Testing vertex connectivity of bowtie 1-plane graphs

by

Karthik Murali

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Karthik Murali is the sole author for Chapters 1, 2, 3, 5, 6, Sections 4.1 and 4.3 of Chapter 4, and Appendix A. These were all written under the supervision of Prof. Therese Biedl.

Section 4.2 was written as manuscript for publication [BM21], and is joint work with Prof. Therese Biedl.

Abstract

A *separating set* of a connected graph G is a set of vertices S such that $G - S$ is disconnected. S is a *minimum separating set* of G if there is no separating set of G with fewer vertices than S. The size of a minimum separating set of G is called the *vertex connectivity* of G. A separating set of G that is a cycle is called a *separating cycle* of G.

Let G be a planar graph with a given planar embedding. Let $\Lambda(G)$ be a supergraph of G obtained by inserting a face vertex in each face of G and connecting the face vertex to all vertices on the boundary of the face. It is well known that a set S is a minimum separating set of a planar graph G if and only if the vertices of S can be connected together using face vertices to get a cycle X of length $2|S|$ that is separating in $\Lambda(G)$.

We extend this correspondence between separating sets and separating cycles from planar graphs to the class of *bowtie 1-plane graphs*. These are graphs that are embedded on the plane such that each edge is crossed at most once by another edge, and the endpoints of each such crossing induce either K_4, $K_4 \setminus \{e\}$ or C_4. Using this result, we give an algorithm to compute the vertex connectivity of a bowtie 1-plane graph in linear time.

Acknowledgements

I would like to thank Prof. Therese Biedl for her support and guidance during my graduate studies. I would also like to thank Prof. Richard Peng and Prof. Jim Geelen for agreeing to be readers for my thesis.

Dedication

I would like to dedicate this thesis to all those people who inspired me to pursue mathematics in graduate school:

- Prof. Manu Basavaraju, my undergraduate advisor at NITK Surathkal.
- Prof. Sunil Chandran, my mentor at IISc Bengaluru.
- Prof. Jasine Babu, my internship guide at IIT Palakkad.
- Prof. V. Murugan, an amazing Math teacher at NITK Surathkal.

Table of Contents

List of Figures ix

1 Introduction 1

1.1 Existing results . . . 2

1.2 Our results . . . 5

2 Preliminaries 8

2.1 Graphs . . . 8

2.2 Graph drawings . . . 10

2.3 Planar and 1-planar embeddings . . . 10

3 Toolbox 13

3.1 Crossings in 1-plane graphs . . . 13

3.2 Minimal separating sets . . . 15

3.3 Radial planarisation . . . 16

3.4 Constrained separating cycles . . . 17

3.5 Separating sets from separating cycles . . . 19

4 Separating cycles from separating sets 24

4.1 Refined 1-plane graphs . . . 24

4.2 Plane and full 1-plane graphs . . . 28

4.3 Almost full 1-plane and bowtie 1-plane graphs . . . 34

5 Computing constrained separating cycles in linear time **48**

5.1 Separating cycles in graphs of bounded treewidth 50

5.2 Constrained separating cycles in graphs of bounded treewidth 66

5.3 The separating subgraph isomorphism algorithm 69

5.4 Computing vertex connectivity in linear time 75

6 Conclusion **77**

References **81**

APPENDIX **85**

A List of corrections to Eppstein's paper **86**

List of Figures

1.1 A planar graph and its radialisation. 4
1.2 Some types of crossings in a 1-planar embedding 5

2.1 Different drawings of the same planar embedding 10
2.2 Planar and 1-planar graphs 11

3.1 A crossing in a 1-plane graph 13
3.2 Some types of crossings in a 1-planar embedding 14
3.3 Adding the kite edge e to the crossing $\{(u, v), (w, x)\}$ 14
3.4 An almost full crossing and a butterfly crossing 15
3.5 Radial planarisation at a butterfly crossing (after removing the spine, if it existed) and at a normal face 17
3.6 Unconstrained and constrained separating cycles 18
3.7 The marking function. (Filled diamonds denote vertices and crossing points on X.) 19
3.8 Paths P in G and P' in $\Lambda(G)$. 21
3.9 An illustration for Case 2 23

4.1 A flowchart showing the steps to obtain a constrained separating cycle in $\Lambda(G)$. 25
4.2 Transforming G to G^+ 26
4.3 Transforming G^+ to G_{ref} 26

4.4 e_1 and e_2 form a closest (ϕ_1, ϕ_2)-pair. . . . 29
4.5 Illustration for Case 2 . . . 31
4.6 Constructing a fence in $\Lambda(G)$ . . . 32
4.7 Transforming X_{ref} to X . . . 33
4.8 Constructing X_{ref} when there is an uncrossed edge parallel to a spine . . . 34
4.9 Triangular contraction . . . 36
4.10 Quadrangular contraction at a spine incident with a full crossing . . . 36
4.11 Quadrangular contraction at a butterfly crossing . . . 36
4.12 A contraction at a spine does not create loop-crossings . . . 37
4.13 Extending X_2 to X_1: Case 1 . . . 39
4.14 Extending X_2 to X_1: Case 2 . . . 40
4.15 Extending X_1 to X_{ref}: Case 3. . . . 41
4.16 The expansion step crucially ensures that $|X_{\text{ref}}| \leq 2k$. . . . 42
4.17 Identifying two face vertices on a fence gives a smaller fence. . . . 44
4.18 Modifying the radial graph at butterfly crossing to get $\Lambda(G)$, which may involve shortening the cycle X. . . . 45
4.19 Re-routing X_{ref} at an almost full crossing to avoid spine faces. . . . 46

5.1 A graph and its tree decomposition. For the indicated bag N, the subgraph G_N is also shown. . . . 51
5.2 An extended partial isomorph. . . . 52
5.3 A partial isomorph boundary . . . 54
5.4 The marking function. (The filled diamonds denote vertices and crossing points on X.) . . . 67
5.5 Construction of the graphs Λ_i . . . 70
5.6 Constructing L_i inductively. The bold edges belong to $T(v_0)$ and the white circles are vertices of G. The dotted edges are added to L_i. The label ϑ for the components denote the labels of all vertices of that component. . . . 71
5.7 The algorithm for computing vertex connectivity of bowtie 1-plane graphs in linear time. . . . 76

6.1 An example of a 4-connected 1-plane graph that does not have a separating 8-cycle in its radial planarisation passing through a minimum separating set (white circles). 78

6.2 An example of a 1-plane graph where the vertices of a minimum separating set are far apart from each other. 78

6.3 An example of a 2-plane graph where the vertices of a minimum separating set are far apart from each other. 79

A.1 A figure that illustrates the necessity of Condition (CB6) for consistent boundaries. 86

A.2 A figure that illustrates that searching for separating cycles in $R(G)$ is not sufficient to find a separating set for G. 87

A.3 Construction of the graphs Λ_i 88

Chapter 1

Introduction

A graph is a mathematical structure that models relationships between objects, with the objects being represented as vertices and relations between objects as edges that connect vertices. The class of *planar graphs*, which are graphs that can be drawn on the plane with no edge crossing another, are fundamental to both graph theory and graph algorithms. The structural properties of planar graphs have been extensively studied, and have been used in the development of many efficient algorithms, even where the more general problem is NP-hard. However, most real-world graphs, such as social networks and biological networks, are non-planar. Consequently, we need to address structural and algorithmic challenges for non-planar graphs.

A natural starting point in the study of non-planar graphs is the class of *near planar graphs*, i.e., graphs that are close to being planar in some sense. The class of *1-planar graphs* is a very frequently studied class of near planar graphs. A graph is *1-planar* if it can be drawn on the plane such that each edge is crossed at most once. (Precise definitions will be given in Chapter 2.) This graph class was first introduced by Ringel in 1965 [Rin65] and has excited much interest recently, both with respect to the structural properties of these graphs and for developing algorithms tailored to this graph class. A large number of results are available for 1-planar graphs; refer to an annotated bibliography [KLM17] as well as some chapters in a recent book [HT20].

In this thesis, we look at the problem of vertex connectivity for 1-planar graphs. The problem of *vertex connectivity* is fundamental in graph theory: given a connected graph G, what is the smallest number of vertices that need to be removed from G to make it disconnected? Such a set of vertices is called a *minimum separating set* of G, and the number of vertices in the set is denoted by $\kappa(G)$. Vertex connectivity has numerous applications;

for example, in network reliability, it measures the resilience of a communication network, and in sociology, vertex connectivity is a measure of social cohesion.

1.1 Existing results

1.1.1 Vertex connectivity in general graphs

Since 1969, there has been a long line of research on efficient algorithms for deciding k-connectivity (i.e. deciding $\kappa(G) \geq k$) or computing the connectivity $\kappa(G)$ of general graphs. For a general graph G on n vertices and m edges, it is very easy to test in linear time (i.e. $O(m+n)$ time) whether $\kappa(G) \geq 1$ by running any graph traversal algorithm. In 1969, Kleitman [Kle69] showed that vertex connectivity can be decided in time $O(k^2nm)$. In particular, when $m \in O(n)$ and $k \in O(1)$, vertex connectivity can be decided in $O(n^2)$ time [Kle69]. Subsequently, [Tar72] and [HT73] presented $O(m)$ time algorithms to decide k-connectivity when $k = 2$ and $k = 3$ respectively. (The algorithm for deciding 3-connectivity had some errors which were pointed out and corrected in 2000 by Gutwenger and Mutzel [GM00].)

In 1974, Aho, Hopcroft and Ullman [AHU74, Problem 5.30] conjectured that there exists an $O(m)$ time algorithm for computing $\kappa(G)$. For a long time, the fastest known algorithms to decide whether $\kappa(G) \geq 4$ ran in $O(n^2)$ time. For $\kappa(G) = 4$, the first $O(n^2)$ algorithm was by Kanevsky and Ramachandran [KR91]. For $\kappa(G) \in O(1)$, the first $O(n^2)$ algorithm was by Nagamochi and Ibaraki [NI92]. For general k and m, the fastest running times are $\tilde{O}(n^\omega + nk^\omega)$ [1] by Linial, Lovász and Wigderson [LLW88] and $\tilde{O}(kn^2)$ by Henzinger, Rao and Gabow [HRG00]. Both algorithms are randomised and are correct with high probability. The fastest deterministic algorithm is by Gabow [Gab06] and takes time $O(m \cdot (n + \min\{k^{5/2}, kn^{3/4}\}))$.

The last few years have seen some breakthroughs for deciding k-connectivity when k is small. Recently, Forster et al. [FNY+20] used fast local-cut algorithms to show that there is a randomised algorithm that takes time $\tilde{O}(m+nk^3)$ to decide whether $\kappa(G) \geq k$. A graph is called *sparse* if $m = \tilde{O}(n)$, and hence, $k = O(\text{polylog}(n))$. For sparse graphs, the algorithm runs in near-linear-time. By combining this with the previous results ([LLW88, HRG00]), the best running time for $k \geq 4$ is $\tilde{O}(m + \min\{nk^3, n^2k, n^\omega + nk^\omega\})$.

[1] $\tilde{O}(g(n)) = O(g(n)\log^c n)$ for some constant c, and ω is the matrix multiplication exponent; currently, $\omega < 2.372$.

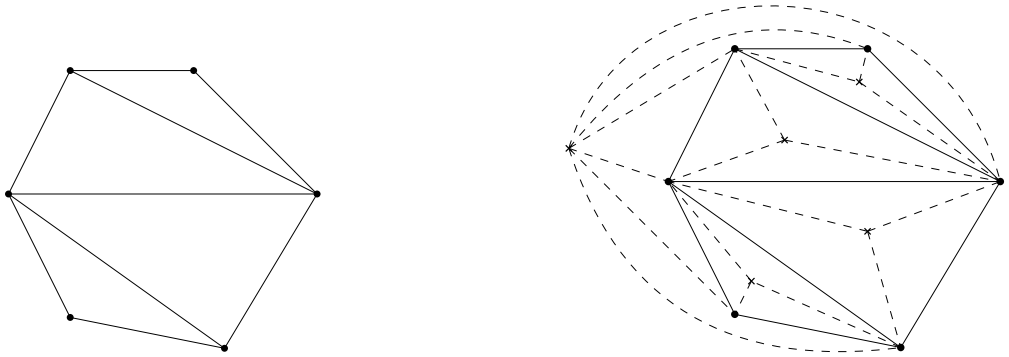
As for deterministic algorithms, the algorithm due to Gabow [Gab06], which was the fastest until recently, has running time no better than $\tilde{O}(n^2)$ even for sparse graphs. However, for such graphs, the $\tilde{O}(n^2)$ bound dates back to the result of Kleitman in 1969 [Kle69]. Recently, Gao et al. [GLN+19] used a sub-quadratic time algorithm for computing balanced sparse cuts to give an algorithm for deciding k-connectivity in time $\widehat{O}(m + \min\{n^{1.75}k^{1+k/2}, n^{1.9}k^{2.5}\})$[2] when $k < n^{1/8}$. This algorithm runs in sub-quadratic time for sparse graphs and hence improves the quadratic bound by Kleitman.

Very recently, Li et al. [LNP+21] gave a randomised algorithm that solves the vertex connectivity problem by reducing it to a set of maxflow instances. This implies that if there is a m^α time maxflow algorithm, for any $\alpha \geq 1$, then vertex connectivity can be solved in $\tilde{O}(m^\alpha)$ time. The current fastest randomised maxflow algorithm is by Chen et al. [CKL+22] and takes time $\widehat{O}(m)$. This implies that vertex connectivity of a general graph can be decided in $\widehat{O}(m)$ time with high probability. On the other hand, the problem of obtaining a deterministic linear time algorithm for deciding vertex connectivity is still open.

We now briefly review some of the literature on topics related to vertex connectivity, but which we will not study in the thesis. The notion of vertex connectivity in directed graphs is slightly different from undirected graphs. A set of vertices S form a separating set of a directed graph G if there are two vertices u, v in $G - S$ such that there is no directed path from u to v in $G - S$. For directed graphs, an $O(m)$ time algorithm is only known for $k \leq 2$ by Georgiadis [Geo10]. For general k and m, the fastest randomised algorithms are $\tilde{O}(n^\omega + nk^\omega)$ by Cheriyan and Reif [CR94], $\tilde{O}(mn)$ by Henzinger et al. [HRG00] and $\tilde{O}(\min\{mk^2, k^3n + k^{3/2}m^{1/2}n\})$ by Forster et al. [FNY+20]. More recently, Li et al. [LNP+21] gave a randomised $mn^{1-1/12+o(1)}$ time algorithm for vertex connectivity of directed graphs which improves upon the $\tilde{O}(mn)$ bound by Henzinger et al. [HRG00]. The fastest deterministic algorithm for directed graphs is by Gabow [Gab06] and takes time $O(m \cdot (n + \min\{k^{5/2}, kn^{3/4}\}))$.

There have also been some approximation algorithms developed for vertex connectivity. There is a deterministic $O(\min\{\sqrt{n}, k\}n^2)$ time 2-approximation algorithm by Henzinger [HRG00] and a randomized $\tilde{O}(m)$ time $O(\log n)$-approximation algorithm by Censor-Hillel, Ghaffari and Kuhn [CHGK14]. While these two algorithms work only on undirected graphs, Forster et. al [FNY+20] give a $(1+\epsilon)$-approximation $\tilde{O}(\min\{mk/\epsilon, n^\omega/\epsilon^2\})$ time randomised algorithm for both directed and undirected graphs.

[2] $\widehat{O}(g(n)) = O(g(n)^{1+o(1)})$



(a) A planar graph G (b) The radialisation of G

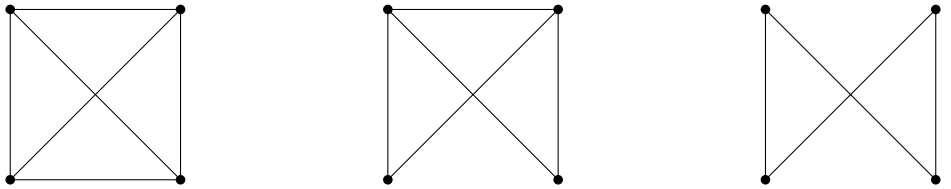
Figure 1.1: A planar graph and its radialisation.

1.1.2 Vertex connectivity in planar graphs

Any simple planar graph has at most $3n - 6$ edges. Hence, any planar graph G contains a vertex with at most five distinct neighbours, implying that $\kappa(G) \leq 5$. Since k-connectivity can be decided in linear time for $k \leq 3$, it only remains to decide whether $\kappa(G) = 4$ or $\kappa(G) = 5$ in linear time. In 1990, Laumond [Lau90] gave a linear time algorithm to compute $\kappa(G)$ for maximal planar graphs, which are planar graphs in which every face is a triangle. In [Epp99], Eppstein gave an algorithm to test vertex connectivity of general planar graphs in linear time.

Eppstein's algorithm is the crucial inspiration for the work in this thesis, and so we review it briefly here. The algorithm is based on the following approach. Given a planar graph G with a fixed planar embedding, let the *radialisation* $\Lambda(G)$ be the planar graph obtained by adding a new face vertex inside each face of G and connecting the face vertex to all the vertices on the boundary of the face (Figure 1.1). Let S be a separating set of G such that $|S| = k$. Then there exists a cycle X of length at most $2k$ in $\Lambda(G)$ that satisfies the following properties: (a) X separates two vertices of G, and (b) X uses only edges added during the radialisation. Observe that for any cycle X in $\Lambda(G)$ that satisfies these two properties, the vertices of G on X form a separating set of G of size $|X|/2$. Thus, computing a minimum separating set of a planar graph G reduces to the problem of finding a shortest separating cycle of length $2k$ (for $k \leq 5$) in $\Lambda(G)$ that satisfies the above properties.

The algorithm to obtain a shortest separating cycle is based on the separating subgraph isomorphism algorithm developed in [Epp99]. In this algorithm, $\Lambda(G)$ is viewed as a *planar host graph* and a separating cycle is viewed as a *pattern graph*. The algorithm searches



(a) Full crossing (b) Almost full crossing (c) Bowtie crossing

Figure 1.2: Some types of crossings in a 1-planar embedding

the host graph for an instance of the pattern graph, and outputs the instance if it exists. The algorithm runs in linear time when the size of the pattern graph is bounded. Since a minimum separating set of a planar graph corresponds to a shortest separating cycle in $\Lambda(G)$, the size of the shortest separating cycle is at most 10. Therefore, a shortest separating cycle of $\Lambda(G)$ can be computed in linear time, and a minimum separating set of G can be then be extracted from the cycle. Thus, $\kappa(G)$ can be computed in linear time.

Although we only study sequential algorithms for vertex connectivity, it is worth mentioning that Gianinazzi and Hoefler [GH20] presented a parallel separating subgraph isomorphism for planar graphs. Their algorithm shows that planar vertex connectivity can be answered in $O(n \log n)$ work and $O(\log^2 n)$ depth. While 2-connectivity and 3-connectivity have long been solved for general graphs with linear work and logarithmic depth [MR92, TV85], no sub-quadratic work and polylogarithmic depth bound was known previously for 4-connected and 5-connected planar graphs.

1.2 Our results

The goal of this thesis is to generalise the correspondence between separating sets and separating cycles from planar graphs to 1-planar graphs and hence obtain a linear time algorithm to compute the vertex connectivity of 1-planar graphs. To our knowledge, there are no previous results for computing connectivity in 1-planar graphs faster than for general graphs.

A 1-planar embedding can contain different types of crossings (Figure 1.2). A crossing is called *full* if its endpoints induce K_4. A crossing is called *almost full* if its endpoints induce $K_4 \setminus \{e\}$, where e is an edge of K_4. A crossing is called a *bowtie crossing* if its endpoints induce C_4. A *bowtie 1-plane graph* is a 1-planar graph with a given 1-planar embedding such that each crossing in the embedding is full, almost full or bowtie.

Theorem 1 is the main result of the thesis.

Theorem 1. *A minimum separating set of a bowtie 1-plane graph can be computed in linear time.*

To prove Theorem 1, we reduce the problem of finding a minimum separating set to one of finding a shortest "constrained separating cycle" in the "radial planarisation" of the bowtie 1-plane graph. In Chapter 3, we give a precise definition of the terms "constrained separating cycle" and "radial planarisation". For now, the following approximate definitions suffice. The *radial planarisation* $\Lambda(G)$ of a 1-plane graph G is obtained by planarising the embedding by adding dummy vertices at all crossing points, and then radialising the resulting embedding. A *constrained separating cycle* of $\Lambda(G)$ is a separating cycle that satisfies a given list of constraints that tell which type of vertices, crossing points and edges of $\Lambda(G)$ can belong to the cycle.

To prove Theorem 1, we prove Theorems 2 and 3 that may be interesting in their own right. Theorems 2 and 3 show that the vertex connectivity of a bowtie 1-plane graph G is half the length of a shortest constrained separating cycle of $\Lambda(G)$. Theorem 4 shows that a shortest constrained separating cycle of $\Lambda(G)$ can be obtained in linear time. The proofs of the the three theorems form the content of Chapters 3, 4 and 5 respectively.

Theorem 2. *Let G be a bowtie 1-plane graph. If $\Lambda(G)$ contains a constrained separating cycle of length $2k$, then G contains a separating set of size at most k. Moreover, the separating set can be computed from the constrained separating cycle in time $O(k)$.*

Theorem 3. *Let G be a bowtie 1-plane graph. If G contains a separating set of size k, then $\Lambda(G)$ contains a constrained separating cycle of length at most $2k$.*

Any 1-planar graph G is at most 7-connected [BSW83]. Therefore, by Theorem 3, $\Lambda(G)$ has a constrained separating cycle of length at most 14. With this, we can show the following theorem.

Theorem 4. *Let G be a bowtie 1-plane graph. A shortest constrained separating cycle of $\Lambda(G)$ can be computed in linear time.*

As for planar graphs, the algorithm to obtain a shortest constrained separating cycle is based on the separating subgraph isomorphism algorithm developed by Eppstein in [Epp99]. However, the presence of constraints on the separating cycle necessitates modifications to the separating subgraph isomorphism algorithm. In Chapter 5, we give a

detailed explanation of Eppstein's algorithm and the modifications required in the algorithm so that it outputs a shortest constrained separating cycle of $\Lambda(G)$. (It also turns out that there were some errors in Eppstein's algorithm; we list these in Appendix A.) We conclude the thesis with open questions in Chapter 6.

Chapter 2

Preliminaries

2.1 Graphs

A *graph* G is a triple consisting of a *vertex set* $V(G)$, an *edge set* $E(G)$, and an *incidence relation* that associates with each edge two vertices called the *endpoints* of the edge. We use n to denote $|V(G)|$ and m to denote $|E(G)|$. A vertex v is *incident* with an edge e if v is an endpoint of e; we also say that e is an edge *at* v. Two vertices u and v are *adjacent* if they are the endpoints of an edge. A *loop* is an edge whose endpoints are the same. Two edges are *parallel* if they have the same pair of endpoints. A *simple graph* is a graph having no loops or parallel edges. An edge of a simple graph is determined by its endpoints, so we can name an edge e by its endpoints $e = (u, v)$. An edge of a graph that is not simple can also be named by its endpoints if the edge under consideration is clear from context. The degree of a vertex v is the number of edge-incidences at v (loops count twice). The neighbourhood of v, written $N(v)$, is the set of vertices adjacent to v.

A *walk* in a graph is a sequence of alternating vertices and edges $v_0, e_1, v_1, e_2, \ldots, e_k, v_k$ such that the endpoints of each edge e_i are v_{i-1} and v_i. A *path* is a sequence of distinct vertices $v_0 v_1 \ldots v_k$ with the property that each vertex in the sequence is adjacent to the next vertex. Equivalently, a path is a walk with no repeated edges or vertices. A *cycle* is a sequence of vertices $v_0 v_1 \ldots v_k v_0$ where $v_0 v_1 \ldots v_k$ is a path and v_k is adjacent to v_0. The length of a path or a cycle is the number of edges in the path or the cycle; a path or a cycle of length k is called *k-path* or a *k-cycle*.

A *subgraph* of a graph G is a graph H such that $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ and the incidence relation of H is a restriction of the incidence relation of G onto $V(H)$. If H is a

subgraph of G, we write $H \subseteq G$; we also say that G is a *supergraph* of H. Deleting a vertex of a graph deletes the vertex and all edges at that vertex. We write $G - v$ or $G - S$ for the subgraph obtained by deleting a vertex v or a set of vertices S. An *induced subgraph* H of a graph G is obtained by deleting all vertices of $V(G) - V(H)$. Equivalently, $E(H)$ consists of all those edges of G whose endpoints are in $V(H)$.

Contracting an edge of a graph refers to the operation of deleting the edge and simultaneously identifying its two endpoints. (Contracting a loop is the same as deleting it.) A graph H is called a *minor* of a graph G if H can be obtained from G by deleting edges and vertices and by contracting edges. The *subdivision of an edge* (u, v) of a graph is obtained by adding a new vertex w and replacing the edge (u, v) by two edges (u, w) and (w, v). A graph that is derived from a sequence of subdivisions of the edges of a graph G is called a *subdivision of* G. If G contains a subgraph that is a subdivision of H, then H is a minor of G.

A *complete graph* on n vertices, denoted by K_n, is a simple graph where every pair of vertices is adjacent. A graph is *bipartite* if its vertex set can be partitioned into two non-empty sets A, B such that all edges of the graph have one endpoint in A and the other endpoint in B. A graph G is called a *complete bipartite graph* if it is bipartite with a bipartition A, B such that every vertex in A is adjacent to every vertex in B. If $|A| = a$ and $|B| = b$, we write $G = K_{a,b}$.

A graph is *connected* if each pair of vertices belongs to a path; otherwise, it is *disconnected*. A *component* of a graph G is a maximal connected subgraph, i.e., it is not contained in any other connected subgraph of G. A connected graph has a single component which is the graph itself. A *separating set* of a connected graph is a set $S \subseteq V(G)$ such that $G - S$ has more than one component; each such component is called a *flap* of S. The *connectivity* of G, written $\kappa(G)$, is the minimum size of a vertex set S such that $G - S$ is disconnected or has only one vertex. A graph is *k-connected* if its connectivity is at least k. A set S is a *minimal separating set* if $S - v$ is not a separating set for any $v \in S$. S is a *minimum separating set* if it is a separating set with the fewest possible number of vertices. By definition, every minimum separating set is also minimal.

We assume familiarity with algorithms, asymptotic notation, and NP-hardness, see e.g. [CLRS09]. We say that an algorithm operating on a graph with n vertices and m edges takes *linear time* if its run-time is in $O(m + n)$.

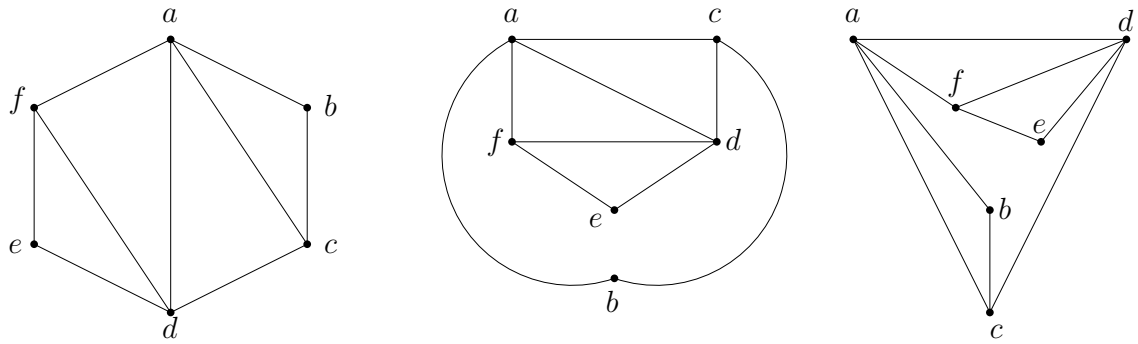


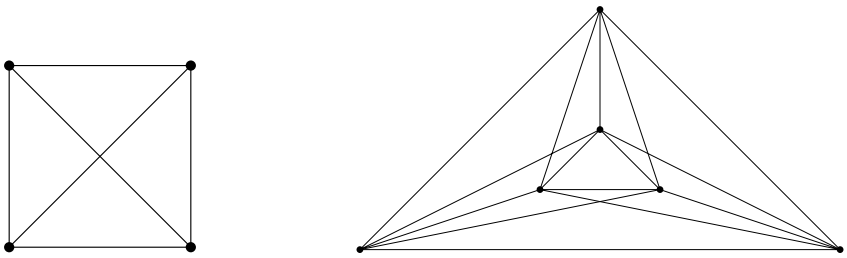
Figure 2.1: Different drawings of the same planar embedding

2.2 Graph drawings

Let $G = (V, E)$ be a graph. A *drawing* of G on the plane is a pair of functions (Γ_V, Γ_E), where Γ_V maps each vertex v to a distinct point $\Gamma_V(v)$ in $\mathbb{R}^2$, and Γ_E maps each edge (u, v) to a curve $\Gamma_E(u, v)$ in $\mathbb{R}^2$ whose endpoints are $\Gamma_V(u)$ and $\Gamma_V(v)$. A *good drawing* of a graph is one in which each edge is drawn as a simple curve (i.e., non-self-intersecting), and any two non-parallel edges intersect at most once, either at a common endpoint or in the interior of the edges. Parallel edges must intersect exactly twice at their common endpoints. Moreover, not more than two edges can cross at a single point. Degenerate drawings having edges touching each other tangentially or vertices lying in the interior of edges are also not considered good drawings. Therefore, if two edges without a common endpoint intersect, then they properly cross each other. Hereafter, we assume that all graph drawings are good.

