# Efficiently Partitioning the Edges of a 1-Planar Graph into a Planar Graph and a Forest

**Sam Barr** ✉
University of Waterloo, Canada

**Therese Biedl** ✉
University of Waterloo, Canada

## Abstract

1-planar graphs are graphs that can be drawn in the plane such that any edge intersects with at most one other edge. Ackerman showed that the edges of a 1-planar graph can be partitioned into a planar graph and a forest, and claims that the proof leads to a linear time algorithm. However, it is not clear how one would obtain such an algorithm from his proof. In this paper, we first reprove Ackerman's result (in fact, we prove a slightly more general statement) and then show that the split can be found in linear time by using an edge-contraction data structure by Holm, Italiano, Karczmarz, Łącki, Rotenberg and Sankowski.

**2012 ACM Subject Classification** Mathematics of computing → Graph algorithms; Theory of computation → Dynamic graph algorithms

**Keywords and phrases** 1-planar graphs, edge partitions, algorithms, data structures

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.16

**Funding** *Therese Biedl*: Research supported by NSERC; FRN RGPIN-2020-03958.

## 1 Introduction

In this paper, we study the class of 1-planar graphs: graphs that can be drawn in the plane such that every edge crosses at most one other edge. 1-planar graphs were introduced by Ringel [21], motivated by the problem of coloring the vertices and faces of a planar graph. Since then, there have been many publications concerning 1-planar graphs, both for theoretical results such as coloring, as well as algorithmic results such as solving drawing and optimization problems. The reader may refer to [18] for an annotated bibliography from 2017 and [15] for a more recent book that includes developments since then.

Many of the results for 1-planar graphs are obtained by first converting the 1-planar graph $G$ into a planar graph $G'$, applying results for planar graphs, and then expanding the result from $G'$ back to $G$. (We will give specific examples below.) There are several ways of how to create $G'$, e.g., by deleting edges or by replacing crossings with dummy-vertices. Of particular interest to us here is to make a 1-planar graph planar by deleting edges. Put differently, we want an *edge partition*, i.e., write $E(G) = E' \cup E''$ such that $E'$ forms a planar graph while $E''$ has some special structure that makes it possible to expand a solution for $G'$ to one for $G$.

**Previous Work.** The main focus of this paper is a result by Ackerman [1]. He established that the edges of a 1-planar graph can be partitioned such that one partition induces a planar graph and the other induces a forest. This was an extension of an earlier result from Czap and Hudák [8], who proved it for optimal 1-planar graphs (simple 1-planar graphs with the maximum $4n-8$ edges). Other partitions of near-planar graphs have also been studied; we list a few here. Lenhart et al. [19] show that optimal 1-planar graphs can be partitioned into a maximal planar graph and a planar graph of maximum degree four (the bound of four is shown to be optimal). Bekos et al. [4] provide edge partition results for some $k$-planar graphs (graphs that can be drawn in the plane such that any edge crosses at most $k$ other edges). Di Giacomo et al. [9] prove edge partition results for so-called NIC-graphs, a subclass of 1-planar graphs.

For algorithmic purposes, we need to find such edge partitions in linear time. With one exception, the above papers either explicitly come with a linear-time algorithm to find the edge-partition, or such an algorithm can easily be derived from the proof. The one exception is the paper by Ackerman [1]. He claims that the partition of a 1-planar graph into a planar graph and a forest can be found in linear time, but provides no details. Moreover, while his proof clearly gives rise to a polynomial-time algorithm, it is not clear how one would achieve linear time (or even $O(n \log n)$ time), since he relies on contracting edges while repeatedly testing whether two vertices share a face and occasionally splitting the graph into subgraphs, and neither of these operations can trivially be done in constant time in planar graphs. (We confirmed this in private communication with Ackerman.)

**Our Results.** In this paper, we show that a partition of a 1-planar graph into a planar graph and a forest can be found in linear time. We were not able to use Ackerman's proof for this directly, so as a first step we re-prove the result in a slightly different way to avoid some problematic situations and so that then a linear-time algorithm can be established. A crucial ingredient for this is a data structure by Holm, Italiano, Karczmarz, Łącki, Rotenberg and Sankowski [14] which allows for efficiently contracting edges of planar graphs. To our knowledge, this data structure has not been implemented. Because of this, we also show that the partition can be computed in $O(n \log n)$ time using a simpler data structure based on incidence lists.

As a consequence of our result, a number of related problems can be solved in linear time:

- Angelini et al. [3] studied the problem of finding simultaneous quasi-planar drawings of graphs where some edges are fixed. They used Ackerman's partition result in order to find such a simultaneous drawing of a 1-planar graph and a planar graph and cited its claimed linear runtime; with our result the linear runtime of [3] is established.
- It is known that every 1-planar graph has *arboricity* 4, i.e., its edges can be partitioned into 4 forests. (This follows from Nash-Williams formula for arboricity [20] since 1-planar graphs have at most $4n - 8$ edges [6].) The arboricity (and the corresponding edge-partition) of a graph can be computed in polynomial time [11], but to our knowledge not in linear time. For *planar* graphs, a split into 3 forests can be found in linear time [24]. So with our result, the partition of a 1-planar graph into 4 forests can be done in linear time as well, by first partitioning into a forest and a planar graph and then applying [24] onto the planar graph.
- If $G$ is a bipartite 1-planar graph, then it has at most $3n - 8$ edges [16] and hence arboricity 3. With our result we can partition it into a forest and a planar bipartite graph in linear time. The planar bipartite graph can be partitioned into two forests in linear time [22]. In consequence, every 1-planar bipartite graph can be partitioned into three forests in linear time.
- From a partition into $d$ forests one easily obtains an edge orientation with in-degree at most $d$. For bipartite graphs, such an edge-orientation can be used to prove ($d$+1)-list-colorability [13], and this list-coloring can be found in $O(dn)$ time [5]. Putting everything together, therefore our paper fills the one missing gap to show that 1-planar bipartite graphs can be 4-list-colored in linear time.

