DYNAMIC COMPACT PLANAR EMBEDDINGS

by

Michael St. Denis

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2023

Dalhousie University is located in Mi'ma'ki, the
ancestral and unceded territory of the Mi'kmaq.
We are all Treaty people.

Contents

List of Figures

Abstract

This thesis presents a way to compactly represent dynamic connected planar embeddings, which may contain self loops and multi-edges, in $4m + o(m)$ bits, to support basic navigation in $O(\lg n)$ time and edge and vertex insertion and deletion in $O(\lg^{1+\epsilon} n)$ time, where n and m are respectively the number of vertices and edges currently in the graph and ϵ is an arbitrary positive constant. Previous works on dynamic succinct planar graphs either do not provide a full set of update operations or are restricted to triangulations where the outer face must be a simple polygon and all inner faces must be triangles. To the best of our knowledge, this thesis presents the first representation of dynamic compact connected planar embeddings that supports a full set of dynamic operations without restrictions on the sizes or shapes of the faces.

Acknowledgements

I would like to express my deep appreciation and sincere gratitude to Dr. Meng He, my supervisor, for all the technical and non-technical guidance and assistance. This work would not have been possible if not for Dr. He's suggestions, deep questions, and thoroughness. I also want to thank Dr. He for his mentorship and willingness to help. An excellent listener with a great sense of humor, I genuinely enjoy our meetings and discussions. Thank you, Dr. He.

I would also like to sincerely thank Dr. Travis Gagie for reading and providing comments that improved this work, and especially for his insight and suggestions in helping to derive the solution.

Thank you Dr. Nauzer Kalyaniwalla for sharing insightful comments that improved this thesis.

I would like to thank my wife for her love and continuous support.

Lastly, I would be remiss without mentioning and thanking my community. Thank you to my in-law family, for their support, humor, and encouragement, and thank you to my friends in Halifax and abroad. Each of you played a role in helping this thesis come together. Thank you.

Chapter 1

Introduction

A particular type of graph, a planar graph, may be used to model the famous initial graph problem known as the seven bridges of Königsberg [24]. Aside from this application, planar graphs are also applicable to some maps in general, VLSI circuits [13], chemical molecules [1], and spatial partitions in geographical information systems (GIS) [13].

A more contemporary problem concerns the dramatic growth of problem sizes with respect to the growth in computer memory [8, 18]. Although computer memories are growing, and our ability to store data in secondary or even tertiary storage is still sufficient, being able to process this data in main memory is becoming more cumbersome. Secondary to this concern is the size of the data structure built on the data which is used to perform queries and updates. These data structures often occupy much more space than the data itself. Hence, Jacobson proposed to study succinct data structures [14].

This thesis exists at the intersection of graph theory and compact data structures, examining a way to represent a connected planar graph embedding using compact data structures. Much research has been conducted on static planar graphs where $O(n)$ bits are used to support common navigation operations such as adjacency testing and efficiently listing a vertices' neighbors [4, 5, 8, 14, 16, 21].

Static data structures offer easy access to the objects they store. However, objects either cannot be added or deleted from the structure or doing so would require rebuilding segments or the entire structure itself. None of the prior works cited support the insertion or deletion of an edge or a vertex. Prior work on dynamic succinct planar graphs is restricted to triangulations where the outer face must be a simple

polygon and all inner faces must be triangles [2] or do not provide a full set of dynamic operations [15]. This thesis presents a way to dynamize a compact representation of a connected planar embedding.

The operations we aim to support include the following:

- Given a vertex v, or a directed edge (v, u), list the edges incident to v in clockwise or counterclockwise order, starting from (v, u) when given.
- Given an edge (v, u) and a face F that (v, u) is adjacent to, list the edges incident to F in clockwise or counterclockwise order starting from (v, u).
- Given a face and two of its vertices, insert an edge connecting these vertices across the face.
- Delete a given edge from G so long as G remains connected.
- Given a vertex v and a face that v is a vertex of, insert a degree-1 vertex u in the face such that v is adjacent to u.
- Given a degree-1 vertex v, delete v and the edge e it is adjacent to from G so long as G remains connected after deleting v and e.

These operations allow for the transformation from one connected planar embedding to another connected planar embedding.

1.1 Our Contribution

Our contribution is summarized by the following theorem:

Theorem 1. *Given a connected planar embedding G, possibly containing multi-edges and self loops, on n vertices and m edges, there is a compact representation of G occupying $4m + o(m)$ bits that can list the edges incident to a given vertex in clockwise or counterclockwise order in $O(\lg n)$ time per edge, list the edges incident to a face in $O(\lg n)$ time per edge, and supports insertion or deletion of an edge or a vertex in $O(\lg^{1+\epsilon} n)$ time for any constant $\epsilon > 0$.*

To the best of our knowledge, this thesis presents the first dynamic compact connected planar embedding that supports fast insertion and deletion of an edge or a vertex and has no restrictions on the sizes or shapes of the faces. Additionally, we present the marker model to support basic navigation operations within a given connected planar embedding. This model is similar to the finger-update model [7] where a finger, or marker, is maintained on a given vertex and updates to the structure are limited to the position of the finger, or marker. The difference between the marker model and the finger-update model is that a marker has an indicator that points to a specific face in the given planar embedding.

1.2 Roadmap

Chapter 2 reviews related work and Capter 3 discusses the preliminaries. The data structures used in our representation of planar embedding G and the model we derive to support navigation and update operations are discussed in Chapter 4. Chapter 5 reviews how we support dynamic updates. Conclusions and future work is discussed in Chapter 6.

Chapter 2

Related Work

We survey previous results on succinct representations of planar embeddings [3, 4, 8, 14, 16, 21, 22]. Succinct planar graph representations that cannot encode an arbitrary embedding are not included; see [4] for a survey including those results. Tutte [22] enumerated rooted planar maps and his results implied that $m \lg 12 = 3.58m$ bits are required to encode an m-edge planar embedding. Turán [21] derived a simple succinct encoding of planar graphs that uses $4m$ bits. Keeler and Westbrook [16] then showed an encoding of planar maps that achieves Tutte's lower space bound, but no operations are supported. Additionally, their encoding and decoding algorithms run in linear time.

Later, researchers designed succinct data structures for planar embeddings, and the operations supported by these structures include listing the neighbors of a given vertex in counterclockwise or clockwise order. Barbay et. al. [3] showed how to represent a simple planar embedding in $18n + o(m)$ bits to support degree and adjacency queries in constant time and the listing of neighbors in constant time per edge. Blelloch and Farzan [4] developed a data structure that occupies $3.58m + o(m)$ bits and provide the same support for queries. Ferres et. al. [8] modified Turán's [21] structure to occupy an additional $o(m)$ bits and showed the following: the edges incident to vertex v can be listed in clockwise or counterclockwise order in constant time per edge, the edges limiting a face can be traversed in constant time per edge, a vertex's degree can be found in $O(f(m))$ for a given function $f(m) \in \omega(1)$ and computing if two given vertices are neighbors in $O(f(m))$ time for any given function $f(m) \in \omega(\lg m)$. This data structure is simpler than previous solutions and can be constructed in parallel efficiently. Fuentes-Sepúlveda et. al. [9] further show that by using *half-edges* and condensing Ferres et. al.'s [8] data structures, a full set of topological queries can be supported in efficient time.

All the above mentioned works concern succinct representations of planar graphs in the static case. With respect to the dynamic case, Aleardi et. al. [2] show a succinct representation of an n-vertex triangulated graph with fixed genus g and a simple polygon boundary that supports standard navigation in $O(1)$ time, vertex addition in amortized $O(1)$ time without supporting access to satellite data associated with each vertex, amortized $O(\lg n)$ time with data access, and vertex deletion or edge flip[1] in amortized $O(\lg^2 n)$ time. This data structure occupies $2.17m + o(m)$ bits and uses an additional $O(g \lg m)$ bits for representing triangulations on genus g surfaces. Kammer and Maintrup [15] provide a dynamic data structure, a modification of Blelloch and Farzan's [4], that supports edge contractions and vertex deletions in $O(n)$ time and $\mathcal{H}(n) + o(n)$ bits, where $\mathcal{H}(n)$ is the entropy of encoding an n vertex planar graph. Edge or vertex insertions are not supported, so their solution is for the decremental setting only.

[1]Edge flipping refers to removing the edge e and replacing it with the other diagonal of Q, where Q is the union of two triangles which create a quadrilateral.

Chapter 3

Preliminaries

We assume the word-RAM model of computation where any given input value U can be represented in a machine word of $O(\lg U)$ bits. In addition, we assume standard arithmetic and boolean bitwise operations are performed in constant time [6].