2.3 Planar and 1-planar embeddings

A drawing of a graph G is *planar* if no two edges of the drawing cross. A *planar graph* is a graph that admits a planar drawing. A planar drawing of a graph G divides the plane into connected regions called *faces*. The infinite region is called the *outer-face*. Every other face is called an *inner face*. A face is identified by its *facial circuit* which is a walk along the boundary of the face such that the face lies to the left of all the edges in the walk. The set of facial circuits of a planar drawing gives a *planar embedding* of the graph. A planar graph with a given planar embedding is called a *plane graph*. The set of embeddings of a



(a) K_4 (b) K_6

Figure 2.2: Planar and 1-planar graphs

planar graph G partition the set of all planar drawings of G into equivalence classes, where two drawings are *equivalent* if and only if they have the same set of facial circuits. Figure 2.1 shows three drawings of a planar graph that have the same embedding. Observe that drawings which are equivalent under the same embedding could have different outer-faces.

The data structure commonly used to store a planar embedding is called a rotation system. The *rotation* at a vertex in a drawing is a cyclic order of edges at that vertex. A *rotation system* is the set of all rotations at all vertices. The rotation at a vertex v is denoted by $\rho(v)$. An *angle* at v is a sequence $\langle u, v, w\rangle$, where (v, w) and (v, u) are consecutive edges in $\rho(v)$. A *bigon* is a closed face that contains exactly two vertices and two angles.

The class of planar graphs is *minor-closed*, which means that a minor of a planar graph is also planar. A well-known theorem in graph theory, known as the Wagner's theorem, states that a graph is planar if and only if it does not contain a minor of K_5 or $K_{3,3}$. In fact, a stronger forbidden characterisation is true: a graph is planar if and only if it does not contain a subgraph that is a subdivision of K_5 or $K_{3,3}$. This is famously known as Kuratowski's theorem.

A drawing of a graph G is *1-planar* if each edge is crossed at most once by another edge. A *1-planar graph* is a graph that admits a 1-planar drawing. Figure 2.2a shows a 1-planar drawing of K_4. Although the drawing is 1-planar, one can draw K_4 without a crossing because it is a planar graph. Figure 2.2b shows a 1-planar drawing of K_6. Unlike K_4, any drawing of K_6 must have a crossing since it contains K_5 which is not a planar graph (by Kuratowski's theorem). The *planarisation* of a 1-planar drawing is obtained by placing a dummy vertex at each crossing point, thereby making the resulting drawing planar. The set of facial circuits of a planarised 1-planar drawing gives a *1-planar embedding* of the graph. For a 1-planar embedding, the terms *rotation system*, *bigon*, and *angle*, all refer to their counterparts in the planarisation of the embedding. A 1-planar graph with a given

1-planar embedding is called a *1-plane graph*.

Given a planar graph, one can derive a planar embedding in linear time [HT74]. In contrast to planar graphs, testing 1-planarity is NP-hard [KM13]. Hence, in all the algorithms for 1-planar graphs that we present here, we assume that the input is a 1-plane graph, that is a 1-planar graph with a given 1-planar embedding.

Chapter 3

Toolbox

In this chapter, we discuss the essential tools and ideas needed to prove Theorems 2-4. We conclude the chapter with a proof of Theorem 2.

Assumptions on the input graph. Throughout the thesis, we assume that we are given a bowtie 1-plane graph G that is connected, loopless, has no bigons, but may have parallel edges (Section 3.1). If G has loops or bigons, they can all be detected and removed in linear time; this does not affect the vertex connectivity of the graph.

3.1 Crossings in 1-plane graphs

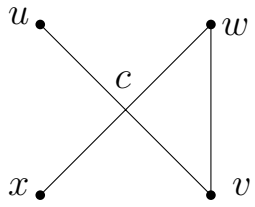
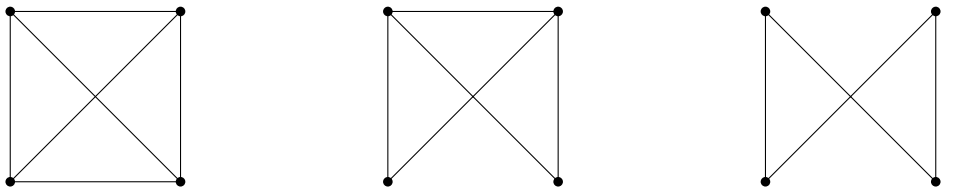


Figure 3.1: A crossing in a 1-plane graph

A *crossing* in a 1-plane graph G is a pair of edges that intersect at a point which is not a vertex. The point of intersection is called the *crossing point*. An *endpoint of a crossing* is a vertex that is incident with either edge of the crossing. Since G has no loops and we assume a good drawing, all the four endpoints of a crossing are distinct. Figure 3.1 shows



(a) Full crossing (b) Almost full crossing (c) Bowtie crossing

Figure 3.2: Some types of crossings in a 1-planar embedding

a crossing $\{(u, v), (w, x)\}$ with c as the crossing point. The endpoints of the crossing are u, v, w, x.

A crossing is called *full* if its endpoints induce K_4. A crossing is called *almost full* if its endpoints induce $K_4 \setminus \{e\}$, where e is an edge of K_4. A crossing is called a *bowtie crossing* if its endpoints induce C_4 (Figure 3.2). The crossing point of a full crossing, almost full crossing and a bowtie crossing are called *full crossing point*, *almost full crossing point* and *bowtie crossing point* respectively.

A *full 1-plane graph* is a 1-plane graph in which every crossing is full. An *almost full 1-plane graph* is a 1-plane graph in which every crossing is either full or almost full. A *bowtie 1-plane graph* is a 1-plane graph in which every crossing is full, almost full or a bowtie crossing.

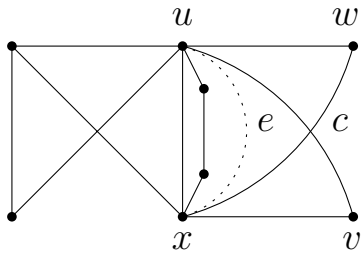
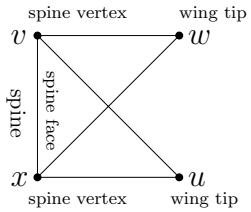
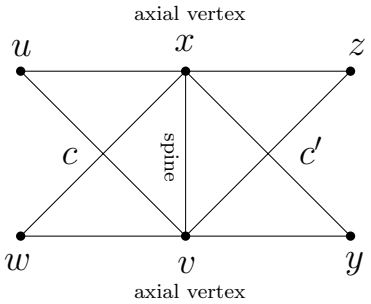


Figure 3.3: Adding the kite edge e to the crossing $\{(u, v), (w, x)\}$

Let $\{(u, v), (w, x)\}$ be a crossing in a 1-plane graph G with c as the crossing point. A *kite face* of the crossing is a face of degree 3 incident with c in the planarisation of G. The edge of a kite face that is not incident with c is called a *kite edge* of the crossing. For example, in Figure 3.3, (u, w, c) and (x, v, c) are two kite faces of the crossing $\{(u, v), (w, x)\}$ with kite edges (u, w) and (x, v) respectively. However, (u, x) is not a kite edge of the crossing since it does not bound a face of degree 3 at c. However, note that we can always add an edge e to G with endpoints u, x so that $e, (c, u), (c, x)$ form a kite face of the crossing.



(a) An almost full crossing with wing tips u, w and spine (v, x)



(b) A butterfly crossing with spine (x, v)

Figure 3.4: An almost full crossing and a butterfly crossing

(Note that the parallel edge added does not create a bigon.) Hence, we assume that in the given bowtie 1-plane graph, a full crossing has 4 kite faces, an almost full crossing has 3 kite faces, and a bowtie crossing has 2 kite faces that touch each other only at the crossing point.

Let $\{(u, v), (w, x)\}$ be an almost full crossing with $(u, w) \notin E(G)$. The vertices u, w are called *wing tips* of the crossing and the edge (v, x) is called the *spine* of the crossing (Figure 3.4a). Vertices v, x are called *spine vertices* of the crossing. The kite face of the crossing incident with the spine is called the *spine face* of the crossing.

Let $\{(u, v), (w, x)\}$ and $\{(x, y), (v, z)\}$ be two crossings with kite edges $(u, x), (w, v)$ and $(x, z), (v, y)$ respectively. Moreover, the crossings do not have kite edges (u, w) and (y, z). Let c and c' be their crossing points respectively. If there is no vertex inside the region bounded by (x, c), (c, v), (v, c'), (c', x), the pair of crossings is called a *butterfly crossing* (Figure 3.4b). The vertices x, v are called the *axial vertices* of the butterfly crossing. If the edge (x, v) is present, then the edge is a spine of both almost full crossings, and is called the *spine of the butterfly crossing*. Note that in a butterfly crossing, the vertices u, w, y, z need not be distinct.

3.2 Minimal separating sets

Recall that a *separating set* of a graph is a set $S \subseteq V(G)$ such that $G - S$ has more than one component; each such component is called a *flap* of $G - S$. A set S is a *minimal separating set* if $S - \{v\}$ is not a separating set for any $v \in S$.

Let G be a 1-plane graph and S be a minimal separating set of G. To prove Theorem 3, we alter the graph G by adding, removing or contracting edges. Whenever we perform such alterations, we will refer to the following observation to show that S remains a minimal separating set in the altered graph.

Observation 5. *Let S be a separating set of a graph G. Then S is a minimal separating set of G if and only if every vertex of S is adjacent to some vertex in every flap of $G - S$.*

Proof. Let $\phi_1, \ldots, \phi_k$ be the flaps of $G - S$, where $k \geq 2$. We prove the forward direction by its contrapositive. Suppose that there is a vertex $v \in S$ and an index $i \in \{1, \ldots, k\}$ such that v has no neighbour in ϕ_i. Then the graph $\phi_1 \cup \ldots \phi_k \cup \{v\}$ is disconnected, with vertices of ϕ_i belonging to one component and the vertex v belonging to another. Therefore, $S - \{v\}$ is a separating set. This shows that S is not minimal.

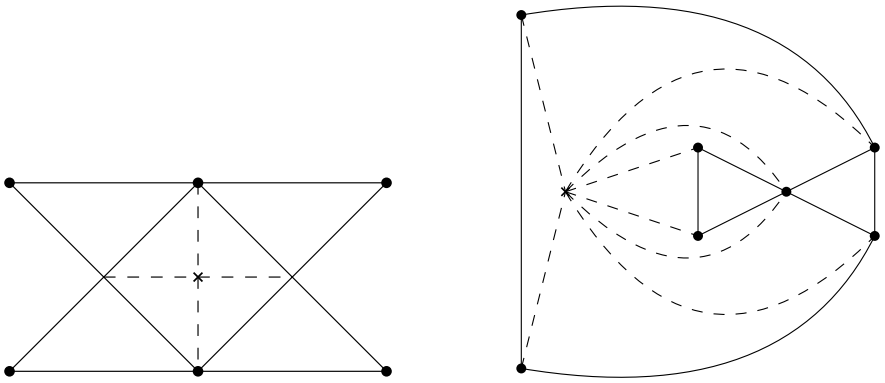
Now, we prove the reverse direction. Suppose that for every vertex $v \in S$ and every index $i \in \{1, \ldots, k\}$, v has a neighbour in ϕ_i. Consider the graph $G - S'$, where $S' = S - \{v\}$ for some $v \in S$. For any two vertices $u_i \in \phi_i$ and $u_j \in \phi_j$, $G - S'$ has a path connecting u_i and u_j passing through v. Therefore, $G - S'$ is connected. Hence, $S - \{v\}$ is not a separating set for any $v \in S$, showing that S is minimal. □

3.3 Radial planarisation

Recall from Section 1.1 that Eppstein [Epp99] used the radialisation of a planar graph for testing vertex connectivity. We now generalise this concept to bowtie 1-plane graphs with some non-trivial modifications at butterfly crossings. For a given bowtie 1-plane graph G, the *radial planarisation* $\Lambda(G)$ is obtained as follows.

1. Planarise G by inserting a dummy vertex at each crossing point.
2. Delete the spines from all butterfly crossings that have a spine.
3. Insert a *face vertex* inside each face, and add edges to the face vertex bisecting every angle of the planarised face.

(Figure 3.5b illustrates Step 3 above.) The resulting graph is called the *radial planarisation* $\Lambda(G)$. The subgraph of $\Lambda(G)$ formed by the edges incident with face vertices is called the *radial graph* of G, denoted by $R(G)$. (The term 'radial graph' is borrowed from



(a) Butterfly crossing (b) A normal face

Figure 3.5: Radial planarisation at a butterfly crossing (after removing the spine, if it existed) and at a normal face

[FT06], where radial graph is defined for plane graphs. In the context of plane graphs, that definition is the same as the one here.) Note that if G is a plane graph, then $\Lambda(G)$ is the same as the radialisation of G discussed in Section 1.1.

Note that $\Lambda(G)$ has three types of vertices: vertices of G, dummy vertices at the crossing points of G, and face vertices. Throughout the thesis, the term *vertices of* G in $\Lambda(G)$ will refer to only the vertices of G, and not dummy vertices or face vertices. If X is a cycle of $\Lambda(G)$, we use the notation $V_G(X)$ to denote the set of vertices of G on X. We also use the term *face vertex at a butterfly crossing* to refer to the face vertex that is adjacent to the axial vertices and the crossing points of a butterfly crossing (Figure 3.5a).

The following observations about the radial graph $R(G)$ follow from the construction of $\Lambda(G)$.

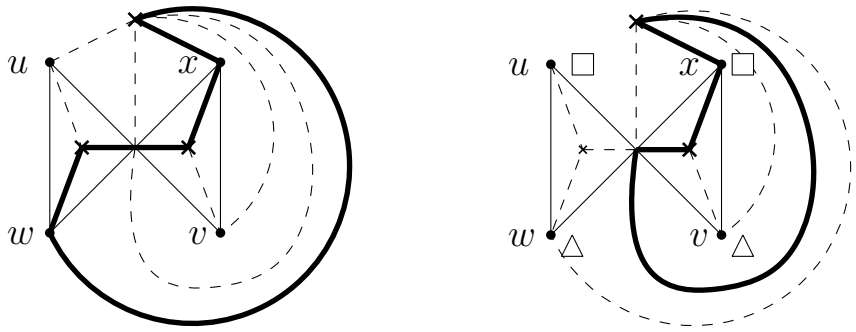
Observation 6. *$R(G)$ is a plane graph.*

Observation 7. *$R(G)$ does not contain any bigons.*

Observation 8. *$R(G)$ is a bipartite graph, with the face vertices on one side of the bipartition, and the remaining vertices on the other.*

3.4 Constrained separating cycles

Recall that the major goal of the thesis is to generalise the correspondence between separating sets and separating cycles from plane graphs to bowtie 1-plane graphs. For a plane



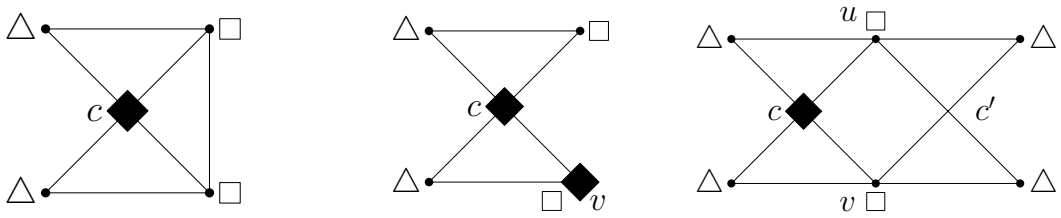
(a) Unconstrained separating cycle (b) Constrained separating cycle

Figure 3.6: Unconstrained and constrained separating cycles

graph G and a separating cycle X of $\Lambda(G)$, it is easy to see that $V_G(X)$ forms a separating set of G. However, such a relation does not extend to 1-plane graphs for arbitrary separating cycles. For example, Figure 3.6a shows a separating cycle in the radial planarisation of a bowtie crossing $\{(u, v), (w, x)\}$ that separates u and v; however, the vertices of G on the cycle, namely w, x, do not separate u and v. Hence, to be able to extract a separating set from a separating cycle, we need suitable restrictions on the separating cycle. These restrictions must also mark vertices suitably so that it becomes easy to extract a separating set from the separating cycle. So we define a separating cycle X of $\Lambda(G)$ to be a *constrained separating cycle* if and only if X satisfies the following constraints.

- (Ψ1): X is a subgraph of the radial graph $R(G)$ and separates two vertices of G.
- (Ψ2): There exists a *marking* function $\beta : V(G) \mapsto \{\triangle, \square\}$ such that for any crossing point that X visits:
  1. The endpoints of each edge of the crossing are marked with opposite symbols.
  2. The endpoints of the crossing marked $\triangle$ are not adjacent in G (Figure 3.7a).
  3. If an endpoint of the crossing is on X, then the endpoint marked $\square$ (Figure 3.7b).
  4. If no endpoint of the crossing is on X, then the crossing is part of a butterfly crossing with the endpoints marked $\square$ as the axial vertices (Figure 3.7c).

The cycle in Figure 3.6a satisfies (Ψ1), but there is no marking function β that satisfies (Ψ2), since (Ψ2.3) requires $\beta(x) = \beta(w) = \square$ but this violates (Ψ2.1). Figure 3.6b shows



(a) (Ψ2.1) and (Ψ2.2) (b) (Ψ2.3) (c) (Ψ2.4)

Figure 3.7: The marking function. (Filled diamonds denote vertices and crossing points on X.)

an example of a constrained separating cycle along with the marking function. Notice that the vertices marked □ give a separating set of the graph. In fact, as we will see in Section 3.5, the set of vertices in $V_G(X)$ together with the endpoints of those crossings visited by X that are marked □ give a separating set of G.

We now make some useful observations that follow from (Ψ2).

Observation 9. *Let G be a bowtie 1-plane graph and X be a constrained separating cycle of $\Lambda(G)$. Then*

1. *X does not visit full crossing points of G.*
2. *If X visits an almost full crossing point of G, then the wing tips of the crossing are marked △ and the spine vertices are marked □.*
3. *If X visits a bowtie crossing point of G, then one set of non-adjacent endpoints of the crossing is marked △ and the other set is marked □.*

Proof. We only prove Observation 9(2) here. The proofs of Observations 9(1) and 9(3) follow similarly, and are left to the reader. Suppose that X visits an almost full crossing point. From (Ψ2.2), the wing-tips must be marked △ since that is the only pair of endpoints which are not adjacent. Consequently, from (Ψ2.1), the spine vertices must be marked □. □

3.5 Separating sets from separating cycles

In this section, we prove Theorem 2 which states that for a given bowtie 1-plane graph G, if $\Lambda(G)$ contains a constrained separating cycle of length $2k$, then G contains a separating

set of size at most k. Moreover, the separating set can be computed from the constrained separating cycle in time $O(k)$.

The following claim gives the essential idea for obtaining separating sets from separating cycles.

Claim 10. *Let G be a bowtie 1-plane graph, and let X be a constrained separating cycle in $\Lambda(G)$. Let $S \subseteq V(G)$ be a set of vertices that satisfy the following conditions:*

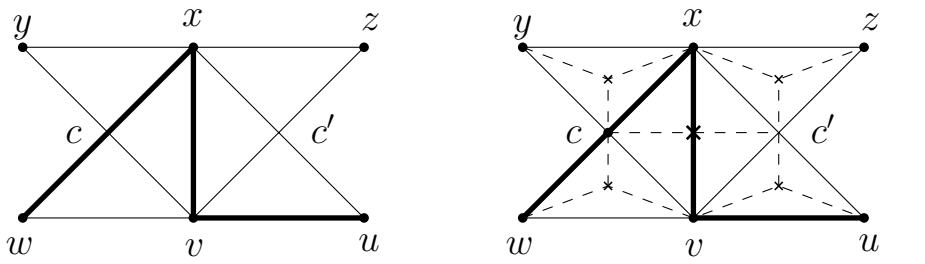
1. *There are at least two flaps of $\Lambda(G) \setminus X$ each of which contain a vertex of G that is not in S.*
2. *Every vertex of G that belongs to X also belongs to S.*
3. *If X visits a crossing point, then one endpoint of each edge of the crossing belongs to S.*
4. *If X visits the face vertex at a butterfly crossing that has a spine, then one axial vertex of the butterfly crossing belongs to S.*

Then S is a separating set of G.

Proof. Since X is a constrained separating cycle of $\Lambda(G)$, there exist two vertices $u, w \in V(G)$ that belong to different flaps of $\Lambda(G) \setminus X$ (by $\Psi 1$). From Condition 10(1), we can assume that $u, w \notin S$. Consider a simple path P connecting u and w in G. Let P' be the path in $\Lambda(G)$ corresponding to P defined as follows (see Figure 3.8 for an illustration): for each edge $e \in P$, (a) if e is uncrossed and also exists in $\Lambda(G)$, add e to P'; (b) if e is crossed, add the two edges of $\Lambda(G)$ corresponding to e to P'; and (c) if e is uncrossed but does not exist in $\Lambda(G)$, then e must be the spine of a butterfly crossing; add the two edges of $\Lambda(G)$ connecting the face vertex at the butterfly crossing to the axial vertices of the butterfly crossing to P'. Note that P' contains at most three types of vertices of $\Lambda(G)$: (a) vertices of G; (b) crossing points of G; (c) face vertices at butterfly crossings that have a spine in G.

Since X separates u and w in $\Lambda(G)$, the path P' intersects X at some vertex $v' \in X$. We now have three cases: (a) $v' \in V(G)$; (b) v' is a crossing point; (c) v' is the face vertex at a butterfly crossing that has a spine. For each of these cases, we show that there is a vertex $v \in V(G)$ on the path P such that $v \in S$.

- Suppose that $v' \in V(G)$. Then by Condition 10(2), $v' \in S$. Set $v := v'$.



(a) A path P in G. (b) Corresponding path P' in $\Lambda(G)$.

Figure 3.8: Paths P in G and P' in $\Lambda(G)$.

- Suppose that v' is a crossing point of G. Let $(x, v'), (y, v')$ be the edges of P' incident with v'. By our method of constructing P' from P, (x, y) must be an edge of P containing the crossing point v'. By Condition 10(3), either $x \in S$ or $y \in S$. By symmetry, suppose that $x \in S$. Set $v := x$.
- Suppose that v' is a face vertex at a butterfly crossing that has a spine. Let $(x, v'), (y, v')$ be the edges of P' incident with v'. By our method of constructing P' from P, (x, y) must be an edge of P that is the spine of the butterfly crossing. By Condition 10(4), either $x \in S$ or $y \in S$. By symmetry, suppose that $x \in S$. Set $v := x$.

In all of the above cases, P contains a vertex $v \in V(G)$ such that $v \in S$. Since $u, w \notin S$, v is in the interior of P. Since P is arbitrary, every path connecting u and w contains a vertex of S in its interior. This shows that $S \subseteq V(G)$ separates u and w. Thus, S is a separating set of G. □

We now prove Theorem 2. Let X be a constrained separating cycle of $\Lambda(G)$. We construct a separating set S of G as follows.

> For each vertex $v \in V_G(X)$, add v to S. If X visits a crossing point, add the endpoints of the crossing marked with □ to S (unless already added).

We now use Claim 10 to show that S is a separating set of G. The constraint (Ψ1) implies that there are two vertices of G, say s and s', which are separated by X. If $s, s' \notin S$, then Condition 10(1) holds. Otherwise, we cannot use s, s' for Condition 10(1), but as we will show now, we can use the same two flaps. Without loss of generality, suppose that

$s \in S$. To show that Condition 10(1) holds, we show that there is a vertex t of G such that $t \notin S$ and s, t belong to the same flap of $\Lambda(G) \setminus X$. Since $s \in S$ but $s \notin X$, the vertex s must have been added due to the second rule, i.e., s is an endpoint of a crossing through which X passes and $\beta(s) = \square$. By Observation 9, the crossing cannot be a full crossing. Also, from Observation 9 (see also Figure 3.7), there is a vertex t such that (s, t) is a kite edge of the crossing and $\beta(t) = \triangle$. As $\beta(t) = \triangle$, $t \notin V_G(X)$ (by Ψ2.3), and since each vertex of $S \setminus V_G(X)$ is marked $\square$, $t \notin S \setminus V_G(X)$. Therefore, $t \notin S$. Since (s, t) is an edge of $\Lambda(G)$ and $s, t \notin V_G(X)$, they belong to the same flap of $\Lambda(G) \setminus X$. Therefore, Condition 10(1) holds.

Condition 10(2) holds trivially by the construction of S. Condition 10(3) follows from Observation 9 and the construction of S. Now we show that 10(4) holds. Suppose that X visits the face vertex of a butterfly crossing that has a spine. If X visits one of the two axial vertices, then that vertex is in S. Otherwise X visits one crossing point of the butterfly crossing. From Observation 9, the axial vertices are marked $\square$ since the butterfly crossing has a spine, and by the construction of S, the axial vertices belong to S. Thus, Condition 10(4) holds. As all the four conditions of Claim 10 hold, S is a separating set of G.

To complete the proof of Theorem 2, we now show that $|S| \leq k$. Since $R(G)$ is bipartite (Observation 8) and $X \subseteq R(G)$ (by Ψ2), if $|X| = 2k$, then there are k vertices and crossing points of G that X visits. As $V_G(X) \subseteq S$, to prove that $|S| \leq k$, it suffices to show that for each crossing point c that X visits, at most one vertex of $V(G) \setminus V_G(X)$ is added to S. We break this down into 2 cases.

Case 1: Suppose that X also visits at least one endpoint of the crossing at c. From (Ψ2.3), this endpoint is marked $\square$. From Observation 9, exactly two endpoints of the crossing are marked $\square$. Therefore, from the construction of S, at most one vertex of $V(G) \setminus V_G(X)$ is added to S.

Case 2: Suppose that X visits no endpoint of the crossing at c. From (Ψ2.4), the crossing is a part of a butterfly crossing with the endpoints marked $\square$ as the axial vertices. Let $\{(u, v), (w, x)\}$ and $\{(x, y), (v, z)\}$ be the pair of crossings forming the butterfly crossing. Let $(u, x), (w, v)$ and $(x, z), (v, y)$ be their kite edges respectively, and let c and c' be their crossing points respectively (Figure 3.9). Note that v, x are the axial vertices of the butterfly crossing, and $\beta(x) = \square = \beta(v)$. As X does not visit any of endpoint of the crossing $\{(u, v), (w, x)\}$, it must visit c'. From Observation 9, $\beta(z) = \triangle = \beta(y)$. From ($\Psi$2.3), this implies that X does not visit the endpoints y, z of the crossing $(x, z), (v, y)$. Therefore, X does not visit any endpoint of the crossing $\{(x, y), (v, z)\}$ either. From the construction of S, both x, v are added to S. Therefore, the set of crossing points that

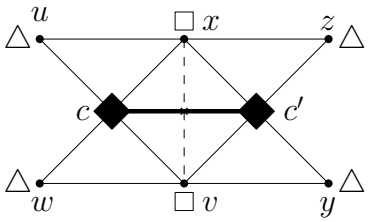


Figure 3.9: An illustration for Case 2

X visits without visiting any other endpoint of the crossing can be partitioned into pairs of crossing points at butterfly crossings, and for each such pair, at most 2 vertices of $V(G) \setminus V_G(X)$ are added to S.

The two cases above show that for each crossing point that X visits, at most one vertex of $V(G) \setminus V_G(X)$ is added to S. This proves that $|S| \leq k$.

Now, we show that S can be computed in $O(k)$ time. (We assume that we are given the marking function β along with the constrained separating cycle. In fact, as we will see in Chapter 5, the algorithm that outputs a separating cycle also outputs the corresponding marking function.) Note that all vertices of S either belong to $V_G(X)$ or belong to the endpoints of a crossing through which X passes. Since a crossing has only 4 endpoints, for each crossing point through which X passes, the endpoints to be added to S can be computed in $O(1)$ time. Therefore, S can be computed in $O(k)$ time. This proves Theorem 2.

Chapter 4

Separating cycles from separating sets

In this chapter, we prove Theorem 3, which states that if a bowtie 1-plane graph G contains a separating set of size k, then $\Lambda(G)$ contains a constrained separating cycle of length at most $2k$. Nearly all the work will be to find the cycle; finding the marking function will be very easy given the separating set. Figure 4.1 shows the steps that we will follow to obtain the constrained separating cycle X in $\Lambda(G)$. (The terms used in the figure will be defined later in the chapter.)

Throughout this chapter, we let S be a separating set of G of size k, and ϕ_1, ϕ_2 be two flaps of $G - S$. We assume that S is a minimal separating set, since otherwise we can choose a subset of S that is a minimal separating set and apply the proof to it.

4.1 Refined 1-plane graphs

To prove Theorem 3, it is convenient to consider a "refinement" of G with respect to S, ϕ_1, and ϕ_2 that restricts the crossings to have some special structures. The refined graph is denoted by $G_{\text{ref}} := G_{\text{ref}}(S, \phi_1, \phi_2)$, and is obtained as follows.

1. For each crossing $\{(u, v), (w, x)\}$, if $(u, x) \notin E(G)$ and u, x do not belong to ϕ_1 and ϕ_2 respectively, add the kite edge (u, x). Then update the graph, the flaps ϕ_1 and ϕ_2, and repeat the above steps until no new edge can be added. Let G^+ be the graph obtained after this step (Figure 4.2).

