Dynamic Compact Planar Embeddings*

Travis Gagie[0000−0003−3689−327X], Meng He[0000−0003−0358−7102], and Michael St Denis[0009−0002−7163−576X]

Dalhousie University, Halifax NS B3H 4R2, Canada
travis.gagie@dal.ca, mhe@cs.dal.ca, michael.stdenis@dal.ca

Abstract. This paper 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 consider decremental settings only 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 paper 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.

Keywords: Planar embedding · Dynamic planar embedding · Dynamic compact data structures.

1 Introduction

A particular type of graphs, planar graphs, may be used to model the famous initial graph problem known as the seven bridges of Königsberg [21]. Aside from this application, planar graphs are also applicable to some maps in general, VLSI circuits [11], chemical molecules [1], and spatial partitions in geographical information systems (GIS) [11].

A more contemporary problem concerns the dramatic growth of problem sizes with respect to the growth in computer memory [7,16]. 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 [12].

This paper 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

* This work was supported by NSERC of Canada.

graphs where $O(n)$ bits are used to support common navigational operations such as adjacency testing and listing a vertex's neighbors [3,4,5,7,12,14,19].

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 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 consider the decremental setting under which the graph is updated by edge contractions and vertex deletions only [13]. This paper 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 an 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 incident to, list the edges incident to F in clockwise or counterclockwise order starting from (v, u).
- Given two corners of a face, insert an edge between their apexes, bisecting these corners. Here, a *corner* is defined as the space between consecutive edges incident to a vertex [10], and this vertex is the *apex* of the corner.
- Delete a given edge from G so long as G remains connected.
- Given a corner with apex v, insert a degree-1 vertex u in the corner as a new neighbor of v.
- Given a degree-1 vertex v, delete v and the edge e it is incident to from G.

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 support 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 paper 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 navigational operations within a given connected planar embedding. This model is similar to the finger-update model [6] 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 corner in the given planar embedding.

2 Related Work

We survey previous results on succinct representations of planar embeddings [3,4,7,14,19,20]. Succinct planar graph representations that cannot encode an arbitrary embedding are not included; see [4] for a survey including those results. Tutte [20] 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 [19] derived a simple succinct encoding of planar graphs that uses $4m$ bits. Keeler and Westbrook [14] 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 neighbor. 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. [7] modified Turán's [19] structure to occupy $o(m)$ additional 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 $O(f(m))$ time for any function $f(m) \in \omega(1)$ and testing if two given vertices are neighbors in $O(f(m))$ time for any 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. [8] further showed that by using *half-edges* and condensing Ferres et al.'s [7] data structures, a full set of topological queries can be supported efficiently.

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] designed 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 $O(1)$ amortized time without supporting access to satellite data associated with each vertex, $O(\lg n)$ amortized time with data access, and vertex deletion or edge flip[1] in $O(\lg^2 n)$ amortized 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 Meintrup [13] provided a dynamic data structure, a modification of Blelloch and Farzan's [4], that encodes a planar graph in $\mathcal{H}(n) + o(n)$ bits to support an arbitrary sequence of edge contractions and vertex deletions in $O(n)$ time, where $\mathcal{H}(n)$ is the entropy of encoding an n-vertex planar graph. It can compute the degree of a vertex in $O(1)$ time and list the neighbors in $O(1)$ time per neighbor. 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.

3 Preliminaries

3.1 Notation

We assume the word-RAM model of computation. For the remainder of this paper, let G be a given connected planar embedding on n vertices, m edges and f faces that may contain multi-edges and self loops. 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):

- `access`(B, i): return the bit in $B[i]$ for any i such that $1 \leq i \leq n$.
- `rank`$_b(B, i)$: return the number of occurrences of b in $B[1..i]$ where $1 \leq i \leq n$.
- `select`$_b(B, j)$: return the index of the jth occurrence of b in B.
- `insert`(B, i, b): inserts bit b between $B[i-1]$ and $B[i]$ where $1 \leq i \leq n$.
- `delete`(B, i): deletes the bit in position $B[i]$ where $1 \leq i \leq n$.
- `link`(B, p, B'): attaches all the bits in B' between bits $B[p]$ and $B[p+1]$ where $1 \leq p \leq n$ and $|B'|$ is another bit vector with $O(n)$ bits.
- `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 $[(j+1)..n]$.