3.1 Notation

For the remainder of this thesis, let G be a given connected planar embedding on n vertices, m edges and f faces that may contain multi-edges and self loops. Let the notation $|H|$ denote the number of elements stored in data structure H. All logarithms are written as lg and are base 2, unless otherwise specified.

3.2 Dynamic Bitvectors

A **dynamic bitvector** $B[1..n]$, supporting the following operations is a key compact data structure ($b \in \{0, 1\}$ in the following definitions):

- $\texttt{access}(B, i)$: return the bit in $B[i]$ for any i such that $1 \leq i \leq n$.
- $\texttt{rank}_b(B, i)$: return the number of occurrences of b in $B[1..i]$ where $1 \leq i \leq n$.
- $\texttt{select}_b(B, j)$: return the index of the jth occurrence of b in B.
- $\texttt{insert}(B, i, b)$: inserts bit b between $B[i-1]$ and $B[i]$ where $1 \leq i \leq n$.
- $\texttt{delete}(B, i)$: deletes the bit in position $B[i]$ where $1 \leq i \leq n$.
- $\texttt{link}(B_1, p, B_2)$: attaches all the bits in B_2 between bits $B_1[p]$ and $B_1[p+1]$ where $1 \leq p \leq |B_1|$ and $|B_1|$ and $|B_2|$ are each at most n.
- $\texttt{cut}(B, i, j)$: returns bitvector B' which contains all the detached bits in B from $B[i]$ up to and including $B[j]$ where $1 \leq i < j \leq n$. B is then the concatenation

of the bit ranges $[1..(i-1)]$ and $[(j+1)..n]$ and B' holds the bits in the range $[i..j]$.

- `flip`(B, i): modifies the bit in $B[i]$ to be a 1 if $B[i] = 0$ or to be a 0 if $B[i] = 1$ where $1 \leq i \leq n$.

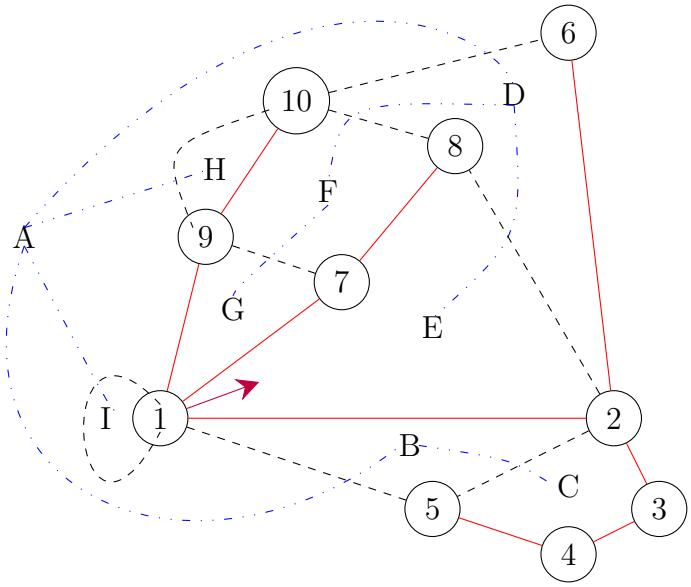
Navarro and Sadakane [19] present the following:

Lemma 1. ([19]). *There exists a succinct dynamic bitvector structure that supports access, rank, select, insert, delete, and flip in* $O(\frac{\lg n}{\lg \lg n})$ *time and supports link and cut in* $O(\lg^{1+\epsilon} n)$ *time, where* n *is the number of bits currently in the bitvector and* ϵ *is an arbitrary positive constant.*

3.3 Planar Graph Traversal

The traversal of a planar embedding G developed by Turán [21], which is used by Ferres et. al. [8] to generate a binary sequence A, is now described. From here forward, we will refer to this traversal as a *Turán traversal*. An arbitrary spanning tree T, which is rooted at some vertex v_0 on the outer face of G, is computed before the traversal. We note that a Turán traversal can be performed in counterclockwise or clockwise order. Without the loss of generality, we use counterclockwise order and design our data structures with respect to this ordering. We call an edge in G that is also in T a *primal edge* and an edge in G not in T a *dual edge*. In this Turán traversal, after we visit a vertex v and traverse or examine an edge, (v, u), incident to it, we view (v, u) as a directed edge by orienting it from v to u, even though the graph G is undirected.

The traversal of G begins from the vertex selected as the root v_0 of T and examines one of its incident edges (v_0, u) such that (v_0, u) is on the boundary of the outer face and the outer face is to its right. The traversal is a modified depth first search (DFS) where we visit vertices in counterclockwise order. In a standard DFS, we examine an edge (v, u) and do not traverse from v to u if u has already been visited, unless we are returning to a parent. In the Turán traversal, we examine an edge (v, u), and if (v, u) is a primal edge, we traverse from v to u and record a 1 in a



0 1 0 1 1 1 0 0 1 1 1 1 0 1 0 **1** 1 1 0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 0

Figure 3.1: Planar embedding G where the red solid edges correspond to edges in T and blue dashed edges correspond to edges in T^*, the dual of G complementary to T. The violet arrow on the vertex labeled 1 and directed at the face labeled E depicts a marker (to be discussed in Section 4.2). Bitvector A contains a Turán traversal beginning with edge $(1, 5)$ and proceeding counterclockwise. The violet boldfaced 1 bit indicates the marker as represented by the violet arrow on vertex 1 which is marker 16 (to be discussed in Section 4.2).

bitvector A. Otherwise (v, u) is a dual edge, and we do not traverse it. Instead, we remain on v, record a 0 in A and examine the next edge in counterclockwise order. This process is repeated recursively until we have visited all vertices and returned to the root v_0 of T. All edges in G have been traversed or examined twice and $A[i]$ indicates whether the ith edge examined in the Turán traversal is a primal or dual edge.

Observe that a Turán traversal of G performs an Euler tour traversal of T and an Euler tour traversal of the spanning tree of the dual of G with respect to T, which we refer to as T^*. More specifically, to define T^*, consider some dual edge (v, u) in G not yet examined. Let f_r (f_l) be the face on the right (left) side of (v, u). Examining (v, u) advances the Euler tour of T^* from f_r to f_l, establishing f_r as the parent of f_l in T^*, and we refer to edge (u, v) as the *entry edge* of f_l. Thus, T^* encodes a spanning tree on the faces of G and each edge in G not in T is crossed by an edge in T^*. In this

way, every connected planar embedding can be represented as interdigitating spanning trees of the primal and dual trees [23]. As the Turán traversal in this thesis is performed in counterclockwise order, the traversal of T^* is performed clockwise from its root, i.e., the outer face. Figure 3.1 gives an example.

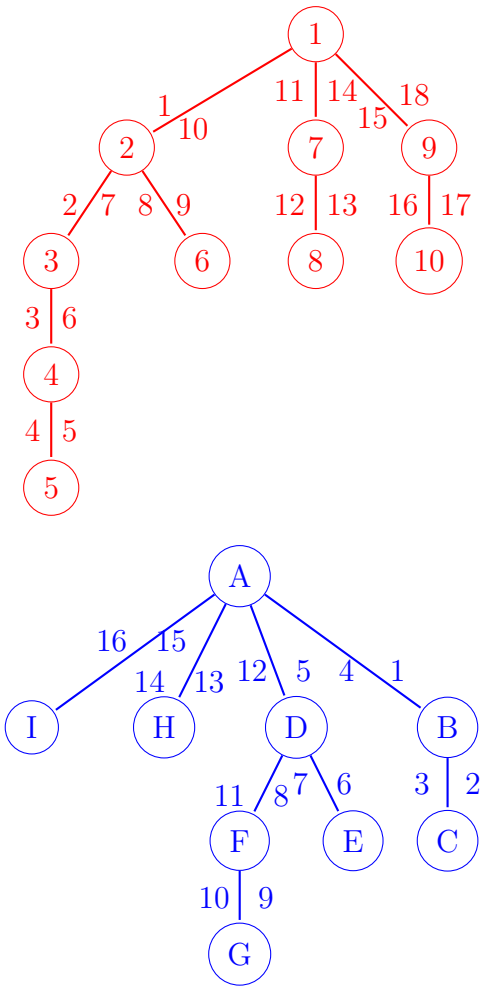


Figure 3.2: The spanning tree T, shown in red, on the vertices of G from Figure 3.1 and the spanning tree T^*, shown in blue, on the faces of the dual of G with respect to T from Figure 3.1.