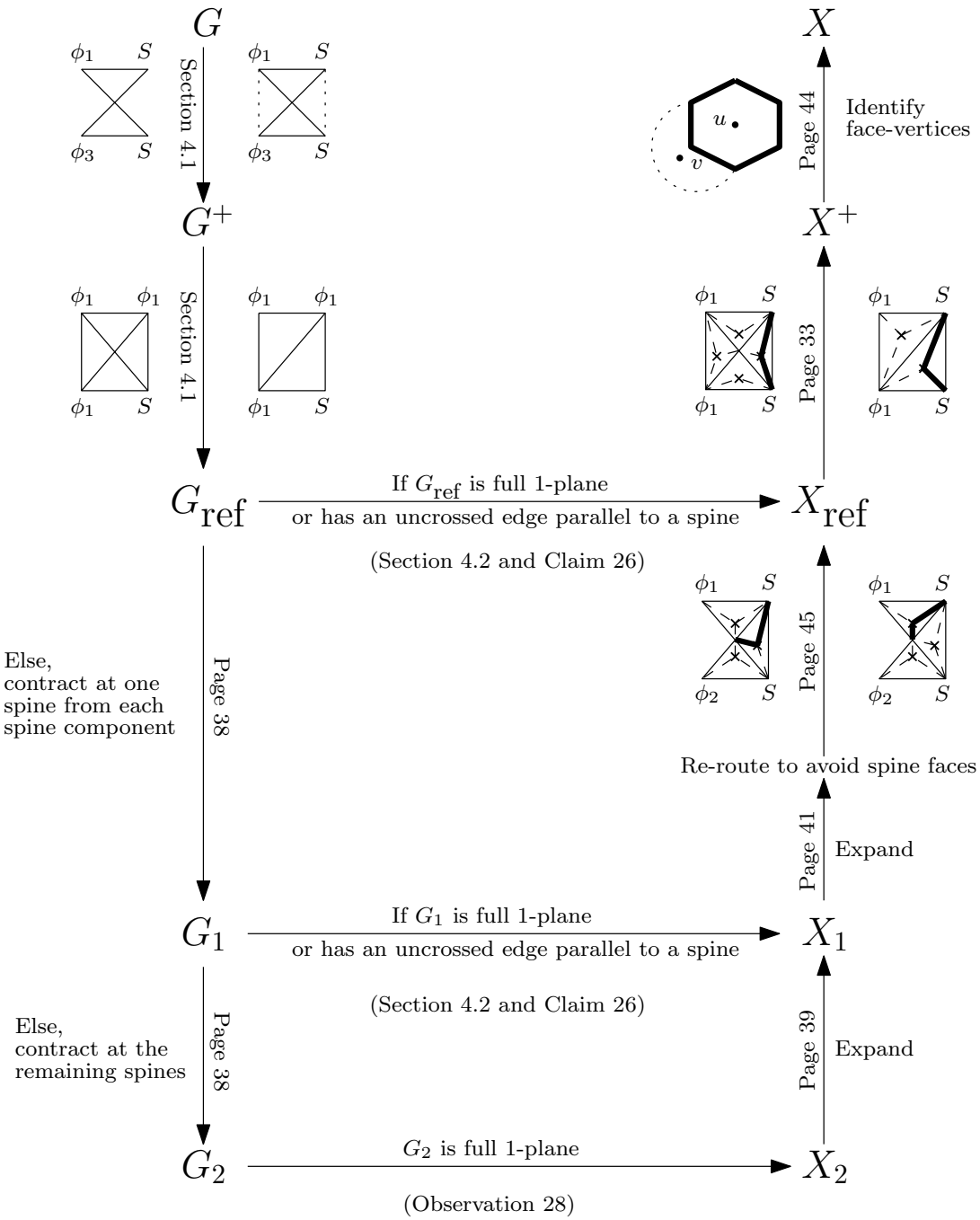


Figure 4.1: A flowchart showing the steps to obtain a constrained separating cycle in $\Lambda(G)$.

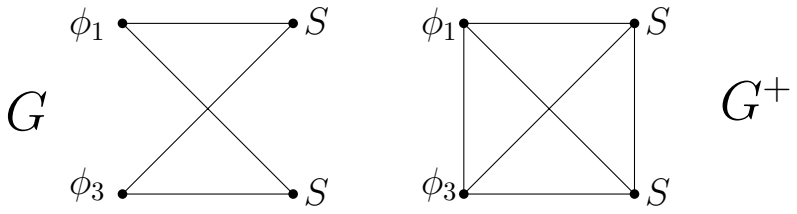


Figure 4.2: Transforming G to G^+

2. For each full crossing $\{(u, v), (w, x)\}$ of G^+, if one endpoint, say u, does not belong to S, then delete the crossing edge not incident with u, that is (w, x). (We will see soon that this step does not affect the flaps.) The graph obtained after applying this step to all full crossings is G_{ref} (Figure 4.3).

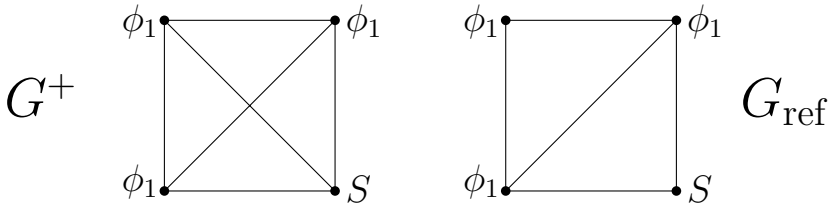


Figure 4.3: Transforming G^+ to G_{ref}

We now make a number of observations about G_{ref}.

Observation 11 follows from the assumption that G does not contain bigons and we add parallel edges only if they do not bound a kite face.

Observation 11. *G_{ref} does not contain bigons.*

Observation 12 follows from the transformation G^+ to G_{ref}.

Observation 12. *All the four endpoints of a full crossing of G_{ref} belong to S.*

Observation 13 shows that G_{ref} has no bowtie crossings, and the almost full crossings of G_{ref} are "special" in some sense.

Observation 13. *G_{ref} is an almost full 1-plane graph. Moreover, for any almost full crossing of G_{ref}, the spine vertices belong to S and the two wing tips belong to ϕ_1 and ϕ_2.*

Proof. Consider a bowtie crossing $\{(u,v),(w,x)\}$ of G with kite edges (u,w), (v,x). If vertices u,x do not belong to ϕ_1 and ϕ_2 respectively, then the edge (u,x) was added in G^+, and the crossing becomes full or almost full. If $u \in \phi_1$ and $x \in \phi_2$, then $v,w \in S$ since no path connects two flaps in $G-S$. This implies that the edge (v,w) was added in G^+, and the crossing becomes almost full. Therefore, every crossing is full or almost full in G^+. Since deleting a crossed edge of a full crossing does not create a bowtie crossing, every crossing is full or almost full in G_{ref}.

Now we prove the second part. Consider an almost full crossing $\{(u,v),(w,x)\}$ of G_{ref} with (u,x) as the spine. Since v and w are not adjacent, they must belong to ϕ_1 and ϕ_2. Since u and x are common neighbours of v and w, the vertices u,x belong to S. □

Claim 14. *S is a minimal separating set of G_{ref}.*

Proof. First, we show that S is a separating set of G_{ref}. By construction, the edges added to G do not connect vertices of ϕ_1 and ϕ_2. Therefore, S is a separating set of G^+. Deleting edges cannot make a graph more connected. Hence, S is a separating set of G_{ref}.

Now we show that S is a minimal separating set of G_{ref}. Since S is a minimal separating set of G, every vertex of S has a neighbour in every flap of $G-S$ (Observation 5). Note that in the transformation from G to G^+, we only add edges to G. Therefore, each vertex of S has a neighbour in each flap of G^+-S. This shows that S is a minimal separating set of G^+ (Observation 5). Consider the transformation from G^+ to G_{ref}, and consider a vertex $s \in S$; we must show that s is still adjacent to all flaps in G_{ref}.

Let (w,x) be an edge that is deleted. Then (w,x) is crossed by an edge (u,v) of G^+, where $\{(u,v),(w,x)\}$ is a full crossing and $u \notin S$. Deleting (w,x) can affect whether s is adjacent to all flaps only if deleting (w,x) creates a new flap or if $s \in \{w,x\}$ and the other vertex of $\{w,x\}$ belongs to a flap.

We first show that deleting (w,x) creates no new flap. Suppose otherwise, for contradiction. Then $w,x \notin S$. However, since $u \notin S$, all of x, u and w must belong to the same flap due to edges $(x,u),(w,u)$ of G^+. But edges $(x,u),(u,w)$ remain also in $G^+ \setminus \{w,x\}$, so x,w remain in the same flap. This is a contradiction.

Now assume $s \in \{w,x\}$, say $s=w$, and x belongs to a flap. Then u,x are in the same flap of G^+-S since $u \notin S$ and $(u,x) \in E(G^+)$. Since w is adjacent to u, the vertex $s=w$ still has a neighbour in the flap that contains x.

Since each vertex of S in $G^+-\{(w,x)\}$ has a neighbour in each flap of G^+-S, the set S is a minimal separating set of $G^+-\{(w,x)\}$ (Observation 5). By induction on the set of all the deleted edges, S is a minimal separating set of G_{ref}. □

Observation 15 shows that no edge within ϕ_1 or ϕ_2 is crossed in G_{ref}.

Observation 15. *The induced embeddings of the flaps ϕ_1 and ϕ_2 of $G_{\text{ref}} - S$ are planar.*

Proof. Let e be an edge of G_{ref} that is crossed. If e is a part of a full crossing, both of its endpoints belong to S (Observation 12), and if e is a part of an almost full crossing, one of its endpoints belong to S (Observation 13). Thus, no edge of ϕ_1 or ϕ_2 is crossed. □

Recall that if $\{(u, v), (w, x)\}$ is an almost full crossing of G_{ref} with $(u, w) \notin E(G_{\text{ref}})$, then (v, x) is called a spine of the crossing. Let a *spine component* be a maximal connected subgraph of G_{ref} that consists of only spines.

Claim 16. *For each spine component H of G_{ref}, there are at most two almost full crossings whose spines belong to H.*

Proof. Suppose, for contradiction, that there are three almost full crossings whose spines belong to H. We will show a $K_{3,3}$ minor in the planarisation of G_{ref}, say G^p_{ref}. Let x_1, x_2 and x_3 be three dummy vertices in G^p_{ref} at the crossing points of the three almost full crossings. From Observation 13, vertex x_i has neighbours in ϕ_1, ϕ_2 and H. Since ϕ_1 and ϕ_2 are connected and drawn without crossings (Observation 15), we can contract each of them into a single vertex, say v_1 and v_2, respectively. Since the edges of H are spines, they are uncrossed. As they are connected in G_{ref}, we can contract H to a single vertex h in G^p_{ref}. Consequently, each x_i is adjacent to v_1, v_2 and h. This is a $K_{3,3}$ minor in G^p_{ref}, which is a contradiction. □

Corollary 17. *Every spine component of G_{ref} has at most two spines.*

Corollary 18. *If a butterfly crossing of G_{ref} has a spine, then the spine itself forms a spine component.*

4.2 Plane and full 1-plane graphs

We first prove Theorem 3 for plane and full 1-plane graphs. That is, we prove that if a plane or a full 1-plane graph G has a separating set of size k, then $\Lambda(G)$ contains a separating cycle of length at most $2k$. Since a plane graph is also full 1-plane, it is sufficient to prove the theorem for full 1-plane graphs.

We first introduce some definitions. Let $G_{\text{ref}} := G_{\text{ref}}(S, \phi_1, \phi_2)$, and v be a vertex of G_{ref}. Let $\rho_{\text{ref}}(v)$ denotes the clockwise order of edges of G_{ref} at v. An edge (v, u) is said

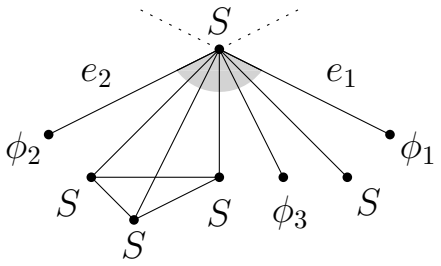


Figure 4.4: e_1 and e_2 form a closest (ϕ_1, ϕ_2)-pair.

to lie *between* (v, w_1) and (v, w_2) in $\rho_{\text{ref}}(v)$ if $\rho_{\text{ref}}(v)$ contains $\langle (v, w_1), (v, u), (v, w_2) \rangle$ as a subsequence. A pair of edges $(v, w_1), (v, w_2)$ forms a *closest* (ϕ_1, ϕ_2)*-pair* in $\rho_{\text{ref}}(v)$ if $v \in S$, $w_1 \in \phi_1$, $w_2 \in \phi_2$, and there is no edge between (v, w_1) and (v, w_2) in $\rho_{\text{ref}}(v)$ that is incident with a vertex of $\phi_1 \cup \phi_2$. Note that there can be edges between (v, w_1) and (v, w_2) that connect v to vertices of S or to flaps other than ϕ_1, ϕ_2 (Figure 4.4).

A *fence* of $\Lambda(G)$ is a cycle in $R(G)$ such that there is a vertex of G inside and a vertex of G outside the cycle. Note that a fence is a separating cycle of $\Lambda(G)$ that separates the vertices of G inside the cycle from the vertices of G outside, but not every separating cycle is a fence.

We now give a construction to obtain a constrained separating cycle in $\Lambda(G)$ of length at most $2k$. For this, we first construct a fence in the radial planarisation of G_{ref}, that is $\Lambda(G_{\text{ref}})$, and then show how to map the fence to a constrained separating cycle in $\Lambda(G)$.

Consider the graph $\Lambda(G_{\text{ref}})$. Mark all the face vertices of $\Lambda(G_{\text{ref}})$ whose faces contain at least two angles at vertices of S, and mark all the vertices of S on the face. (Recall from Chapter 2 that an *angle* at a vertex or crossing point in a 1-planar embedding refers to an angle at the corresponding vertex or crossing point in the planarisation of the embedding. In particular, the edges that define angles are edges of the planarisation and hence may connect a vertex and a crossing point, so may be 'half' of an edge of G.)

Observation 19 follows immediately from the procedure above.

Observation 19. *Each marked face vertex has at least two edges in $R(G_{\text{ref}})$ connecting it to marked vertices of S on the face.*

Observation 20 will be used to prove Claim 21.

Observation 20. *Let $\langle u, v, w \rangle$ be an angle of some face F of G_{ref} with $v \in S$. If (a) $u \in S$ or $w \in S$, or (b) u and w are vertices of G that belong to different flaps of $G_{\text{ref}} - S$, then the face vertex of F is marked.*

Proof. If $u \in S$ or $w \in S$, then F has two vertices of S (including v), and therefore the face vertex of F is marked. Suppose that u and w belong to different flaps of $G_{\text{ref}} - S$. Consider a directed walk along the boundary of F, starting from the edge (v, w) in the direction v to w. Since F has at least three vertices of G, it is not a kite face, and since G_{ref} is full 1-plane, F does not contain any crossing points. Since u and w belong to different flaps, the walk must switch from "in the flap of w" to "not in the flap of w". Since no vertices in different flaps can be adjacent, the walk either sees a vertex of S different from v before seeing u, or sees v again at another angle of the face. This implies that F has at least two angles at vertices of S. Therefore, the face vertex of F is marked. □

Claim 21 shows that each marked vertex of S is adjacent to two marked face vertices in a particular way. For a vertex $v \in G$, we let $\rho_\Lambda(v) := \rho_{\Lambda(G_{\text{ref}})}(v)$, which is the clockwise order of edges at v in $\Lambda(G_{\text{ref}})$. For an edge (v, t) of G_{ref}, let $(v, \bar{t})$ refer to the edge in $\Lambda(G_{\text{ref}})$ that is incident with v and corresponds to (v, t). That is, $\bar{t} = t$ if (v, t) is uncrossed; else $\bar{t}$ is the dummy vertex on (v, t).

Claim 21. *Let v be a marked vertex of S. Let (v, t_1) and (v, t_2) be two edges of G_{ref} at v such that $t_1 \in \phi_1$ and $t_2 \in \phi_2$ (these exist by Observation 5). Then there exist marked face vertices f_1, f_2 adjacent to v in $\Lambda(G_{\text{ref}})$ such that (v, f_1) is between $(v, \bar{t_1})$ and $(v, \bar{t_2})$, and (v, f_2) is between $(v, \bar{t_2})$ and $(v, \bar{t_1})$ in $\rho_\Lambda(v)$.*

Proof. Since v is a marked vertex of S, there exists an edge (v, f) in $R(G_{\text{ref}})$ where f is a marked face vertex. Assume without loss of generality that (v, f) is between $(v, \bar{t_2})$ and $(v, \bar{t_1})$ in $\rho_\Lambda(v)$. Let $f_2 := f$. We need to show that there is a marked face vertex f_1 such that (v, f_1) is between $(v, \bar{t_1})$ and $(v, \bar{t_2})$ in $\rho_\Lambda(v)$. Since $t_1 \in \phi_1$ and $t_2 \in \phi_2$, there is a pair of edges $(v, w_1), (v, w_2)$ between (v, t_1) and (v, t_2) in $\rho_{\text{ref}}(v)$ that form a closest (ϕ_1, ϕ_2)-pair. If we show that there is a marked face vertex f_1 between $(v, \bar{w_1})$ and $(v, \bar{w_2})$, we are done.

Case 1. Suppose that (v, w_1) and (v, w_2) are consecutive in $\rho_{\text{ref}}(v)$. Since $w_1, w_2 \notin S$, both (v, w_1) and (v, w_2) are uncrossed (Observation 12). Therefore, $\langle w_1, v, w_2 \rangle$ form an angle in some face F of G_{ref}. By Observation 20, the face vertex of F is marked. By letting f_1 be this face vertex, we are done.

Case 2. Suppose that (v, w_1) and (v, w_2) form a closest (ϕ_1, ϕ_2)-pair but are not consecutive in $\rho_{\text{ref}}(v)$. Then there is an edge (v, v') between (v, w_1) and (v, w_2) in $\rho_{\text{ref}}(v)$. Choose (v, v') to be the first edge that comes after (v, w_1). Note that $v' \notin \phi_1 \cup \phi_2$ since (v, w_1) and (v, w_2) form a closest (ϕ_1, ϕ_2)-pair.

First we show that (v, v') is uncrossed. Suppose, for contradiction, that (v, v') is crossed by an edge (u, u'). Since the crossing is full, there exist kite edges (v, u) and (v, u'). Without

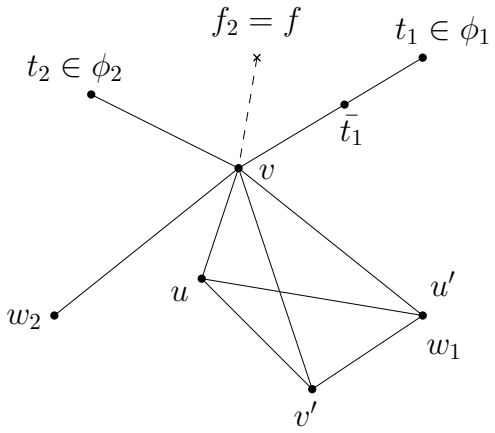


Figure 4.5: Illustration for Case 2

loss of generality, suppose that (v, u) comes after (v, u') in $\rho_{\text{ref}}(v)$ (Figure 4.5). Since (v, v') was chosen to be the first edge after (v, w_1) in $\rho_{\text{ref}}(v)$, $u' = w_1$. This contradicts Observation 12 since $w_1 \notin S$.

As (v, v') and (v, w_1) are both uncrossed (Observation 12), the edges (v, w_1) and (v, v') form an angle in some face F of G_{ref}. Since $v' \notin \phi_1 \cup \phi_2$ and $w_1 \in \phi_1$, by Observation 20, the face vertex of F is marked. By letting f_1 be this face vertex, we are done. □

Constructing a fence in $\Lambda(G_{\text{ref}})$. Now, we show how to construct a fence X_{ref} in $\Lambda(G_{\text{ref}})$. Construct a simple path $P = v_1 \dots v_k$ that alternates between marked face vertices and marked vertices of S that is maximal in the following sense: $v_k \in S$, and there does not exist a marked face vertex, say v_{k+1}, and a marked vertex of S, say v_{k+2}, such that $P \cup \{(v_k, v_{k+1}), (v_{k+1}, v_{k+2})\}$ is a simple path. Since $v_k \in S$, there exist vertices $t_1 \in \phi_1$ and $t_2 \in \phi_2$ that are adjacent to v_k (Observation 5). From Claim 21, there are marked face vertices f_1 and f_2 such that (v_k, f_1) is between $(v_k, \bar{t_1})$ and $(v_k, \bar{t_2})$ and (v_k, f_2) is between $(v_k, \bar{t_2})$ and $(v_k, \bar{t_1})$ in $\rho_\Lambda(v)$. Without loss of generality, we may assume that the edge (v_k, f_1) does not belong to P. Let $f := f_1$.

- Suppose that $f = v_i \in P$, for some $1 \leq i \leq k-1$ (Figure 4.6a). Consider the cycle $X_{\text{ref}} := v_i v_{i+1} \dots v_k v_i$. Since $t_1 \in \phi_1$ and $t_2 \in \phi_2$, so they cannot be in S, and hence are not on X_{ref}. Hence, X_{ref} is fence separating t_1 from t_2 since they are on opposite sides of the cycle.
- Suppose that $f \notin P$. Since P is maximal, and each face vertex is adjacent to two marked vertices of S (Observation 19), f has an edge to a vertex $v_i \in P$, for some

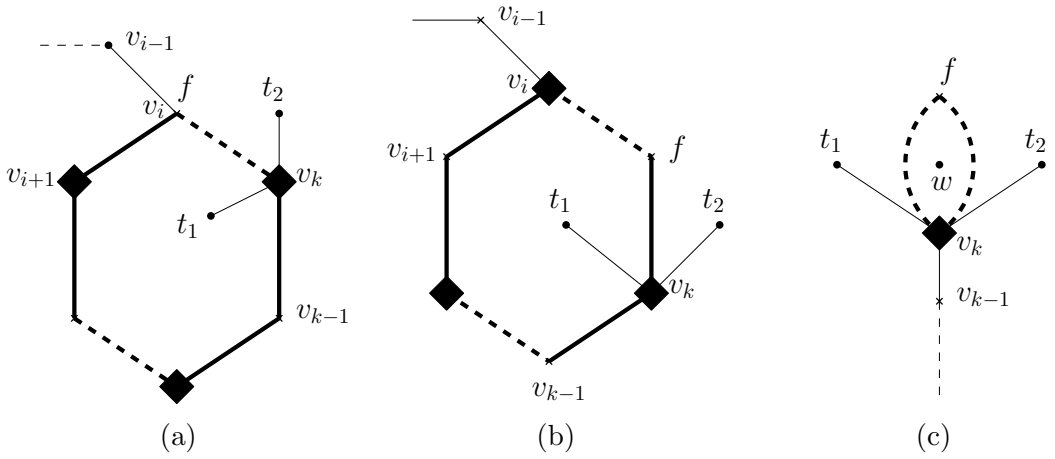


Figure 4.6: Constructing a fence in $\Lambda(G)$

$1 \leq i \leq k$. If $1 \leq i \leq k-1$ (Figure 4.6b), then $X_{\text{ref}} := v_i v_{i+1} \dots v_k f v_i$ is a fence separating t_1 from t_2. If $i = k$, then there must exist a vertex $u \in \Lambda(G)$ inside the 2-cycle $v_k f v_k$ because $R(G_{\text{ref}})$ does not contain bigons (Observation 7). We now show that there must be a vertex w of G inside the 2-cycle $v_k f v_k$ (Figure 4.6c). If u is a vertex of G, we are done by simply setting $w := u$. Otherwise, if u is a crossing point of G, then all four endpoints of the crossing (possibly excluding v_k) must lie inside the cycle $v_k f v_k$ since $R(G)$ is a plane graph (Observation 6). Set w to be one of these endpoints that lie inside the cycle $v_k f v_k$. Therefore, $X_{\text{ref}} := v_k f v_k$ is a fence separating w from t_1 (and t_2).

Observation 22 follow immediately from the construction of X_{ref}. (Recall the notation that $V_G(X)$ denotes the vertices of G on X.)

Observation 22. *X_{ref} is a fence of $\Lambda(G_{\text{ref}})$, $V_G(X_{\text{ref}}) \subseteq S$ and X_{ref} does not visit any crossing points.*

Observation 23 follows by noting that $|S| \leq k$, and that X_{ref} alternates between vertices of S and face vertices.

Observation 23. *$|X_{\text{ref}}| \leq 2k$.*

From X_{ref} to a fence in $\Lambda(G)$. We now show how to map X_{ref} to a fence X that is a constrained separating cycle of $\Lambda(G)$ with $|X| \leq 2k$. The mapping is described algorithmically by giving a step-by-step transformation from $\Lambda(G_{\text{ref}})$ to $\Lambda(G)$. This transformation essentially involves undoing the steps that were done to obtain G_{ref} from G.

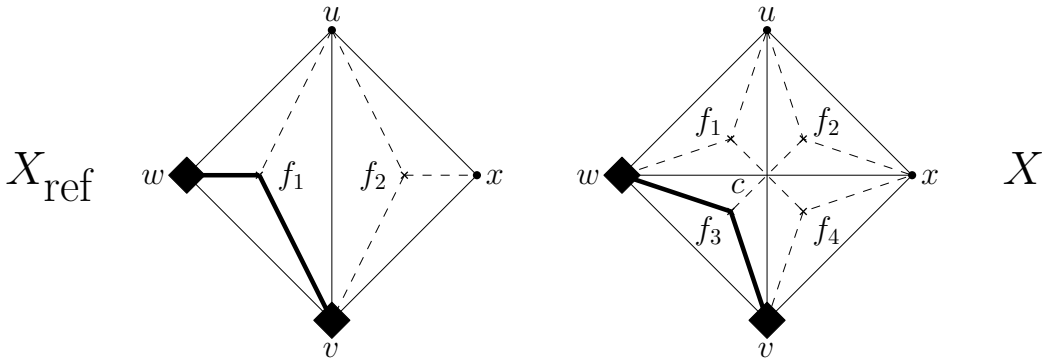


Figure 4.7: Transforming X_{ref} to X

Lemma 24. *If G is full 1-plane, then any fence X_{ref} of G_{ref} that does not visit crossing points can be converted into a fence X of G that does not visit crossing points with $|X| = |X_{\text{ref}}|$.*

Proof. Recall that G_{ref} was obtained from G by inserting some kite edges to get G^+, and then by deleting crossed edges of some full crossings of G^+. Since G is full 1-plane, $G^+ = G$, and $\Lambda(G^+) = \Lambda(G)$. Since $G = G^+$, we only need to add to G_{ref} all those crossed edges of G that were deleted during refinement to get back G.

Suppose that an edge (w, x) of G was deleted. Then (w, x) was part of a full crossing $\{(u, v), (w, x)\}$ of G in which at least one of u, v, say u, did not belong to S. This implies that u does not belong to X_{ref} (Observation 22). In $G - \{(w, x)\}$, we have faces (u, w, v) and (u, x, v); let f_1 and f_2 be the two corresponding face vertices of $\Lambda(G^+)$. Nothing needs to be changed if $f_1, f_2 \notin X_{\text{ref}}$, so suppose that $f_1 \in X_{\text{ref}}$. Since $u \notin X_{\text{ref}}$, this implies that $w, v \in X_{\text{ref}}$. Then X_{ref} can modified at the corresponding full crossing of G to use the face vertex of the kite face incident with the edge (w, v) (Figure 4.7). This local modification of X_{ref} at the full crossing does not change the set of vertices of G that lie inside and outside of X_{ref}. Therefore, X_{ref} still remains a fence. Furthermore, $|X_{\text{ref}}| \leq 2k$ since one vertex was added and one removed. Similarly, if $f_2 \in X_{\text{ref}}$, then $x, v \in X_{\text{ref}}$, and X_{ref} can be modified to use the face vertex at the kite face incident with (v, x). When the above steps are repeated for all face vertices at the crossings that were deleted during refinement, we get back $\Lambda(G)$, and a cycle corresponding to X_{ref}, say X. □

Observation 25. *If G is a full 1-plane graph, then there is a fence X of $\Lambda(G)$ such that $V_G(X) \subseteq S$, X does not visit any crossing points, and $|X| \leq 2k$.*

We now show that X is a constrained separating cycle of $\Lambda(G)$. As X is a fence of

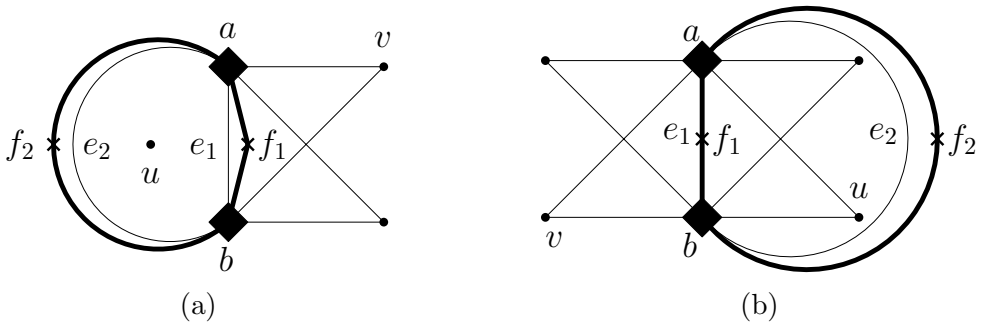


Figure 4.8: Constructing X_{ref} when there is an uncrossed edge parallel to a spine

$\Lambda(G)$, it is separates the vertices of G inside the cycle from the vertices of G outside. Therefore, ($\Psi 1$) holds. Since X does not visit any crossing points (Observation 25), ($\Psi 2$) holds vacuously for any marking function. Therefore, X is a constrained separating cycle of $\Lambda(G)$. This proves Theorem 3 for plane and full 1-plane graphs.

4.3 Almost full 1-plane and bowtie 1-plane graphs

In this section, we prove Theorem 3 for almost full 1-plane and bowtie 1-plane graphs. That is, we prove that if an almost full 1-plane or a bowtie 1-plane graph G contains a separating set of size k, then $\Lambda(G)$ has a constrained separating cycle of length at most $2k$. Since an almost full 1-plane graph is also bowtie 1-plane, it is sufficient to prove the theorem for bowtie 1-plane graphs. We prove the theorem by first constructing a fence X_{ref} of length at most $2k$ in $\Lambda(G_{\text{ref}})$, and then mapping the fence to a constrained separating cycle X in $\Lambda(G)$. (Recall that we had assumed the given separating set S to be minimal, and $G_{\text{ref}} := G_{\text{ref}}(S, \phi_1, \phi_2)$, where ϕ_1 and ϕ_2 are two fixed flaps of $G - S$.)