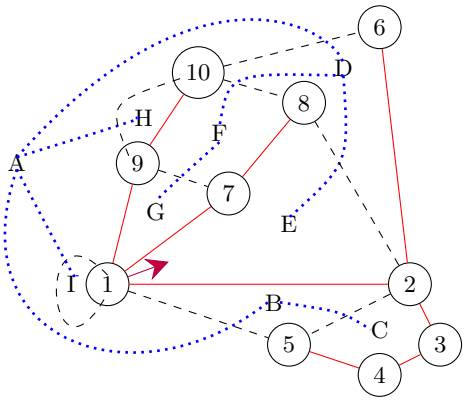
Navarro and Sadakane [17] present the following:

Lemma 1. ([17]). *There exists a succinct dynamic bitvector structure that supports* `access`*,* `rank`*,* `select`*,* `insert` *and* `delete` *in* $O(\frac{\lg n}{\lg \lg n})$ *time and* `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 [19], which is used by Ferres et al. [7] 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



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

Fig. 1: Planar embedding G where the red solid edges correspond to edges in T, the black dashed edges correspond to edges in $G \setminus T$, and blue dotted edges correspond to edges in T^*, the dual of G complementary to T. Each vertex of G is labeled by its preorder rank in T, while each face is labeled alphabetically in the order they are first visited in the Euler tour of 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.

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 bitvector A. Afterwards, we examine the next edge incident to u after (u, v) in counterclockwise order. 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 the dual [18]. As the Turán traversal in this paper is performed in counterclockwise order, the traversal of T^* is performed clockwise from its root, i.e., the outer face. Figure 1 gives an example.

3.4 Dynamic Succinct Euler-Tour Trees

Gagie and Wild [9] 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 contains 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. Each tree in the forest is unrooted, and its Euler tour determines the current parent-child relationship among its nodes; this relationship may change during updates. 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 [9], the `merge` and `split` operations are implied, but we state them explicitly as operations 10 and 11. Throughout this paper, we refer to operations 3, 4 and 5 in the lemma below as `vertex`, `entry` and `inverse`, respectively. Note that Gagie and Wild did not state the support for `entry`, but their existing data structures can support it easily. The following lemma summerizes their results:

Lemma 2. ([9]). *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 5 below take constant time, operations 6 and 7 take $O(\lg n)$ time, and operations 8 through 11 take $O(\lg^{1+\epsilon} n)$ time for any 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 given directed edge e in the Euler tour of T.*
4. *return the directed edge e encountered in the Euler tour traversal of T such that, after traversing e, the Euler tour arrived at the given vertex v the first time, i.e., e links the parent of v in the Euler tour traversal to v;*
5. *return the edge e' such that, given a tree T and an edge e encountered in the Euler tour of T, edge e' corresponds to the inverse edge of e.*
6. *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.*
7. *return the Euler tour distance between the given edges e and e' so long as the two edges are in the same tree.*
8. *delete the given edge e from the tree containing it and return the representations of the two resulting trees.*
9. *insert an edge between T and T' at the given corners, bisecting those corners, and return the representation of the resulting tree.*
10. *merge the adjacent vertices u and v to become one vertex and retain all other edges adjacent to u and v.*

11. *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 9 supports the insertion of an edge between two arbitrary vertices in two trees. This operation cannot be supported by the dynamic succinct tree representation of Navarro and Sadakane [17] which is based on a balanced parenthesis representation, but it is needed in our dynamic planar graph representation. We also comment that, to support operations 1-5 in constant time, each edge is identified by a unique internal identifier. Operations 6 and 7 can perform mapping between this identifier and the rank of the edge in the Euler tour in $O(\lg n)$ time. Thus, when the context is clear, we may also pass or return the rank of an edge in the Euler tour when calling `vertex`, `entry` or `inverse`, and the increase in running time does not affect the complexity of our solution.

4 Data Structure and The Marker Model

4.1 Data Structure

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. By Euler's formula [15], the total space cost of our data structure is $2m + o(m) + 2n + o(n) + 2(m - n + 2) = 4m + o(m)$ bits.

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 an index i in A, where $1 \leq i \leq 2m$. Recall that a Turán traversal of G induces an Euler tour traversal on T and T^*. Thus, we say that a marker stands on the vertex most recently visited in the Euler tour of T and points to 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 traversed 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 1 shows marker 16 as an example.

The following lemma shows how the face that a marker i points to relates to the edge $A[i]$ represents. Its proof is omitted due to space constraints.

Lemma 3. *Let (v, u) be the directed edge represented by $A[i]$. If (v, u) is a primal edge, then the marker i 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).*

Because we do not require G to be bi-connected, 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, but each marker points to a different corner of the face. Therefore, we define the *orientation* of a marker as the vertex it stands on and the corner it points to. More formally, let marker i refer to directed edge (v, u). By Lemma 3, if (v, u) is primal, then marker i stands on u and points to the corner that has u as its apex and is on the right of directed edge (v, u). If (v, u) is a dual edge, then marker i stands on v and points to the corner that has v as its apex and is to the left of directed edge (v, u).