3.4 Succinct Euler-Tour Trees

Gagie and Wild [10] describe how to succinctly represent a set of Euler-Tour trees of an n vertex forest in $2n + o(n)$ bits. An Euler-Tour tree is an unrooted tree

containing directed edges (u, v) and (v, u) for every undirected edge in the given forest and preserves an encoding of the order in which edges are visited in an Euler-Tour tree. Tarjan and Vishkin [20] showed how to represent trees as an Euler tour. Later, Henzinger and King [11] showed how to use an Euler tour representation of a tree to support queries and updates. A *corner* is defined as the space between consecutive edges incident to a vertex [12]. We use $\{a, b\}$ to refer to the corner of Euler-Tour tree T between the two edges traversed at the ath and bth steps of the Euler tour of T. In [10], the `merge` and `split` operations are implied, but we state them explicitly. Throughout this thesis, we refer to operations 3 and 4 in the lemma below as `vertex` and `inverse`, respectively. Gagie and Wild present the results summarized in this lemma:

Lemma 2. ([10]). *Given a planar embedding of a forest F on n vertices, F can be encoded in a data structure occupying $2n + o(n)$ bits such that operations 1 through 4 below take constant time, operations 5 and 6 take $O(\lg n)$ time, and operations 7 through 10 take $O(\lg^{1+\epsilon} n)$ time for some constant $\epsilon > 0$.*

1. *return the predecessor and successor in the Euler tour of the tree containing the given directed edge e.*

2. *return the predecessor and successor in the counterclockwise order of the edges incident to u when given directed edge (u, v).*

3. *return vertex v such that v is the vertex arrived at after traversing the kth edge in the Euler tour of T.*

4. *return the e'th edge encountered in the Euler tour traversal of T such that, given a T and the eth edge encountered in the Euler tour of T, the e'th edge encountered in the Euler tour of T corresponds to the inverse edge of e.*

5. *return the edge e' such that the distance from the given directed edge e to e' is the given distance t in the Euler tour of the tree containing e.*

6. *return the Euler tour distance between the given edges e and e' so long as the two edges are in the same tree.*

7. *delete the given edge e from the tree containing it and return the representations of the two resulting trees.*

8. *insert an edge between T and T' at the given corners, bisecting those corners, and return the representation of the resulting tree.*

9. *merge the adjacent vertices u and v to become one vertex and retain all other edges adjacent to u and v at the given edge e that connects u and v.*

10. *split v into two adjacent vertices, v_1 and v_2, where v_1 is a parent of v_2. Two incident boundary edges e_i and e_j are also given as parameters so that edges incident to v starting from e_i to e_j in counterclockwise order are to be incident to v_1, while the remaining edges are to be incident to v_2.*

Operation 8 supports the insertion of an edge between two arbitrary vertices in two trees. This update operation cannot be supported by the dynamic succinct tree representation of Navarro and Sadakane [19] which is based on a balanced parenthesis representation, but it is needed in our dynamic planar graph representation. Thus, we use the result of Gagie and Wild [10] in this thesis.

Chapter 4

Data Structure and Marker Model

4.1 Data Structures

Our representation of connected planar embedding G on n vertices, m edges, and f faces, contains the following components:

- A dynamic bitvector, A, which encodes a Turán traversal of G described in Section 3.3. It is represented by Lemma 1 in $2m + o(m)$ bits.
- A spanning tree, T, of G as defined in Section 3.3. It is represented by Lemma 2 in $2n + o(n)$ bits.
- A spanning tree, T^*, of the dual of G as defined in Section 3.3. It is represented by Lemma 2, in $2f + o(f)$ bits.

Observe that T and T^* represent succinct Euler-Tour trees on the vertices and faces of G, respectively, so T requires $2n+o(n)$ bits of space and T^* requires $2f+o(f)$ bits of space. By Euler's formula [17], $n-m+f = 2$, the latter is $2(m-n+2)+o(m)$ bits. Thus the total space cost of our data structure is $2m+o(m)+2n+o(n)+2(m-n+2) = 4m + o(m)$.

4.2 The Marker Model

The marker model provides a way to map an index in A to specific vertices in T and T^*. A marker, or a marker's value, is denoted by a single integer value i, where $1 \leq i \leq 2m$ and is some index in A. Recall that a Turán traversal of G induces an Euler tour traversal on T and T^*. We define the *orientation* of a marker as the vertex most recently visited in the Euler tour of T and the face most recently visited in the Euler tour of T^*. The number of 1's (0's) in A corresponds to the primal (dual) Euler-Tour tree edges just crossed in an Euler tour of T (T^*). Therefore, a marker with

value i stands on the vertex of G that corresponds to node $\texttt{vertex}(T, \texttt{rank}_1(A, i))$ in T, and the face it points to corresponds to node $\texttt{vertex}(T^*, \texttt{rank}_0(A, i))$ in T^*. Figure 3.1 shows marker 16 as an example.

If more than one marker is maintained at a given time then, after an update, all markers must be updated to preserve their orientation. The index a marker refers to can be updated in $O(1)$ time via a constant number of comparisons and arithmetic operations due to the way we support updates. Thus, the run time to preserve the orientation of all markers after an update is linear in the number of markers maintained.

In the lemma below, we consider the orientation of a marker i in G where $A[i]$ corresponds to some directed edge (v, u).

Lemma 3. *Let (v, u) be a directed edge currently enumerated in a Turán traversal. If (v, u) is a primal edge, then the marker is standing on u and pointing to the face on the right side of (v, u). If (v, u) is a dual edge, then the marker is standing on v and pointing to the face on the left side of (v, u).*

Proof. We prove this by induction on i, where $A[i]$ represents the ith enumerated edge in the Turán traversal of G. For the base case $i = 1$, which represents the first edge (v, u) enumerated in the Turán traversal. There are two cases, depending on whether (v, u) is a primal or a dual edge.

If (v, u) is a primal edge, then the Turán traversal follows this edge to visit u, making the marker stand on u. Since this does not advance the traversal of T^*, the marker points at the outer face, which is to the right of (v, u).

If (v, u) is a dual edge, then the Turán traversal does not follow (v, u) but rather continues to stand on v. Since (v, u) is a dual edge, we advance in the traversal of T^* by going from the face on one side of (v, u) to the face on the other side. When we start the Turán traversal, the face of G we visit was to the right of (v, u), so the marker now points to the face to the left of (v, u).

We now assume this lemma is true after the ith step of the Turán traversal, and we prove that it still holds after the $(i+1)$st step. In our proof, let (v, u) be the edge examined in the ith step, and let F be the face that the marker points to after this step. Then there are two cases depending on whether (v, u) is primal or dual.

In case 1, (v, u) is a primal edge. By the induction hypothesis, F is to the right of (v, u). Furthermore, after the ith step, the marker stands on u after traversing (v, u), and thus the next edge to examine is an edge that connects u to another vertex, w. As F is to the right of (v, u) it is to the left of (u, v). Since (u, w) is the next edge in counterclockwise order from (u, v), F is on the right side of (u, w). There are two subcases, depending on whether (u, w) is primal or dual.

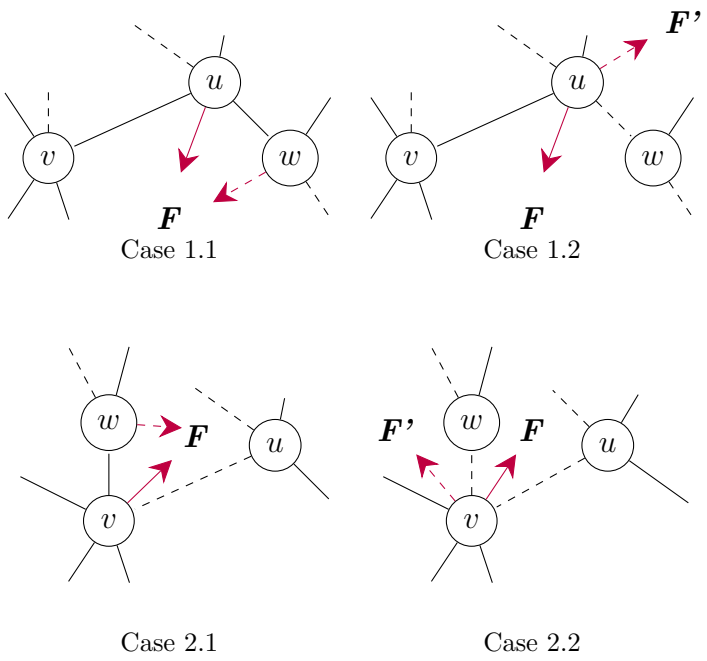


Figure 4.1: Inductive cases, showing how the marker updates from the solid arrow to the dashed arrow, when processing the next edge in counterclockwise order following edge (v, u) where F is the initial face the marker is pointing to and F' is the second face the marker points to, if the update changes the face the marker is pointing to.

