c = O(1/\epsilon^2)$ vertices. Then the boundary of this r-division has at most $O(|X|/\epsilon)$ vertices. We can obtain a new solution by exchange the vertices of L in one region with the vertices of O strictly inside of the same region. Since each region has at most c vertices, we know this new solution cannot be larger than solution L, otherwise we can improve L further by the local search algorithm. This can give us a bound for the size of L in any region R: $|L \cap R| \geq |O \cap \text{int}(R)|$ where $\text{int}(R)$ represents the interior of R. Summing over all the regions, we obtain a bound of L: $|L| \geq |O| - |B|$ where B will be the multiset of boundary vertices. Since $|B|$ is at most $O(|X|/\epsilon)$, we can obtain that $|L| \geq (1+\epsilon)|O|$ by setting the constant appropriately.

1.3.5 Spanner Framework

In this subsection, we consider edge-weighted graphs. In the literature, a *spanner* is a subgraph that approximates the distance between any pair of vertices. We call such subgraph as *distance spanner*. Here we refer to a more general version of a spanner with respect to an optimization problem. A spanner for a minimization problem has the following two properties:

(1) its weight is bounded by constant times of the weight of an optimal solution;

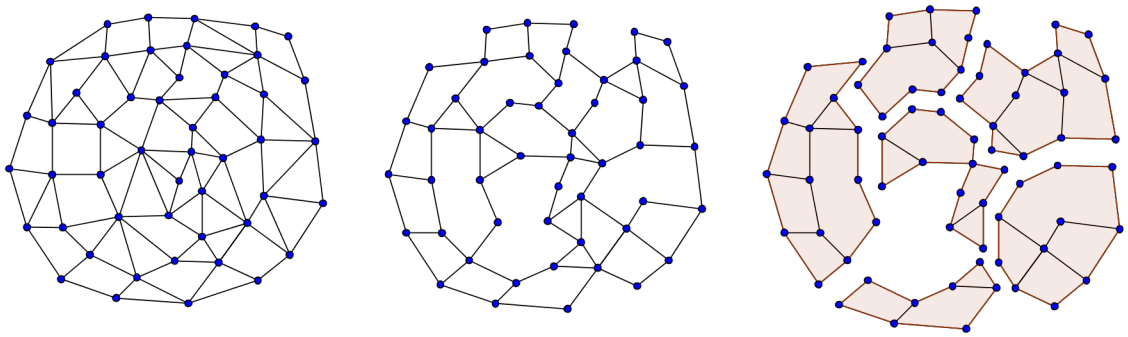

Figure 1.4: Example for the first two steps in the spanner framework. Left: the original graph. Middle: a spanner of the original graph. Right: a decomposition of the spanner.

(2) it contains a $(1+\epsilon)$-approximate solution.

The spanner framework was designed to handle classes of problems not amenable to the previous techniques, such as TSP and Steiner tree problem in edge-weighted planar graphs. Arora et al. [4] first used distance spanner as a spanner to obtain a PTAS for TSP in edge-weighted planar graphs. Their PTAS combines the distance spanner and separator technique and runs in $O(n^{1/\epsilon^2})$ time. Klein [88] improve the above result into an EPTAS running in linear time by applying the shifting technique in the dual graph of the distance spanner. A general framework is abstracted from Klein's work, and is used to obtain PTAS for other problems in planar graphs, including Steiner tree [23], Steiner forest [8, 49] and planar group Steiner tree [7]. The framework contains the following steps. (Figure 1.4 shows the first two steps in this frame work.)

Step 1: Construct a spanner.

Step 2: Decompose the spanner into a set of pieces, such that each piece has bounded treewidth and the weight of all shared edges is an ϵ fraction of the weight of the spanner.

Step 3: Solve the problem optimally in each piece by dynamic programming.

Step 4: Convert the optimal solutions from the previous step to a solution for the original graph.

For most problems, the first step is the most challenging since when we construct the spanner with small weight, we may loose the nearly optimal solution. So different problems require different techniques to prove the existence of a nearly optimal solution in the spanner constructed. For example, the PTAS for Steiner tree problem relies on a structure theorem to prove its correctness, which shows the existence of the nearly optimal solution in the spanner. The second step could be seen as applying the separator or shifting technique in the dual graph of the spanner. For shifting technique, it can be shown that the shared levels form a set of cycles, whose addition to the solution from Step 3 guarantees the connectivity of the final solution. The third step can be done by standard dynamic programming, and the last step usually combines those solutions and add some edges in the shared levels. The error from this framework usually consists of two parts: the first part is from the use of spanner instead of the original graph, and the second part is from the additional edges added in the last step, which is similar to the error introduced by the separator or shifting technique.

1.3.6 Beyond Planar Graphs

In this section, we briefly introduce how to generalize some of the previous PTAS techniques to two more general graph classes: *bounded-genus graphs* and *H-minor-free graphs*.

When we draw a non-planar graph on a plane, we can remove a crossing in the drawing by adding a "handle". The *genus* of a graph is the minimum number of handles needed to draw the graph without introducing any crossing. Since any planar graph can be drawn without any handle, it has genus 0. A class of graphs has *bounded-genus* if every graph in that class has genus at most g for a fixed g.

Recall that a minor of a graph G is obtained from G by a sequence of edge contractions, edge deletions and vertex deletions. A graph class is *minor-closed* if any minor of any graph in this class is also a member of this class. A graph class is *H-minor-free* if it is minor-closed and excludes some fixed graph H. Note that the minor-free graph families include bounded-genus graphs since bounded-genus graphs are minor-closed and there exist graphs that are excluded in this class (i.e. those have larger genus).

Generalizing PTAS results from planar graphs to bounded-genus graphs is not immediate but possible. Since bounded-genus graphs can be drawn in a surface with bounded genus, it is natural to try to reduce the problems to planar graphs. And this is the

general idea to obtain PTAS results in bounded-genus graphs. There are different ways to reduce the problems. For the spanner framework, we only need to compute a planar spanner from the bounded-genus graph, and then we can solve the problem as in planar graphs. Based on this idea, Borradaile et al. [18] generalize the spanner framework for subset connectivity-problems, including subset TSP, Steiner tree and Steiner forest, to bounded-genus graphs. Specifically, they show that it is possible to find a light subgraph such that if we cut along the graph, we can obtain a planar graph in which we compute a spanner using planar techniques. For the separator technique, we may be able to find a light subgraph such that if we remove this subgraph, we can obtain a planar graph where the weight of separator can be bounded easier than in the original graph. The PTAS results of Cohen-Addad et al. [34] for the vertex-weighted connected dominating set and vertex-weighted feedback vertex set problem exemplify this idea.

Some approaches described in previous sections are shown to be able to generalize to H-minor-free graphs. For example, Formin et al. [55] show bidimensionality theory can be used to obtain EPTAS for a set of problems in H-minor-free graphs. Most existing local search PTAS results are given for the H-minor-free graphs. To generalize shifting technique from planar graphs to H-minor-free graphs, people give two kinds of decomposition: (vertex or edge) deletion decomposition and (edge) contraction decomposition. Demaine et al. [41] first give the deletion decomposition for H-minor-free graphs to generalize the shifting technique and obtain PTAS results for a set of problems, such as independent set. Later, they [43] give the contraction decomposition for H-minor-free graphs and obtain PTAS results for contraction-closed problems, such as TSP. To generalize spanner techniques to H-minor-free graphs seems harder, and this is because the structures of H-minor-free graphs are much more complicated than planar graphs. The existing spanner results are for TSP [24] by Borradaile et al. and subset TSP [93] by Le.

1.4 Contributions of This Thesis

Following this research line, we expand the existing PTAS design approaches to some new problems, including minimum three-edge-connected spanning subgraph [122], minimum three-vertex-connected spanning subgraph [122], relaxed minimum-weight subset three-edge-connected subgraph [26] and minimum feedback vertex set in planar graphs.

There exist PTASes for the minimum two-edge-connected spanning subgraph prob-

lem [37, 12] and the relaxed minimum-weight subset two-edge-connected subgraph problem [22]. However, these PTASes cannot be immediately or simply generalized to stronger connectivity like three-edge-connectivity and three-vertex-connectivity. This is because when we want to maintain two-edge-connectivity or two-vertex-connectivity, a cycle will be enough; but beyond that, there is no obvious structure to maintain the stronger connectivity. We present the first PTAS for the relaxed minimum-weight subset three-edge-connected subgraph problem in planar graphs, which runs in $O(n \log n)$ time. This algorithm is based on the spanner framework and some structural properties of the optimal solution. Different from previous subset-connectivity problems, our work currently cannot be generalized to bounded-genus graphs by the framework given by Borradaile et al. [18]. To generalize that, we may need stronger structural result. This work was previously published [26]. We also give the first PTASes for the minimum three-edge-connected spanning subgraph and the minimum three-vertex-connected spanning subgraph problem in planar graphs. These algorithms run in $O(n)$ time and are based on Baker's shifting techniques. This work [122] is submitted to Journal of Graph Algorithms and Applications.

The minimum feedback vertex set problem asks for a minimum set of vertices in a given graph such that removing this set will result in a forest. There exist PTASes for this problem in planar graphs based on balanced separators [34] and bidimensionality [55]. However, these algorithms are very complicated to implement and hard to apply in practice. We give a PTAS for this problem in H-minor-free graphs by local search, which is very simple to implement. For a given error parameter ϵ, our algorithm runs in $n^{O(1/\epsilon^2)}$ time.

Although the algorithm by local search is simple to implement, its running time is not efficient compared to other techniques. We ask:is there a practical PTAS for minimum feedback vertex set problem in planar graphs? If not, can we design a practical heuristic algorithm that works like a PTAS? We present an $O(n \log n)$ PTAS using a linear kernel and balanced separators, and a heuristic algorithm using kernelization and local search. We implemented these algorithms and compared their performance with Becker and Geiger's 2-approximation algorithm [10]. We observe that while our PTAS is competitive in terms of solution quality with the 2-approximation algorithm on large planar graphs, its running time is much longer. Further, our heuristic algorithm can produce better solutions than the 2-approximation algorithm on most large planar graphs and provide

a trade-off between running time and solution quality, i.e. a PTAS behavior.

Chapter 2: PTAS for Relaxed Minimum-weight Subset Three-edge-connected Subgraph in Planar Graphs

The survivable network design problem aims to find a low-weight subgraph that connects a subset of vertices and will remain connected despite edge failures, an important requirement in the field of telecommunications network design. This problem can be formalized as the I-edge connectivity problem for an integer set I as follows: for an edge-weighted graph G with a requirement function on its vertices $r : V(G) \to I$, we say a subgraph H is a feasible solution if for any pair of vertices $u, v \in V(G)$, H contains $\min\{r(u), r(v)\}$ edge-disjoint u-to-v paths; the goal is to find the cheapest such subgraph. In the *relaxed* version of the problem, H may contain multiple (up to $\max I$) copies of G's edges (H is a *multi*-subgraph) in order to achieve the desired connectivity, paying for the copies according to their multiplicity; otherwise we refer to the problem as the *strict* version. Thus $I = \{1\}$ corresponds to the minimum spanning tree problem and $I = \{0, 1\}$ corresponds to the minimum Steiner tree problem. Here our focus is when $\max I \geq 2$.

This problem and variants have a long history. The I-edge connectivity problem, except when $I = \{1\}$ and $I = \{0\}$, is MAX-SNP-hard [39]. There are constant-factor approximation algorithms for the strict $\{k\}$-edge-connectivity problem: for $k = 2$, Frederickson and Jájá [59] gave a 3-approximation for this problem, and Sebő and Vygen [112] gave a 4/3-approximation for this problem in unweighted graphs; for any k, Khuller and Vishkin [84] gave a 2-approximation for this problem. Klein and Ravi [108] gave a 2-approximation for the strict $\{0, 1, 2\}$-edge-connectivity problem. For general requirements, Jain [77] gave a 2-approximation for both the strict and relaxed versions of the problem.

In planar graphs, the I-edge connectivity problem, except when $I = \{1\}$ and $I = \{0\}$, is NP-hard (by reduction from Hamiltonian cycle). Berger, Czumaj, Grigni, and Zhao [11] gave a PTAS for the relaxed $\{1, 2\}$-edge-connectivity problem, and Berger and Grigni [12] gave a PTAS for the strict $\{2\}$-edge-connectivity problem. Borradaile and Klein [20] gave an EPTAS for the relaxed $\{0, 1, 2\}$-edge-connectivity problem. The

only planar-specific algorithm for non-spanning, *strict* edge-connectivity is a PTAS for the following problem: given a subset R of edges, find a minimum weight subset S of edges, such that for every edge in R, its endpoints are two-edge-connected in $R \cup S$ [89]; otherwise, the best known results for the strict versions of the edge-connectivity problem when I contains 0 and 2 are the constant-factor approximations known for general graphs.

In this chapter, we give an EPTAS for the relaxed $\{0, 1, 2, 3\}$-edge-connectivity problem in planar graphs. This is the first PTAS for connectivity beyond 2-connectivity in planar graphs:

Theorem 2.1. *For any $\epsilon > 0$ and any planar graph instance of the relaxed $\{0, 1, 2, 3\}$-edge connectivity problem, there is an $O(n \log n)$-time algorithm that finds a solution whose weight is at most $1 + \epsilon$ times the weight of an optimal solution.*

2.1 Overview

In the this section, we overview the spanner framework for network design problems in planar graphs [22] that we use for the relaxed $\{0, 1, 2, 3\}$-edge connectivity problem. In this overview we highlight the technical challenges that arise from handling 3-edge connectivity. We then overview why we use properties of vertex connectivity to address an edge connectivity problem and state our specific observations about triconnected planar graphs that we require for the PTAS framework to apply. In the remainder, 2-EC refers to "relaxed $\{0, 1, 2\}$-edge-connectivity" and 3-EC refers to "relaxed $\{0, 1, 2, 3\}$-edge-connectivity".

2.1.1 Overview of 2-EC PTAS

The spanner framework grew out of a PTAS for travelling salesperson problem [88] and has been used to give PTASes for Steiner tree [19, 23], Steiner forest [8] and 2-EC [22] problems. For simplicity of presentation, we follow the PTAS whose running time is doubly exponential in $1/\epsilon$ [19]; this can be improved to singly exponential as for Steiner tree [23]. Note that for all these problems (except Steiner forest, which requires a preprocessing step to the framework), the optimal value OPT of the solution is within a constant factor of the optimal value of a Steiner tree on the same terminal set where we refer to vertices with non-zero requirement as *terminals*. In the following, O_ϵ-notation

hides factors depending on ϵ.

The spanner framework for a planar connectivity problem in graph G consists of the following steps. We describe the steps in terms of the relaxed I-edge connectivity problem, which, at this high level, are easy to generalize from the application of this framework to Steiner tree [19] and 2-EC [22]:

Step 1: Find the *spanner* subgraph H (described below) having the properties:

(S1) $w(H) = O_\epsilon(OPT)$, and

(S2) H contains a feasible solution to the connectivity problem of value at most $(1+\epsilon)OPT$.

To find a $(1+O(\epsilon))$-approximate solution in G, it is sufficient to find a $(1+\epsilon)$-approximate nearly-optimal solution in H by (S2).

Step 2: Decompose the spanner into a set of subgraphs, called *slices*, such that:

(A1) each slice has *branchwidth* $O_\epsilon(1)$,

(A2) the boundary of a slice is a set of cycles and every cycle bounds exactly two slices,

(A3) the weight of all boundary edges is at most ϵOPT.

The slice boundaries correspond to every k^{th} breadth-first level in the dual graph; this gives property (A2). By choosing $k = O_\epsilon(1)$, we get property (A1). Property (A3) follows from (S1) for k sufficiently large.

Step 3: Add artificial terminals to slice boundaries and assign connectivity requirements so that:

(B1) For each slice, there is a feasible solution over the original and artificial terminals whose weight is bounded by the weight of the slice boundary plus the weight of the optimal solution in the slice.

(B2) The union of these slice solutions is a feasible solution for the original original.

This can be done by adding a terminal to a boundary cycle if the cycle separates any two original terminals and assigning this terminal a connectivity requirement equal to the maximum connectivity requirement the cycle separates (e.g. 2 if the cycle separates two terminals each having a connectivity requirement of 2); this process and the fact that edge connectivity is transitive guarantees property (B2). Property (B1) is guaranteed by property (A3) as seen by adding $2 \max I$ *copies of the slices to a solution in* H*.*

Step 4: Solve the problem with respect to original and artificial terminals in each slice. *By property (A1), we can do this by dynamic programming over the branch decomposition.*

Step 5: Return the union of the slice solutions.

We apply this framework to the 3-EC problem. Algorithmically, the modifications needed for 3-EC (as compared to 2-EC or Steiner tree) are limited to Step 4; in Section 2.5, we give an $O_\epsilon(n)$-time dynamic program for the I-edge connectivity problem on graphs with branchwidth $O_\epsilon(1)$, which is inspired by that for the k-vertex-connectivity spanning subgraph problem in Euclidean space given by Czumaj and Lingas in [38, 39]. We will argue that the spanner construction (with larger constants) is the same as used for Steiner tree and 2-EC; this argument is the bulk of the technical challenge of this work. Borradaile, Klein and Mathieu show that Step 1 can be done in $O_\epsilon(n \log n)$ time [23, 22] and Steps 2 and 3 can be done in $O(n)$ time. Therefore, we will achieve an $O_\epsilon(n \log n)$ running time for 3-EC.

Spanners for connectivity problems The spanner construction for Steiner tree and 2-EC [23] (and, as we will argue, for 3-EC) starts with finding the *mortar graph* MG of the input graph G. The mortar graph is a grid-like subgraph of G that spans all the terminals and has total weight bounded by $O_\epsilon(1)$ times the minimum weight of a Steiner tree spanning all the terminals (i.e. weight $O_\epsilon(OPT)$). To construct the mortar graph, we first find an approximate Steiner tree connnecting all terminals and recursively add some short paths. Each face of MG is bounded by four $(1+\epsilon)$ approximations to short paths; the subgraph of G that is enclosed by a face of MG is called a *brick*.

A *structure theorem* shows that there is a nearly optimal solution for Steiner tree and 2-EC whose intersection with each brick is a set of non-crossing trees with $O_\epsilon(1)$ leaves

that are *portals* (a subset of $O_\epsilon(1)$ designated vertices of the boundary of the brick) [22]. Each such tree can be computed efficiently since each is a Steiner tree with vertices on the boundary of a planar graph (a brick) [51].

We compute the *spanner* subgraph H by starting with the mortar graph, assigning $O_\epsilon(1)$ vertices of each brick boundary to be portals and adding to the spanner all Steiner trees for each subset of portals in each brick. Since there are $O_\epsilon(1)$ Steiner trees per brick and each has weight at most the boundary of the brick, the spanner has weight $O_\epsilon(OPT)$. By the structure theorem, it is sufficient to solve the given problem in the spanner.

Extension to the 3-EC problem To prove that the spanner framework extends to 3-edge connectivity, we need to show this construction results in a spanner for 3-EC, that is, that H contains a $(1+\epsilon)$-approximate solution to 3-EC. This is the main technical challenge of this work. We will prove:

Theorem 2.2 (Structure Theorem for 3-EC)**.** *For any $\epsilon > 0$ and any planar graph instance (G, w, r) of the 3-EC problem, there exists a feasible solution S in the spanner H such that*

- *the weight of S is at most $(1 + c\epsilon)$OPT where c is an absolute constant, and*
- *the intersection of S with the interior of any brick is a set of $O_\epsilon(1)$ trees whose leaves are on the boundary of the brick and each tree has $O_\epsilon(1)$ leaves.*

The *interior* of a brick is the set of brick edges that are not on the boundary of the brick (that is, not in MG). We denote the interior of a brick B by $\text{int}(B)$. Consider a brick B of G whose boundary is a face of MG and consider the intersection of OPT with the interior of this brick, $\text{OPT} \cap \text{int}(B)$. To prove the Structure Theorem, we will show that:

(P1) $\text{OPT} \cap \text{int}(B)$ can be partitioned into a set of trees $\mathcal{T}$ whose leaves are on the boundary of B.

(P2) If we replace any tree in $\mathcal{T}$ with another tree spanning the same leaves, the result is a feasible solution.

(P3) There is another set of $O(1)$ trees $\mathcal{T}'$ and a set of brick boundary edges B' that costs at most a $1+\epsilon$ factor more than $\mathcal{T}$, such that each tree of $\mathcal{T}'$ has $O(1)$ leaves and $(\text{OPT} \setminus \mathcal{T}) \cup \mathcal{T}' \cup B'$ is a feasible solution.

Property P1 implies that we can decompose an optimal solution into a set of trees inside of bricks plus some edges of MG. Property P2 shows that we can treat those trees independently with regard to connectivity, and this gives us hope that we can replace $\text{OPT} \cap \text{int}(B)$ with some Steiner trees with terminals on the boundary which we can efficiently compute in planar graphs [51]. Property P3 shows that we can compute an approximation to $\text{OPT} \cap \text{int}(B)$ by guessing $O(1)$ leaves.

For the Steiner tree problem, P1 and P2 are nearly trivial to argue; the bulk of the work is in showing P3 [19].

For the 2-EC problem, P1 depends on first converting G and OPT into G' and OPT' such that OPT' biconnects (two-vertex connects) the terminals requiring two-edge connectivity and using the relatively easy-to-argue fact that every cycle of OPT' contains at least one terminal. By this fact, a cycle in OPT' must contain a vertex of the brick's boundary (since MG spans the terminals), allowing the partition of $\text{OPT}' \cap \text{int}(B)$ into trees. P2 and P3 then require that two-connectivity across the brick is maintained.

For the 3-EC problem, P1 is quite involved to show, but further to that, showing Property P2 is also involved; the issues are illustrated in Figure 2.1 and are the focus of Sections 2.2 and 2.3. As with 2-EC, we convert OPT into a vertex connected graph to simplify the arguments. Given Properties P1 and P2, we illustrate Property P3 by following a similar argument as for 2-EC; since this requires reviewing more details of the PTAS framework, we cover this in Section 2.4.

Non-planar graphs We point out that, while previously-studied problems that admit PTASes in planar graphs (e.g. independent set and vertex cover [6], TSP [88, 87, 4], Steiner tree [23] and forest [8], 2-EC [22]) generalize to surfaces of bounded genus [18], it is not clear how to generalize the method presented here for 3-EC to higher genus surfaces. In the generalization to bounded genus surfaces, the graph is preprocessed (by removing some provably unnecessary edges) so that one can compute a mortar graph whose faces bound disks. This guarantees that even though the input graph is not planar, the bricks are; this is sufficient for proving above-numbered properties in the case of TSP,

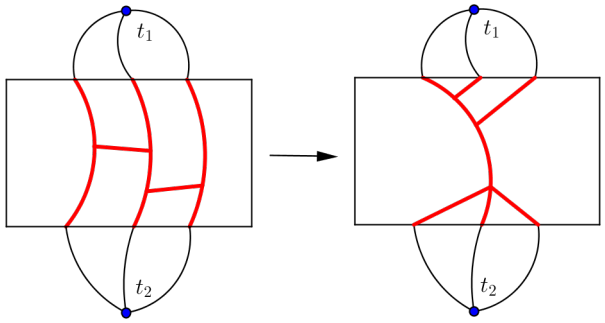

Figure 2.1: If the bold red tree (left) is OPT $\cap$ int(B) (where B is denoted by the rectangle), replacing the tree with another tree spanning the same leaves (right) could destroy 3-connectivity between t_1 and t_2. We will show that such a tree cannot exist in a minimally connected graph.

Steiner tree and forest and 2-EC. However, for 3-ECP, in order to prove P2, we require *global* planarity, not just planarity of the brick. To the authors' knowledge, this is the only problem that we know to admit a PTAS in planar graphs that does not naturally generalize to toroidal graphs.

2.1.2 Reduction to Vertex Connectivity

Now we overview how we use vertex connectivity to argue about the structural properties of edge-connectivity required for the spanner properties.

We require a few definitions. Vertices x and y are k-vertex-connected in a graph G if G contains k pairwise vertex disjoint x-to-y paths. If $k = 3$ ($k = 2$), then x and y are also called triconnected (biconnected). For a subset Q of vertices in G and a requirement function $r : Q \to \{2, 3\}$, subgraph H is said to be (Q, r)-vertex-connected if every pair of vertices x, y in Q is k-vertex-connected where $k = \min\{r(x), r(y)\}$. We call the vertices of Q *terminals*. If $r(x) = 3$ ($r(x) = 2$) for all $x \in Q$, we say H is Q-triconnected (Q-biconnected). We say a (Q, r)-vertex-connected graph is *minimal*, if no edge or vertex can be deleted without violating the connectivity requirements.

We *cleave* vertices to transform edge-connectivity into vertex-connectivity. Informally, cleaving a vertex is splitting the vertex into two copies and adding a zero-weight edge between the copies; incident edges choose between the copies in a planarity-

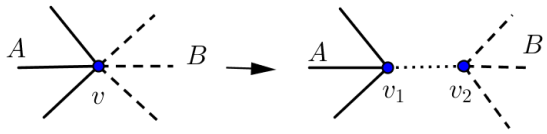

Figure 2.2: Vertex v is cleaved into vertices v_1 and v_2. The edges incident to v are partitioned into two sets A and B to become incident to distinct copies.

preserving way (Figure 2.2). We show in Section 2.4.2.2 how to cleave the vertices of OPT, creating OPT$'$, so that if two terminals are k-edge-connected in OPT, there are corresponding terminals in OPT$'$ that are k-vertex-connected. We will prove that OPT$'$ satisfies Properties P1 and P2 and since OPT$'$ is obtained from OPT by cleavings, these two properties also hold for OPT.

To prove that OPT$'$ satisfies Property P1, we show that every cycle in OPT$'$ contains at least one terminal (Section 2.2). To prove that OPT$'$ satisfies Property P2, we define the notion of a *terminal-bounded component*: a connected subgraph is a terminal-bounded component if it is an edge between two terminals or obtained from a maximal terminal-free subgraph S (a subgraph containing no terminals), by adding edges from S to its neighbors (which are all terminals by maximality of S). In Section 2.3, we prove that in a minimal Q-triconnected graph any terminal-bounded component is a tree whose leaves are terminals as well as:

Theorem 2.3 (Connectivity Separation Theorem)**.** *Given a minimal (Q, r)-vertex-connected planar graph, for any pair of terminals x and y that require triconnectivity (biconnectivity), there are three (two) vertex disjoint paths from x to y in G such that any two of them do not contain edges of the same terminal-bounded tree.*

Corollary 2.4. *Given a minimal (Q, r)-vertex-connected planar graph, for any pair of terminals x and y that require triconnectivity (biconnectivity), there exist three (two) vertex disjoint x-to-y paths such that any path that connects any two of those x-to-y paths contains a terminal.*

This corollary can be viewed as a generalization of the following by Borradaile and Klein for 2-ECP [22]:

Theorem 2.5. (Theorem 2.8 [22]). *Given a graph that minimally biconnects a set of terminals, for any pair of terminals x and y and for any two vertex disjoint x-to-y paths, any path that connects these paths must contain a terminal.*

Note that Theorem 2.5 holds for general graphs while we only know Corollary 2.4 to hold for planar graphs, underscoring why our PTAS does not generalize to higher-genus graphs. Further *"for any"* is sufficient for biconnectivity (Theorem 2.5) whereas *"there exists"* is necessary for triconnectivity (Corollary 2.4) as illustrated by the example in Figure 2.3. Higher connectivity comes at a price.

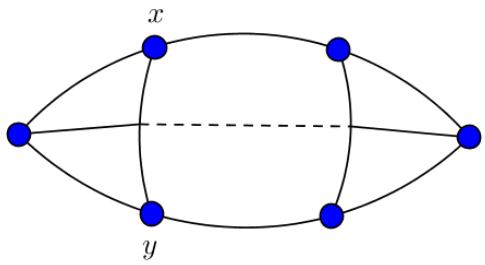

Figure 2.3: A minimal Q-triconnected graph. The bold vertices are terminals. The dashed path connects two x-to-y paths but it does not contain any terminal.

For OPT$'$, Corollary 2.4 implies Property P2. Consider the set of disjoint paths guaranteed by Corollary 2.4. If any tree replacement in a brick merges any two disjoint paths, say P_1 and P_2, in the set (the replacement in Figure 2.1 merges three paths), then the replaced tree must contain at least one vertex of P_1 and one vertex of P_2. This implies the replaced tree contains a P_1-to-P_2 path P such that each vertex in P has degree at least two in the replaced tree. Further, P contains a terminal by Corollary 2.4. However, all the terminals are in the mortar graph, which forms the boundaries of the bricks. So P must have a common vertex with the boundary of the brick. By Property P1, the replaced tree, which is in the intersection of OPT$'$ with the interior of the brick, can only contain leaves on the boundary of the brick. Therefore, the replaced tree can not contain such a P_1-to-P_2 path, otherwise there is a vertex in P that has degree one in the tree. We give the complete proof of this implication in Section 2.4.

2.2 Vertex-connectivity Basics

In this section, we consider minimal (Q, r)-vertex-connected graphs for a subset Q of vertices and a requirement function $r : Q \to \{2, 3\}$. We use the properties given in this section throughout the proof of the Connectivity Separation Theorem.

Lemma 2.6. *A minimal (Q, r)-vertex-connected graph is biconnected.*

Proof. For a contradiction, assume that a minimal (Q, r)-vertex-connected graph H has a cut-vertex u and let the subgraphs be H_i for $0 < i \leq k$ and $k \geq 2$ after removing u. Then for any vertex $x \in H_i$ and $y \in H_j$ $(i \neq j)$, every x-to-y path must contain u. If H_i and H_j $(i \neq j)$ both have terminals, then terminals in those different subgraphs do not achieve the required vertex-connectivity. It follows that there exists one subgraph $H_i \cup \{u\}$ that contains all the terminals. For any two terminals $x, y \in H_i \cup \{u\}$, the paths witnessing their connectivity are simple and so can only visit u once. Therefore, $H_i \cup \{u\}$ is a smaller subgraph that is (Q, r)-vertex-connected, contradicting the minimality. □

2.2.1 Ear Decompositions

An *ear decomposition* of a graph is a partition of its edges into a sequence of cycles and paths (the *ears* of the decomposition) such that the endpoints of each ear belong to the union of earlier ears in the decomposition. An ear is *open* if its two endpoints are distinct from each other. An ear decomposition is *open* if all ears but the first are open. A graph containing more than one vertex is biconnected if and only if it has an open ear decomposition [118]. Ear decompositions can be found greedily starting with any cycle as the first ear. It is easy to see that a more general ear decomposition can start with any biconnected subgraph:

Observation 2.7. *For any biconnected subgraph H of a biconnected graph G, there exists an open ear decomposition $E_1, E_2, \ldots, E_k$ of G such that $H = \bigcup_{i \leq j} E_i$ for some $j \leq k$.*

Let G be a minimal (Q, r)-vertex-connected graph, and let H be a minimal Q_3-triconnected subgraph of G where $Q_3 = r^{-1}(3)$. Then more strongly, we can assume that each ear of G that is within the parts of $G \setminus H$ contains a terminal. (G is biconnected by Lemma 2.6.) We do so by starting with an open ear decomposition of H and then for

each terminal that is not yet spanned in turn, we add an ear through it; such an ear exists because these terminals require biconnectivity and must have two disjoint paths to the partially constructed ear decomposition. Any remaining edges after the terminals have been spanned would contradict the minimality of G. Formally and more specifically:

Observation 2.8. *For G and H, there is an open ear decomposition $E_1, E_2, \ldots, E_k$ of G such that for any component χ of $G \setminus H$, $\chi = \bigcup_{i=a}^{b} E_i$ for some $a \leq b \leq k$ and E_i contains a terminal for $i = a, \ldots, b$.*

Lemma 2.9. *For G and H, there is an open ear decomposition $E_1, E_2, \ldots, E_k$ of G such that for any component $\chi = \bigcup_{i=a}^{b} E_i$ of $G \setminus H$, any path in χ (or $\chi \setminus E_a$) with both endpoints in H (or $H \cup E_a$) strictly contains a vertex of Q.*

Proof. We prove the lemma for the path in χ with endpoints in H; the other case can be proved similarly. We prove this by induction on the index of the ear decomposition guaranteed by Observation 2.8. A path P in $G \setminus H$ with both endpoints in H belongs to a component χ of $G \setminus H$. Suppose that the lemma is true for every H-to-H path in $\bigcup_{i=a}^{c} E_i$; we prove the lemma true for such a path in $\bigcup_{i=a}^{c+1} E_i$. Since E_{c+1} is an open ear, any path with two endpoints in H that uses an edge of E_{c+1} would have to contain the entirety of E_{c+1}, which contains a terminal. □

2.2.2 Removable Edges

Holton, Jackson, Saito and Wormald study the *removability* of edges in triconnected graphs [72]. For an edge $e = uv$ of a simple, triconnected graph G, removing e consists of (i) deleting uv from G, (ii) if u or v now have degree 2, contracting incident edges, and (iii) deleting parallel edges. The resulting graph is denoted by $G \ominus e$. If $G \ominus e$ is triconnected, then e is said to be removable. We use the following theorems of Holton et al. [72].

Theorem 2.10 (Theorem 1 [72])**.** *Let G be a triconnected graph of order at least six and $e \in E(G)$. Then e is nonremovable if and only if there exists a set S containing exactly two vertices such that $G \setminus \{e, S\}$ has exactly two components A, B with $|A| \geq 2$ and $|B| \geq 2$.*

In the above theorem, we call (e, S) a separating pair. For a separating pair (e, S) of G, we say S is the *separating set* for e.

Theorem 2.11 (Theorem 2 [72])**.** *Let G be a triconnected graph of order at least six, and let (e, S) be a separating pair of G. Let $e = xy$, and let A and B be the two components of $G \setminus \{e, S\}$, $x \in A$, and $y \in B$. Then every edge joining S and $\{x, y\}$ is removable.*

Theorem 2.12 (Theorem 6 part (a) [72])**.** *Let G be a triconnected graph of order at least six and C be a cycle of G. Suppose that no edges of C are removable. Then there is an edge yz in C and a vertex x of G such that xy and xz are removable edges of G, $d_G(y) = d_G(z) = 3$ and $d_G(x) \geq 4$.*

2.2.3 Properties of Minimal (Q, r)-vertex-connected Graphs

For a Q-triconnected graph H, we can obtain another graph H' by contracting all the edges incident to the vertices of degree two in H. We say H' is the *contracted version of H* and, alternatively, is *contracted* Q-triconnected.

Lemma 2.13. *A contracted minimal Q-triconnected graph is triconnected.*

Proof. For a contradiction, we assume that a contracted minimal Q-triconnected graph H has a pair of cut-vertices $\{u, v\}$ and let the subgraphs be H_i for $0 < i \leq k$ and $k \geq 2$ after removing $\{u, v\}$. Then for any vertices $x \in H_i$ and $y \in H_j$ $(i \neq j)$, every x-to-y path must use either u or v. If H_i and H_j $(i \neq j)$ both contain terminals, then terminals in those different subgraphs do not satisfy the triconnectivity. It follows there exists one strict subgraph $H_i \cup \{u, v\}$ containing all the terminals. For any two terminals $x, y \in H_i \cup \{u, v\}$, the paths witnessing their connnectivity are simple and can only visit u and v once. So $H_i \cup \{u, v\}$ is a smaller subgraph that is Q-triconnected, which contradicts the minimality. □

Lemma 2.14. *If H is a minimal Q-triconnected graph, then the contracted version of H is also a minimal Q-triconnected graph.*

Proof. Let H' be the contracted Q-triconnected graph obtained from H. For a contradiction, assume H' is not minimal Q-triconnected. Then we can delete at least one edge, say e, in H' while maintaining Q-triconnectivity. Then e corresponds a path in H, deleting which will not affect the Q-triconnectivity. This contradicts the minimality of H. □

Lemma 2.15. *For $|Q| = 3$, a contracted minimal Q-triconnected graph is simple or is a triangle with three pairs of parallel edges. For $|Q| > 3$, a contracted minimal Q-triconnected graph is simple.*

Proof. Let H be a minimal Q-triconnected graph, and H' the contracted graph. We have the following observation.

Observation 2.16. *If there are parallel edges between any pair of vertices in H', the paths witnessing the vertex-connectivity between any other pair of terminals can only use one of the parallel edges.*

By the above observation, the parallel edges can only be between terminals in H'.

Claim 2.17. *For $|Q| > 2$, there can not exist three parallel edges between any two terminals in H'.*

Proof. Let t_1, t_2 and t_3 be three terminals and assume there are three parallel edges, say e_1, e_2 and e_3, between t_1 and t_2. Let P_1 be the path in H corresponding to e_1. We will argue that $H \setminus P_1$ is Q-triconnnected by showing that $H' - e_1$ is Q-triconnected. There must be a path in H' from t_1 to t_3 that does not use e_1, e_2 and e_3 by Observation 2.16 and a path in H' from t_2 to t_3 that does not use e_1, e_2 and e_3. These paths witness a t_1 to t_2 path R in H' that does not use e_1, e_2 and e_3. So after deleting e_1, t_1 and t_2 are still triconnected by e_2, e_3 and R. By Observation 2.16, deleting e_1 does not affect the triconnectivity of other pairs of terminals. □

We first prove the first statement of Lemma 2.15. Let t_1, t_2 and t_3 be the three terminals, and suppose H' is not simple; let e_1 and e_2 be parallel edges w.l.o.g. between t_1 and t_2. By Observation 2.16, there must be two disjoint paths in H' from t_1 to t_3 that do not use e_1 and e_2. Therefore, there is a simple cycle, called C_{13}, through t_1 and t_3. Similarly, there is another cycle, called C_{23}, through t_2 and t_3. If C_{13} and C_{23} have only one common vertex t_3, then $C_{13} \cup C_{23} \cup \{e_1, e_2\}$ is a subgraph of H' that is Q-triconnected, and must be a triangle with three pairs of parallel edges by the minimality of H' and Lemma 2.14. If C_{13} and C_{23} have more than one common vertex, then $C_{13} \cup C_{23}$ contains a simple cycle through t_1 and t_2. So $H' - e_1$ will be a smaller Q-triconnected graph, contradicting the minimality of H'.

Now we prove the second statement. For any four terminals $t_1, t_2, t_3, t_4 \in Q$, without loss of generality, assume there are two parallel edges e_1 and e_2 between t_1 and t_2 in H' by Claim 2.17. We will prove that $H' - e_1$ is also Q-triconnected, contradicting the minimality of H. To prove this, we argue that t_1 and t_2 are four-vertex-connected in H' and biconnected in $H' \setminus \{e_1, e_2\}$.

For a contradiction, we assume t_1 and t_2 are simply connected but not biconnected in $H' \setminus \{e_1, e_2\}$. By Claim 2.17, every other pair of terminals is at least biconnected in $H' \setminus \{e_1, e_2\}$. Consider the block-cut tree of $H' - e_1$, the tree whose vertices represent maximal biconnected components and whose edges represent shared vertices between those components [52].

The biconnectivity of t_1 and t_3 implies t_1 and t_3 are in a common block B_{13} in the block-cut tree. Likewise, t_2 and t_3 are in a common block B_{23}. If $t_2 \in B_{13}$ then t_1 and t_2 are biconnected, a contradiction. Now we have $B_{13} \cap B_{23} = \{t_3\}$. The biconnectivity of t_1 and t_4 implies that t_1 and t_4 are in the same block. Since t_3 is a cut for t_1 and t_2, t_3 is also a cut for t_4 and t_2, which contradicts the biconnectivity of t_2 and t_4. So t_1 and t_2 are biconnected in $H' \setminus \{e_1, e_2\}$. □

Lemma 2.18. *Let H be a contracted minimal Q-triconnected simple graph. Then for any $e \in E(H)$, if neither of the two endpoints of e are terminals, e is nonremovable.*

Proof. By Lemma 2.13, H is triconnected. Let $e \in H$ be an edge, neither of whose endpoints are terminals. For a contradiction, suppose e is removable, then $H \ominus e$ is triconnected. Since neither of the two endpoints of e are terminals, $Q \subseteq V(H \ominus e)$. So $H \ominus e$ is a smaller Q-triconnected graph than H, contradicting minimality of H. □

2.2.4 Cycles Must Contain Terminals

Borradaile and Klein proved that in a minimal Q-biconnected graph, every cycle contains a terminal (Theorem 2.5 [22]). We prove the following:

Lemma 2.19. *Let H be a contracted minimal Q-triconnected graph. Then every cycle in H contains a vertex of Q.*

Proof. H is triconnected by Lemma 2.13. If H is not simple, then either $|Q| = 2$ or $|Q| = 3$. If $|Q| = 2$, then by the minimality of H, H consists of three parallel edges. If $|Q| = 3$, then by Lemma 2.15, H is a triangle with three pairs of parallel edges.

Now we assume H is a simple graph and by Lemma 2.15, $|Q| \geq 3$. For a contradiction, assume there is a cycle C in H on which there is no terminal. Since H is simple, by Lemma 2.18, every edge on C will be nonremovable. Since $|C| \geq 3$ and $|Q| \geq 3$, $|V(H)| \geq 6$. By Theorem 2.12 if there is no removable edges on C, then there exists an edge yz on C and another vertex x of H such that xy and xz are removable, $d_H(y) = d_H(z) = 3$ and $d_H(x) \geq 4$. In $H - xy$, y is the only possible degree 2 vertex, and contracting yz does not introduce any parallel edges, so $H \ominus xy$ contains Q and is Q-triconnected, which contradicts the minimality of H. □

Theorem 2.20. *For a requirement function $r : Q \to \{2, 3\}$, let G be a minimal (Q, r)-vertex-connected graph. Then every cycle in G contains a vertex of Q.*

Proof. Let G' be a minimal Q-triconnected graph that is a supergraph of G. Let G'' be the contracted minimal Q-triconnected graph obtained from G'. Then by Lemma 2.19, every cycle in G'' contains a vertex of Q; G' also has this property since G' is a subdivision of G'' and clearly, G as a subgraph of G', also has this property. □

Lemma 2.21. *Let H be a contracted minimal Q-triconnected graph. If a cycle in H only contains one vertex v of Q, then one edge of that cycle incident to v is removable.*

Proof. For a contradiction, assume both edges in the cycle incident to v are nonremovable. By Lemma 2.18 all the other edges in the cycle are also nonremovable, so all the edges in the cycle are nonremovable. By Theorem 2.12, there is an edge yz in the cycle and a vertex x in H such that xy and xz are removable, $d_H(y) = d_H(z) = 3$ and $d_H(x) \geq 4$. Then one of y and z can not be in Q. W.l.o.g. assume y is not in Q. In $H - xy$, y is the only possible degree 2 vertex, and contracting yz does not introduce any parallel edges, so $H \ominus xy$ contains Q and is Q-triconnected, which contradicts the minimality of H. □

2.3 Connectivity Separation

In this section we continue to focus on vertex connectivity and prove the Connectivity Separation Theorem. The Connectivity Separation Theorem for biconnectivity follows easily from Theorem 2.5. To see why, consider two paths P_1 and P_2 that witness the

biconnectivity of two terminals x and y. For an edge of P_1 to be in the same terminal-bounded component as an edge of P_2, there would need to be a P_1-to-P_2 path that is terminal-free. However, such a path must contain a terminal by Theorem 2.5. Herein we mainly focus on triconnectivity.