Our paper is structured as follows: In Section 2 we go over necessary terminology and present Ackerman's proof in order to demonstrate that it does not immediately lead to a linear time algorithm. In Section 3 we present our new proof. In Section 4 we use our alternative proof to design an efficient algorithm for finding the partition, before concluding in Section 5.

## 2 Background

We assume basic familiarity with graph theory (see e.g. [10]). All graphs in this paper are finite and connected, but not necessarily simple.

For a (multi)graph $G$ and a vertex $x$ of $G$, $d(x)$ is the number of edges incident to $x$.

We recall that a (multi)graph $G$ is called *planar* if it can be drawn in the plane without edges crossing. Let $G$ be a planar (multi)graph given with a drawing $\Gamma$. The maximal regions of $\mathbb{R}^2 \setminus \Gamma$ are the *faces* of $G$. We add a *chord* to a face $f$ by adding an edge between two non-adjacent vertices on the boundary of $f$, and drawing the edge through the region of $f$. For each vertex $x$ with incident edges $e_1, \dots, e_k$, the drawing $\Gamma$ places these edges in some rotational clockwise order around $x$. The space between two edges which are adjacent in this rotational order form an *angle*. The *degree* of a face $f$ is the number of angles contained in the face. We say that a face $f$ is a *quadrangle* if it has degree 4. Note that this includes both faces with 4 vertices on their boundary, and some faces with fewer than 4 vertices on their boundary (see Figure 1(a)). If $f$ is a quadrangle with exactly 4 vertices on its boundary, then we call $f$ a *simple quadrangle* (the quadrangles of a simple planar graph will all be simple quadrangles). A face of degree 3 is a *triangle*, and a face of degree 2 is a *bigon*.

The *facial cycle* of a quadrangle $f$ is a 4-tuple $\langle z_0, z_1, z_2, z_3 \rangle$ such that each $z_i$ is on the boundary of $f$ and there are edges $z_i z_{i+1}$ and $z_i z_{i-1}$ (arithmetic modulo 4) which form an angle in $f$. Note that $z_0, z_1, z_2, z_3$ need not be distinct if $f$ has loops or parallel edges, or if it is incident to a bridge. We say that $z_0$ and $z_2$ are *opposing vertices in $f$* (we will often omit mentioning the face when it is clear from context). Likewise $z_1$ and $z_3$ are opposing vertices in $f$. Note that it is possible for a vertex to oppose itself in a quadrangle.

*Stellating* a face $f$ of a planar graph is the process of adding a new vertex $s$ inside $f$, and adding an edge from $s$ to every vertex on the boundary of $f$. Given two vertices $a$ and $b$, we *contract* $a$ and $b$ by creating a new vertex $c$, adding an edge $vc$ for each edge $va$ and $vb$, and deleting $a$ and $b$ and all their incident edges. If $a$ and $b$ were adjacent, then $c$ has a loop, and if $a$ and $b$ were both adjacent to a vertex $v$, then $c$ will have parallel edges to $v$. If $a$ and $b$ were both on the boundary of some face $f$ in a planar graph, then we can *contract $a$ and $b$ through $f$* by placing this new vertex $c$ inside $f$. This preserves planarity, but destroys the face $f$.

A *1-planar* graph is a graph $G$ that can be drawn in the plane such that any edge intersects with at most one other edge. Here "drawing" always means a *good drawing* (see e.g. [23]), in particular this means that no three edges cross in a point, that no edge intersects itself, and that incident edges do not cross. From now on, whenever we speak of a 1-planar graph, we assume that one particular 1-planar drawing has been fixed. A pair of edges that intersect each other are a *crossing pair*, and the point where they intersect is known as the *crossing point*. The *planarization* is the graph $G^\times$ obtained by replacing each crossing point with a new vertex. A 1-planar graph $G$ is *planar-maximal* if no *uncrossed edge* (i.e., an edge which does not intersect any other edge) can be added to the fixed drawing of $G$ without adding a loop or a bigon.

For algorithmic purposes, a drawing of a planar graph can be specified by giving the rotational clockwise order of edges at every vertex; this specifies the circuits bounding the faces uniquely. A drawing of a 1-planar graph can be specified by giving a drawing of its planarization, with the vertices resulting from a crossing point marked as such. Since testing 1-planarity is NP-hard [12], we assume that any 1-planar graph $G$ is given with such a drawing. We also assume that $G$ is planar-maximal, because any 1-planar graph can be made planar-maximal in linear time by adding edges [2], and having more edges can only make partitioning more difficult.

For ease of notation, we define a shortcut for our partition problem.

▶ **Definition 1.** *A graph $G$ has a* PGF-partition *if its edge-set $E(G)$ can be partitioned into two sets $A$ and $B$ such that $G[A]$ is a planar graph and $G[B]$ is a forest.*

## 2.1 Ackerman's Proof

To establish the difficulties of achieving a linear time algorithm, we briefly review here Ackerman's proof for the existence of a PGF-partition.

Let $G$ be a (planar-maximal) 1-planar graph without loops drawn in the plane. Remove all crossing pairs of $G$. Call the resulting graph $H$ the *(planar) skeleton* of $G$ [2]. Observe that the faces of $H$ are either bigons, triangles, or quadrangles, and that there is a 1-1 mapping between the quadrangles of $H$ and the crossing pairs of $G$. Moreover, by this 1-1 mapping and the fact that the two edges forming a crossing pair do not share an endpoint, one can see that the quadrangles of $H$ are in fact simple quadrangles. Ackerman, similarly to Czap and Hudák [8], establishes the following.