In subcase 1.1, (u, w) is a primal edge. When the degree of u is 1, we have $w = v$, and our reasoning below applies to both this special case and the general case. As (u, w) is a primal edge, at the $(i + 1)$st step, (u, w) is traversed which relocates the

vertex the marker is standing on from u to w and advances one edge in the Euler tour of T. However, this does not advance the traversal in T^*, so the marker still points to F, which is on the right side of (u, w). This can be viewed in Figure 4.1(a).

In subcase 1.2, (u, w) is a dual edge. As (u, w) is a dual edge, at the $(i+1)$st step, the Turán traversal does not traverse (u, w), so the marker continues to stand on u. What the traversal does instead is to continue in the Euler tour of T^* by going from F to the face, F', on the other side of (u, w), making the marker point to F'. Since F is on the right side of (u, w), F' is on the left side of (u, w), and the induction also goes through in this case. Figure 4.1(b) shows a depiction.

In case 2, (v, u) is a dual edge. By the induction hypothesis, F is to the left of (v, u). Furthermore, after examining (v, u) in the ith step, the marker continues to stand on v, and thus the next edge to examine is an edge that connects v to another vertex, w. As F is on the left side of (v, u) and the next edge in counterclockwise order incident to v is (v, w), F is on the right side of (v, w). There are two subcases, depending on whether (v, w) is primal or dual.

In subcase 2.1, (v, w) is a primal edge. As (v, w) is a primal edge, at the $(i+1)$st step, (v, w) is traversed which relocates the vertex the marker is standing on from v to w and advances one edge in the Euler tour of T. However, this does not advance the traversal of T^*, so the marker continues to point to F, which is on the right side of (v, w). An example of this can be viewed in Figure 4.1(c).

In subcase 2.2, (v, w) is a dual edge. As (v, w) is a dual edge, at the $(i+1)$st step, the Turán traversal does not traverse (v, w), so the marker continues to stand on v. What the traversal does instead is to continue in the Euler tour of T^* by going from F to the face, F', on the other side of (v, w), making the marker point to F'. Since F is on the right of (v, w), F' is on the left side of (v, w), and the induction again goes through in this case. Figure 4.1(d) shows an example. □

Because of the presence of sticks, a vertex can be incident to multiple corners of the same face. This means multiple markers can stand on the same vertex and point

to the same face. For simplicity, in the remainder of the thesis, we do not consider this case except in the discussions related to vertex insertion and deletion. This case can be addressed easily by refining our definition of an orientation. Let marker i refer to edge (v, u). If (v, u) is a primal edge, then marker i stands on u and points to the face on the right of (v, u) with u as the apex. If (v, u) is a dual edge, then marker i stands on v and points to the face to the left of (v, u) with v as the apex. Then we can modify our discussions trivially to address this case.

4.3 Navigation

Now we define two rotation operations and a traverse operation. Either rotate operation changes the face the marker is pointing to and the traverse operation changes the vertex the marker is standing on. These operations are necessary to move the marker to support queries and updates on our representation of G. For the operations described below, let v be the vertex marker i currently stands on and F be the face marker i currently points to; they will also be referred to when we discuss how to support these operations.

- `rotate_ccw`(i): Compute a new orientation of the marker such that the marker is still standing on v but is pointing to the face next to F when listing all faces incident to v in counterclockwise order.

- `rotate_cw`(i): Compute a new orientation of the marker such that the marker is still standing on v but is pointing to the face next to F when listing all faces incident to v in clockwise order.

- `traverse`(i): Let the edge (v, w) be the $(i + 1)$st edge examined in the Turán traversal. Compute a new orientation of the marker such that the marker is now standing on w but still pointing to F.

4.3.1 Rotating Counterclockwise

We now describe how to support `rotate_ccw`(i). To do this, first consider the ith edge examined in a Turán traversal. One of its endpoints is v, let u be the other

endpoint, and let (v, w) be the next edge after (v, u) in counterclockwise order. If (v, u) is a primal edge, then, by Lemma 3, it is oriented from u to v in the ith step of the traversal. Furthermore, face F is to the right of (u, v) and is thus to the left of (v, u). If this edge is a dual edge, then by Lemma 3, it is oriented from v to u, and face F is again to the left of (v, u). In either case, since (v, w) is the edge next to (v, u) in counterclockwise order, F is to the right of (v, w). Therefore, our goal is to compute a marker that is still standing on v but pointing to the face, F', to the left of (v, w), since F' is the face next to F when listing all faces incident to v in counterclockwise order.

There are now two cases, depending on whether the $(i+1)$st edge, (v, w), enumerated in a Turán traversal, is a dual or primal edge, i.e. whether $A[i + 1]$ is 0 or 1. If $A[i+1] = 0$, then by Lemma 3, marker $i+1$ continues to stand on v but point to F'. Therefore we return $i + 1$ as the answer. Otherwise, $A[i + 1] = 1$ and the answer is computed as the index in A when the Turán traversal returns to v from edge (w, v). This is computed by $j =$ $\texttt{select}_1(A, \texttt{inverse}(T, \texttt{rank}_1(A, i + 1)))$. This follows from Lemma 3; since (w, v) is a primal edge, the marker j is standing on v and pointing to the face to the right of (w, v). As the right side of (w, v) is the left side of (v, w), the marker computed is also pointing to F' in this case.

Let us refer to the planar embedding and the bitvector in Figure 3.1 for an example. We arbitrarily choose the marker that stands on vertex 2 and points to face B. Thus, $i = 2$. Let us rotate counterclockwise about vertex 2. First, we examine the bit in $\texttt{access}(A, i + 1)$ and observe that $\texttt{access}(A, i + 1) = 0$, meaning that the edge to be rotated over is a dual edge. Therefore, we update i by performing the operation corresponding to the case that $i + 1$ is a dual edge: $i = \texttt{mod}(i + 1, 2m) = 3$. This is correct as the third edge examined in a Turán traversal is $(2, 5)$ and because it is a dual edge, by Lemma 3, the marker $i = 3$ stands on vertex 2 and points to face C.

To observe counterclockwise rotations over a primal edge, we will perform another counterclockwise rotation about vertex 2 when the marker is pointing to face C. As noted above, with the marker standing on vertex 2 and pointing to face C, $i = 3$.

We first inspect $\texttt{access}(A, i+1)$ and see that $\texttt{access}(A, i+1) = 1$. Thus, the edge we intend to rotate over is a primal edge. Applying the operation for the case that $\texttt{access}(A, i+1) = 1$ results in computing an updated marker value of:

$$\begin{aligned} i &= select_1(A, inverse(, T, rank_1(A, 3+1))) \\ &= select_1(A, inverse(T, 2)) \\ &= select_1(A, 7) \\ &= 11 \end{aligned}$$

We confirm this is correct by observing that the 11th edge enumerated in a Turán traversal corresponds to the marker standing on vertex 2 and pointing to face A.

4.3.2 Rotating Clockwise

We now describe how to support $\texttt{rotate\_cw}(i)$. Consider the ith edge examined in a Turán traversal. One of its endpoints is v and let u be the other endpoint. There are two cases, depending on whether the ith edge enumerated in a Turán traversal is a dual edge or a primal edge, i.e. whether $A[i]$ is 0 or 1. If $A[i] = 0$, then the ith edge is a dual edge oriented from v to u. By Lemma 3, face F is to the left of (v, u), and the other face, F', that (v, u) is incident to is to its right, so F' is the face next to F when listing all faces incident to v in clockwise order. Since, by examining (u, v) in the ith step, the Turán traversal advances the Euler tour of T^* from F' to F without changing the vertex that the marker stands on, marker $i-1$ stands on v and points to F'. Therefore, we return $i-1$ as the answer. Otherwise, $A[i] = 1$, and the ith edge is a primal edge oriented from u to v. By Lemma 3, face F is to the right of (u, v). Step $j = \texttt{select}_1(A, \texttt{inverse}(T, \texttt{rank}_1(A, i)))$ traverses this edge in reverse order, i.e., from v to u, and by Lemma 3, marker j points to F' which is to the right of (v, u). Since traversing (v, u) in the jth step makes the marker stand on u without changing the face it points to, marker $j-1$ stands on v and points to F', so it is the answer we return.