If more than one marker is maintained at a given time, then, after an update, all markers must be updated to preserve 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 time to update all markers is linear in the number of markers maintained.

4.3 Navigation

Now we define two rotation operations and a traverse operation. Either rotate operation changes the corner 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 corner C of face F be the corner marker i currently points to; they will also be referred to when we discuss how to support these operations later.

- `rotate_ccw`(i): Compute a new orientation of the marker such that the marker is still standing on v but is pointing to the corner next to C when listing all the corners 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 corner next to C when listing all the corners 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.

To support `rotate_ccw`(i), 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 the ith edge is a primal edge, then, by Lemma 3, it is oriented from u to v. Furthermore, corner C 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 corner C is again to the left of

(v, u). In either case, since (v, w) is the edge next to (v, u) in counterclockwise order, C is to the right of (v, w). Therefore, our goal is to compute a marker still standing on v but pointing to the corner, C', to the left of (v, w); C' is next to C when listing all the corners 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 and the definition of marker orientation, marker $i+1$ continues to stand on v but points to C'. 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 $j =$`select`$_1(A,$`inverse`$(T,$`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 corner 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 C' in this case.

The details of how to support `rotate_cw` and `traverse` operations are omitted due to space constraints. In the worst case, at most two dynamic bitvector operations and one succinct Euler-Tour tree operation are performed to support each navigational operation. Combining this with Lemmas 1 and 2, we have

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

These rotation and traverse operations imply the support for listing the edges incident to a vertex or a face of G. For example, to list the edges incident to a vertex v in counterclockwise order, we start from a marker standing on v and call `rotate_ccw` repeatedly. Details are omitted due to space limitations. Thus, we have proved the support of navigational operations stated in Theorem 1.

5 Dynamization

We now prove the support of updates stated in Theorem 1. As the support for edge insertions and deletions (especially edge deletions) is more interesting, we discuss them here. Descriptions of how to insert or delete vertices are omitted due to space limitations.

5.1 Inserting an Edge

To minimize the changes to the Turán traversal, a new edge is always inserted as a dual edge. To insert an edge, we need two markers, i and j, pointing to two corners of the same face F. 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. We assume, without the loss of generality, that $i < j$.

The `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, no matter if this edge is a dual edge or a primal edge, the corner, C, that marker i points to is to the left of (v, w). If we rotate counterclockwise from (v, w), with v as the pivot, C is the first corner encountered and is encountered before any other edge incident to v. Therefore, when inserting an edge bisecting C, 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. An example depicting this is omitted due to space constraints. Thus, we have the following lemma:

Lemma 5. *Given two corners of the same face, an edge connecting their apexes and bisecting these corners can be inserted into G in $O(\lg^{1+\epsilon} n)$ time.*

5.2 Deleting an Edge

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 disconnect G. As the steps for dual edge deletion are symmetric to those for edge insertion, their descriptions are omitted while focusing on primal edge deletion in this section.

To discuss primal edge deletion, 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 remains connected after deleting (v, u), how do we choose a dual edge to promote to primal? The third item is, how are T and T^* affected by primal edge deletion? Lastly, 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).

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 it is the only edge linking two connected components of $G \setminus \{(v, u)\}$. If the faces are different, then (v, u) can be deleted while G remains connected.

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 reconnects these two trees. 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 indices in A corresponding to the entry edges of F_1 and F_2, respectively, and let k_1 and k_2 be the indices of the reverse of the edges corresponding to $A[g_1]$ and $A[g_2]$, respectively. 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, if the entry edge intervals of two faces intersect, one must enclose the other. Since $g_1 < g_2$, we have $[g_2, k_2] \subset [g_1, k_1]$, and the node, f_2, of T^* representing F_2 is a descendant of the node, f_1, of T^* representing F_1. 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. Then marker g stands on w while marker k stands on x. We promote dual edge (w, x) to primal because:

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 strictly between g and k. Therefore, $[i, j]$ and $[g, k]$ must intersect, and $[g, k] \not\subseteq [i, j]$. As F is the face whose entry edge interval, $[g, k]$, does not enclose $[i, j]$, we observe the following two cases:

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

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

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. By Lemma 3, 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). Hence, we perform $v_1 = \texttt{vertex}(T^*, \texttt{rank}_0(A, i))$ and $v_2 = \texttt{vertex}(T^*, \texttt{rank}_0(A, j))$ to compute the nodes of T^* that respectively represent 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 intervals of these faces are $[\texttt{select}_0(A, \texttt{entry}(T^*, v_1)), \texttt{select}_0(A, \texttt{inverse}(T^*, \texttt{entry}(T^*, v_1)))]$ and $[\texttt{select}_0(A, \texttt{entry}(T^*, v_2)), \texttt{select}_0(A, \texttt{inverse}(T^*, \texttt{entry}(T^*, v_2)))]$. We compare these intervals to $[i, j]$ to determine which of these two faces should be chosen to be F so that F is a face incident to (v, u) whose entry edge's interval, $[g, k]$, does not enclose $[i, j]$. Let F' be the other face incident to (v, u).