▶ **Lemma 2** (Ackerman [1]). *Let $G$ be a 1-planar graph, and let $H$ be the skeleton of $G$. If we can add a chord to every quadrangle of $H$ such that the chords induce a forest, then $G$ has a PGF-partition.*

**Proof.** Let $C$ be the set of chords added to $H$. By the 1-1 mapping between quadrangles of $H$ and crossing pairs of $G$, we know that exactly one edge from each crossing pair of $G$ is contained in $C$. In particular, each edge $e \in C$ forms a crossing pair with some edge $e'$ of $G$. Let $C'$ be the set of these edges $e'$. By assumption $G[C]$ is a forest. Moreover, the graph $H \cup C'$ is the graph $H$ plus a chord added to each quadrangle of $H$, and so is a planar graph. As $H \cup C'$ is also the graph induced by the edge-set $E(G) \setminus C$, this gives us the desired partition. ◀

Thus, in order to prove the existence of a PGF-partition, it suffices to show (typically by induction on the number of quadrangles) that such a set of chords can be found. For the induction to go through, Ackerman additionally forbids the chords from containing a path between two adjacent pre-specified vertices $x, y$.

▶ **Theorem 3** (Ackerman [1]). *Let $H$ be a planar multigraph without loops such that every face has degree at most four and all quadrangles are simple. Let $x, y$ be a pair of adjacent vertices of $H$. Then we can add a chord to every simple quadrangle of $H$ such that the subgraph induced by the chords is a forest and does not contain a path between $x$ and $y$.*

**Proof.** Proceed by induction on the number of quadrangles in $H$. If $H$ has no quadrangles, then the statement is trivial. Otherwise, let $f$ be a quadrangle with facial cycle $\langle z_0, z_1, z_2, z_3 \rangle$; by assumption $f$ is simple.

**Case 1.** The only face containing $z_0$ and $z_2$ is $f$; in particular $z_0$ and $z_2$ are not adjacent. Contract $z_0$ and $z_2$ through $f$. Let $H'$ be the graph resulting from this contraction. Observe that $H'$ has one fewer quadrangle than $H$. All other quadrangles remain simple since $f$ was the only face containing $z_0$ and $z_2$. Apply induction on $H'$ with the same pair of adjacent vertices $x, y$ to receive a set of chords $C'$, and then further add the chord $z_0z_2$. Chords have now been added to every quadrangle of $H$, and it is easy to see that we have not added a cycle, or a path from $x$ to $y$, in the chords.

**Case 2.** There is some face $f' \neq f$ containing $z_0$ and $z_2$, but the only face containing $z_1$ and $z_3$ is $f$. Proceed as in Case 1, except contract $z_1$ and $z_3$.

**Case 3.** None of the above. Then there is a face $f' \neq f$ containing $z_0$ and $z_2$, and there is a face $f'' \neq f$ containing $z_1$ and $z_3$. Ackerman argues that $f' = f''$ (see also Lemma 5), and therefore $f'$ is also a simple quadrangle. Observe that $G$ can be split into four connected subgraphs $H_0, H_1, H_2, H_3$, where $H_i$ contains $z_i$ and $z_{i+1}$ (addition modulo 4) on the boundary (see Figure 1(e)). One of these subgraphs, say $H_0$, will contain the adjacent pair $x, y$. Apply induction on $H_0$ with $x, y$, and apply induction on the other $H_i$ with the pair $z_i, z_{i+1}$. After induction, add the chords $z_0z_2$ in $f$, and $z_1z_3$ in $f'$. One verifies that the added chords do not add a path between $x$ and $y$ and that the chords do not create a cycle. ◀

## 3 An Alternative Existence Proof

While Ackerman's proof clearly leads to a polynomial time algorithm for finding the partition, it is not obviously linear since distinguishing between the cases and contracting are not obviously doable in constant time:

1. We need to test whether a given pair of vertices share more than one face.
2. The graph changes via contractions, and it is not obvious whether the existing data structures for efficiently contracting edges in planar graphs (e.g. [14]) would support (1) in constant time.
3. In Case 3 of Ackerman's proof, we need to identify the four subgraphs $H_0, H_1, H_2, H_3$. Furthermore, we need to determine which of these subgraphs contains the pair $x, y$. Neither operation is obviously doable in constant time.

We now give a different proof of the existence of a PGF-partition that either avoids these issues or addresses explicitly how to resolve them. The biggest change is how we handle Case 3 of Ackerman's proof. Ackerman used here a split into four graphs, which is necessary in order to maintain that all quadrangles are simple. We prove a more general statement that permits non-simple quadrangles and hence avoids having to split the graph. Moreover, we generalize Ackerman's "forbidden pair" $x$ and $y$ by choosing chords in such a way that the chords do not induce a path between *any* pair of vertices that were adjacent in the original graph $H$. Doing so simplifies the induction since we no longer need to keep track of where the vertices $x$ and $y$ are. Before we state this result we need a few helper-results that hold for all quadrangles (simple or not).

▶ **Lemma 4.** *Let $H$ be a plane multigraph without loops, let $f$ be a quadrangle of $H$, and let $\langle z_0, z_1, z_2, z_3\rangle$ be the facial cycle of $f$. If $z_i = z_{i+2}$ (addition modulo 4) for some $i$, then $z_{i+1} \neq z_{i+3}$, and there is no face $f' \neq f$ that contains $z_{i+1}$ and $z_{i+3}$.*

**Proof.** Up to renaming, we may assume that $i = 0$, so $z_0 = z_2$ (see Figure 1(a)). Assume for contradiction that $z_1 = z_3$. Then $f$ consists of several parallel edges between $z_0 = z_2$ and $z_1 = z_3$, and thus $f$ is not a quadrangle.