Suppose our marker in G in Figure 3.1 is standing on vertex 2 and pointing to face E. This orientation corresponds to $i = 15$. We arrive at $i = 15$ by computing that $\texttt{rank}_1(A, 15)$ corresponds to having traversed the 9th edge in the Euler tour of

T and that $\texttt{rank}_0(A, 15)$ corresponds to having traversed the 6th edge in the Euler tour of T^*. As the 9th edge in T points to vertex 2 and the 6th edge in T^* points to the vertex E, $i = 15$ corresponds to the marker standing on vertex 2 and pointing to face E. Now suppose we wish to rotate clockwise about vertex 2. This single clockwise rotation should orient the marker to continue to be standing on vertex 2 but point to face D. This can be confirmed visually in Figure 3.1. We begin by examining the bit in $\texttt{access}(A, i)$ and we observe that $\texttt{access}(A, i) = 0$. Therefore, to compute the new marker value after a clockwise rotation about vertex 2, we compute the result of $i = i - 1 = 14$. As stated above, after this clockwise rotation, the marker should be standing on vertex 2 and pointing to face D. This is correct as the 14th edge examined in a Turán traversal is $(6, 2)$ which, by Lemma 3, corresponds to the marker standing on 2 and pointing to face to the right of $(6, 2)$, which is face D.

We can perform one more clockwise rotation about vertex 2 to observe a rotation over a primal edge. From the same orientation as discussed above, we perform one clockwise rotation. After doing so, we expect the orientation to be standing on vertex 2 and pointing to face A. Examining $\texttt{access}(A, i)$ where $i = 14$, as previously calculated, confirms that $\texttt{access}(A, 14) = 1$. We therefore perform the primal edge computation case. Specifically, we compute the following: $\texttt{select}_1(A, \texttt{inverse}(T, \texttt{rank}_1(A, 14)))-1 = \texttt{select}_1(A, \texttt{inverse}(T, 9))-1 = \texttt{select}_1(A, 8)-1 = 12-1 = 11$. This is also correct, as the 11th edge enumerated in a Turán traversal corresponds to $(3, 2)$ and, by Lemma 3, the marker is standing on vertex 2 and pointing to face A.

4.3.3 Traversing an Edge

Now we describe how to support $\texttt{traverse}(i)$. There are two cases to consider, depending on whether (v, w) is a primal or a dual edge. If $A[i + 1] = 1$, then the $(i+1)$st edge is a primal edge. In this case, the Turán traversal traverses (v, w) in the $(i+1)$st step, making marker $i+1$ stand on w without changing the face it points to. Therefore, we return $i + 1$ as our answer. Otherwise, $A[i + 1] = 0$, then the $(i + 1)$st edge, (v, w), is a dual edge. By similar reasoning to Section 4.3.1, F is to the right of (v, w). By Lemma 3, our goal is to compute the marker j that examines (v, w) in reverse order, i.e., from w to v, resulting in marker j standing on w and pointing to

the face to the left of (w, v), and thereby the face to the right of (v, w), which is F. Marker j is computed by $j =$`select`$_0(A,$ `inverse`$(T^*,$ `rank`$_0(A, i + 1)))$ and this is the answer we return.

We observe that, except for differences with respect to the bits in the `rank` and `select` queries used, the Euler-Tour trees T^* and T, and the cases when $A[i+1] = 0$ and $A[i + 1] = 1$, the `rotate_ccw`(i) and `traverse`(i) operations are symmetric.

In the worst case, at most two dynamic bitvector operations and one succinct Euler-Tour tree operation are computed in the navigation operations described above. Combining this with Lemmas 1 and 2, the run time of the operations described in this section is summarized in the following lemma:

Lemma 4. *The data structures in this section can support* `rotate_ccw`*,* `rotate_cw`*, and* `traverse` *in* $O(\lg n)$ *time.*

4.4 Listing Edges of a Vertex or a Face

Listing Edges Incident to a Vertex

Given a vertex v or a directed edge (v, u), we wish to list the edges incident to v in clockwise or counterclockwise order starting from (v, u) when given. Following the discussion from Section 4.2, it is easy to see that given v or (v, u), to list the edges in clockwise (counterclockwise) order from v, we record the index in A, associated with the marker's orientation, in some temporary value y and continuously call `rotate_cw` (`rotate_ccw`) and report the corresponding edges until the marker's orientation equals y again. By Lemma 4, this takes at most $O(\lg n)$ time per edge.

Listing Edges Incident to a Face

Given some directed edge (v, u) and face F, where (v, u) is incident to F, we wish to list all the edges in G incident to F in clockwise or counterclockwise order. To do this, we must first ensure the marker is pointing to F.

When given (v, u), we check if (v, u) is in T or T^*. Without the loss of generality, suppose $(v, u) \in T$. The distance *dist* from the root r_T to (v, u), via Lemma 2, is computed. Let $i = \texttt{select}_0(A, dist)$. The face the marker i is currently pointing to, F', is computed by: $\texttt{vertex}(T^*, \texttt{rank}_0(A, i))$. If F' is not F, then we use `rotate_ccw` or `rotate_cw` to compute a marker that stands on u and points to F, and we update i by the index of this marker. In either case, by reasoning similar to Section 4.3.1, F is to the right of the edge (u, w), examined in step $i + 1$ of the Turán traversal, so (u, w) is the edge after (v, u) in clockwise order that is incident to F.

This implies the following algorithm: To list edges incident to F in clockwise order, we store i, associated with the marker's orientation, in a temporary value y. The edge corresponding to $A[i]$ is reported, we continuously call $\texttt{traverse}(i)$, report the subsequent edges, and immediately stop once $i = y$.

To list edges incident to F in counterclockwise order, we perform the above algorithm to report edges in clockwise order and, instead of immediately reporting the edges, we temporarily store them. Then, we report the stored edges in reverse order. By Lemma 4, listing edges incident to a face in clockwise or counterclockwise order takes at most $O(\lg n)$ time per edge.

Chapter 5

Dynamization

We now examine insertions and deletions of edges and vertices and their effect on our representation of G. With respect to inserting and deleting vertices and edges, we consider how to update A, T, and T^* in each dynamic operation. Edge insertions and deletions are discussed in Section 5.1. For vertex insertion and deletion, we support the insertion and deletion of a degree-1 vertex that is connected to some other vertex in G, which we refer to as a *stick*. This is discussed in Section 5.2.

5.1 Inserting and Deleting Edges

We now describe insertion and deletion of edges in our representation of G. Recall from Section 3.3 that a Turán traversal traverses primal edges, does not traverse dual edges, and encodes this primal edge and dual edge sequence in a bitvector, A. To minimize the changes to the Turán traversal of G, all new edges will be made a dual edge upon insertion into G. It is possible that this inserted dual edge may be promoted to a primal edge in the future due to the deletion of a primal edge.

5.1.1 Inserting an Edge

We now discuss edge insertion. To insert an edge, we need two markers, i and j, and a face, F, such that the edges examined in the ith and jth step of the Turán traversal are two of the edges that delimit F. This specifies the face to draw the edge across. Let v be the vertex marker i stands on and let u be the vertex marker j stands on. These vertices, v and u, are the endpoints of the edge to be inserted. Since a single `rotate_ccw` or `rotate_cw` operation can make a marker point to the face on the other side of the edge that the marker corresponds to, we assume that both marker i and marker j point to F for simplicity. We also assume, without the loss of generality, that $i < j$.

The $\texttt{rank}_0$ query on A, with parameters i and j, computes the Euler tour edges in T^* just processed at the ith and jth step in the Turán traversal. We denote these Euler tour edges as i' and j'. Recall that v is one endpoint of the ith edge examined in a Turán traversal of G, and let w be the other endpoint. By Lemma 3, and similar to the reasoning from Section 4.3.1, no matter if this edge is a dual edge or a primal edge, F is to the left of (v, w). If we rotate counterclockwise from (v, w), with v as the pivot, F is the first face encountered and is encountered before any other edge incident to v. Therefore, when inserting an edge across F, the new edge, (v, u), is the next edge examined in a Turán traversal. This means that (v, u) will be inserted as the dual edge examined in the $(i+1)$st step of the Turán traversal. Thus, we perform $\texttt{insert}(A, i + 1, 0)$. Due to the insertion of a new bit, we also increment j, so that marker j corresponds to the same edge after insertion. Then, by similar reasoning, we additionally perform $\texttt{insert}(A, j + 1, 0)$ to indicate that (u, v) is the dual edge examined in the $(j+1)$st step. To update T^*, we observe that drawing an edge across F splits the face. Therefore, we perform $\texttt{split}(\texttt{vertex}(T^*, i'), i', j')$. As inserting a dual edge only affects the faces and not the vertices, T is unaffected. Lastly, we increment m by 1. Figure 5.1 shows an example.