For a requirement function $r : Q \to \{2, 3\}$, let G be a minimal (Q, r)-vertex-connected planar graph. We say a subgraph is *terminal-free* if it is connected and does not contain any terminals. It follows from Theorem 2.20 that any terminal-free subgraph of G is a tree. We partition the edges of G into *terminal-bounded* components as follows: a terminal-bounded component is either an edge connecting two terminals or is obtained from a *maximal* terminal-free tree T by adding the edges from T to its neighbors, all of which are terminals. Theorem 2.22 will show that any terminal-bounded subgraph is also a tree.

For a connected subgraph χ of G and an embedding of G with outer face containing no edge of χ, let $C(\chi)$ be the simple cycle that strictly encloses the fewest faces and all edges of χ, if such a cycle exists. (Note that $C(\chi)$ does not exist if there is no aforementioned choice for an outer face.) In order to prove the Connectivity Separation Theorem for bi- and triconnectivity, we start with the following theorem:

Theorem 2.22 (Tree Cycle Theorem)**.** *Let T be a terminal-bounded component in a minimal Q-triconnected planar graph H. Then T is a tree and $C(T)$ exists with the following properties*

(*a*) *The internal vertices of T are strictly inside of $C(T)$.*

(*b*) *All vertices strictly inside of $C(T)$ are on T.*

(*c*) *All leaves of T are in $C(T)$.*

(*d*) *Any pair of distinct maximal terminal-free subpaths of $C(T)$ does not contain vertices of the same terminal-bounded tree.*

We can also obtain an interesting property by this theorem.

Theorem 2.23. *In a planar graph that minimally pairwise triconnects a set of terminal vertices, every cycle contains at least two terminals.*

Proof. For a contradiction, assume there is a cycle in H that only containing one terminal, then there is a terminal-bounded component containing that cycle, which can not be a tree, contradicting the Tree Cycle Theorem. □

We give the proof the Tree Cycle Theorem in Subsection 2.3.2. First, let us see how the Tree Cycle Theorem implies the Connectivity Separation Theorem.

2.3.1 The Tree Cycle Theorem Implies the Connectivity Separation Theorem

For a requirement function $r : Q \to \{2, 3\}$, let G be a minimal (Q, r)-vertex-connected planar graph. Let Q_3 be the set of terminals requiring triconnectivity, and let H be a minimal Q_3-triconnected subgraph of G. Let $Q_2 = Q \setminus Q_3$. We will prove the theorem for different types of pairs of terminals, based on their connectivity requirement. Lemma 2.9 allows us to focus on H when considering terminals in H (note that terminals of Q_2 may be in H), for a simple corollary of Lemma 2.9 is that if two trees are terminal-bounded in H, then they cannot be subtrees of the same terminal-bounded tree in G. Note that if we consider a subset of the terminals in defining free-ness of terminals, the same properties will hold for Q, as adding terminals only further partitions terminal-bounded trees. Consider two terminals x and y.

2.3.1.1 Connectivity Separation for $\mathbf{x}, \mathbf{y} \in \mathbf{Q_3}$

For now, we consider only Q_3 to be terminals. We say connected components *share* a terminal-bounded tree if they contain edges (and so internal vertices) of that tree. We prove the following lemma which can be seen as a generalization of Connectivity Separation for contracted triconnected graphs. We use this generalization to prove Connectivity Separation for terminals both of which may not be in Q_3. Connectivity Separation for terminals in Q_3 follows from this lemma by considering three vertex disjoint paths matching A to B where $A = \{x, x, x\}$ and $B = \{y, y, y\}$. Note that the lemma may swap endpoints of paths; in particular, for vertex disjoint a-to-b and c-to-d paths, the lemma may only guarantee a-to-d and c-to-b paths that do not share terminal-bounded trees.

Lemma 2.24. *If two multisets of vertices A and B (where $|A| = |B| = 2$, resp. 3) satisfy the following conditions, then there are two (resp. three) internally vertex-disjoint paths from A to B such that no two of them share the same terminal-bounded tree.*

1. *Distinct vertices in A are in distinct terminal-bounded trees and distinct vertices in B are in distinct terminal-bounded trees.*

2. *There are two (resp. three) vertex-disjoint paths from A to B.*

Proof. We prove the lemma for three paths as the two-paths version is proved by the first case of three-paths version. Let P_1, P_2 and P_3 be the three vertex disjoint paths whose endpoints are in different terminal-bounded trees. For $i = 1, 2, 3$, let a_i and b_i be the endpoints of P_i. Let $\mathcal{T}$ be the collection of terminal-bounded trees shared by two or more of P_1, P_2 and P_3. We prove, by induction on the size of $\mathcal{T}$, that we can modify the paths to satisfy Connectivity Separation. We pick a tree $T \in \mathcal{T}$ shared by (w.l.o.g.) P_1 and modify the paths so that T is not shared by the new paths; further, we show that the terminal-bounded trees shared by the new paths are a subset of $\mathcal{T} \setminus \{T\}$.

We order the vertices of P_i from a_i to b_i for $i = 1, 2, 3$. Among all the trees in $\mathcal{T}$ shared by P_1, let T be the tree sharing the first vertex from a_1 to b_1 along P_1. Without loss of generality, assume that T is shared by P_2. (P_3 may also share T.) Let C be the cycle guaranteed by the Tree Cycle Theorem for T in H. Since x and y are terminals, they are not strictly enclosed by C (property (b)). Further, P_1 and P_2 must both contain vertices of C because P_1 and P_2 both contain internal vertices of T and the internal vertices of T are strictly enclosed by C (property (a)).

Augmenting the paths First we augment each path to simplify the construction, but may make the paths non-simple. If there is a terminal-bounded tree shared by only P_i and C (and not P_j, $j \neq i$), then there is a terminal-free path from P_i to C; add two copies of this path to P_i (P_i travels back and forth along this path). We repeat this for every possible shared tree and $i = 1, 2, 3$. We let P_1, P_2, P_3 be the resulting paths. Note that adding such paths does not introduce any new shared terminal-bounded trees between P_1, P_2, P_3 and P_1, P_2, P_3 are still vertex-disjoint.

Let u_i and v_i be the first and last vertex of P_i that is in C. There are two cases:

Case 1: $\mathbf{P_3}$ is disjoint from C. In this case there were no applicable augmentations to P_3 as described above (and u_3 and v_3 are undefined). Since T is the first tree in $\mathcal{T}$ along

P_1, u_1 and u_2 cannot be internal vertices of the same terminal-bounded tree. Further, by planarity and disjointness of the three paths, $\{u_1, v_1\}$ and $\{u_2, v_2\}$ do not interleave around C. Let C_i be the u_i-to-v_i subpath of C disjoint from $\{u_{3-i}, v_{3-i}\}$ for $i = 1, 2$: C_1 and C_2 are disjoint. By construction, $P_1 \cup P_2 \cup C_1 \cup C_2 \setminus \{P_1[u_1, v_1], P_2[u_2, v_2]\}$ (by $P[a, b]$ or $T[a, b]$ we denote the a-to-b subpath of path P or tree T) contains two disjoint A-to-B paths; these paths replace P_1 and P_2. By the definition of u_i and v_i for $i = 1, 2$ and the above-described path augmentation, if C_i shares a terminal-bounded tree with another path, then that tree was already in $\mathcal{T}$. Therefore we have reduced the number of shared terminal-bounded trees (since T is no longer shared) without introducing any new shared terminal-bounded trees.

Case 2: P_3 and C have at least one common vertex. In this case, P_3 may or may not contain internal vertices of T. By planarity and disjointness of the paths, the sets $\{u_1, u_2, u_3\}$ and $\{v_1, v_2, v_3\}$ do not interleave around C. Let C_u and C_v be the minimal subpaths of C that span $\{u_1, u_2, u_3\}$ and $\{v_1, v_2, v_3\}$, respectively. Let C_1 and C_2 be the components of $C \setminus \{C_u, C_v\}$. Then C_1 and C_2 are disjoint paths that connect two vertices of $\{u_1, u_2, u_3\}$ to two vertices of $\{v_1, v_2, v_3\}$.

By planarity and disjointness of the paths, there must be a leaf of T that is an internal vertex of C_u in order for paths to reach T from that leaf of T; the same property holds for C_v. Let C_3 be the simple path from the middle vertex of $\{u_1, u_2, u_3\}$ to the middle vertex of $\{v_1, v_2, v_3\}$ in $C_u \cup C_v \cup T$. Then the to-C prefices of P_i, C_i and the from-C suffices of P_i (for $i = 1, 2, 3$) together form three vertex-disjoint paths between the same endpoints.

The resulting path that contains C_3 is the only of the three resulting paths that contains internal vertices of T since C_1 and C_2 do not share the same terminal-bounded tree because the endpoints of $C_3 \cap T$ are terminals (property (d) of the Tree Cycle Theorem).

By the definition of u_i and v_i and the above-described path augmentation, if C_i shares a terminal-bounded tree with another path, then that tree was already in $\mathcal{T}$. Therefore we have reduced the number of shared terminal-bounded trees (since T is no longer shared) without introducing any new shared terminal-bounded trees. □

2.3.1.2 Connectivity Separation for $\mathbf{x}, \mathbf{y} \in \mathbf{H} \cap \mathbf{Q}$

Note that H may span vertices of Q_2. Let P_1 and P_2 be vertex-disjoint x-to-y paths. The first and last edges of P_1 and P_2 are in different terminal-bounded trees, since x and y are terminals. If either P_1 or P_2 is an edge, then we could obtain Connectivity Separation for x and y. Otherwise, let x_1 and x_2 be x's neighbors on P_1 and P_2 respectively; similarly define y_1 and y_2. Let P_1' and P_2' be the paths guaranteed by Lemma 2.24 when applied to $P_1[x_1, y_1]$ and $P_2[x_2, y_2]$. Then $P_1' \cup P_2' \cup \{xx_1, xx_2, yy_1, yy_2\}$ contain vertex disjoint x-to-y paths that satisfy the requirements of Connectivity Separation Theorem.

2.3.1.3 Connectivity Separation for $\mathbf{x} \in \mathbf{Q_2} \setminus \mathbf{H}, \mathbf{y} \in \mathbf{Q}$

Since x and y only require biconnectivity, we will prove that there exists a simple cycle C containing x and y, such that every C-to-C path strictly contains a terminal, from which Connectivity Separation follows as argued in the beginning of Section 2.3. The following claim gives a sufficient condition for such cycle C.

Claim 2.25. *If every terminal-bounded tree of H and every component of $G \setminus E(H)$ contain at most one strict subpath of C, then every C-to-C path contains a terminal.*

Proof. Let P be a C-to-C path and let C_χ be the subpath of C in the component χ of $G \setminus H$. Then C_χ has two endpoints in H and it must contain a terminal by Lemma 2.9. Every subpath of P in χ has endpoints in $H \cup C_\chi$. We can take C_χ as the first ear for χ, and then every subpath of P in χ contains a terminal by Lemma 2.9. So if P contains an edge of χ, we have the claim.

If P does not contain an edge in $G \setminus H$, then $P \subseteq H$. Further, if P does not contain any terminal, then it must be in some terminal-free tree T by condition of the claim T contains only one subpath C_T of C. Since P is a C-to-C path, C_T and P form a cycle in T, which contradicts T is a tree. So P must contain a terminal. □

The following claim will allow us to use Lemma 2.24 on parts of the x-to-y paths.

Claim 2.26. *Each connected component of $G \setminus H$ has at most one non-terminal vertex in common with any terminal-bounded tree of H.*

Proof. For a contradiction, suppose a and b are two non-terminal vertices of component χ of $G \setminus H$ that are both in some terminal-bounded tree T of H. Let P be an a-to-b

path in χ and let R be a maximal suffix of $T[a, b]$ every internal vertex of which has degree 2 in G. We show that deleting R maintains the required connectivity of G and this contradicts minimality of G.

Let $H' = (H \setminus R) \cup P$. First we show that H' is Q_3-triconnected. Since a and b are not terminals, they are not leaves of T. Then $(T \setminus R) \cup P$ is a tree that contains all the terminals of T as leaves. Since H satisfies Connectivity Separation, as argued (Section 2.3.1.1 and 2.3.1.2), any two terminals have three vertex disjoint paths, no two of which contain edges from the same terminal-bounded tree. Therefore replacing T with $(T \setminus R) \cup P$ will preserve the connectivity.

To prove $G \setminus R$ is biconnected, we construct an open ear decomposition. Let R' be the maximal superpath of R in H every internal vertex of which has degree 2 in H. (Note that the endpoints of R need not have degree > 2 in H.) Since the contracted version of H is triconnected, the endpoints of R' are triconnected in H and R' becomes an edge in the contracted version of H. Therefore $H \setminus R'$ is biconnected. Consider an ear decomposition of G that starts with an ear decomposition of $H \setminus R'$ as guaranteed by Observation 2.7. Since every internal vertex of R has degree 2 in G and since the endpoints of R have degree > 2 in G, we can greedily select ears of $(G \setminus H) \cup (R' \setminus R)$ to span the resting vertices. □

Let χ be the component of $G \setminus H$ that contains x, and let a and b be two vertices of $\chi \cap H$. Then a and b are in distinct terminal-bounded trees of H: if either of a and b is terminal, then it could be in at least two terminal-bounded trees; otherwise by Claim 2.26 they can not be in the same terminal-bounded tree. We consider an ear decomposition of G that is guaranteed by Lemma 2.9, starting with an ear decomposition of H with consecutive ears composing χ. There are two cases.

Case 1: $\mathbf{y} \in \mathbf{H}$ In this case we construct a simple cycle C to satisfy Claim 2.25 containing x and y as follows. We first find an a-to-b path P_1 of H that contains y: add another new vertex t and two new edges ta and tb into H, then $H \cup \{ta, tb\}$ is biconnected and there exist two vertex disjoint paths from t to y through a and b respectively. So there exist two disjoint paths P_a and P_b from $A = \{y, y\}$ and $B = \{a, b\}$, and let $P_1 = P_a \cup P_b$. Let P_2 be the a-to-b path that is taken as the first ear of χ. If P_a and P_b share any terminal-bounded trees, we could modify the two paths by Lemma 2.24 for $A = \{y, y\}$ and $B = \{a, b\}$ such that the new paths P'_a and P'_b from A to B are vertex disjoint

and do not share any terminal-bounded tree. Further, we shortcut P'_a and P'_b such that they have at most one subpath in each terminal-bounded tree. Since y is a terminal, $C = P'_a \cup P'_b \cup P_2$ satisfies the conditions of Claim 2.25, giving the Connectivity Separation.

Case 2: $\mathbf{y \in G \setminus H}$ In this case x and y may or may not be in the same component of $G \setminus H$.

Suppose x and y are in the same component χ of $G \setminus H$.

If there is an a-to-b path P_1 in χ that contains both of x and y, we take P_1 as the first ear of χ. Let P_2 be an a-to-b path in H. We shortcut P_2 in each terminal-bounded tree such that each terminal-bounded tree contains at most one subpath of P_2. Let the cycle C be composed by P_1 and P_2. Then by Claim 2.25, every C-to-C path contains a terminal, giving the Connectivity Separation.

If there is no such P_1, we take as the first ear of χ an a-to-b path that contains x, and take the second ear containing y. Then there is a cycle C containing x and y in the first two ears. Let P be any C-to-C path. Since any pair of vertices in $\chi \cap H$ are not in the same terminal-bounded tree (as argued for a and b), P will contain a terminal if it contains an edge in H. If P does not contain an edge of H, then P will contain a terminal by a similar proof of Lemma 2.9. So P always contains a terminal, giving the Connectivity Separation.

Suppose x and y are not in the same component. Let χ' be the component of $G \setminus H$ that contains y and let a' and b' be two common vertices of χ' and H. Then a' and b' are not in the same terminal-bounded tree of H as previously argued. We argue there exist two disjoint paths from $A = \{a, b\}$ to $B = \{a', b'\}$ in H: add two new vertices s and t, and four new edges sa, sb, ta' and tb' into H, then $H \cup \{sa, sb, ta', tb'\}$ is biconnected and there are two vertex disjoint paths from s to t that contains two paths P_a and P_b from A to B. Let P_1 (or P_2) be the a-to-b (or a'-to-b') path that is taken as the first ear of χ (or χ'). If P_a and P_b share any terminal-bounded trees, we can modify them by Lemma 2.24 for A and B such that the new paths P'_a and P'_b from A to B are vertex disjoint and share no terminal-bounded tree. Further, we can shortcut P'_a and P'_b such that each terminal-bounded tree contains at most one subpath of P'_a and P'_b. Let the cycle C be composed by P'_a, P'_b, P_1 and P_2. Then every terminal-bounded tree of H and every component of $G \setminus H$ contain at most one strict subpath of C. By Claim 2.25 every C-to-C path contains a terminal, giving the Connectivity Separation.

This completes the proof of the Connectivity Separation Theorem.

2.3.2 Proof of Tree Cycle Theorem

Let G be a minimal Q_3-triconnected planar graph. We prove the Tree Cycle Theorem for the contracted Q_3-triconnected graph H obtained from G. If the theorem is true for H, then it is true for G since subdivision will maintain the properties of the theorem. By Lemmas 2.14 and 2.15, if there are parallel edges in H, then either $|Q_3| = 2$ and H consists of three parallel edges or $|Q_3| = 3$ and H is a triangle with three pairs of parallel edges. The Tree Cycle Theorem is trivial for these two cases.

Proof Overview We focus on a maximal terminal-free tree T^*, rooted arbitrarily, of H and the corresponding terminal-bounded component T (that is, $T^* \subset T$). We view T^* as a set $\mathcal{P}$ of root-to-leaf paths. We show that we find a cycle for each path in $\mathcal{P}$ that strictly encloses only vertices on the paths. The outer cycle of the cycles for all the paths in $\mathcal{P}$ defines $C(T)$. See Figure 2.4. Property (a) directly follows from the construction. Property (b) is proved by induction on the number of root-to-leaf paths of T: when we add a new cycle for a path from $\mathcal{P}$, the new outer cycle will only strictly enclose vertices of the root-to-leaf paths so far considered. After that, we show any two terminals are triconnected when T is a tree: by modifying the three paths between terminals in a similar way to Lemma 2.24, only one path will require edges in T. Since T is connected, this proves T is a tree by minimality of H. Combining the above properties and triconnectivity of H, we can obtain property (c). Property (d) is proved by contradiction: if there is another terminal-bounded tree T' that shares two terminal-free paths of $C(T)$, then there is a terminal-free path in T'. We can show there is a removable edge in this path of T', contradicting Lemma 2.18.

Property (a) To prove that $C(T)$ exists, we will prove the following after proving some lemmas regarding terminal-free paths (Section 2.3.2.1), which guarantees that there is a drawing of H such that T^* is enclosed by some cycle:

Lemma 2.27. *There is a face of H that does not touch any internal vertex of T^*.*

We will also prove the following two lemmas in Section 2.3.2.1:

Lemma 2.28. *For any terminal-free path P, there is a drawing of H in which there is a simple cycle that strictly encloses all the vertices of P and only the vertices of P.*

Lemma 2.29. *Let C_1 and C_2 be two nested simple cycles of H such that the edges of C_1 are enclosed by C_2, and C_1 and C_2 share at most one subpath. Let xy be an edge strictly enclosed by C_2 and not enclosed by (or on) C_1. If H satisfies the following conditions, then xy is removable:*

1. *$C(xy)$ is vertex-disjoint with the common subpath of C_1 and C_2, and consists of two vertex-disjoint C_1-to-C_2 paths, a subpath of C_1 and a subpath of C_2 respectively.*
2. *For every neighbor z of xy in $C(xy) \setminus \{C_1, C_2\}$, there is a z-to-C_i path that shares only z with $C(xy)$ (for $i = 1$ or $i = 2$).*

Taking the face of H guaranteed by Lemma 2.27 as the infinite face, for any path of T^* this drawing guarantees a cycle as given by Lemma 2.28. Arbitrarily root T^* and let $\mathcal{P}$ be a collection of root-to-leaf paths that minimally contains all the edges of T^*. For path $P \in \mathcal{P}$, let C_P be the cycle that is guaranteed by Lemma 2.28 which encloses the fewest faces. By the maximality of T^*, the neighbors of P's endpoints in C_P are terminals. Since P is a path of T^* and T^* is a maximal terminal-free tree and since, by Lemma 2.28, C_P strictly encloses only the vertices of P, the neighbors of P on C_P are either terminals or vertices of T^*.

We construct $C(T)$ from $\bigcup_{P \in \mathcal{P}} C_P$. Consider any order $P_1, P_2, \ldots, P_{|\mathcal{P}|}$ of the paths in $\mathcal{P}$. Let $C^1 = C_{P_1}$ and let C^i be the cycle bounding the outer face of $C^{i-1} \cup C_{P_i}$ for $i = 2, \ldots, |P|$. Inductively, C^{i-1} bounds a disk, and strictly encloses $P_1, \ldots, P_{i-1}$. Also, C_{P_i} bounds a disk that overlaps C^{i-1}'s disk. We define $C(T) = C^{|\mathcal{P}|}$. It follows that $C(T)$ strictly encloses T^* (giving property (a)). That $C(T)$ encloses the fewest faces will follow from properties (b) and (c). An example is given in Figure 2.4.

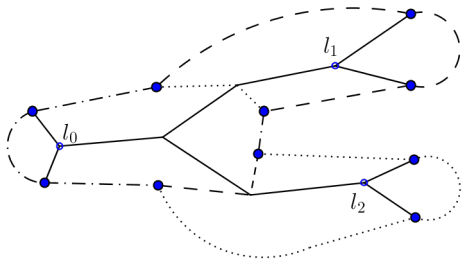

Figure 2.4: Illustration of $C(T)$. The dashed cycle is C_P for P from l_0 to l_1 and the dotted cycle is $C_{P'}$ for P' from l_0 to l_2. The outer boundary forms $C(T)$.

Property (b) We prove the following lemma and Property (b) follows from $C(T) = C^{|\mathcal{P}|}$.

Lemma 2.30. *C^i strictly encloses only the vertices of $\bigcup_{j\le i} P_j$ and encloses fewest faces.*

Proof. We prove this lemma by induction by assuming the lemma is true for C^{i-1}. The base case (C^1) follows from Lemma 2.28. Refer to Figure 2.5 (a). Since all the paths in $\mathcal{P}$ share the root r of T^* as one endpoint, C^{i-1} and C_{P_i} both strictly enclose r and so must enclose a disk enclosing r. This disk is bounded by a cycle C consisting of four subpaths: two vertex-disjoint subpaths R_1 and R_2 of $C^{i-1} \cap C_{P_i}$ and subpaths $R_3 \subseteq C^{i-1}$ and $R_4 \subseteq C_{P_i}$ each connecting R_1 and R_2.

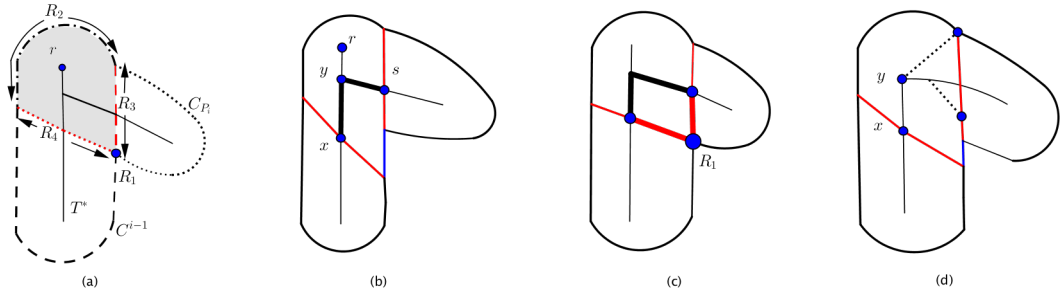

Figure 2.5: Examples for Lemma 2.30. (a) The dotted cycle is C_{P_i} and the dashed cycle is C^{i-1}. The two red paths are R_3 and R_4. R_1 is trivial. The shaded region represents the common faces enclosed by C^{i-1} and C_{P_i}. (b) The bold path is P^* and y is strictly inside of C. (c) If R_1 is a vertex, then the bold cycle only contains one terminal: R_1, which has two neighbors in $\bigcup_{j\le i} P_j$. (d) The dotted paths represent possible y-to-R_3 paths inside of C that is different from P_i.

Notice that R_3 must be enclosed by C_{P_i} and contains a vertex of T^*, since C^{i-1} strictly encloses only $\bigcup_{j<i} P_j$ and so is crossed by P_i at some vertex in R_3. Similarly, R_4 must be enclosed by C^{i-1} and contains a vertex in T^*. So there is an R_4-to-R_3 path P^* in $\bigcup_{j\le i} P_j$ enclosed by C. Let xy be the first edge in P^* with $x \in R_4$ (Figure 2.5 (b)). Then we have the following observations about xy:

Observation 2.31. *Edge xy is nonremovable (since neither x nor y are terminals, Lemma 2.18).*

Observation 2.32. *Vertex y is strictly enclosed by C.*

Proof. Let s be the endpoint of P^* in R_3. Then $s \in P_i$ since s is an internal vertex of R_3 and strictly enclosed by C_{P_i}. The r-to-s path Ψ_1 of T^* is strictly enclosed by C_{P_i}, so Ψ_1 is a subpath of P_i. The r-to-x path Ψ_2 of T^* is strictly enclosed by C^{i-1}, so Ψ_2 is a subpath of $\bigcup_{j<i} P_j$. Note the lowest common ancestor $LCA_{T^*}(x, s)$ of x and s in T^* is in $\Psi_1 \cap \Psi_2$. So $LCA_{T^*}(x, s) \neq x$ since $LCA_{T^*}(x, s) \in \Psi_1 \subseteq P_i$ and $LCA_{T^*}(x, s) \neq s$ since $LCA_{T^*}(x, s) \in \Psi_2 \subseteq \bigcup_{j<i} P_j$. See Figure 2.5 (b). It follows $LCA_{T^*}(x, s)$ is strictly enclosed by C. Since y is between x and $LCA_{T^*}(x, s)$ in P^*, we know $y \notin R_3$ and the claim follows. □

We argue that R_1 and R_2 must each contain at least one edge and so it will follow that R_3 and R_4 are vertex-disjoint. For a contradiction, assume R_1 is a vertex. Refer to Figure 2.5 (c). Consider the cycle C^* formed by $\bigcup_{j\leq i} P_j$, a subpath of R_3, R_1 and a subpath of R_4. By Theorem 2.20, C^* must contain a terminal. By construction, any terminal in C^* must be in R_1. So R_1 is a terminal t. Since t is the crossing of C^{i-1} and C_{P_i}, t's degree is at least four. By Lemma 2.21, there is an edge in C incident to t that is removable, which contradicts the minimality of H. The same argument holds for R_2. By the same reasoning, we have the following observation.

Observation 2.33. *Any vertex in* $(C^{i-1} \cup C_{P_i}) \setminus \bigcup_{j\leq i} P_j$ *has at most one neighbor in* $\bigcup_{j\leq i} P_i$.

Next, we will prove by contradiction that the subpath $S = C_{P_i} \setminus E(\bigcup_j R_j)$ only shares endpoints with C^{i-1}. This will imply that C^i strictly encloses only vertices of P_i and vertices strictly inside of C^{i-1}. Further, this will also imply that C^i encloses fewest faces. If there is another cycle C^I that strictly encloses all vertices of $\bigcup_{j\leq i} P_j$ and fewer faces than C^i, then there is some face that is enclosed by C^i but not enclosed by C^I. If that face is inside of C^{i-1}, then C^{i-1} is not the cycle that encloses $\bigcup_{j<i} P_j$ and fewest faces, contradicting our inductive hypothesis; if that face is inside of C_{P_i}, then C_{P_i} is not the cycle that encloses P_i and fewest face, contradicting our choice of C_{P_i}. So there can not be such cycle C^I, giving the lemma.

To prove that S only shares endpoints with C^{i-1}, we first have some claims.

Claim 2.34. *The y-to-R_3 subpath of P_i is the only one y-to-R_3 path whose edges are strictly enclosed by C.*

Proof. By Observation 2.32, y is strictly enclosed by C. For a contradiction, assume there is another y-to-R_3 path Ψ enclosed by C that is not a subpath of P_i. Notice that the endpoint of Ψ in R_3 can not be an endpoint of R_3, for otherwise this endpoint of R_3 has two neighbors in $\bigcup_{j\leq i} P_j$, contradicting Observation 2.33. Then we know $\Psi \subseteq T^*$, and we have two y-to-R_3 paths in T^*, which together with R_3 form a cycle without any terminals, contradicting Theorem 2.20. See Figure 2.5 (d). □

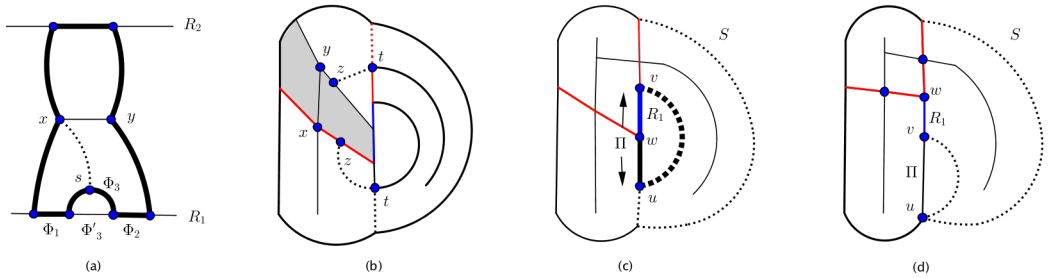

Figure 2.6: Examples for Lemma 2.30. (a) The bold cycle is $C(xy)$. (b) $C(xy)$ bounds the shaded region contains. The dotted paths are some possible examples for Φ. (c) The dotted path is S, which shares a subpath with C^{i-1}. The bold cycle is C'. R_3 and R_4 are disjoint. Π contains R_1. (d) Π does not contain R_1, and the endpoint w of R_1 has two neighbors in $\bigcup_{j\leq i} P_j$.

Claim 2.35. *$C(xy)$ consists of two vertex-disjoint R_1-to-R_2 paths, a subpath of R_1 and a subpath of R_2.*

Proof. We show the two x-to-y subpaths of $C(xy)$ share at least an edge with R_1 and R_2 respectively, that implies $C(xy)$ must contain two disjoint R_1-to-R_2 paths: one through x and the other through y.

We first argue that the two x-to-y paths in $C(xy)$ must each contain at least one terminal in $R_1 \cup R_2$. $C(xy) \cup xy$ contains two cycles whose intersection is xy. By Theorem 2.20, each of these cycles must contain a terminal. Since neither x nor y are terminals, each of the x-to-y paths in $C(xy)$ must contain a terminal. Note that $C(xy)$ is enclosed by C and there is no terminal strictly enclosed by C. Further, since R_3 is enclosed by C_{P_i}, all its internal vertices are not terminals. Similarly, all internal vertices of R_4 are not terminals. So all the terminals in C are either in R_1 or R_2, and each x-to-y path in $C(xy)$ contains at least one terminal in $R_1 \cup R_2$.

Next, we show the two terminals can not be both in R_1 or R_2, which implies the two paths must share vertices with R_1 and R_2 respectively. Notice that P^* divides the region inside of C into two parts, each of which has R_1 and R_2 in the bounding cycle respectively. If $C(xy)$ only contains vertices in R_1 or R_2, then $C(xy)$ must cross P^* and there is another cycle that encloses xy and fewer faces than $C(xy)$, contradicting the definition of $C(xy)$. Then $C(xy)$ must contain two R_1-to-R_2 paths: one through x and the other through y.

Both of $C(xy) \cap R_1$ and $C(xy) \cap R_2$ can not be a vertex, for otherwise the vertex has two neighbors in $\bigcup_{j \leq i} P_j$, contradicting Observation 2.33. So the two R_1-to-R_2 paths are vertex disjoint.

We then argue that $C(xy)$ could only share one subpath with R_1, and the same argument holds for R_2. For a contradiction, assume there are more than one subpath of $C(xy)$ in R_1. Then let Φ_1 and Φ_2 be two such subpaths that are connected by a subpath Φ_3 of $C(xy)$ that is not in R_1. Refer to Figure 2.6 (a). Notice that Φ_3 is enclosed by a cycle C' consisting of R_1, an x-to-R_1 subpath of $C(xy)$, an y-to-R_1 subpath of $C(xy)$ and edge xy. Let Φ_3' be the subpath of R_1 that has the same endpoints as Φ_3. If Φ_3 is an edge, then $(C^{i-1} \cup \Phi_3) \setminus \Phi_3'$ is a cycle that strictly encloses $\bigcup_{j<i} P_j$ and fewer faces than C^{i-1}, contradicting our inductive hypothesis. If Φ_3 contains an internal vertex s, then $s \in \bigcup_{j<i} P_j$ since it is strictly enclosed by C^{i-1}. So there is an s-to-x path in $\bigcup_{j<i} P_j$ whose edges must be enclosed by C'. And then by replacing the x-to-R_1 subpath of $C(xy)$ with the s-to-x path, we obtain a cycle that encloses xy and fewer faces than $C(xy)$, contradicting the definition of $C(xy)$. □

Claim 2.36. *For every neighbor z of xy in $C(xy) \setminus \{R_1, R_2\}$, there is a z-to-$(C^i \cup R_1)$ path that shares only z with $C(xy)$.*

Proof. If any neighbor z of xy in $C(xy)$ is not in $R_1 \cup R_2$, then z must be on an R_1-to-R_2 subpath of $C(xy)$, whose internal vertices are in $\bigcup_{j \leq i} P_j$. We construct a z-to-$(C^i \cup R_1)$ path Φ as follows: first find a z-to-C^{i-1} subpath Φ_1 through $\bigcup_{j<i} P_j$, and then find a subpath Φ_2 of C^{i-1} that connects Φ_1 with $C^i \cup R_1$ and is disjoint with $C(xy)$. Let t be the endpoint of Φ_1 in C^{i-1}. If $z \in C^{i-1}$, then Φ_1 is empty and $t = z$. If $t \in C^i \cup R_1$, then Φ_2 is empty. We define $\Phi = \Phi_1 \cup \Phi_2$. In the following, we first argue that if Φ_1 is not empty then it only shares z with $C(xy)$, and then show that if Φ_2 is not empty, then Φ only shares z with $C(xy)$.

Assume Φ_1 is not empty. Since H is triconnected and $C(xy)$ encloses fewest faces, there exists a path Φ_1 from z to C^{i-1} that is outside of $C(xy)$. Since $(\Phi_1 \setminus \{t\}) \subseteq \bigcup_{j<i} P_j$, Φ_1 can not share any internal vertex with $C(xy)$. Further, Φ_1 and $C(xy)$ can not share t, for otherwise t will be an endpoint of $C(xy) \cap R_1$ or $C(xy) \cap R_2$ and it has two neighbors in $\bigcup_{j\leq i} P_j$: one via Φ_1 and the other via $C(xy)$, contradicting Observation 2.33. So Φ_1 only shares z with $C(xy)$.

Consider the possible position of t in C^{i-1}. If it is in $C^i \cup R_1$, then Φ_2 is empty. If not, then it will be strictly enclosed by C^i. Refer to Figure 2.6 (b). There are two cases.

Case 1. If t is in R_3, then we choose as Φ_2 a subpath of R_3.

Note that in this case z could only be y's neighbor by planarity and Claim 2.35. Consider the position of z.

If $z \notin R_3$, we argue R_3 and $C(xy)$ are vertex-disjoint, which will imply that there always exists a t-to-C^i subpath of R_3. For a contradiction, assume R_3 and $C(xy)$ are not disjoint. Then $C(xy)$ must contains a y-to-R_3 subpath by planarity. Further, $\{yz\} \cup \Phi_1$ witnesses another y-to-R_3 path whose edges are strictly enclosed by C, contradicting Claim 2.34. So R_3 and $C(xy)$ are vertex-disjoint.

If $z \in R_3$, we argue R_3 contains a z-to-$(C^i \cup R_1)$ subpath, which shares only z with $C(xy)$. By Claim 2.35, $C(xy)$ contains two R_1-to-R_2 paths, one of which contains y. Then there are two y-to-C subpaths of $C(xy)$: an y-to-R_1 subpath and an y-to-R_2 subpath, which are in distinct regions divided by P^* and enclosed by C. By Claim 2.34, there is only one y-to-R_3 path whose edges are strictly enclosed by C. So only one of the y-to-C subpath of $C(xy)$ shares vertices with R_3, and the other one is vertex-disjoint with R_3. If the y-to-R_i subpath is disjoint with R_3 for $i = 1$ or $i = 2$, then there is a z-to-R_i subpath of R_3 that shares only z with $C(xy)$. Since $R_2 \subseteq C^i$, we have the z-to-$(C^i \cup R_1)$ subpath of R_3 that shares only z with $C(xy)$.

Case 2. If t is not in R_3, then we choose as Φ_2 the t-to-C^i subpath of C^{i-1} that is enclosed in C^i. This subpath always exists and is disjoint with $C(xy)$.

□

Now we prove that S only shares endpoints with C^{i-1}. For a contradiction, assume S has an internal vertex that is in C^{i-1}. Then by construction C^i would enclose either

R_1 or R_2; w.l.o.g. assume R_1 is enclosed by C^i. Let Π be the minimal subpath of C^{i-1} that is enclosed by C^i and connects an internal vertex u of S with the common endpoint v of R_1 and S. Let w be the other endpoint of R_1. There is a simple cycle C' consisting of two u-to-v subpaths: one is Π and the other is a subpath of S. See Figure 2.6 (c). Further, Π, and also C', contains R_1, for otherwise R_3 and R_4 share w as an endpoint and w has two neighbors in $\bigcup_{j\leq i} P_j$: one via R_3 and the other via R_4, contradicting Observation 2.33. See Figure 2.6 (d). It follows that R_3 and R_4 are vertex disjoint C'-to-C^i paths.

We argue H and xy satisfy the conditions of Lemma 2.29 with $C_1 = C'$ and $C_2 = C^i$, which shows xy is removable, giving a contradiction to Observation 2.31:

Condition 1. Note that C_1 and C_2 could only share at most one vertex which is u and shown in Figure 2.6 (d). Since C_{P_i} is simple, $u \neq w$, so $u \notin C$. Since $C(xy)$ is enclosed by C, $u \notin C(xy)$. So $C(xy)$ does not contain the common vertex of C_1 and C_2. Then the first condition in Lemma 2.29 follows from Claim 2.35.

Condition 2. Since $R_1 \subseteq C_1$, the second condition follows from Claim 2.36. Note that the z-to-C^i path Φ constructed in Claim 2.36 may contain a z-to-C_1 subpath, which shares only z with $C(xy)$.

□

T is a tree To prove this, we show that when T is a tree, H is Q_3-triconnected. That is, for any pair of terminals x and y, there are three x-to-y internally vertex-disjoint paths only one of which contains internal vertices of T when T is a tree. Since T is connected, this implies T can not contain any cycle, for otherwise H is not minimal. The proof is similar to that of Lemma 2.24 but simpler: we modify the paths between x and y such that only one of them contains internal vertices of T while maintaining them internally vertex-disjoint.

Let R_1, R_2 and R_3 be three x-to-y disjoint paths. If there is only one path containing internal vertices of T, then it is sufficient for T to be simply connected for triconnectivity between x and y. So we assume there are at least two paths containing internal vertices of T. By Lemma 2.30, $C(T)$ strictly encloses all internal vertices of T, so any path that contains an internal vertex of T must touch $C(T)$. We order the vertices of the three

paths from x to y. Let u_i and v_i be the first and last vertex of R_i that is in $C(T)$ for $i = 1, 2, 3$. If R_i is disjoint from $C(T)$, we say u_i and v_i is undefined. Let C_1 and C_2 be the disjoint minimum paths of $C(T)$ that connect $\{u_1, u_2, u_3\}$ and $\{v_1, v_2, v_3\}$. If there are only two paths, say R_1 and R_2, containing vertices of $C(T)$, we can replace $R_1[u_1, v_1] \cup R_2[u_2, v_2]$ with $C_1 \cup C_2$. Then the new paths are disjoint and do not contain any internal vertex of T. If all the three paths contain vertices of $C(T)$, then let C_3 be the simple path from the middle vertex of $\{u_1, u_2, u_3\}$ to the middle vertex of $\{v_1, v_2, v_3\}$ in $(C(T) \cup T) \setminus (C_1 \cup C_2)$. Now the to-$C(T)$ prefices of R_i, C_i and the from-$C(T)$ suffices of R_i (for $i = 1, 2, 3$) together form three vertex-disjoint paths between x and y. Further, among the three resulting paths, only the path containing C_3 contains internal vertices of T. Therefore, it is sufficient for T to be simply connected for H to be Q_3-triconnected.

Property (c) By triconnectivity, every leaf of T^* has at least two neighbors on T that are terminals. So each leaf of T^* can not be a leaf of T and then all leaves of T are terminals. By Lemma 2.30, $C(T)$ only strictly encloses all vertices of T^*. So all its neighbors, which are terminals in T by the maximality of T^*, are on $C(T)$, giving property (c).

Since each terminal-bounded component is a tree, any terminal on T must be a leaf by the construction of terminal-bounded component. Therefore, we have the following lemma.

Lemma 2.37. *A vertex of a terminal-bounded tree T is a terminal if and only if it is a leaf of T.*

Property (d) Let T_1 be any terminal-bounded tree. For a contradiction, assume there is a terminal-bounded tree T_2 whose vertices are in distinct maximal terminal-free paths of $C(T_1)$. By Lemma 2.37, the leaves of T_2 are terminals. So each component of $C(T_1) \cap T_2$ is a terminal-to-terminal path and contains at most one maximal terminal-free path. Then by the assumption for the contradiction, there are two vertex-disjoint non-trivial components (paths) Π_1 and Π_2 of $C(T_1) \cap T_2$.