A special case. We already know how to find a constrained separating cycle if G_{ref} is full 1-plane. We now dispense here with another special case which will be needed later.

Claim 26. *If G_{ref} has two parallel edges e_1 and e_2, where e_1 is the spine of an almost full crossing and e_2 is an uncrossed edge, then $\Lambda(G_{\text{ref}})$ has a fence X_{ref} such that $|X_{\text{ref}}| \leq 2k$, $V_G(X_{\text{ref}}) \subseteq S$ and X_{ref} does not visit any crossing points.*

Proof. Let a, b be the endpoints of e_1 and e_2. Since e_1 is a spine edge, $a, b \in S$ (Observation 13). Since G_{ref} does not contain bigons (Observation 11), there are vertices u and v inside

and outside the 2-cycle formed by e_1 and e_2. Since e_1 and e_2 are uncrossed, a and b form a separating set of G_{ref} that separate u from v; since S is minimal, $S = \{a, b\}$ and $k = 2$.

Since e_1 is a spine, e_2 cannot be a spine of a butterfly crossing because otherwise the spine component $\{e_1, e_2\}$ would have three almost full crossings, contradicting Claim 16. So e_2 also exists in $\Lambda(G_{\text{ref}})$ and has two incident faces in G_{ref}. Let f_2 be the face vertex of $\Lambda(G_{\text{ref}})$ that belongs to the face incident with e_2 and is outside the 2-cycle (e_1, e_2). If e_1 is not the spine of a butterfly crossing, let f_1 be the face vertex in the spine face of the crossing (Figure 4.8a); otherwise let f_1 be the face vertex at the butterfly crossing (Figure 4.8b). The 4-cycle $X_{\text{ref}} := (a, f_1, b, f_2, a)$ is a fence that separates u, v, and such that $|X_{\text{ref}}| \leq 2k$, where $k = 2$. □

Contractions. From now on, we assume that G_{ref} has no uncrossed edge parallel to a spine. From Observation 13, all crossings of G_{ref} are either full crossings or almost full crossings. The plan is to remove all the almost full crossings of G_{ref} so that the resulting graph becomes full 1-plane. We achieve this by contracting all spine faces of G_{ref} into vertices; i.e., we contract all the edges in the planarisation of G_{ref} that correspond to the boundary of spine faces of G_{ref}. However, care needs to be taken to ensure that the resulting drawing is good and loopless so that we can apply the method for finding constrained separating cycles from Section 4.2 to the resulting full 1-plane graph.

We distinguish two types of contractions to remove almost full crossings. Let (u, v) be a spine in G_{ref}. From Claim 16, there are at most two almost full crossings with (u, v) as the spine. Let (u, v, c) be the spine face of one such crossing with crossing point c. If (u, v, c) is the only kite face incident with (u, v), then a *triangular contraction at* (u, v) deletes (u, c) and then contracts (v, c) and (v, u) to a single vertex z_{uvc} (Figure 4.9). On the other hand, if there is another kite face (u, v, d) (of a full crossing or an almost full crossing) incident with (u, v), then a *quadrangular contraction at* (u, v) deletes $(u, c), (u, d)$ and contracts (v, c), (v, d), (v, u) to a single vertex z_{uvcd} (Figure 4.10 and 4.11). We use the term *contracted vertex* to mean the vertex obtained after a triangular or a quadrangular contraction; the remaining vertices are *uncontracted vertices*.

Claim 27. *If G_{ref} has no uncrossed edge parallel to a spine, then a triangular or a quadrangular contraction at a spine (u, v) of G_{ref} produces a loopless good 1-planar drawing.*

Proof. Let (u, v, c) be a spine face incident with (u, v). Let D be the drawing obtained by either a triangular or a quadrangular contraction at (u, v). (D is essentially the same drawing as G_{ref} except at the spine face that undergoes contraction.) Since either type of

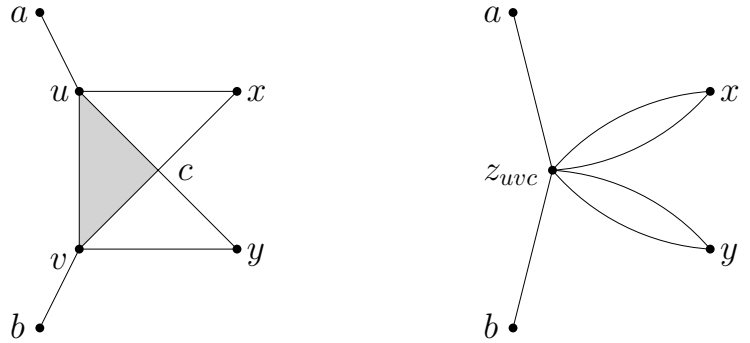


Figure 4.9: Triangular contraction

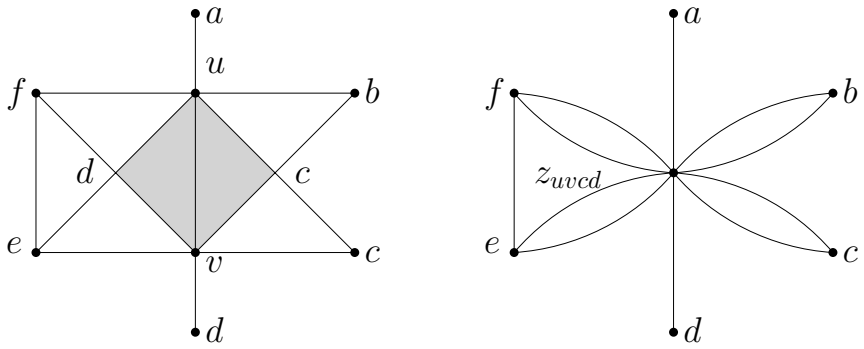


Figure 4.10: Quadrangular contraction at a spine incident with a full crossing

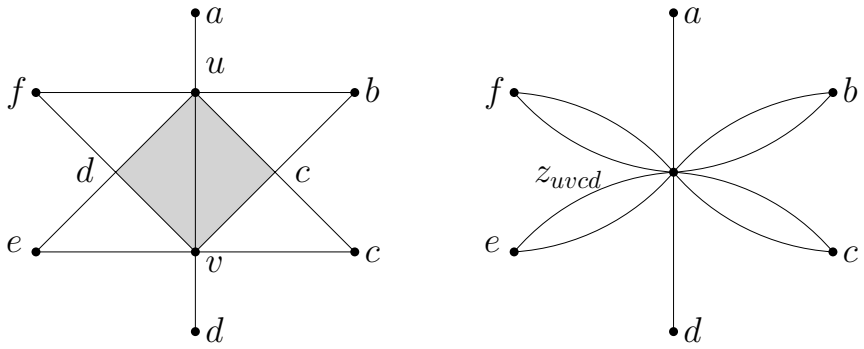


Figure 4.11: Quadrangular contraction at a butterfly crossing

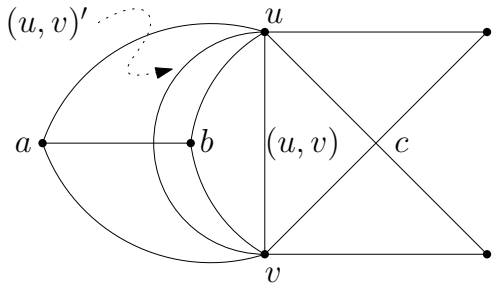


Figure 4.12: A contraction at a spine does not create loop-crossings

contraction does not create new crossing point, D stays 1-planar. Let z be the contracted vertex in D.

To show that D has a good drawing, the only non-trivial property that we need to verify is that no two edges incident with z cross. Suppose, for contradiction, that there are two edges (a,z) and (b,z) crossing at some point, say x. Since the drawing of G_{ref} was good, this crossing must correspond in G_{ref} to a crossing $\{(a,u),(b,v)\}$ with x as the crossing point. Since $u,v \in S$, the crossing has a kite face bounded by u, v and x. However, by our assumption that G_{ref} has no uncrossed edge parallel to a spine, it follows that (u,v,x) and (u,v,c) must be two kite faces incident with the common kite edge (u,v). This implies that the contraction at (u,v) must have been quadrangular, but then this contraction eliminates the crossing point x, which is a contradiction.

To show that D is loopless, we only need to show that there is no loop at z. Suppose, for contradiction, that there is a loop at z. Then this loop must be crossed, since otherwise the loop corresponds to an uncrossed edge parallel to (u,v). Let (a,b) be the edge that crosses the loop. This implies that G_{ref} contains a crossing $\{(a,b),(u,v)'\}$, where $(u,v)'$ is an edge parallel to (u,v). Since (u,v,c) is a spine face, $u,v \in S$ (Observation 13). Since $u,v \in S$, the crossing $\{(a,b),(u,v)'\}$ must be full (Observation 13), and $a,b \in S$ (Observation 12). Therefore, $\{a,u,b,v\} \subseteq S$. We shall arrive at a contradiction by showing that S is not a minimal separating set. Assume without loss of generality that b lies inside the triangle (u,v), (v,a), (a,u) (Figure 4.12). Since $|S| \geq 4$, G_{ref} has at least 6 vertices. Let y be a vertex of G_{ref} that is distinct from $\{a,u,b,v\}$. The vertex y cannot lie inside any of the four kite faces of the crossing $\{(a,b),(u,v)'\}$. If y lies inside the triangle (u,v), (v,b), (b,u), then $\{u,v,b\}$ is a separating set (since (u,v), (v,b), (b,u) is an uncrossed triangle) that separates a and y. If y lies outside the triangle (u,v), (v,b), (b,u), then $\{u,v,a\}$ is a separating set (since (u,v), (v,a), (a,u) is an uncrossed triangle) that separates b and y. Either way, this contradicts the minimality of S. □

We now give a procedure to find a fence X_{ref} in $\Lambda(G_{\text{ref}})$ using repeated triangular and quadrangular contractions at spines. (Refer also to Figure 4.1.)

(C1) For each spine component of G_{ref}, do a triangular or a quadrangular contraction at exactly one spine, and delete any resulting bigons. Let G_1 be the resulting graph. Let S_1 be the set of all contracted vertices of G_1 together with the uncontracted vertices of G_1 that belonged to S in G_{ref}. Let this graph be G_1. If G_1 is full 1-plane or has an uncrossed edge parallel to a spine, stop further contractions; otherwise, go to (C2).

(C2) Now, there is only one spine left in each spine component in G_1 (Corollary 17). Do a triangular or a quadrangular contraction at each of the remaining spines, and delete any resulting bigons. Let G_2 be the resulting graph. Let S_2 be the set of all contracted vertices of G_2 together with the uncontracted vertices of G_2 that belonged to S_1 in G_1. Let this graph be G_2.

We need a few observations about the resulting graphs.

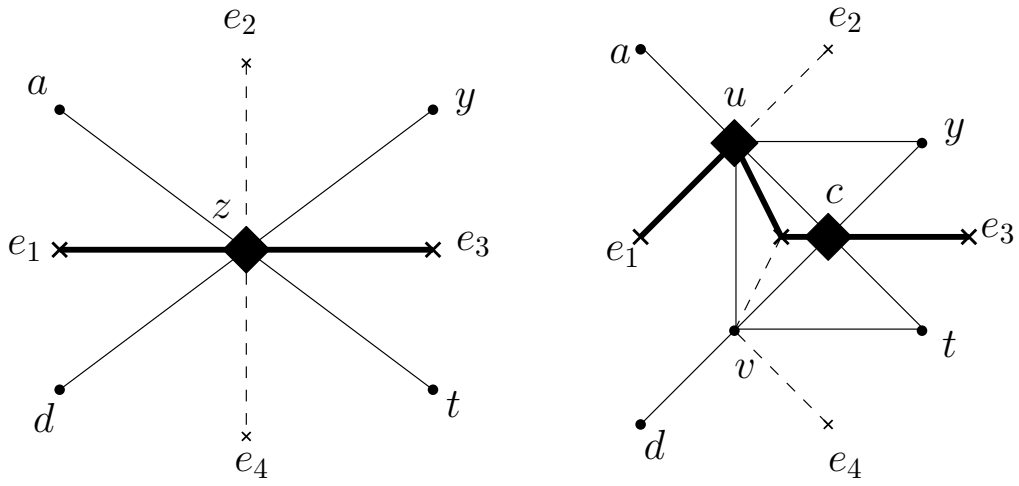
Observation 28. *G_2 is a full 1-plane graph.*

Observation 29. *G_1 has no butterfly crossings.*

Proof. The spine faces of a butterfly crossing in G_{ref} get contracted during (C1) since there can be no other spine in the same spine component (Corollary 18). Moreover, a triangular or a quadrangular contraction at a spine cannot create a new butterfly crossing as this would imply a spine component in G_{ref} with three spines, contradicting Claim 16. □

Observation 30. *S_1 (resp. S_2) is a minimal separating set of G_1 (resp. G_2).*

Proof. We show that S_1 is a minimal separating set of G_1. The proof that S_2 is a minimal separating set of G_2 follows similarly. Every edge (u, v) of G_{ref} with $u \notin S$ and $v \in S$ corresponds to an edge (u, w) of G_1, where either $w = v$, or w is a contracted vertex resulting from a contraction at some spine component containing v. Therefore, deleting v from G_{ref} has the same effect upon u as deleting w from G_1. Therefore, the flaps of $G_{\text{ref}} - S$ are the same as the flaps of $G_1 - S_1$. By Observation 5, S_1 is a minimal separating set of G_1. □



(a) X_2 in $\Lambda(G_2)$ (b) X_1 in $\Lambda(G_1)$

Figure 4.13: Extending X_2 to X_1: Case 1

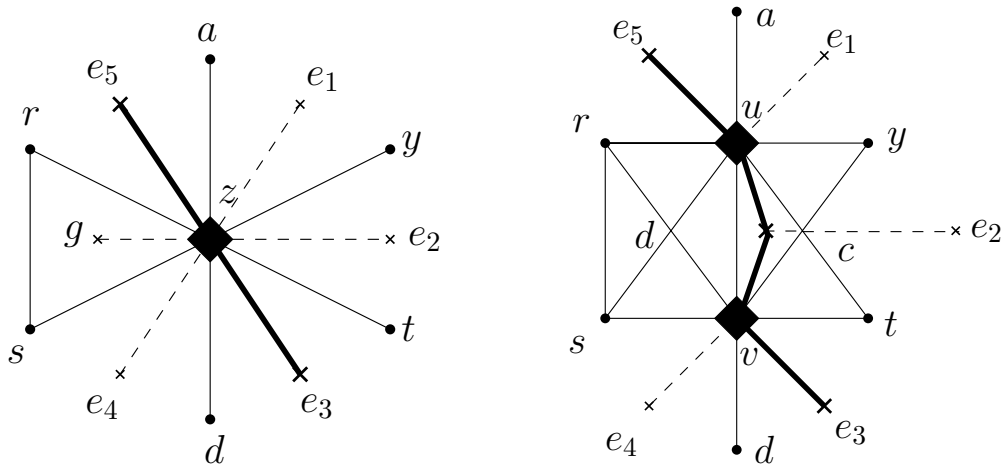
We now find a fence X_1 in $\Lambda(G_1)$ or a fence X_2 in $\Lambda(G_2)$ as follows. If G_1 is full 1-plane, we construct a fence X_1 using the method in Section 4.2, and leave X_2 undefined. Else if G_1 has an uncrossed edge parallel to a spine, we construct a fence X_1 using the method in Claim 26, and leave X_2 undefined. Else, G_2 is full 1-plane (Observation 28), and we construct a fence X_2, again by the method in Section 4.2, and leave X_1 undefined. By construction, the cycles X_1 and X_2 have the following properties.

Observation 31. *For $i = 1, 2$, if X_i exists, then X_i is a fence of $\Lambda(G_i^-)$, $V_G(X_i) \subseteq S_i$, $|X_i| \leq 2|S_i|$, and X_i does not visit full crossing points of G_i^-.*

Proof. See Observation 25 and Claim 26. □

Expansions. Suppose that G_1 is neither full 1-plane nor has an uncrossed edge parallel to a spine. In this case, we constructed X_2 in $\Lambda(G_2)$ and left X_1 undefined. We show how to extend X_2 into a cycle X_1 in $\Lambda(G_1)$.

(E1) Add the deleted bigons of G_2, and expand each contracted vertex of G_2 to get G_1. The extension of X_2 into a fence X_1 of $\Lambda(G_1)$ through the expanded spine faces depends upon the type of contraction that occurred. Let (u, v) be a spine of G_1 at which the contraction occurred. By Observation 29, only the following cases are possible.



(a) X_2 in $\Lambda(G_2)$ (b) X_1 in $\Lambda(G_1)$

Figure 4.14: Extending X_2 to X_1: Case 2

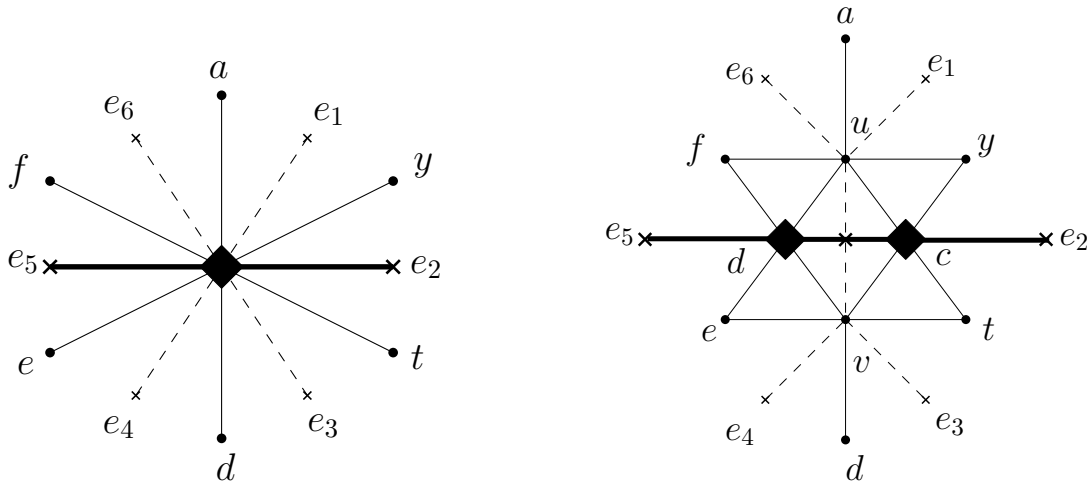
(a) **Case 1:** A triangular contraction occurred at (u, v).

Let (u, v, c) be the spine face of G_1 that is contracted into a vertex z. Each edge of $R(G_2)$ at z corresponds to an edge of $R(G_1)$ at u, v or c (Figure 4.13). Suppose that X_2 uses the edges e, e' at z. Let f, f' be the edges of $R(G_1)$ that correspond to e, e' of $R(G_2)$. Include f, f' in X_1. If f and f' do not have a common endpoint in $\{u, v\}$, connect the two edges via the face vertex of the spine face and its three edges to $\{u, v, c\}$, and add the connecting edges to X_1.

(b) **Case 2:** A quadrangular contraction occurred at (u, v), where (u, v) is the kite edge of a full crossing.

Suppose that (u, v, d) is a kite face of a full crossing $\{(u, s), (v, r)\}$ with crossing point d. Let (u, v, c) and (u, v, d) be the two kite faces of G_1 that are contracted into a vertex z. The kite face (r, s, d) remains a face after the contraction, so (r, s, z) is a face of G_2; let g be the corresponding face vertex of $R(G_2)$.

We can assume that X_2 does not use the edge (g, z) for the following reason. Without loss of generality, suppose that X_2 uses the edges $(s, g), (g, z)$. Let g' be the face vertex of $R(G_2)$ inside the other face incident with (s, z). Then X_2 can be re-routed to use the edges $(s, g'), (g'z)$ instead. Since we can assume that X_2 does not use the edge (g, z), each edge of $R(G_2)$ at z that X_2 could use corresponds to an edge of $R(G_1)$ at u, v or c (Figure 4.14). Suppose that X_2 uses the edges e, e' at z. Let f, f' be the edges of $R(G_1)$ that correspond to e, e' of $R(G_2)$. Include f, f' in X_1. If f and f' do not have a common endpoint in



(a) X_1 in $\Lambda(G_1)$ (b) X_{ref} in $\Lambda(G_{\text{ref}})$

Figure 4.15: Extending X_1 to X_{ref}: Case 3.

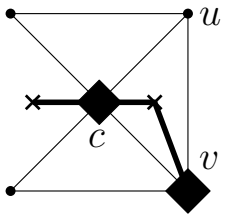
$\{u, v\}$, connect the two edges via the spine face vertex and its three edges to $\{u, v, c\}$, and add the connecting edges to X_1.

Now, we have a cycle X_1 in $\Lambda(G_1)$ which is obtained either by the construction in Section 4.2 applied to G_1 (when G_1 is full 1-plane) or Claim 26 (when G_1 has an uncrossed edge parallel to a spine), or by the extension of X_2 in $\Lambda(G_2)$ to X_1. We show how to extend X_1 into a cycle X_{ref} in $\Lambda(G_{\text{ref}})$.

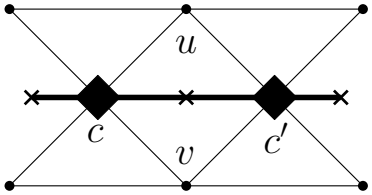
(E2) Add the deleted bigons of G_1, and expand each contracted vertex of G_1 to get G_{ref}. Let (u, v) be a spine of G_{ref} at which the contraction occurred. The extension of X_1 to X_{ref} is similar to (E1), and either Case 1 or Case 2 can occur. However, since G_{ref} can have butterfly crossings, there is also a Case 3.

Case 3: A quadrangular contraction occurred at (u, v), where (u, v) is the spine of a butterfly crossing in G_{ref}.

Let (u, v, c) and (u, v, d) be the kite faces that are contracted into a vertex z. Each edge of $R(G_1)$ at z corresponds to an edge of $R(G_{\text{ref}})$ at u, v, c or d (Figure 4.15). Suppose that X_1 uses the edges e, e' at z. Let f, f' be the edges of $R(G_{\text{ref}})$ that correspond to e, e' of $R(G_1)$. Include f, f' in X_{ref}. If f and f' do not have a common endpoint in $\{u, v\}$, connect the two edges via the face vertex in $\Lambda(G_{\text{ref}})$ at the spineless butterfly crossing and its four edges to $\{u, v, c, d\}$, and add the connecting edges to X_{ref}.



(a) X_{ref} passes through c, v but not through u (since u was previously contracted).



(b) X_{ref} passes through c, c', but not through u, v (since u, v were previously contracted).

Figure 4.16: The expansion step crucially ensures that $|X_{\text{ref}}| \leq 2k$.

By studying the cases of the expansion (Figures 4.13 - 4.15), we can verify the following observations. (Refer Figure 4.16 for Observation 32.)

Observation 32. *If X_{ref} visits a crossing point of an almost full crossing, then it visits at most one spine vertex of the crossing.*

Observation 33. *If X_{ref} visits a crossing point without visiting a spine vertex of the crossing, then the crossing must be a part of a butterfly crossing of G_{ref}.*

In Claim 34, we show that the properties of X_i, for $i = 1, 2$, shown in Observation 31 continue to hold for X_{ref}. These properties will be useful to prove that the extension of X_{ref} to a cycle X in $\Lambda(G)$ is a constrained separating cycle of length at most $2k$. Note that the properties of X_{ref} listed in Claim 34 also apply when X_{ref} is obtained directly from $\Lambda(G_{\text{ref}})$ without the contraction-expansion procedure (Claim 26.

Claim 34. *X_{ref} is a fence of $\Lambda(G_{\text{ref}})$ such that X_{ref} does not visit crossing points of full crossings of $\Lambda(G_{\text{ref}})$, $V_G(X_{\text{ref}}) \subseteq S$ and $|X_{\text{ref}}| \leq 2k$.*

Proof. Before the two expansion steps, we started with X_1 or X_2 (depending on $\Lambda(G_1)$). We will only show here that the properties are maintained as we expand from X_1 to X_{ref}. This proves that they always hold because if we started the expansion at X_1, then they hold for X_1 (Observation 31), and if we started at X_2, then they hold for X_2, and a similar argument shows that they are maintained as we expand from X_2 to X_1.

Since X_1 is a fence, there exist vertices inside and outside of X_1. Let w be a vertex inside X_1. If w is an uncontracted vertex, it remains inside X_{ref} as well. If w is a contracted vertex, then the spine vertices of G_{ref} that were a part of the contraction into w are inside

X_{ref}. Either way, there is a vertex of G_{ref} inside X_{ref}. Similarly, there is a vertex of G_{ref} outside X_{ref}. This shows that X_{ref} is a fence of $\Lambda(G_{\text{ref}})$.

To show that X_{ref} does not visit crossing points of full crossings of $\Lambda(G_{\text{ref}})$, note that X_1 does not visit crossing points of full crossings (Observation 31). After the expansion, X_{ref} may only visit crossing points of almost full crossings, but not crossing points of full crossings.

We now show that $V_G(X_{\text{ref}}) \subseteq S$. Note that $V_G(X_1) \subseteq S_1$ (Observation 31), where S_1 consists of uncontracted vertices of S from G_{ref} and contracted vertices. After the expansion steps, X_{ref} passes through the same set of uncontracted vertices as X_1, and additionally passes through some spine vertices, which also belong to S (Observation 13).

Next we show that $|X_{\text{ref}}| \leq 2k$. Note that $|X_1| \leq 2|S_1|$ (Observation 31). During each step of the expansion, we crucially maintain that for every crossing point of an almost full crossing that the cycle visits (after the expansion), there is a distinct spine vertex of the crossing that the cycle does not visit (Observation 32 and Figure 4.16). This ensures that the total number of crossing points and vertices of S through which X_{ref} passes is at most the total number of vertices of S. Therefore, $|X_{\text{ref}}| \leq 2k$. □

From X_{ref} to a constrained separating cycle in $\Lambda(G)$. We now show how to map X_{ref} to a constrained separating cycle X of $\Lambda(G)$ such that $|X| \leq 2k$. The mapping is described algorithmically by giving a step-by-step transformation from $\Lambda(G_{\text{ref}})$ to $\Lambda(G)$. This transformation essentially involves undoing the steps that were done to obtain G_{ref} from G.

Recall that G_{ref} was obtained from G by inserting some kite edges to get G^+, and then by deleting crossed edges of some full crossings of G^+. Therefore, to get back from $\Lambda(G_{\text{ref}})$ to $\Lambda(G)$, we first add all those crossed edges of G^+ that were deleted during refinement, and locally modify the radial graph and the cycle X_{ref} at the crossings (exactly as in Lemma 24; see Figure 4.7). The resulting graph is $\Lambda(G^+)$, and let X^+ be the resulting fence.

We now give the broad idea for transforming $\Lambda(G^+)$ to $\Lambda(G)$, and then fine-tune it so that the transformation works correctly. The broad idea is as follows. First, we delete kite edges of G^+ that are not in G. Since deleting edges of G^+ merges faces, this step results in some faces of G having more than one face vertex. For each such face, we identify all the face vertices in the face (i.e., temporarily add edges connecting the face vertices and contract all of these edges into a single loopless vertex) and delete the resulting bigons. If this process results in two face vertices of X^+ getting identified, one of the resulting smaller cycles must be a fence. (See Figure 4.17 for an illustration. Here, the bold edges depict

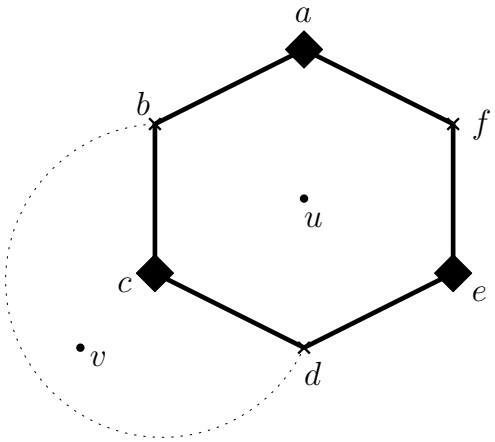


Figure 4.17: Identifying two face vertices on a fence gives a smaller fence.

the fence X^+ in $\Lambda(G^+)$ that separates u and v. If the face vertices b, d are in the same face of G, then identifying them into one vertex z_{bd} gives rise to a smaller fence (c, z_{bd}) that separates u and v.) Update X^+ to be this smaller cycle, and repeat this step for all faces of G. Let X be the resulting fence and $\widehat{\Lambda}$ be the resulting graph.

Although the transformation above mostly works correctly, we fine-tune the above procedure so that we get a constrained separating cycle in $\Lambda(G)$.

First, $\widehat{\Lambda}$ may not exactly be $\Lambda(G)$ because building $\Lambda(G)$ involves a special rule near butterfly crossings with a spine, but not all such crossings in G remain butterfly crossings in G^+. Specifically, assume that G has a butterfly crossing $\{(u, v), (w, x)\}$ and $\{(w, y), (v, z)\}$ with spine (v, w). Recall that $\Lambda(G)$ removes edge (v, w) and creates one face vertex in its place (Figure 4.18b). But it may happen that G^+ adds one or both of the kite edges (u, x) and (z, y), which means that the pair of crossings $\{(u, v), (w, x)\}$ and $\{(w, y), (v, z)\}$ are not a butterfly crossing in G^+. Hence, edge (v, w) remains in $\Lambda(G^+)$ and has two incident face vertices (Figure 4.18a). To obtain $\Lambda(G)$, we must delete the spine (v, w), identify the two face vertices, delete the bigons, and choose one of the smaller cycles that is a fence. Update the cycle X and the graph, and repeat for all such instances. The graph that we get after this step is $\Lambda(G)$.