Next we update T and T^* to reflect the deletion of (v, u) and the promotion of (w, x). 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, after the deletion of (v, u). 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 $\texttt{merge}(T^*, \texttt{vertex}(T^*, \ell), \texttt{vertex}(T^*, \ell'))$. By Lemma 2, merging vertices and deleting edges in a succinct Euler-Tour tree takes at most $O(\lg^{1+\epsilon} n)$ time.

Finally, we show how to update A. There are two cases. In the first case, inequality 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 `insert`, `delete`, `cut`, and `link` operations over A in $O(\lg^{1+\epsilon} n)$ time. The correctness can be shown by analyzing how the Turán traversal works after edge deletion; details are omitted due to space constraints. With respect to the second case, inequality 2 holds, and 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.

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

References

1. Akram, M., Mohsan Dar, J., Farooq, A.: Planar graphs under Pythagorean fuzzy environment. Mathematics **6**(12), 278 (2018)
2. Aleardi, L.C., Devillers, O., Schaeffer, G.: Dynamic updates of succinct triangulations. Tech. rep. (2005)
3. Barbay, J., Castelli Aleardi, L., He, M., Munro, J.I.: Succinct representation of labeled graphs. Algorithmica **62**, 224–257 (2012)
4. Blelloch, G.E., Farzan, A.: Succinct representations of separable graphs. In: Annual Symposium on Combinatorial Pattern Matching. pp. 138–150. Springer (2010)
5. Chiang, Y.T., Lin, C.C., Lu, H.I.: Orderly spanning trees with applications. Society for Industrial and Applied Mathematics Journal on Computing **34**(4), 924–945 (2005)
6. Farzan, A., Munro, J.I.: Dynamic succinct ordered trees. In: International Colloquium on Automata, Languages, and Programming. pp. 439–450. Springer (2009)
7. Ferres, L., Fuentes-Sepúlveda, J., Gagie, T., He, M., Navarro, G.: Fast and compact planar embeddings. Computational Geometry **89**, 101630 (2020)
8. Fuentes-Sepúlveda, J., Navarro, G., Seco, D.: Navigating planar topologies in near-optimal space and time. Computational Geometry **109**, 101922 (2023)
9. Gagie, T., Wild, S.: Succinct Euler-Tour Trees. In: He, M., Sheehy, D. (eds.) Proceedings of the 33rd Canadian Conference on Computational Geometry, August 10-12, 2021, Dalhousie University, Halifax, Nova Scotia, Canada. pp. 368–376 (2021)
10. Holm, J., Rotenberg, E.: Dynamic planar embeddings of dynamic graphs. Theory of Computing Systems **61**, 1054–1083 (2017)
11. Irribarra-Cortés, A., Fuentes-Sepúlveda, J., Seco, D., Asín, R.: Speeding up compact planar graphs by using shallower trees. In: 2022 Data Compression Conference. pp. 282–291. IEEE (2022)
12. Jacobson, G.: Space-efficient static trees and graphs. In: 30th annual symposium on foundations of computer science. pp. 549–554. IEEE Computer Society (1989)
13. Kammer, F., Meintrup, J.: Succinct planar encoding with minor operations. arXiv Computing Research Repository **abs/2301.10564** (2023). https://doi.org/10.48550/arXiv.2301.10564, https://doi.org/10.48550/arXiv.2301.10564
14. Keeler, K., Westbrook, J.: Short encodings of planar graphs and maps. Discrete Applied Mathematics **58**(3), 239–252 (1995)
15. Levin, O.: Discrete mathematics: An open introduction (2021)
16. Munro, J.I.: Tables. In: International Conference on Foundations of Software Technology and Theoretical Computer Science. pp. 37–42. Springer (1996)
17. Navarro, G., Sadakane, K.: Fully functional static and dynamic succinct trees. Association for Computing Machinery Transactions on Algorithms **10**(3), 1–39 (2014)
18. von Staudt, K.G.C.: Geometrie de Lage. Bauer und Raspe, Nürnberg (1847)
19. Turán, G.: On the succinct representation of graphs. Discrete Applied Mathematics **8**(3), 289–294 (1984)
20. Tutte, W.T.: A census of planar maps. Canadian Journal of Mathematics **15**, 249–271 (1963)
21. Wilson, R.J.: Introduction to Graph Theory. Prentice Hall/Pearson, New York (2010)