Since $C(T_2)$ strictly encloses only internal vertices of T_2, the interiors of $C(T_2)$ and $C(T_1)$ overlap, and so $C(T_2) \cap T_1 \neq \emptyset$. Then $C(T_2) \cap T_1$ contains only paths whose endpoints are terminals on $C(T_1)$. Let $\mathcal{R}$ be the set consisting of all maximal paths

of $C(T_2) \cap T_1$. Refer to Figure 2.7 (a) and (b). Since $C(T_2)$ is simple, and since the endpoints of paths in $\mathcal{R}$ are leaves of T_1, we have

Observation 2.38. *Any two paths in $\mathcal{R}$ are vertex disjoint.*

Note that if T_1 is an edge or a star, $|\mathcal{R}| \leq 1$ and so we would already have our contradiction.

For any path $R_i \in \mathcal{R}$ with endpoints u_i and v_i, there is an u_i-to-v_i subpath R'_i of $C(T_1)$ such that R_i and R'_i form a cycle enclosing a region that is enclosed by both of $C(T_1)$ and $C(T_2)$. Since $C(T_1)$ and $C(T_2)$ only strictly enclose edges of T_1 and T_2 respectively, and since $T_1 \neq T_2$, this region must be a face. Notice that any path of $C(T_1) \cap T_2$ could only be subpath of R'_i for some $R_i \in \mathcal{R}$ for $C(T_2)$ encloses T_2. By the following observation, there exist R_1 and R_2 such that $\Pi_1 \subseteq R'_1$ and $\Pi_2 \subseteq R'_2$.

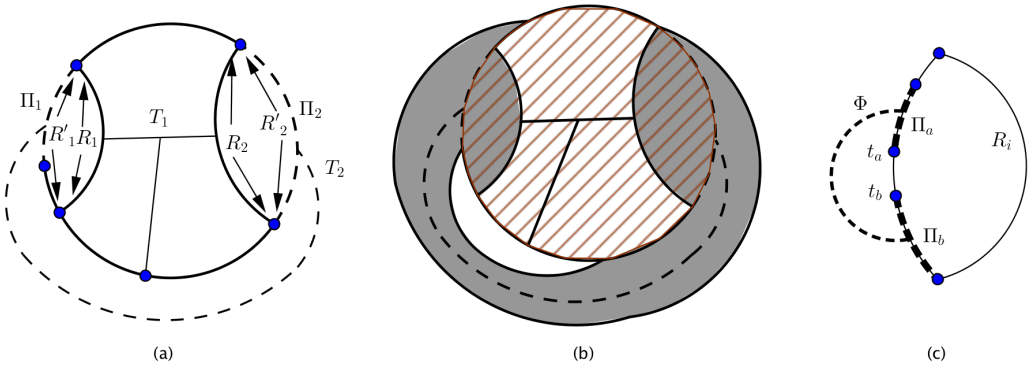

Figure 2.7: (a) The dashed tree is T_2 and there are two paths Π_1 and Π_2 of T_2 on $C(T_1)$. The blue vertices are terminals. (b) The two shaded cycles are (T_1) and $C(T_2)$, which shares two regions enclosed by R_i and R'_i for $i = 1, 2$. (c) The dashed subtree is in T_2. R'_i contains two subpaths of T_2 and two terminals t_a and t_b are enclosed.

Observation 2.39. *Every R'_i contains at most one maximal path of T_2.*

Proof. For a contradiction, assume there is a path R'_i that contains more than one maximal path of T_2. Let Π_a and Π_b be two successive such paths. By the concept of terminal-bounded tree, there is a terminal-free path Φ of T_2 connecting Π_a with Π_b that only share endpoints with $C(T_1)$. Refer to Figure 2.7 (c). So Φ, R_i and R'_i (which contains two R_i-to-Φ paths) witness a cycle C that strictly encloses an endpoint t_a of

Π_a and an endpoint t_b of Π_b. Since the endpoints of Π_a and Π_b are leaves of T_2, t_a and t_b should be in $C(T_2)$ by Property (c). So there is a subpath P of $C(T_2)$ that is enclosed by C such that $t_a \in P$. Since the region enclosed by C is divided into two parts by a subpath Φ' of R'_i (which have the same endpoints as Φ), and since one of the two parts is the face enclosed by R_i and R'_i, we know P could only be in the region bounded by Φ' and Φ. However, P can not cross Φ by Property (a) since Φ is terminal-free and every vertex of Φ is an internal vertex of T_2; and P can not cross any internal vertex of Φ' for otherwise P will enter the face enclosed by R_i and R'_i. Therefore, P and R_i can not be connected, contradicting P is a subpath of $C(T_2)$. □

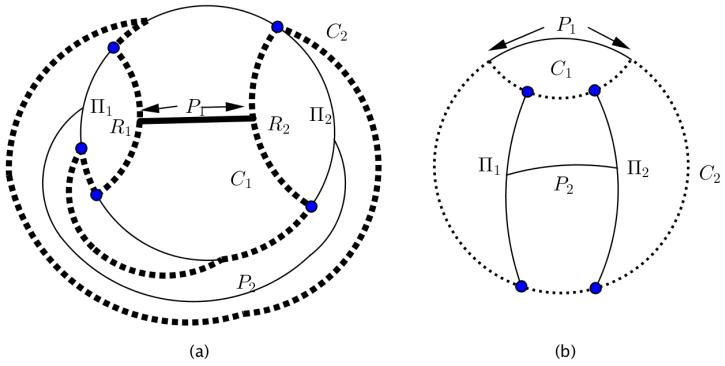

Figure 2.8: (a) The dotted cycle is $C(T_2)$. The bold cycles C_1 and C_2 share P_1. (b) An example for Simplified C_1 and C_2 in (a). The dotted cycle is $C(T_2)$ and the outer cycle is C_2. C_1 and C_2 share P_1.

Next, we construct two cycles C_1 and C_2 that share a subpath. Refer to Figure 2.8 (a). Since T_1 is a tree and R_1 and R_2 are vertex disjoint by Observation 2.38, there is an R_1-to-R_2 subpath P_1 in T_1. Since $C(T_2)$ is simple and contains R_1 and R_2, $C(T_1) \cup P_1$ contains two simple cycles C_1 and C_2 whose intersection is P_1. W.l.o.g. assume C_1 is enclosed by C_2.

Since T_2 is a tree and Π_1 and Π_2 are vertex disjoint, there is a Π_1-to-Π_2 path P_2 in T_2. Note that P_2 is terminal-free, since P_2 does not contain any leaf of T_2 and terminals in T_2 are leaves by Lemma 2.37. Let xy be an edge of P_2. Then by Lemma 2.18, xy is nonremovable. However, H and xy also satisfy Lemma 2.29 with C_1 and C_2, which shows xy is removable, giving a contradiction.

Condition 1. Note that $C(xy)$ is enclosed by the cycle C consisting of Π_1, Π_2, and two Π_1-to-Π_2 subpaths of $C(T_2)$: one is of C_1 and the other is of C_2. Refer to Figure 2.8 (a) and (b). Since C is disjoint with the common subpath P_1 of C_1 and C_2, $C(xy)$ is also disjoint with P_1.

Showing the remainder of the first condition of Lemma 2.29 is similar to that of Claim 2.35 if we replace R_1 and R_2 with the two Π_1-to-Π_2 subpaths of C_1 and C_2. Note that we have a stronger version of Observation 2.33 here, since T_2 is a tree.

Condition 2. If any neighbor z of xy in $C(xy)$ is not in $C_1 \cup C_2$, it will be a non-leaf vertex in T_2, since all leaves of T_2 are in $C(T_2) \subseteq C_1 \cup C_2$. Then z is not a terminal by Lemma 2.37. We find a subpath Φ in T_2 from z to $C(T_2) \subseteq C_1 \cup C_2$ such that Φ only shares z with $C(xy)$. By triconnectivity, there are at least three disjoint paths from z to $C(T_2)$. Since $C(xy)$ contains two such paths and encloses the fewest faces, there is a path Φ from z to $C(T_2)$ outside of $C(xy)$. If Φ shares any vertex with $C(xy)$ other than z, then Φ and the C_1-to-C_2 path that contains z witness a cycle in T_2, a contradiction.

This proves the Tree Cycle Theorem.

2.3.2.1 Terminal-free Paths

Let P be a terminal-free a-to-b path of H such that there exists a cycle that strictly encloses the internal vertices of P. Then $a, b \in C(P)$, since H is triconnected by Lemma 2.13. Let $P_1(P)$ and $P_2(P)$ be the two a-to-b subpaths of $C(P)$.

Lemma 2.40. *If for every edge $e \in P$ there is a separating set $S_e \in C(P)$, then all the vertices inside of $C(P)$ are in P.*

Proof. For a contradiction, assume there is a vertex u strictly inside of $C(P)$ that is not on P. There can not be more than one path from u to $C(P)$ disjoint from P, otherwise there will be another cycle which encloses fewer faces than $C(P)$ (Figure 2.9 (a)). By Lemma 2.13, H is triconnected, so there are at least two disjoint paths R_1 and R_2 from u to v_1 and v_2 on P disjoint from $C(P)$ (Figure 2.9 (b)). For an edge e on P between v_1 and v_2, every separating set for e must include a vertex on the path $R_1 \cup R_2$, however, this contradicts the assumption that there is a separating set for e in $C(P)$. □

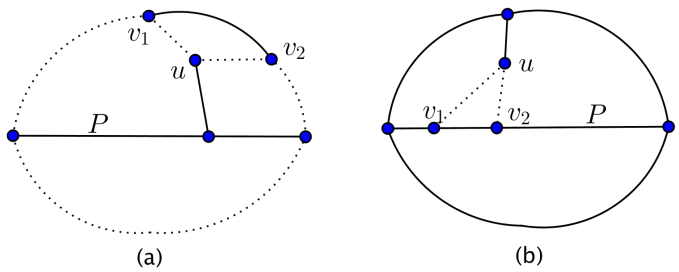

Figure 2.9: (a) If vertex u has two paths to v_1 and v_2 on $C(P)$ that are disjoint from P, then there is a smaller cycle (dotted) through v_1, u, v_2 that encloses P. (b) The separating set for an edge e on P between v_1 and v_2 must include a vertex of the path (dotted) strictly enclosed by $C(P)$, contradicting there is a separating set for e in $C(P)$.

Lemma 2.41. *No pair of adjacent vertices in P has a common neighbor in $C(P)$.*

Proof. For a contradiction, assume there are adjacent vertices u and v with a common neighbor z in $C(P)$. Since u, v, z forms a cycle, S_{uv} must contain z. Therefore, by Theorem 2.11, uz is removable. Further, both components of $H \setminus \{uv, S_{uv}\}$ must contain a vertex distinct from u and v and each of those vertices must have vertex disjoint paths to S_{uv} and uv; therefore the degree of z is at least 4. Therefore, removing uz will not result in contracting any edges incident to z and so will preserve triconnectivity of the terminals, contradicting the minimality of H. □

For a separating pair (e, S_e), let $\Sigma(e, S_e)$ be a closed curve that only intersects the drawing of H in an interior point of e and two vertices of S_e and partitions the plane according to the components of $H \setminus (\{e\} \cup S_e)$. Each portion of $\Sigma(e, S_e) \setminus H$ is contained in a face of H. Since this is true for any $\Sigma(e, S_e)$, for two separating pairs (e_1, S_{e_1}) and (e_2, S_{e_2}) we may assume that the curves $\Sigma(e_1, S_{e_1})$ and $\Sigma(e_2, S_{e_2})$ are drawn so they cross each other at most 3 times. $\Sigma(e_1, S_{e_1})$ and $\Sigma(e_2, S_{e_2})$ cross either at a point that is interior to a face of H or at one vertex of $S_{e_1} \cup S_{e_2}$. Since they are simple closed curves, they cross each other twice or not at all.

Lemma 2.42. *For every edge e in P, there exists a separating set for e in $C(P)$.*

Proof. We prove by induction on the subpaths of P: we assume the lemma is true for every strict subpath of P. The base case is when P is one edge xy: S_{xy} must include

one vertex of $P_1(xy)$ and one vertex of $P_2(xy)$ but does not include x or y, giving the lemma.

Let xy be any edge of P. Without loss of generality, we assume $x \neq a$ and $y \notin P[a, x]$. By the inductive hypothesis, there exists a separating set $S_{xy} = \{s_1, s_2\}$ in $C(P[x, b])$. The following claim simplifies our proof, which we prove after using this claim to prove that $S_{xy} \in C(P)$.

Claim 2.43. *S_{xy} does not contain an internal vertex of P.*

There are two cases:

1. If $b \in S_{xy}$, w.l.o.g. assume $b = s_1$. Then we only need to show s_2 is in $C(P)$. By the induction hypothesis, $s_2 \in C(P[x, b])$. By Claim 2.43, s_2 can not be internal vertex of $P[a, y]$, so a and x must be in the same component of $H \setminus \{xy, S_{xy}\}$. Then $C(P[a, y])$ must contain s_2 since $C(P[a, y])$ must intersect $\Sigma(xy, S_{xy})$ in vertices of S_{xy}. Therefore, s_2 is in both of $C(P[a, y])$ and $C(P[x, b])$. By planarity, it must be in $C(P)$, for otherwise there will be other cycles which enclose fewer faces than $C(P[a, y])$ and $C(P[x, b])$ and do not contain s_2.

2. If $b \notin S_{xy}$, by Claim 2.43, S_{xy} is not in P, so a and b are in the distinct components of $H \setminus \{xy, S_{xy}\}$ since they are connected to x and y by $P[a, x]$ and $P[y, b]$ respectively. That is, b is strictly inside of $\Sigma(xy, S_{xy})$ and a is outside of $\Sigma(xy, S_{xy})$. Then $C(P)$ must intersect $\Sigma(xy, S_{xy})$ twice in vertices of S_{xy}. Because $C(P)$ is simple, it can not intersect $\Sigma(xy, S_{xy})$ in the same vertex of S_{xy}. Therefore, both vertices of S_{xy} are in $C(P)$.

This completes the proof of Lemma 2.42. □

Proof of Claim 2.43. For a contradiction, assume s_1 is an internal vertex of P. Then it must in $P[a, x]$ since S_{xy} is in $C(P[x, b])$. Further, s_2 can not be also in $P[a, x]$ since x and y are triconnected to $C(P)$, and then $C(P)$ and the paths from x and y to $C(P)$ contains a path from x to y disjoint from $P[a, x]$.

Since a and x are on different sides of $\Sigma(xy, S_{xy})$, $C(P[a, x])$ must cross $\Sigma(xy, S_{xy})$ twice. $\Sigma(xy, S_{xy})$ could only intersect H at xy and S_{xy}, so $C(P[a, x])$ must cross $\Sigma(xy, S_{xy})$ at xy and s_2, for s_1 is in $P[a, x]$ and $C(P[a, x])$ is simple; w.l.o.g. assume $P_1(P[a, x])$ contains s_2.

In $P[s_1, x]$, there must be a vertex between s_1 and x, for otherwise by Theorem 2.11 edge s_1x is removable, which contradicts Lemma 2.18. Let zx be an edge of $P[s_1, x]$. Then there exists a separating set S_{zx} for zx in $C(P[a, x])$ by induction hypothesis. We claim $\Sigma(zx, S_{zx})$ must intersect $P_1(P[a, x])$ between s_2 and x. If not, then it will intersect $P_1(P[a, x])$ outside of $\Sigma(xy, S_{xy})$. However, the z-to-s_2 path, together with the s_2-to-x subpath of $P_1(P[a, x])$ and edge zx form a cycle inside of $\Sigma(xy, S_{xy})$. See Figure 2.10 (a). Since $\Sigma(zx, S_{zx})$ must cross the edge zx, it must intersect the described cycle twice. Note that cycle is disjoint from $P_2(P[a, x])$. So $\Sigma(zx, S_{zx})$ will intersect the drawing of H four times: twice at the described cycle inside of $\Sigma(xy, S_{xy})$, once at $P_1(P[a, x])$ outside of $\Sigma(xy, S_{xy})$ and once at $P_2(P[a, x])$. This contradicts the definition of $\Sigma(zx, S_{zx})$. Let s be the vertex of S_{zx} in $P_1(P[a, x])$. By the above argument, s is between s_2 and x. There are two cases.

1. If $s = s_2$, the only vertex of $P_1(P[a, x])$ inside both of $\Sigma(zx, S_{zx})$ and $\Sigma(xy, S_{xy})$ is s_2. Then there is a path between z and s_2 edge disjoint from $C(P[a, x])$ and P. By Lemma 2.40, all vertices inside of $C(P[a, x])$ are in $P[a, x]$, so z and s_2 are adjacent and this edge is the only path between z and s_2 disjoint from P in $\Sigma(xy, S_{xy})$. Consider $C(P[z, b])$. It must intersect $\Sigma(xy, S_{xy})$ at s_1 and s_2. Then edge zs_2 must be in $C(P[z, b])$ since it is the only path between z and s_2 disjoint from P in $\Sigma(xy, S_{xy})$. By Lemma 2.40, the x-to-s_2 path disjoint from P in $\Sigma(xy, S_{xy})$ is an edge. However, the two edges zs_2 and xs_2 contradict Lemma 2.41.

2. If $s \neq s_2$, S_{zx} must contain vertex a, for otherwise $\Sigma(zx, S_{zx})$ will intersect $P_1[a, x]$ between a and s_2, which is the second intersection for $\Sigma(zx, S_{zx})$ and $P_1[a, x]$ and the fourth for $\Sigma(zx, S_{zx})$ and H. Then s_1 and z are both connected to s by paths edge disjoint from $C(P[a, x])$ and P since they are inside of $\Sigma(zx, S_{zx})$. By Lemma 2.40 they are both adjacent to s. See Figure 2.10 (c). However, this contradicts Lemma 2.41. □

Lemma 2.44. *Let u_1 and u_2 be the neighbors of an endpoint of P on $C(P)$. Then there is a u_1-to-u_2 path whose internal vertices are strictly outside of $C(P)$.*

Proof. Without loss of generality, let u_i be the neighbor of a on $P_i(P)$, $i = 1, 2$. Let $\{s_1, s_2\}$ be the separating set for $ac \in P$ such that $s_i \in P_i(P)$, $i = 1, 2$, as guaranteed by Lemma 2.42.

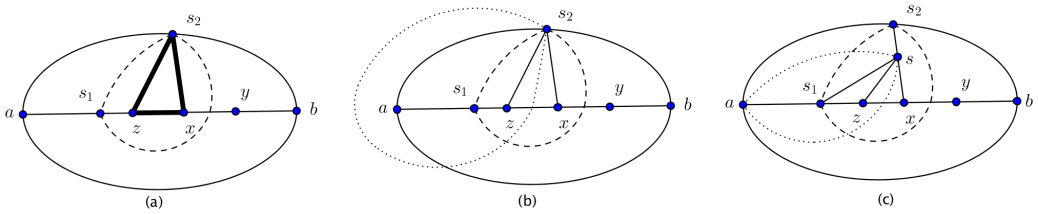

Figure 2.10: $\Sigma(xy, S_{xy})$ is the dashed cycle and $\Sigma(zx, S_{zx})$ is the dotted cycle. (a) The bold cycle must be intersected by $\Sigma(zx, S_{zx})$ twice. (b) $\Sigma(zx, S_{zx})$ intersects the drawing of H at s_2. (c) $\Sigma(zx, S_{zx})$ intersects the drawing of H at s and a.

If $u_i = s_i$, $i = 1, 2$, then the component of $H \setminus \{ac, s_1, s_2\}$ that contains a must contain another vertex x and x must have three vertex-disjoint paths to a, s_1 and s_2. The latter two of these witness the u_1-to-u_2 path that gives the lemma.

If $u_1 \neq s_1$ and $u_2 = s_2$, then the component of $H \setminus \{ac, s_1, s_2\}$ that contains u_1 must have three vertex-disjoint paths to a, s_1 and s_2 and the latter of these paths witness the u_1-to-u_2 path that gives the lemma. The case $u_1 = s_1$ and $u_2 \neq s_2$ is symmetric.

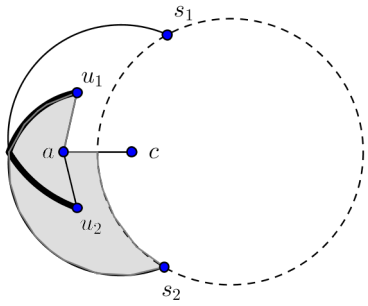

Figure 2.11: The construction of paths for Lemma 2.44. $\Sigma(ac, \{s_1, s_2\})$ is the dashed cycle. The u_2 side of the cycle that separates u_2 and s_1 is shaded. The path witnessing the lemma is bold and is formed by the u_2-to-s_1 and u_1-to-s_2 paths that avoid edges u_1a and u_2a.

If $u_i \neq s_i$, $i = 1, 2$, then the component of $H \setminus \{ac, s_1, s_2\}$ that contains u_1 and u_2 must have three vertex-disjoint paths from u_i to a, s_1 and s_2 for $i = 1, 2$. The first of these, we may assume is the edge u_ia. Consider the u_1-to-s_2 path; together with the edges u_1a and ac and the s_2-to-ac portion of $\Sigma(ac, \{s_1, s_2\})$ that does not contain s_1, these form a closed curve in the plane that separates u_2 and s_1 (see Figure 2.11). The

above-described u_2-to-s_1 path must therefore cross the u_1-to-s_2 path; these paths witness the u_1-to-u_2 path that gives the lemma. □

Lemma 2.45. *For any terminal-free path P, there is a drawing of H such that all internal vertices of P are strictly enclosed by a simple cycle.*

Proof. For a contradiction, assume there is a terminal-free path P whose internal vertices is not strictly enclosed by any cycle for any choice of infinite face of H: that is, every face of H contains an internal vertex of P. Let P be a minimal such path, let a and b be P's endpoints, and let c be a's neighbor on P. Note that $b \neq c$, for otherwise P is an edge and H would have at most two faces (each containing P), but every triconnected graph has at least three faces.

First observe that there is a face f whose bounding cycle strictly encloses all internal vertices of $P[c, b]$. Let f_1 and f_2 be the two faces that contains edge ac: f is one of f_1 and f_2. One of f_1 and f_2 only contains one internal vertex, namely c, of P, for otherwise both faces would contain at least two internal vertices of P and every face of H would contain an internal vertex of $P[c, b]$, contradicting the minimality of P.

Take f, defined in the previous paragraph, to be the infinite face of H. Since f's bounding cycle strictly encloses the internal vertices of $P[c, b]$, $C(P[c, b])$ exists. Let u and v be c's neighbors in $C(P[c, b])$. Note that ac may or may not be in $C(P[c, b])$. By Lemma 2.44, there is a u-to-v path R whose internal vertices are strictly outside of $C(P[c, b])$, so there is a drawing of H so that $R \cup C(P[c, b]) \setminus \{cu, cv\}$ is a cycle that strictly encloses internal vertices of P, a contradiction. See Figure 2.12 (a). □

Proof of Lemma 2.27. If T^* is an edge, then this claim is trivial. Suppose otherwise.

For a contradiction, assume every face of H contains an internal vertex of T^*. Let ac be a leaf edge of T^*, where a is a leaf of T^*. Recall that if a terminal-bounded component is not an edge, then it is obtained from a maximal terminal-free tree. By the maximality of T^*, the two neighbors, t_1 and t_2, of a on $C(ac)$ are terminals. By Lemma 2.44, there exists a t_1-to-t_2 path P' whose internal vertices are strictly outside of $C(ac)$. Choose P' such that $C' = P' \cup \{at_1, at_2\}$ encloses the fewest faces. Then C' does not strictly enclose any vertex since H is triconnected by Lemma 2.13. By the assumption for the contradiction and the choice of P', P' must contain an internal vertex b of T^*.

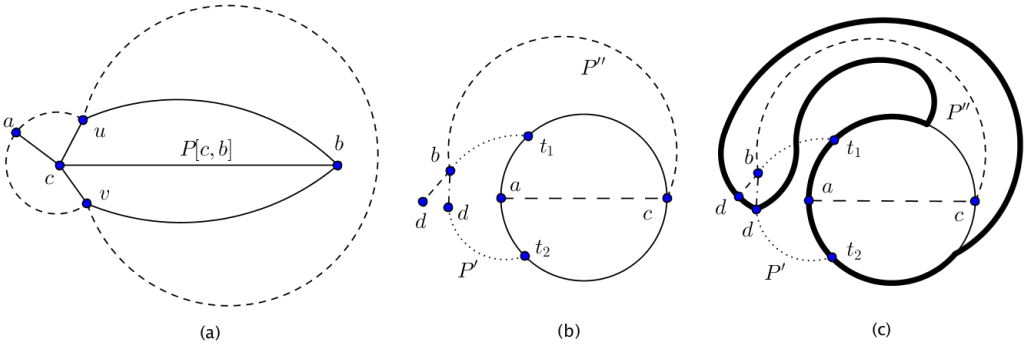

Figure 2.12: (a) The dashed paths show two possible u-to-v paths R outside of $C(P[c,b])$. The cycle $R \cup C(P[c,b]) \setminus \{cu, cv\}$ strictly encloses a face that does not contain any internal vertex of P for an appropriate choice of infinite face. (b) The dotted path is P' and dashed path is P''. Note that d may or may not be in P'. (c) The bold cycle is $C(P'')$ and it crosses P'.

Let P^* be the a-to-d path of T^* containing vertex b. Note that d may or may not be in P' (see Figure 2.12 (b)). By Lemma 2.45, $C(P^*)$ exists and by Lemmas 2.40 and 2.42, $C(P^*)$ strictly encloses only internal vertices of P^*. Then C' and $C(P^*)$ must cross each other and there exists a subpath of $C(P^*)$ that is strictly enclosed by C', which contradicts that C' encloses fewest faces. See Figure 2.12 (c). □

Proof of Lemma 2.28. By Lemma 2.45, $C(P)$ exists. Then $P_1(P)$ and $P_2(P)$ both contain a terminal, for otherwise there is a cycle composed by P and one of $P_1(P)$ and $P_2(P)$ that does not contain any terminal, contradicting Theorem 2.20. We first prove that at least one of $P_1(P)$ and $P_2(P)$ contains more than one internal vertex, and then construct the cycle strictly enclosing P. By Lemma 2.40 and 2.42, all vertices inside of $C(P)$ are in P. If $P_1(P)$ and $P_2(P)$ both only contain one internal vertex, then the endpoints of the first edge of P are both adjacent to the internal vertex of $P_1(P)$ or $P_2(P)$, which contradicts Lemma 2.41. Let x_a and y_a (or x_b and y_b) be the neighbors of a (or b) in $P_1(P)$ and $P_2(P)$ respectively. Then at least one of $\{x_a, x_b\}$ and $\{y_a, y_b\}$ contains two distinct vertices.

By Lemma 2.44, there is an x_a-to-y_a path whose internal vertices are strictly outside of $C(P)$. We choose such an x_a-to-y_a path R_a such that the cycle $R_a \cup \{ax_a, ay_a\}$ encloses fewest faces. Then this cycle does not strictly enclose any vertex, for otherwise

the vertex strictly inside of the cycle is triconnected to the cycle and we can find another cycle through that vertex which could enclose fewer faces, contradicting the choice of R_a. Similarly, for x_b and y_b, we can find an x_b-to-y_b path R_b such that $R_b \cup \{bx_b, by_b\}$ does not strictly enclose any vertex. Since at least one of $\{x_a, x_b\}$ and $\{y_a, y_b\}$ contains two distinct vertices, R_1 and R_2 are distinct. So the cycle $(R_a \cup R_b \cup C(P)) \setminus \{ax_a, bx_b, ay_a, by_b\}$ strictly encloses all the vertices of P and only the vertices of P. □

Proof of Lemma 2.29. For a contradiction, assume edge xy is nonremovable and consider $\Sigma(xy, S_{xy})$.

First note that $\Sigma(xy, S_{xy})$ must be enclosed by C_2 and not enclosed by C_1 for otherwise, $\Sigma(xy, S_{xy})$ would intersect C_1 and C_2 in more than one point, and since C_1 and C_2 share at most one common subpath that is vertex disjoint from $C(xy)$ (by condition of the lemma), this would result in $|S_{xy}| \geq 3$, a contradiction. Therefore S_{xy} contains, w.l.o.g., two vertices of the C_1-to-C_2 path through $C(xy)$; let a be the vertex of S_{xy} on the x-to-C_1 path. Refer to Figure 2.13.

Next note that a must be a neighbor of x. For otherwise the neighbor z of x must be on the x side of $\Sigma(xy, S_{xy})$. By condition of the lemma, there is a path from z to C_1 or C_2 that is disjoint from $C(xy)$, however, the only way to cross $\Sigma(xy, S_{xy})$ is via a vertex of $C(xy)$, a contradiction. Therefore a is a neighbor of x.

Then a has degree 2 in the part of H on the x side of $\Sigma(xy, S_{xy})$: one degree is given by the edge ax and the other is given by the existence of a vertex $v \neq x$ on the x side of $\Sigma(xy, S_{xy})$ which has vertex disjoint paths to x and each vertex in S_{xy}. For the same reason, x has degree 4: degree 2 via the C_1-to-C_2 path, degree 1 via y and degree 1 via the v-to-x path.

Further we will argue that a has degree 2 on the y side of $\Sigma(xy, S_{xy})$ as well; a then has degree 4. By Theorem 2.11, xa is removable. Since x and a both have degree 4, removing xa will not result in any edge contractions; this maintains the triconnectivity of the terminals, and contradicts the minimality of H.

To show that a has degree 2 on the y side of $\Sigma(xy, S_{xy})$, we have two cases. If $a \in C_1$, then this follows from the two edges of C_1 incident to a. If $a \notin C_1$, then by condition of the lemma, there is a path from a to C_1 or C_2 that is disjoint from $C(xy)$ and so must be on the y side of $\Sigma(xy, S_{xy})$ and is notably disjoint from the 3 edges incident to a on the C_1-to-C_2 path of $C(xy)$ and on the v-to-a path. □

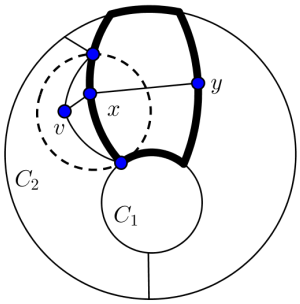

Figure 2.13: The bold cycle is $C(xy)$ and the dashed cycle is $\Sigma(xy, S_{xy})$.

2.4 Correctness of Spanner

In this section, we prove the correctness of our spanner. Let OPT be the weight of an optimal solution for 3-ECP. Then the correctness requires two parts: (1) bounding its weight by $O(\text{OPT})$ and (2) showing it contains a $(1+\epsilon)$-approximation of the optimal solution. The weight of our spanner is bounded by the weight of mortar graph, which we briefly introduce in Subsection 2.4.1.

The following Structure Theorem guarantees that there is a nearly-optimal solution in our spanner and completes the correctness of our spanner:

Theorem 2.46 (Structure Theorem)**.** *For any $\epsilon > 0$ and any planar graph instance (G, w, r) of 3-ECP, there exists a feasible solution S in our spanner such that*

- *the weight of S is at most $(1 + c\epsilon)\text{OPT}$ where c is an absolute constant, and*
- *the intersection of S with the interior of any brick is a set of trees whose leaves are on the boundary of the brick and each tree has a number of leaves depending only on ϵ.*

We prove the Structure Theorem in Subsection 2.4.2. The idea is similar to that for 2-ECP, that is we transform an optimal solution to a feasible solution satisfying the theorem. Throughout we indicate where the transformation for 3-ECP departs from those of 2-ECP. In the following, we denote by Q the set of terminals, which are the vertices with positive requirement.

2.4.1 Mortar Graph, Bricks and Portals

First we introduce some properties of the mortar graph and bricks. For a brick B, let ∂B be its boundary and $\text{int}(B) = E(B) \setminus E(\partial B)$ be its interior. A path is *ϵ-short* if the distance between every pair of vertices on that path is at most $(1+\epsilon)$ times the distance between them in G. Bricks have the following properties.

Lemma 2.47. (Lemma 6.10 [23] rewritten) *The boundary of a brick B, in counterclockwise order, is the concatenation of four paths W_B, S_B, E_B and N_B (west, south, east and north) such that:*

1. *Every vertex of $Q \cap B$ is in $N_B \cup S_B$.*
2. *N_B is 0-short and every proper subpath of S_B is ϵ-short.*

The paths that form eastern and western boundaries of bricks are called *supercolumns*, and the weight of all edges in supercolumns is at most ϵOPT (Lemma 6.6 [23]).

The *Mortar graph* is a grid-like subgraph of G that (1) spans Q and (2) has weight at most $9\epsilon^{-1}$ times the weight of a minimum Steiner tree that spans Q. Since the weight of minimum Steiner tree is no more than OPT, the weight of mortar graph is no more than $9\epsilon^{-1}\text{OPT}$. A brick B is the subgraph of G that is enclosed by a face of the mortar graph (including the boundary of the face); it has boundary ∂B and interior $\text{int}(B) = G[E(B) \setminus E(\partial B)]$ (a subgraph induced by $S \subseteq V(G)$ or $S \subseteq E(G)$ in G is denoted by $G[S]$). Further, bricks have the following property:

Lemma 2.48. (Lemma 6.10 [23]) *The boundary of a brick B, in counterclockwise order, is the concatenation of four paths W_B, S_B, E_B and N_B (west, south, east and north) such that:*

1. *The set of edges $B \setminus \partial B$ is nonempty.*
2. *Every vertex of $Q \cap B$ is in N_B and S_B.*
3. *N_B is 0-short and every proper subpath of S_B is ϵ-short.*
4. *There exists a number $t \leq \kappa(\epsilon)$ and vertices $s_0, s_1, \ldots, s_t$ ordered from west to east along S_B such that for any vertex x of $S_B[s_i, s_{i+1})$, the distance from x to s_i along S_B is less than ϵ times the distance from x to N_B in B.*

For the above lemma, a path is ϵ-*short* if the distance between every pair of vertices on that path is at most $(1+\epsilon)$ times the distance between them in G and $\kappa(\epsilon) = 4\epsilon^{-2}(1+\epsilon^{-1})$. The paths that forms eastern and western boundaries of bricks are called *supercolumns*, and further satisfy:

Lemma 2.49. (Lemma 6.6 [23]) *The sum of the weight of all edges in supercolumns is at most* ϵOPT.

To obtain the spanner, we add a set of Steiner trees in each brick B whose terminals are vertices of ∂B. The terminals are drawn from a subset of *portal* vertices evenly spaced on the boundary of each brick. We bound the number of portals per brick by $\theta(\epsilon) = O(\epsilon^{-2}\alpha(\epsilon))$ and $\alpha(\epsilon)$ in turn depends on the number of connections required to allow a nearly optimal solution, which is bounded by $o(\epsilon^{-5.5})$. The portals satisfy:

Lemma 2.50. (Lemma 7.1 [23]) *For any vertex x on ∂B, there is a portal y such that the weight of x-to-y subpath of ∂B is at most $1/\theta(\epsilon)$ times the weight of ∂B.*

Since the weight of each Steiner tree in a brick B can be bounded by the weight of the weight of ∂B, and since there are only constant number (that is $2^{\theta(\epsilon)}$) of such Steiner trees, the weight of all trees we add in B is at most $f(\epsilon)$ times the weight of ∂B. So the total weight of our spanner is bounded by $(2f(\epsilon) + 9\epsilon^{-1})$ times the weight of MG, which is $O(\text{OPT})$.

2.4.2 Proof of the Structure Theorem

We transform OPT for the instance (G, Q, r) so that it satisfies the following properties (repeated from the introduction):

P1: $\text{OPT} \cap \text{int}(B)$ can be partitioned into a set of trees $\mathcal{T}$ whose leaves are on the boundary of B.

P2: If we replace any tree in $\mathcal{T}$ with another tree spanning the same leaves, the result is a feasible solution.

P3: There is another set of $O(1)$ trees $\mathcal{T}'$ that costs at most a $1+\epsilon$ factor more than $\mathcal{T}$, such that each tree of $\mathcal{T}'$ has $O(1)$ leaves and $(\text{OPT} \setminus \mathcal{T}) \cup \mathcal{T}'$ is a feasible solution.

To argue about the leaves of trees on the boundary of bricks, we use the following definition:

Definition 2.51. (Joining vertex [22]). *Let H be a subgraph of G and P be a subpath of ∂G. A joining vertex of H with P is a vertex of P that is the endpoint of an edge of $H - P$.*

The transformation consists of the following steps:

Augment We add four copies of each supercolumn; we take two copies each to be interior to the two adjacent bricks. After this, connectivity between the east and west boundaries of a brick will be transformed to that between the north and south boundaries.

By Lemma 2.49, this only increases the weight by an small fraction of OPT.

Cleave By cleaving a vertex, we split it into multiple copies while keeping the connectivity as required by adding artificial edges of weight zero between two copies and maintaining a planar embedding. We call the resulting solution OPT_C. In this step, we turn k-edge-connectivity into k-vertex-connectivity for $k = 1, 2, 3$. By Theorem 2.20, we can obtain Property P1: $\mathrm{OPT}_C \cap \mathrm{int}(B)$ can be partitioned into a set $\mathcal{T}$ of trees whose leaves are in ∂B. By Corollary 2.4, we can obtain Property P2: we can obtain another feasible solution by replacing any tree in $\mathcal{T}$ with another tree spanning the same leaves.

Flatten For each brick B, we consider the connected components of $\mathrm{OPT}_C \cap \mathrm{int}(B)$. If the component only spans vertices in the north or south boundary, we replace it with the minimum subpath of the boundary that spans the same vertices. This will not increase the weight much by the ϵ-shortness of the north and south boundaries. Note that vertex-connectivity may bread as a result, but edge-connectivity is maintained. In the remainder, we only maintain edge-connectivity. We call the resulting solution OPT_F.

Restructure For each brick B, we consider the connected components of $\mathrm{OPT}_F \cap \mathrm{int}(B)$. We replace each component with a subgraph through a mapping ϕ. The new subgraph may be a tree or a subgraph $\widehat{C}$ given by Lemma 2.53. The mapping ϕ has the following properties:

1. For any component χ of $\mathrm{OPT}_F \cap \mathrm{int}(B)$, $\phi(\chi)$ is connected and spans $\chi \cap \partial B$.
2. For two components χ_1 and χ_2 of $\mathrm{OPT}_F \cap \mathrm{int}(B)$, if $\phi(\chi_i) \neq \widehat{C}$ for at least one of $i = 1, 2$, then $\phi(\chi_1)$ and $\phi(\chi_2)$ are edge-disjoint, taking into account edge multiplicities.
3. The new subgraph $\phi(\mathrm{OPT}_M \cap \mathrm{int}(B))$ has $\alpha(\epsilon) = o(\epsilon^{-5.5})$ joining vertices with ∂B.

We can prove that the total weight is increased by at most $\epsilon \mathrm{OPT}_F$, giving Property P3. We call the resulting solution OPT_R.

Redirect We connect each joining vertex j of $\mathrm{OPT}_R \cap \mathrm{int}(B)$ to the nearest portal p on ∂B by adding multiple copies of the short j-to-p subpath of ∂B. We call the resulting solution $\widehat{OPT}$.

We give more details of these transformations in the following subsections and argue that the transformations guarantee the required connectivity as we do so. These details are very similar to that used for 2-ECP; we note the differences. The Restructure step which requires a different structural lemma than used for Borradaile and Klein's PTAS for 2-ECP [22]; the difference between Lemma 2.53 and that used by Borradaile and Klein is that $\widehat{C}$ need only be a cycle for the 2-ECP, whereas to maintain 3-edge connectivity, a more complicated subgraph $\widehat{C}$ is required. We borrow the following lemma from [23] to prove Lemma 2.53.

Lemma 2.52. (Simplifying a tree with one root, Lemma 10.4 [23]). *Let r be a vertex of T. There is another tree $\widehat{T}$ that spans r and the vertices of $T \bigcap P$ such that $w(\widehat{T}) \leq (1 + 4\epsilon)w(T)$ and $\widehat{T}$ has at most $11\epsilon^{-1.45}$ joining vertices with P.*

Lemma 2.53. *Let $\mathcal{F}$ be a set of non-crossing trees whose leaves are joining vertices with ϵ-short paths P_1 and P_2 on the boundary of the graph, and each tree in $\mathcal{F}$ has leaves on both paths. Then there is a subgraph or empty set $\widehat{C}$, a set $\widehat{\mathcal{F}}$ of trees, and a mapping $\phi : \mathcal{F} \to \widehat{\mathcal{F}} \cup \widehat{C}$ with the following properties*

- *For every tree T in $\mathcal{F}$, $\phi(T)$ spans T's leaves.*
- *For two trees T_1 and T_2 in $\mathcal{F}$, if $\phi(T_i) \neq \widehat{C}$ for at least one of $i = 1, 2$ then $\phi(T_1)$ and $\phi(T_2)$ are edge-disjoint (taking into account edge multiplicities).*

- *The subgraph* $\bigcup \widehat{\mathcal{F}} \cup \{\widehat{C}\}$ *has* $o(\epsilon^{-2.5})$ *joining vertices with* $P_1 \cup P_2$.
- $w(\widehat{C}) + w(\widehat{\mathcal{F}}) \leq 6w(P_2) + (1 + d \cdot \epsilon)w(\mathcal{F})$, *where* d *is an absolute constant.*
- *Any two vertices of* $\widehat{C} \cap (P_1 \cup P_2)$ *are three-edge-connected.*
- *For any three pairs of vertices of* $\widehat{C} \cap (P_1 \cup P_2)$ *(where vertices may be repeated),* $\widehat{C}$ *contains three edge-disjoint paths connecting them respectively.*

Proof. Call P_1 the top path and P_2 the bottom path. Order the trees of $\mathcal{F}$: $T_1, T_2, \ldots, T_k$ according to their leaves on P_2 from left-to-right; the trees are well ordered since they are non-crossing and have leaves on both paths. There are two cases:

$k > \epsilon^{-1}$: In this case we reduce the number of trees by incorporating a subgraph $\widehat{C}$. Let a be the smallest index such that $w(T_a) \leq \epsilon w(\mathcal{F})$ and b the largest index such that $w(T_b) \leq \epsilon w(\mathcal{F})$. We replace trees $T_a, T_{a+1}, \ldots, T_b$ with a subgraph $\widehat{C}$. Let P_1' be the minimal subpath of P_1 that spans all the leaves of tree T_i for $a \leq i \leq b$. P_2' is likewise the minimal subpath on P_2. Let u_i and v_i be P_i''s endpoints (with u_i the left end) for $i = 1, 2$. Let P_a (P_b) be the u_1-to-u_2 (v_1-to-v_2) subpath in T_a (T_b). Since P_1 is ϵ-short, we have

$$w(P_1') \leq (1 + \epsilon)w(P_a \cup P_b \cup P_2').$$

Let $\widehat{C}$ be the multisubgraph $\{P_a, P_a, P_b, P_1', P_1', P_2', P_2'\}$. We replace $\bigcup_{i=a}^{b} T_i$ with $\widehat{C}$ and set $\phi(T_i) = \widehat{C}$ for $a \leq i \leq b$.