Second, we will need the equivalent result of Observation 33 for X to prove (Ψ2.4) (for a suitable marking function below). However, if X^+ visits an almost full crossing point and a spine vertex of the crossing, we may lose the part of X^+ that visited the spine vertex when we update X^+ to be a much smaller subcycle. Hence, the equivalent of Observation 33 does not automatically hold for X. This problem can be addressed by adding the following

Figure 4.18: Modifying the radial graph at butterfly crossing to get $\Lambda(G)$, which may involve shortening the cycle X.

modification step after (E2).

(M) Assume X_{ref} visits an almost full crossing point c, and uses the edges $(c, f_1), (f_1, u)$, where f_1 is the face vertex at the spine face and u is a spine vertex (Figure 4.19). Then re-route X_{ref} to use the edges $(c, f_2), (f_2, u)$, where f_2 is the other face vertex at (u, c). Repeat this step at all such almost full crossings.

Clearly, Observation 33 and Claim 34 continue to hold even for the modified X_{ref}. The following claim is essential to prove that X satisfies (Ψ2.4) (for a marking function defined later).

Claim 35. *If X visits a crossing point c in $\Lambda(G)$ without visiting any other endpoint of the crossing, then X visits c in $\Lambda(G_{\text{ref}})$ and the crossing is part of a butterfly crossing in G_{ref}.*

Proof. First, note that the transformation from X_{ref} to X does not add crossing points, so c also belongs to X_{ref}. Now, we prove the claim by its contrapositive. Suppose that c is not part of a butterfly crossing of G_{ref}. Then the crossing must be almost full in G_{ref} (Claim 34). Let $\{(u, v), (w, x)\}$ be the crossing where (u, x) is the spine. X_{ref} does not visit the face vertex at the spine face since the modification (M) above re-routes X_{ref} to avoid the face vertex at the spine face. Therefore, X_{ref} must visit a face vertex f of a kite face that is not a spine face. Since (u, x) is the spine, the kite edge at f also exists in G, which means that the face vertex f never undergoes a contraction as we modify X_{ref} into X. Since f belongs to X, one endpoint of the kite edge at f must belong to X. Hence, X visits an endpoint of the crossing. □

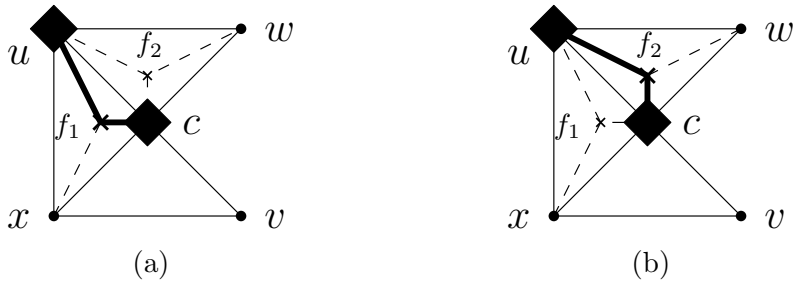


Figure 4.19: Re-routing X_{ref} at an almost full crossing to avoid spine faces.

Finding the marking function. With the above modifications in place, we now prove that X is a constrained separating cycle of $\Lambda(G)$ after defining a suitable marking function.

- ($\Psi 1$): Since X is a fence of $\Lambda(G)$, it is a subgraph of $R(G)$ and separates two vertices of G.
- ($\Psi 2$): Let the marking function $\beta : V(G) \mapsto \{\triangle, \square\}$ be the function that assigns all vertices of S to $\square$ and the other vertices to $\triangle$.

  Suppose that X visits a crossing point c. Since the non-facial vertices of X are a subset of the non-facial vertices of X_{ref}, the cycle X_{ref} also visits c in $\Lambda(G_{\text{ref}})$. Therefore, the crossing is an almost full crossing of G_{ref} (Claim 34). Now observe the conditions on β.

  1. Since the spine vertices of almost full crossings belong to S and wing tips do not belong to S (Observation 13), the endpoints of each edge of the crossing are marked with opposite symbols as required for ($\Psi 2.1$).
  2. The vertices marked $\triangle$ are wing tips of almost full crossings which are not adjacent, so ($\Psi 2.2$) holds.
  3. If X visits an endpoint of the crossing, so does X_{ref}. Since $V_G(X_{\text{ref}}) \subseteq S$ (Claim 34), this endpoint must be a vertex of S and marked $\square$. Hence, ($\Psi 2.3$) holds.
  4. If no endpoint of the crossing is on X, then the crossing is a part of a butterfly crossing of G_{ref} (Claim 35). By Observation 13, G_{ref} has no bowtie crossings, so this butterfly crossing must have a spine. Also, by Observation 13, the axial vertices (which are endpoints of the spine) belong to S, and hence are marked $\square$. The transformation from G to G_{ref} does not add butterfly crossings, so the crossing is also part of a butterfly crossing in G. Hence, ($\Psi 2.4$) holds.

This proves that X is a constrained separating cycle of $\Lambda(G)$. Since $|X_{\text{ref}}| \leq 2k$ (Claim 34), $|X| \leq 2k$. This concludes the proof of Theorem 3.

Chapter 5

Computing constrained separating cycles in linear time

In this chapter, we give a linear time algorithm that takes a planar graph as its input, and outputs a constrained separating cycle of a given fixed length (if it exists) in the graph. The problem of finding constrained separating cycles in a planar graph can be solved by solving the more general *planar subgraph isomorphism problem*. An *isomorphism* from a graph H to a graph G is a bijection $f : V(H) \mapsto V(G)$ such that $(u, v) \in E(H)$ if and only if $(f(u), f(v)) \in E(G)$. A graph H *occurs* in a graph G if there is an isomorphism from H to a subgraph of G. The problem of subgraph isomorphism is to find and list (if it exists) an occurrence of H in G. When the *host graph* G and the *pattern graph* H are both planar, the problem is called the planar subgraph isomorphism problem.

In [Epp99], Eppstein gives a linear time algorithm for solving the planar subgraph isomorphism problem. The algorithm is based on a technique of partitioning the host graph G into pieces of bounded treewidth (Section 5.1 defines the treewidth of a graph), and then applying a dynamic programming algorithm to find the pattern graph within each piece. [Epp99] also contains adaptations of the dynamic programming algorithm and/or the planar subgraph isomorphism algorithm for solving various problems associated with planar subgraph isomorphism. These include the problem of finding a bounded length (unconstrained) separating cycle in a planar graph. More formally, given a planar host graph G and a pattern graph H that is a cycle of bounded length, find a subgraph $J \subseteq G$ isomorphic to H (if it exists) such that the vertices of J form a separating set of G. In general, we call the problem of finding separating subgraphs as the *separating subgraph isomorphism problem*.

One alternative to doing explicit dynamic programming algorithm for finding separating cycles is to use Courcelle's Theorem [Cou90]. This theorem states that every graph property definable in a logic form, called the *monadic second order logic* (MSOL), can be decided in linear time if the treewidth of the graph and the size of the logic formula are bounded.

MSOL of graphs allows the following logic operations:

- Variables that are vertices and vertex sets, for example $v \in V$ and $I \subseteq V$.
- Membership tests, for example, $v \in I$.
- Boolean operations such as $\neg, \wedge, \vee, \Rightarrow$.
- Second-order predicates, for example, $adj(u, v)$ which is true if and only if u and v are adjacent.
- Quantifiers such as $\exists$ and $\forall$ for variables (but not for predicates).

When describing MSOL formulas, it is acceptable to use expressions such as "v_1, v_2, v_3 are distinct", or "for all $i = 1, 2, 3$" provided that it is clear how one could translate such expressions into MSOL.

We now formulate the problem of finding a separating cycle of length p in a graph of bounded treewidth using MSOL. Let $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ be the set of integers modulo p.

$\exists$ distinct vertices v_i for $i \in \mathbb{Z}_p$ such that:

- $adj(v_i, v_{i+1})$ for $i \in \mathbb{Z}_p$ and
- $\exists A, B \subseteq V(G)$ such that:
  - $A \neq \emptyset$ and $B \neq \emptyset$
  - $\forall v \in A \cup B$ and $i \in \mathbb{Z}_p$: $v \neq v_i$
  - $\forall v \in V(G) \setminus \{v_i : i \in \mathbb{Z}_p\}$: $v \in A \Leftrightarrow v \notin B$ and
  - $\forall v, w \in V(G)$: $adj(v, w) \Rightarrow (v \notin A \vee w \notin B)$

The standard proof of Courcelle's theorem is to construct an automaton that recognizes a tree decomposition of the graph. However, the size of the automaton can become extremely large, making the problem hard to tackle in practice. The hidden constant in

the running time, which is the running time's dependence on the graph's treewidth and the formula describing the problem, is in fact (in the worst case) a tower of exponentials. For this reason, we do not use MSOL and instead resort to the explicit dynamic programming approach taken by Eppstein [Epp99].

For the separating cycle problem, [Epp99] only sparingly discusses the details. Moreover, there are some errors in the overall separating subgraph isomorphism algorithm (Appendix A). In this chapter, we give a comprehensive description of the complete dynamic programming algorithm, and discuss how to incorporate the constraints ($\Psi 1$)-($\Psi 2$) in the dynamic programming to obtain constrained separating cycles. Finally, we give the overall separating subgraph isomorphism algorithm for finding constrained separating cycles, giving corrections to the errors in the separating subgraph isomorphism algorithm of [Epp99].

5.1 Separating cycles in graphs of bounded treewidth

To compute separating cycles of a graph G, we follow the same overall idea as Eppstein: partition the graph into pieces of bounded treewidth and then looking for a separating cycle within each piece by dynamic programming. In this section, we describe the dynamic programming algorithm for finding (unconstrained) separating cycles in graphs of bounded treewidth. While this section is mostly a review of the work by Eppstein, we fill in many details and proofs that were left to the reader. We begin by formally describing the tree decomposition of a graph.

5.1.1 Tree decomposition

A *tree decomposition* of a graph $G = (V, E)$ is a tree T where

1. Each node of T, referred to as a *bag*, is labelled by a subset of V.
2. Each edge of G is in the subgraph induced by the vertices of some bag of T. This is called the *edge coverage property*.
3. For each vertex v of G, the set of bags containing v forms a connected subtree of T. This is called the *coherence property*.

Figure 5.1 shows an example of a tree decomposition. Tree decompositions are generally not unique, and a graph may have several different tree decompositions. The *width* of a

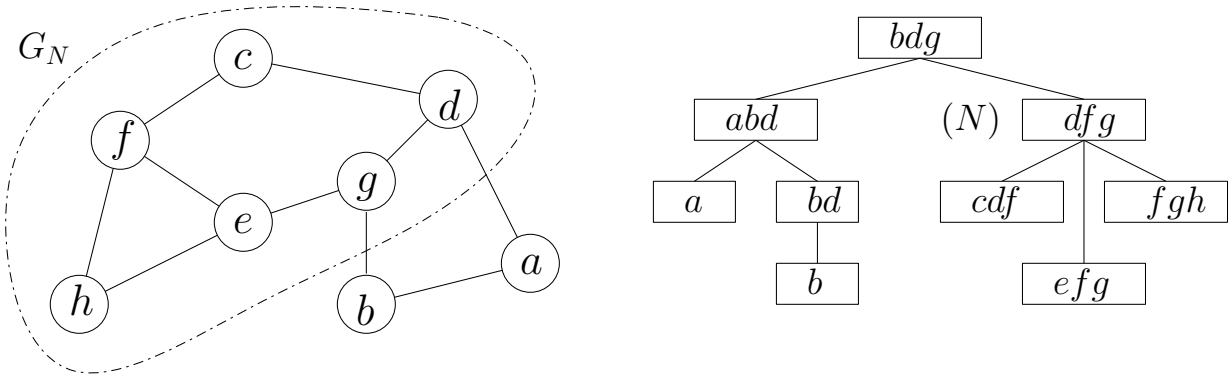


Figure 5.1: A graph and its tree decomposition. For the indicated bag N, the subgraph G_N is also shown.

tree decomposition is one less than the maximum bag size of that tree decomposition. For example, the width of the tree decomposition in Figure 5.1 is 2. The *treewidth* of a graph is the minimum integer k such that there exists a tree decomposition of the graph having width k.

Claim 36 gives a simple way to modify a tree decomposition to obtain another tree decomposition. This claim will be useful later in Section 5.2 where we discuss how to incorporate constraints ($\Psi 1$) - ($\Psi 2$) into the dynamic programming algorithm.

Claim 36. *Let T be a tree decomposition of a graph G. Let (u, v) be an edge of G. Adding v to each bag of T containing u gives another tree decomposition T' of G.*

Proof. The only property of tree decompositions that is non-trivial to verify is the coherence property for v in T'. As T is a tree decomposition of G, the bags containing u and v each form a connected subtree of T, say T_u and T_v respectively. Since (u, v) is an edge of G, T_u and T_v intersect in at least one bag. Adding v to all bags of T_u maintains the coherence property for v since $T_u \cup T_v$ is also a subtree. Thus, T' is a tree decomposition of G. □

Without loss of generality, we may assume that the tree decomposition of G is a rooted binary tree with $O(n)$ bags, where $n = |V(G)|$ ([Klo94]). If N is a bag in a given tree decomposition T, we let G_N be the subgraph of G induced by the vertices in all the bags of the sub-tree of T rooted at N (Figure 5.1). We use $L(N)$ to denote the set of vertices in a bag N of the tree decomposition. (These notation are borrowed from [Epp99]).

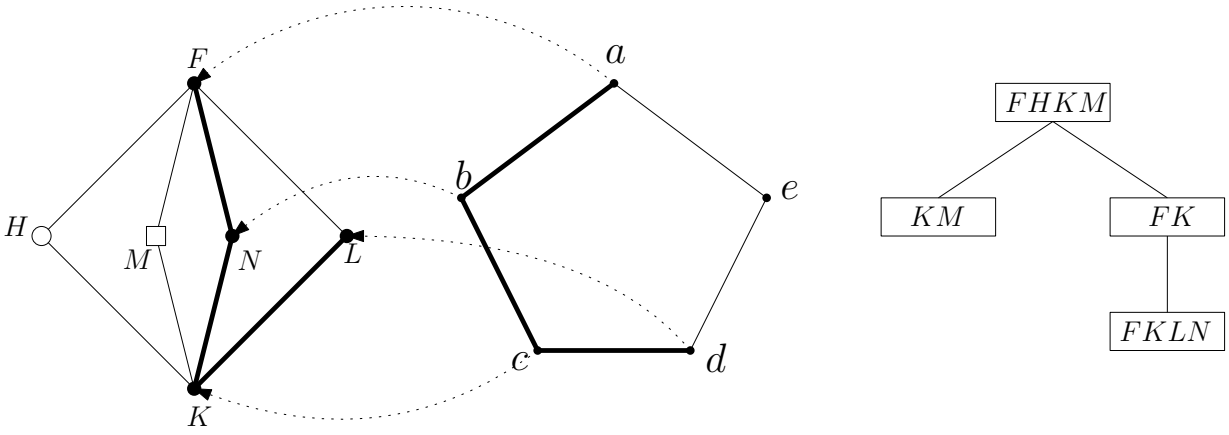


Figure 5.2: An extended partial isomorph.

5.1.2 Extended partial isomorphs and extended partial isomorph boundaries

The dynamic programming algorithm constructs an isomorphism from the pattern graph H (which for our application is a cycle of some fixed length, but the results in this section work for an arbitrary graph H) to a subgraph of G_R, where R is the root of the tree, using what we call "partial isomorphs" (defined next) at the children of R, which is in turn constructed using partial isomorphs of their children, and so on. A *partial isomorph* of H at a bag N of the tree is an injective function $I : V(H') \mapsto V(G_N)$, where H' is an induced subgraph of H, such that for every edge $(u, v) \in E(H')$, $(I(u), I(v)) \in E(G_N)$. See Figure 5.2 for an illustration. The graph on the left is the host graph; its tree decomposition is on the right. The pattern graph is in the middle and the dotted arrows show the partial isomorph at node $FHKM$.

The dynamic programming algorithm also outputs a certificate that gives a partition of the vertices of $V(G) \setminus H$. This is achieved by defining *separator functions*. For a set $U \subseteq V(G)$, a function $S : U \mapsto \{-1, 0, 1\}$ is a *separator function* if for any two vertices u, v with $S(u) = -1$ and $S(v) = 1$, $(u, v) \notin E(G)$. Note that if the pre-image sets $S^{-1}(1)$ and $S^{-1}(-1)$ are both non-empty, then the pre-image set $S^{-1}(0)$ gives a separator of G. Since we are interested in obtaining a separating cycle along with a certificate that gives the separation, our objects of interest are a combination of both a partial isomorph and a separator function.

An *extended partial isomorph* $\mathcal{I}$ at a bag N of the tree decomposition is a pair of functions $\mathcal{I} = (I, S)$ where I is a partial isomorph at N, and S is a separator function with

$\text{dom}(S) = V(G_N)$ and $S(v) = 0$ if and only if $v \in \text{Range}(I)$. (Here, 'dom' and 'Range' refer to the domain and the range of the functions respectively.) The functions I and S are respectively called the *component partial isomorph* and the *component separator function* of $\mathcal{I}$.

It is important to note a fundamental difference between partial isomorphs and separator functions, namely, the domain of a partial isomorph is a subgraph of H, whereas the domain of a separator function is a subgraph of G. In Figure 5.2, the separator function is illustrated by the three different ways in which vertices of G are marked. The disks are vertices mapped to 0, the circles are vertices mapped to (say) 1, and the squares the vertices mapped to -1.

The following observation shows that a separating subgraph in G that is isomorphic to H naturally gives rise to a corresponding extended partial isomorph at the root node in the tree decomposition of G.

Observation 37. *If G contains H as a separating subgraph, then there is an extended partial isomorph $\mathcal{I} = (I, S)$ at the root node where $dom(I) = H$, and $S^{-1}(1)$ and $S^{-1}(-1)$ are both non-empty.*

Proof. Since G contains H as a separator, the vertices of $V(G) \setminus H$ has at least two flaps. Let F_1 be the set of vertices in one flap and F_2 be the set of vertices in all other flaps. Let I be the isomorphism from H to the given separating subgraph of G. Let $S(u) = -1$ for all $u \in F_1$, $S(u) = 1$ for all $u \in F_2$ and $S(u) = 0$ for all vertices in the image of I. Then $\mathcal{I} = (I, S)$ is an extended partial isomorph with the desired property. □

An efficient dynamic programming algorithm that computes extended partial isomorphs at a bag N would work locally on the set of vertices in N and its children, rather than all the vertices of G_N. For this purpose, we define "partial isomorph boundaries". A *homomorphism* from a graph H to a graph G is a function $f : V(H) \mapsto V(G)$ such that if $(u, v) \in E(H)$, then $(f(u), f(v)) \in E(G)$. (Note that unlike an isomorphism, a homomorphism need not be a bijection, and $(f(u), f(v)) \in E(G)$ does not imply that $(u, v) \in E(H)$.) Recall that $L(N)$ denotes the set of vertices in a bag N of the tree decomposition. Let G'_N be the graph which contains the subgraph of G induced by $L(N)$, and contains two additional vertices x_N and y_N each of which is connected to all the vertices in $L(N)$. Furthermore, x_N and y_N are joined by an edge and are each given a self loop. A *partial isomorph boundary* B at a bag N is a homomorphism from H to G'_N with the property that the restriction of B on the set $\{v : B(v) \in L(N)\}$ is an injective function Figure 5.3 shows a partial isomorph boundary at bag $FHKM$ in the tree decomposition shown in Figure 5.2.

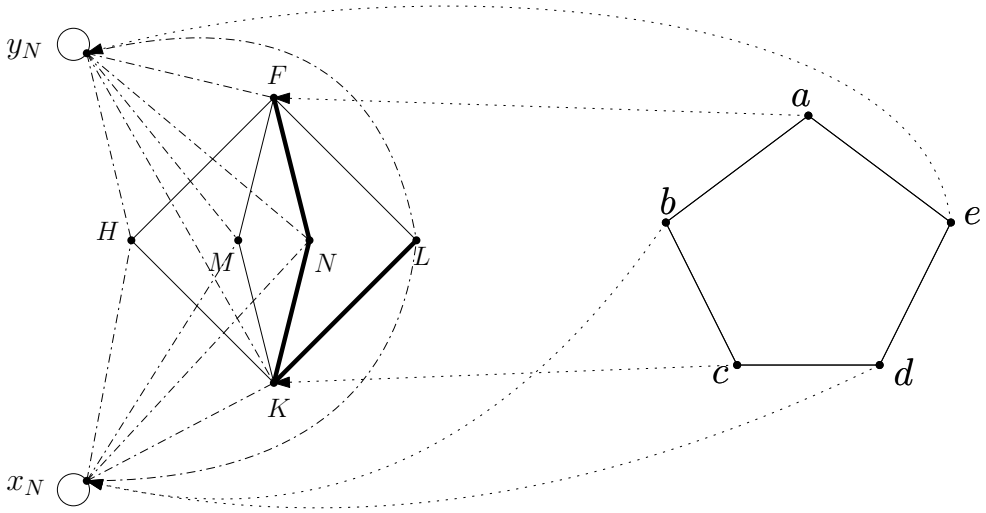


Figure 5.3: A partial isomorph boundary

Analogous to extended partial isomorphs, we define extended partial isomorph boundaries. An *extended partial isomorph boundary* $\mathcal{B}$ is a pair of functions $\mathcal{B} = (B, T)$, where B is a partial isomorph boundary, and T is a separator function where $\text{dom}(T) = \{v \in L(N)\}$ and $T(v) = 0$ if and only if $v \in \text{Range}(B)$. The functions B and T are respectively called the *component boundary* and *component separator function* of $\mathcal{B}$.

As the names suggest, there is a strong relationship between partial isomorphs and partial isomorph boundaries, which we explore now. From a given extended partial isomorph $\mathcal{I} = (I, S)$ at N, we can create an extended partial isomorph boundary $\mathcal{B} = (B, T)$ as follows. Let $I : V(H') \mapsto V(G_N)$. For each vertex v in H', if $I(v) \in L(N)$, then set $B(v) = I(v)$. For any other vertex v in H', set $B(v) = x_N$, and for all vertices in $H - H'$, set $B(v) = y_N$. We say that B is the *summary* of I, and write $I \rightsquigarrow B$. (Figure 5.3 shows the partial isomorph boundary that is the summary of the partial isomorph in Figure 5.2.) Define T be the restriction of S onto $L(N)$. That is, $\text{dom}(T) := \{v \in L(N)\}$, and for each vertex $v \in \text{dom}(T)$, $T(v) := S(v)$. We say that $\mathcal{B} = (B, T)$ is the *summary* of $\mathcal{I}$, and write $\mathcal{I} \rightsquigarrow \mathcal{B}$.

For a given a partial isomorph boundary B at a node N, we will frequently need the restrictions of B onto $L(N)$, x_N, y_N, or a combination of them. For this purpose, we introduce the following notations. The restriction of B on $L(N)$ is denoted by B_N. If $\mathcal{B} = (B, T)$ is an extended partial isomorph boundary, we let $\mathcal{B}_N$ denote the pair of functions (B_N, T). For a partial isomorph boundary B at a bag N, the restriction of B onto the set $\{v : B(v) \in L(N) \cup \{x_N\}\}$ is denoted by B^x_N. Note that if $I \rightsquigarrow B$, then

$\mathrm{dom}(I) = \mathrm{dom}(B_N^x)$.

Throughout this section, we will follow some notational conventions:

- Calligraphic fonts such as $\mathcal{B}$ or $\mathcal{I}$ denote extended boundaries and extended partial isomorphs, whereas normal fonts such as B or I denote the usual boundaries and partial isomorphs.
- If $\mathcal{I}$ is an extended partial isomorph, then I represents the component partial isomorph of $\mathcal{I}$, and if $\mathcal{B}$ is an extended partial isomorph boundary, then B represents the component boundary of $\mathcal{B}$.
- We use the letters S and T to denote the component separator functions of extended partial isomorphs and extended boundaries respectively.

5.1.3 Leaf boundary, consistent boundaries and compatible triples

In Observation 37, we saw that a separating subgraph in G gives rise to a corresponding extended partial isomorph at the root node. Consider the restriction of this extended partial isomorph to the subtrees at different nodes of the tree decomposition. The summaries of these extended partial isomorphs must be "consistent" in some sense with the summaries of the extended partial isomorphs at their children. In this section, we formalise this notion of consistency through the definitions of leaf boundaries, consistent boundaries and compatible triples. (The meaning and necessity of these conditions will become clearer in the next subsection.)

We begin with the following definition. A pair of functions f_1 and f_2 with the property that $f_1(v) = f_2(v)$ for all $v \in \mathrm{dom}(f_1) \cap \mathrm{dom}(f_2)$ is said to be *gluable*. If f_1 and f_2 are a pair of gluable functions, then the *gluing* of f_1 and f_2 gives the function $f_1 \oplus f_2$ such that $\mathrm{dom}(f_1 \oplus f_2) = \mathrm{dom}(f_1) \cup \mathrm{dom}(f_2)$, and for all $v \in \mathrm{dom}(f_1)$, $(f_1 \oplus f_2)(v) = f_1(v)$, and for all $v \in \mathrm{dom}(f_2)$, $(f_1 \oplus f_2)(v) = f_2(v)$. We can similarly define the gluing operation for tuples of functions. Two tuples of functions (f_1, g_1) and (f_2, g_2) are said to be *gluable* if f_1, f_2 and g_1, g_2 are gluable; the *gluing* of (f_1, g_1) and (f_2, g_2) gives the function $(f_1, g_1) \oplus (f_2, g_2) = (f_1 \oplus f_2, g_1 \oplus g_2)$.

We now give the formal definitions of leaf boundaries, consistent boundaries and compatible triples.

1. **Leaf boundary.** An extended boundary $\mathcal{B}$ at a bag N is a *leaf boundary* if

(LB1) N is a leaf.

(LB2) There is no $v \in H$ such that $B(v) = x_N$.

2. **Consistent boundaries.** Two boundaries $\mathcal{B} = (B, T)$ and $\mathcal{B}_1 = (B_1, T_1)$ at bags N and N_1 are *consistent* if

(CB1) N_1 is the only child of N.

(CB2) For any vertex $v \in H$, if $B(v) = x_N$, then $B_1(v) \neq y_{N_1}$.

(CB3) T and T_1 are gluable.

(CB4) B and B_1 agree on their shared range in $L(N) \cap L(N_1)$. That is, for any vertex $v \in H$, if $B(v) \in L(N_1)$ or $B_1(v) \in L(N)$, then $B(v) = B_1(v)$.

(CB5) For any vertex $v \in H$, if $B_1(v) = x_{N_1}$ or $B_1(v) \in L(N_1) \setminus L(N)$, then $B(v) = x_N$.

(CB6) For every $(u, v) \in E(H[\mathrm{dom}(B_N^x)])$, either $B(u), B(v) \in L(N)$ or $B_1(u), B_1(v) \in \{x_{N_1}\} \cup L(N_1)$.

3. **Compatible triple.** Three boundaries $\mathcal{B} = (B, T)$, $\mathcal{B}_1 = (B_1, T_1)$ and $\mathcal{B}_2 = (B_2, T_2)$ at bags N, N_1 and N_2 form a *compatible triple* if:

(CT1) N_1 and N_2 are the two children of N.

(CT2) For any vertex $v \in H$, if $B(v) = x_N$, then *exactly* one of $B_1(v) = y_{N_1}$ and $B_2(v) = y_{N_2}$ occurs.

(CT3) B, B_1 and B, B_2 both satisfy conditions (CB3), (CB4), (CB5), and at least one of B, B_1 and B, B_2 satisfies (CB6).

5.1.4 From extended partial isomorphs to extended boundaries

In the previous section, we hinted at how the summaries of extended partial isomorphs at a node and its children must be consistent in some sense. We formalise this notion through the Lemmas 38-40. (In [Epp99], these lemmas were not explicitly stated and their proofs were not given.)

Lemma 38. *Let $\mathcal{I} = (I, S)$ be an extended partial isomorph at a bag N that is a leaf. Then the summary $\mathcal{B} = (B, S)$ of $\mathcal{I}$ is a leaf boundary.*

Proof. Since N is a leaf, $I(v) \in L(N)$ for all $v \in \text{dom}(I)$. Therefore, there is no $v \in H$ such that $B(v) = x_N$. Hence, $\mathcal{B}$ is a leaf boundary. □

Let $\mathcal{I} = (I, S)$ be an extended partial isomorph at a bag N. Let N_1 be a child of N. The *restriction* of $\mathcal{I}$ onto G_{N_1} is the partial isomorph $\mathcal{J}_1 = (J_1, S_1)$ where: (i) $\text{dom}(J_1) := \{v : I(v) \in G_{N_1}\}$ and $J_1(v) := I(v)$ for all $v \in \text{dom}(J_1)$, and (ii) $\text{dom}(S_1) := V(G_{N_1})$ and $S_1(v) := S(v)$ for all $v \in \text{dom}(S_1)$.

Lemma 39. *Let $\mathcal{I} = (I, S)$ be an extended partial isomorph at a bag N with a single child N_1. Let $\mathcal{J}_1 = (J_1, S_1)$ be the restriction of $\mathcal{I}$ onto G_{N_1}. Let $\mathcal{B} = (B, T)$ and $\mathcal{B}_1 = (B_1, T_1)$ be the summaries of $\mathcal{I}$ and $\mathcal{J}_1$ respectively. Then $\mathcal{B}$ and $\mathcal{B}_1$ consistent and $\mathcal{I} = \mathcal{B}_N \oplus \mathcal{J}_1$.*

Proof. Since $\mathcal{J}_1$ is the restriction of $\mathcal{I}$ onto G_{N_1} and $\mathcal{B}$ is the summary of $\mathcal{I}$, we have $S = T \oplus S_1$ and $I = B_N \oplus J_1$. Hence, $\mathcal{I} = \mathcal{B}_N \oplus \mathcal{J}_1$. To show that $\mathcal{B}$ and $\mathcal{B}_1$ are consistent, we show that they satisfy all the conditions necessary to be consistent.