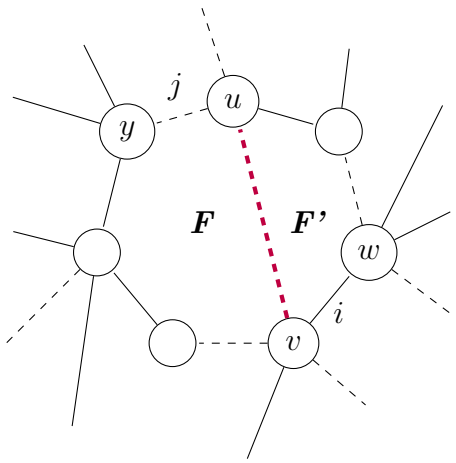


Figure 5.1: Splitting face F into two faces, F and F', when inserting a new edge, (v, u), the violet edge, between marker i and marker j, where marker i corresponds to primal edge (v, w) and marker j corresponds to dual edge (u, y).

Lemma 5. *Given two vertices u and v and an adjacent face F, an edge connecting u and v across F can be inserted in G in $O(\lg^{1+\epsilon} n)$ time.*

5.1.2 Deleting an Edge

We now discuss edge deletion. We perform an edge deletion only if, after removing the edge, G remains connected. Deleting dual edges does not disconnect G, as the primal edges form the spanning tree T, connecting all vertices of G. However, G could become disconnected when deleting primal edges. Therefore, we allow primal edge deletions only if the deletion of that primal edge does not result in G becoming disconnected. First we discuss dual edge deletion in the subsection immediately below, followed by our discussion of primal edge deletion.

Deleting a Dual Edge

Now, we discuss dual edge deletion. We delete the dual edge associated with marker i, which corresponds to the ith edge, (v, u), enumerated in a Turán traversal. Let F be the face that marker i points to. By Lemma 3, since (v, u) is a dual edge, F is to the left of (v, u). Let F' be the face to the right of (v, u). We compute the inverse of (v, u), i.e., (u, v), and let j be the index in A that corresponds to (u, v). Then marker j corresponds to the marker standing on u and pointing to F'. Additionally, we also assume, without the loss of generality, that $i < j$.

By similar reasoning as Section 5.1.1, we compute the Euler tour edges in T^* just processed at the ith and jth step in the Turán traversal and store these values in i' and j' respectively. Our goal is to remove the edge examined at the ith and jth steps of a Turán traversal. As marker i points to F and marker j points to F', removing the edge (v, u) removes this edge that separates F and F'. Therefore, removing a dual edge corresponds to merging faces in G thus, T^* must merge the vertices associated with F and F'. To do this, we perform $\texttt{merge}(T^*, \texttt{vertex}(T^*, i'), \texttt{vertex}(T^*, j'))$. To delete the bits in A at indices i and j, we first perform $\texttt{delete}(A, i)$. Due to the deletion of a bit, we decrement j so that the marker corresponds to the same edge. We then perform $\texttt{delete}(A, j)$ and decrement m. As deleting a dual edge only affects a dual edge and the faces separated by the dual edge, T remains unaffected.

Deleting a Primal Edge

We now discuss primal edge deletion. Throughout this discussion, let edge (v, u) correspond to the edge enumerated at the ith step in a Turán traversal of G and be the primal edge we wish to delete. When deleting a primal edge from our representation of G, there are four items to consider. The first item to consider is how to determine if G would be disconnected after deleting (v, u). Second, if G will not be disconnected after deleting (v, u), how do we choose a dual edge to promote to primal? Third, recall from Section 3.3 that a Turán traversal follows primal edges, and thus, primal edge deletion changes the Turán traversal of G. We must then determine how we update A to reflect a valid Turán traversal after deleting (v, u). Lastly, the fourth item we must consider is, how are T and T^* affected by primal edge deletion.

To determine if G would be disconnected after deleting (v, u), we inspect the faces on either side of (v, u). If the faces are the same, then (v, u) cannot be deleted as (v, u) is linking two connected components in G. Without (v, u), these components become disconnected. We can delete (v, u) such that G remains connected only if the faces on either side of (v, u) are different.

Assuming the faces adjacent to (v, u) are different, we now discuss how to select a dual edge to promote to primal. Recall that the ith step in a Turán traversal corresponds to traversing (v, u) and let the jth step be the step traversing the same edge in the reverse direction, i.e. from u to v. We assume, without the loss of generality, that $i < j$. Observe that deleting a primal edge in G corresponds to deleting an edge in T and therefore disconnecting T into two trees. Our goal is to select a dual edge to promote to primal that connects the two trees in T. The following lemma will be useful when we select such an edge; when proving it, we define the interval of A corresponding to an edge of G (henceforth the interval of this edge for short) to be $[a, b]$ if this edge is examined in steps a and b of the Turán traversal with $a < b$, e.g., the interval of (v, u) is $[i, j]$.

Lemma 6. *Between the two faces incident to (v, u), at least one of them has the property that the interval of its entry edge does not enclose $[i, j]$.*

Proof. Let F_1 and F_2 be the two faces incident to (v, u). Let g_1 and g_2 be the entry edge of F_1 and F_2, respectively, and let k_1 and k_2 be the reverse of g_1 and g_2, respectively. For simplicity, we assume, without the loss of generality, that $g_1 < g_2$.

Assume to the contrary that both $[g_1, k_1]$ and $[g_2, k_2]$ enclose $[i, j]$. Then $[g_1, k_1]$ and $[g_2, k_2]$ must intersect. Furthermore, the endpoints of the interval of the entry edge of a face correspond to the first and the last time we visit the node of T^* representing this face in an Euler tour traversal of T^*. Therefore, $[g_2, k_2] \subset [g_1, k_1]$, and the node, f_2, of T^* representing F_2 is a descendant of the node, f_1, of T^* representing F_1. By the definition of an Euler tour, this means that between steps g_2 and k_2 of the Turán traversal, the induced Euler tour of T^* only visits nodes that are descendants of f_2 in T^*, including f_2 itself. Since f_1 is the parent of f_2, no marker between g_2 and k_2 can point to face F_1. However, either marker i or marker j points to F_1, and $[i, j] \subset [g_2, k_2]$, which is a contradiction. □

Let F be a face incident to (v, u) such that the interval, $[g, k]$, of its entry edge, (w, x), does not enclose $[i, j]$; if both faces incident to (v, u) satisfy this condition, we choose F arbitrarily between them. Assume without the loss of generality that marker g stands on w while marker k stands on x. We pick dual edge (w, x) to promote to primal. The following lemma proves the correctness.

Lemma 7. $(T \setminus \{(v, u)\}) \cup \{(w, x)\}$ *is a spanning tree of* G.

Proof. All the markers that point to F are in $[g, k]$. Since either marker i or marker j points to F, i or j must be in $[g, k]$. Therefore, $[i, j]$ and $[g, k]$ must intersect, and $[g, k] \not\subseteq [i, j]$.

As F corresponds to the face whose entry edge interval, $[g, k]$, does not enclose $[i, j]$, we observe the following two cases:

$$1 \leq g < i < k < j \leq 2m \tag{5.1}$$

$$1 \leq i < g < j < k \leq 2m \tag{5.2}$$

To prove our lemma in either case, observe that the removal of edge (v, u) disconnects T into two connected components. One of these two components is T_u, the

subtree rooted at vertex u. By the definition of a Turán traversal, a marker stands on a vertex in T_u if and only if this marker is in $[i, j-1]$. The inequalities for these two cases then guarantee that the vertex that g stands on (which is vertex w) and the vertex that k stands on (which is vertex x) are in different components of $T \setminus \{(u, v)\}$, and the lemma follows. □

We are now ready to describe our algorithm for primal edge deletion. To delete the primal edge (v, u) enumerated at the ith step of a Turán traversal of G, we first compute the step j where the Turán traversal traverses (v, u) in the reverse direction, i.e., from u to v. As marker i points to the face on one side of (v, u) and marker j points to the face on the other side of (v, u), this follows from Lemma 3, we then perform $\texttt{vertex}(T^*, \texttt{rank}_0(A, i))$ and $\texttt{vertex}(T^*, \texttt{rank}_0(A, j))$ to compute the faces adjacent to (v, u). If these faces are the same, then deleting (v, u) would disconnect G, so we do not remove (v, u) and immediately return. Otherwise, the faces adjacent to (v, u) are different. Recall that we define F to be a face incident to (v, u) whose entry edges' interval range does not enclose the range $[i, j]$. Let F' be the other face incident to (v, u). We now compute $[g, k]$, the entry edge interval of F as follows. Since F is incident to (v, u), one of the markers, i or j, points to F. Thus, one of $\texttt{rank}_0(A, i)$ or $\texttt{rank}_0(A, j)$ corresponds to the entry edge of F in T^*. We apply $\texttt{select}_0$ to the rank of 0's corresponding to the marker, i or j, that points to F and call this value g. Then, $k = \texttt{select}_0(A, \texttt{inverse}(T^*, \texttt{rank}_0(A, g)))$. Without loss of generality we assume $g < k$.