$k \leq \epsilon^{-1}$: In this case, the number of trees is already bounded and we set $\widehat{C} = \emptyset$.

In both cases, we transform the remaining trees T_i ($i \neq a, a+1, \ldots, b$) as follows. Let T_i' be a minimal subtree of T_i that spans all leaves of T_i on P_1 and exactly one vertex r on P_2. Let R_i be the minimal subpath on P_2 that spans all leaves of T_i on P_2. By Lemma 2.52, there is another tree, say T_i'', for T_i' with root r and path P_1. We replace T_i with $\widehat{T}_i = T_i'' \cup R_i$, and set $\phi(T_i) = \widehat{T}_i$ for $i \neq a, \ldots, b$. Then $\widehat{T}_i$ spans all leaves of T_i.

$\widehat{C}$ (if non-empty) has six joining vertices with $P_1 \cup P_2$. Each tree $\widehat{T}_i$ has one joining vertex with P_2 and by Lemma 2.52 $o(\epsilon^{-1.5})$ joining vertices with P_1. By the choice of a and b, there are at most ϵ^{-1} trees mapped to $\widehat{T}_i$. So $\widehat{\mathcal{F}} \cup \{\widehat{C}\}$ has totally at most $o(\epsilon^{-2.5})$ joining vertices with $P_1 \cup P_2$.

The total weight of $\widehat{C}$ is

$$\begin{aligned} w(\widehat{C}) &\leq 2w(P_1') + 2w(P_2') + 2w(P_a) + w(P_b) \\ &\leq 2(1+\epsilon)[w(P_a) + w(P_b) + w(P_2')] + 2w(P_2') + 2w(P_a) + w(P_b) \\ &\leq (4+2\epsilon)w(P_a) + (3+2\epsilon)w(P_b) + (4+2\epsilon)w(P_2') \\ &\leq (7+4\epsilon)\epsilon \cdot w(\mathcal{F}) + (4+2\epsilon)w(P_2'). \end{aligned}$$

And the total weight of $\widehat{T}_i$ is

$$\textstyle\sum_{i=1,\ldots,a-1,b+1,\ldots,k} w(\widehat{T}_i) \quad \leq \sum_{i=1,\ldots,a-1,b+1,\ldots,k}(w(R_i) + (1+4\epsilon)w(T_i)).$$

Since all the trees in $\mathcal{F}$ are non-crossing, R_i and P_2' are disjoint. The total weight of our replacement is at most $6w(P_2) + (1+O(\epsilon))w(\mathcal{F})$.

Now we prove the connectivity properties for $\widehat{C}$. By its construction, $\widehat{C}$ spans all the leaves of tree T_i for $a \leq i \leq b$ and vertices of $\widehat{C} \cap (P_1 \cup P_2)$ are three-edge-connected. So we only need to prove $\widehat{C}$ contains three edge-disjoint paths connecting any three pair of vertices of $\widehat{C} \cap (P_1 \cup P_2)$ respectively. This can be seen by case analysis, as described and illustrated below. Let $P^* = P_a \cup P_1' \cup P_2'$, then every edge of P^* has multiplicity of two in $\widehat{C}$ by construction.

We first consider the case that any two pairs do not contain identical vertex. For $i = 1, 2$ let x_i, y_i and z_i be the three pair of vertices. If there are two pairs of vertices are not interleaving in P^*, then we could connect these two pairs by $P^* \cup P_b$, which is a cycle. And the other pair could be connected by a subpath of P^* which is edge-disjoint from the other two paths in $\widehat{C}$. Otherwise, we have the two sets, $\{x_i, y_j, z_l\}$ and $\{x_{3-i}, y_{3-j}, z_{3-l}\}$, which do not interleave each other and appear in the same order in P^*. See Figure 2.14: there are three edge-disjoint paths connecting the three pairs respectively.

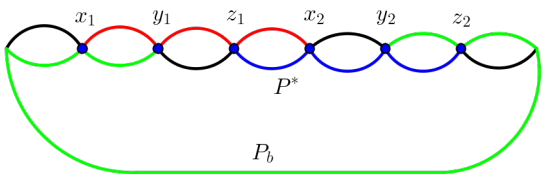

Figure 2.14: If every two pairs of the three interleave, there are three edge-disjoint paths, shown in different colors, connecting them respectively in $\widehat{C}$.

If there are two pairs containing identical vertices, we could connect these two pairs by $P^* \cup P_b$ and then connect the other pair by another subpath of P^*. Since every edge in P^* is multiple in $\widehat{C}$, these paths are edge-disjoint. □

2.4.2.1 Augment

For each supercolumn P common to two bricks B_1 and B_2, we do the following:

- Add four copies of P, P_1, P_2, P_3, P_4 to OPT. We consider P_1 and P_2 to be internal to B_1, and P_3 and P_4 to be internal to B_2. We also cut open the graph along P, creating a new face between P_2 and P_3. Call the result OPT'_A.
- Remove edges from OPT'_A until what remains a minimal subgraph satisfying the connectivity requirements. Similar to the argument illustrated in Figure 2.14, maintaining connectivity in the presence of this new face is achievable.

Let the resulting solution be OPT_A. By this construction, OPT_A has no joining vertices with internal vertices of the supercolumns:

Lemma 2.54. *For every brick B, the joining vertices of $OPT_A \cap B$ with ∂B belong to N_B and S_B.*

2.4.2.2 Cleave

In this part, we call the vertices with positive requirement terminals. Given a vertex v with a non-interleaving bipartition A and B of the edges incident to v, we define *cleaving* v as the following: split v into two copies v_1 and v_2, such that all the edges in A (B) are incident to v_1 (v_2); and then we add one zero-weight edge between v_1 and v_2. See Figure 2.15. We have two types of cleavings:

Simplifying cleavings Let C be a non-self-crossing, non-simple cycle that visits vertex v twice. For $1 \leq i \leq 4$, let e_i be the edge of C incident to v such that e_1, e_2, e_3 and e_4 are embedded clockwise and e_1 and e_4 are in the same subcycle of C. Define a bipartition A, B of the edges incident to v as follows: given the clockwise embedding of those edges, let A contain e_1, e_2 and all the edges between them clockwise.

Lengthening cleavings Let C be a cycle and v a vertex on C. Let e_1 and e_2 be the edges incident to v strictly inside of C, and let e'_1 and e'_2 be the edges of C incident to v.

Define a bipartition A and B of the edges incident to v as follows: given the embedding of those edges with e_1, e_2, e'_2 and e'_1 in the clockwise order, A contains e_1, e'_1 and all the edges from e'_1 to e_1 clockwise.

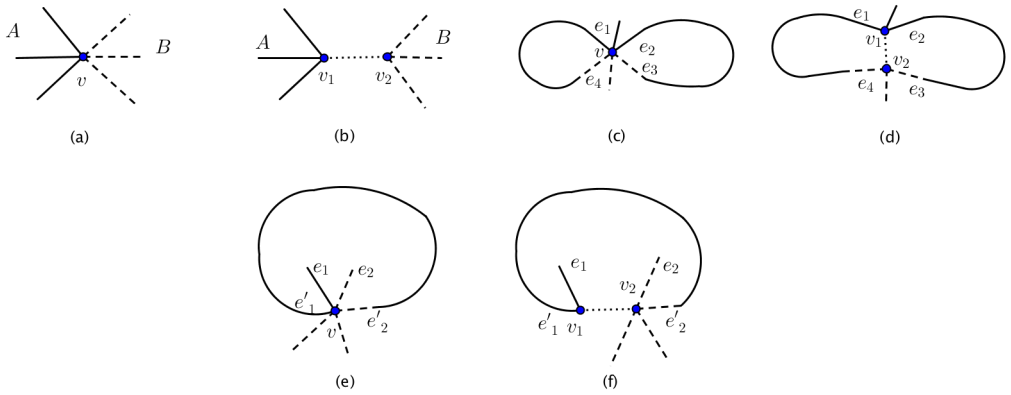

Figure 2.15: Cleaving examples. The bipartition of edges incident to v is illustrated by solid edges A and dashed edges B, and the added artificial edge is illustrated by the dotted edge. (a) and (b) give an example before and after a cleaving. (c) and (d) illustrate a simplifying cleaving. (e) and (f) show a lengthening cleaving.

We perform simplifying cleavings for all the non-simple cycles of OPT_A until every cycle is simple and call the resulting solution OPT'_S and the resulting graph G_S. Note that after simplifying cleaving a terminal, we take only one copy that is in the mortar graph as the terminal. These cleavings will reduce edge-connectivity to vertex-connectivity by the following lemmas.

Lemma 2.55. *For the cleaved vertex v, the copies of v are three-edge-connected in OPT'_S.*

Proof. The original cycle will give two edge-disjoint paths between the two copies. And the artificial edge will be the third path. □

Lemma 2.56. *Let $\widehat{H}$ be the graph obtained from H by a simplifying cleaving a vertex v. Then for $k = 1, 2, 3$, if a vertex u and v are k-edge-connected in H, then u and any copy of v are k-edge-connected in $\widehat{H}$.*

Proof. If u and v are k-edge-connected, then there are k edge-disjoint paths from u to $\{v_1, v_2\}$. Since k is no more than three, and v_1 and v_2 are three-edge-connected by

Lemma 2.55, u and v_1 are k-edge-connected. Similarly, u and v_2 are k-edge-connected. □

Corollary 2.57. *For $k = 1, 2, 3$, if two vertices are k-edge-connected in OPT_A, then any of their copies are k-edge-connected in OPT'_S.*

We then remove edges from OPT'_S until the rest is a minimal subgraph satisfying the connectivity requirement. We call the resulting solution OPT_S. Since every cycle in OPT_S is simple, we have the following lemma:

Lemma 2.58. *For $k = 1, 2, 3$, if two terminals are k-edge-connected in OPT_A, then they are k-vertex-connected OPT_S.*

Let Q_{23} be set of terminals whose requirement is at least two and H be the minimal (Q_{23}, r)-vertex-connected subgraph in OPT_S. Since every cycle in OPT_S is simple, we know that H also has this property. It follows that the degree of any vertex v of H is no more than three, since H is biconnected by Lemma 2.6: for otherwise, let the first ear (a cycle) contain v and let the second and third ear start with the next two edges incident to v – from these ears it is easy to construct two cycles that only meet at v, witnessing a non-simple cycle.

We perform lengthening cleavings w.r.t. the boundary of each brick and the edges of OPT_S that are incident to a boundary vertex in that brick until the vertices of the brick boundaries have at most one edge in the solution in the interior of each incident brick. We add the introduced zero-length edges to the solution. Call the new solution OPT_C and the new graph and mortar graph G_C and MG_C, respectively.

For each cleaved terminal, we assign only one copy to be the terminal: we pick the copy of highest degree in OPT_C to be the terminal; note that terminals are still in the mortar graph. Although the flatten, restructure and redirect steps may break the vertex connectivity guaranteed by Lemma 2.55, the following lemmas will guarantee that edge-connectivity will not be preserved.

Lemma 2.59. *For $k = 1, 2, 3$, if terminals are k-vertex-connected in OPT_S, then they are k-vertex-connected OPT_C. Further, OPT_C is minimal.*

Proof. Borradaile and Klein prove (Lemma 5.9 [22]) that lengthening cleavings maintain biconnectivity, so we need only argue that terminals a and b that are triconnected in OPT_S have terminal copies a' and b' that are triconnected in OPT_C.

Consider three, vertex-disjoint a-to-b paths P_1, P_2, P_3 in OPT_S. As argued earlier, the degrees of a and b in OPT_S are three. The terminal copy a' of a likewise has degree 3 in OPT_C: if a is lengthening cleaved, then there is a copy of a that has degree 3.

The paths P_1, P_2, P_3 map to paths P_1', P_2', P_3' between copies of a and b in OPT_C: if an internal vertex of P_i is lengthening cleaved, then include the introduced edge in the path. The endpoints of P_i' that map to copies of, w.l.o.g., a are likewise connected by introduced edges (if a is subject to a lengthening cleaving). Let a' be the terminal copy of a; we augment the paths P_i' to connect to a' via an introduced, zero-weight edge. Doing so for b' as well gives three vertex disjoint a'-to-b' paths in OPT_C.

That OPT_C is minimal follows directly from the fact that OPT_S is minimal. □

The following lemma was proved by Borradaile and Klein for 2-ECP (Lemma 5.10 [22]); their proof only relied on the fact that every cycle contains a terminal. Since this is true for the 3-ECP too (Theorem 2.20), we get the same lemma:

Lemma 2.60. *Let B be a brick in G_C with respect to MG_C. The intersection $OPT_C \cap \text{int}(B)$ is a forest whose joining vertices with ∂B are all the leaves of the forest.*

Lemma 2.61. *Let x and y be two terminals in OPT_C such that $k = \min\{r(x), r(y)\} \geq 2$ and let B be a brick. There exist k vertex-disjoint paths from x to y such that for any two such paths P_1 and P_2, any connected component of $P_1 \cap \text{int}(B)$ and any connected component of $P_2 \cap \text{int}(B)$ belong to distinct components of $OPT_C \cap \text{int}(B)$.*

Proof. Let H_C be the minimal (Q_{23}, r)-vertex-connected subgraph in OPT_C. Then each connected component of $\text{OPT}_C \setminus H_C$ is a tree and is connected to H_C by one edge. By Corollary 2.4, there are $min\{r(x), r(y)\}$ vertex-disjoint paths from x to y in H_C (and then in OPT_C) such that any path connecting any two of those x-to-y paths contains a terminal. Let P_1 and P_2 be any two such x-to-y paths. Since all terminals are on the boundaries of bricks, any P_1-to-P_2 path in $\text{int}(B)$ will be divided into two subpaths by some lengthening cleaving. So any connected component of $\text{OPT}_C \cap \text{int}(B)$ can not contain the components of both $P_1 \cap \text{int}(B)$ and $P_2 \cap \text{int}(B)$. □

2.4.2.3 Flatten

This step is the same as described by Borradaile and Klein for 2-ECP [22]. For each brick B, consider the edges of $\text{OPT}_C \cap \text{int}(B)$. By Lemma 2.60, the connected components of

$\text{OPT}_C \cap \text{int}(B)$ are trees. By Lemma 2.54, every leaf is either on N_B or S_B. For every tree whose leaves are all on N_B (S_B), we replace the tree with the minimal subpath of N_B (S_B) that contains all its leaves. Let the resulting solution be OPT_F. By Lemma 2.61, this guarantees 2- and 3-edge-connectivity between terminals as required; trees may be flattened against a common ϵ-short path that is the northern boundary of one brick and the southern boundary of another, so vertex-connectivity gets broken at this stage.

2.4.2.4 Restructure

This step is the same as described by Borradaile and Klein for 2-ECP [22], except we apply our 3-ECP specific lemma (Lemma 2.53). Restructuring replaces $\text{OPT}_F \cap \text{int}(B)$, which is a set of non-crossing trees (Lemma 2.60) with leaves on the ϵ-short north and south brick boundaries (Lemma 2.54), with subgraphs guaranteed by Lemma 2.53. The resulting solution is OPT_R.

Let $\mathcal{P}_{xy}$ be a set of 2 (3) vertex disjoint paths in OPT_F for terminals x, y requiring bi- (tri-)connectivity. Each path in $\mathcal{P}_{xy}$ is broken into a sequence of small paths, each of which is either entirely in the interior of a brick or entirely in the mortar graph. Let $\mathcal{P}'_{xy}$ be the set of these path sequences.

We define a map $\hat{\phi}$ for the paths in $\mathcal{P}'_{xy}$. For a path P of $\mathcal{P}'_{xy}$, if P is on mortar graph, we define $\hat{\phi}(P) = P$; otherwise we define $\hat{\phi}(P) = \phi(T)$ where T is the tree in OPT_M containing P. Since $\phi(T)$ spans all leaves of T, $\hat{\phi}(P)$ also spans leaves of T and connects endpoints of P.

Let P_1 and P_2 be any two paths from distinct path-sequences inside of the same brick B. By Lemma 2.61, P_1 and P_2 can not belong to the same component of $\text{OPT}_M \cap \text{int}(B)$. So if $\hat{\phi}(P_1) \neq \hat{\phi}(P_2)$, then $\hat{\phi}(P_1)$ and $\hat{\phi}(P_2)$ are edge disjoint by the constructions of ϕ and $\hat{\phi}$. Otherwise, we know the image is a subgraph $\widehat{C}$ which guarantees triconnectivity for all vertices of $\widehat{C} \cap \partial B$ by Lemma 2.53. However, there may be more than one paths from any path-sequence whose image is $\widehat{C}$, and the new x-to-y paths may not be edge-disjoint in $\widehat{C}$. For this situation, we could shortcut the paths in $\widehat{C}$ such that each new x-to-y path only contain one subpath in $\widehat{C}$. Since there are at most three such subpaths and there endpoints are in ∂B, $\widehat{C}$ contains edge-disjoint paths connecting the endpoints of those subpaths by the last property of Lemma 2.53.

Therefore, the restructure step maintains that if terminals were 1-, 2- or 3-edge

connected in OPT_F, then they still are in OPT_R. We also see that, by construction of Lemma 2.53, the intersection OPT_R with the interior of a brick B is a set of trees with leaves on ∂B. Since the Redirect step will only add edges of ∂B, this property does not change, proving one of the guarantees of the Structure Theorem.

Further, the number of joining vertices is guaranteed by Lemma 2.53 and the construction that is used for 2-ECP. The number of joining vertices is on the same order as for 2-ECP, which depends only on ϵ as required.

2.4.2.5 Redirect

For every joining vertex j of $\mathrm{OPT}_R \cap B$ with ∂B for a brick B, we add the path from j to the nearest portal p on ∂B. This guarantees that the trees guaranteed by the Restructure step have leaves that are portals: this allows us to efficiently enumerate all possible Steiner trees in bricks whose terminals are portals to compute the spanner graph.

2.4.2.6 Analysis of weight increase

The analysis of the weight increase is exactly the same as for 2-ECP by Borradaile and Klein; the only difference are the weight in Lemma 2.53 which is on the order of the weight in the equivalent Lemma used in 2-ECP.

This completes the proof of the Structure Theorem.

2.5 Dynamic Programming for k-ECP on Graphs with Bounded Branchwidth

In this section, we give a dynamic program to compute the optimal solution of k-ECP problem on graphs with bounded branchwidth. This is inspired by the work of Czumaj and Lingas [38, 39]. Note that such graphs need not be planar. This can be used in our PTAS after the contraction step of the framework.

A *branch decomposition* of a graph $G = (V(G), E(G))$ is a hierarchical clustering of $E(G)$. It can be represented by a binary tree, called the *decomposition tree*, the leaves of which are in bijection with the edges of G. After deleting an edge e of this decomposition

tree, $E(G)$ is partitioned into two parts E_1 and E_2 according to the edges mapped to the leaves of the two subtrees. All the vertices common to E_1 and E_2 comprise the *separator* corresponding to e in the decomposition. The *width* of the decomposition is the maximum size of the separator in that decomposition. The *branchwidth* of G is the minimum width of any branch decomposition of G.

Let $G = (V(G), E(G), r)$ be an instance of k-ECP. Then $r \in \{0, 1, \cdots, k\}$. We call a vertex a terminal if its requirement is positive. We first augment G such that each edge becomes k parallel edges. Our dynamic programming will work on this new graph G. Given a branch decomposition of G, root the decomposition tree T at an arbitrary leaf. For any node q in T, let L be the separator corresponding to its parent edge, and E_1 be the subset of $E(G)$ mapped to the leaves in the subtree rooted at q. Let H be a subgraph of $G[E_1]$ such that it contains all terminals in $G[E_1]$. An *separator completion* of L is a multiset of edges between vertices of L, each of which may appear up to k times. A *configuration* of a terminal v of H in L is a tuple $(A, B, r(v))$, where A is a tuple $(a_1, a_2, \ldots, a_{|L|})$, representing that there are a_i edge-disjoint paths from v to the ith vertex of L in H, and B is a set of tuples (x_i, y_i, b_i), representing that there are b_i edge-disjoint paths between the vertices x_i and y_i of L in H. All the $\sum_{i=1}^{|L|} a_i + \sum_i b_i$ paths in a configuration are mutually edge-disjoint in H. We adapt a definition of Czumaj and Lingas [38, 39]:

Definition 2.62. *For any pair of terminals u and v in H, let $Com_H(u, v)$ be the set of separator completions of H each of which augments H to a graph where u and v satisfy the edge-connectivity requirement. For each terminal v in H, let $Path_H(v)$ be a set of configurations of v on L. Let $Path_H$ be the set of all the non-empty B in which all tuples can be satisfied in H. Let C_H be the set consisting of one value in each $Com_H(u, v)$ for all pairs of terminals u and v in H, and P_H be the set consisting of one value in each $Path_H(v)$ for all terminal v in H. We call the tuple $(C_H, P_H, Path_H)$ the* connectivity characteristic *of H, and denote it by $Char(H)$.*

Let w be the width of the decomposition. Then $|L| \leq w$. Note that H may correspond to multiple C_H and P_H, so H may have multiple connectivity characteristics. Further, each value in P_H represents at least one terminal. For any L, there are at most $k^{O(w^2)}$ distinct separator completions ($O(w^2)$ pairs of vertices, each of which can be connected by at most k parallel edges) and at most $2^{k^{O(w^2)}}$ distinct sets C_H of separator completions.

For any L, there are at most $k^{O(w^2)}$ different configurations for any terminal in H since the number of different sets A is at most k^w, the number of different sets B is at most $k^{O(w^2)}$ (the same as the number of separator completions) and k different choices for $r(v)$. So there are at most $2^{k^{O(w^2)}}$ different sets of configurations P_H, and at most $2^{k^{O(w^2)}}$ different sets B. Therefore, there are at most $2^{k^{O(w^2)}}$ distinct connectivity characteristics for a fixed L.

Definition 2.63. *A configuration of v on L is* connecting *if for any terminal u in $V(G)\backslash V(H)$ the inequality $\sum_{i=1}^{|L|} a_i \geq \min\{r(v), r(u)\}$ holds where a_i is the ith coordinate in A. That is, there are enough edge-disjoint paths from v to the separator which can connect u and v. $Char(H)$ is connecting if all configurations in its P_H set are connecting. H is connecting if at least one of $Char(H)$ is connecting. In the following, we only consider connecting connectivity characteristics and subgraphs.*

In the following, we need as a subroutine an algorithm to solve the following problem: when given a set of demands (x_i, y_i, b_i) and a multigraph, we want to decide if there exist b_i edge-disjoint paths between vertices x_i and y_i in the graph and all the $\sum_i b_i$ paths are mutually edge-disjoint. Although we do not have a polynomial time algorithm for this problem, we only need to solve this on graphs with $O(w)$ vertices, $O(kw^2)$ edges and $O(w^2)$ demands. So even an exponential time algorithm is acceptable for our purpose here. Let ALG be an algorithm for this problem, whose running time is bounded by a function $f(k, w)$, which may be exponential in both k and w.

For a node p of degree three in the decomposition tree T, let q_1 and q_2 be its two children and q be its parent. Let T_i be the subtree of T rooted at q_i, let E_i be the subset of $E(G)$ corresponding to T_i and let L_i be the separator corresponding to the edge pq_i for $i = 1, 2$. Let L be the separator corresponding to pq. For $i = 1, 2$, let H_i be a subgraph of $G[E_i]$ that contains all the terminals. Let $H = H_1 \cup H_2$. Then we have the following lemma.

Lemma 2.64. *For any pair of $Char(H_1)$ and $Char(H_2)$, all the possible $Char(H)$ that could be obtained from $Char(H_1)$ and $Char(H_2)$ can be computed in $O(k^{w^2} f(k, w) + k^{w^2 k^{w^2}})$ time.*

Proof. We compute all the possible sets for the three components of $Char(H)$.

Compute all possible $\mathbf{C_H}$ C_H contains two parts: the first part covers all pairs of terminals in the same H_i for $i = 1, 2$ and the second part covers all pairs of terminals from distinct subgraphs.

For the first part, we generalize each value $C \in C_{H_i}$ for $i = 1, 2$ into a possible set X_C. Notice that each separator completion can be represented by a set of demands (x, y, b). For a candidate separator completion C' on L, we combine C' with each $B \in Path_{H_{3-i}}$ to construct a graph H' and define the demand set the same as C. By running ALG on this instance, we can check if C' is a legal generalization for C. This may be computed in $k^{O(w^2)}w^2 + k^{O(w^2)}f(k, w)$ time for each C. All the legal generalizations for C form X_C.

Now we compute the second part. For any pair of configurations $(A^1, B^1, r(u)) \in P_{H_1}$ and $(A^2, B^2, r(v)) \in P_{H_2}$ for $u \in H_1$ and $v \in H_2$, we compute possible $Com_H(u, v)$. Let $L' = L_1 \cap L_2$. We first count how many edge-disjoint paths between u and v could go through L' by checking A^1 and A^2, and then check if a candidate separator completion C' on L can provide the remaining paths. All those C' that are capable of providing enough paths form $Com_H(u, v)$. This can be computed in $w^2k^{O(w^2)}$ time for each pair of values.

A possible C_H consists of each value in X_C for every $C \in C_{H_i}$ for $i = 1, 2$ and each value in $Com_H(u, v)$ for all pairs of configurations of P_{H_1} and P_{H_2}. To compute all the sets, we need at most $k^{O(w^2)}w^2 + k^{O(w^2)}f(k, w)$ time. There are at most $k^{O(w^2)}$ sets and each may contain at most $k^{O(w^2)}$ values. Therefore, to generate all the possible C_H from those sets, we need at most $k^{w^2k^{O(w^2)}}$ time.

Compute all possible $\mathbf{P_H}$ We generalize each configuration $(A, B, r(v))$ of v in P_{H_i} into a possible set Y_v. For each set B' in $Path_{H_{3-i}}$, we construct a graph H' by A, B and B' on vertex set $L_1 \cup L_2 \cup \{v\}$: if there are b disjoint paths between a pair of vertices represented in A, B or B', we add b parallel edges between the same pair of vertices in H', taking $O(w^2)$ time. For a candidate value $(A^*, B^*, r(v))$ corresponding to L, we define a set of demands according to A^* and B^* and run ALG on all the possible H' we construct for sets in $Path_{H_{3-i}}$. If there exists one such graph that satisfies all the demands, then we add this candidate value into Y_v. We can therefore compute each set Y_v in $k^{O(w^2)}w^2 + k^{O(w^2)}f(k, w)$ time. A possible P_H consists of each value in Y_v. There are at most $k^{O(w^2)}$ such sets and each may contain at most $k^{O(w^2)}$ values. So we can generate all possible P_H from those sets in $k^{w^2k^{O(w^2)}}$ time.

Compute Path$_H$ For each pair of $B^1 \in Path_{H_1}$ and $B^2 \in Path_{H_2}$, we construct a graph H' on vertex set $L_1 \cup L_2$: if two vertices are connected by b disjoint paths, we add b parallel edges between those vertices in H'. Since each candidate B' on L can be represented by a set of demands, we only need to run ALG on all possible H' to check if B' can be satisfied. We add all satisfied candidates B' into $Path_H$. This can be computed in $k^{O(w^2)}w^2 + k^{O(w^2)}f(k,w)$ time.

Therefore, the total running time is $O(k^{w^2}f(k,w) + k^{w^2k^{w^2}})$. For each component we enumerate all possible cases, and the correctness follows. □

Our dynamic programming is guided by the decomposition tree T from leaves to root. For any node q in T, let T_q be the subtree of T rooted at q and L_q be the separator corresponding q's parent edge. Let E_q be the subset of $E(G)$ corresponding to T_q. For each node q, our dynamic programming table is indexed by all the possible connectivity characteristics on the corresponding separator L_q. Each entry indexed by the connectivity characteristic $Char$ in the table is the weight of the minimum-weight subgraph of $G[E_q]$ that contains all the terminals in $G[E_q]$ and has $Char$ as its connectivity characteristic.

Base case For each leaf of T, the only subgraph H is the edge uv contained in the leaf and the separator only contains its endpoints u and v. There are three cases.

1. Both u and v are not terminals. $Com_H(u,v)$ contains all subsets of the multiset of edge uv (up to k times), including the empty set. P_H is empty since there is no terminal. $Path_H$ contains one set: $\{(u,v,1)\}$.

2. Only one of u and v is a terminal. W.l.o.g. assume u is the terminal. $Com_H(u,v)$ contains all subsets of the multiset of edge uv (up to k times), including the empty set. $Path_H(u)$ contains two configurations: $((k,0),\{(u,v,1)\},r(u))$ and $((k,1),\emptyset,r(u))$. $Path_H$ contains one set: $\{(u,v,1)\}$.

3. Both u and v are terminals. $Com_H(u,v)$ contains the multisets of edge uv that appears at least $\min\{r(u),r(v)\}-1$ times. $Path_H(u)$ contains two configurations: $((k,0),\{(u,v,1)\},r(u))$ and $((k,1),\emptyset,r(u))$, and $Path_H(v)$ contains two configurations: $((0,k),\{(u,v,1)\},r(v))$ and $((1,k),\emptyset,r(v))$. $Path_H$ contains one set: $\{(u,v,1)\}$.

For each non-leaf node q in T, we combine every pair of connectivity characteristics from its two children to fill in the dynamic programming table for q. The root can be seen as a base case, and we can combine it with the computed results. The final result will be the entry indexed by $(\emptyset, \emptyset, \emptyset)$ in the table of the root. If $E(G) = km$, then the size of the decomposition tree T is $O(km)$. By Lemma 2.64, we need $O(k^{w^2} f(k,w) + k^{w^2 k^{w^2}})$ time to combine each pair of connectivity characteristics. Since there are at most $2^{k^{O(w^2)}}$ connectivity characteristics for each node, the total time will be $O(2^{k^{w^2}} k^{w^2} f(k,w)m + 2^{k^{w^2}} k^{w^2 k^{w^2}} m)$.

Correctness The separator completions guarantee the connectivity for the terminals in H, and the connecting configurations enumerate all the possible ways to connect terminals in H and terminals of $V(G) \backslash V(H)$. So the connectivity requirement is satisfied . The correctness of the procedure follows from Lemma 2.64.

Chapter 3: PTASes for Minimum Three-edge-connected Spanning Subgraph and Minimum Three-vertex-connected Spanning Subgraph in Planar Graphs

Given an undirected unweighted graph G, the *minimum k-edge connected spanning subgraph problem* (k-ECSS) asks for a spanning subgraph of G that is k-edge connected (remains connected after removing any $k-1$ edges) and has a minimum number of edges. The *minimum k-vertex connected spanning subgraph problem* (k-VCSS) asks for a k-vertex connected (remains connected after removing any $k-1$ vertices) spanning subgraph of G with minimum number of edges. These are fundamental problems in network design and have been well studied. When $k = 1$, the solution is simply a spanning tree for both problems. For $k \geq 2$, the two problems both become NP-hard [62, 31], so people put much effort into achieving polynomial-time approximation algorithms. Cheriyan and Thurimella [31] give algorithms with approximation ratios of $1 + 1/k$ for k-VCSS and $1 + 2/(k+1)$ for k-ECSS for simple graphs. Gabow and Gallagher [61] improve the approximation ratio for k-ECSS to $1 + 1/(2k) + O(1/k^2)$ for simple graphs when $k \geq 7$, and they give a $(1 + 21/(11k))$-approximation algorithm for k-ECSS in multigraphs. Some researchers have studied these two problems for the small connectivities k, especially $k = 2$ and $k = 3$, and obtained better approximations. The best approximation ratio for 2-ECSS in general graphs is 4/3 of Sebő and Vygen [112], while for 2-VCSS in general graphs, the best ratio is 9/7 of Gubbala and Raghavachari [67]. Gubbala and Raghavachari [68] also give a 4/3-approximation algorithm for 3-ECSS in general graphs. Neither k-ECSS nor k-VCSS have a PTAS even in graphs of bounded degree for $k = 2$ unless P = NP [39]. Czumaj et al. [37] show that there are PTASes for both of 2-ECSS and 2-VCSS in planar graphs. Both problems are NP-hard in planar graphs (by a reduction from Hamiltonian cycle). Later, Berger and Grigni improved the PTAS for 2-ECSS to run in linear time [12].

Following their PTASes for 2-ECSS and 2-VCSS, Czumaj et al. [37] ask the following: *can we extend the PTAS for 2-ECSS to a PTAS for 3-ECSS in planar graphs?* A PTAS

for 3-VCSS in planar graphs is additionally listed as an open problem in the *Handbook of Approximation Algorithms and Metaheuristics* (Section 51.8.1) [65]. In this chapter, we answer these questions affirmatively by giving the first PTASes for both 3-ECSS and 3-VCSS in planar graphs. Our main results are the following theorems.

Theorem 3.1. *For 3-ECSS, there is an algorithm that, for any $\epsilon > 0$ and any undirected planar graph G, finds a $(1+\epsilon)$-approximate solution in linear time.*

Theorem 3.2. *For 3-VCSS, there is an algorithm that, for any $\epsilon > 0$ and any undirected planar graph G, finds a $(1+\epsilon)$-approximate solution in linear time.*

In the following, we assume there are no self-loops in the input graph for both of 3-ECSS and 3-VCSS. For 3-ECSS, we allow parallel edges in G, but at most 3 parallel edges between any pair of vertices are useful in a minimal solution. For 3-VCSS, parallel edges are unnecessary, so we assume the input graph is simple. Since three-vertex connectivity (triconnectivity) and three-edge connectivity can be verified in linear time [111, 101], we assume the input graph G contains a feasible solution. W.l.o.g. we also assume $\epsilon < 1$.

3.1 Overview

Our PTASes follow the spanner framework. For most applications of this framework, the challenging step is to illustrate the existence of a spanner subgraph. However, for 3-ECSS and 3-VCSS, we could simply obtain a spanner from the input graph G by deleting additional parallel edges since by planarity there are at most $O(n)$ edges in G and the size of an optimal solution is at least n, where $n = |V(G)|$. So, different from those previous applications, the real challenge for our problems is to illustrate the slicing step and the combining step. For the slicing step, we want to identify a set of slices that have two properties: (1) three-edge-connectivity for 3-ECSS or triconnectivity for 3-VCSS, and (2) bounded branchwidth. With these two properties, we can solve 3-ECSS or 3-VCSS on each slice efficiently. For the combining step, we need to show that we can obtain a nearly optimal solution from the optimal solutions of all slices found in slicing step and the shared edges of slices, that means the solution should satisfy the connectivity requirement and its size is at most $1+\epsilon$ times of the size of an optimal solution for the original input graph.

To identify slices, we generalize a decomposition used by Baker [6]. Before sketching our method, we briefly mention the difficulty in applying previous techniques. The PTAS for TSP [88] identifies slices in a spanner G' by doing a breadth-first search in its planar dual to decompose the edge set into some levels, and any two adjacent slices could only share all edges of the same level, which form a set of edge-disjoint simple cycles. This is enough to achieve simple connectivity or biconnectivity between vertices of different slices. But our problems need stronger connectivity for which one cycle is not enough. For example, we may need a non-trivial subgraph outside of slice H to maintain the triconnectivity between two vertices in slice H. See Figure 3.1 (a).

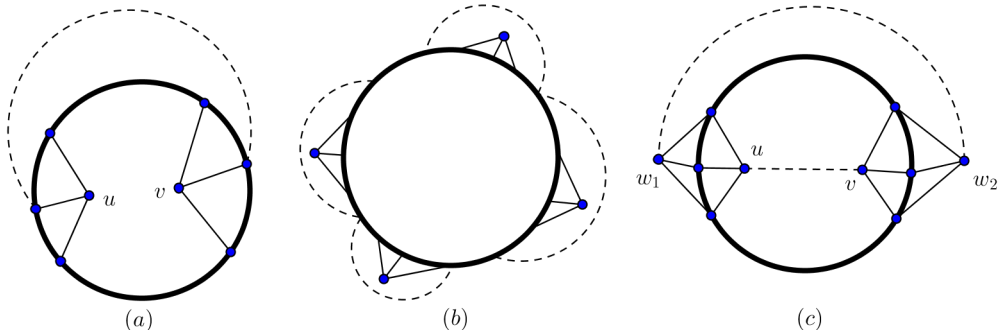

Figure 3.1: (a) The bold cycle encloses a slice H. To maintain the triconnectivity between two vertices u and v in H, we need the dashed path outside of H. (b) The bold cycle encloses a 3EC slice H. The dashed edges divide the outer face of H into distinct regions, which may contain contracted nodes. (c) The bold cycle encloses a 3EC slice H. The dashed path P between w_1 and w_2 will be contracted to obtain a node x. A solution for 3-ECSS on H may contain x but not the dashed edge between u and v. Then the union S of feasible solutions on all 3EC slices is not feasible if it does not contain path P.

For 3-ECSS, we construct a graph called *3EC slice*. We contract each component of the spanner that is not in the current 3EC slice. Since contraction can only increase edge-connectivity, this will give us the 3EC slices that are three-edge connected (Lemma 3.18 in Section 3.3). However, if we directly apply this contraction method in the slicing step of the PTAS for TSP, the branchwidth of the 3EC slice may not be bounded. This is because there may be many edges that are not in the slice but have both endpoints in the slice, and such edges may divide the faces of the slice into distinct regions, each of which may contain a contracted node. See Figure 3.1 (b). To avoid this problem, we apply the contractions in the decomposition used by Baker [6], which define a slice based

on the levels of vertices instead of edges. We can prove that each 3EC slice has bounded branchwidth in this decomposition (Lemma 3.16 in Section 3.2).

Although each 3EC slice is three-edge connected, the union S of their feasible solutions may not be three-edge connected. Consider the following situation. In a solution for a slice, a contracted node x is contained in all vertex-cuts for a pair of vertices u and v. But in S, the subgraph induced by the vertex set X corresponding to x may not be connected. Therefore, u and v may not satisfy the connectivity requirement in S. See Figure 3.1 (c).

In their paper, Czumaj et al. [37] proposed a structure called *bicycle*. A bicycle consists of two nested cycles, and all in-between faces visible from one of the two cycles. This can be used to maintain the three-edge-connectivity between those connecting endpoints in cycles. This motivates our idea: we want to combine this structure with Baker's shifting technique so that two adjacent slices shared a subgraph similar to a bicycle. In this way, we could maintain the strong connectivity between adjacent slices by including all edges in this shared subgraph, whose size could be bounded by the shifting technique. Specifically, we define a structure called *double layer* for each level i based on our decomposition, which intuitively contains all the edges incident to vertices of level i and all edges between vertices of level $i+1$. Then we define a 3EC slice based on a maximal circuit such that any pair of 3EC slices can only share edges in the double layer between them. In this way, we can obtain a feasible solution for 3-ECSS by combining the optimal solutions for all the slices and all the shared double layers (Lemma 3.21 in Section 3.3). By applying shifting technique on double layers, we can prove that the total size of the shared double layers is a small fraction of the size of an optimal solution. So we could add those shared double layers into our solution without increasing its size by much, and this gives us a nearly optimal solution.

For 3-VCSS, we construct a *3VC slice* based on a simple cycle instead of a circuit. The construction is similar to that of 3EC slices. However, contraction does not maintain vertex connectivity. So we need to prove each 3VC slice is triconnected (Lemma 3.25 in Section 3.4). Then similar to 3-ECSS, we also need to prove that the union of the optimal solutions of all 3VC slices and all shared double layers form a feasible solution (Lemma 3.27 in Section 3.4).

For dynamic-programming step, we need to solve a minimum-weight 3-ECSS problem in each 3EC slice and a minimum-weight 3-VCSS problem in each 3VC slice. This is

because we need to carefully assign weights to edges in a slice so that we can bound the size of our solution. We provide a dynamic program for the minimum-weight 3-ECSS problem in graphs of bounded branchwidth in Section 3.5, which is similar to that in the works of Czumaj and Lingas [38, 39]. A dynamic program for minimum-weight 3-VCSS can be obtained in a similar way. Then we have the following theorem.

Theorem 3.3. *Minimum-weight 3-ECSS problem and minimum-weight 3-VCSS problem both can be solved on a graph G of bounded branchwidth in $O(|E(G)|)$ time.*

Remark 3.4. *We point out that, for edge-weighted planar graphs, our PTAS fails at the Spanner step. That is, we cannot find a spanner for 3-ECSS or 3-VCSS for weighted planar graphs. If we can find a spanner for edge-weighted planar graphs for 3-ECSS or 3-VCSS, then our algorithm can give a PTAS for the corresponding problem. And it will be interesting to know if there is a PTAS for 3-ECSS and 3-VCSS in edge-weighted planar graphs.*

3.2 Preliminaries

Let G be an undirected planar graph with vertex set $V(G)$ and edge set $E(G)$. We denote by $G[S]$ the subgraph of G induced by S where S is a vertex subset or an edge subset. We simplify $|E(G)|$ to $|G|$. We assume we are given an embedding of G in the plane. We denote by $\partial(G)$ the subgraph induced by the edges on the outer boundary of G in this embedding. A *circuit* is a closed walk that may contain repeated vertices but not repeated edges. A *simple cycle* is a circuit that contains no repetition of vertices, other than the repetition of the starting and ending vertex. A simple cycle bounds a finite region in the plane that is a topological disk. We say a simple cycle *encloses* a vertex, an edge or a subgraph if the vertex, edge or subgraph is embedded in the topological disk bounded by the cycle. We say a circuit *encloses* a vertex, edge or subgraph if the vertex, edge or subgraph is enclosed by a simple cycle in the circuit.