1. As N_1 is the only child of N, it follows that $\mathcal{B}$ and $\mathcal{B}_1$ satisfy (CB1).
2. If there is a vertex $v \in H$ such that $B(v) = x_N$, then $I(v) \in G_N \setminus L(N)$. But as N_1 is the only child of N, $I(v) \in G_{N_1} \setminus L(N)$. This implies that $v \in \text{dom}(J_1)$, and therefore, $B_1(v) \neq y_{N_1}$. Hence, $\mathcal{B}$ and $\mathcal{B}_1$ satisfy (CB2).
3. T and T_1 are gluable because they are just the restrictions of S onto $L(N)$ and $L(N_1)$ respectively. Hence, $\mathcal{B}$ and $\mathcal{B}_1$ satisfy (CB3).
4. For any vertex $v \in H$, if $B(v) \in L(N_1)$, then $B(v) \notin \{x_N, y_N\}$. Hence $B(v) \in L(N)$. This implies $B(v) \in L(N) \cap L(N_1)$, and therefore, $B(v) = I(v) = J_1(v) = B_1(v)$. Similarly, if $B_1(v) \in L(N)$, then $B_1(v) \in L(N) \cap L(N_1)$, and therefore, $B_1(v) = J_1(v) = I(v) = B(v)$. Hence, $\mathcal{B}$ and $\mathcal{B}_1$ satisfy (CB4).
5. For any vertex $v \in H$, if $B_1(v) = x_{N_1}$ or $B_1(v) \in L(N_1) \setminus L(N)$, then $I(v) = J_1(v) \in G_{N_1} \setminus L(N)$. Therefore, $B(v) = x_N$. Hence, $\mathcal{B}$ and $\mathcal{B}_1$ satisfy (CB5).
6. Since $I \rightsquigarrow B$, we have $\text{dom}(I) = \text{dom}(B^x_N)$. As I is a partial isomorph, if $(u, v) \in E(H[\text{dom}(B^x_N)])$, then $(I(u), I(v))$ is an edge of G_N. From the edge coverage property of tree decompositions, either $I(u), I(v) \in L(N)$ or $I(u), I(v) \in G_{N_1}$. Correspondingly, either $B(u), B(v) \in L(N)$ or $B_1(u), B_1(v) \in \{x_{N_1}\} \cup L(N_1)$ occurs. Therefore, $\mathcal{B}$ and $\mathcal{B}_1$ satisfy (CB6).

Thus, $\mathcal{B}$ and $\mathcal{B}_1$ are consistent. □

Lemma 40. *Let $\mathcal{I} = (I, S)$ be an extended partial isomorph at a bag N with two children N_1 and N_2. For $i = 1, 2$, let $\mathcal{J}_i = (J_i, S_i)$ be the restrictions of $\mathcal{I}$ onto G_{N_i}. Let $\mathcal{B} = (B, T)$, and for $i = 1, 2$, let $\mathcal{B}_i = (B_i, T_i)$ be the summaries of $\mathcal{I}$ and $\mathcal{J}_i$ respectively. Then $\mathcal{B}$, $\mathcal{B}_1$ and $\mathcal{B}_2$ form a compatible triple and $\mathcal{I} = \mathcal{B}_N \oplus \mathcal{J}_1 \oplus \mathcal{J}_2$.*

Proof. Since $\mathcal{J}_1$ and $\mathcal{J}_2$ are the restrictions of $\mathcal{I}$ onto G_{N_1} and G_{N_2}, and $\mathcal{B}$ is the summary of $\mathcal{I}$, we have $S = T \oplus S_1 \oplus S_2$ and $I = B_N \oplus J_1 \oplus J_2$. Hence, $\mathcal{I} = \mathcal{B}_N \oplus \mathcal{J}_1 \oplus \mathcal{J}_2$. To show that $\mathcal{B}$, $\mathcal{B}_1$ and $\mathcal{B}_2$ form a compatible triple, we show that they satisfy all the conditions necessary to be compatible.

1. Since N has two children N_1 and N_2, it follows that $\mathcal{B}$, $\mathcal{B}_1$ and $\mathcal{B}_2$ satisfy (CT1).

2. If there is a vertex $v \in H$ such that $B(v) = x_N$, then $I(v) \notin L(N)$. This implies that $I(v) \in G_{N_1}$ or $I(v) \in G_{N_2}$, but not both (from the coherence property of tree decompositions). That is, $v \in \mathrm{dom}(J_1)$ or $v \in \mathrm{dom}(J_2)$, but not both. Therefore, exactly one of $B_1(v) = y_{N_1}$ and $B_2(v) = y_{N_2}$ occurs. Hence, $\mathcal{B}$, $\mathcal{B}_1$ and $\mathcal{B}_2$ satisfy (CT2).

3. T, T_1 and T, T_2 agree on their respective shared domains because T, T_1 and T_2 are just the restrictions of S onto $L(N)$, $L(N_1)$ and $L(N_2)$ respectively. Hence, B, B_1 and B, B_2 satisfy (CB3)

4. For $i = 1, 2$ and any vertex $v \in H$, if $B(v) \in L(N_i)$, then $B(v) \in L(N) \cap L(N_i)$, and therefore, $B(v) = I(v) = J_i(v) = B_i(v)$. If $B_i(v) \in L(N)$, then $B_i(v) \in L(N_i) \cap L(N)$, and therefore, $B_i(v) = J_i(v) = I(v) = B(v)$. Hence, B, B_1 and B, B_2 satisfy (CB4).

5. For $i = 1, 2$ and any vertex $v \in H$, if $B_i(v) = x_{N_i}$ or $B_i(v) \in L(N_i) \setminus L(N)$, then $I(v) = J_i(v) \in G_{N_i} \setminus L(N)$. Therefore, $B(v) = x_N$. Hence, B, B_1 and B, B_2 satisfy satisfy (CB5).

6. Since $I \rightsquigarrow B$, we have $\mathrm{dom}(I) = \mathrm{dom}(B^x_N)$. Therefore, if $(u, v) \in E(H[\mathrm{dom}(B^x_N)])$, then $(I(u), I(v))$ is an edge of G_N. From the edge coverage property of tree decompositions, either $I(u), I(v) \in L(N)$ or $I(u), I(v) \in G_{N_1}$ or $I(u), I(v) \in G_{N_2}$. Correspondingly, either $B(u), B(v) \in L(N)$ or $B_1(u), B_1(v) \in \{x_{N_1}\} \cup L(N_1)$ or $B_2(u), B_2(v) \in \{x_{N_2}\} \cup L(N_2)$ occurs. Therefore, at least one of B, B_1 and B, B_2 satisfy (CB6).

Thus, $\mathcal{B}$, $\mathcal{B}_1$ and $\mathcal{B}_2$ form a compatible triple. □

5.1.5 From extended boundaries to extended partial isomorphs

In the previous section, we saw that an extended partial isomorph $\mathcal{I}$ with a summary $\mathcal{B}$ at a non-leaf bag N gives rise to extended partial isomorphs at the children of N whose summaries are either consistent or compatible with $\mathcal{B}$. However, to find a separator isomorphic to H in G, we need to work in the reverse direction. That is, we need to search for extended partial isomorphs at each node of the tree, from the leaves to the root, looking at all possible leaf boundaries, consistent boundaries and compatible triples, and finally reporting a complete isomorphism at the root node (if it exists). For this strategy to work correctly, we need to prove some lemmas. We start with the simplest case of leaf boundaries.

Lemma 41. *Let N be a bag that is a leaf, and let $\mathcal{B} = (B, T)$ be a leaf boundary at N. Then $\mathcal{I} := (B_N, T)$ is an extended partial isomorph at N with $\mathcal{I} \rightsquigarrow \mathcal{B}$.*

Proof. Recall from the definition of a partial isomorph boundary that B is a homomorphism and that the restriction of B onto $L(N)$ is an injective function. Therefore, B_N is a partial isomorph. This immediately implies that $\mathcal{I}$ is an extended partial isomorph. Next, we show that $\mathcal{I} \rightsquigarrow \mathcal{B}$. Since $\mathcal{B}$ is a leaf boundary, there is no vertex $v \in H$ such that $B(v) = x_N$. Moreover, $B(u) = B_N(u)$ for all u such that $B(u) \neq y_N$. Hence, $\mathcal{I} \rightsquigarrow \mathcal{B}$. □

In the following lemmas, we show that for any extended boundary $\mathcal{B}$ at a non-leaf bag N, gluing $\mathcal{B}_N$ together with extended partial isomorphs whose summaries are consistent or compatible with $\mathcal{B}$ gives an extended partial isomorph.

Lemma 42. *Let N be a bag that has a single child N_1, and let $\mathcal{B} = (B, T)$ and $\mathcal{B}_1 = (B_1, T_1)$ be a pair of consistent boundaries at N and N_1 respectively. If there exists an extended partial isomorph $\mathcal{J}_1 = (J_1, S_1)$ such that $\mathcal{J}_1 \rightsquigarrow \mathcal{B}_1$, then $\mathcal{B}_N$ and $\mathcal{J}_1$ are gluable, and $\mathcal{I} = (I, S) = \mathcal{B}_N \oplus \mathcal{J}_1$ is an extended partial isomorph at N with $\mathcal{I} \rightsquigarrow \mathcal{B}$.*

Proof. First we prove that $\mathcal{B}_N$ and $\mathcal{J}_1$ are gluable. For this, we need to show that B_N and J_1 are gluable, and T and S_1 are gluable. First, we show that B_N and J_1 are gluable. Let v be a vertex such that $v \in \mathrm{dom}(B_N) \cap \mathrm{dom}(J_1)$. Then, $B(v) \in L(N)$ and $B_1(v) \neq y_{N_1}$ (since $J_1 \rightsquigarrow B_1$). If $B_1(v) \notin L(N)$, then from (CB5), $B(v) = x_N$, which is a contradiction. Therefore, $B_1(v) \in L(N)$. From (CB4), $B_1(v) = B(v)$. Therefore, $J_1(v) = B_N(v)$. Thus, B_N and J_1 are gluable. Next, we show that S and T_1 are gluable. Let v be a vertex such that $v \in \mathrm{dom}(T) \cap \mathrm{dom}(S_1)$. Since $v \in \mathrm{dom}(T) = L(N)$, by the coherence property of tree decompositions, $v \in \mathrm{dom}(T_1)$, where T_1 is the component separator function of $\mathcal{B}_1$. From

(CB3), we have that T and T_1 agree on their shared domain. Hence, $T(v) = S_1(v)$, and hence, T and S_1 are gluable. Thus, we can construct $\mathcal{I} = (I, S)$ such that $\mathcal{I} = \mathcal{B}_N \oplus \mathcal{J}_1$.

For the remaining parts of the proof, it will be helpful to first show that $\text{dom}(I) = \text{dom}(B_N^x)$. Since $I = B_N \oplus J_1$, we have $\text{dom}(I) = \text{dom}(B_N) \cup \text{dom}(J_1)$. For any vertex $v \in \text{dom}(J_1)$, either $J_1(v) \in L(N)$ or $B(v) = x_N$ (CB5), and if $J_1(v) \in L(N)$, we have $B(v) \in L(N)$ (CB4). Moreover, for any vertex v such that $B(v) = x_N$, $v \in \text{dom}(J_1)$ (CB2). Therefore, $\text{dom}(I) = \text{dom}(B_N) \cup \text{dom}(J_1) = \{v : B(v) \in L(N)\} \cup \{v : B(v) = x_N\} = \text{dom}(B_N^x)$.

To show that $\mathcal{I} = (I, S)$ is an extended partial isomorph, we need to show that I is a partial isomorph. Let $H' := H[\text{dom}(I)]$. We show that for every $(u, v) \in E(H')$, $(I(u), I(v)) \in E(G)$. As $\text{dom}(I) = \text{dom}(B_N^x)$, $H' = H[\text{dom}(I)] = H[\text{dom}(B_N^x)]$. From (CB6) of consistent boundaries, it follows that for every $(u, v) \in E(H')$, $B(u), B(v) \in L(N)$ or $B_1(u), B_1(v) \in \{x_{N_1}\} \cup L(N_1)$. If $B(u), B(v) \in L(N)$, then $(B(u), B(v)) = (I(u), I(v)) \in E(G)$ since B is a homomorphism. If $B_1(u), B_1(v) \in \{x_{N_1}\} \cup L(N_1)$ then $u, v \in \text{dom}(J_1)$, which implies that $(J_1(u), J_1(v)) = (I(u), I(v)) \in E(G)$ since J_1 is a partial isomorph.

To complete the proof that I is a partial isomorph, we also need to show that I is an injective function. Assume for contradiction that there are two distinct vertices u and v such that $I(u) = I(v)$. Since B_N and J_1 are both injective functions, it must follow that $u \in \text{dom}(B_N)$ and $v \in \text{dom}(J_1)$ (or vice-versa). Therefore, $I(u) \in L(N)$ and $I(v) \in G_{N_1}$. Since $I(u) = I(v)$, it must follow from the coherence property of tree decompositions that $I(v) = J_1(v) \in L(N_1)$, and therefore $B_1(v) \in L(N_1)$. Now, we have that $B(u) \in L(N)$, $B_1(v) \in L(N_1)$ and $B(u) = B_1(v)$. From (CB4), $B(v) = B_1(v) = B(u) \in L(N)$. This shows that B_N is not injective, which is a contradiction. Therefore, I is an injective function.

We have shown that I is a partial isomorph. To show that $\mathcal{I} = (I, S)$ is an extended partial isomorph, we show that $S = T \oplus S_1$ is a separator function. First we show that $S(u) = 0$ if and only if $u \in \text{Range}(I)$. To see this, note that $S(u) = 0$ if and only if $T(u) = 0$ or $S_1(u) = 0$ if and only if $u \in \text{Range}(B_N)$ or $u \in \text{Range}(J_1)$ if and only if $u \in \text{Range}(I)$. Next, we show that there do not exist two vertices u, v such that $S(u) = -1$, $S(v) = 1$, but $(u, v) \in E(G)$. Suppose not, for contradiction. Since T and S_1 are both separator functions, u and v cannot both belong to $\text{dom}(T)$ or $\text{dom}(S_1)$, and $u \in \text{dom}(T)$ and $v \in \text{dom}(S_1)$ (or vice-versa). This implies that $u \in L(N) \setminus G_{N_1}$ and $v \in G_{N_1} \setminus L(N)$. From the edge coverage and coherence property of tree decomposition, this implies that $(u, v) \notin E(G)$. This is a contradiction. Hence, S is a separator function. Since we have already shown that I is a partial isomorph, it follows that $\mathcal{I} = (I, S)$ is an extended partial isomorph.

Next, we prove that $\mathcal{I} \rightsquigarrow \mathcal{B}$. For this, we need to show that $I \rightsquigarrow B$. Since $I = B_N \oplus J_1$, $I(u) \in L(N)$ implies that either $B_N(u) \in L(N)$ or $J_1(u) \in L(N)$. If $J_1(u) \in L(N)$, then $B_1(u) \in L(N)$ which implies that $B_1(u) = B(u) \in L(N)$ (CB4), which in turn implies that $B_N(u) \in L(N)$. Therefore, if $I(u) \in L(N)$, then $B_N(u) \in L(N)$. Since $I = B_N \oplus J_1$, this shows that $I(u) = B(u)$. On the other hand, if $B(u) \in L(N)$, then $I(u) = B_N(u) = B(u)$ (since $I = B_N \oplus J_1$). Since $\mathrm{dom}(I) = \mathrm{dom}(B_N^x)$ and $I(u) \in L(N)$ if and only if $B(u) \in L(N)$, we have $I(u) \notin L(N)$ if and only if $B(u) = x_N$. This shows that $I \rightsquigarrow B$.

To complete the proof that $\mathcal{I} \rightsquigarrow \mathcal{B}$, we show that T is the restriction of S onto $L(N)$. For this, we need to show that for all $v \in \mathrm{dom}(T)$, $T(v) = S(v)$. This follows from noting that $S = T \oplus T_1$. Hence $\mathcal{I} \rightsquigarrow \mathcal{B}$. □

Lemma 43. *Let N be a bag that has two children N_1 and N_2, and let $\mathcal{B}$, $\mathcal{B}_1$ and $\mathcal{B}_2$ be a set of compatible triples at N, N_1 and N_2 respectively. If there exist extended partial isomorphs $\mathcal{J}_1$ and $\mathcal{J}_2$ such that $\mathcal{J}_1 \rightsquigarrow \mathcal{B}_1$ and $\mathcal{J}_2 \rightsquigarrow \mathcal{B}_2$, then $\mathcal{B}_N$, $\mathcal{J}_1$ and $\mathcal{J}_2$ are gluable, and $\mathcal{I} = \mathcal{B}_N \oplus \mathcal{J}_1 \oplus \mathcal{J}_2$ is an extended partial isomorph at N with $\mathcal{I} \rightsquigarrow \mathcal{B}$.*

Proof. The proof is similar to the proof of Lemma 42, and is left to the reader. □

5.1.6 The dynamic programming algorithm

In this section, we present a dynamic programming algorithm that solves the decision version of the problem of finding separating subgraphs in a graph with a given tree decomposition. That is, the algorithm only reports whether the host graph contains a separating subgraph of a given size (number of vertices). The constructive version of the algorithm can be obtained from the decision version in the usual way for dynamic programming by keeping track of values at each step of the algorithm (see [Epp99, Lemma 3] for an example).

For the dynamic programming algorithm, we classify separator functions into four classes according to their range.

- $\mathcal{S}_0 = \{S : S(u) = 0 \text{ for all } u \in \mathrm{dom}(S)\}$
- $\mathcal{S}_p = \{S : S(u) \in \{0, 1\} \text{ for all } u \in \mathrm{dom}(S)$, and there exists a vertex u with $S(u) = 1\}$. (The subscript 'p' in $\mathcal{S}_p$ stands for 'positive'.)
- $\mathcal{S}_n = \{S : S(u) \in \{0, -1\} \text{ for all } u \in \mathrm{dom}(S)$, and there exists a vertex u with $S(u) = -1\}$. (The subscript 'n' in $\mathcal{S}_n$ stands for 'negative'.)

- $\mathcal{S}_2 = \{S :$ there exist two vertices u, v with $S(u) = -1$ and $S(v) = 1\}$.

In a similar vein, we categorise extended partial isomorphs according to the type of the component separator functions. An extended partial isomorph $\mathcal{I} = (I, S)$ is of *type* i if $S \in \mathcal{S}_i$, where $i \in \{0, p, n, 2\}$.

The algorithm

For an extended boundary $\mathcal{B}$ and $i \in \{0, p, n, 2\}$, let $\chi_i(\mathcal{B})$ be an indicator variable such that $\chi_i(\mathcal{B}) = 1$ if and only if there exists an extended partial isomorph of type i whose summary is $\mathcal{B}$. We write χ_i when the extended boundary is clear from context. The algorithm computes $\chi_i(\mathcal{B})$ for each boundary $\mathcal{B}$ at each bag N. These values are computed in a bottom-up fashion from the leaf bags to the root bag.

1. Let N be a leaf bag. Compute and store all extended partial isomorph boundaries $\mathcal{B} = (B, T)$ at N. For each $i \in \{0, p, n, 2\}$, if $\mathcal{B}$ is a leaf boundary and $T \in \mathcal{S}_i$, then set $\chi_i(\mathcal{B}) := 1$; otherwise set $\chi_i(\mathcal{B}) := 0$. Store all the computed values of χ_i along with $\mathcal{B}$.

2. Let N be a bag with a single child N_1. Compute all extended boundaries $\mathcal{B} = (B, T)$ at N. For each such extended boundary, set $\chi_i := 0$ for all i. For each boundary $\mathcal{B}_1$ at N_1 consistent with $\mathcal{B}$, and for each $i \in \{0, p, n, 2\}$, do the following. Look up the value $\chi_i^1 := \chi_i(\mathcal{B}_1)$. Let $\chi_\star^1 := \max\{\chi_0^1, \chi_p^1, \chi_n^1, \chi_2^1\}$.

   (a) If $T \in \mathcal{S}_0$, then for all $i \in \{0, p, n, 2\}$, set $\chi_i := \max\{\chi_i, \chi_i^1\}$
   (b) If $T \in \mathcal{S}_p$, then set $\chi_p := \max\{\chi_p, \chi_0^1, \chi_p^1\}$ and set $\chi_2 := \max\{\chi_2, \chi_n^1, \chi_2^1\}$
   (c) If $T \in \mathcal{S}_n$, then set $\chi_n := \max\{\chi_n, \chi_0^1, \chi_n^1\}$ and set $\chi_2 := \max\{\chi_2, \chi_p^1, \chi_2^1\}$
   (d) If $T \in \mathcal{S}_2$, then set $\chi_2 := \max\{\chi_2, \chi_\star^1\}$

   Store all the computed values of χ_i along with $\mathcal{B}$.

3. Let N be a bag with two children N_1 and N_2. Compute all extended boundaries $\mathcal{B} = (B, T)$ at N. For each such extended boundary, set $\chi_i := 0$ for all i. We update χ_i by iterating through all extended boundaries at N_1 and N_2 that are compatible with $\mathcal{B}$.

   For each pair of boundaries $\mathcal{B}_1$ and $\mathcal{B}_2$ compatible with $\mathcal{B}$, and for each $i \in \{0, p, n, 2\}$, do the following. Look up $\chi_i^1 := \chi_i(\mathcal{B}_1)$ and $\chi_i^2 := \chi_i(\mathcal{B}_2)$. For $a, b \in \{0, p, n, 2\}$, let $\chi_{ab} := \chi_a^1 \cdot \chi_b^2$. For $a \in \{0, p, n, 2\}$, let $\chi_{a\star} := \max\{\chi_{a0}, \chi_{ap}, \chi_{an}, \chi_{a2}\}$ and $\chi_{\star a} := \max\{\chi_{0a}, \chi_{pa}, \chi_{na}, \chi_{2a}\}$. For $k \in \{1, 2\}$, let $\chi_\star^k := \max\{\chi_0^k, \chi_p^k, \chi_n^k, \chi_2^k\}$.

(a) If $T \in \mathcal{S}_0$, then set $\chi_0 := \max\{\chi_0, \chi_{00}\}$, $\chi_p := \max\{\chi_p, \chi_{0p}, \chi_{p0}, \chi_{pp}\}$, $\chi_n := \max\{\chi_n, \chi_{0n}, \chi_{n0}, \chi_{nn}\}$ and $\chi_2 := \max\{\chi_2, \chi_{pn}, \chi_{np}, \chi_{2\star}, \chi_{\star 2}\}$.

(b) If $T \in \mathcal{S}_p$, then set $\chi_p := \max\{\chi_p, \chi_{00}, \chi_{0p}, \chi_{p0}, \chi_{pp}\}$ and $\chi_2 := \max\{\chi_2, \chi_{n\star}, \chi_{\star n}, \chi_{2\star}, \chi_{\star 2}\}$.

(c) If $T \in \mathcal{S}_n$, then set $\chi_n := \max\{\chi_n, \chi_{00}, \chi_{0n}, \chi_{n0}, \chi_{nn}\}$ and $\chi_2 := \max\{\chi_2, \chi_{p\star}, \chi_{\star p}, \chi_{2\star}, \chi_{\star 2}\}$.

(d) If $T \in \mathcal{S}_2$, then set $\chi_2 := \max\{\chi_2, \chi^1_\star \cdot \chi^2_\star\}$.

Store all the computed values of χ_i along with $\mathcal{B}$.

Finally, at the root bag, output the maximum of all the values of $\chi_2(\mathcal{B})$ over all boundaries $\mathcal{B} = (B, T)$ such that $B(v) \neq y_N$ for all $v \in H$.

Running time

We will use the above algorithm only in the special case when the width of the tree decomposition is in $O(w)$, where $w = |V(H)|$, and so analyze the algorithm only for this case. Since we assumed in the beginning that the tree decomposition has $O(n)$ bags, where $n = |V(G)|$, the running time of the algorithm is $O(n \cdot t)$, where t is the time taken to generate all possible extended boundaries $\mathcal{B}$ and calculate $\chi_i(\mathcal{B})$ for all $i \in \{0, n, p, 2\}$ at each bag. Calculating $\chi_i(\mathcal{B})$ at a bag requires generating all possible extended boundaries at the children of the bag, and then testing for consistency/compatibility with $\mathcal{B}$.

The number of homomorphisms from H to a bag of the tree decomposition is $w^{O(w)} = 2^{O(w \log w)}$ since the width of the tree decomposition is $O(w)$, where $w = |V(H)|$. Therefore, the number of partial isomorph boundaries at each bag is $2^{O(w \log w)}$. The number of separator functions whose domain is the set of vertices in a bag is $3^{O(w)} = 2^{O(w)}$. Therefore, the number of extended partial isomorph boundaries at a bag is $2^{O(w \log w)} \cdot 2^{O(w)} = 2^{O(w \log w)}$. The time taken for testing consistency and compatibility of boundaries is polynomial in w. Therefore, the time taken to calculate $\chi_i(\mathcal{B})$ at each bag is $t = 2^{O(w \log w)}$. Hence, the running time is $O(t \cdot n) = 2^{O(w \log w)} n$.

Thus, when the number of vertices w of the pattern graph is constant and when the width of the tree decomposition is bounded, the running time of the dynamic programming algorithm is linear in n.

Proof of correctness

Lemma 44. *For each bag N and each extended boundary $\mathcal{B}$ at N, the algorithm sets $\chi_i(\mathcal{B}) = 1$ if and only if there is an extended partial isomorph $\mathcal{I}$ of type i at N with $\mathcal{I} \rightsquigarrow \mathcal{B}$.*

Proof. We prove the lemma by induction on the height of bags in the tree decomposition, where the *height of a bag* N is the number of edges in the longest path connecting N with a leaf bag in the subtree rooted at N.

Basis: Let N be a bag at height 0, i.e., let N be a leaf bag. Suppose that $\mathcal{I} = (I, S)$ is an extended partial isomorph of type i at N. From Lemma 38, the summary $\mathcal{B} = (B, S)$ of $\mathcal{I}$ is a leaf boundary. Since $S \in \mathcal{S}_i$, from Step (1) of the algorithm, $\chi_i(\mathcal{B}) = 1$. For the other direction, let $\mathcal{B} = (B, T)$ be a leaf boundary such that the algorithm sets $\chi_i(\mathcal{B}) = 1$ for the first time. From Lemma 41, $\mathcal{I} = (B_N, T)$ is an extended partial isomorph with $\mathcal{I} \rightsquigarrow \mathcal{B}$. Since the algorithm set $\chi_i(\mathcal{B}) = 1$, $T \in \mathcal{S}_i$. Since $T \in \mathcal{S}_i$, $\mathcal{I}$ is of type i.

Induction step: Suppose that the lemma holds true for all bags at height at most $h - 1$. Let N be a bag at height $h > 0$ in the tree. We prove the lemma for the case when N has two children N_1 and N_2, and when $i = 2$; the proof for the remaining cases is similar (and easier) and left to the reader.

Let $\mathcal{I} = (I, S)$ be an extended partial isomorph of type 2 at a bag N with $\mathcal{I} \rightsquigarrow \mathcal{B}$, where $\mathcal{B} = (B, T)$. Let $\mathcal{J}_1 = (J_1, S_1)$ and $\mathcal{J}_2 = (J_2, S_2)$ be the restrictions of $\mathcal{I}$ onto G_{N_1} and G_{N_2} respectively. Let $\mathcal{J}_1 \rightsquigarrow \mathcal{B}_1$ and $\mathcal{J}_2 \rightsquigarrow \mathcal{B}_2$. From Lemma 40, $\mathcal{B}$, $\mathcal{B}_1$ and $\mathcal{B}_2$ form a compatible triple. Depending upon the type of the separator function T, the following scenarios can occur:

- $T \in \mathcal{S}_0$

  Since $\mathcal{I}$ is of type 2, we have $S(u) = -1$ and $S(v) = 1$ for some $u, v \in \mathrm{dom}(S)$. But since $T \in \mathcal{S}_0$, we have $u, v \notin L(N)$. Therefore, u and v are in $G_{N_1} \cup G_{N_2}$. This gives rise to the following sub-cases: (a) one of S_1 and S_2 belong to $\mathcal{S}_2$; (b) $S_1 \in \mathcal{S}_p$ and $S_2 \in \mathcal{S}_n$ (or vice-versa). Correspondingly, by induction hypothesis, one of the following occur: (a) $\chi_2(\mathcal{B}_1) = 1$ or $\chi_2(\mathcal{B}_2) = 1$; (b) $\chi_p(\mathcal{B}_1) = 1$ and $\chi_n(\mathcal{B}_2) = 1$ (or vice-versa). In either case, from Step (3a) of the algorithm, we have $\chi_2(\mathcal{B}) = 1$.

- $T \in \mathcal{S}_p$

  Since $\mathcal{I}$ is of type 2, one similarly argues that one of the following sub-cases occur: (a) one of S_1 and S_2 belong to $\mathcal{S}_2$; (b) one of S_1 and S_2 belong to $\mathcal{S}_n$. Correspondingly,

by induction hypothesis, one of the following occur: (a) $\chi_2(\mathcal{B}_1) = 1$ or $\chi_2(\mathcal{B}_2) = 1$; (b) $\chi_n(\mathcal{B}_1) = 1$ or $\chi_n(\mathcal{B}_2) = 1$. In either case, from Step (3b) of the algorithm, we have $\chi_2(\mathcal{B}) = 1$.

- $T \in \mathcal{S}_n$

  (This is symmetric to the previous case.)

- $T \in \mathcal{S}_2$

  Let $S_1 \in \mathcal{S}_j$ and $S_2 \in \mathcal{S}_k$, where $j, k \in \{0, p, n, 2\}$. By induction hypothesis, $\chi_j(\mathcal{B}_1) = 1$ and $\chi_k(\mathcal{B}_2) = 1$. From Step (3d) of the algorithm, $\chi_2(\mathcal{B}) = 1$.