Assume by way of contradiction that there is some face $f' \neq f$ which contains $z_1$ and $z_3$. Subdivide one of the $z_0z_3$ edges to obtain a new vertex $y$ adjacent to $z_0$ and $z_3$, and further add an edge $yz_1$ within $f$. We have split $f$ and attained a simple quadrangle $f''$ with facial cycle $\langle z_0, z_1, y, z_3 \rangle$. Stellate $f''$ with a new vertex $c$, and add an edge $z_1z_3$ through the face $f'$. All these steps maintain the planarity of $H$. Moreover, the five vertices $z_0, z_1, y, z_3, c$ are pairwise adjacent. But this forms a $K_5$ which is not planar, a contradiction. ◀

The following lemma was shown (without being stated explicitly) in Case 3 of Ackerman's proof.

▶ **Lemma 5.** *Let $H$ be a plane multigraph without loops, let $f$ be a quadrangle of $H$, and let $\langle z_0, z_1, z_2, z_3 \rangle$ be the facial cycle of $f$. If $z_i$ and $z_{i+2}$ (addition modulo 4) are both on some face $f' \neq f$ for some $i$, then no face $f'' \neq f, f'$ contains both $z_{i+1}$ and $z_{i+3}$.*

**Proof.** Up to renaming we may assume that $i = 0$, so $z_0$ and $z_2$ are on $f$ and $f'$ (see Figure 1(b-e)). Suppose for contradiction that such a face $f''$ exists. By Lemma 4, $z_1 \neq z_3$ and $z_0 \neq z_2$. Stellate $f$ with a new vertex $c$, add an edge $z_0z_2$ through $f'$, and add an edge $z_1z_3$ through $f''$. The original multigraph $H$ was planar, and all of these operations preserve planarity. However, the five vertices $z_0, z_1, z_2, z_3$, and $c$ are pairwise adjacent and form a $K_5$, which is not planar, a contradiction. ◀

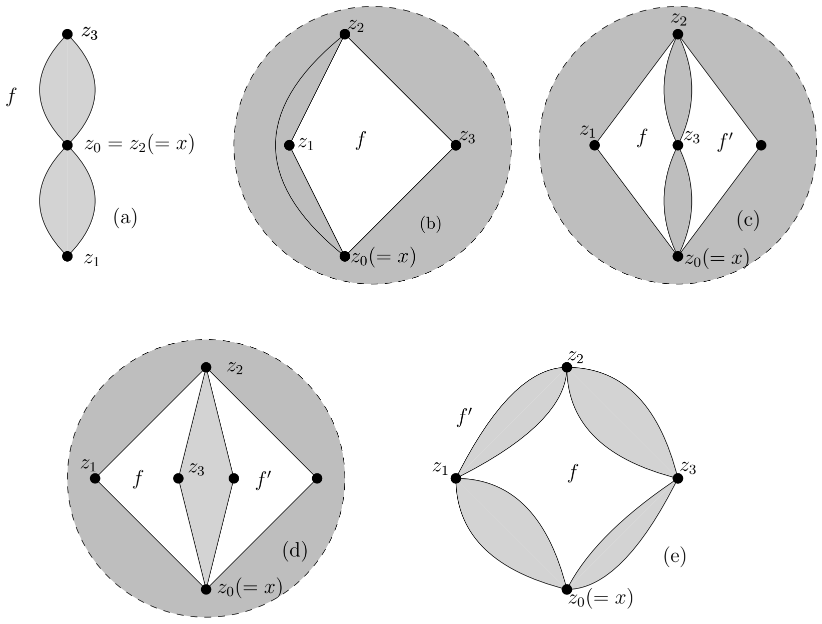

**Figure 1** Some configurations where we contract $z_1$ and $z_3$.

Now we reprove the existence of quadrangle-chords that form a forest.

▶ **Theorem 6.** *Let $H$ be a plane multigraph without loops such that every face has degree at most 4. Then it is possible to add a chord to every quadrangle of $H$ such that the graph induced by the chords is a forest. Moreover, for all pairs of adjacent vertices $a$ and $b$ of $H$, there is no path from $a$ to $b$ in the chords.*

**Proof.** As in Ackerman's proof, we prove the claim by induction on the number of quadrangles in $H$ and remove each quadrangle by contracting an opposing pair of vertices in the quadrangle. If $H$ has no quadrangles, the claim is trivial. Otherwise, there are quadrangles left. Pick an arbitrary vertex $x$ that is still incident to some quadrangles (later in our algorithm we will deal with all its incident quadrangles). Let $f$ be one of its incident quadrangles with facial cycle $\langle x = z_0, z_1, z_2, z_3 \rangle$. We first pick two opposing vertices of $f$ to contract.

**Case 1.** This case covers when we choose to contract $z_1$ and $z_3$, and has three sub-cases. We contract $z_1$ and $z_3$ whenever
**Case 1.a** $x = z_2$ are the same vertex, or
**Case 1.b** $x$ and $z_2$ are adjacent, or
**Case 1.c** $x$ and $z_2$ are opposing vertices of some quadrangle $f' \neq f$. [1]

Figure 1 illustrates possible configurations of face $f$ where Case 1 applies: Case 1.a applies to (a), Case 1.b applies to (b), and Case 1.c applies to (c,d,e). Note that Case 1.c covers Case 3 of Ackerman's proof, where $x, z_1, z_2, z_3$ all belong to two simple quadrangles $f, f'$ (see also Figure 1(e)). Our contraction turns $f'$ into a non-simple quadrangle, but our proof can handle this.

**Case 2.** Otherwise, we contract $x$ and $z_2$.

Table 1 demonstrates when we pick Case 1 and when we pick Case 2, and crucially shows cases which are impossible by Lemmas 4 and 5. To see that these lemmas apply in the second row and column, observe that adjacent vertices always share at least one face, and in particular if two opposing vertices are adjacent then they must share two faces other than the quadrangle they are opposing in.