The *level* of a vertex of G is defined as follows [6]: a vertex has level 0 if it is on the infinite face of G; a vertex has level i if it is on the infinite face after deleting all the vertices of levels less than i. Let V_i be the set of vertices of level i. Let E_i be the edge set of G in which each edge has both endpoints in level i. Let $E_{i,i+1}$ be the edge set of G where each edge has one endpoint in level i and one endpoint in level $i+1$. See

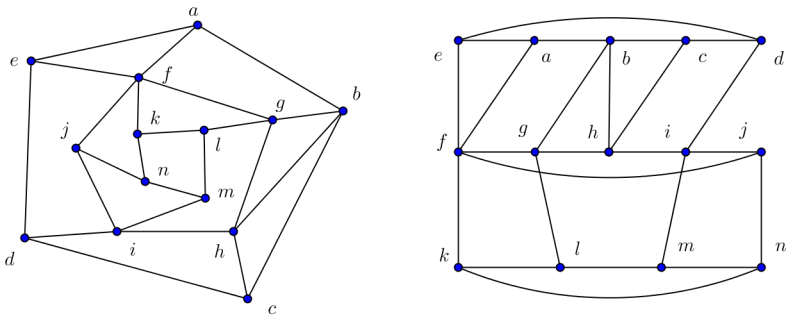

Figure 3.2: The three horizontal lines in the right figure show the three levels in the left figure. In this example, $V_0 = \{a, b, c, d, e\}$, $V_1 = \{f, g, h, i, j\}$ and $V_2 = \{k, l, m, n\}$.

Figure 3.2 as an example. Then we have the following observations.

Observation 3.5. *For any level $i \geq 0$, the boundary of any non-trivial two-edge connected component in $G[\cup_{j \geq i} V_j]$ is a maximal circuit in $\partial(G[V_i])$.*

Observation 3.6. *For any level $i \geq 0$, the boundary of any non-trivial biconnected component in $G[\cup_{j \geq i} V_j]$ is a simple cycle in $\partial(G[V_i])$.*

For any $i \geq 0$, we define the ith *double layer*

$$D_i = E_{i-1,i} \cup E_i \cup E_{i,i+1} \cup E_{i+1}$$

as the set of edges in $G[V_{i-1} \cup V_i \cup V_{i+1}] \setminus E_{i-1}$. See Figure 3.3

Let k be a constant that depends on ϵ. For $j = 0, 1, \ldots, k-1$, let $R_j = \cup_{i \mod k = j} D_i$. Let $t = \text{argmin}_j |R_j|$ and $R = R_t$. Since $\sum_{j=0}^{k-1} |R_j| \leq 2|G|$, we have the following upper bound for the size of R.

$$|R| \leq 2/k \cdot |G| \tag{3.1}$$

Let $f(i) = ik - k + t$ for integer $i \geq 0$. If $ik - k + t < 0$ for any i, we let $f(i) = 0$. Let $G_i = G[\cup_{f(i)-1 \leq j \leq f(i+1)+1} V_j]$ be the subgraph of G induced by vertices in level $[f(i) - 1, f(i + 1) + 1]$ and $H_i = G_i \setminus E_{f(i)-1}$ be a subgraph of G_i. See Figure 3.3. Note that H_i contains exactly the edges of double layers $D_{f(i)}$ through $D_{f(i+1)}$. Therefore, so long as $k \geq 2$, we have $H_i \cap H_{i+1} = D_{f(i+1)} \subseteq R$, and $H_i \cap H_j = \emptyset$ for any $j \neq i$ and

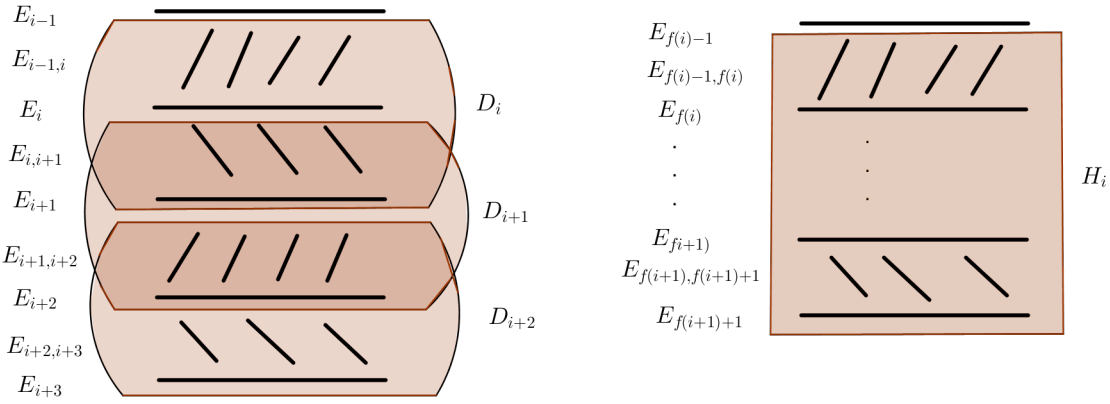

Figure 3.3: The horizontal lines represent edge set in the same level and slashes and counter slashes represent the edge set between two adjacent levels. Left: there are three double layers: D_i, D_{i+1} and D_{i+2} represented by the shaded regions. Right: G_i contains all edges in this figure, but H_i, represented by shaded region, does not contain $E_{f(i)-1}$.

$|i-j| \geq 2$. So for any $j \neq i$ we have

$$(H_i \setminus R) \cap (H_j \setminus R) = \emptyset. \tag{3.2}$$

For each $i \geq 0$ and each maximal circuit C_a in $\partial(G[V_{f(i)}])$, we construct a graph H_i^a, called a *3EC slice*, from G as follows. (See Figure 3.4.) Let U be the subset of vertices of $H_i \setminus (V_{f(i)-1} \cup V_{f(i+1)+1})$ that are enclosed by C_a. We contract each connected component of $G \setminus U$ into a node. After all the contractions, we delete self-loops and additional parallel edges if there are more than three parallel edges between any pair of vertices. The resulting graph is the 3EC slice H_i^a. We call these contracted vertices *nodes* to distinguish them from the original vertices of G. We call a contracted node *inner* if it is obtained by contracting a component that is enclosed by C_a; otherwise it is *outer*. Note that a 3EC slice is still planar, and two 3EC slices only share edges in a double layer: the common edges of two 3EC slices H_i^a and H_{i+1}^b must be in the set $E_{f(i+1)-1,f(i+1)} \cup E_{f(i+1)} \cup E_{f(i+1),f(i+1)+1}$, while the common edges of H_i^a and H_i^c must be in the set $E_{f(i)}$. In the similar way, we can construct a simple graph H_i^a, called *3VC slice*, for each $i \geq 0$ and each simple cycle C_a in $\partial(G[V_{f(i)}])$.

Remark 3.7. *There can be a 3EC slice H_i^a containing only two vertices in $V_{f(i)}$. Then*

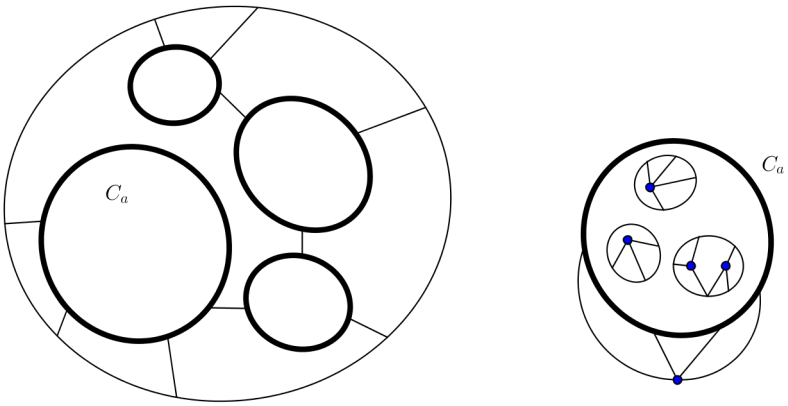

Figure 3.4: Example for the construction of H_i^a. Left: a component of G_i. The bold cycles represent maximal circuits in $\partial(G[V_{f(i)}])$. Right: an example of H_i^a. The nodes represent the contracted nodes. The cycles inside of C_a must belong to $E_{f(i+1)}$.

the slice must contain at least two parallel edges between the two vertices in $V_{f(i)}$. But any 3EC slice H_i^a cannot contain only one vertex in $V_{f(i)}$ since we define 3EC slice based on a maximal circuit and we assume there is no self-loop in G. Similarly, any 3VC slice H_i^a contains at least three vertices in $V_{f(i)}$.

Lemma 3.8. *If G is two-edge connected (biconnected), then each 3EC (3VC) slice obtained from G has at most one outer node.*

Proof. We first prove the following claims, and then by these claims we prove the lemma.

Claim 3.9. *For any $l \geq 0$, subgraph $G[\cup_{0\leq j\leq l} V_j]$ is connected.*

Proof. We prove by induction on l that subgraph $G[\cup_{0\leq j\leq l} V_j]$ is connected for any $l \geq 0$. The base case is $l = 0$. Since V_0 is the set of vertices on the boundary of G, and since G is connected, subgraph $G[V_0]$ is connected. Assume subgraph $G[\cup_{0\leq j\leq l} V_j]$ is connected for $l \geq 0$. Then we claim subgraph $G[\cup_{0\leq j\leq l+1} V_j]$ is connected. This is because for each connected component X of $G[V_{l+1}]$, there exists at least one edge between X and $G[V_l]$, otherwise graph G cannot be connected. Since subgraph $G[\cup_{0\leq j\leq l} V_j]$ is connected, we have $G[\cup_{0\leq j\leq l+1} V_j]$ is connected. □

Claim 3.10. *If G is two-edge connected, then for any two distinct maximal circuits C_a and C_b in $\partial(G[V_l])$, there is a path between C_b and $G[V_{l-1}]$ that is vertex disjoint from C_a.*

Proof. Note that C_a and C_b are vertex-disjoint, otherwise C_a is not maximal. We argue that there cannot be two edge-disjoint paths between C_a and C_b in $G[V_l]$. If there are such two edge-disjoint paths, say P_1 and P_2, then C_a cannot be a maximal circuit in $\partial(G[V_l])$: if P_1 and P_2 have the same endpoint in C_a, then C_a is not maximal; otherwise there is some edge of C_a that cannot be in $\partial(G[V_l])$. So we know that any vertex in C_a and any vertex in C_b cannot be two-edge connected in $G[V_l]$. Since G is two-edge connected and since $G[V_{l-1}]$ must be outside of C_a and C_b, vertices in C_a and those in C_b must be connected through $G[V_{l-1}]$. Therefore, there exists a path from C_b to $G[V_{l-1}]$ that does not contain any vertex in C_a. □

Similarly, we can obtain the following claim.

Claim 3.11. *If G is biconnected, then for any two distinct simple cycles C_a and C_b in $\partial(G[V_l])$, there is a path between C_b and $G[V_{l-1}]$ that is vertex disjoint from C_a.*

Now we prove the lemma. Let H be a 3EC slice based on some maximal circuit C_a in $\partial(G[V_l])$ for some $l \geq 0$. Let $W = \cup_{0 \leq j < l} V_j$ be the set of all vertices of G that have levels less than l, and Q be a two-edge connected component in $G[V_l]$ disjoint from H. Then the boundary of Q is a maximal circuit C in $\partial(G[V_l])$. Note that Q could be trivial and then C is also trivial. Since G is connected, each simple cycle must enclose a connected subgraph of G. So circuit C must enclose a connected subgraph of G. By Claim 3.10, there is a path between C and $G[V_{l-1}]$ disjoint from C_a. Since $G[\cup_{0 \leq j < l} V_j]$ is connected by Claim 3.9, the set of vertices that are not enclosed by C_a induces a connected subgraph of G, giving the lemma for H. For 3VC slice, we can obtain the lemma in the same way by Claim 3.9 and Claim 3.11. □

Using this lemma, we show how to construct all the 3EC slices in linear time. First we compute the levels of all vertices in linear time by using an appropriate representation of the planar embedding such as that used by Lipton and Tarjan [95]. We construct all 3EC slices H_i^a from G_i in $O(|V(G_i)|)$ time. We first contract all the edges between vertices of level $f(i+1)+1$. Next, we identify all two-edge connected components in

$G[V_{f(i)}]$, which can be done in linear time by finding all the edge cuts by the result of Tarjan [114]. Each such component contains a maximal circuit in $\partial(G[V_{f(i)}])$. Based on these two-edge connected components of $G[V_{f(i)}]$, we could identify $V(H_i^a) \setminus \{r_i^a\}$ for all 3EC slices H_i^a in $O(|V(G_i)|)$ time, where r_i^a is the outer contracted node for H_i^a. This is because the inner contracted nodes of a 3EC slice H_i^a is the same as those contracted in G_i if they are enclosed by C_a. Then for each 3EC slice H_i^a we add the outer node r_i^a, and for each vertex $u \in V(C_a)$ we add an edge between r_i^a and u if there is an edge between u and some vertex v that is not enclosed by C_a. To add those edges, we only need to traverse all the edges of subgraph $G_i[V_{f(i)-1} \cup V_{f(i)}]$. Since all these steps run in $O(|V(G_i)|$ time, and since $\sum_{i \geq 0} |V(G_i)| = O(|V(G)|)$, we can obtain the following lemma.

Lemma 3.12. *All 3EC slices can be constructed in $O(|V(G)|)$ time.*

Since we can compute all the biconnected components in $G[V_{f(i)}]$ in linear time based on depth-first search by the result of Hopcroft and Tarjan [74], we can obtain a similar lemma for 3VC sllices in a similar way.

Lemma 3.13. *All 3VC slices can be constructed in $O(|V(G)|)$ time.*

We review the definition of *branchwidth* given by Seymour and Thomas [113]. A *branch decomposition* of a graph G is a hierarchical clustering of its edge set. We represent this hierarchy by a binary tree, called the *decomposition tree*, where the leaves are in bijection with the edges of the original graph. If we delete an edge α of this decomposition tree, the edge set of the original graph is partitioned into two parts E_α and $E(G) \setminus E_\alpha$ according to the leaves of the two subtrees. The set of vertices in common between the two subgraphs induced by E_α and $E(G) \setminus E_\alpha$ is called the *separator* corresponding to α in the decomposition. The *width* of the decomposition is the maximum size of the separator in that decomposition, and the *branchwidth* of G is the minimum width of any branch decomposition of G. We borrow the following lemmas from Klein and Mozes [86], which are helpful in bounding the branchwidth of our graphs.

Lemma 3.14. (Lemma 14.5.1 [86]) *Deleting or contracting edges does not increase the branchwidth of a graph.*

Lemma 3.15. (Lemma 14.6.1 [86] rewritten) *There is a linear-time algorithm that, given a planar embedded graph G, returns a branch-decomposition whose width is at most twice of the depth of a rooted spanning tree of G.*

Lemma 3.16. *If G is two-edge connected (biconnected), the branchwidth of any 3EC (3VC) slice is $O(k)$.*

Proof. We prove this lemma for 3EC slices when G is two-edge connected; by the same proof we can obtain the lemma for 3VC slices when G is biconnected. Let H_i^a be a 3EC slice. By Lemma 3.8, there is at most one outer contracted node r for H_i^a. Let the level of r be $f(i) - 1$, and the level of all inner contracted nodes be $f(i+1) + 1$. Now we add edges to ensure that every vertex of level l has a neighbor of level $l - 1$ for each $f(i) \leq l \leq f(i) + 1$, while maintaining planarity. Call the resulting graph K_i^a. Then the branchwidth of H_i^a is no more than that of K_i^a by Lemma 3.14. Now we can find a breadth-first-tree of K_i^a rooted at r that has depth at most $k + 3$. By Lemma 3.15, the branchwidth of K_i^a is $O(k)$ and that of H_i^a is at most $O(k)$. □

3.3 PTAS for 3-ECSS

In this section, we prove Theorem 3.1. Our PTAS for 3-ECSS is shown in Algorithm 2.

Algorithm 2 A PTAS FOR 3-ECSS IN PLANAR GRAPHS

Input: a 3EC planar graph G and $\epsilon > 0$
Output: a $(1 + \epsilon)$-approximate 3-ECSS for G
Remove additional edges so that there are at most three parallel edges between any pair of vertices
Compute shared edge set R and all 3EC slices by Lemma 3.12 for $k = 36/\epsilon$
Define edge weights for each slice H_i^a: assign weight 0 to edges in $D_{f(i)} \cup D_{f(i+1)}$ and weight 1 to other edges
Solve the minimum-weight 3-ECSS in each slice by Theorem 3.3
Output the union of R and solutions from all slices

W.l.o.g. we assume G has at most three parallel edges between any pair of vertices. Then G is our spanner. Let $\mathrm{O}(G)$ be an optimal solution for G. Since each vertex in

$\mathrm{O}(G)$ has degree at least three, we have

$$2|\mathrm{O}(G)| \geq 3|V(G)|. \tag{3.3}$$

If G is simple, then by planarity the number of edges is at most three times of the number of vertices. Since there are at most three parallel edges between any pair of vertices, we have

$$|G| \leq 9|V(G)|. \tag{3.4}$$

Combining (3.3) and (3.4), we have $|G| \leq 6|\mathrm{O}(G)|$.

In this section, we only consider 3EC slices. So when we say slice, we mean 3EC slice in this section. We construct all the slices from G. By (3.1), we have the following

$$|R| \leq 2/k \cdot |G| \leq 12/k \cdot |\mathrm{O}(G)|. \tag{3.5}$$

We borrow the following lemma from Nagamochi and Ibaraki [103].

Lemma 3.17. (Lemma 4.1 (2) [103] rewritten) *Let G be a k-edge connected graph with more than 2 vertices. Then after contracting any edge in G, the resulting graph is still k-edge connected.*

Recall that our slices are obtained from G by contractions and deletions of self-loops. By the above lemma, we have the following lemma.

Lemma 3.18. *If G is three-edge connected, then any slice is three-edge connected.*

Since we can include all the edges in shared double layers, they are "free" to us. So we would like to include those edges as many as possible in the solution for each slice. This can be achieved by defining an edge-weight function w for each slice H_i^a: assign weight 0 to edges in $D_{f(i)} \cup D_{f(i+1)}$ and weight 1 to other edges. By Lemma 3.18, any slice is three-edge connected. We solve the minimum-weight 3-ECSS problem on H_i^a in linear time by Theorem 3.3. Let $Sol(H_i^a)$ be a feasible solution for the minimum-weight 3-ECSS problem on H_i^a. Then it is also a feasible solution for 3-ECSS on H_i^a. Let $\mathrm{O}_w(H_i^a)$ be an optimal solution for the minimum-weight 3-ECSS problem on H_i^a. Then we have the following observation.

Observation 3.19. *The weight of any solution $Sol(H_i^a)$ is the same as the number of its common edges with $H_i^a \setminus R$, that is*

$$w(Sol(H_i^a)) = |Sol(H_i^a) \cap (H_i^a \setminus R)|.$$

Let $\mathcal{C}_i$ be the set of all maximal circuits in $\partial(G[V_{f(i)}])$. Then we have the following lemmas.

Lemma 3.20. *For any $i \geq 0$, let $S_i = \bigcup_{C_a \in \mathcal{C}_i} \mathrm{O}_w(H_i^a)$. Then we can bound the number of edges in S_i by the following inequality*

$$|S_i| \leq |\mathrm{O}(G) \cap (H_i \setminus R)| + |D_{f(i)}| + |D_{f(i+1)}|.$$

Proof. We show that $\mathrm{O}(G) \cap H_i^a$ is a feasible solution for the minimum-weight 3-ECSS problem on H_i^a, and then we bound the size of S_i. Let Y_i^a be the set of vertices of H_i^a that are not contracted nodes. We first contract connected components of $\mathrm{O}(G) \setminus Y_i^a$ just as constructing H_i^a from G. Then we need to identify any two contracted nodes, if their corresponding components in $\mathrm{O}(G)$ are in the same connected component in $G \setminus Y_i^a$. See Figure 3.5. Finally, we delete all the self-loops and extra parallel edges if

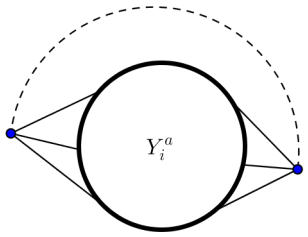

Figure 3.5: The bold cycle encloses Y_i^a. The dashed edge is in G but not in $\mathrm{O}(G)$. Its two endpoints will be identified, since the dashed edge will be contracted to obtain H_i^a but it will not be contracted when contracting connected components of $\mathrm{O}(G) \setminus Y_i^a$.

there are more than three parallel edges between any two vertices. The resulting graph is a subgraph of $\mathrm{O}(G) \cap H_i^a$ and spans $V(H_i^a)$. Since identifying two nodes maintains edge-connectivity, and since contractions also maintain edge-connectivity by Lemma 3.17, the resulting graph is three-edge connected. So $\mathrm{O}(G) \cap H_i^a$ is a feasible solution for minimum-weighted 3-ECSS problem on H_i^a. Then by the optimality of $\mathrm{O}_w(H_i^a)$, we

have $w(\mathrm{O}_w(H_i^a)) \leq w(\mathrm{O}(G) \cap H_i^a)$. And by Observation 3.19, we have

$$|\mathrm{O}_w(H_i^a) \cap (H_i^a \setminus R)| \leq |(\mathrm{O}(G) \cap H_i^a) \cap (H_i^a \setminus R)| = |\mathrm{O}(G) \cap (H_i^a \setminus R)|. \tag{3.6}$$

Note that for any slice H_i^a, we have $E(H_i^a) \subseteq E(H_i)$ and $(H_i^a \cap R) \subseteq (H_i \cap R) \subseteq (D_{f(i)} \cup D_{f(i+1)})$. Since for distinct (vertex-disjoint) maximal circuits C_a and C_b in $\mathcal{C}_i$, subgraphs $H_i^a \setminus R$ and $H_i^b \setminus R$ are vertex-disjoint, we have the following equalities.

$$H_i \setminus R = \bigcup\nolimits_{C_a \in \mathcal{C}_i} (H_i^a \setminus R) \tag{3.7}$$

$$S_i \cap (H_i \setminus R) = \bigcup\nolimits_{C_a \in \mathcal{C}_i} (\mathrm{O}_w(H_i^a) \cap (H_i^a \setminus R)) \tag{3.8}$$

Then

$$\begin{aligned} |S_i \cap (H_i \setminus R)| &= \left|\textstyle\bigcup_{C_a \in \mathcal{C}_i} (\mathrm{O}_w(H_i^a) \cap (H_i^a \setminus R))\right| && \text{by (3.8)} \\ &\leq \textstyle\sum_{C_a \in \mathcal{C}_i} |\mathrm{O}_w(H_i^a) \cap (H_i^a \setminus R)| \\ &\leq \textstyle\sum_{C_a \in \mathcal{C}_i} |\mathrm{O}(G) \cap (H_i^a \setminus R)| && \text{by (3.6)} \\ &\leq |\mathrm{O}(G) \cap (H_i \setminus R)|. && \text{by (3.7)} \end{aligned}$$

So we have $|S_i| = |S_i \cap (H_i \setminus R)| + |S_i \cap (H_i \cap R)| \leq |\mathrm{O}(G) \cap (H_i \setminus R)| + |D_{f(i)}| + |D_{f(i+1)}|$. □

Lemma 3.21. *The union* $\left(\bigcup_{i \geq 0, C_a \in \mathcal{C}_i} Sol(H_i^a)\right) \cup R$ *is a feasible solution for* G.

Proof. For any $i \geq 0$ and any maximal circuit $C_a \in \mathcal{C}_i$, let M_i^a be the graph obtained from H_i^a by uncontracting all its inner contracted nodes. See Figure 3.6. By Lemma 3.8, there is at most one outer node r_i^a for each slice H_i^a.

Define a tree T based on all the slices: each slice is a node of T, and two nodes H_i^a and H_j^b are adjacent if they share any edge and $|i - j| = 1$. Root T at the slice H_0^a, which contains the boundary of G. Let $T(H_i^a)$ be the subtree of T that roots at slice H_i^a. See Figure 3.6 as an example. For each child H_{i+1}^b of H_i^a, let C_b be the boundary of $H_{i+1}^b \setminus \{r_{i+1}^b\}$. Then C_b is the maximal circuit in $\mathcal{C}_{i+1}$ that is shared by H_i^a and H_{i+1}^b. Further, by the construction of H_i^a, graph $M_{i+1}^b \setminus \{r_{i+1}^b\}$ is a subgraph of M_i^a.

We prove the lemma by induction on this tree T from leaves to root. Assume for each child H_{i+1}^b of H_i^a, there is a feasible solution S^b for the graph M_{i+1}^b such that $S^b = \left(\bigcup_{H \in T(H_{i+1}^b)} Sol(H)\right) \cup \left(M_{i+1}^b \cap \left(\bigcup_{j \geq i+1} D_{f(j+1)}\right)\right)$. We prove that there is a feasible

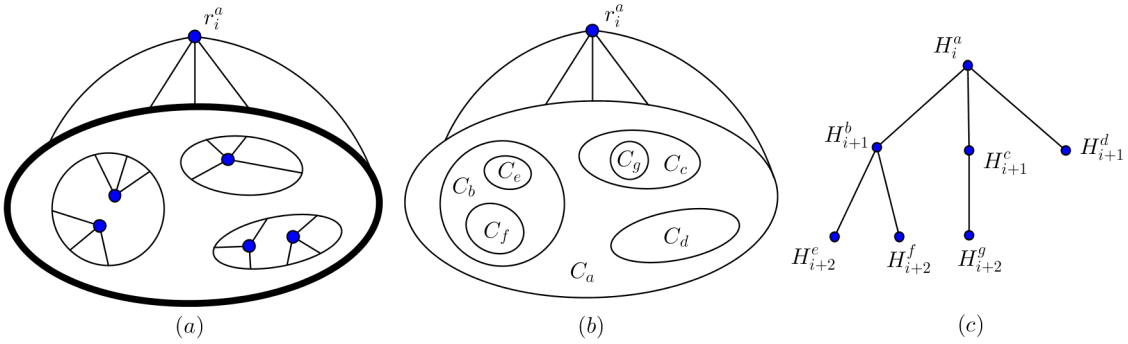

Figure 3.6: (a) A slice H_i^a: the bold cycle is a maximal circuit C_a in $\mathcal{C}_i$ and the nodes represent all the contracted nodes. (b) The graph M_i^a obtained from H_i^a by uncontracting inner nodes of H_i^a. (c) The subtree $T(H_i^a)$.

solution S^a for M_i^a such that $S^a = \left(\bigcup_{H \in T(H_i^a)} Sol(H)\right) \cup \left(M_i^a \cap \left(\bigcup_{j \geq i} D_{f(j+1)}\right)\right)$. For the root H_0^a of T, we have $M_0^a \cap \left(\bigcup_{j \geq 0} D_{f(j+1)}\right) \subseteq R$, and then the lemma follows from the case $i = 0$.

The base case is that H_i^a is a leaf of T. When H_i^a is a leaf, there is no inner contracted node in H_i^a and we have $M_i^a = H_i^a$. So $Sol(H_i^a)$ is a feasible solution for M_i^a.

Recall that H_i^a and H_{i+1}^b only share edges of $(E_{f(i+1)-1,f(i+1)} \cup E_{f(i+1)} \cup E_{f(i+1),f(i+1)+1}) \subseteq D_{f(i+1)}$ and vertices of $V_{f(i+1)}$. Let x be any inner contracted node of H_i^a and X be the vertex set of the connected component of G corresponding to x. We need the following claim.

Claim 3.22. *If $X \subseteq M_{i+1}^b$ for some H_{i+1}^b, then $(S^b \cup D_{f(i+1)}) \cap G[X]$ is connected.*

Proof. By the construction of levels, all the vertices on the boundary of $G[X]$ have level $f(i+1)+1$. See Figure 3.7. Then all the edges of $\partial(G[X])$ are in $E_{f(i+1)+1} \subseteq D_{f(i+1)}$. So subgraph $(S^b \cup D_{f(i+1)}) \cap \partial(G[X])$ is connected. Let u be any vertex in X and let v be any vertex in M_{i+1}^b that has level $f(i+1)$. Then v is not in X. Since S^b is a feasible solution for M_{i+1}^b, there exists a path from u to v in S^b. This path must intersect $\partial(G[X])$ by planarity. So u and any vertex on the boundary of $G[X]$ are connected in $(S^b \cup D_{f(i+1)}) \cap G[X]$, giving the claim. □

Let u and v be any two vertices of M_i^a. To prove the feasibility of S^a, we prove u and v are three-edge connected in S^a. Let $M = \left(\bigcup_{H_{i+1}^b \text{ is a child of } H_i^a} V(M_{i+1}^b \setminus \{r_{i+1}^b\})\right)$

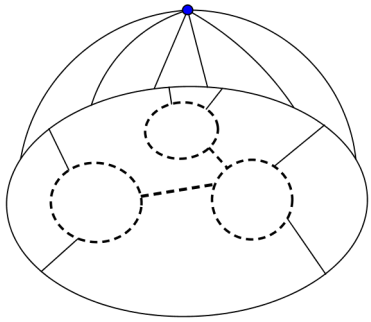

Figure 3.7: The dashed subgraph is the boundary of $G[X]$. All the vertices in the dashed subgraph are in level $f(i+1)+1$, and all its edges are in $E_{f(i+1)+1}$.

and $Y_i^a = V(H_i^a) \setminus \{$ inner contracted nodes of $H_i^a\}$. Then $V(M_i^a) = Y_i^a \cup M$. Depending on the locations of u and v, we have three cases.

Case 1: $\mathbf{u}, \mathbf{v} \in \mathbf{Y_i^a}$. Note that we could construct S^a in the following way. Initially we have $S^* = Sol(H_i^a) \cup (D_{f(i+1)} \cap H_i^a)$. For any inner contracted node x of H_i^a, let X be the vertex set of its corresponding connected component in G. Then there exists a child H_{i+1}^b of H_i^a such that $X \subseteq V(M_{i+1}^b)$, and we replace x with $(S^b \cup D_{f(i+1)}) \cap G[X]$ in S^*. We do this for all inner contracted nodes of H_i^a. Finally we add some edges of $D_{f(i+1)}$ into the resulting graph such that $D_{f(i+1)} \subseteq S^*$. Then the resulting S^* is the same as S^a by the definition of S^a. We prove that any pair of the remaining vertices in $V(H_i^a)$ are three-edge connected during the construction. This includes the remaining inner contracted nodes of H_i^a during the process, but after all the replacements, there is no such inner contracted nodes, proving the case.

By the definition of $Sol(H_i^a)$, any pair of vertices of H_i^a are three-edge connected in $Sol(H_i^a)$. Assume after the first k replacements, any pair of the remaining vertices in $V(H_i^a)$ are three-edge connected in the resulting graph S^*. Let x be the next inner contracted node to be replaced, X be the vertex set of its corresponding component and S' be the resulting graph after replacing x. Let H_{i+1}^b be the child of H_i^a such that $X \subseteq V(M_{i+1}^b)$. Let C be the simple cycle in $\partial(M_{i+1}^b \setminus \{r_{i+1}^b\})$ that encloses X. Then all vertices of C have level $f(i+1)$ and are shared by H_i^a and H_{i+1}^b. Further, $C \subseteq H_i^a \cap D_{f(i+1)} \subseteq S'$. Let u and v be any two remaining vertices

of $V(H_i^a)$. There are three edge-disjoint u-to-x paths and three edge-disjoint v-to-x paths in S^*, all of which must intersect C. So there exist three edge-disjoint u-to-X paths and three edge-disjoint v-to-X paths in S'. Now we delete two edges in S'. If these two edges are not both in C, then the vertices of C are still connected. Then one remaining u-to-C path and one remaining v-to-C path together with the rest of C witness the connectivity between u and v. If the two deleted edges are both in C, then there exist one u-to-X path and one v-to-X path after the deletion. By Claim 3.22, subgraph $(S^b \cup D_{f(i+1)}) \cap G[X]$ is connected. So all vertices of X are connected in S'. Then u and v are connected after the deletion. Finally, after replacing all the inner contracted nodes, we only add edges of $D_{f(i+1)}$ into S^*, which will not break three-edge-connectivity between any pair of vertices. This finishes the proof of Case 1.

Case 2: $\mathbf{u}, \mathbf{v} \in \mathbf{M}$. Let $M_{i+1}^{b_1}$ be the graph contains u and $M_{i+1}^{b_2}$ be the graph contains v. (The two graphs could be identical.) Let C_u (resp. C_v) be the simple cycle in $\partial(M_{i+1}^{b_1} \setminus \{r_{i+1}^{b_1}\})$ (resp. $\partial(M_{i+1}^{b_2} \setminus \{r_{i+1}^{b_2}\})$) that enclose u (resp. v). (The two cycles C_u and C_v could be identical.) Since S^{b_1} is three-edge connected, there are three edge-disjoint paths from u to some vertex of C_u in S^{b_1}. All these three paths must intersect C_u, so there are three edge-disjoint paths from u to C_u in $(S^{b_1} \setminus \{r_{i+1}^{b_1}\}) \subseteq S^a$. Similarly, there are three edge-disjoint paths from v to C_v in $(S^{b_2} \setminus \{r_{i+1}^{b_2}\}) \subseteq S^a$. Now we delete any two edges in S^a. After the deletion, there exist one u-to-w_1 path and one v-to-w_2 path where $w_1 \in C_u$ and $w_2 \in C_v$. Since all vertices in $V(C_u \cup C_v)$ have level $f(i+1)$ and are in Y_i^a, they are three-edge connected in S^a by Case 1. This means there exists a path from w_1 to w_2 after the deletion. Therefore, u and v are connected after deleting any two edges in S^a, giving the three-edge-connectivity.

Case 3: $\mathbf{u} \in \mathbf{Y_i^a}$ and $\mathbf{v} \in \mathbf{M}$. Let M_{i+1}^b be the graph containing v. Then there is a vertex w in $Y_i^a \cap (M_{i+1}^b \setminus \{r_{i+1}^b\})$. By Case 1, vertices u and w are three-edge connected, and by Case 2, vertices v and w are three-edge connected. Then vertices u and v are three-edge connected by the transitivity of three-edge-connectivity.

This completes the proof of Lemma 3.21. □

Proof of Theorem 3.1. We first prove correctness of our algorithm, and then prove its

running time. By Lemma 3.21, $S = \left(\bigcup_{i\geq 0, C_a\in\mathcal{C}_i} \mathrm{O}_w(H_i^a)\right) \cup R$ is a feasible solution. Thus

$$\begin{aligned}
|S| &\leq \left|\left(\bigcup_{i\geq 0, C_a\in\mathcal{C}_i} \mathrm{O}_w(H_i^a)\right)\right| + |R| \\
&\leq \sum_{i\geq 0} \left|\bigcup_{C_a\in\mathcal{C}_i} \mathrm{O}_w(H_i^a)\right| + |R| \\
&\leq \sum_{i\geq 0} \left(|\mathrm{O}(G)\cap(H_i\setminus R)| + |D_{f(i)}| + |D_{f(i+1)}|\right) + |R| && \text{by Lemma 3.20} \\
&\leq \sum_{i\geq 0} |\mathrm{O}(G)\cap(H_i\setminus R)| + |R| + |R| + |R| \\
&\leq |\mathrm{O}(G)| + 3|R| && \text{by (3.2)} \\
&\leq |\mathrm{O}(G)| + 36/k\cdot|\mathrm{O}(G)| && \text{by (3.5)} \\
&\leq (1+36/k)|\mathrm{O}(G)|
\end{aligned}$$

Let $k = 36/\epsilon$, and then we obtain $|S| \leq (1+\epsilon)|\mathrm{O}(G)|$.

Let $n = |V(G)|$ be the number of vertices of graph G. We could find R and construct all slices in $O(n)$ time by Lemma 3.12. By Lemma 3.16, each slice has branchwidth $O(k)$. So by Theorem 3.3, we could solve the minimum-weight 3-ECSS on each slice in linear time for fixed k. Based on those optimal solutions for all slices, we could construct our solution in $O(n)$ time. Therefore, our algorithm runs in $O(n)$ time. □

3.4 PTAS for 3-VCSS

In this section, we prove Theorem 3.2. Our PTAS for 3-VCSS is shown in Algorithm 3.

Algorithm 3 A PTAS FOR 3-VCSS IN PLANAR GRAPHS

Input: a 3VC planar graph G and $\epsilon > 0$
Output: a $(1+\epsilon)$-approximate 3-VCSS for G
Remove additional parallel edges so that the resulting graph is simple
Compute shared edge set R and all 3VC slices by Lemma 3.13 for $k = 12/\epsilon$
Define edge weights for each slice H_i^a: assign weight 0 to edges in $H_i^a \cap (D_{f(i)} \cup D_{f(i+1)})$ and weight 1 to other edges
Solve the minimum-weight 3-VCSS in each slice by Theorem 3.3
Output the union of R and solutions from all slices

W.l.o.g. we assume G has at most three parallel edges between any pair of vertices. Then G is our spanner. Let $\mathrm{O}(G)$ be an optimal solution for G. Since each vertex in $\mathrm{O}(G)$ has degree at least three, W.l.o.g. assume G is simple. Then G is our spanner. Let

$\mathrm{O}(G)$ be an optimal solution for G. Since G is simple and planar, we have $|G| \leq 3|V(G)|$. Then by (3.3) we have $|G| \leq 2|\mathrm{O}(G)|$. In this section, we only consider 3VC slices. So in the following, we simplify 3VC slice to slice. We first construct slices from G. By (3.1), we have the following

$$|R| \leq 2/k \cdot |G| \leq 4/k \cdot |\mathrm{O}(G)|. \tag{3.9}$$

Similar to 3-ECSS, we want to solve a minimum-weight 3-VCSS problem on each slice. But before defining the weights for this problem on each slice, we first need to show any slice is triconnected. The following lemma is proved by Vo [116], we provide a proof for completeness.

Lemma 3.23. ([116]) *Let C be a simple cycle of G that separates $G \setminus C$ into two parts: A and B. Let H be any connected component of A. If G is triconnected, then G/H, the graph obtained from G by contracting H, is triconnected.*

Proof. Let x be the contracted node of G/H. Then x and any other vertex of G/H are triconnected since G is triconnected. Let u and v be any two vertices of G/H distinct from x. To prove the lemma, we show u and v are triconnected. Since x and u are triconnected, there are three vertex-disjoint paths between u and x. Note that all the three paths must intersect cycle C since $V(C)$ form a cut for x and all the other vertices in G/H. Similarly, there are three vertex-disjoint paths between v and x, all of which intersect cycle C. Now we delete any two vertices different from u and v in G/H. If the two deleted vertices are both in C, then there exist one u-to-x path and one v-to-x path after the deletion, which witness the connectivity between u and v. If the two deleted vertices are not both in C, the remaining vertices in C are connected and then the remaining u-to-C path and the remaining v-to-C path together with the rest of edges in C witness the connectivity between u and v. So u and v are triconnected. □

By the same proof, we can obtain the following lemma.

Lemma 3.24. *Let C be a simple cycle of G. Let u and v be two vertices of $G \setminus C$ whose neighbors in G are all in C. Then if G is triconnected, the graph obtained from G by identifying u and v is triconnected.*

Let $\mathcal{C}_i$ be the set of all simple cycles in $\partial(G[V_{f(i)}])$. Then we have the following lemma.

Lemma 3.25. *For any $i \geq 0$ and any simple cycle $C_a \in \mathcal{C}_i$, the slice H_i^a is triconnected.*

Proof. Let Y_i^a be the set of vertices of H_i^a that are not contracted nodes. We could obtain H_i^a by contracting each connected component of $G \setminus Y_i^a$ into a node. Each time we contract a connected component H of $G \setminus Y_i^a$, there is a simple cycle C that separates H and other vertices: if H is outside of C_a, then $C = C_a$; otherwise C is some simple cycle in $\mathcal{C}_{i+1}$ that encloses H. Then by Lemma 3.23 the resulting graph is still triconnected after each contraction. Therefore, the final resulting graph H_i^a is triconnected. □

Now we define the edge-weight function w on a slice H_i^a: we assign weight 0 to edges in $H_i^a \cap (D_{f(i)} \cup D_{f(i+1)})$ and weight 1 to other edges. Then we solve the minimum-weight 3-VCSS problem on slice H_i^a by Theorem 3.3. Let $Sol(H_i^a)$ be a feasible solution for the minimum-weight 3-VCSS problem on H_i^a. Then it is also a feasible solution for 3-ECSS on H_i^a. Let $\mathrm{O}_w(H_i^a)$ be an optimal solution for this problem on H_i^a. Then we can prove the following two lemmas, whose proofs follow the same outlines of the proofs of Lemmas 3.20 and 3.21 respectively.

Lemma 3.26. *For any $i \geq 0$, let $S_i = \bigcup_{C_a \in \mathcal{C}_i} \mathrm{O}_w(H_i^a)$. Then we can bound the number of edges in S_i by the following inequality*

$$|S_i| \leq |\mathrm{O}(G) \cap (H_i \setminus R)| + |D_{f(i)}| + |D_{f(i+1)}|.$$

Proof. We first show $\mathrm{O}(G) \cap H_i^a$ is a feasible solution for minimum-weighted 3-VCSS problem on any slice H_i^a. Let Y_i^a be the set of vertices of H_i^a that are not contracted nodes. We first contract each component of $\mathrm{O}(G) \setminus Y_i^a$ into a node. The resulting graph after each contraction is still triconnected by Lemma 3.23. After all the contractions, we identify any two contracted nodes x_1 and x_2 if their corresponding components in $\mathrm{O}(G)$ are connected in $G \setminus Y_i^a$. This implies there exists a simple cycle C in $\mathcal{C}_i$ or $\mathcal{C}_{i+1}$ such that all neighbors of x_1 and x_2 are in C. So by Lemma 3.24 the resulting graph after each identification is also triconnected. Finally we delete parallel edges and self-loops if possible. After identifying all possible nodes, the resulting graph has the same vertex set as H_i^a and is triconnected. Since the resulting graph is a subgraph of $\mathrm{O}(G) \cap H_i^a$, we know $\mathrm{O}(G) \cap H_i^a$ is a feasible solution for minimum-weighted 3-VCSS problem on H_i^a.