Next we update T and T^* to reflect the deletion of (v, u) and the promotion of (w, x). With respect to T, deleting a primal edge from G corresponds to deleting an edge from T, thereby disconnecting T. By Lemma 7, after deleting (v, u) and promoting (w, x), T remains a spanning a tree of G. Let T_v be the tree containing v and T_u be the tree containing u, before the deletion of (v, u). We delete (v, u) from T. By promoting (w, x), we are connecting T_v and T_u at corners $\{\texttt{rank}_1(A, g), \texttt{rank}_1(A, g) + 1\}$ and $\{\texttt{rank}_1(A, k), \texttt{rank}_1(A, k) + 1\}$. As for T^*, promoting (w, x) to primal corresponds to deleting the edge connecting the faces on either side of (w, x) so, we delete the Euler tour edge in T^* corresponding to $\texttt{rank}_0(A, g)$. This creates two subtrees in T^*, ${T^*}_F$ and ${T^*}_{F'}$, where one subtree contains F and the other contains F'.

Deleting (v, u) corresponds to merging faces F and F' and thereby reconnecting T^*. To merge these faces in T^* we first add an edge to connect the two subtrees, ${T^*}_F$ and ${T^*}_{F'}$, at the corners $\{\texttt{rank}_0(A, i), \texttt{rank}_0(A, i) + 1\}$ and $\{\texttt{rank}_0(A, j), \texttt{rank}_0(A, j) + 1\}$ and temporarily store a reference to this newly added edge, ℓ, and its inverse, ℓ'. Then, we merge F and F' by performing $\texttt{merge}(T^*, \texttt{vertex}(T^*, \ell), \texttt{vertex}(T^*, \ell'))$. Lastly, we decrement m. By Lemma 2, merging vertices and deleting edges in a succinct Euler-Tour tree takes at most $O(\lg^{1+\epsilon} n)$ time. Figure 5.2 shows an example.

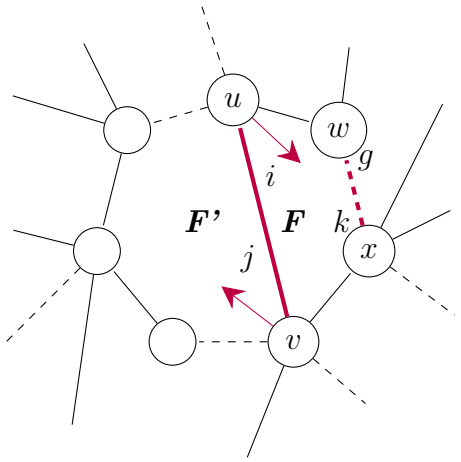


Figure 5.2: Deleting primal edge (v, u) where marker i corresponds to the edge (v, u), stands on u, and points to the face, F, to the right of (v, u), marker j corresponds to the edge (u, v), stands on v, and points to the face, F', to the left of (v, u), and (w, x) refers to the dual edge that corresponds to the face F whose entry edge interval $[g, k]$ does not enclose $[i, j]$. The marker g corresponds to (w, x) and marker k corresponds to (x, w). After deleting (v, u), the faces F and F' will merge together and (w, x) will be promoted to primal.

Finally, we show how to update A. There are two cases. In the first case, inequality 5.1 holds. In this case, we update A to $A[1, g-1].1.A[k+1, j-1].A[i+1, k-1].1.A[g+1, i-1].A[j+1, 2m]$, where "." is the concatenation operator for bitvectors. This can be done using a constant number of `flip`, `delete`, `cut`, and `link` operations over A in $O(\lg^{1+\epsilon} n)$ time. To see the correctness, consider how the Turán traversal works after edge deletion; in subsequent discussions, the tth edge refers to

the tth edge enumerated in the Turán traversal before edge deletion unless otherwise stated. First, observe that the Turán traversal begins with the same first edge, and, therefore, the newly computed bitvector also begins with $A[1, g-1]$. The bit at $A[g]$ should be flipped from 0 to 1, to reflect the promotion of (w, x) to primal. Here, we append a 1 after $A[1, g-1]$. Recall that (w, x) corresponds to this dual edge. By flipping the bit in $A[g]$ to reflect the promotion to a primal edge, the marker would be standing on x in the gth step of the Turán traversal after edge deletion by Lemma 3.

Since the kth edge represents the inverse of the gth edge, the kth edge will not be examined until the Turán traversal has explored all other vertices counterclockwise from x and then lastly examines (x, w), the kth edge. Thus, after the marker arrives on x, it continues the Turán traversal counterclockwise from (w, x) when standing on x. As the kth edge was a dual edge before edge deletion, marker k would be standing on x and examine the next edge counterclockwise from (x, w). Thus, the next edge examined, after promoting the gth edge to primal, is the $(k+1)$st edge. The Turán traversal would continue until reaching the jth edge. As we are deleting the jth edge, it should not be included in our newly computed bitvector. Therefore, we then concatenate the bits $A[k+1, j-1]$.

Since we are removing the jth edge, which is the inverse of the ith edge, the Turán traversal would continue to stand on vertex u rotating counterclockwise. Thus, the next edge encountered after the $(j-1)$st edge, after removing the jth edge, and thereby the ith edge, would be the $(i+1)$st edge. The Turán traversal would then continue its traversal, exploring G after the $(i+1)$st edge until reaching the kth edge. As this corresponds to the edge we wish to promote to primal, the Turán traversal would be different after it. Therefore, we then concatenate $A[i+1, k-1]$. The bit at $A[k]$ should be flipped from 0 to 1 so we immediately concatenate a 1 bit after $A[i+1, k-1]$.

Recall that the gth edge used to be a dual edge and that the kth edge is the inverse of the gth edge. As the kth edge was just promoted to primal, by concatenating a 1 bit after the $k-1$ bit above, the marker would then traverse (x, w) and, by Lemma 3,

be standing on w. We observe that the next edge in the counterclockwise direction on w from (x, w) would be the $(g+1)$st edge. The Turán traversal would continue exploring G from the $(g+1)$st edge until reaching the ith edge. As we wish to delete the ith edge, it should not be included in our range. Thus, we then concatenate $A[g+1, i-1]$.

After examining the $(i-1)$st edge, the Turán traversal would have continued along the ith edge. However, we have deleted the ith edge, and thereby the jth edge, from G. Therefore, the next edge examined in counterclockwise order after the $(i-1)$st edge would be the $(j+1)$st edge. The Turán traversal would then traverse the remainder of G. This final range corresponds to a Turán traversal from the $(j+1)$st edge to the $(2m)$th edge, the last edge, so we concatenate $A[j+1, 2m]$.

With respect to the second case, inequality 5.2 holds. In this second case, we update A to $A[1, i-1].A[j+1, k-1].1.A[g+1, j-1].A[i+1, g-1].1.A[k+1, 2m]$. This bitvector is obtained by similar reasoning as the first case above. Thus, we have the following lemma:

Lemma 8. *An edge can be deleted from G in $O(\lg^{1+\epsilon} n)$ time, so long as G remains connected.*

5.2 Inserting and Deleting Vertices

Recall from the introduction of Section 5 that we support insertions and deletions of sticks. This is done to minimize the changes to the Turán traversal. We observe that by supporting insertions and deletions of sticks, a face of G is never split and the connectivity and planarity of G is maintained throughout. Figure 5.3 shows an example. Vertex deletion and insertion is described in the subsections that follow.