**Table 1** All possible cases for the quadrangle $f$ with facial cycle $\langle x, z_1, z_2, z_3 \rangle$. We either indicate which case in the proof of Theorem 6 would be chosen, or indicate the lemma that demonstrates that this case is impossible.

| | $x = z_2$ | $x \neq z_2$; $(x, z_2) \in E(H)$ | $x \neq z_2$; $x$, $z_2$ are opposing in $f'$ | Otherwise |
|---|---|---|---|---|
| $z_1 = z_3$ | Impossible (Lemma 4) | Impossible (Lemma 4) | Impossible (Lemma 4) | Case 2 |
| $z_1 \neq z_3$; $(z_1, z_3) \in E(H)$ | Impossible (Lemma 4) | Impossible (Lemma 5) | Impossible (Lemma 5) | Case 2 |
| $z_1 \neq z_3$; $z_1$, $z_3$ are opposing in $f''$ | Impossible (Lemma 4) | Impossible (Lemma 5) | Impossible if $f' \neq f''$ (Lemma 5), Case 1.c otherwise | Case 2 |
| Otherwise | Case 1.a | Case 1.b | Case 1.c | Case 2 |

Let $z_i, z_{i+2}$ be two vertices chosen for contraction and let $H'$ be the graph resulting from contracting $z_i$ and $z_{i+2}$. By Table 1, $z_i$ and $z_{i+2}$ are not adjacent, and they are distinct. Therefore our contraction has destroyed the quadrangle $f$ and not added any loops, so we can apply induction on $H'$. Let $C'$ be the set of chords added to $H'$. By the inductive

[1] In fact, our proof does not require Case 1.c to be separated out; we could equally have contracted $x$ and $z_2$ in this case.

hypothesis, $C'$ induces a forest and for any edge $ab \in H'$, there is no path from $a$ to $b$ in $C'$. Uncontract $z_i$ and $z_{i+2}$, and add a chord $e := z_i z_{i+2}$ between them. Define $C := C' \cup \{e\}$. We now verify that $C$ satisfies all conditions.

Let $a, b$ be a pair of adjacent vertices of $H$. Assume by way of contradiction that there is a path from $a$ to $b$ in $C$. By the inductive hypothesis, the path must use $e$. Furthermore, $e$ cannot be the edge $ab$ since $z_i$ and $z_{i+2}$ are not adjacent, so the path must use some edges from $C'$. Let $c_1, \ldots, c_{k_1}, e, c_{k_1+1}, \ldots, c_{k_2}$ be the edges on this path, $k_2 \geq 1$. But then $c_1, \ldots, c_{k_2}$ would be a path from $a$ to $b$ within $C'$ in $H' = H/e$, a contradiction.

Assume by way of contradiction that $C$ induces a cycle in $H$. Since $e$ is not a loop in $H$, the cycle must use edges from $C'$. Let $e, c_1, \ldots, c_k$ be the cycle, $k \geq 2$. Then $c_1, \ldots, c_k$ would induce a cycle within $C'$ in $H' = H/e$, a contradiction. ◀

## 4 Efficient Implementation

It is still not immediately clear how one would implement the above theorem in order to achieve linear runtime, since as in Ackerman's proof we need to repeatedly test how many quadrangles two vertices share. However, if we are more careful about the order in which we contract each quadrangle, an efficient implementation can be achieved. The crucial idea will be to pick some vertex $x$ and contract all quadrangles incident to $x$. This will allow us to store additional information relative to $x$ and hence speed up testing which case applies. We note that this idea alone would not suffice to make Ackerman's proof run in linear time, as one would still need to find a way to implement Case 3 (where he splits the graph into four subgraphs and determines which subgraph contains a given pair of vertices) of Ackerman's proof efficiently.

### 4.1 Data Structure Interface

As mentioned earlier, one of the major ingredients to achieve fast run-time is to use the data structure by Holm et al. [14] for contraction in planar graphs, but we will also provide a (simpler but slower) alternative. We will discuss these later (in Subsection 4.4) when we analyze the run-time, but note here the two operations provided by [14] that will be needed:

- $x$ = `contract`$(e)$ takes a reference to an edge $e$, contracts $e$, and returns the vertex resulting from the contraction.

  Note that contracting $e$ creates a loop in the graph, especially if there are multiple copies of this edge, while the proof of Theorem 6 assumed that the graph has no loops. We could remove loops (the data structure by Holm et al. can report newly created loops after `contract`), but this turns out to be unnecessary: We will only contract edges at artificial gadgets inserted into the graph, and the created loops are at quadrangles that are destroyed afterwards and those will not pose problems.

- `neighbors`$(x)$ returns an iterator over $\{\langle xv, v \rangle : xv \in E\}$ where $E$ is the edge set of the graph. In other words, it returns an iterator to tuples containing each edge incident to $x$ and the endpoint of this edge. No guarantee is given as to the order of the neighbors.

  We assume that `neighbors`$(x)$ has $O(1)$ runtime and that the returned list can be iterated over in $O(d(x))$ time (recall that $d(x)$ denotes the number of edges incident to $x$). Since edge-contraction can create parallel edges, it is possible that `neighbors`$(x)$ contains parallel edges, and hence the second element of the tuple need not be unique. Again this will not pose problems later.

In Subsection 4.2, we will add labels and other meta-data to vertices of our graph. We make no assumptions as to how the meta-data are updated when two vertices are contracted, and so we will maintain those manually.