Note that for any slice H_i^a, we have $(H_i^a \cap R) \subseteq (H_i \cap R) \subseteq (D_{f(i)} \cup D_{f(i+1)})$. By the optimality of $\mathrm{O}_w(H_i^a)$, we have $w(\mathrm{O}_w(H_i^a)) \leq w(\mathrm{O}(G) \cap H_i^a)$. Since all the nonzero-

weighted edges are in $H_i^a \setminus R$, Observation 3.19 still holds. Then we have

$$|\mathrm{O}_w(H_i^a) \cap (H_i^a \setminus R)| \leq |(\mathrm{O}(G) \cap H_i^a) \cap (H_i^a \setminus R)| = |\mathrm{O}(G) \cap (H_i^a \setminus R)|. \tag{3.10}$$

Since for distinct (edge-disjoint) simple cycles C_a and C_b in $\mathcal{C}_i$, subgraphs $H_i^a \setminus R$ and $H_i^b \setminus R$ are vertex-disjoint, we have the following equalities.

$$H_i \setminus R = \bigcup\nolimits_{C_a \in \mathcal{C}_i} (H_i^a \setminus R) \tag{3.11}$$

$$S_i \cap (H_i \setminus R) = \bigcup\nolimits_{C_a \in \mathcal{C}_i} (\mathrm{O}_w(H_i^a) \cap (H_i^a \setminus R)) \tag{3.12}$$

Then

$$\begin{aligned} |S_i \cap (H_i \setminus R)| &= \left|\bigcup\nolimits_{C_a \in \mathcal{C}_i} (\mathrm{O}_w(H_i^a) \cap (H_i^a \setminus R))\right| && \text{by (3.12)} \\ &\leq \sum\nolimits_{C_a \in \mathcal{C}_i} |\mathrm{O}_w(H_i^a) \cap (H_i^a \setminus R)| \\ &\leq \sum\nolimits_{C_a \in \mathcal{C}_i} |\mathrm{O}(G) \cap (H_i^a \setminus R)| && \text{by (3.10)} \\ &\leq |\mathrm{O}(G) \cap (H_i \setminus R)|. && \text{by (3.11)} \end{aligned}$$

So we have $|S_i| = |S_i \cap (H_i \setminus R)| + |S_i \cap (H_i \cap R)| \leq |\mathrm{O}(G) \cap (H_i \setminus R)| + |D_{f(i)}| + |D_{f(i+1)}|$. □

Lemma 3.27. *The union* $\left(\bigcup_{i \geq 0, C_a \in \mathcal{C}_i} Sol(H_i^a)\right) \cup R$ *is a feasible solution for* G.

Proof. For any $i \geq 0$ and any simple cycle $C_a \in \mathcal{C}_i$, let M_i^a be the graph obtained from slice H_i^a by uncontracting all the inner contracted nodes of H_i^a. By Lemma 3.8, there is at most one outer contracted node r_i^a for any slice H_i^a.

Define a tree T based on all the slices: each slice is a node of T, and two nodes H_i^a and H_j^b are adjacent if they share any edge and $|i - j| = 1$. Root T at the slice H_0^a, which contains the boundary of G. Let $T(H_i^a)$ be the subtree of T that roots at slice H_i^a. For each child H_{i+1}^b of H_i^a, let C_b be the simple cycle in $\mathcal{C}_{i+1}$ that is shared by H_i^a and H_{i+1}^b. Then C_b is the boundary of $H_{i+1}^b \setminus \{r_{i+1}^b\}$.

We prove the lemma by induction on this tree from leaves to root. Assume for each child H_{i+1}^b of H_i^a, there is a feasible solution S^b for the graph M_{i+1}^b such that $S^b = \left(\bigcup_{H \in T(H_{i+1}^b)} Sol(H)\right) \cup \left(M_{i+1}^b \cap \left(\bigcup_{j \geq i+1} D_{f(j+1)}\right)\right)$. We prove that there is a feasible solution S^a for M_i^a such that $S^a = \left(\bigcup_{H \in T(H_i^a)} Sol(H)\right) \cup \left(M_i^a \cap \left(\bigcup_{j \geq i} D_{f(j+1)}\right)\right)$. For the root H_0^a of T, we have $M_0^a \cap \left(\bigcup_{j \geq 0} D_{f(j+1)}\right) \subseteq R$, and then the lemma follows from

the case $i = 0$.

The base case is that H_i^a is a leaf of T. When H_i^a is a leaf, there is no inner contracted node in H_i^a and we have $M_i^a = H_i^a$. So $Sol(H_i^a)$ is a feasible solution for M_i^a.

We first need a claim the same as Claim 3.22. Note that any inner contracted node of H_i^a is enclosed by some cycle C_b. Let x be any inner contracted node of H_i^a that is enclosed by C_b, and X be the vertex set of the connected component of G corresponding to x. Then we have the following claim, whose proof is the same as that of Claim 3.22.

Claim 3.28. *If $X \subseteq M_{i+1}^b$ for some H_{i+1}^b, then $(S^b \cup D_{f(i+1)}) \cap G[X]$ is connected.*

Now we ready to prove S^a is a feasible solution for M_i^a. That is, we prove it is triconnected. Let u and v be any two vertices of M_i^a. Let $Y_i^a = V(H_i^a) \backslash \{$ inner contracted nodes of $H_i^a\}$. Since $V(M_i^a) = Y_i^a \cup \left(\bigcup_{H_{i+1}^b \text{ is a child of } H_i^a} V(M_{i+1}^b \setminus \{r_{i+1}^b\})\right)$, we have four cases.

Case 1: $\mathbf{u, v \in Y_i^a}$. For any contracted component X in G that corresponds to an inner contracted node of H_i^a, by Claim 3.28 all vertices in X are connected in $(S^b \cup D_{f(i+1)}) \cap G[X]$ if $X \subseteq M_{i+1}^b$. Then all vertices of X are connected in $S^a \cap G[X]$, since for any child H_{i+1}^b of H_i^a we have $(S^b \cup D_{f(i+1)}) \subseteq S^a$. By the triconnectivity of $Sol(H_i^a)$, there are three vertex-disjoint paths between u and v in $Sol(H_i^a)$. Since each inner contracted node of H_i^a could be in only one path witnessing connectivity, the three vertex-disjoint u-to-v paths in $Sol(H_i^a)$ could be transferred into another three vertex-disjoint u-to-v paths in S^a by replacing each contracted inner contracted node x with a path in the corresponding component X. So u and v are triconnected in S^a.

Case 2: $\mathbf{u, v \in M_{i+1}^b \setminus \{r_{i+1}^b\}}$. Since $V(C_b)$ is a cut for vertices enclosed by C_b and those not enclosed by C_b, by the triconnectivity of G we have $|V(C_b)| \geq 3$. By inductive hypothesis, S^b is a feasible solution for M_{i+1}^b, so there are three vertex-disjoint u-to-r_{i+1}^b paths in S^b. All these three paths must intersect C_b by planarity, so there are three vertex-disjoint u-to-C_b paths in $(S^b \setminus \{r_{i+1}^b\}) \subseteq S^a$. Similarly, there are three vertex-disjoint v-to-C_b paths in $(S^b \setminus \{r_{i+1}^b\}) \subseteq S^a$. If we delete any two vertices in S^a, then there exist at least one u-to-w_1 path and one v-to-w_2 path for some vertices $w_1, w_2 \in C_b$. Since all vertices in C_b have level $f(i+1)$, they are in Y_i^a. Then by Case 1, vertices w_1 and w_2 are triconnected in S^a, so they are

connected after deleting any two vertices. Therefore, u and v are also connected after the deletion.

Case 3: $\mathbf{u \in Y_i^a}$ and $\mathbf{v \in M_{i+1}^b \setminus \{r_{i+1}^b\}}$. If one of u and v is in C_b, they are triconnected by Case 1 or 2. So w.l.o.g. we assume u is not enclosed by C_b and v is strictly enclosed by C_b. Since G is triconnected, we have $|V(C_b)| \geq 3$. We could delete any two vertices in S^a and there exists at least one vertex w in C_b. By Case 1, vertices u and w are connected after the deletion, and by Case 2, vertices v and w are connected after the deletion. So u and v are connected after the deletion.

Case 4: $\mathbf{u \in M_{i+1}^{b_1} \setminus \{r_{i+1}^{b_1}\}}$ and $\mathbf{v \in M_{i+1}^{b_2} \setminus \{r_{i+1}^{b_2}\}}$. W.l.o.g. assume u is strictly enclosed by C_{b_1} and v is strictly enclosed by C_{b_2}, otherwise, by Case 3 they are triconnected. Since G is triconnected, we have $|V(C_{b_1})| \geq 3$. After deleting any two vertices in S^a, there exists a vertex $w \in C_{b_1}$. By Case 2, vertices u and w are connected after deletion, and by Case 3 vertices v and w are connected after deletion. So u and v are connected after deletion.

This completes the proof of Lemma 3.27. □

Proof of Theorem 3.2. We first prove the correctness, and then prove the running time. Let the union $S = \left(\bigcup_{i\geq 0, C_a \in \mathcal{C}_i} \mathrm{O}_w(H_i^a)\right) \cup R$ be our solution. By Lemma 3.27, the solution S is feasible for G. Then we have

$$
\begin{aligned}
|S| &= \left|\left(\bigcup_{i\geq 0, C_a \in \mathcal{C}_i} \mathrm{O}_w(H_i^a)\right) \cup R\right| \\
&= \left|\left(\bigcup_{i\geq 0, C_a \in \mathcal{C}_i} \mathrm{O}_w(H_i^a)\right)\right| + |R| \\
&\leq \textstyle\sum_{i\geq 0} \left|\bigcup_{C_a \in \mathcal{C}_i} \mathrm{O}_w(H_i^a)\right| + |R| \\
&\leq \textstyle\sum_{i\geq 0} \left(|\mathrm{O}(G) \cap (H_i \setminus R)| + |D_{f(i)}| + |D_{f(i+1)}|\right) + |R| && \text{by Lemma 3.26} \\
&\leq \textstyle\sum_{i\geq 0} |\mathrm{O}(G) \cap (H_i \setminus R)| + |R| + |R| + |R| \\
&\leq |\mathrm{O}(G)| + 3|R| && \text{by (3.2)} \\
&\leq (1 + 12/k)|\mathrm{O}(G)|. && \text{by (3.9)}
\end{aligned}
$$

We set $k = 12/\epsilon$ and then we have $|S| \leq (1+\epsilon)|\mathrm{O}(G)|$.

Let $n = |V(G)|$ be the number of vertices in graph G. We could find the edge set R in linear time. By Lemma 3.13 we could construct all slices in $O(n)$ time. So the slicing step runs in linear time. By Lemma 3.16, the branchwidth of each slice is $O(k) = O(1/\epsilon)$.

Therefore, we could solve the minimum-weight 3-VCSS problem on each slice in linear time by Theorem 3.3. Based on the optimal solutions for all the slices, we could construct our final solution S in linear time. So our algorithm runs in linear time. □

3.5 Dynamic Programming for Minimum-Weight 3-ECSS on Graphs with Bounded Branchwidth

In this section, we give a dynamic program to compute the optimal solution of minimum-weighted 3-ECSS problem on a graph G with bounded branchwidth w. This will prove Theorem 3.3 for the minimum-weight 3-ECSS problem. Our algorithm is inspired by the work of Czumaj and Lingas [38, 39]. Note that G need not be planar.

Given a branch decomposition of G, we root its decomposition tree T at an arbitrary leaf. For any edge α in T, let L_α be the separator corresponding to it, and E_α be the subset of $E(G)$ mapped to the leaves in the subtree of $T \setminus \{\alpha\}$ that does not include the root of T. Let H be a spanning subgraph of $G[E_\alpha]$. We adapt some definitions of Czumaj and Lingas [38, 39]. An *separator completion* of α is a multiset of edges between vertices of L_α, each of which may appear up to 3 times.

Definition 3.29. *A* configuration *of a vertex v of H for an edge α of T is a pair (A, B), where A is a tuple $(a_1, a_2, \ldots, a_{|L_\alpha|})$, representing that there are a_i edge-disjoint paths from v to the ith vertex of L_α in H, and B is a set of tuples (x_i, y_i, b_i), representing that there are b_i edge-disjoint paths between the vertices x_i and y_i of L_α in H. (We only need those configurations where $|a_i| \leq 3$ for all $0 \leq i \leq |L_\alpha|$ and $|b_i| \leq 3$ for all $i \geq 0$.) All the $\sum_{i=1}^{|L_\alpha|} a_i + \sum_i b_i$ paths in a configuration should be mutually edge-disjoint in H.*

Definition 3.30. *For any pair of vertices u and v in H, let $Com_H(u, v)$ be the set of separator completions of α each of which augments H to a graph where u and v are three-edge connected. For each vertex v in H, let $Path_H(v)$ be a set of configurations of v for α. Let $Path_H$ be the set of all the non-empty B in which all tuples can be satisfied in H. Let C_H be the set consisting of one value in each $Com_H(u, v)$ for all pairs of vertices u and v in H, and P_H be the set consisting of one value in each $Path_H(v)$ for all vertices v in H. We call the tuple $(C_H, P_H, Path_H)$ the* connectivity characteristic *of H, and denote it by $Char(H)$.*

Note that $|L_\alpha| \leq w$ for any edge α. Subgraph H may correspond to multiple C_H

and P_H, so H may have multiple connectivity characteristics. Further, each value in P_H represents at least one vertex. For any edge α, there are at most $4^{O(w^2)}$ distinct separator completions ($O(w^2)$ pairs of vertices, each of which can be connected by at most 3 parallel edges) and at most $2^{4^{O(w^2)}}$ distinct sets C_H of separator completions. For any edge α, there are at most $4^{O(w^2)}$ different configurations for any vertex in H since the number of different sets A is at most 4^w, the number of different sets B is at most $4^{O(w^2)}$ (the same as the number of separator completions). So there are at most $2^{4^{O(w^2)}}$ different sets of configurations P_H, and at most $2^{4^{O(w^2)}}$ different sets B. Therefore, there are at most $2^{4^{O(w^2)}}$ distinct connectivity characteristics for any edge α.

Definition 3.31. *A configuration (A, B) of vertex v for α is* connecting *if the inequality $\sum_{i=1}^{|L|} a_i \geq 3$ holds where a_i is the ith coordinate in A. That is, there are enough edge-disjoint paths from v to the corresponding separator L_α which can connect v and vertices outside L_α. $Char(H)$ is connecting if all configurations in its P_H set are connecting. Subgraph H is connecting if at least one of $Char(H)$ is connecting. In the following, we only consider connecting subgraphs and their connecting connectivity characteristics.*

In the following, we need as a subroutine an algorithm to solve the following problem: when given a set of demands (x_i, y_i, b_i) and a multigraph, we want to decide if there exist b_i edge-disjoint paths between vertices x_i and y_i in the graph and all the $\sum_i b_i$ paths are mutually edge-disjoint. Although we do not have a polynomial time algorithm for this problem, we only need to solve this on graphs with $O(w)$ vertices, $O(w^2)$ edges and $O(w^2)$ demands. So even an exponential time algorithm is acceptable for our purpose here. Let ALG be an algorithm for this problem, whose running time is bounded by a function $f(w)$, which may be exponential in w.

For an edge α in the decomposition tree T, let β and γ be its two child edges. Let H_1 (H_2) be a spanning subgraph of $G[E_\beta]$ ($G[E_\gamma]$). Let $H = H_1 \cup H_2$. Then we have the following lemma.

Lemma 3.32. *For any pair of $Char(H_1)$ and $Char(H_2)$, all the possible $Char(H)$, that could be obtained from $Char(H_1)$ and $Char(H_2)$, can be computed in $O(4^{w^2} f(w) + 4^{w^2 4^{w^2}})$ time.*

Proof. We compute all the possible sets for the three components of $Char(H)$.

Compute all possible $\mathbf{C_H}$ Each C_H contains two parts: the first part covers all pairs of vertices in the same H_i for $i = 1, 2$ and the second part covers all pairs of vertices from distinct subgraphs.

For the first part, we generalize each value $C \in C_{H_i}$ for $i = 1, 2$ into a possible set X_C. Notice that each separator completion can be represented by a set of demands (x, y, b) where x and y are in the separator. For a candidate separator completion C' of α, we combine C' with each $B \in Path_{H_{3-i}}$ to construct a graph H' and define the demand set the same as C. By running ALG on this instance, we can check if C' is a legal generalization for C. This could be computed in $4^{O(w^2)}w^2 + 4^{O(w^2)}f(w)$ time for each C. All the legal generalizations for C form X_C.

Now we compute the second part. Let $(A_1, B_1) \in P_{H_1}$ and $(A_2, B_2) \in P_{H_2}$ be the configurations for some pair of vertices $u \in H_1$ and $v \in H_2$ respectively. We will compute possible $Com_H(u, v)$. We first construct a graph H' on $L_\beta \cup L_\gamma \cup \{u, v\}$ by the two configurations: add i parallel edges between two vertices if there are i paths between them represented in the configurations. Then we check for each candidate separator completion C' if u and v are three-edge connected in $H' \cup C'$. We need $O(w^3)$ for this checking if we use Orlin's max-flow algorithm [104]. All those C' that are capable of providing three-edge-connectivity with H' form $Com_H(u, v)$. This can be computed in $4^{O(w^2)}w^3$ time for each pair of configurations.

A possible C_H consists of each value in X_C for every $C \in C_{H_i}$ for $i = 1, 2$ and each value in $Com_H(u, v)$ for all pairs of configurations of P_{H_1} and P_{H_2}. To compute all the sets, we need at most $4^{O(w^2)}w^3 + 4^{O(w^2)}f(w)$ time. There are at most $4^{O(w^2)}$ sets and each may contain at most $4^{O(w^2)}$ values. Therefore, to generate all the possible C_H from those sets, we need at most $4^{w^2 4^{O(w^2)}}$ time.

Compute all possible $\mathbf{P_H}$ We generalize each configuration (A, B) of v in P_{H_i} $(i = 1, 2)$ into a set Y_v of possible configurations. For each set B' in $Path_{H_{3-i}}$, we construct a graph H' by A, B and B' on vertex set $L_\beta \cup L_\gamma \cup \{v\}$: if there are b disjoint paths between a pair of vertices represented in A, B or B', we add b parallel edges between the same pair of vertices in H', taking $O(w^2)$ time. For a candidate value (A^*, B^*) for α, we define a set of demands according to A^* and B^* and run ALG on all the possible H' we construct for sets in $Path_{H_{3-i}}$. If there exists one such graph that satisfies all the demands, then we add this candidate value into Y_v. We can therefore compute each set Y_v in $4^{O(w^2)}w^2 + 4^{O(w^2)}f(w)$ time. A possible P_H consists of each value in Y_v for all

$v \in V(H)$. There are at most $4^{O(w^2)}$ such sets and each may contain at most $4^{O(w^2)}$ values. So we can generate all possible P_H from those sets in $4^{w^2 4^{O(w^2)}}$ time.

Compute Path$_\mathbf{H}$ For each pair of $B_1 \in Path_{H_1}$ and $B_2 \in Path_{H_2}$, we construct a graph H' on vertex set $L_\beta \cup L_\gamma$: if two vertices are connected by b disjoint paths, we add b parallel edges between those vertices in H'. Since each candidate B' for α can be represented by a set of demands, we only need to run ALG on all possible H' to check if B' can be satisfied. We add all satisfied candidates B' into $Path_H$. This can be computed in $4^{O(w^2)}w^2 + 4^{O(w^2)}f(w)$ time.

Therefore, the total running time is $O(4^{w^2}f(w) + 4^{w^2 4^{w^2}})$. For each component we enumerate all possible cases, and the correctness follows. □

Our dynamic programming is guided by the decomposition tree T from leaves to root. For each edge α, our dynamic programming table is indexed by all the possible connectivity characteristics. Each entry indexed by the connectivity characteristic $Char$ in the table is the weight of the minimum-weight spanning subgraph of $G[E_\alpha]$ that has $Char$ as its connectivity characteristic.

Base case For each leaf edge uv of T, the only subgraph H is the edge uv and the separator only contains the endpoints u and v. $Com_H(u, v)$ contains the multisets of edge uv that appears twice. $Path_H(u)$ contains two configurations: $((3, 0), \{(u, v, 1)\})$ and $((3, 1), \emptyset)$, and $Path_H(v)$ contains two configurations: $((0, 3), \{(u, v, 1)\})$ and $((1, 3), \emptyset)$. $Path_H$ contains one set: $\{(u, v, 1)\}$.

For each non-leaf edge α in T, we combine every pair of connectivity characteristics from its two child edges to fill in the dynamic programming table for α. The root can be seen as a base case, and we can combine it with the computed results. The final result will be the entry indexed by $(\emptyset, \emptyset, \emptyset)$ in the table of the root. Let $m = |E(G)|$. Then the size of the decomposition tree T is $O(m)$. By Lemma 3.32, we need $O(4^{w^2}f(w) + 4^{w^2 4^{w^2}})$ time to combine each pair of connectivity characteristics. Since there are at most $2^{4^{O(w^2)}}$ connectivity characteristics for each node, the total running time will be $O(2^{4^{w^2}}f(w)m + 4^{w^2 4^{w^2}}m)$. Since the branchwidth w of G is bounded, the running time will be $O(|E(G)|)$.

Correctness The separator completions guarantee the connectivity for the vertices in H, and the connecting configurations enumerate all the possible ways to connect vertices in H and vertices of $V(G) \setminus V(H)$. So the connectivity requirement is satisfied. The correctness of the procedure follows from Lemma 3.32.

Chapter 4: Local Search PTAS for Minimum Feedback Vertex Set in Minor-free Graphs

Given an undirected graph, the *minimum feedback vertex set problem* (FVS) asks for a minimum set of vertices such that after removing this set, the resulting graph has no cycle. This problem arises in a variety of applications, including deadlock resolution, circuit testing, artificial intelligence, and analysis of manufacturing processes [53]. Because of its importance, the FVS problem has been studied for a long time in algorithms . It is one of Karp's 21 original NP-Complete problems [80] and is NP-hard even in planar graphs [120]. The current best approximation ratio for the FVS problem in general graphs is 2 due to Becker and Geiger [10] and Bafna, Berman and Fujito [5].

There exist some PTAS results for this problem in some special classes of graphs. Kleinberg and Kumar [90] gave the first PTAS for the FVS problem in planar graphs, and Cohen-Addad, Colin de Verdière, Klein, Mathieu, and Meierfrankenfeld [34] gave a PTAS for the weighted version of this problem in bounded-genus graphs. Demaine and Hajiaghayi [42] gave an EPTAS for single-crossing-minor-free graphs. Fomin, Lokshtanov, Rauman and Saurabh [55] gave an EPTAS for H-minor-free graphs. However, their algorithm for H-minor-free graphs relies on a tree decomposition construction and other algorithmic tools based on bounded treewidth, such as Courcelle's theorem [36] or the similar result of Borie, Parker and Tovey [17], which makes the algorithm hard to apply in practice [78]. On the other hand, local search is a simple heuristic and there have been several experimental works on applying local search to FVS problem in general graphs, which show that local search gave good approximate solutions [121, 107]. However, no theoretical analysis of local search for the FVS problem was known before.

In this chapter, we show that a simple local search algorithm is a PTAS for the FVS problem in H-minor-free graphs. The algorithm is depicted in Algorithm 4. Intuitively, the local search algorithm starts with an arbitrary solution for the problem and tries to change a constant number (depending on ϵ) of vertices in the current solution to obtain a better solution. The algorithm outputs the current solution when it cannot obtain a better solution in this way.

Algorithm 4 PTAS for FVS by Local Search

Input: graph G and error parameter ϵ
$S \leftarrow$ an arbitrary solution of G
$c \leftarrow$ a constant depending on ϵ
while there is a solution S' such that $|S \setminus S'| \leq c$, $|S' \setminus S| \leq c$ and $|S'| < |S|$ **do**
    $S \leftarrow S'$
Output S

Theorem 4.1. *For any fixed $\epsilon > 0$, there is a local search algorithm that finds a $(1+\epsilon)$-approximate solution for the FVS problem in H-minor-free graphs with running time $O(n^c)$ where $c = \frac{\text{poly}(|V(H)|)}{\epsilon^2}$.*

To complement our positive result, we provide several negative results. First, we show that the FVS problem is APX-hard in 1-planar graphs. That implies the FVS problem is unlikely to have a PTAS beyond H-minor-free graphs. Our negative result contrasts with the positive results of Har-Peled and Quanrud [70], who show that local search provides PTASes for many problems including vertex cover, independent set, dominating set and connected dominating set, in graphs with polynomial expansion, which generalize 1-planar graphs and H-minor-free graphs. Second, we show that two closely related variants of the FVS problem, namely: *odd cycle transversal* and *subset feedback vertex set*, do not have such simple local search PTASes even in planar graphs. The odd cycle transversal (also called *bipartization*) problem asks for a minimum set of vertices in an undirected graph whose removal results in a bipartite graph. Given an undirected graph and a subset U of vertices, the subset feedback vertex set problem asks for a minimum set S of vertices such that after removing S the resulting graph contains no cycle that passes through any vertex of U. We show by examples, that the simple local search with constant exchanges cannot give a constant approximation in planar graphs.

4.1 Overview

Local search has been used before to obtain PTASes for other problems in H-minor-free graphs. Cabello and Gajser [29] gave a local search PTAS for the maximum independent set problem, the minimum vertex cover problem and the minimum dominating set

problem in H-minor-free graphs. Cohen-Addad, Klein and Mathieu [35] showed that local search yields PTASes for k-means, k-median and uniform uncapacitated facility location in H-minor-free graphs. All their analyses relies on the *exchange graph*, a graph constructed from the optimal solution[1] O and the local search solution L. For independent set and vertex cover, the exchange graph is simply the subgraph induced by $\mathrm{O} \cup \mathrm{L}$, and for the other problems, the exchange graph is built by contracting other vertices to nearest vertices in $\mathrm{O} \cup \mathrm{L}$. Then the local properties of these problems naturally appear in the exchange graphs: if we consider a small neighborhood R in the exchange graph and replace the vertices of L in R with the vertices of O in R and the boundary of R, the resulting vertex set is still a feasible solution for the original graph. Then by decomposing the exchange graph into small neighborhoods, we can bound the size of L by the size of O and all the boundaries of those neighborhoods.

However, the FVS problem does not have such local property if we construct exchange graph only by deletion or only by contraction. This is because for a cycle C in the original graph, the vertex of L that covers C may be inside of some neighborhood but the vertex of O that covers C may be outside of that neighborhood. One may try to argue the boundary of the neighborhood could cover C. But unfortunately, the boundary may not be helpful since the crossing vertices of C and the boundary may not be in both solutions and then they may be deleted or contracted to other vertices.

To solve this problem, we will construct an exchange graph with the following property: for any cycle C of the original graph, in our exchange graph there is either a vertex in $\mathrm{O} \cap \mathrm{L} \cap C$, or an edge between a vertex in $\mathrm{O} \cap C$ and a vertex in $\mathrm{L} \cap C$, or another cycle C' such that vertices in C' is a subset of vertices in C and $C' \cap (\mathrm{O} \cup \mathrm{L}) = C \cap (\mathrm{O} \cup \mathrm{L})$. To achieve this goal, we will apply both deletion and contraction to construct our exchange graph. Furthermore, we need to introduce vertices that are not in both solutions into the exchange graph, which is different from all previous exchange graph constructions. Meanwhile, we need to guarantee that the number of such vertices is linear in the size of $\mathrm{O} \cup \mathrm{L}$. The linear size bound is essential to the correctness of our algorithm and we prove this size bound by a structural lemma which may be of independent interest.

[1] For k-means and k-median, the exchange graph is constructed from L and a nearly optimal solution O', which is obtained by removing some vertices of O.

4.2 Preliminaries

For a graph G, we denote the vertex set and the edge set of G by $V(G)$ and $E(G)$, respectively. For a subgraph X of G, the *boundary* of X is the set of vertices that are in X but have at least one incident edge that is not in X. We denote by $int(X)$ the set of vertices of X that are not in the boundary of X. The *degree* of a vertex is the number of its incident edges.

Recall that a graph H is a *minor* of G if H can be obtained from G by a sequence of vertex deletions, edge deletions and edge contractions. G is *H-minor-free* if G does not contain a fixed graph H as a minor. The following theorem of Mader shows that H-minor-free graph is sparse.

Theorem 4.2 (Mader [98])**.** *An H-minor-free graph of n vertices has $O(\sigma_H n)$ edges where $\sigma_H = |V(H)|\sqrt{\log |V(H)|}$.*

Recall that a *balanced separator* of a graph is a set of vertices whose removal partitions the graph roughly in half. A separator theorem typically provides bounds for the size of each part and the size of the balanced separator. Usually, the size of the balanced separator is sublinear w.r.t. the size of the graph. Separator theorems have been found for planar graphs [47, 46, 63, 95], bounded-genus graphs [48, 64, 83], and minor-free graphs [2, 14, 82, 109, 119]. An *r-division* is a decomposition of a graph, which was first introduced by Frederickson [54] for planar graphs to speed up planar shortest path algorithm.

Definition 4.3. *For an integer r, an r-division of a graph G is a collection of edge-disjoint subgraphs of G, called* regions*, with the following properties.*

1. *Each region contains at most r vertices.*
2. *The number of regions is at most $c_d\frac{n}{r}$.*
3. *The number of boundary vertices, summed over all regions, is at most $c_d\frac{n}{\sqrt{r}}$.*

where c_d is a constant.

We say a graph is *r-divisible* if it has an r-division. Frederickson [54] gave a construction for the r-division of a planar graph which only relies on the separator theorem

in planar graphs [95]. It is straightforward to extend the construction to any family of graphs with balanced separator of sublinear size. Since H-minor-free graphs are known to have balanced separators (Alon, Seymour, and Thomas [2]), we have:

Theorem 4.4 (Alon, Seymour, and Thomas [2] + Frederickson [54])**.** *Minor-free graphs are r-divisible with $c_d = \text{poly}(|V(H)|)$.*

4.3 Exchange Graph Implies PTAS by Local Search

In this section, we show that if for an H-minor-free graph G, we can construct another graph, called an *exchange graph*, that it is r-divisible, then Algorithm 4 is a PTAS for the FVS problem. Let O be an optimal solution of the FVS problem and L be the output of the local search algorithm. We say a vertex u a *solution vertex* if $u \in \text{O} \cup \text{L}$ and a *Steiner vertex* otherwise. Unlike prior works [29, 70], we allow *Steiner vertices* in our exchange graphs.

Definition 4.5. *A graph* Ex *is an exchange graph for the optimal solution* O *and the local solution* L *of the FVS problem in a graph G if it satisfies the following properties:*

(1) $\text{L} \cup \text{O} \subseteq V(\text{Ex}) \subseteq V(G)$.

(2) $|V(\text{Ex})| \leq c_e(|\text{L}| + |\text{O}|)$ *for some constant c_e.*

(3) For every cycle C of G, there is (3a) a vertex of C in $\text{O} \cap \text{L}$ *or (3b) an edge $uv \in E(\text{Ex})$ between a vertex $u \in \text{L}$ and a vertex $v \in \text{O}$ in C or (3c) a cycle C' of* Ex *such that $V(C') \subseteq V(C)$ and $C \cap (\text{O} \cup \text{L}) = C' \cap (\text{O} \cup \text{L})$.*

Theorem 4.6. *If graph G has an r-divisible exchange graph for an optimal solution* O *and a local solution* L*, then Algorithm 4 is a polynomial-time approximation scheme for the FVS problem in G, whose running time is $n^{O(1/\epsilon^2)}$.*

Proof. We set the constant c in Algorithm 4 as $1/\delta^2$ where $\delta = \frac{\epsilon}{2c_d c_e(2+\epsilon)} = O(\frac{\epsilon}{c_d c_e})$. Note that c_d and c_e are constants in Definition 4.3 and Definition 4.5, respectively. Since in each iteration, the size of the solution is reduced by at least one, there are at most n iterations. Since each iteration can be implemented in $n^{O(c)}$ time by enumerating all possibilities, the total running time is $n^{O(c)} = n^{O(1/\epsilon^2)}$. We now show that the output L has size at most $(1+\epsilon)|\text{O}|$.

Let Ex be an r-divisible exchange graph for O and L. Since Ex is r-divisible, we can find an r-division of Ex for $r = c = \lceil 1/\delta^2 \rceil$. Let B be the multi-set containing all the boundary vertices in the r-division. By the third property of r-division, $|B|$ is at most $c_d \frac{|V(\text{Ex})|}{\sqrt{r}}$. By the second property of exchange graph, $|V(\text{Ex})| \leq c_e(|\text{O}| + |\text{L}|)$. Thus, we have:

$$|B| \leq c_d c_e \delta(|\text{O}| + |\text{L}|) \tag{4.1}$$

In the following, we will show that:

$$|\text{L}| \leq |\text{O}| + 2|B| \tag{4.2}$$

Then by Equation 4.1, we have:

$$|L| \leq |\text{O}| + 2c_d c_e \delta(|\text{O}| + |\text{L}|) = |\text{O}| + \frac{\epsilon}{2+\epsilon}(|\text{O}| + |\text{L}|)$$

that implies $|L| \leq (1+\epsilon)|\text{O}|$.

To prove Equation (4.2), we need to study some properties of Ex. For any region R_i of the r-division, let B_i be the boundary of R_i and M_i be the union of $\text{L} \setminus R_i$, $\text{O} \cap R_i$ and B_i.

Claim 4.7. *M_i is a feedback vertex set of G.*

Proof. For a contradiction, assume that there is a cycle C of G that is not covered by M_i. Then C does not contain any vertex of $\text{L} \setminus R_i$, $\text{O} \cap R_i$ and B_i. So C can only be covered by some vertices of $(\text{L} \setminus \text{O}) \cap int(R_i)$ and some vertices of $\text{O} \setminus (\text{L} \cup R_i)$. This implies that C does not contain any vertex of $\text{O} \cap \text{L}$ and there is no edge in Ex between $C \cap \text{O}$ and $C \cap \text{L}$. By the third property of exchange graph, there must be a cycle C' in Ex such that $V(C') \subseteq V(C)$ and $C \cap (\text{O} \cup \text{L}) = C' \cap (\text{O} \cup \text{L})$. Let u be the vertex of $(\text{L} \setminus \text{O}) \cap int(R_i)$ in C and v be the vertex of $\text{O} \setminus (\text{L} \cup R_i)$ in C. Then cycle C' contains both u and v, which implies C' crosses the boundary of R_i, that is $C' \cap B_i \neq \emptyset$. Let w be a vertex in $C' \cap B_i$, then w also belongs to C in G. This implies M_i contains a vertex of C, a contradiction. □

By the construction of M_i, we know the difference between L and M_i is bounded by

the size of the region R_i, that is r. Recall that $c = r = 1/\delta^2$. Since L is the output of Algorithm 4, we know L cannot be improved by changing at most r vertices. So we have $|\mathrm{L}| \leq |M_i|$. By the construction of M_i, this implies

$$|\mathrm{L} \cap R_i| \leq |M_i \cap R_i| \leq |\mathrm{O} \cap int(R_i)| + |B_i|.$$

That implies

$$|\mathrm{L} \cap int(R_i)| \leq |\mathrm{L} \cap R_i| \leq |\mathrm{O} \cap int(R_i)| + |B_i|.$$

Since $int(R_i)$ and $int(R_j)$ are vertex-disjoint for any two distinct i and j, by summing over all regions in the r-division, we can have

$$|\mathrm{L}| - |B| \leq \sum_i |\mathrm{L} \cap int(R_i)| \leq \sum_i (|\mathrm{O} \cap int(R_i)| + |B_i|) \leq |\mathrm{O}| + |B|.$$

This proves Equation (4.2). □

4.4 Exchange Graph Construction

Recall that $\sigma_H = |V(H)|\sqrt{\log |V(H)|}$ is the sparsity of H-minor-free graphs. In this section, we will show that:

Theorem 4.8. *H-minor-free graphs have exchange graphs for the FVS problem with $c_e = O(\sigma_H)$.*

Observe that Theorem 4.1 immediately follows from Theorem 4.8 and Theorem 4.6. The running time is $n^{O(c)}$ where $c = \frac{\mathrm{poly}(|V(H)|)}{\epsilon^2}$ since both c_d and c_e are polynomial in $|V(H)|$.

We construct the exchange graph in three steps:

Step 1 We delete all edges in G that are incident to vertices of $\mathrm{O} \cap \mathrm{L}$. We then remove all components that do not contain any solution vertex. Note that the removed components are acyclic.

Step 2 We contract edges that have an endpoint that is not a solution vertex and has degree at most two until there is no such an edge left. Since L and O are both feedback vertex set of G, every cycle after the contraction must contain a vertex

from L and a vertex from O. Since edges incident to vertices of O∩L are removed, there is no self-loop after this step.

Step 3 We keep the graph simple by removing all but one edge in each maximal set of parallel edges.

Let K be the resulting graph. We now show that K satisfies three properties in Definition 4.5. Property (1) is obvious because we never delete a vertex in $L \cup O$ from K. To show property (3), let C be a cycle of G. If any edge of C is removed in Step 1, C must contain a vertex in $\mathrm{O} \cap \mathrm{L}$; implying (3a). Thus, we can assume that no edge of C is deleted after Step 1. Since contraction does not destroy cycles, after the contraction in Step 2, there is a cycle C' such that $V(C') \subseteq V(C)$. If $|V(C')| = 2$ (C' is a cycle of two parallel edges), then (3b) holds. Thus, we can assume that every edge of C' remains intact after removing parallel edges. But that implies (3c) since we never remove solution vertices from G. Thus, K satisfies property (3).

It remains to show K satisfies property (2) in Definition 4.5, that is, $|V(K)| \leq O(\sigma_H)(|\mathrm{L}| + |\mathrm{O}|)$. By Step 2 we can have the following observation.

Observation 4.9. *Every Steiner vertex of K has degree at least 3.*

Since $\mathrm{O} \cup \mathrm{L}$ is a feedback vertex set of K, $K \setminus (\mathrm{O} \cup \mathrm{L})$ is a forest F containing only Steiner vertices. For each tree T in F, we define the *degree* of T, denoted by $\deg_K(T)$, as the number of edges in K between T and O∪L. Let $\ell(T)$ be the number of leaves of T. By Observation 4.9, every internal vertex of T has degree at least 3. Thus, $|V(T)| \leq 2\ell(T)$. That implies:

$$|V(T)| \leq 2 \deg_K(T). \tag{4.3}$$

We contract each tree T of F into a single Steiner vertex s_T. Let K' be the resulting graph. Then we have the following observation.

Observation 4.10. *Graph K' is simple.*

Proof. Since every cycle of K must contains a vertex from L and a vertex from O, there cannot be any solution vertex in K that is incident to more than one vertex of a tree T of F. So there cannot be parallel edges in K'. □

To bound the size of K', we need the following structural lemma. We remark that this lemma holds for general graphs.

Lemma 4.11. *For a graph G and its two disjoint nonempty vertex subsets A and B, let $D = V(G) \setminus (A \cup B)$. If (i) D is an independent set, (ii) every vertex in D has degree at least 3 in G and (iii) for every cycle C in G, we have $C \cap A \neq \emptyset$ and $C \cap B \neq \emptyset$, then we have $|V(G)| \leq 2(|A| + |B|)$.*

Proof. We remove every edge that only has endpoints in $A \cup B$ and let the resulting graph be G'. Then G' is a bipartite graph with $A \cup B$ in one side and D in the other side since D is an independent set. Let D_A (D_B) be the subset of D only containing the vertices that have at least two neighbors in A (B). Since every vertex of D has degree at least 3, we have $D_A \cup D_B = D$.

Let H_A be the subgraph of G' induced by $A \cup D_A$. Then H_A is acyclic since otherwise every cycle of H_A would correspond to a cycle in G that does not contain any vertex in B. We now construct a graph H_A^* on vertex set A. For each vertex $v \in D_A$, we arbitrarily choose its two neighbors x and y in A and add an edge between x and y in H_A^*. By construction, there is a one-to-one mapping between edges of H_A^* and vertices of D_A.

Since H_A is acyclic, H_A^* is also acyclic. Thus, $|E(H_A^*)| \leq V(H_A^*) = |A|$. That implies $|D_A| \leq |A|$. By a similar argument, we can show that $|D_B| \leq |B|$. Thus, $|D| = |D_A \cup D_B| \leq |A| + |B|$, and the lemma follows. □

Let Z be an arbitrary component of K' that contains at least one Steiner vertex. Then the two sets $V(Z) \cap \mathrm{O}$ and $V(Z) \cap \mathrm{L}$ must be disjoint since any vertex in $\mathrm{O} \cap \mathrm{L}$ is isolated in K'. And each of the two sets cannot be empty since there must be a cycle in Z through the Steiner vertex which also contains a vertex of O and a vertex of L respectively. Let X be the set of Steiner vertices in Z. By the construction of K', vertex set X is an independent set of Z. By Observation 4.9, every vertex of X has degree at least 3. So we can apply Lemma 4.11 for Z, $V(Z) \cap \mathrm{O}$ and $V(Z) \cap \mathrm{L}$, and obtain $|V(Z)| \leq 2(|V(Z) \cap \mathrm{O}| + |V(Z) \cap \mathrm{L}|) = 2(|V(Z) \cap \mathrm{O}| + |V(Z) \cap (\mathrm{L} \setminus \mathrm{O})|)$. Note that this bound holds trivially if Z does not contain any Steiner vertex. Thus, summing over all components of K', we have $|V(K')| \leq 2(|V(K') \cap \mathrm{O}| + |V(K') \cap (\mathrm{L} \setminus \mathrm{O})|) \leq 2(|\mathrm{O}| + |\mathrm{L}|)$.

Since K' is a minor of G , it is also H-minor-free. By Theorem 4.2, we have

$$\begin{aligned} |E(K')| &= O(\sigma_H |V(K')|) \\ &= O(\sigma_H)(|\mathrm{O}| + |\mathrm{L}|) \end{aligned} \tag{4.4}$$

We now ready to bound the size of $V(K)$. We have:

$$\begin{aligned} |V(K) \setminus (\mathrm{O} \cup \mathrm{L})| &= \textstyle\sum_{T \in F} |V(T)| \\ &\leq 2 \textstyle\sum_{T \in F} \deg_K(T) && \text{(Equation (4.3))} \\ &= 2 \textstyle\sum_{T \in F} \deg_{K'}(s_T) \\ &\leq 2|E(K')| && (\{s_T | T \in F\} \text{ is an independent set}) \\ &= O(\sigma_H)(|\mathrm{O}| + |\mathrm{L}|) && \text{(Equation (4.4))} \end{aligned}$$

This implies $V(K) \leq O(\sigma_H)(|\mathrm{O}| + |\mathrm{L}|)$. Thus K satisfies property (2) in Definition 4.5 with $c_e = O(\sigma_H)$, thereby, implying Theorem 4.8.