Thus, in all the cases above, $\chi_2(\mathcal{B}) = 1$.

Now, we prove the other direction. Let $\mathcal{B} = (B, T)$ be an extended boundary at a bag N with two children N_1 and N_2 such that the algorithm sets $\chi_2(\mathcal{B}) = 1$. There are various possible ways in which $\chi_2(\mathcal{B})$ could have been set equal to 1 for the first time. For example, from Step (3a) of the algorithm, one possible way is that $T \in \mathcal{S}_0$ and $\chi_{pn} = 1$. For this case, we show that there exists an extended partial isomorph $\mathcal{I}$ of type 2 such that $\mathcal{I} \rightsquigarrow \mathcal{B}$. The proof for the remaining cases is similar, and we omit it.

From the algorithm and the case assumption, there exist extended boundaries $\mathcal{B}_1$ at N_1 and $\mathcal{B}_2$ at N_2 that are compatible with $\mathcal{B}$, and such that $\chi_p(\mathcal{B}_1) = 1$ and $\chi_n(\mathcal{B}_2) = 1$. By induction hypothesis, there exist extended partial isomorphs $\mathcal{J}_1$ of type p and $\mathcal{J}_2$ of type n such that $\mathcal{J}_1 \rightsquigarrow \mathcal{B}_1$ and $\mathcal{J}_2 \rightsquigarrow \mathcal{B}_2$. From Lemma 43, there exists an extended partial isomorph $\mathcal{I}$ at N such that $\mathcal{I} = \mathcal{B}_N \oplus \mathcal{J}_1 \oplus \mathcal{J}_2$ and $\mathcal{I} \rightsquigarrow \mathcal{B}$. Since $\mathcal{I} = \mathcal{B}_N \oplus \mathcal{J}_1 \oplus \mathcal{J}_2$, and $\mathcal{J}_1$ and $\mathcal{J}_2$ are of of type p and type n, $\mathcal{I}$ is of type 2. This proves the lemma. □

Theorem 45. *The algorithm outputs 1 if and only if there is a separating subgraph isomorphic to H in G.*

Proof. Suppose that the algorithm outputs 1. From the last step of the algorithm, there is an extended boundary $\mathcal{B}$ such that $\chi_2(\mathcal{B}) = 1$ and $B(v) \neq y_N$ for any $v \in H$. From Lemma 44, there is an extended partial isomorph $\mathcal{I}$ of type 2 at the root with $\mathcal{I} \rightsquigarrow \mathcal{B}$. Since $B(v) \neq y_N$ for any $v \in H$, $I(v) \in G_N$ for all $v \in H$. Therefore, I is an isomorphism between H and some subgraph of G. Since $\mathcal{I}$ is of type 2, the subgraph isomorphic to H in G is a separator, with the separated sets of vertices being $F_1 := \{v : S(v) = -1\}$ and $F_2 := \{v : S(v) = 1\}$.

Suppose that there is a separating subgraph X isomorphic to H in G. Let I be the isomorphism from H to X. Since X is separator, $V(G \setminus X) = F_1 \cup F_2$, where F_1 and F_2

are two sets of flaps of $G \setminus X$ such that no vertex of F_1 is adjacent to a vertex of F_2. Let S be the separator function with $\mathrm{dom}(S) = V(G)$ such that $S(u) = 0$ for all $u \in \mathrm{Range}(I)$, $S(u) = -1$ for all $u \in F_1$ and $S(u) = 1$ for all $u \in F_2$. Then $\mathcal{I} = (I, S)$ is a type 2 extended partial isomorph at the root, and the summary $\mathcal{B} = (B, T)$ of $\mathcal{I}$ is such that $B(v) \neq y_N$ for all $v \in H$. From Lemma 44, the algorithm sets $\chi_2(\mathcal{B}) = 1$. This extended boundary causes the algorithm to output 1 in the last step. □

We put everything together into one theorem that we will need in the next section.

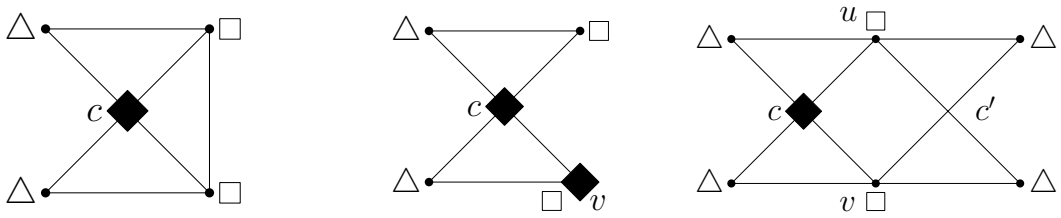
Theorem 46. *Let H be a cycle on w vertices, and G be a graph with a given $O(w)$-width tree decomposition. There exists an algorithm that tests in $2^{O(w \log w)}n$ time whether G contains H as a separator.*

5.2 Constrained separating cycles in graphs of bounded treewidth

In the previous section, we discussed a dynamic programming algorithm to find (unconstrained) separating cycles in graphs of bounded treewidth (Theorem 46). In this section, we explain how to modify the algorithm to find constrained separating cycles in graphs of bounded treewidth.

Recall that for our application, the host graph is the radial planarisation $\Lambda(G)$ of a given bowtie 1-plane graph G. The pattern graph is a bounded length cycle X (length at most 14) that must also satisfy the following constraints.

- (Ψ1): X is a subgraph of $R(G)$ and separates two vertices of G.
- (Ψ2): There exists a *marking* function $\beta : V(G) \mapsto \{\triangle, \square\}$ such that for any crossing point that X visits:
  1. The endpoints of each edge of the crossing are marked with opposite symbols.
  2. The endpoints of the crossing marked $\triangle$ are not adjacent in G (Figure 5.4a).
  3. If an endpoint of the crossing is on X, then the endpoint marked $\square$. (Figure 5.4b).
  4. If no endpoint of the crossing is on X, then the crossing is part of a butterfly crossing with the endpoints marked $\square$ as the axial vertices (Figure 5.4c).



(a) (Ψ2.1) and (Ψ2.2) (b) (Ψ2.3) (c) (Ψ2.4)

Figure 5.4: The marking function. (The filled diamonds denote vertices and crossing points on X.)

(See Section 3.1 for the definition of a butterfly crossing and axial vertices of a butterfly crossing.)

Now, we discuss modifications to the dynamic programming algorithm to incorporate the constraints (Ψ1)-(Ψ2). Let X be a cycle on w vertices. We assume that $\Lambda(G)$ has treewidth $O(w)$, and that we are given an $O(w)$-width tree decomposition of $\Lambda(G)$. (As we will see in Section 5.3, the real input to the dynamic programming algorithm will be a modified bounded treewidth subgraph of $\Lambda(G)$. The input graph will also have a list of edges that need to be excluded from being considered for partial isomorphs and a list of face vertices and/or crossing points that are to be treated like vertices of G. Enforcing these additional restrictions in the dynamic programming algorithm is relatively simple, and we do not take these into consideration for the following discussion.)

Constraint (Ψ1)

To restrict X to be a subgraph of $R(G)$, we define a variant of partial isomorphs and partial isomorph boundaries, called $R(G)$-*partial isomorphs* and $R(G)$-*partial isomorph boundaries*, which are basically partial isomorphs and partial isomorph boundaries that map edges of X only to edges of $R(G)$.

To enforce the constraint that X separates two vertices of G, we define a variant of separator functions, called $V(G)$-*separator functions*, which can distinguish vertices of G from other types of vertices of $\Lambda(G)$ (such as face vertices and crossing points). For a set $U \subseteq V(\Lambda(G))$, a function $S : U \mapsto \{-1, -1/2, 0, 1/2, 1\}$ is a $V(G)$-separator function if: (a) for any $v \in U$ that is a vertex of G, $S(v) \in \{0, -1, 1\}$, and for any other vertex $v \in U$ (i.e., face-vertices, crossing points), $S(v) \in \{0, -1/2, 1/2\}$; (b) for any two vertices u, v with $S(u) > 0$ and $S(v) < 0$, $(u, v) \notin E(\Lambda(G))$. As before, if S is a component $V(G)$-separator function of an $R(G)$-extended partial isomorph or an $R(G)$-extended partial isomorph

boundary, then $S(u) = 0$ if and only if u is in the range of the component $R(G)$-partial isomorph or the component $R(G)$-partial isomorph boundary. Similar to the four classes of separator functions seen before, we define four classes of $V(G)$-separator functions $\mathcal{S}_0$, $\mathcal{S}_p$, $\mathcal{S}_n$ and $\mathcal{S}_2$. (We retain the same names so that the dynamic program can be applied verbatim.)

- $\mathcal{S}_0 = \{S : S(u) \in \{0, -1/2, 1/2\}$ for all $u \in \text{dom}(S)\}$.
- $\mathcal{S}_p = \{S : S(u) \in \{0, -1/2, 1/2, 1\}$ for all $u \in \text{dom}(S)$, and there exists a vertex u with $S(u) = 1\}$.
- $\mathcal{S}_n = \{S : S(u) \in \{0, -1/2, 1/2, -1\}$ for all $u \in \text{dom}(S)$, and there exists a vertex u with $S(u) = -1\}$.
- $\mathcal{S}_2 = \{S :$ there exist two vertices u, v with $S(u) = -1$ and $S(v) = 1\}$.

With these changes, the dynamic programming algorithm outputs 1 if and only if there is a type 2 $R(G)$-extended partial isomorph at the root of the tree decomposition. If such an $R(G)$-extended partial isomorph exists, then the component $R(G)$-partial isomorph gives a subgraph of $R(G)$ that is isomorphic to X, and the component $V(G)$-separator function gives the set of vertices of G that are separated by the $R(G)$-partial isomorph.

Constraint ($\Psi 2$)

To implement ($\Psi 2$), we assume that for each crossing point, we have a list that tells which pairs of its endpoints (if any) are the axial vertices of a butterfly crossing. This list is of size at most two, and can be computed in total linear time for all crossing points of $\Lambda(G)$.

As a pre-processing step, we modify the tree decomposition of $\Lambda(G)$ as follows. For each bag that contains a crossing point, include all the four endpoints of the crossing in the bag. This step yields another tree decomposition (Claim 36) with a width that increases at most by a factor of 5, and is hence still in $O(w)$.

To implement ($\Psi 2$), we expand the definitions of $R(G)$-extended partial isomorphs and $R(G)$-extended boundaries by introducing *partial marking functions*. The domain of a partial marking function β is a subset of $V(G)$ and the range is the set $\{\triangle, \square\}$. A *marking $R(G)$-extended partial isomorph* at a bag N is a 3-tuple of functions $\mathcal{I} = (I, \{S, \beta_I\})$, where I is an $R(G)$-partial isomorph, S is a $V(G)$-separator function, and β_I is a partial marking function such that $\text{dom}(\beta_I) = \text{dom}(S) \cap V(G)$. A *marking $R(G)$-extended partial isomorph*

boundary at a bag N is a 3-tuple of functions $\mathcal{B} = (B, \{T, \beta_B\})$, where B is an $R(G)$-partial isomorph boundary, T is a $V(G)$-separator function, and β_B is a partial marking function such that $\text{dom}(\beta_B) = \{v \in L(N)\}$. In addition, a marking $R(G)$-extended partial isomorph boundary $\mathcal{B} = (B, \{T, \beta_B\})$ must satisfy the following conditions that express ($\Psi 2$). If a crossing point c is in the range of B, then

1. The endpoints of each edge of the crossing are marked (by β_B) with opposite symbols.
2. The endpoints of the crossing marked $\triangle$ are not adjacent in G (Figure 5.4a).
3. If an endpoint of the crossing is in the range of B, then the endpoint is marked $\square$. (Figure 5.4b).
4. If no endpoint of the crossing is in the range of B, then the crossing is part of a butterfly crossing with the endpoints marked $\square$ as the axial vertices (Figure 5.4c).

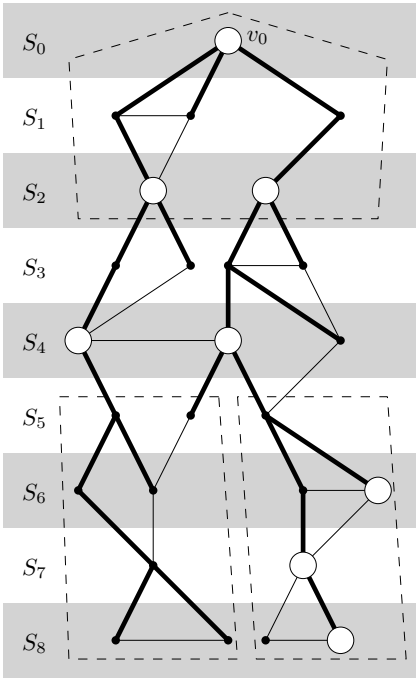
The first three conditions are easy to test since each bag that contains a crossing point also contains the four endpoints of the crossing. To test the fourth condition, we use the list that we created for each crossing point that tells which pairs of its endpoints are the axial vertices of a butterfly crossing.

The summary of a marking $R(G)$-extended partial isomorph is defined as before: if $\mathcal{I} = (I, \{S, \beta_I\})$ and $\mathcal{B} = (B, \{T, \beta_B\})$, then $\mathcal{I} \rightsquigarrow \mathcal{B}$ if and only if: (a) $I \rightsquigarrow B$; (b) T is the restriction of S onto $L(N)$; (c) β_B is the restriction of β_I onto $L(N)$. For two marking $R(G)$-extended boundaries to be consistent/compatible, we additionally require that the component marking functions agree on their shared domain.

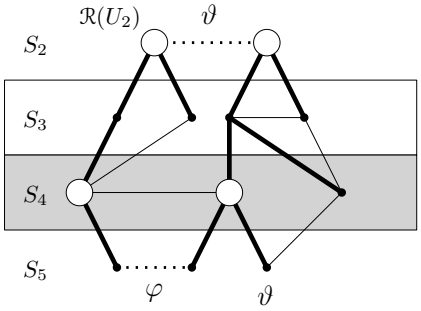
With these changes in place, the dynamic programming algorithm outputs 1 if and only if there is a constrained separating cycle isomorphic to X in $\Lambda(G)$.

5.3 The separating subgraph isomorphism algorithm

In the previous section, we saw that when $\Lambda(G)$ has a tree decomposition of bounded width, we can find a bounded length constrained separating cycle in linear time. In this section, we show that even when $\Lambda(G)$ does not have a bounded treewidth, we can find a bounded length constrained separating cycle in linear time. We achieve this by partitioning $\Lambda(G)$ into pieces of bounded treewidth, and then searching each of the pieces for a constrained



(a) Bold edges show $T(v_0)$, and the white circles are vertices of G. The dashed edges show $\Lambda[S_0 \cup \cdots \cup S_2]$ and $\Lambda[S_5 \cup \cdots \cup S_8]$.



(b) Construction of Λ_3 with $w = 2$. The dotted edges are added to obtain U_2 and L_5.

Figure 5.5: Construction of the graphs Λ_i

separating cycle using dynamic programming. The ideas for this are taken from [Epp99], but the details are quite different. See Appendix A for further discussion.

As before, our pattern graph is a bounded length cycle on w vertices. We give a procedure to partition the host graph $\Lambda(G)$ into pieces that have treewidth $O(w)$. Let v_0 be a vertex of G in $\Lambda(G)$. For $i \geq 0$, let S_i consist of the vertices at distance i from v_0. Note that S_i is the set of vertices at depth i in the breadth first search (BFS) tree with v_0 as the root (Figure 5.5a). We denote this BFS tree by $T(v_0)$. Let S_p be the last layer of the BFS tree. When $i < 0$ and $i > p$, we let $S_i = \emptyset$. For each $i \geq 0$, we construct a graph Λ_i on the vertex set $S_{i-1} \cup \cdots \cup S_{i+w}$, where Λ_i is the union of the following graphs: (a) a graph U_{i-1} (defined below) on the vertex set S_{i-1}, (b) the subgraph of $\Lambda(G)$ induced by $S_{i-1} \cup \cdots \cup S_{i+w}$, and (c) a graph L_{i+w} (defined below) on the vertex set S_{i+w}. (Henceforth, for notational convenience, we write $\Lambda[S_i \cup \cdots \cup S_j]$ to denote the subgraph of $\Lambda(G)$ induced by $S_i \cup \cdots \cup S_j$.) As we will show below, our construction of L_{i+w} is done such that two

vertices of L_{i+w} are in the same component if and only if there is a path connecting the two vertices that uses only vertices of $S_t, t \geq i+w$. Similarly, two vertices of U_{i-1} will be in the same component if and only if there is a path connecting the two vertices that uses only vertices of $S_t, t \leq i-1$.

In essence, each component of L_{i+w} (resp. U_{i-1}) represents the contraction of a component of $\Lambda(G)$ induced by $S_t, t \geq i+w$ (resp. $S_t, t < i$) into the layer S_{i+w} (resp. S_{i-1}) (Figure 5.5b). If this component of $\Lambda(G)$ contains a vertex of $V(G)$, we label all the vertices of the corresponding component of L_{i+w} (resp. U_{i-1}) with the symbol ϑ (we will use this later to indicate that these vertices should be treated as if they were original vertices of G, hence the v-like symbol). The set of edges in L_i and U_i, and the labels ϑ for their components, are computed for all S_i in a single pass through the layers of $S_0, \ldots, S_p$.

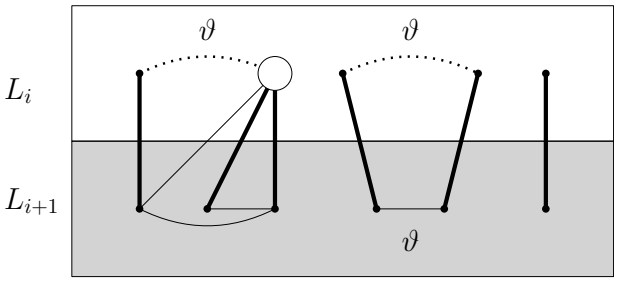


Figure 5.6: Constructing L_i inductively. The bold edges belong to $T(v_0)$ and the white circles are vertices of G. The dotted edges are added to L_i. The label ϑ for the components denote the labels of all vertices of that component.

We construct L_i inductively in the reverse order, starting from the last layer S_p. For S_p, the graph L_p is simply $\Lambda[S_p]$. If a component of L_p contains a vertex of $V(G)$, label all vertices of the component with ϑ. Suppose that L_{i+1} has been constructed for the layer S_{i+1}, where $i+1 \leq p$, and the label ϑ is assigned for its components. We describe the construction of L_i for the layer S_i (Figure 5.6). Initially, add all the edges of $\Lambda[S_i]$ to L_i. For each vertex u in S_{i+1}, let $N_i(u)$ be the set of neighbours of u in S_i. Let $\pi(u)$ be the parent of u in $T(v_0)$. Add edges to L_i that connect $\pi(u)$ to $N_i(u) \setminus \{\pi(u)\}$. If u is labelled ϑ, mark $\pi(u)$ with ϑ. Next, for every edge (u, v) in L_{i+1}, add an edge to L_i connecting $\pi(u)$ and $\pi(v)$, unless $\pi(u) = \pi(v)$. Finally, remove all parallel edges from L_i by doing a bucket sort on the adjacency list of each vertex of L_i. Compute all connected components of L_i and for each component, if there is a vertex of $V(G)$ or if there is a vertex marked ϑ, label all vertices of the component with ϑ.

Lemma 47. *The total time for computing L_i for all the layers S_i is $O(n)$.*

Proof. The construction of L_i imitates the process of contracting the edges of $T(v_0)$ in $L_{i+1} \cup \Lambda[S_i \cup S_{i+1}]$. Hence, each L_i is a minor of $\Lambda(G)$. Since $\Lambda(G)$ is planar and has no bigons, and L_{i+1} is planar (being a minor of $\Lambda(G)$) and simple, the number of edges in $L_{i+1} \cup \Lambda[S_i \cup S_{i+1}]$ is $O(|S_i| + |S_{i+1}|)$. Therefore, by construction, the number of edges in L_i is $O(|S_i| + |S_{i+1}|)$. Hence, the time taken to construct L_i, remove parallel edges from L_i by bucket sort, and compute and parse connected components is $O(|S_i| + |S_{i+1}|)$. Note that we compute L_i for all the layers S_i in a single pass through the layers $S_0, \ldots, S_p$. Thus, the total time to construct all the L_i is $O(\sum_{i=0}^{p} |S_i|) = O(n)$. □

As the construction of L_i imitates the process of contracting the edges of $T(v_0)$ in $L_{i+1} \cup \Lambda[S_i \cup S_{i+1}]$, by induction, two vertices are in the same component of L_i if and only if they are connected by a path that only uses vertices of layers $S_t, t \geq i$.

Likewise, to construct U_i, we must add edges to $\Lambda[S_i]$ so that two vertices of S_i are in the same component if and only if they are connected by a path that only uses vertices of the layers $S_t, t \leq i$. Note that U_i must have a single component since each vertex of S_i is connected to every other vertex of S_i via a path through the root v_0 of $T(v_0)$. Moreover, all vertices of the component must be labelled ϑ since v_0 was chosen to be a vertex of G. Therefore, to construct U_i, fix an arbitrary vertex of S_i, calling it $\mathcal{R}(U_i)$, and connect it to all the other vertices v of S_i.

Lemma 48. *The total time for computing U_i for all the layers S_i is $O(n)$.*

Proof. The time to construct U_i for each layer is simply $O(|S_i|)$. Therefore, the time taken to compute U_i for all the layers is $O(\sum_{i=0}^{p} |S_i|) = O(n)$. □

Having constructed L_i and U_i for each layer S_i, we obtain a graph Λ_i by taking the union of U_{i-1}, $\Lambda[S_{i-1} \cup \cdots \cup S_{i+w}]$ and L_{i+w}. As we will see in Lemma 52, testing whether $\Lambda(G)$ has a constrained separating cycle is equivalent to testing whether some Λ_i has a constrained separating cycle. We first show that each Λ_i has treewidth $O(w)$, and that an $O(w)$-width tree decomposition of Λ_i can be computed in time $O(w|\Lambda_i|)$. To prove this, we will need the following lemma.

Lemma 49 ([Bak94] and [Epp99, Lemma 4])**.** *If a planar graph G has a rooted spanning tree of height at most l, then a tree decomposition of G with width at most $3l$ can be found in time $O(ln)$, where $n = |V(G)|$.*

Now, we prove the following lemma.

Lemma 50. *Each Λ_i has treewidth $O(w)$, and an $O(w)$-width tree decomposition of Λ_i can be computed in time $O(w|\Lambda_i|)$.*

Proof. From Lemma 49, it is sufficient to show that Λ_i is planar and has a spanning tree of height $O(w)$. First we show that Λ_i is planar. The graph L_{i+w} can be thought of as being obtained by contracting the edges of $T(v_0)$ whose endpoints are in $S_t, t \geq i+w$, and then simplifying the resulting graph. On the other hand, the graph U_{i-1} can be thought of as being obtained by contracting all edges of $\Lambda(G)$ with endpoints in $S_t, t < i-1$ into a supervertex, and then contracting the supervertex into $\mathcal{R}(U_{i-1})$. Therefore, Λ_i is a minor of $\Lambda(G)$. Since planar graphs are closed under taking minors, Λ_i is also planar.

Note that the edges incident with $\mathcal{R}(U_{i-1})$ in U_{i-1} together with the edges of $T(v_0)$ in $\Lambda[S_i \cup \ldots S_{i+w}]$ form a spanning tree of Λ_i of height $w+2$. Therefore, by Lemma 49, Λ_i has treewidth $O(w)$, and an $O(w)$-width tree decomposition of Λ_i can be computed in time $O(w|\Lambda_i|)$. □

In Lemma 51, we show that the total time to compute all the Λ_i and their $O(w)$-width tree decompositions is linear in n, where n is the number of vertices in the given bowtie 1-plane graph G.

Lemma 51. *The total time for finding the Λ_i and computing their $O(w)$-width tree decompositions is $O(w^2 n)$.*

Proof. To compute Λ_i, we first need to compute $\Lambda(G)$, then the BFS tree $T(v_0)$, and then the graphs L_i and U_i for each layer S_i. Since G is a bowtie 1-plane graph on n vertices without bigons, the number of edges in G is $O(n)$. This implies that the number of vertices and edges of $\Lambda(G)$ is $O(n)$. Hence, the time to compute $\Lambda(G)$, and its BFS tree $T(v_0)$, is $O(n)$. Since computing L_i and U_i for all the layers take $O(n)$ time in total (Lemma 47 and 48), the time taken to compute all the Λ_i is $O(n)$. Since the time for finding a tree decomposition of width $O(w)$ for each Λ_i is $O(w|\Lambda_i|)$ (Lemma 50), the total time for finding all Λ_i and computing their $O(w)$-width tree decompositions is $O(n) + O(w\sum_{i=0}^{p}|\Lambda_i|) = O(w^2 n)$ since each vertex of $\Lambda(G)$ is included in at most $w+2$ graphs of Λ_i (as each Λ_i intersects only $w+2$ layers of $S_0, \ldots, S_p$). □

5.3.1 Finding a constrained separating cycle in $\Lambda(G)$

Suppose that $\Lambda(G)$ has a constrained separating cycle X on w vertices. Let $S_i, \ldots, S_{i+w-1}$ be w consecutive layers of $\Lambda(G)$ that contain X (they may not be unique). Consider the

graph $\Lambda_i = U_{i-1} \cup \Lambda[S_{i-1} \cup \cdots \cup S_{i+w}] \cup L_{i+w}$. Let X_i denote the cycle in Λ_i isomorphic to X. We will show that X_i is also a constrained separating cycle of Λ_i (under some suitable assumptions).

First consider the constraint ($\Psi 2$). Since X_i is restricted only to $S_i \cup \cdots \cup S_{i+w-1}$, it does not contain any vertex of L_{i+w} or U_{i-1}. So for every crossing point that X_i visits, all the four endpoints of the crossing are in Λ_i. Let β be the marking function that satisfies ($\Psi 2$) for X in $\Lambda(G)$. Then the restriction of β on Λ_i must also satisfy ($\Psi 2$) for X_i in Λ_i. Therefore, X_i satisfies ($\Psi 2$). Next, consider the constraint ($\Psi 1$). Since X is a subgraph of $R(G)$, X_i must also be a subgraph of $R(G)$ in Λ_i. Let u, v be two vertices of G that X separates in $\Lambda(G)$. If both u, v are in Λ_i, then X_i also separates u, v in Λ_i. Suppose that one of the vertices, say u, is not in Λ_i. Then there must be a component of $\Lambda(G)$ induced by $S_t, t > i + w$ or $S_t, t < i - 1$ that contains u. Without loss of generality, suppose that $u \in \Lambda[S_{i+w+1} \cup \cdots \cup S_p]$. Let $C(u)$ denote the component of $\Lambda[S_{i+w+1} \cup \cdots \cup S_p]$ that contains u. Since u is a vertex of G, by construction, all vertices of $C(u)$ must be labelled ϑ. Then the cycle X_i either separates $C(u)$ from v, or separates $C(u)$ from $C(v)$, where $C(v)$ is the analogue of $C(u)$ for v. (Note that $v \notin C(u)$ and $C(u) \neq C(v)$, since otherwise this would imply that u and v are connected by a path outside $\Lambda[S_i \cup \cdots \cup S_{i+w-1}]$ contradicting the fact that X separates u and v in $\Lambda(G)$.) This shows that if the vertices of the components of L_{i+w} and U_{i-1} labelled ϑ are treated like vertices of G, then X_i separates two vertices of G in Λ_i. Therefore, X_i satisfies ($\Psi 1$) in Λ_i. In summary, X_i is a constrained separating cycle of Λ_i if the vertices labelled ϑ are considered vertices of G.

The other direction is similar. Suppose that Λ_i contains a constrained separating cycle X_i that only uses the edges of $\Lambda[S_i \cup \cdots \cup S_{i+w-1}]$ and that separates two vertices of G in Λ_i or two vertices of $\Lambda(G)$ labeled ϑ, then the corresponding cycle X must be a constrained separating cycle of $\Lambda(G)$ (details are left to the reader).

We summarise this through the following lemma.

Lemma 52. *$\Lambda(G)$ has a constrained separating cycle of length w if and only if some Λ_i, for $0 \leq i \leq p$, has a constrained separating cycle in $\Lambda[S_i \cup \cdots \cup S_{i+w-1}]$ assuming that vertices marked ϑ are considered vertices of G (for purposes of $\Psi 1$).*

By Lemma 52, the search for constrained separating cycles in $\Lambda(G)$ can be restricted to the bounded treewidth graphs Λ_i. Theorem 53 shows that searching for a bounded length constrained separating cycle in all of Λ_i takes $O(n)$ time in total.

Theorem 53. *Let X be a cycle on w vertices. If $\Lambda(G)$ contains X as a constrained separating cycle, then the subgraph of $\Lambda(G)$ isomorphic to X can be found in time $2^{O(w \log w)}n$.*

Proof. From Lemma 51, all the graphs Λ_i and their $O(w)$-width tree decompositions can be computed in time $O(w^2 n)$. For each Λ_i and its tree decomposition, use the dynamic programming algorithm (Section 5.2) to find a constrained separating cycle isomorphic to X, but with a small modification: pre-process the graph Λ_i so that all vertices labelled ϑ are treated as vertices of G, and restrict the range of a partial isomorph only to the edges of $\Lambda[S_i \cup \cdots \cup S_{i+w-1}]$. The dynamic programming algorithm takes time $2^{O(w \log w)} \cdot |\Lambda_i|$. Therefore, the total running time is $O(w^2 n) + \sum 2^{O(w \log w)} \cdot |\Lambda_i| = O(w^2 n) + 2^{O(w \log w)} \cdot \sum |\Lambda_i|$, which is equal to $O(w^2 n) + 2^{O(w \log w)} \cdot O(wn)$, because each vertex of G is included in at most $w + 2$ subgraphs of Λ_i. Thus, the total running time is $2^{O(w \log w)} n$. □

5.4 Computing vertex connectivity in linear time

We now put things together and give an algorithm to test the vertex connectivity of bowtie 1-plane graphs in linear time (Figure 5.7). Let G be a given simple bowtie 1-plane graph. Compute all kite edges and the radial planarisation $\Lambda(G)$. This can be done in linear time (Section 3.3). For each $w = 2, 4, \ldots, 14$, use the separating subgraph isomorphism algorithm to test for a constrained separating cycle of length w (Theorem 53 and Figure 5.7) [1] Since a 1-plane graph is at most 7-connected, Theorem 3 ensures that $\Lambda(G)$ must contain a constrained separating cycle of length at most 14. Output the shortest constrained separating cycle together with the corresponding separator and marking functions. Then use Theorem 2 to extract a separating set from the constrained separating cycle and the marking function. From Theorem 2 and 3, the separating set must be a minimum separating set of G. This proves Theorem 1, and shows that the vertex connectivity of a bowtie 1-plane graph can be computed in linear time.