## 4.2 Preprocessing

We take as input a 1-planar graph $G$, given by specifying its planarization via the rotational clockwise order of edges at the vertices, and assuming that vertices of the planarization resulting from crossing points are marked as such. $G$ need not be simple, but we assume that it has no loops (they can always be added to the planar part) and for ease of stating bounds we assume that it has $O(n)$ edges. From $G$, we can construct a planar-maximal supergraph $G^+$ in linear time [2], and along the way construct the planarization $(G^+)^\times$ of $G^+$. As before we use $H$ to denote the skeleton of $G^+$, but we do not construct it explicitly. Instead, notice that the vertices of $(G^+)^\times$ marked as crossings correspond uniquely to the quadrangles of $H$. For this reason, we assume that these vertices are marked with a label *quad* and we call such a vertex a *quadrangle-vertex* (whereas the corresponding face of $H$ is called a *quadrangle-face*).

Our proof of Theorem 6 relies heavily on having faces, while the data structure of Holm et al. makes no provisions for accessing faces. For this reason, we keep the quadrangle-vertices in the graph as representatives of the quadrangle-faces. This will make it possible to implement the operation of "contract $z_i$ and $z_{i+2}$ within quadrangle $f$" used in Theorem 6 via edge-contractions at the corresponding quadrangle-vertex $f$. (See Procedure 1 for details.) We also assume that any quadrangle-vertex $f$ has references to the two original edges in $G$ that crossed; when doing such a contraction within $f$ we can hence also record the corresponding edge $z_iz_{i+2}$ for inclusion in the forest-part of the partition.

**Procedure 1** ContractThrough$(u, v, f)$.

```
Result: Contract two vertices $u$ and $v$ in $H$ through a quadrangle-face $f$, and return
        the resulting vertex $y$.
// pre: $f$ is labelled $quad$
// pre: $u$ and $v$ are opposing on face of $H$ corresponding to $f$
Find the original edge $uv$ of $G$ that is stored with $f$.
Record edge $uv$ as belonging to the forest of the partition.
for $\langle e, w\rangle$ in neighbors($f$) do
    if $v$ equals $w$ or $v$ equals $u$ or $w$ has label $quad_i$ for some $0 \le i \le 3$ then
        $y :=$ contract($e$)
Remove labels $quad, quad_i$ from $y$
return $y$
```

Our proof of Theorem 6 also requires knowing the order of vertices along a quadrangle, and the data structures do not support this directly. Therefore at any quadrangle-vertex $f$ we stellate each of the four incident triangular faces; see Figure 2. Since we have not yet contracted any edges, we have access to the rotational clockwise order of edges at $f$; we can hence label the added vertices with $quad_i$ (for $0 \le i \le 3$) in clockwise order around $f$. With this, we can retrieve the clockwise order $\langle z_0, \dots, z_3\rangle$ of vertices on the quadrangle-face corresponding to $f$ in constant time. (See Procedure 2 for details.)

We use $H^\diamond$ for the graph that results after all these modifications (it can be viewed as the planar skeleton $H$ with a "diamond"-gadget inserted into each quadrangle-face). We also add the following meta-data to each vertex of $H^\diamond$:

- A boolean `adj`, initialized to **false**.
- A boolean `in_worklist`, initialized to **false**.
- An integer `opposing`, initialized to 0.

**Procedure 2** FacialCycle($f$).

```
Result: Reconstruct the facial circuit of a quadrangle-face, given the corresponding
        quadrangle-vertex.
// pre: $f$ is a vertex of $H^{\diamond}$ with label $quad$
for vertex $v$ in neighbors($f$) do
    for $i = 0, \ldots, 3$ do
        if $v$ has label $quad_i$ then $f_i := v$
// By construction $f_i$ has neighbours $\{z_{i-1}, z_i, f\}$ (indices are mod 4)
for $i = 0, \ldots, 3$ do
    $N_i :=$ neighbors($f_i$) $\cap$ neighbors($f_{i+1}$)
    if $|N_i|$ equals 2 then $z_i := N_i \setminus \{f\}$
// If $f$ is not simple then $|N_i| = 3$ for two values of $i$, see Fig.2.
   But then $z_i$ is determined since we know $z_{i-1} = z_{i+1}$ already
for $i = 0, \ldots, 3$ do
    if $|N_i|$ equals 3 then $z_i := N_i \setminus N_{i+2}$
return $\langle z_0, z_1, z_2, z_3 \rangle$
```

The main idea of our algorithm is to iteratively contract all the quadrangles incident to some vertex $x$. As we do this, we will use `adj` to mark vertices that are adjacent to $x$, `in_worklist` to mark unprocessed quadrangles incident to $x$, and `opposing` to keep track of the number of quadrangle-faces where vertex $y$ is the opposing vertex of $x$.

Since $G$ has $O(n)$ edges, $H$ has $O(n)$ faces, so $H^{\diamond}$ has $O(n)$ edges. All steps in this preprocessing can hence be done in linear time.

## 4.3 Handling the Quadrangles around a Vertex

The main subroutine of our algorithm handles all quadrangles incident to some vertex $x$ by contracting each of them using the criteria laid out in Theorem 6 to decide which vertices to contract. To do so, we will initialize and maintain a work-list of faces incident to $x$ that we need to contract (using `in_worklist` to avoid putting duplicate quadrangles into the worklist). We also mark vertices in $H^{\diamond}$ as `opposing` and `adj` to $x$ as needed; this can be done in $O(d(x))$ time by retrieving all neighbours of $x$ via `neighbours`. (See Procedure 3 for details.)

Now we iteratively contract all the faces in the worklist according to Theorem 6; with `adj` and `opposing` we can determine the correct case in constant time. (See Procedure 4 for details.) In Case 1, we need to update some values of opposing: the vertex $z_2$ which was opposing $x$ now no longer opposes $x$ in $f$ (since $f$ was destroyed), so we decrement $z_2$.`opposing`. Likewise the new vertex $v$ resulting from the contraction will be opposing $x$ in all those quadrangles in which $z_1$ and $z_3$ previously opposed $x$, so we set $v$.`opposing` correspondingly. In Case 2, when we contract some vertex $z_2$ into $x$, we need to add the quadrangles incident to $z_2$ to our worklist, for which we can re-use Procedure 3.