4.5 Negative Results

In this section, we show some negative results for the FVS problem and its variants. A graph is *1-planar* if it can be drawn in the Euclidean plane such that every edge has at most one crossing, where it crosses a single additional edge. We first show that FVS is APX-hard in 1-planar graphs. Then for the two variants, odd cycle transversal and subset feedback vertex set, we construct examples where local search with constant-size exchanges cannot give a constant approximation in planar graphs.

Theorem 4.12. *Given a graph G, we can construct a 1-planar graph H in polynomial time, such that G has a feedback vertex set of size at most k if and only if H has a feedback vertex set of size at most k.*

Proof. Consider a drawing of non-planar G on the plane where each pair of edges can cross at most once. For each crossed edge e in G, we subdivide e into edges so that there is exactly one crossing per new edge. Let H be the resulting graph. By construction, graph H is 1-planar.

Let n be the size of G. Since there are at most $O(n^2)$ crossings per edge in the drawing, the size of H is at most $O(n^4)$. Sine we only subdivide edges, there is a one-

to-one mapping between cycles of G and cycles of H. It is straightforward to see that any feedback vertex set of G is also a feedback vertex set of H.

Let S be a feedback vertex set of H. If $S \subseteq V(H) \cap V(G)$, then it is also a feedback vertex set for G. Otherwise, let $v \in V(H) \setminus V(G)$ be a vertex in S. Then v must be added to subdivide an edge, say e, in G. We remove v from S and add an arbitrary endpoint of e in G to S. Then S is still a feedback vertex set for H. We repeat this process until S is a subset of $V(H) \cap V(G)$. Observe that S is a feedback vertex set of size at most k for G. Thus, the lemma holds. □

Since the FVS problem is APX-hard in general graphs (by an approximation preserving reduction [80] from vertex cover problem, which is APX-hard [45]), Theorem 4.12 implies that FVS is APX-hard in 1-planar graphs.

To show that simple local search cannot give a constant approximation for the odd cycle transversal problem and the subset feedback vertex set problem, we construct a counter-example from a $k \times k$ grid as shown in Figure 4.1. The grid could be arbitrarily large. We add one edge in some diagonal cells of the grid. We first consider the odd cycle transversal problem. In the left figure, circles represent vertices of the optimal solution, and triangles represent vertices of the local search solution. The grid could be arbitrarily large. We add one edge in some diagonal cells of the grid. Since any grid is bipartite and does not contain any odd cycle, any odd cycle in the example must contain an edge in the diagonal cell. All the vertices in the diagonal, represented by triangles, give a solution that is locally optimal, that is, we cannot improve this solution by changing a small number of vertices. This is because each triangle vertex and each new edge, together with some other edges, can form at least one odd cycle in the graph. For a constant c that is smaller than the size of optimal solution, if we remove c triangle vertices, say V', in the locally optimal solution, there will be c vertex-disjoint odd cycles in the resulting graph, each of which contains one removed triangle. Thus, there is no subset of size less than c that can replace V'. Then the ratio between the two solutions could be arbitrarily big if the gird is arbitrarily big and the number of added diagonal edges is super-constant and sublinear to the size of the diagonal. For subset feedback vertex set, the diamonds in the right figure represent the vertices in the given set U. Similarly, any cycle through a given vertex must contain the two edges in the diagonal cell. By the same reason, the local search solution cannot be improved.

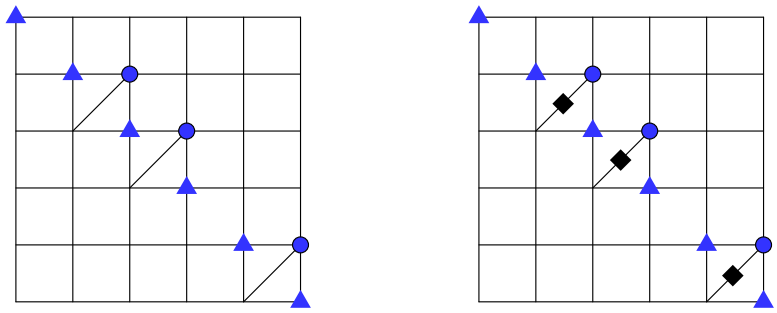

Figure 4.1: Counterexamples for local search on odd cycle transversal (left) and subset feedback vertex set (right). Circles represent vertices of the optimal solution, and triangles represent vertices of the local search solution. The grid could be arbitrarily large. We add one edge in some diagonal cells of the grid.

Chapter 5: Practical PTAS and Heuristics for Minimum Feedback Vertex Set in Planar Graphs

Researchers have proposed some heuristic algorithms for FVS and evaluated their performance in experiments. For example, Pardalos et al. [106] developed a greedy randomized adaptive search procedure for FVS, Brunetta et al. [28] proposed a system based on local search and a branch-and-cut algorithm, Zhang et al. [121] presented a variable depth-based local search algorithm with a randomized scheme, and Qin and Zhou [107] introduced a simulated annealing local search algorithm for FVS. However, all of these works focus on general graphs. Brunetta et al. [28] included planar graphs in their experiments, but their test planar graphs were not very large, having at most one thousand of vertices. So it is natural to ask the following question:

which algorithm is preferred for FVS on large planar graphs in practice?

One potential answer to this question is a PTAS. There are two reasons supporting this choice: theoretically, PTASes can provide the best approximation ratio; practically, it has been shown that PTASes can be made practical for the minimum dominating set problem [99], the Steiner tree problem [115] and the traveling salesman problem (TSP) [9] in large planar graphs. Unfortunately, we find that a simple PTAS for FVS does not find a more accurate solution than the 2-approximation algorithm in most real-world graphs and some synthetic graphs. For those test graphs where it can find better solutions, the improvement is less than 1 percent. Furthermore, the PTAS is much slower than the 2-approximation algorithm. So another question to ask is

can we obtain a practical algorithm that can find more accurate solutions than the 2-approximation algorithm in large planar graphs?

To answer this question, we propose a heuristic algorithm and show that it can outperform the 2-approximation algorithm in real-world graphs and most large synthetic graphs.

5.1 Overview

To answer the first question, we implemented Becker and Geiger's 2-approximation algorithm [10] as our baseline, which is simpler than the algorithm of Bafna, Berman and Fujito [5]. In Section 5.3.1, we evaluate this implementation on some graphs where we can obtain optimal solutions or good lower bounds for the optimal solutions. We find that the 2-approximation algorithm finds solutions that are very close to the optimal in these instances.

To outperform this baseline, in Section 5.2.3 we propose a simple-to-implement $O(n \log n)$ PTAS for FVS in planar graphs, which starts with a linear kernel for FVS (see Section 5.2.2) and then uses a balanced separator (see Section 5.2.3.1) applied recursively to decompose the graph into a set of small subgraphs in which we will solve the problem optimally. The approach based on balanced separators has been applied to obtain PTASes for the maximum independent set problem [96] and the minimum vertex cover problem [32] in planar graphs. However, this approach is criticized in literature for two reasons: (1) a good approximation ratio can only be obtained in very large graphs [33, 6, 42], and (2) it needs the size of the optimal solution to be linear w.r.t. the size of input graph [66]. We overcome both issues. For the first, we relate the error parameter ϵ to the largest size of the decomposed graphs instead of the size of the original graph, and the algorithm can provide good approximation ratio for any graph in this way. For the second, we use a linear kernel as a proxy to achieve the linear bound for the optimal solution. Since many problems [56, 57, 15] admit linear kernels in minor-free graphs, which is a more general graph family and admits balanced separators of sublinear size [2], we believe this approach could be applied more generally.

Other PTASes for planar FVS that have been proposed are either complicated to implement (relying on dynamic programming over tree decompositions) or not sufficiently efficient (having running time of the form $n^{O(1/\epsilon)}$). Compared to those, our PTAS has some obvious advantages: (1) it only relies on some simple algorithmic components like the kernelization algorithm, which consists of a sequence of simple reduction rules, and balanced separators, which are known to be practical [1, 73, 58]; (2) it has very few parameters to optimize; (3) the constants behind the big O notation are small enough and (4) its running time is theoretically faster than previous PTASes. Performance of this PTAS on different large planar graphs is discussed in Section 5.3.2. We also incorporated

heuristic steps (Section 5.2.4.1) to improve its solution and analyzed the influence of the parameters of the heuristic. However, counter to the success stories for the Steiner tree and TSP PTASes, we find that the solution found by this PTAS does not outperform the precision of the 2-approximation algorithm significantly.

Although our PTAS does not outperform the 2-approximation algorithm significantly, we use it as inspiration to engineer a PTAS-like heuristic we call a *Heuristic Approximation Scheme (HAS)*, that is a heuristic with a running-time/precision trade-off. Our HAS has two main steps. The first step (see Section 5.2.4.2) is a hybrid algorithm that alternates the reduction rules of the linear kernel and the greedy step of the 2-approximation algorithm. The second step (see Section 5.2.4.3) is a variant of local search. Many local search heuristics start with a feasible solution and repeatedly construct a smaller solution by replacing a subset A of the original solution with another smaller subset of the non-solution vertices, if it is feasible. We notice that this could be inefficient, since there are too many ways to replace the subset A and so only very small values of $|A|$ can be handled. Instead, we use a *fixed-parameter tractable* (FPT) algorithm to determine the replacement for A. An algorithm is FPT if it can solve a given problem optimally in running time $f(k) \cdot n^{O(1)}$, where k is given as a fixed parameter (such as the size of the optimal solution, as for FVS) and f is an arbitrary computable function. This kind of algorithm is very efficient when the parameter k is small. Now given a feasible solution, our local search heuristic will repeatedly improve the solution by selecting a set A from the solution, constructing a graph as the union of the non-solution vertices and set A, solving the problem in this graph optimally with the FPT algorithm, and replacing A with the obtained optimal solution.

We implemented and evaluated HAS on different large planar graphs and analyzed the effects of its parameters (Section 5.3.3). Our result shows that even its first step is able to find better solutions than the 2-approximation algorithm on most of our test graphs and its second step improves these solutions further. As a result, the total improvement for all real-world graphs is at least 5 percent, which is more than 30000 vertices in the largest test graph.

We find that HAS is very flexible and provides a kind of "PTAS behavior". Its first step is competitive w.r.t. the running time so we can obtain a good solution quickly. And its second step can be applied for a longer time to obtain a better solution when the running time is not strictly limited. Thus, it can provide a trade-off between the

running time and the solution quality by choices of the number of local search iterations. Therefore, we believe this algorithm will be a better choice in practice.

5.2 The Algorithms for FVS in Planar Graphs

In this section, we briefly summarize the FVS algorithms we implemented for planar graphs: the 2-approximation algorithm of Becker and Geiger [10], Bonamy and Kowalik's linear kernel [16] for FVS in planar graphs (with optimizing modifications that we designed), our new PTAS using this linear kernel and balanced planar separators, and our proposed Heuristic Approximation Scheme.

5.2.1 The 2-Approximation Algorithm

Becker and Geiger's 2-approximation algorithm [10] works for vertex-weighted FVS in general graphs and consists of two steps: (1) computes a greedy solution and (2) removes vertices to obtain a minimal solution. In the first step, the algorithm assigns a score for each vertex (weight of the vertex divided by its degree), and repeatedly removes a vertex with minimum score from the graph and adds it to the greedy solution. Each time a vertex is removed, the scores of its neighbors are updated. In the second step, the algorithm tries to remove the vertices from the greedy solution in the reverse order in which they were added, to obtain a minimal feasible solution.

5.2.2 Kernelization Algorithm

A parameterized decision problem with a parameter k admits a *kernel* if there is a polynomial time algorithm (where the degree of the polynomial is independent of k), called a *kernelization algorithm*, that outputs a decision-equivalent instance whose size is bounded by some function $h(k)$. If the function $h(k)$ is linear in k, then we say the problem admits a *linear* kernel.

Bonamy and Kowalik's linear kernel for planar FVS [16] consists of a sequence of 17 reduction rules. Each rule replaces a particular subgraph with another (possibly empty) subgraph, and possibly marks some vertices that must be in an optimal solution. The first 12 rules are simple and sufficient to obtain a $15k$-kernel [16]. Since the remaining

rules do not improve the kernel by much, and since Rule 12 is a rejecting rule (that is to return a trivial no-instance for the decision problems when the resulting graph has more than $15k$ vertices), we only implement the first 11 rules (provided in the appendix), all of which are local and independent of the parameter k. The algorithm starts by repeatedly applying the first five rules to the graph and initializes two queues: queue Q_1 contains some vertex pairs that are candidates to check for Rule 6 and queue Q_2 contains vertices that are candidates to check for the last five rules. While Q_1 is not empty, the algorithm repeatedly applies Rule 6, reducing $|Q_1|$ in each step. Then the algorithm repeatedly applies the remaining rules in order, reducing $|Q_2|$ until Q_2 is empty. After applying any rule, the algorithm updates both queues as necessary, and will apply the first five rules if applicable.

We list the reduction rules we applied in our implementation. The original Rule 10 has two parts, and we list it as two independent rules here. We denote the set of neighbors of a vertex x by $N(x)$ and the degree of x in a multigraph by $deg(x)$.

Rule 1. If there is a loop at a vertex v, then remove v and add it to the optimal solution.

Rule 2. Delete vertices of degree at most one.

Rule 3. If a vertex u is of degree two, with incident edges uv and uw, then delete u and add the edge vw. (Note that if $v = w$ then a loop is added.)

Rule 4. If a vertex u has exactly two neighbors v and w, edge uv is double, and edge uw is simple, then delete v and u and add v into the optimal solution.

Rule 5. If there are at least three edges between a pair of vertices, remove all but two of the edges.

Rule 6. Assume that there are five vertices a, b, c, v, w such that 1) both v and w are neighbors of each of a, b, c and 2) each vertex $x \in \{a, b, c\}$ is incident with at most one edge xy such that $y \notin \{v, w\}$. Then remove all the five vertices and add v and w into the optimal solution.

Rule 7. If a vertex u has exactly three neighbors v, w and x, vertex v is also adjacent to w and x, and both edges uw and ux are simple, then contract uv and add an edge wx (increasing its multiplicity if it already exists). If edge uv was not simple, add a loop at v.

Rule 8. Assume there are seven vertices v_1, v_2, v_3, u_1, u_2, w_1, w_2, such that $N(u_1) = \{w_1, w_2, u_2\}$, $N(v_1) = \{w_1, w_2, v_2\}$, $N(v_2) = \{w_1, v_1, v_3\}$, and $deg(v_2) = deg(u_1) = 3$. Then contract the edge v_1v_2 to a new vertex y and add an edge w_1v_3.

Rule 9. Assume $u_1u_2u_3u_4$ is an induced path such that for two vertices w_1, w_2 outside the path, $N(u_1) = \{u_2, w_1, w_2\}$, $N(u_2) = \{u_1, u_3, w_1\}$ and $N(u_3) = \{u_2, u_4, w_2\}$, $deg(u_3) = 3$, and $\min\{deg(u_1), deg(u_2)\} = 3$. Then remove u_1 and u_2 and add a new vertex y, simple edges yu_3 and u_3w_1, and double edges yw_1 and yw_2.

Rule 10. Assume there are five vertices v_1, v_2, u, w_1, w_2 such that $N(v_1) = \{v_2, w_1, w_2\}$, $\{w_1, w_2\} \subseteq N(u)$, there is at most one edge incident to v_2 and a vertex outside $\{w_1, w_2, v_1\}$, and there is at most one edge incident to u and a vertex outside $\{w_1, w_2\}$. Then remove w_1 and add it to the optimal solution.

Rule 11. Assume there are five vertices u_1, u_2, u_3, w_1, w_2 such that $N(u_1) = \{w_1, w_2, u_2\}$, $\{u_1, u_3\} \subseteq N(u_2) \subseteq \{w_1, w_2, u_1, u_3\}$, and there is at most one edge incident to u_3 and a vertex outside $\{w_1, w_2, u_2\}$. Moreover, the edges v_1v_2 and v_2v_3 are simple. Then remove w_1 and add it to the optimal solution.

Rule 12. Assume there is an induced path with endpoints u and v and with six internal vertices $v_1, \ldots, v_6$ such that for some vertices w_1, w_2 outside the path $N(\{v_1, \ldots, v_6\})\backslash\{u, v\} = \{w_1, w_2\}$. If $|N(w_1) \cap \{v_1, \ldots, v_6\}| \geq |N(w_2) \cap \{v_1, \ldots, v_6\}|$, then remove w_1 and add it to the optimal solution.

We remark that in the original paper [16], the algorithm runs in expected $O(n)$ time, and each rule can be detected in $O(1)$ time for each candidate with a hash table. However, we choose to use a balanced binary search tree instead of a hash table for a better practical performance.

The original algorithm works for the decision problem, so when applying it to the optimization problem, we need an additional step, called *lifting*, to convert a kernel solution to a feasible solution for the original graph. If a reduction rule does not introduce new vertices, then the lifting step for it will be trivial. Otherwise, we need to handle the vertices introduced by reduction steps, if they appear in the kernel solution. Among all the reduction rules, there are two rules, namely Rule 8 and Rule 9, that will introduce new vertices into the graph. For Rule 8, the following observation, whose proof is provided in the appendix, shows that the new vertex can be avoided.

Observation 5.1. *The new vertex introduced by Rule 8 can be replaced by a vertex from the original graph.*

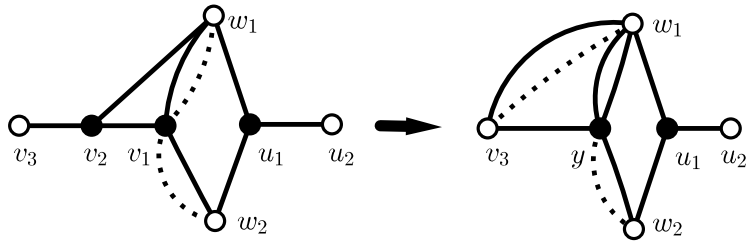

Figure 5.1: Reduction rule 8 replaces the left subgraph with the right subgraph. All dashed edges are optional, and all incident edges of black vertices are drawn as solid or dashed, while the white vertices may have other edges in the graph not drawn in the figure.

Proof. Rule 8 is illustrated in Figure 5.1, which will not modify the size of the optimal solution. We will show the new vertex y can be replaced by the vertex v_1 in the resulting graph after applying this rule. Let G and G' be the graph before and after applying Rule 8, and let S' be an FVS solution for G'. We only need to prove the following: if $y \in S'$, then $(S' \setminus \{y\}) \cup \{v_1\}$ is an FVS solution for G; otherwise S' is an FVS solution for G.

Assume $y \in S'$. Then we claim that any FVS solution for $G' \setminus \{y\}$ is also an FVS solution for $G \setminus \{v_1\}$. This is because we can obtain $G' \setminus \{y\}$ from $G \setminus \{v_1\}$ by applying Rule 3 (that is, if a vertex has degree two and its incident edges are not parallel edges, we remove this vertex and add an edge between its two neighbors) to vertex v_2, which will not affect the size of the optimal solution. Then we know $G \setminus (\{v_1\} \cup (S' \setminus \{y\}))$ is a forest, which implies $(S' \setminus \{y\}) \cup \{v_1\}$ is an FVS for G.

Now assume $y \notin S'$. Then we know $w_1 \in S'$ since there are parallel edges between y and w_1. We claim that any FVS solution for $G' \setminus \{w_1\}$ that does not contain y is also an FVS for $G \setminus \{w_1\}$. This is because after applying Rule 3 to v_2 in $G \setminus \{w_1\}$ the resulting graph is the same as $G' \setminus \{w_1\}$ except that the label for the vertex v_1 is y in $G' \setminus \{w_1\}$. Since $y \notin S'$, we know $S' \setminus \{w_1\}$ is an FVS for $G \setminus \{w_1\}$, which implies S' is an FVS for G. □

For Rule 9, we record in a structure the related vertices for each application of this rule, and store the structures in a list in the same order as we apply the rule. To lift the

solution to the original graph, we store all the vertices of the solution in a balanced search tree and then check the structures in the reverse order to see if the recorded vertices in a structure are also in the solution. If there are involved vertices in the solution, we modify the solution according to the reversed Rule 9. Since Rule 9 decreases the size of the graph, it can be applied at most $O(n)$ times. So there are at most $O(n)$ structures to check, each of which contains only constant number of vertices. To check if a vertex in the solution we need $O(\log n)$ time, so the total replacement can be done in $O(n \log n)$ time. Then the lifting step first adds back all vertices marked by the kernelization algorithm which can be done in linear time and then replaces the introduced new vertices with original vertices, which needs at most $O(n \log n)$ time. So lifting can be accomplished in $O(n \log n)$ time.

5.2.3 Polynomial-Time Approximation Scheme

In this section, we introduce a PTAS for FVS in planar graphs, using linear kernel and balanced separator.

5.2.3.1 Balanced Separator

A *separator* is a set of vertices, removing which will partition the graph into two parts. A separator is α-*balanced* if those two parts both have at most an α-fraction of the original vertex set. Lipton and Tarjan [95] first introduced a separator theorem for planar graphs, which says a planar graph with n vertices admits a $\frac{2}{3}$-balanced separator of size at most $2\sqrt{2n}$, and they gave a linear-time algorithm to compute such a balanced separator. This algorithm starts by computing a breadth-first search (BFS) tree for the graph, partitioning the vertices into levels. Then it tries to find the separator in three phases:

(P1) if there is any BFS level satisfying the requirements, then it is returned as a result;

(P2) if there is no such level, then the algorithm tries to find two BFS levels that can form a balanced separator together;

(P3) if both previous phases fail, then the algorithm identifies two BFS levels and con-

structs a fundamental cycle[1] to separate the subgraph between these two levels, such that the union of these two levels and the fundamental cycle form a balanced separator.

Though we followed a textbook version [91] of this separator algorithm, our implementation still guarantees the $2\sqrt{2n}$ bound for the size of the separator. We remark that we did not apply heuristics in our implementation for the separator algorithm. This is because we did not observe separator size improvement by some simple heuristics in the early stage of this work, and these heuristics may slow down the separator algorithm. Since our test graphs are large (up to 2 million vertices) and we will apply the algorithm recursively in our PTAS, these heuristics may slow down our PTAS even more.

5.2.3.2 PTAS for Planar FVS

Lipton and Tarjan [96] designed a PTAS for maximum independent set in planar graphs using their balanced separator, which depends on the fact that the input is already a constant-factor approximation to the maximum independent set and contains the optimal solution. Here we use the linear kernel as a proxy for the constant factor approximation that can be used to obtain a nearly optimal solution for FVS and relate the error parameter ϵ to the largest size of decomposed graphs instead of the size of the input graph as previous works [96, 32]. This idea can be used for other problems admitting linear kernels in graph families admitting balanced separators:

(1) Compute a linear kernel H for the original graph G, that is, $|V(H)|$ is at most $c_1|OPT(H)|$ for some constant c_1.

(2) Decompose the kernel H by recursively applying the separator algorithm and remove the separators until each resulting graph has at most r vertices for some constant r. The union of all the separators has at most $c_2|V(H)|/\sqrt{r} = \epsilon|OPT(H)|$ vertices for r chosen appropriately.

(3) Solve the problem optimally for all the resulting graphs.

[1] Given a spanning tree for a graph, a fundamental cycle consists of a non-tree edge and a path in the tree connecting the two endpoints of that edge.

(4) Let U_H be the union of all separators and all solutions of the resulting graphs. Lift U_H to a solution U_G for the original graph.

We can prove the following theorem.

Theorem 5.2. *There is an $O(n \log n)$ time PTAS for FVS in planar graphs.*

We need the following lemma to prove Theorem 5.2.

Lemma 5.3. *Given any FVS solution U_H for a linear kernel H obtained by Bonamy and Kowalik's algorithm, we can obtain an FVS solution U_G for the original graph G such that $|U_G| - |U_H| \leq |OPT(G)| - |OPT(H)|$.*

Proof. We can classify the reduction rules in Bonamy and Kowalik's kernelization algorithm into three types according to their effects on the optimal solution.

- Do not affect the optimal solution, such as removing vertices of degree one.
- Remove some vertices from the graph that must be in the optimal solution $OPT(G)$, such as removing vertices with self-loops.
- Add some new vertices into the graph without changing the size of the optimal solution, such as replacing a subgraph with another subgraph that has some new vertices.

When lifting the solution of H to that of G, we have their corresponding effects:

- Do not change the current solution.
- Add some new vertices to the current solution.
- Replace some vertices in the current solution with other vertices without increasing its size.

The rules of the third type will maintain the size of optimal solution unchanged, so we know $x = |OPT(G)| - |OPT(H)|$ is equal to the total number of vertices added by the second type of rules. Since the rules of the third type cannot increase the size of the solution during the lifting step, we know the size difference after applying a reverse rule of the third type is non-positive. That is, when we apply any reverse rule of the third

type to a solution U_0 and obtain a new solution U_1, we have $|U_1| - |U_0| \leq 0$. Let y be the sum of size differences over all third type rules applied during the lifting step. Then we know y is also non-positive. Note that the rules of the second type will only add vertices that is not in the kernel, so it contributes the same vertices to $OPT(G)$ as to any other solution U_G. Therefore, we know $|U_G| - |U_H| = x + y$, which implies the lemma. □

Now we are ready to prove Theorem 5.2.

Proof of Theorem 5.2. We first give a bound for the size of the final solution. For an integer $i > 0$, let H_i be a resulting graph after the decomposition step. Note that these graphs are vertex-disjoint. Let $OPT(H)$ and $OPT(G)$ be an optimal solution for H and G respectively. We know $OPT(H) \cap H_i$ is a solution for FVS in H_i since H_i is a subgraph of H and $OPT(H)$ is an optimal solution for H. So we have $|OPT(H_i)| \leq |OPT(H) \cap H_i|$ and then

$$\sum_{i>0} |OPT(H_i)| \leq \sum_{i>0} |OPT(H) \cap H_i| \leq |OPT(H)|. \tag{5.1}$$

Let S be the union of all separators found in the second step and n be the size of $V(H)$. Recall that parameter r is the largest size of H_i. Lipton and Tarjan [96] showed that the size of S is at most $c_2 n/\sqrt{r}$ for some constant c_2. If we choose $r = c_1^2 c_2^2/\epsilon^2$, then we have

$$|S| \leq c_2 n/\sqrt{r} = \epsilon n/c_1 \leq \epsilon |OPT(H)|. \tag{5.2}$$

Since U_H is the union of S and $OPT(H_i)$ for all $i > 0$, by combining (5.1) and (5.2), we have

$$|U_H| = |S| + \sum_{i>0} |OPT(H_i)| \leq (1+\epsilon)|OPT(H)|.$$

Since the kernelization algorithm can only decrease the size of the optimal solution, we have $|OPT(H)| \leq |OPT(G)|$. By Lemma 5.3, we have $|U_G| - |U_H| \leq |OPT(G)| - |OPT(H)|$. Then we obtain the size bound for U_G:

$$|U_G| \leq |U_H| + |OPT(G)| - |OPT(H)| \leq (1+\epsilon)|OPT(G)|.$$

Bonamy and Kowalik [16] showed that a linear kernel for planar FVS can be constructed in $O(n \log n)$ deterministic time. Each balanced separator can be computed in linear time by Lipton and Tarjan's algorithm [95], so we can finish the second step in $O(n \log n)$ time as done in [96]. The third step can be finished in $O(2^c n)$ time since each subgraph has size at most $c = r = O(1/\epsilon^2)$. And the last step can be done in $O(n \log n)$ time as we described in Section 5.2.2. So the total running time of this algorithm is $O(2^c n + n \log n)$ where $c = O(1/\epsilon^2)$. □

5.2.4 Heuristics

In this subsection, we introduce some heuristics that can help improve the quality of the FVS solution. We first provide two heuristics that improve the quality of our PTAS solutions. Then we introduce a hybrid algorithm that combines the greedy method of the 2-approximation algorithm and the reduction rules of the kernelization algorithm. Finally, we use local search to improve the solution from any algorithm. Our proposed Heuristic Approximation Scheme is a combination of the hybrid algorithm and the local search heuristic.

5.2.4.1 Heuristic Improvements to PTAS

The solution from our PTAS may not be a minimal one, so we use the post-processing step from the 2-approximation algorithm to convert the final solution of the PTAS to a minimal one. This involves iterating through the vertices in the solution and trying to remove redundant vertices from the solution while maintaining feasibility. In fact, we only need to iterate through the vertices in separators, since vertices in the optimal solutions of small graphs are needed for feasibility.

We additionally apply the kernelization algorithm right after we compute a separator in Step (2). Note that there is a decomposition tree corresponding to the decomposition step, where each node corresponds to a subgraph in the decomposition step. To apply the second heuristic, we need to record the whole decomposition tree with all the corresponding separators such that we can lift the solutions in the right order. For example, if we want to lift a solution for a subgraph G_w corresponding to some node w in the decomposition tree, we first need to lift all solutions for the subgraphs corresponding to

the children of node w in the decomposition tree.

5.2.4.2 Hybrid Algorithm

We notice the cost for detecting applicable reduction rules for a candidate vertex is relatively low ($O(\log n)$ time), and each of these rules can reduce either the size of the graph or the size of the optimal solution. It is beneficial, therefore, to apply them as much as possible. When there are no applicable reductions in the current graph, we can remove a vertex greedily just as the 2-approximation algorithm does, and this will change the current graph such that there may be applicable reductions again. Based on this idea, we propose a hybrid algorithm which interleaves the 2-approximation algorithm and the kernelization algorithm:

(i) Compute a temporary solution by repeating the following two steps until the graph is empty.

  (a) Apply reduction rules from the kernelization algorithm in order until there are no applicable rules.

  (b) Remove a vertex of highest degree from the graph and add it into the temporary solution.

(ii) Lift the temporary solution to a feasible solution for the original graph and then compute a minimal solution by removing redundant vertices from this solution.

The running time of the first step is similar to the running time of the kernelization algorithm since the greedy step can be seen as another "rule" added into the kernelization algorithm and it can be done in $O(\log n)$ time if we store the degree information in a binary search tree. So the first step runs in $O(n \log n)$ time. The lifting step needs $O(n \log n)$ time, and to compute a minimal solution we need $O(n \log n)$ time as done in Becker and Geiger's 2-approximation algorithm. So the total running time of our hybrid algorithm is $O(n \log n)$.

5.2.4.3 Local Search

In Chapter 4, we show that local search gives a PTAS for FVS in H-minor-free graphs. The algorithm is not practical, with running time $n^{O(1/\epsilon^2)}$. We modify the algorithm

here. Assume we are given a feasible FVS solution U for a planar graph G, and we would like to improve this solution. To achieve this goal, we propose a local search heuristic. Assume we have a fixed parameter tractable (FPT) algorithm *FA* for the FVS problem (either for planar graphs or for general graphs). Then our local search with size k consists of the following two steps.

(S1) Select a subset X of size k from U randomly.

(S2) Run the algorithm *FA* on graph $G \setminus (U \setminus X)$. Stop *FA* if a solution is not found in a reasonable amount of time. If *FA* finds a solution Y in graph $G \setminus (U \setminus X)$, then return $(U \setminus X) \cup Y$; otherwise return U.

The solution $(U \setminus X) \cup Y$ is a feasible solution for graph G since $(G \setminus (U \setminus X)) \setminus Y$ is a forest and $(G \setminus (U \setminus X)) \setminus Y = G \setminus ((U \setminus X) \cup Y)$.

5.3 Experiments

In this section, we evaluate the performance of the algorithms described in the last section. We implemented those algorithms in `C++` and the code is compiled with `g++` (version 4.8.5) on the `CentOS` (version 7.3.1611) operating system. Our PTAS implementation is built on Boyer's implementation[2] of Boyer and Myrvold's planar embedding algorithm [27]. In our experiments, we also use the implementation[3] of Iwata and Imanishi for FVS in general graphs, which is implemented in `java` and includes a linear-time kernel [75] and a branch-and-bound based FPT algorithm [76] for FVS in general graphs. The `java` version in our machine is 1.8.0 and our experiments were performed on a machine with Intel(R) Xeon(R) CPU (2.30GHz) running `CentOS` (version 7.3.1611) operating system.

To test the algorithms, we collect five different classes of graphs:

- **pace** are the planar graphs used in PACE (The Parameterized Algorithms and Computational Experiments Challenge) 2016 Track B: Feedback Vertex Set;
- **random** are random planar graphs generated by `LEDA` (version 6.4) [100];

[2] http://jgaa.info/accepted/2004/BoyerMyrvold2004.8.3/planarity.zip
[3] https://github.com/wata-orz/fvs

- **triangu** are triangulated random graphs generated by `LEDA`, whose outer faces are not triangulated;
- **grid** are rectangular grid graphs;
- The graphs **NY**, **BAY**, **COL**, **NW**, **CAL**, **FLA**, **LKS**, **NE**, **E** and **W** are road networks used in the 9th DIMACS Implementation Challenge-Shortest Paths [44]. We interpret each graph as a straight-line embedding and we add vertices whenever two edges intersect geometrically.

Since we are interested in the performance of the algorithms in large planar graphs, the synthetic graphs we generated have at least 450000 vertices. And the real network graphs have at least 260000 vertices. Although the **pace** graphs are smaller than that, we only use them to evaluate the 2-approximation algorithm since we can obtain optimal solutions of those small graphs. All the detailed experimental results, including solution sizes and running time, are also provided in the appendices.

5.3.1 The 2-Approximation Algorithm and Optimal Solution

To evaluate Becker and Geiger's 2-approximation algorithm [10], we compare its solution with the optimal solution on graphs up to size 2000000. The optimal solution is obtained by applying the kernelization algorithm first and then the FPT algorithm implemented by Iwata and Imanishi. For each test graph, we run Iwata and Imanishi's implementation for 30 seconds and stop it if it cannot terminate. Among all the test graphs, this method can solve 21 graphs of the 31 **pace** graphs, 9 graphs of the 10 **random** graphs. Although we cannot solve the large rectangular grid graphs by this method, we can apply Luccio's [97] lower bound for the optimal solution in rectangular grids here. We observe that the solutions obtained by 2-approximation algorithm are very close to the optimal solutions for these test graphs. For **pace** graphs, the 2-approximation algorithm can solve 14 graphs optimally and the difference between the two solutions is at most three. The approximation ratio over all these graphs is at most 1.143, and this ratio is at most 1.001 for the **grid** graphs and at most 1.006 for these **random** graphs.

5.3.2 The PTAS and 2-Approximation Algorithm

Recall that the third step of our PTAS is to solve the problem on all the decomposed graphs optimally, which needs an exact algorithm for the problem. The trivial exact algorithm that enumerates each possible vertex subset can only solve a graph of size about 25 in a few seconds, with which the PTAS may not be able to give competitive solutions for large graphs. So we apply the exact algorithm described in the last subsection, which combines the kernelization algorithm and the FPT algorithm implemented by Iwata and Imanishi. In the early stage of this work, this exact algorithm did not find a solution for a **pace** graph of size 66 in 30 seconds. So we first set as 60 the largest size r of the decomposed graphs and let the FPT algorithm run for at most 15 seconds. In this setting, all the decomposed graphs can be solved optimally in our experiments and we can evaluate the heuristics proposed for our PTAS. To do that, we compare three variants of our PTAS:

- the **vanilla** variant is the vanilla version of our PTAS, for which no heuristic is applied;
- the **minimal** variant applies the post processing heuristic to our PTAS, which will remove redundant vertices in separators;
- the **optimized** variant applies both heuristics to our PTAS, which will apply kernelization algorithm whenever each separator is computed and removed during the decomposition step and always returns a minimal final solution.

The result is illustrated in Figure 5.2, where the solution size is normalized by the 2-approximation algorithm solutions. We observe that for the road network graphs, **random** graphs and **triangu** graphs, the **optimized** variant provides the best solutions among the three algorithms, which implies the two heuristics both help improve the solutions. However, for the **grid** graphs, the **minimal** variant gives the best results, which means the kernelization algorithm is not very helpful. We think this is because a large rectangular grid graph itself is already a $4k$-kernel by the lower bound for the optimal solution [97], and the kernelization algorithm can only remove four vertices from such graphs. For the **random** graphs (not pictured in Figure 5.2), the improvement from the two heuristics is mild, but the obtained solutions are already very close to the

optimal solutions, that is the differences are less than 60 vertices when the solutions have more than 190000 vertices. We also find that the post processing heuristic will not affect the running time by much, while the kernelization heuristic can increase the running time at most by a factor of 5.

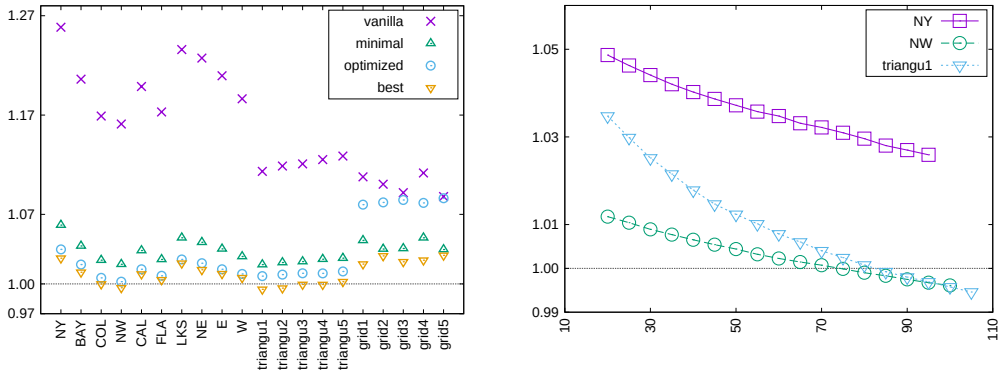

Figure 5.2: Results of our PTAS implementation. The Y axis represents the solution size normalized by the solutions of 2-approximation algorithm. Left: solutions of the three variants of our PTAS with $r = 60$ and the best solutions of our PTAS with largest possible values of r. Right: effect of parameter r on the solution sizes where X axis represents the value of r.

Recall that the largest size r of the decomposed graphs is the only parameter in our PTAS. Now we analyze the effect of this parameter on the performance of our PTAS. Since the **optimized** variant works best for most of the test graphs, we focus on its performance affected by parameter r. We start with $r = 20$ and each time increase it by 5 until our implementation cannot compute a feasible solution, which is caused by the fact that the FPT implementation cannot solve some decomposed graphs of size r in the given time. The result is shown in Figure 5.2. We can see in the figure that our implementation can solve the instance for relatively large value of r, and the solution is improved when the parameter r increases. This is because when r is bigger, the total size of the separators, which is the error part of our PTAS, is smaller. Since we set a time limit for the FPT implementation, the effect of parameter r on the running time is not significant, although we observed a mildly increasing tendency on the running time when r increases.

Now we can compare our PTAS with the 2-approximation algorithm. Based on the above results, we know the largest value of r may be different for different graphs. So to get the best result, we start with $r = 60$ and each time increase it by 5 until we cannot find feasible solution. Since different variants work best for different graph classes, we choose the best variant for each graph class, that is, we apply the **minimal** variant for **grid** graphs, and the **optimized** variant for other graphs. The largest value r we find varies from 80 to 125 for our test graphs. The final result is plotted in Figure 5.2 marked as **best**. We can see that although our PTAS implementation is competitive with the 2-approximation algorithm on all graphs, it cannot outperform the 2-approximation solution on most road network graphs. The reason is that the subgraphs for which we can solve the problem optimally are still not large enough, which results in a fact that the separator fraction in the solution is large. On the **grid** graphs, the 2-approximation algorithm outperforms our PTAS by about 2 percent. On four of the five **triangu** graphs, our PTAS is slightly better than the 2-approximation algorithm, where the difference for each graph is less than 1 percent. While on the **random** graphs the improvement is mild, the final solutions are very close to the optimal. We also find the running time of our PTAS is much longer than the 2-approximation algorithm. Specifically, the running time of the 2-approximation algorithm varies from a few seconds to a few minutes for our test graphs, while our PTAS needs a few hours to finish for the large graphs, where most time is spent on running the FPT algorithm.

5.3.3 Heuristic Approximation Scheme

We evaluate the performance of the following two algorithms and compare them with the 2-approximation algorithm (**2approx**).

- The **hybrid** algorithm combines the kernelization reduction rules and the greedy step of the **2approx** to find a solution, and then lifts it to a minimal feasible solution for the original graph.
- **HAS** first computes a solution by the **hybrid** algorithm and then applies the local search heuristic with an FPT algorithm to improve the solution quality.

In our following experiments, the FPT algorithm used in the local search consists of two parts: Bonamy and Kowalik's linear kernelization algorithm and Iwata and Iman-

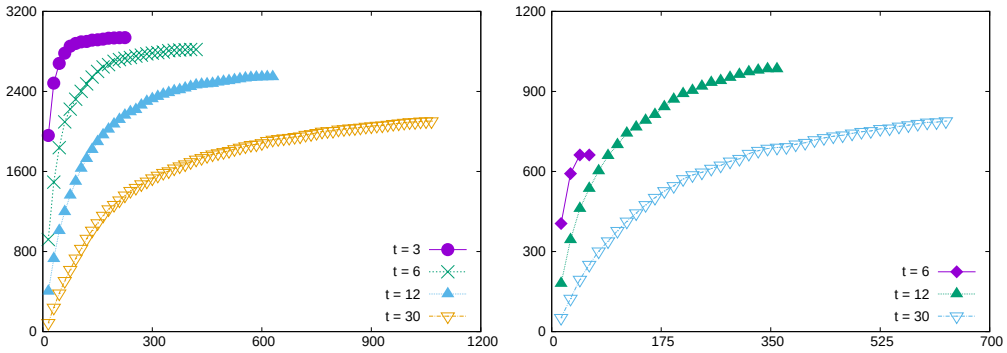

Figure 5.3: Local search improvement affected by different values of t. The X axis represents the number of local search iterations and the Y axis represents the number of vertices the initial solution is improved by. Left: results for graph **triangu1**. Right: results for graph **COL**.

ishi's implementation, which is run for at most 15 seconds.