5.2.1 Inserting a Vertex

We now discuss vertex insertion. Given a vertex v and face F, that v is a vertex of, we wish to insert u into our representation of G on F as a neighbor of v. Since a single `rotate_ccw` or `rotate_cw` operation can make a marker point to the face on the other side of the edge that the marker corresponds to, we assume that marker i stands on

v and points to F for simplicity. As we wish to insert u on F, we attach u to v by adding the primal edge, (v, u), and its inverse, (u, v). This corresponds to inserting two consecutive 1 bits into A. As the edges inserted are primal edges, attaching u to v with a primal edge corresponds to having the $(i + 1)$st edge enumerated in a Turán traversal be (v, u). The marker $(i + 1)$ would then stand on u but continue to point to F. Since u is a stick, the edge after the $(i + 1)$st edge, the $(i + 2)$nd edge, enumerated in a Turán traversal would be (u, v), where marker $(i + 2)$ would stand on v and continue to point to F. Thus, we update A to reflect a new vertex u attached to v with marker i by $\texttt{insert}(A, i+1, 1)$. We then increment i, and perform $\texttt{insert}(A, i+1, 1)$ again. As we have inserted u, and a new primal edge connecting it to v, T must be updated to reflect an inserted vertex. We compute the corner in T to attach the new vertex to. The corner is computed as the most recently traversed edge and the next edge in T. Thus, we insert a new vertex into T at the corner computed by $\{\texttt{rank}_1(A, i), \texttt{rank}_1(A, i) + 1\}$. Lastly, we increment n and m to reflect the new vertex and edge in G. These steps use $O(\lg^{1+\epsilon} n)$ time. Thus, we have the following lemma:

Lemma 9. *Given a vertex v and a face that v is a vertex of, a degree-1 vertex u can be inserted in the face such that v is adjacent to u in $O(\lg^{1+\epsilon} n)$ time.*

5.2.2 Deleting a Vertex

We now discuss vertex deletion. Given a degree-1 vertex v, we wish to delete v and its adjacent edge in our representation of G. We assume marker i stands on v for simplicity. As a stick in a Turán traversal is visited, via (u, v), and then immediately traversed back from, via (v, u), the bits in A corresponding to the primal edges (u, v) and (v, u) are next to each other in A. Since marker i stands on v, this means that ith edge enumerated in a Turán traversal is (u, v) and the $(i + 1)$st edge corresponds to (v, u). To remove v from A, we perform $\texttt{delete}(A, i)$. Since (u, v) was removed due to this call to `delete`, the marker i now refers to (v, u). So, we perform $\texttt{delete}(A, i)$ again. As we are deleting a vertex from G, T must be updated to reflect this. To update T we delete the edge connecting the corresponding vertex to T and then delete the isolated vertex. To delete the edges connecting v to T, we remove the Euler tour edge $\texttt{rank}_1(A, i)$ from T. Lastly, we decrement n and m. These steps use $O(\lg^{1+\epsilon} n)$

time. Thus, we have the following lemma:

Lemma 10. *Given a degree-1 vertex v, v and its adjacent edge can be deleted from G in $O(\lg^{1+\epsilon} n)$ time.*

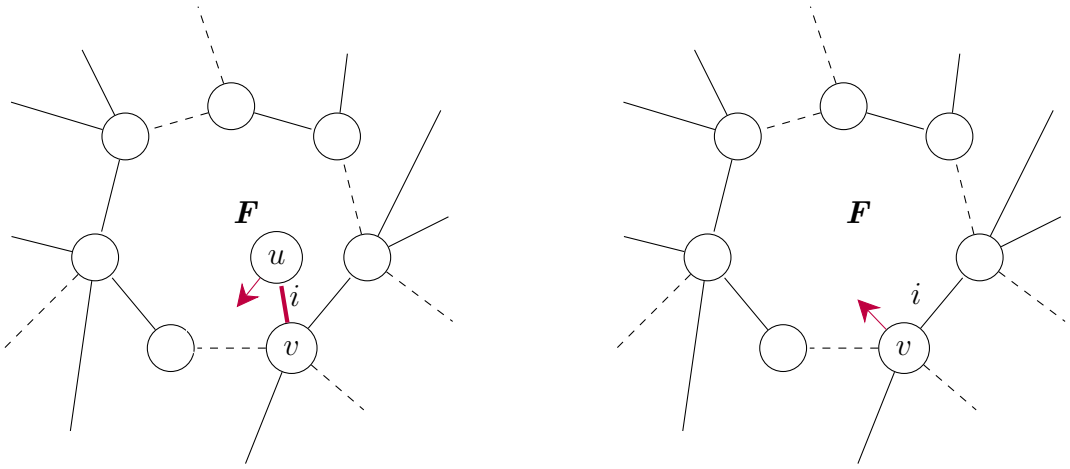


Figure 5.3: The insertion and deletion of degree-1 vertex u into G on face F. Going from the right figure to the left figure, this demonstrates how the insertion of u corresponds to inserting two consecutive 1 bits into A to represent (v, u) and (u, v). Going from the left figure to the right figure, this depicts how the deletion of a degree-1 vertex corresponds to deleting two consecutive 1 bits from A that represent (v, u) and (u, v). Looking in either direction shows how the insertion or deletion of a degree-1 vertex does not split F.

Chapter 6

Conclusion

We have studied the problem of representing a compact connected planar embedding under insertions and deletions of edges and vertices. Edge and vertex insertion and deletion are supported in $O(\lg^{1+\epsilon} n)$ time for any constant $\epsilon > 0$. We also presented a way to navigate within our representation of G in $O(\lg n)$ time.

Although related prior works exist on dynamic succinct planar graphs, those works either do not support a full set of update operations [15] or have restrictions on the sizes and shapes of the faces [2] in G. To the best of our knowledge, this thesis presents the first known representation of dynamic compact connected planar embeddings that supports a full set of update operations without limitations on the sizes or shapes of the faces.

Possible future work includes designing additional $o(m)$ bit data structures to reduce the time to compute the degree of a given vertex. Another interesting problem would be to extend or modify this representation to handle disconnected planar embeddings.

Bibliography

[1] Muhammad Akram, Jawaria Mohsan Dar, and Adeel Farooq. Planar graphs under Pythagorean fuzzy environment. *Mathematics*, 6(12):278, 2018.

[2] Luca Castelli Aleardi, Olivier Devillers, and Gilles Schaeffer. Dynamic updates of succinct triangulations. Technical report, 2005.

[3] Jérémy Barbay, Luca Castelli Aleardi, Meng He, and J Ian Munro. Succinct representation of labeled graphs. *Algorithmica*, 62:224–257, 2012.

[4] Guy E Blelloch and Arash Farzan. Succinct representations of separable graphs. In *Annual Symposium on Combinatorial Pattern Matching*, pages 138–150. Springer, 2010.

[5] Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications. *Society for Industrial and Applied Mathematics Journal on Computing*, 34(4):924–945, 2005.

[6] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[7] Arash Farzan and J Ian Munro. Dynamic succinct ordered trees. In *International Colloquium on Automata, Languages, and Programming*, pages 439–450. Springer, 2009.

[8] Leo Ferres, José Fuentes-Sepúlveda, Travis Gagie, Meng He, and Gonzalo Navarro. Fast and compact planar embeddings. *Computational Geometry*, 89:101630, 2020.

[9] José Fuentes-Sepúlveda, Gonzalo Navarro, and Diego Seco. Navigating planar topologies in near-optimal space and time. *Computational Geometry*, 109:101922, 2023.

[10] Travis Gagie and Sebastian Wild. Succinct Euler-Tour Trees. In Meng He and Don Sheehy, editors, *Proceedings of the 33rd Canadian Conference on Computational Geometry, August 10-12, 2021, Dalhousie University, Halifax, Nova Scotia, Canada*, pages 368–376, 2021.

[11] Monika R Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the Association for Computing Machinery*, 46(4):502–516, 1999.

[12] Jacob Holm and Eva Rotenberg. Dynamic planar embeddings of dynamic graphs. *Theory of Computing Systems*, 61:1054–1083, 2017.

[13] Alexander Irribarra-Cortés, José Fuentes-Sepúlveda, Diego Seco, and Roberto Asín. Speeding up compact planar graphs by using shallower trees. In *2022 Data Compression Conference*, pages 282–291. IEEE, 2022.

[14] Guy Jacobson. Space-efficient static trees and graphs. In *30th annual symposium on foundations of computer science*, pages 549–554. IEEE Computer Society, 1989.

[15] Frank Kammer and Johannes Meintrup. Succinct planar encoding with minor operations. *arXiv Computing Research Repository*, abs/2301.10564, 2023.

[16] Kenneth Keeler and Jeffery Westbrook. Short encodings of planar graphs and maps. *Discrete Applied Mathematics*, 58(3):239–252, 1995.

[17] Oscar Levin. Discrete mathematics: An open introduction. 2021.

[18] J Ian Munro. Tables. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 37–42. Springer, 1996.

[19] Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *Association for Computing Machinery Transactions on Algorithms*, 10(3):1–39, 2014.

[20] Robert E Tarjan and Uzi Vishkin. An efficient parallel biconnectivity algorithm. *Society for Industrial and Applied Mathematics Journal on Computing*, 14(4):862–874, 1985.

[21] György Turán. On the succinct representation of graphs. *Discrete Applied Mathematics*, 8(3):289–294, 1984.

[22] William Thomas Tutte. A census of planar maps. *Canadian Journal of Mathematics*, 15:249–271, 1963.

[23] K. G. C. von Staudt. *Geometrie de Lage*. Bauer und Raspe, Nürnberg, 1847.

[24] Robin J. Wilson. *Introduction to Graph Theory*. Prentice Hall/Pearson, New York, 2010.