Lastly, once our worklist is empty (and hence there are no more quadrangles incident to $x$), we reset the meta-data of $x$ and its neighbors, so that when repeating the procedure with a different vertex as $x$ there are no stray vertices with meta-data set to erroneous values. (See Procedure 5.)

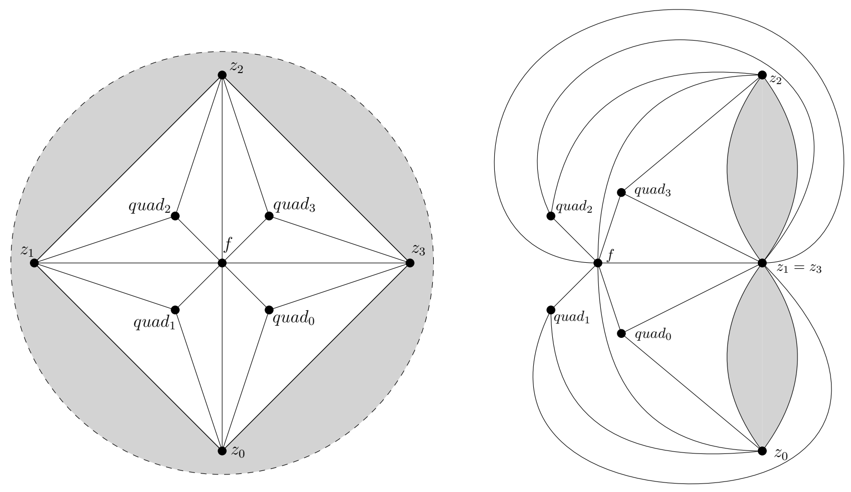

**Figure 2** The gadget added to every quadrangle of $H$, shown for both a quadrangle with four vertices on its boundary (left) and one with three vertices on its boundary (right).

**Procedure 3** InitializeAtOneVertex($y$, `worklist`).

```
Result: Adds all quadrangle-vertices incident to a vertex $y$ to a worklist, taking care
        not to put duplicates in the worklist.
// pre: $y$ is a vertex of $H$
// pre: $y$ equals $x$ (the vertex we currently work on), or $y$ will be
   contracted into $x$
for vertex $v \in$ neighbors($y$) do
    $v$.adj := true
    if $v$ has label quad and not $v$.in_worklist then
        $v$.in_worklist := true
        worklist.push($v$)
        $\langle z_0, z_1, z_2, z_3 \rangle$ := FacialCycle($v$)
        relabel $z_i$ such that $y$ equals $z_0$
        $z_2$.opposing += 1
```

## 4.4 Putting it All Together

The following summarizes our algorithm: after preprocessing, and for as long as there is a quadrangle-face $f$ left, process all quadrangles at a vertex $x$ on $f$ and record all edges that belong to the forest along the way. See Procedure 6 for a detailed description.

It remains to analyze the run-time. For now, we ignore the time required to perform the contractions and analyze the time for handling all quadrangles at one vertex $x$. Initialization takes $O(d(x))$ time, and most other steps take constant time per handled quadrangle, with one notable exception: When we contract some vertex $z_2$ into $x$, we must update the worklist, which takes time $O(d(z_2))$. Complicating matters further, $z_2$ may actually be the result of prior contractions, so its degree may be more than what it was in $H^\diamondsuit$, and we must ensure that degrees of vertices are not counted repeatedly.

To handle this, let $H^\diamondsuit_f$ be the graph that results from $H^\diamondsuit$ after *all* quadrangle-vertices have been contracted, and let $s(x)$ be the set of vertices that were contracted into $x$, either directly (when handling the quadrangles at $x$) or indirectly (i.e., if they had been contracted

**Procedure 4** HandleQuadsAtOneVertex($x$).

```
Result: Contract all the quadrangle-faces incident to a vertex $x$
worklist := []
InitializeAtOneVertex($x$, worklist)                              // see Proc. 3
for quadrangle-vertex $f$ in worklist do
    $\langle z_0, z_1, z_2, z_3 \rangle$ := FacialCycle($f$)      // see Proc. 2
    relabel $z_i$ such that $x$ equals $z_0$
    if $z_2$ equals $x$ or $z_2$.adj is true or $z_2$.opposing $\geq 2$ then    // Case 1
        opposing1 := $z_1$.opposing
        opposing3 := $z_3$.opposing
        $v$ := ContractThrough($z_1, z_3, f$)                     // see Proc. 1
        $v$.adj := true
        $v$.opposing := opposing1 + opposing3
        $z_2$.opposing −= 1
    else                                                          // Case 2
        InitializeAtOneVertex($z_2$, worklist)
        $x$ := ContractThrough($x, z_2, f$)
CleanupAtOneVertex($x$)                                           // see Proc. 5
```

**Procedure 5** CleanupAtOneVertex($y$).

```
Result: Cleanup the metadata at a vertex $y$ and its neighbors.
// pre: $y$ is a vertex of $H$
for vertex $v \in$ neighbors($y$) $\cup \{y\}$ do
    $v$.adj := false
    $v$.in_worklist := false
    $v$.opposing := 0
```

**Procedure 6** FindPGFPartition($G$).

```
Result: Find a PGF-partition of a graph $G$.
Add edges to make $G$ planar maximal 1-planar
Compute planarization $G^\times$, mark vertices of crossings with quad
foreach vertex $f$ marked quad do
    Insert four vertices in four incident faces of $f$
    Mark these vertices with $quad_0, \dots, quad_3$ according to embedding
while there remains a vertex $f$ labeled quad do
    $x$ := some neighbor of $f$ not labeled $quad_i$
    HandleQuadsAtOneVertex($x$)
Return all edges that were recorded as forest $F$ and $G \setminus F$ as planar graph
```

into one of the vertices $z_2$ that later get contracted into $x$). Crucially, note that if $x, x'$ are two vertices that are parameters during a call to Procedure 4 (i.e., we contract all quadrangles incident to the vertex), then $s(x)$ and $s(x')$ are disjoint. This holds because once we are done with the first of them (say $x$), all vertices in $s(x)$ have been combined with $x$ and no longer have any incident quadrangles. Since we only contract vertices into $x'$ that are incident to quadrangles, none of the vertices in $s(x)$ becomes part of $s(x')$.