Before showing the final comparison of these algorithms, we first need to optimize their parameters. For the **hybrid** algorithm, there is a potential parameter: how often should the kernelization reduction rules be applied? We evaluated different values from 1 to 100 for this frequency parameter to understand its effect on the solution size. We find that for the road network graphs the difference is at most 0.2 percent of the solution size and for the other test graphs the difference is less than 0.01 percent. So we only consider the road network graphs to optimize this parameter. Although we did not find an optimal value for this parameter that could always give smallest solution, we can avoid some values that always give larger solutions. For this goal, we choose the frequency as 41 in our later experiments, that means we apply the reduction rules after removing 41 vertices greedily.

For the local search heuristic in **HAS**, the only parameter we need to optimize is the size k of the random set. We choose this parameter as a $1/t$ fraction of the solution size for an integer t, so that we only need to optimize the value of t. We evaluated four different values (3, 6, 12, 30) for t and terminate the process when there is no improvement for twelve consecutive search rounds. Figure 5.3 shows the results for two graphs. We can see in the figures that the total improvement is not always monotone with t in this range. For the synthetic graphs like **triangu1**, the total improvement for $t = 3$ is the largest, while for most road network graphs, the largest total improvement is

obtained for $t = 12$. The former phenomenon can be explained by the result in Chapter 4, which shows local search is a PTAS for FVS in minor-free graphs so the solution will be better when the size of the random set replaced is larger. However, this will not hold when the FPT algorithm cannot solve the problem in a reasonable amount of time for large random set, which corresponds to the smaller t, and then the improvement will be limited. This could explain the results for the road network graphs. Moreover, while local search with larger value of k tends to give better improvements, the number of local search iterations is bigger for smaller value of k, which implies the improvement is consecutive and stable. Based on these observations, we will iterate through the values of t in an increasing order to maximize the improvement in our later experiments, that is, we will increase the value of t by 3 when there are six consecutive search rounds that find no improvement. The range of t is still from 3 to 30.

To better understand the local search heuristic, we illustrate the improvement fraction and running time fraction distributed on different values of t in Figure 5.4, where the running time is represented by the number of local search iterations. We observe that for the synthetic graphs, represented by **triangu5** in the figure, the total improvement comes from the search with $t = 3$, while for the road network graphs, the improvement distributes on different small values of t. The distribution of the running time has the similar tendency as the improvement.

We notice that the local search heuristic in **HAS** can give us a "PTAS behavior", that is we can obtain better solutions if we spend more time doing so. So one natural question to ask is how long can that improvement process last? To answer this, we tried looping the values of t in the range [3, 30], and found that only one iteration over the range is enough, and that additional iterations only give minor improvement.

Now we can set the parameters and compare these algorithms. For each graph, we run **HAS** five times, each of which is run with a different random seed, and compute the average of the five solutions. Figure 5.5 illustrates the results where each solution size is normalized by the **2approx** solution. We can see in the figure that for most graphs, **hybrid** finds better solutions than **2approx**, and **HAS** gives the best solutions. The improvement of **HAS** for all road network graphs is more than 5 percent, which could be over 30000 vertices for large graphs like **W**. For all **triangu** graphs the improvement of **HAS** is at least 3 percent. For **random** graphs the improvement is not very significant, this is because the final solutions are already very close to the optimal solutions. Since

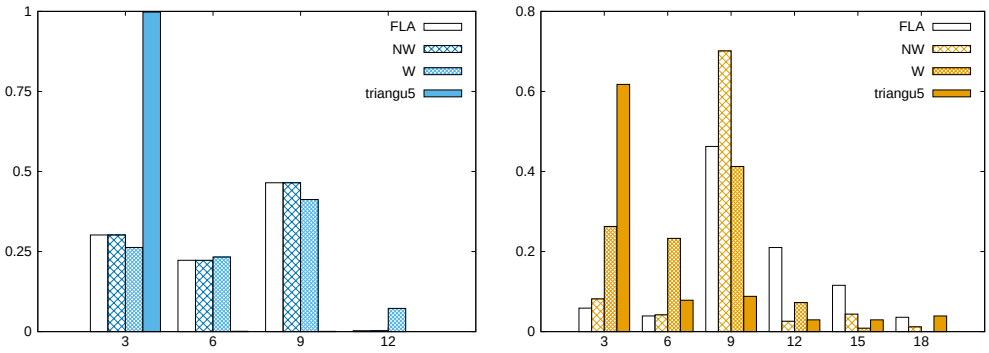

Figure 5.4: Local search improvement and number of iterations distributed on different values of t. The X axis is the value of t. Left: the Y axis represents the fraction of total improvement. Right: the Y axis represents the fraction of the total iterations.

the **2approx** works very well on the **grid** graphs as shown before, it is hard to outperform it on these graphs though our heuristics is able to find competitive solutions on these graphs.

We report the running time and the solution size of **hybrid** and **HAS** on some graphs in Table 5.1. Additionally, we show the running time in second per improved vertex for these graphs. We notice the running time of **HAS** is relatively long compared with **hybrid**. For example, **hybrid** can terminate in a few minutes for graph **W**, but **HAS** needs more than 35 hours to terminate for this graph. However, the time spend per improved vertex is relatively short for these graphs. And this also reflects that the longer running time is the result of larger improvement, which is the "PTAS behavior" that people can utilize to balance the running time and the solution quality by setting a proper number of local search iterations.

5.4 Detailed Experimental Results

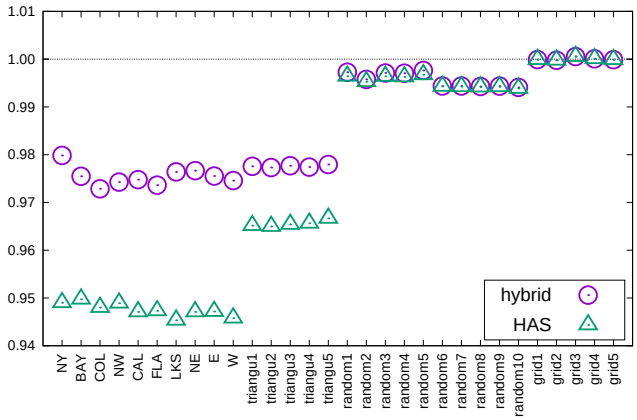

Figure 5.5: Results of **hybrid** and **HAS**. The Y axis represents the solution sizes normalized by the solutions of the 2-approximation algorithms. The solution of **HAS** is the average over the five runs with different random seeds.

Table 5.1: Results on the running time and solution size of **hybrid** and **HAS** on some graphs. The running time of HAS is averaged over five runs of HAS with different random seeds.

graph	vertices	edges	hybrid runtime	hybrid solution	HAS runtime	HAS solution	time per improved vertex
NY	264953	366250	1m 28s	40868	1h 6m 30s	39582	0.330s
BAY	322694	400233	1m 9s	33372	54m 53s	32492	0.273s
CAL	1898842	2331204	7m 32s	187610	6h 10m 28s	182282	0.245s
E	3608115	4372928	14m 23s	333997	16h 49m 22s	324286	0.163s
W	6286759	7608797	23m 46s	579822	35h 41m 39s	562687	0.135s
triangu1	600000	1799963	1m 46s	228711	1h 32m 37s	225802	0.534s
triangu2	800000	2399961	1m 58s	304288	2h 11m 39s	300443	0.494s
random1	699970	2000000	1m 54s	193071	6m 54s	192908	0.543s
random4	1999760	5600000	5m 12s	538958	21m 2s	538533	0.447s
random5	2199977	6400000	5m 58s	618238	30m 15s	617674	0.387s

Table 5.2: Compare the 2-approximation algorithm solutions and the optimal solutions. The italic numbers are lower bounds from [97].

graph	vertices	edges	2approx	opt	approx ratio
pace1	49	107	15	15	1.0
pace2	118	179	18	18	1.0
pace3	62	78	7	7	1.0
pace4	118	179	18	18	1.0
pace5	59	104	16	16	1.0
pace6	70	85	8	8	1.0
pace7	48	64	6	6	1.0
pace8	74	92	8	8	1.0
pace9	67	83	9	8	1.125
pace10	90	103	8	7	1.143
pace11	55	81	12	11	1.091
pace12	110	147	16	15	1.067
pace13	66	127	24	21	1.143
pace14	153	177	12	12	1.0
pace15	149	193	16	16	1.0
pace16	73	95	10	10	1.0
pace17	45	64	8	8	1.0
pace18	145	186	17	16	1.063
pace19	158	189	15	15	1.0
pace20	61	78	8	7	1.143
pace21	4960	9462	898	898	1.0
grid1	450000	1796400	149527	*149401*	1.001
grid2	600000	2396800	199552	*199468*	1.001
grid3	1000000	3996000	332669	*332668*	1.0
grid4	1680000	6714800	559252	*559134*	1.001
grid5	2100000	8394200	699285	*699034*	1.001
random1	699970	2000000	193601	192902	1.004
random2	1197582	3000000	285550	284195	1.005
random3	1399947	4000000	387216	385813	1.004
random4	1999760	5600000	540550	538524	1.004
random6	873280	1200000	87300	86796	1.006
random7	1061980	1500000	111973	111324	1.006
random8	1227072	1700000	125374	124635	1.006
random9	1520478	2200000	167709	166737	1.006
random10	2050946	3300000	270981	269315	1.006

Table 5.3: Summary of the solution sizes of different PTAS variants and the 2-approximation algorithm. The results for the three variants ("vanilla", "minimal" and "optimized") of the PTAS are computed for $r = 60$. The results for "best" are computed for different values of r by different variants of PTAS depending on the graphs. For **grid** graphs, "minimal" variant is applied for "best" results, and for other graphs "optimized" variant is applied.

graph	vertices	edges	vanilla	minimal	optimized	best	2approx
NY	264953	366250	52487	44178	43159	42790	**41709**
BAY	322694	400233	41264	35518	34877	34612	**34211**
COL	437294	524437	45831	40142	39445	**39201**	39205
NW	1214463	1423402	111149	97627	95951	**95362**	95735
FLA	1074167	1351411	147632	128941	126869	126361	**125841**
CAL	1898842	2331204	230712	198924	195303	194379	**192465**
LKS	2763392	3407840	342592	290065	284034	282972	**277196**
NE	1528387	1941840	222569	188936	185115	183916	**181336**
E	3608115	4372928	414132	354444	347418	345891	**342371**
W	6286759	7608797	705851	611351	600898	598667	**594943**
triangu1	600000	1799963	260475	238502	235796	**232695**	233958
triangu2	800000	2399961	348305	318058	314257	**310034**	311349
triangu3	1000000	2999955	434793	396644	392045	**387690**	387908
triangu4	1200000	3599966	521141	474683	468072	**462865**	463157
triangu5	1400000	4199957	606425	551194	544006	538496	**537270**
random1	699970	2000000	192970	192925	192925	**192907**	193601
random2	1197582	3000000	**284195**	**284195**	**284195**	**284195**	285550
random3	1399947	4000000	385928	385837	385866	**385818**	387216
random4	1999760	5600000	538620	538556	538573	**538529**	540550
random5	2199977	6400000	617940	617761	617768	**617681**	619710
random6	873280	1200000	**86796**	**86796**	**86796**	**86796**	87300
random7	1061980	1500000	**111324**	**111324**	**111324**	**111324**	111973
random8	1227072	1700000	**124635**	**124635**	**124635**	**124635**	125374
random9	1520478	2200000	**166737**	**166737**	**166737**	**166737**	167709
random10	2050946	3300000	**269315**	**269315**	**269315**	**269315**	270981
grid1	450000	1796400	165650	156071	161462	152515	**149527**
grid2	600000	2396800	219589	206563	215920	205189	**199552**
grid3	1000000	3996000	363242	344549	360783	340078	**332669**
grid4	1680000	6714800	621736	585211	604861	572608	**559252**
grid5	2100000	8394200	760797	723507	759568	719410	**699285**

Table 5.4: Summary of the running time of different variants of our PTAS. The parameter r is set as 60 for all PTAS variants.

graph	vertices	edges	vanilla	minimal	optimized	2approx
NY	264953	366250	5m 45s	5m 39s	13m 46s	9s
BAY	322694	400233	4m 25s	4m 17s	10m	9s
COL	437294	524437	5m 8s	4m 56s	10m 46s	13s
NW	1214463	1423402	13m 21s	12m 49s	26m 2s	37s
FLA	1074167	1351411	16m 38s	16m 37s	34m 18s	33s
CAL	1898842	2331204	35m 36s	33m 2s	1h 12m 20s	58s
LKS	2763392	3407840	1h 26s	1h 1m 46s	2h 48m 44s	1m 31s
NE	1528387	1941840	41m 13s	43m 32s	1h 53m 1s	48s
E	3608115	4372928	1h 39m 17s	1h 40m 7s	3h 34m 25s	1m 52s
W	6286759	7608797	3h 44m 38s	3h 43m 26s	7h 39m 21s	3m 15s
triangu1	600000	1799963	1h 6m 23s	55m 39s	1h 56m 32s	55s
triangu2	800000	2399961	2h 28m 7s	1h 57m 15s	4h 15m 24s	1m 7s
triangu3	1000000	2999955	4h 17m 57s	3h 41m 50s	6h 47m 30s	1m 22s
triangu4	1200000	3599966	6h 42m 30s	5h 48m 12s	9h 59m 24s	1m 39s
triangu5	1400000	4199957	9h 20m 14s	8h 14m 59s	14h 16m 45s	1m 58s
random1	699970	2000000	50m 40s	53m 18s	1h 22m 10s	1m 20s
random2	1197582	3000000	1h 19m 18s	1h 22m 27s	1h 37m 57s	1m 58s
random3	1399947	4000000	2h 42m 55s	2h 25m 9s	2h 5m 3s	2m 34s
random4	1999760	5600000	4h 1m 48s	4h 30m 10s	3h 32m 19s	3m 53s
random5	2199977	6400000	4h 57m 53s	5h 38m 16s	4h 39m 10s	4m 24s
random6	873280	1200000	14m 9s	16m 8s	13m 8s	58s
random7	1061980	1500000	19m 54s	22m 33s	18m 10s	1m 5s
random8	1227072	1700000	21m 44s	24m 29s	19m 41s	1m 14s
random9	1520478	2200000	30m 52s	35m 13s	28m 9s	1m 33s
random10	2050946	3300000	1h 1m	1h 10m 29s	55m 41s	2m 20s
grid1	450000	1796400	25m 41s	24m 28s	1h 13m 18s	24s
grid2	600000	2396800	35m 24s	34m 48s	1h 58m 24s	29s
grid3	1000000	3996000	1h 11m 9s	1h 6m 49s	4h 56m 29s	48s
grid4	1680000	6714800	2h 56m 47s	2h 18m 7s	10h 7m 46s	1m 25s
grid5	2100000	8394200	3h 48m 16s	3h 0m 23s	15h 1m 48s	1m 44s

Table 5.5: Summary of the solution sizes of different heuristic algorithms. HAS (avg) is the solution size averaged over five runs of HAS with different random seeds. HAS (min) is the minimum size among the five values. The improv value is computed as 1-HAS(avg)/2approx.

graph	vertices	edges	2approx	hybrid	HAS (avg)	HAS (min)	improv
NY	264953	366250	41709	40868	39582	**39537**	0.051
BAY	322694	400233	34211	33372	32492	**32446**	0.050
COL	437294	524437	39205	38141	37167	**37141**	0.052
NW	1214463	1423402	95735	93272	90844	**90816**	0.051
FLA	1074167	1351411	125841	122521	119211	**119163**	0.053
CAL	1898842	2331204	192465	187610	182282	**182191**	0.053
LKS	2763392	3407840	277196	270651	262041	**261917**	0.055
NE	1528387	1941840	181336	177101	171742	**171619**	0.053
E	3608115	4372928	342371	333997	324286	**324056**	0.053
W	6286759	7608797	594943	579822	562687	**562606**	0.054
triangu1	600000	1799963	233958	228711	225802	**225791**	0.035
triangu2	800000	2399961	311349	304288	300443	**300430**	0.035
triangu3	1000000	2999955	387908	379251	374485	**374468**	0.035
triangu4	1200000	3599966	463157	452692	447236	**447206**	0.034
triangu5	1400000	4199957	537270	525416	519361	**519229**	0.033
random1	699970	2000000	193601	193071	192908	**192904**	0.004
random2	1197582	3000000	285550	284340	284198	**284195**	0.005
random3	1399947	4000000	387216	386095	385818	**385815**	0.004
random4	1999760	5600000	540550	538958	538533	**538526**	0.004
random5	2199977	6400000	619710	618238	617674	**617671**	0.003
random6	873280	1200000	87300	86809	86802	**86796**	0.006
random7	1061980	1500000	111973	111343	111326	**111324**	0.006
random8	1227072	1700000	125374	124655	124641	**124636**	0.006
random9	1520478	2200000	167709	166762	166739	**166739**	0.006
random10	2050946	3300000	270981	269378	269319	**269318**	0.006
grid1	450000	1796400	149527	**149511**	**149511**	**149511**	0.001
grid2	600000	2396800	199552	**199506**	**199506**	**199506**	0.001
grid3	1000000	3996000	**332669**	332847	332847	332847	-0.001
grid4	1680000	6714800	**559252**	559298	559298	559298	-0.001
grid5	2100000	8394200	699285	**699195**	**699195**	**699195**	0.001

Table 5.6: Summary of the running time of different heuristic algorithms. HAS (avg) is the running time averaged over five runs of HAS with different random seeds. HAS (min) is the running time for the HAS run with minimum solution size.

graph	vertices	edges	2approx	hybrid	HAS (avg)	HAS (min)
NY	264953	366250	9s	1m 28s	1h 6m 30s	1h 12m 33s
BAY	322694	400233	9s	1m 9s	54m 53s	1h 8m 2s
COL	437294	524437	13s	1m 19s	51m 22s	1h 10m 49s
NW	1214463	1423402	37s	3m 28s	2h 18m 40s	2h 24m 32s
FLA	1074167	1351411	33s	4m 12s	3h 57m 25s	4h 7m 32s
CAL	1898842	2331204	58s	7m 32s	6h 10m 28s	6h 12m 30s
LKS	2763392	3407840	1m 31s	11m 38s	13h 21m 22s	14h 35m 33s
NE	1528387	1941840	48s	7m 10s	5h 52m 16s	6h 49m 51s
E	3608115	4372928	1m 52s	14m 23s	16h 49m 22s	16h 27m 17s
W	6286759	7608797	3m 15s	23m 46s	35h 41m 39s	38h 10m 5s
triangu1	600000	1799963	55s	1m 46s	1h 32m 37s	1h 39m 7s
triangu2	800000	2399961	1m 7s	1m 58s	2h 11m 39s	2h 35m
triangu3	1000000	2999955	1m 22s	2m 20s	2h 33m 5s	2h 42m 50s
triangu4	1200000	3599966	1m 39s	2m 55s	3h 15m 9s	3h 54m 12s
triangu5	1400000	4199957	1m 58s	3m 30s	4h 38m	3h 23m 16s
random1	699970	2000000	1m 20s	1m 54s	6m 54s	7m 31s
random2	1197582	3000000	1m 58s	2m 26s	7m 47s	8m 57s
random3	1399947	4000000	2m 34s	3m 31s	15m 5s	15m 24s
random4	1999760	5600000	3m 53s	5m 12s	21m 2s	21m 53s
random5	2199977	6400000	4m 24s	5m 58s	30m 15s	32m 43s
random6	873280	1200000	58s	1m 17s	1m 45s	2m 12s
random7	1061980	1500000	1m 5s	1m 31s	2m 4s	1m 55s
random8	1227072	1700000	1m 14s	1m 42s	2m 25s	2m 13s
random9	1520478	2200000	1m 33s	2m 17s	3m 17s	3m 18s
random10	2050946	3300000	2m 20s	3m 29s	5m 10s	5m 56s
grid1	450000	1796400	24s	3m 49s	1h 1m 16s	1h 2m 26s
grid2	600000	2396800	29s	5m 19s	1h 18m 18s	1h 18m 54s
grid3	1000000	3996000	48s	8m 48s	1h 55m 10s	1h 56m 43s
grid4	1680000	6714800	1m 25s	14m 47s	3h 1m 44s	3h 4m 5s
grid5	2100000	8394200	1m 44s	17m 44s	3h 52m 47s	3h 42m 23s

Chapter 6: Conclusion

Approximation algorithms for NP-hard problems in graphs have been studied for decades. Among those, a polynomial-time approximation scheme (PTAS) is the best polynomial-time approximation algorithm that we can hope for such problems. This is because when given any fixed approximation ratio α, a PTAS can find an α-approximation solution in polynomial time. Unfortunately, there are some problems that do not admit PTASes in general graphs unless $P = NP$, such as the maximum independent set problem [105] and the traveling salesperson problem (TSP) [81]. However, we can obtain PTASes for some of these problems when restricted to planar graphs, including the above two problems. In the last decade, we see many PTAS results in planar graphs appear with some exciting algorithmic techniques.

In this thesis, we summarized the commonly used approaches to design PTAS for NP-hard problems in planar graphs. Based on these approaches, we presented the first PTAS results for some NP-hard problems in planar graphs that require strong connectivity, including relaxed minimum-weight subset three-edge-connected subgraph problem, minimum three-edge-connected spanning subgraph problem and minimum three-vertex-connected spanning subgraph problem. For the first problem, our PTAS runs in $O(n \log n)$ time, and for the other two problems, our PTAS runs in $O(n)$ time. We also gave two different PTASes for minimum feedback vertex set problem (FVS): one for minor-free graphs that runs in $O(n^{1/\epsilon^2})$ time based on local search, and the other for planar graphs that runs in $O(n \log n)$ time based on linear kernel and balanced separators.

To understand the performance of PTASes in large planar graphs, we implemented our $O(n \log n)$-time PTAS for FVS and compare it with Becker and Geiger's 2-approximation algorithm [10]. Our results show that our PTAS is competitive in terms of solution quality with the 2-approximation algorithm in most large planar graphs. We then presented a heuristic algorithm for FVS, which is based on the reduction rules in the linear kernelization algorithm and the idea in our local search PTAS. This heuristic algorithm can provide a trade-off between the running time and the solution quality, just like

a PTAS. We find that our heuristic algorithm can produce better solutions than the 2-approximation on both synthetic planar graphs and real-world planar graphs.

6.1 Frontiers

Many open problems remain, which relate to or could be built on the works of this thesis. In the following, we describe some of the open problems, that we think are most exciting, in this field.

PTAS for nonuniform facility location problem Uncapacitated facility location problem is a widely studied model in the discrete facility location problem. In this model, given a finite metric space d, a set of points C and a cost function f, we want to find a set S of points that minimizes the total cost $\sum_{i\in S} f(i) + \sum_{j\in C} \min_{i\in S} d(i,j)$. The problem is *uniform* if every point in C has the same cost, that is $f(i) = f(j)$ for any i and j in C, otherwise *nonuniform*. Both of uniform facility location (UFL) and nonuniform facility location (NFL) are NP-hard in planar graphs [69, 79]. Cohen-Addad, Klein and Mathieu [35] gave the first PTAS for uniform facility location (UFL) in minor-free graphs by local search. *Can we obtain a PTAS for NFL in planar graphs?* Since the local search method is hard to generalize to vertex weight, it seems we need a different method for NFL PTAS. The only clustering problem that admits (bicriteria) PTAS in vertex-weighted planar graphs is the k-center problem. However, it is not clear how to apply those techniques to NFL.

PTAS for k-minimum spanning tree problem (k-MST) The k-MST asks for a tree subgraph of minimum weight that spans any subset of exactly k vertices. This problem admits PTAS when the input consists of points in the Euclidean plane [3]. For many problems, including TSP, Steiner tree and Steiner forest, PTAS results are first developed for the Euclidean plane and then developed for planar graphs. This is because the techniques for the latter are often inspired by the techniques for the former. So it is natural to ask: *is there a PTAS for k-MST in planar graphs?*

PTAS for minimum-weight subset two-edge-connected subgraph problem (subset 2-EC) Subset 2-EC asks for a minimum-weight subgraph that maintains two-

edge-connectivity for a set of given vertices. This problem can be seen as a stronger version of Steiner tree problem, since it requires stronger connectivity for a vertex subset. Borradaile and Klein [22] gave an EPTAS for a relaxed version of this problem in planar graphs, where each edge can be used multiple times. And the only planar-specific algorithm for non-spanning, *strict* edge-connectivity is a PTAS for the following problem: given a subset R of edges, find a minimum weight subset S of edges, such that for every edge in R, its endpoints are two-edge-connected in $R \cup S$ [89]; otherwise, the best known results for the strict version of subset 2-EC is the constant-factor approximations known for general graphs. *Can we obtain a PTAS for subset 2-EC in planar graphs?* We note that all the PTASes mentioned above are based on the spanner framework, and the challenge part to apply this framework to subset 2-EC is to construct a spanner: it is still not clear how to maintain the two-edge-connectivity and the weight bound of the spanner. We think the PTAS for the spanning version of this problem [12] could provide some inspiration: it does not construct a complete spanner, but a partial spanner, which does not contain a nearly optimal solution but has some properties that can be used to construct a nearly optimal solution by a dynamic programming.

EPTAS for minimum-weight connected dominating set problem (CDS) CDS asks for a minimum-weight vertex set in a vertex-weighted graph such that it induces a connected subgraph and every vertex in the graph is either in the set or adjacent to some vertex in the set. This problem is NP-hard in planar graphs [62] and is shown to admit PTAS by separator techniques in planar graphs [34]. But this PTAS is not efficient. *Can we obtain an EPTAS for CDS in planar graphs?* One potential technique for this problem is the shifting technique. This is because the shifting technique can be seen as an alternative way to find a set of separators efficiently. And we have seen this kind of improvement appears for TSP and maximum independent set problem in planar graphs. However, it is not clear how to bound the error in the shifting technique.

PTAS Engineering In Chapter 5, we designed and implemented an $O(n \log n)$ time PTAS for the minimum feedback vertex set problem (FVS) in planar graphs based on linear kernel and balanced separators. In fact, we can also obtain an $O(n \log n)$ time PTAS by applying the shifting techniques instead of balanced separators. To implement this PTAS, we will need to compute a branch decomposition and apply dynamic pro-

gramming for the resulting graph. This method is used in previous PTAS engineering works [115, 9]. *Does this PTAS outperform the 2-approximation algorithm and our algorithms?* Further, in our experimental work, we only focus on FVS. However, we believe the ideas behind our heuristics can also work for other problems. For example, the idea of combining reduction rules and another approximation algorithm can also be applied to other problems like dominating set and vertex cover. Similarly, our local search heuristic can also be generalized to new problems if there are FPT algorithms for them. So we think it will be interesting to see their performance on different problems.

Bibliography

[1] Lyudmil Aleksandrov, Hristo Djidjev, Hua Guo, and Anil Maheshwari. Partitioning planar graphs with costs and weights. *Journal of Experimental Algorithmics (JEA)*, 11:1–5, 2007.

[2] Noga Alon, Paul Seymour, and Robin Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society*, 3(4):801–808, 1990.

[3] Sanjeev Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[4] Sanjeev Arora, Michelangelo Grigni, David R Karger, Philip N Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 33–41, 1998.

[5] Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999.

[6] Brenda S Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

[7] MohammadHossein Bateni, Erik D Demaine, MohammadTaghi Hajiaghayi, and Dániel Marx. A PTAS for planar group Steiner tree via spanner bootstrapping and prize collecting. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 570–583. ACM, 2016.

[8] MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. *J. ACM*, 58(5):21, 2011.

[9] Amariah Becker, Eli Fox-Epstein, Philip N Klein, and David Meierfrankenfeld. Engineering an approximation scheme for traveling salesman in planar graphs. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 75, 2017.

[10] Ann Becker and Dan Geiger. Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83(1):167–188, 1996.

[11] André Berger, Artur Czumaj, Michelangelo Grigni, and Hairong Zhao. Approximation schemes for minimum 2-connected spanning subgraphs in weighted planar graphs. In *Proceedings of the 13th European Symposium on Algorithms*, volume 3669 of *Lecture Notes in Computer Science*, pages 472–483, 2005.

[12] André Berger and Michelangelo Grigni. Minimum weight 2-edge-connected spanning subgraphs in planar graphs. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming*, volume 4596 of *Lecture Notes in Computer Science*, pages 90–101, 2007.

[13] Norman L. Biggs, E. Keith Lloyd, and Robin J. Wilson. *Graph Theory 1736-1936*. Oxford University Press, 1986.

[14] Punyashloka Biswal, James R Lee, and Satish Rao. Eigenvalue bounds, spectral partitioning, and metrical deformations via flows. *Journal of the ACM*, 57(3):13, 2010.

[15] Hans L Bodlaender, Fedor V Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M Thilikos. (Meta) kernelization. *Journal of the ACM (JACM)*, 63(5):44, 2016.

[16] Marthe Bonamy and Łukasz Kowalik. A $13k$-kernel for planar feedback vertex set via region decomposition. *Theoretical Computer Science*, 645:25–40, 2016.

[17] Richard B Borie, R Gary Parker, and Craig A Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(1-6):555–581, 1992.

[18] Glencora Borradaile, Erik D Demaine, and Siamak Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68(2):287–311, 2014.

[19] Glencora Borradaile, Claire Kenyon-Mathieu, and Philip Klein. A polynomial-time approximation scheme for Steiner tree in planar graphs. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 7, pages 1285–1294, 2007.

[20] Glencora Borradaile and Philip Klein. The two-edge connectivity survivable network problem in planar graphs. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, pages 485–501, 2008.

[21] Glencora Borradaile and Philip Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *Journal of the ACM*, 56(2):1–30, 2009.

[22] Glencora Borradaile and Philip Klein. The two-edge connectivity survivable-network design problem in planar graphs. *ACM Transactions on Algorithms*, 12(3):30, 2016.

[23] Glencora Borradaile, Philip Klein, and Claire Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms*, 5(3):1–31, 2009.

[24] Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. Minor-free graphs have light spanners. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 767–778, 2017.

[25] Glencora Borradaile, Hung Le, and Baigong Zheng. Designing practical PTASes for minimum feedback vertex set in planar graphs. *CoRR*, abs/1804.07869, 2018.

[26] Glencora Borradaile and Baigong Zheng. A PTAS for three-edge-connected survivable network design in planar graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 3:1–3:13, 2017.

[27] John M Boyer and Wendy J Myrvold. On the cutting edge: Simplified $O(n)$ planarity by edge addition. *J. Graph Algorithms Appl.*, 8(2):241–273, 2004.

[28] Lorenzo Brunetta, Francesco Maffioli, and Marco Trubian. Solving the feedback vertex set problem on undirected graphs. *Discrete Applied Mathematics*, 101(1-3):37–51, 2000.

[29] Sergio Cabello and David Gajser. Simple PTAS's for families of graphs excluding a minor. *Discrete Applied Mathematics*, 189(C):41–48, 2015.

[30] Timothy M Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. In *Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry*, SocG'09, pages 333–340, 2009.

[31] Joseph Cheriyan and Ramakrishna Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM Journal on Computing*, 30(2):528–560, 2000.

[32] Norishige Chiba, Takao Nishizeki, and Nobuji Saito. Applications of the Lipton and Tarjan's planar separator theorem. *Journal of information processing*, 4(4):203–207, 1981.

[33] Norishige Chiba, Takao Nishizeki, and Nobuji Saito. An approximation algorithm for the maximum independent set problem on planar graphs. *SIAM Journal on Computing*, 11(4):663–675, 1982.

[34] Vincent Cohen-Addad, Éric Colin de Verdière, Philip N Klein, Claire Mathieu, and David Meierfrankenfeld. Approximating connectivity domination in weighted bounded-genus graphs. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 584–597. ACM, 2016.

[35] Vincent Cohen-Addad, Philip N Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in Euclidean and minor-free metrics. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, FOCS' 16, 2016.

[36] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

[37] Artur Czumaj, Michelangelo Grigni, Papa Sissokho, and Hairong Zhao. Approximation schemes for minimum 2-edge-connected and biconnected subgraphs in planar graphs. In *Proceedings of the fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 496–505. Society for Industrial and Applied Mathematics, 2004.

[38] Artur Czumaj and Andrzej Lingas. A polynomial time approximation scheme for Euclidean minimum cost k-connectivity. In *Automata, Languages and Programming*, pages 682–694. Springer, 1998.

[39] Artur Czumaj and Andrzej Lingas. On approximability of the minimum cost k-connected spanning subgraph problem. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 281–290, 1999.

[40] Erik D Demaine, Fedor V Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *Journal of the ACM (JACM)*, 52(6):866–893, 2005.

[41] Erik D Demaine, Mohammad Taghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 637–646, 2005.

[42] Erik D Demaine and MohammadTaghi Hajiaghayi. Bidimensionality: New connections between FPT algorithms and PTASs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'05, pages 590–601, 2005.

[43] Erik D Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Contraction decomposition in H-minor-free graphs and algorithmic applications. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 441–450, 2011.

[44] Camil Demetrescu, Andrew V Goldberg, and David S Johnson. Implementation challenge for shortest paths. In *Encyclopedia of Algorithms*, pages 1–99. Springer, 2008.

[45] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

[46] Hristo N Djidjev and Shankar M Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34:231–243, 1997.

[47] Hristo Nicolov Djidjev. On the problem of partitioning planar graphs. *SIAM Journal on Algebraic Discrete Methods*, 3(2):229–240, 1982.

[48] Hristo Nicolov Djidjev. A linear algorithm for partitioning graphs of fixed genus. *Serdica. Bulgariacae mathematicae publicationes*, 11(4):369–387, 1985.

[49] David Eisenstat, Philip Klein, and Claire Mathieu. An efficient polynomial-time approximation scheme for Steiner forest in planar graphs. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 626–638. SIAM, 2012.

[50] David Eppstein, Zvi Galil, Giuseppe F Italiano, and Thomas H Spencer. Separator based sparsification: I. Planarity testing and minimum spanning trees. *Journal of Computer and System Sciences*, 52(1):3–27, 1996.

[51] Ranel E Erickson, Clyde L Monma, and Arthur F Veinott Jr. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12:634–664, 1987.

[52] Kapali P Eswaran and R Endre Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.

[53] Guy Even, Joseph Naor, Baruch Schieber, and Leonid Zosin. Approximating minimum subset feedback sets in undirected graphs with applications. *SIAM Journal on Discrete Mathematics*, 13(2):255–267, 2000.

[54] Greg N Federickson. Fast algorithms for shortest paths in planar graphs with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.

[55] Fedor V Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Bidimensionality and EPTAS. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'11, pages 748–759, 2011.

[56] Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M Thilikos. Bidimensionality and kernels. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 503–510, 2010.

[57] Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M Thilikos. Linear kernels for (connected) dominating set on H-minor-free graphs. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 82–93, 2012.

[58] Eli Fox-Epstein, Shay Mozes, Phitchaya Mangpo Phothilimthana, and Christian Sommer. Short and simple cycle separators in planar graphs. *Journal of Experimental Algorithmics (JEA)*, 21:2–2, 2016.

[59] Greg N Frederickson and Joseph Jájá. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.

[60] Alan M Frieze, Gary L Miller, and Shang-Hua Teng. Separator based parallel divide and conquer in computational geometry. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pages 420–429, 1992.

[61] Harold N Gabow and Suzanne R Gallagher. Iterated rounding algorithms for the smallest k-edge connected spanning subgraph. *SIAM Journal on Computing*, 41(1):61–103, 2012.

[62] Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman & Co., 1979.

[63] Hillel Gazit and Gary L Miller. Planar separators and the Euclidean norm. *Algorithms*, pages 338–347, 1990.

[64] John R Gilbert, Joan P Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs of bounded genus. *Journal of Algorithms*, 5(3):391–407, 1984.

[65] Teofilo F Gonzalez. *Handbook of Approximation Algorithms and Metaheuristics*. CRC Press, 2007.

[66] Martin Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23(4):613–632, 2003.

[67] Prabhakar Gubbala and Balaji Raghavachari. Approximation algorithms for the minimum cardinality two-connected spanning subgraph problem. In *Integer Programming and Combinatorial Optimization*, pages 422–436. Springer, 2005.

[68] Prabhakar Gubbala and Balaji Raghavachari. A 4/3-approximation algorithm for minimum 3-edge-connectivity. In *Algorithms and Data Structures*, pages 39–51. Springer, 2007.

[69] Yuri Gurevich, Larry Stockmeyer, and Uzi Vishkin. Solving NP-hard problems on graphs that are almost trees and an application to facility location problems. *Journal of the ACM (JACM)*, 31(3):459–473, 1984.

[70] Sariel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. *SIAM Journal on Computing*, 46(6):1712–1744, 2017.

[71] Monika R Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

[72] Derek A Holton, Bill Jackson, Akira Saito, and Nicholas C Wormald. Removable edges in 3-connected graphs. *J. Graph Theory*, 14:465–475, 1990.

[73] Martin Holzer, Frank Schulz, Dorothea Wagner, Grigorios Prasinos, and Christos Zaroliagis. Engineering planar separator algorithms. *Journal of Experimental Algorithmics (JEA)*, 14:5, 2009.

[74] John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[75] Yoichi Iwata. Linear-time kernelization for feedback vertex set. *arXiv preprint arXiv:1608.01463*, 2016.

[76] Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, LP-branching, and FPT algorithms. *SIAM Journal on Computing*, 45(4):1377–1411, 2016.

[77] Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 2001(1):39–60, 21.

[78] David S. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 8(3):438–448, 1987.

[79] Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

[80] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[81] Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. *Journal of Computer and System Sciences*, 81(8):1665–1677, 2015.

[82] Ken-ichi Kawarabayashi and Bruce Reed. A separator theorem in minor-closed classes. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 153–162, 2010.

[83] Jonathan A Kelner. Spectral partitioning, eigenvalue bounds, and circle packings for graphs of bounded genus. *SIAM Journal on Computing*, 35(4):882–902, 2006.

[84] Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41(2):214–235, 1994.

[85] Valerie King, Satish Rao, and Rorbert Tarjan. A faster deterministic maximum flow algorithm. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 157–164, 1992.

[86] Philip Klein and Shay Mozes. *Optimization algorithms for planar graphs.* In preparation, manuscript at http://planarity.org.

[87] Philip N Klein. A subset spanner for planar graphs, with application to subset TSP. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 749–756, 2006.

[88] Philip N Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM Journal on Computing*, 37(6):1926–1952, 2008.

[89] Philip N Klein, Claire Mathieu, and Hang Zhou. Correlation clustering and two-edge-connected augmentation for planar graphs. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 554–567. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

[90] Jon Kleinberg and Amit Kumar. Wavelength conversion in optical networks. *Journal of Algorithms*, 38:25–50, 2001.

[91] Dexter C Kozen. *The design and analysis of algorithms.* Springer Science & Business Media, 2012.

[92] Kazimierz Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

[93] Hung Le. A PTAS for subset TSP in minor-free graphs. *arXiv preprint arXiv:1804.01588*, 2018.

[94] Hung Le and Baigong Zheng. Local search is a PTAS for feedback vertex set in minor-free graphs. *CoRR*, abs/1804.06428, 2018.

[95] Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[96] Richard J Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM journal on computing*, 9(3):615–627, 1980.

[97] Flaminia L Luccio. Almost exact minimum feedback vertex set in meshes and butterflies. *Inf. Process. Lett.*, 66(2):59–64, 1998.

[98] W. Mader. Homomorphiesätze für graphen. *Mathematische Annalen*, 178(2):154–168, 1968.

[99] Marjan Marzban, Qian-Ping Gu, and Xiaohua Jia. Computational study on planar dominating set problem. *Theoretical Computer Science*, 410(52):5455–5466, 2009.

[100] Kurt Mehlhorn, Stefan Näher, and Christian Uhrig. The LEDA platform for combinatorial and geometric computing. In *International Colloquium on Automata, Languages, and Programming*, pages 7–16. Springer, 1997.

[101] Kurt Mehlhorn, Adrian Neumann, and Jens M Schmidt. Certifying 3-edge-connectivity. *Algorithmica*, 77(2):309–335, 2017.

[102] Nabil H Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

[103] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.

[104] James B Orlin. Max flows in $O(nm)$ time, or better. In *Proceedings of the forty-fifth Annual ACM Symposium on Theory of Computing*, pages 765–774. ACM, 2013.

[105] Christos H Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 229–234. ACM, 1988.

[106] Panos M Pardalos, Tianbing Qian, and Mauricio GC Resende. A greedy randomized adaptive search procedure for the feedback vertex set problem. *Journal of Combinatorial Optimization*, 2(4):399–412, 1998.

[107] Shao-Meng Qin and Hai-Jun Zhou. Solving the undirected feedback vertex set problem by local search. *The European Physical Journal B*, 87(11):273, 2014.

[108] R Ravi and Philip Klein. When cycles collapse: A general approximation technique for constraind two-connectivity problems. In *Proceedings of the 3rd International Conference on Integer Programming and Combinatorial Optimization*, pages 39–55, 1993.

[109] Bruce Reed and David R Wood. A linear-time algorithm to find a separator in a graph excluding a minor. *ACM Transactions on Algorithms*, 5(4):39, 2009.

[110] Neil Robertson, Daniel Sanders, Paul Seymour, and Robin Thomas. The four-colour theorem. *Journal of Combinatorial Theory, Series B*, 70(1):2–44, 1997.

[111] Jens M Schmidt. Contractions, removals, and certifying 3-connectivity in linear time. *SIAM Journal on Computing*, 42(2):494–535, 2013.

[112] András Sebő and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014.

[113] Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[114] Robert Endre Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 1974.

[115] Siamak Tazari and Matthias Müller-Hannemann. Dealing with large hidden constants: Engineering a planar Steiner tree PTAS. *Journal of Experimental Algorithmics (JEA)*, 16(3–6), 2011.

[116] Kiem-Phong Vo. Finding triconnected components of graphs. *Linear and multilinear algebra*, 13(2):143–165, 1983.

[117] Klaus Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114:570–590, 1937.

[118] Hassler Whitney. Non-separable and planar graphs. *Trans. Amer. Math. Soc.*, 34:339–362, 1932.

[119] Christian Wulff-Nilsen. Separator theorems for minor-free and shallow minor-free graphs with applications. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 37–46, 2011.

[120] Mihalis Yannakakis. Node-and edge-deletion NP-complete problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 253–264, 1978.

[121] Zhiqiang Zhang, Ansheng Ye, Xiaoqing Zhou, and Zehui Shao. An efficient local search for the feedback vertex set problem. *Algorithms*, 6(4):726–746, 2013.

[122] Baigong Zheng. Linear-time approximation schemes for planar minimum three-edge connected and three-vertex connected spanning subgraphs. *arXiv preprint arXiv:1701.08315*, 2017.