[1] Actually, $w = 6, 8, 10, 12$ would be enough because k-connectivity can be tested in linear time for $k \leq 3$. If $w \leq 12$ fails, then G is 7-connected, and a vertex of degree 7 can be found in linear time.

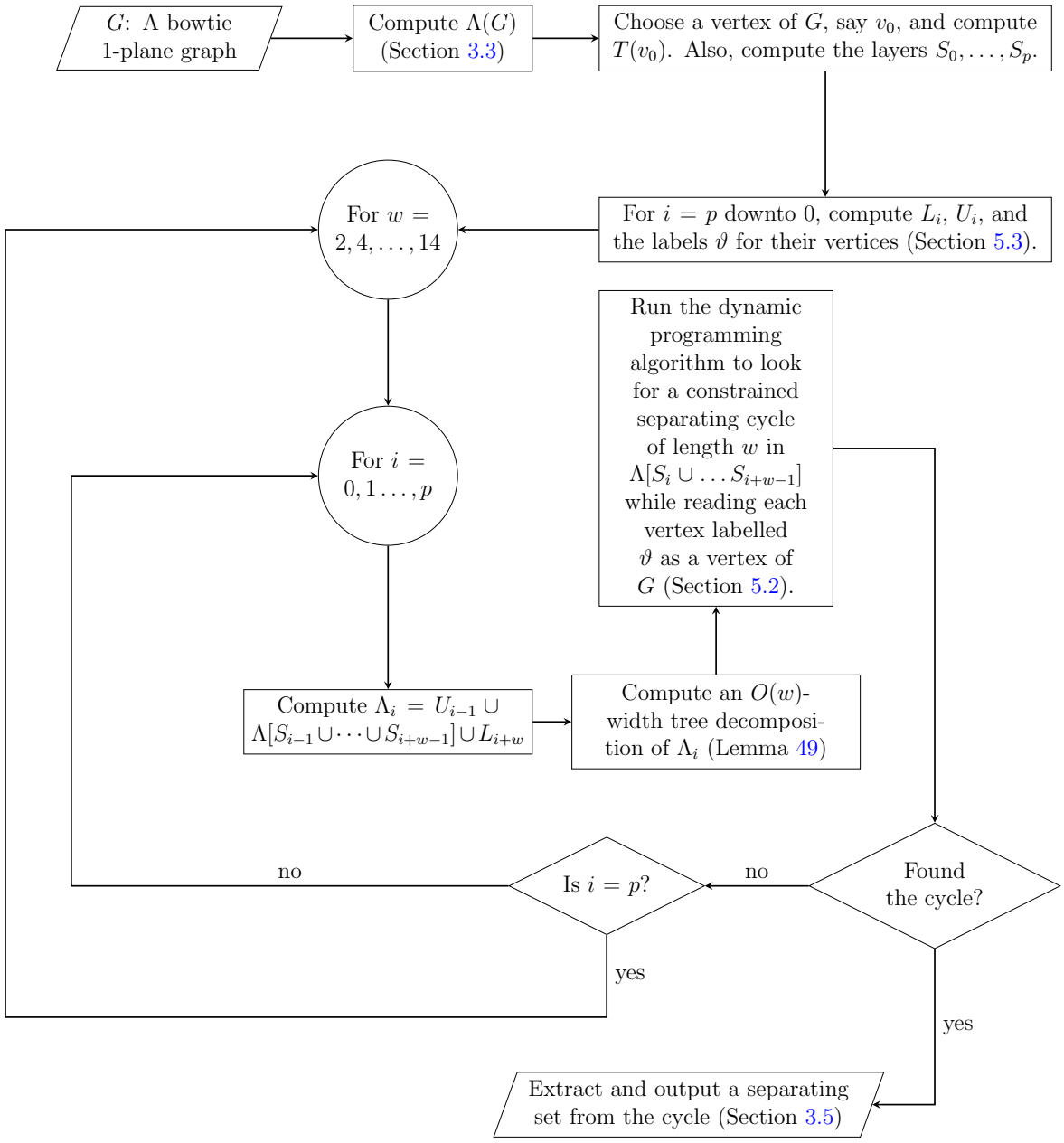


Figure 5.7: The algorithm for computing vertex connectivity of bowtie 1-plane graphs in linear time.

Chapter 6

Conclusion

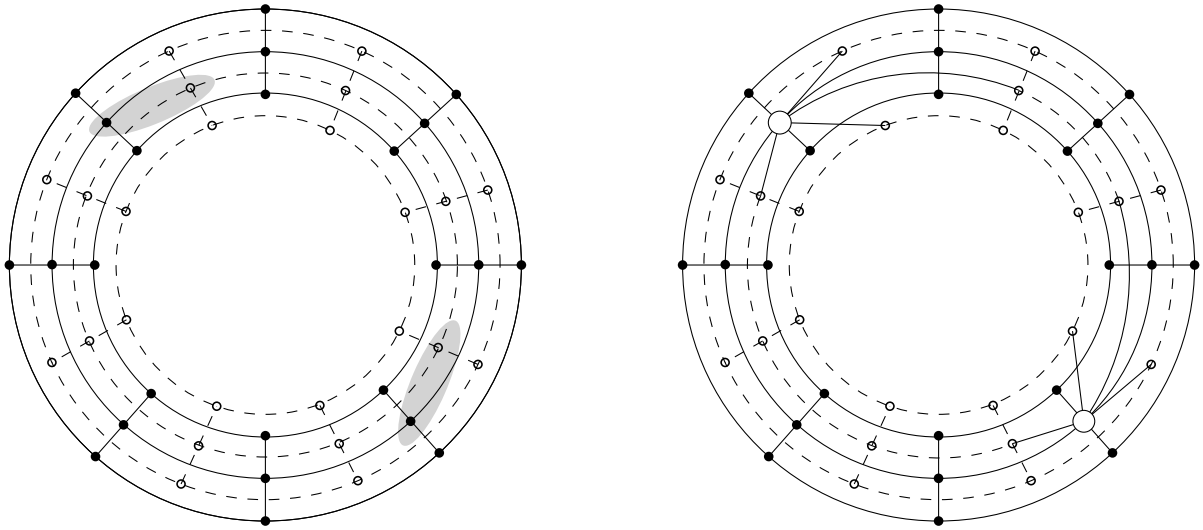
In this thesis, we explored the correspondence between separating sets and separating cycles of bowtie 1-plane graphs. More specifically, we proved that every separating set of size k in a bowtie 1-plane graph G corresponds to a constrained separating cycle of length at most $2k$ in the radial planarisation $\Lambda(G)$ (Theorem 2 and Theorem 3). This correspondence implies that the vertex connectivity of G is half the length of a shortest constrained separating cycle of $\Lambda(G)$. To find a shortest constrained separating cycle of $\Lambda(G)$, we used techniques from the separating subgraph isomorphism algorithm of [Epp99], where we partition $\Lambda(G)$ into pieces of bounded treewidth graphs and use dynamic programming to search for constrained separating cycles in each of the pieces. The algorithm runs in linear time (Theorem 4) since the length of a shortest constrained separating cycle is bounded (it is at most 14, since the vertex connectivity of a 1-planar graph is at most 7). Thus, the vertex connectivity of a bowtie 1-plane graph can be computed in linear time (Theorem 1).

It is unlikely that this correspondence between separating sets and separating cycles extends to all 1-plane graphs. Consider Figure 6.1a for example. The figure shows a 1-plane graph where the endpoints of every crossing induce $K_{1,3} \cup \{e\}$ (and hence are not bowtie crossings). The graph is 4-connected and a minimum separating set is shown by vertices marked with white circles. Figure 6.1b shows the radial planarisation of the graph, and one can see that there can be no 8-cycle in $R(G)$ that connects all the four white vertices.

In the example of Figure 6.1, even though the vertices of the separating set do not lie on a short cycle, they lie close to each other in the radial planarisation. This however does not extend to arbitrary 1-plane graphs. Consider the example in Figure 6.2. Figure 6.2a shows two copies of a graph that are interleaved to produce a 1-planar embedding such that the endpoints of each crossing form an independent set. When these two graphs are fused

(a) (b)

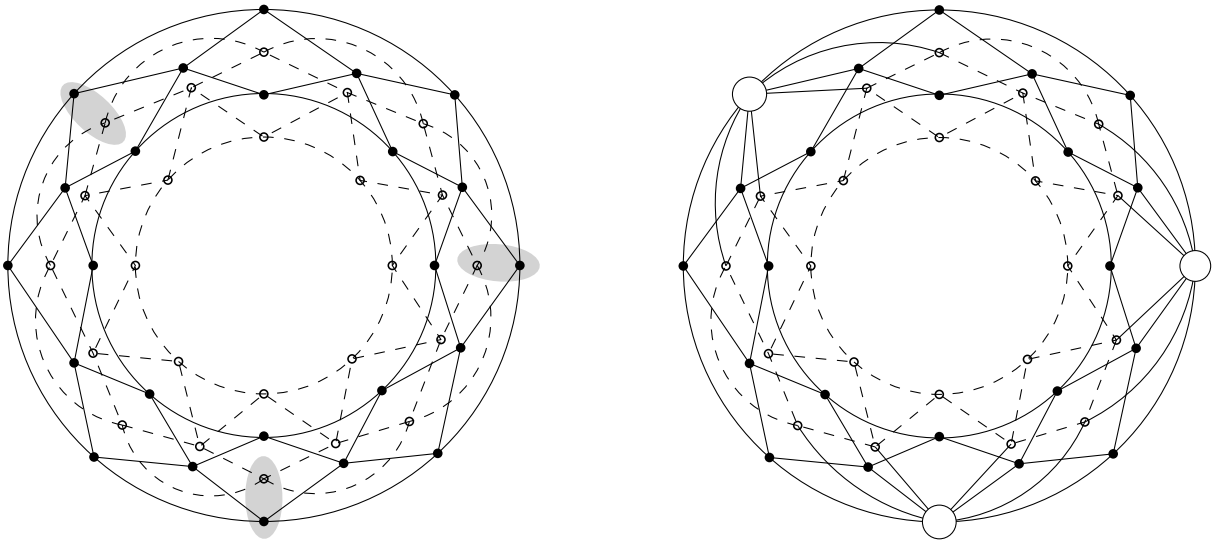
Figure 6.1: An example of a 4-connected 1-plane graph that does not have a separating 8-cycle in its radial planarisation passing through a minimum separating set (white circles).



(a) A 1-planar embedding of two interleaved 3-connected graphs.

(b) A 1-plane graph formed by identifying two pairs of vertices.

Figure 6.2: An example of a 1-plane graph where the vertices of a minimum separating set are far apart from each other.



(a) A 2-planar embedding of two interleaved 4-connected graphs

(b) A 1-plane graph formed by identifying three pairs of vertices.

Figure 6.3: An example of a 2-plane graph where the vertices of a minimum separating set are far apart from each other.

together by identifying two pairs of vertices where one pair is diametrically opposite to the other (shown by grey blobs in Figure 6.2a), we get the graph in Figure 6.2b. The two fused vertices form a separating set of the graph. Moreover, this is the only separating set since both the graphs in Figure 6.2a are 3-connected. This example can be extended (by adding more concentric layers and more vertices within each layer) to show that the distance between the two fused vertices can be made arbitrarily large in the radial planarisation of the graph.

In the example of Figure 6.2, the vertices of the separating set lie far away from each other; however, the graph is only 2-connected, and the separating pair of vertices can be obtained in linear time ([HT73]). We tried to extend this example to 3-connected 1-planar graphs but could not find an example with a unique separating set of size 3. If we allow for some edges to be crossed twice (i.e., the graph is 2-planar), we can construct a 3-connected graph with a unique separating set of size 3, where the distance between the three vertices is not bounded. Consider Figure 6.3 for example, which is constructed similar to Figure 6.2. Figure 6.3a shows two copies of a 4-connected graph that are interleaved to produce a 2-planar embedding. When three pairs of vertices of this graph are identified together, we get the 2-planar graph in Figure 6.3b. By adding more concentric layers and more vertices

within each layer, we can make the the distance between vertices of the separating set arbitrarily large.

We conclude this section with a some open questions. We begin with the most natural question.

Open problem 1. *Let G be an arbitrary 1-plane graph. Can the vertex connectivity of G be computed in linear time?*

Throughout the thesis, we assumed that the input was always a 1-plane graph, that is, a 1-planar graph with a given 1-planar embedding. This is because testing 1-planarity in general is NP-hard [KM13]. However, some classes of 1-planar graphs, such as *optimal 1-planar graphs*, can be recognised in linear time. A 1-planar graph is called *optimal* if it has exactly $4n - 8$ edges, which is the maximum possible for any 1-planar graph. Given any graph G with $4n - 8$ edges, there exists an $O(n)$ time algorithm to decide whether G is optimal 1-planar, and if it is optimal 1-planar, then the algorithm returns a valid embedding of G [Bra18]. Since an optimal 1-planar graph G must be full 1-planar, we can obtain a full 1-planar embedding of G in linear time, and then run our linear time algorithm to compute the vertex connectivity of G. But for arbitrary 1-planar graphs, we have no algorithm to find a 1-planar embedding, and so the following question is natural.

Open problem 2. *Let G be a given 1-planar graph (without a 1-planar embedding). Can the vertex connectivity of G be computed in linear time?*

This thesis explored the connection between vertex connectivity of a 1-planar graph and its geometric properties, and used it to obtain a linear time algorithm for vertex connectivity of bowtie 1-planar graphs. As part of future work, it would be interesting to explore other structural properties (related to graph cuts) such as edge connectivity and minimum bisection for graph classes that are defined via geometric properties such as k-planar graphs, quasi-planar graphs and fan-planar graphs (see [HT20]).

References

[AHU74] Alfred V Aho, John E Hopcroft, and Jeffrey D Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Bak94] Brenda S Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

[BM21] Therese Biedl and Karthik Murali. Computing the vertex connectivity of a locally maximal 1-plane graph in linear time. *CoRR*, abs/2112.06306, 2021.

[Bra18] Franz J Brandenburg. Recognizing optimal 1-planar graphs in linear time. *Algorithmica*, 80(1):1–28, 2018.

[BSW83] Rainer Bodendiek, Heinz Schumacher, and Klaus Wagner. Bemerkungen zu einem Sechsfarbenproblem von G Ringel. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 53:41–52, 1983.

[CHGK14] Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. Distributed connectivity decomposition. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, pages 156–165, 2014.

[CKL+22] Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *CoRR*, abs/2203.00671, 2022.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

[CR94] Joseph Cheriyan and John H Reif. Directed s-t numberings, rubber bands, and testing digraph k-vertex connectivity. *Combinatorica*, 14(4):435–451, 1994.

[EGIS98] David Eppstein, Zvi Galil, Giuseppe F Italiano, and Thomas H Spencer. Separator-based sparsification ii: Edge and vertex connectivity. *SIAM Journal on Computing*, 28(1):341–381, 1998.

[Epp99] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3):1–27, 1999.

[FNY+20] Sebastian Forster, Danupon Nanongkai, Liu Yang, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 2046–2065. SIAM, 2020.

[FT06] Fedor V Fomin and Dimitrios M Thilikos. New upper bounds on the decomposability of planar graphs. *Journal of Graph Theory*, 51(1):53–81, 2006.

[FY19] Sebastian Forster and Liu Yang. A faster local algorithm for detecting bounded-size cuts with applications to higher-connectivity problems. *CoRR*, abs/1904.08382, 2019.

[Gab06] Harold N Gabow. Using expander graphs to find vertex connectivity. *Journal of the ACM (JACM)*, 53(5):800–844, 2006.

[Geo10] Loukas Georgiadis. Testing 2-vertex connectivity and computing pairs of vertex-disjoint s-t paths in digraphs. In *International Colloquium on Automata, Languages, and Programming*, pages 738–749. Springer, 2010.

[GH20] Lukas Gianinazzi and Torsten Hoefler. Parallel planar subgraph isomorphism and vertex connectivity. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 269–280, 2020.

[GLN+19] Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Deterministic graph cuts in subquadratic time: Sparse, balanced, and k-vertex. *CoRR*, abs/1910.07950, 2019.

[GM00] Carsten Gutwenger and Petra Mutzel. A linear time implementation of SPQR-trees. In *Graph Drawing*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 2000.

[HIK+17] Jacob Holm, Giuseppe F Italiano, Adam Karczmarz, Jakub Lacki, Eva Rotenberg, and Piotr Sankowski. Contracting a planar graph efficiently. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 50:1–50:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[HRG00] Monika R Henzinger, Satish Rao, and Harold N Gabow. Computing vertex connectivity: new bounds from old techniques. *Journal of Algorithms*, 34(2):222–250, 2000.

[HT73] John E Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.

[HT74] John E Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM (JACM)*, 21(4):549–568, 1974.

[HT08] Bernhard Haeupler and Robert E Tarjan. Planarity algorithms via PQ-trees. *Electronic Notes in Discrete Mathematics*, 31:143–149, 2008.

[HT20] Seok-Hee Hong and Takeshi Tokuyama, editors. *Beyond Planar Graphs, Communications of NII Shonan Meetings*. Springer, 2020.

[Kle69] Daniel Kleitman. Methods for investigating connectivity of large graphs. *IEEE Transactions on Circuit Theory*, 16(2):232–233, 1969.

[KLM17] Stephen Kobourov, Giuseppe Liotta, and Fabrizio Montecchiani. An annotated bibliography on 1-planarity. *Computer Science Review*, 25:49–67, 2017.

[Klo94] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[KLS20] Tarun Kathuria, Yang P Liu, and Aaron Sidford. Unit capacity maxflow in almost $O(m^{4/3})$ time. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 119–130. IEEE, 2020.

[KM13] Vladimir P Korzhik and Bojan Mohar. Minimal obstructions for 1-immersions and hardness of 1-planarity testing. *Journal of Graph Theory*, 72(1):30–71, 2013.

[KR91] Arkady Kanevsky and Vijaya Ramachandran. Improved algorithms for graph four-connectivity. *J. Comput. Syst. Sci.*, 42(3):288–306, 1991.

[Lau90] Jean-Paul Laumond. Connectivity of plane triangulations. *Information Processing Letters*, 34(2):87–96, 1990.

[LLW88] Nathan Linial, Laszlo Lovasz, and Avi Wigderson. Rubber bands, convex embeddings and graph connectivity. *Combinatorica*, 8(1):91–102, 1988.

[LNP+21] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in polylogarithmic max-flows. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 317–329, 2021.

[MR92] Gary L Miller and Vijaya Ramachandran. A new graph triconnectivity algorithm and its parallelization. *Combinatorica*, 12(1):53–76, 1992.

[MT01] Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins series in the mathematical sciences. Johns Hopkins University Press, 2001.

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

[PT97] János Pach and Géza Tóth. Graphs drawn with few crossings per edge. *Combinatorica*, 17(3):427–439, 1997.

[Rin65] Gerhard Ringel. Ein Sechsfarbenproblem auf der Kugel. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 29(1):107–117, 1965.

[Tar72] Robert E Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[TV85] Robert E Tarjan and Uzi Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14(4):862–874, 1985.

APPENDIX

Appendix A

List of corrections to Eppstein's paper

The contents of Chapter 5 are strongly inspired by Eppstein's paper [Epp99]. A major contribution of this chapter is a detailed explanation of the dynamic programming algorithm, which was omitted from [Epp99]. The process of expanding out the details of the dynamic programming algorithm revealed some errors in [Epp99]. In this section, we list the main errors and discuss how these were rectified and incorporated in Chapter 5.

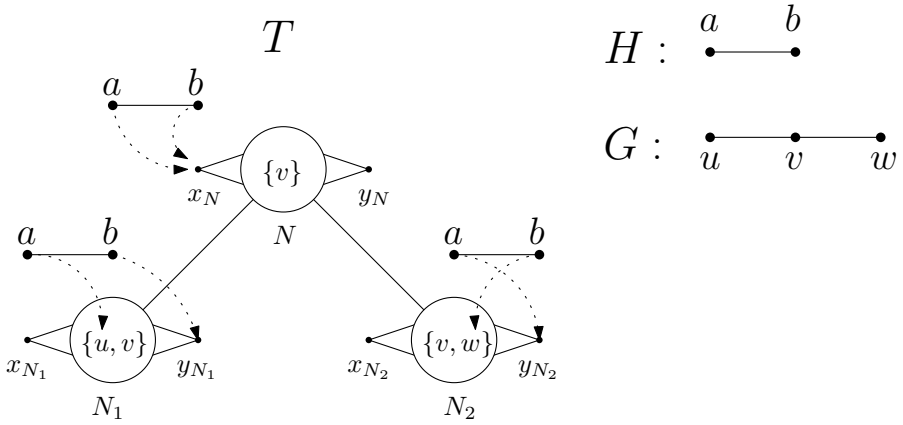


Figure A.1: A figure that illustrates the necessity of Condition (CB6) for consistent boundaries.

Condition (CB6) for consistent boundaries. The definitions of consistent boundaries (and compatible triples) in Section 5.1 include the Condition (CB6), which is missing in [Epp99]. This condition says that if two extended partial isomorph boundaries $\mathcal{B} = (B, T)$

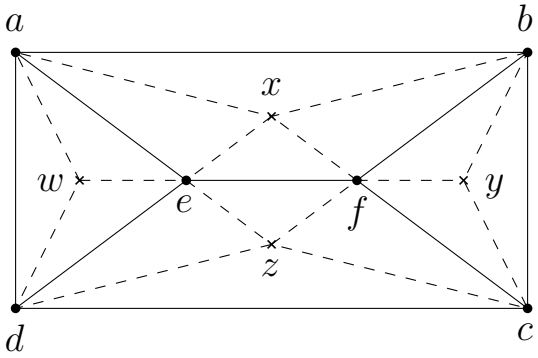
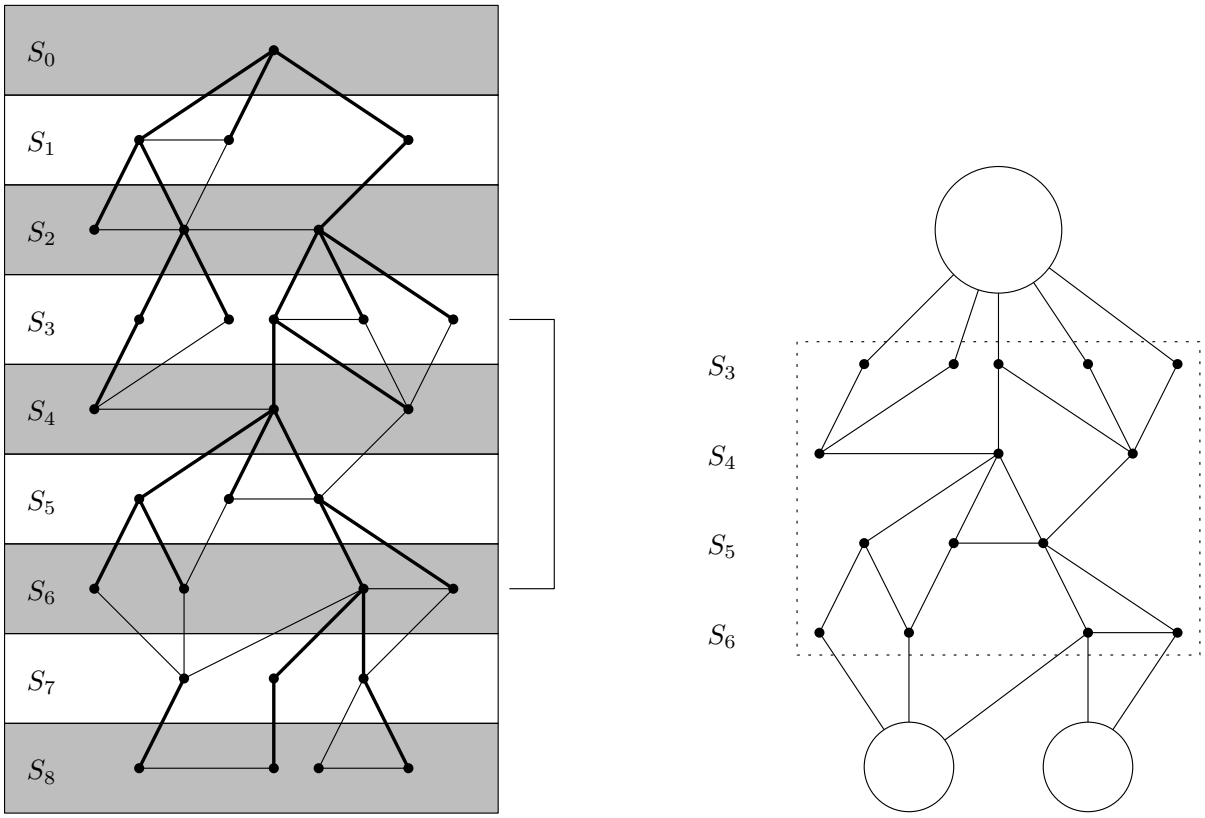


Figure A.2: A figure that illustrates that searching for separating cycles in $R(G)$ is not sufficient to find a separating set for G.

and $\mathcal{B}_1 = (B_1, T_1)$ at nodes N and N_1 are consistent, then for any edge (u, v) of H whose endpoints map to B_N^x, we must either have $B(u), B(v) \in L(N)$ or $B_1(u), B_1(v) \in \{x_{N_1}\} \cup L(N_1)$. This condition is essential to generate partial isomorphs correctly. Consider Figure A.1 for an example. (The example omits separator functions and only considers partial isomorph boundaries.) The host graph and pattern graph are represented by G and H respectively. The tree T is the tree decomposition of G. The three partial isomorph boundaries B, B_1 and B_2 at the nodes N, N_1 and N_2 are depicted by arcs from H to the the three nodes. If Condition (CB6) is omitted, then B, B_1 and B_2 form a compatible triple. Therefore, the three boundaries generate a partial isomorph that maps a to u and b to w. This is absurd because G has no edge with endpoints u and w.

Host graph as $\Lambda(G)$ instead of $R(G)$. The algorithm for finding separating cycles in [Epp99] uses $R(G)$ as the host graph instead of $\Lambda(G)$. This is problematic since a cycle X that separates two vertices u, v of G in $R(G)$ does not guarantee that $V_G(X)$ separates u, v in G. See Figure A.2 for an example. The solid edges belong to G and the dashed edges belong to $R(G)$. In this example, the cycle $(a, x, b, y, c, z, d, w, a)$ separates e, f in $R(G)$, but $\{a, b, c, d\}$ is not a separating set of G. The approach of having only $R(G)$ as the host graph succeeds if one can find a separating cycle in $R(G)$ that separates two vertices of G such that one vertex is inside the cycle and the other outside the cycle. (Recall from Chapter 4 that such a cycle is called a *fence*.) In fact, Lemma 8 of [Epp99] uses this approach and assumes that one can find a fence of $R(G)$ by simple modifications to the dynamic programming algorithm. However, it is not clear how one can modify the algorithm that outputs only those separating cycles that form a fence in $R(G)$.



(a) BFS tree of G and the layers S_i

(b) Construction of the graph Λ_3

Figure A.3: Construction of the graphs Λ_i

The graphs L_i and U_i. In Section 5.3, we computed the graphs L_i and U_i for each layer S_i. The main objective of constructing these graphs was to ensure that a cycle is separating in $\Lambda(G)$ if and only if it is separating in $\Lambda_i = U_{i-1} \cup \Lambda[S_{i-1} \cup \cdots \cup S_{i+w}] \cup L_{i+w}$ for some $i = 0, \ldots, p$. The construction of these graphs is a novel idea and does not appear in [Epp99]. The approach taken in [Epp99] to compute Λ_i (the graph is actually called G_i in [Epp99]) is to contract each component of $\Lambda(G)$ in the layers $S_t, t \leq i-1$ and $S_t, t \geq i+w$ into a supervertex. But it is not immediate how one could compute the contractions for all Λ_i in $O(n)$ time. The paper [HIK+17] by Holm et al. gives a data structure that can maintain a planar graph under edge contractions in $O(n)$ time. However, to compute $\Lambda_1, \ldots, \Lambda_p$ in overall $O(wn)$ time, we cannot afford to do contractions separately for $i = 1, \ldots, p$. Instead, we need to compute Λ_i from Λ_{i+1} (or vice-versa) by contracting some edges and uncontracting others. Even with the powerful data structure

by Holm et al., it is not clear how the uncontractions can be done. Hence it is not clear how to compute $\Lambda_1, \ldots, \Lambda_p$ in linear time via contractions.

In our algorithm, we also had to maintain the labels ϑ for vertices of Λ_i as an indicator that certain vertices of $\Lambda(G)$, which may well be face vertices and crossing points, count as vertices of G. This was necessary since components of L_{i+w} can have vertices of G contracted into it and a cycle that separates a vertex of L_{i+w} from another vertex of G in fact separates two vertices of G. This aspect of the algorithm is new and was not part of [Epp99].