Hence for each vertex $x$ of $H_f^\diamondsuit$, the amount of work done in Procedure 4 is proportional to the sum of the degrees $d_{H^\diamondsuit}(y)$ for each $y \in s(x)$. Since $H^\diamondsuit$ has $O(n)$ edges, and all other parts of the algorithm take constant time per quadrangle-vertex, the total amount of work done is at most

$$O\left(\sum_{x\in V\left(H_f^\diamondsuit\right)} \sum_{y\in s(x)} d_{H^\diamondsuit}(y)\right) = O\left(\sum_{x\in V(H^\diamondsuit)} d_{H^\diamondsuit}(x)\right) = O\left(\left|V\left(H^\diamondsuit\right)\right|\right) = O(|V(G)|)$$

hence the algorithm is linear (ignoring the time for contractions).

Now we consider the run-time of possible data structures for contractions. Our first approach is to represent the graph with incidence lists, where every vertex has a list of incident edges, each edge knows both of its endpoints, and every list knows its length. We can implement `neighbors`$(x)$ in constant time by simply returning an iterator to the incidence list at $x$. Contracting two vertices $u$ and $v$ can be done in $O(\min\{d(u), d(v)\})$ time by re-attaching the edges of the vertex with smaller degree to the vertex with larger degree. As with Union-Find data structures implemented with linked lists (see e.g. Section 4.6 of [17]), one shows that the amortized time for this is $O(\log n)$ per contraction. In particular, for graphs with linearly many edges, a set of $\Theta(n)$ contractions can be done in $O(n \log n)$ time. With this we have our first result.

▶ **Theorem 7.** *Let $G$ be a 1-planar graph implemented with incidence lists and given with a 1-planar embedding. It is possible to find a PGF-partition of $G$ in $O(n \log n)$ time.*

To improve this runtime, we appeal to the following result by Holm et al.

▶ **Theorem 8** (Holm et al. [14])**.** *Let $G$ be a planar graph with $n$ vertices and $m$ edges. Then there exists a data structure that supports* `contract` *and* `neighbors` *and that can be initialized in $O(n+m)$ time. Any calls to* `neighbors` *can be processed in worst case constant time, and any sequence of calls to* `contract` *can be performed in time $O(n+m)$.*

Since our graph has $O(n)$ edges, we have the main result of this paper.

▶ **Theorem 9.** *Let $G$ be a 1-planar graph with a given 1-planar embedding. Then in $O(n)$ time we can find an edge-partition of $G$ into a forest and a planar graph.*

## 5 Conclusion

In this paper, we reproved a result from Ackerman that all 1-planar graphs admit a partition into a planar graph and a forest. Our proof is more general than Ackerman's: the forest we find is guaranteed to not contain a path between adjacent vertices of the input graph. Using this proof and a data structure from Holm et al. for efficiently contracting the edges of a planar graph, we were able to find this partition in linear time. In consequence, a number of results for 1-planar graphs (such as splitting into 4 forests or 4-list-coloring if the graph is bipartite) can now be achieved in linear time. We also showed that the same algorithm can be implemented in $O(n \log n)$ time with a simpler data structure that uses only incidence lists.

As for open problems, the most interesting one is whether the partition could be found even *without* being given the 1-planar drawing. (Recall that it is NP-hard to find such a drawing [12], though it is polynomial for optimal 1-planar graphs [7].) All papers listed in the introduction for finding various edge partitions of 1-planar graphs require such an embedding. If this is difficult, could we at least do some of the implications (such as splitting into 4 forests or orienting such that all in-degrees are at most 4) in linear time without a given 1-planar drawing?

## References

1 Eyal Ackerman. A note on 1-planar graphs. *Discrete Appl. Math.*, 175:104–108, 2014. doi:10.1016/j.dam.2014.05.025.

2 Md. Jawaherul Alam, Franz J. Brandenburg, and Stephen G. Kobourov. Straight-line grid drawings of 3-connected 1-planar graphs. In Stephen Wismath and Alexander Wolff, editors, *Graph Drawing*, volume 8242 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 2013. doi:10.1007/978-3-319-03841-4_8.

3 Patrizio Angelini, Henry Förster, Michael Hoffmann, Michael Kaufmann, Stephen Kobourov, Giuseppe Liotta, and Maurizio Patrignani. The QuaSEFE problem. In *International Symposium on Graph Drawing and Network Visualization*, volume 11904 of *Lecture Notes in Computer Science*, pages 268–275. Springer, 2019. doi:10.1007/978-3-030-35802-0_21.

4 Michael A. Bekos, Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, Fabrizio Montecchiani, and Chrysanthi Raftopoulou. Edge partitions of optimal 2-plane and 3-plane graphs. *Discrete Mathematics*, 342(4):1038–1047, 2019. doi:10.1016/j.disc.2018.12.002.

5 Therese Biedl, Anna Lubiw, and Owen Merkel. List coloring bipartite graphs embedded on a surface. Unpublished Manuscript, 2019.

6 R. Bodendiek, H. Schumacher, and K. Wagner. Bemerkungen zu einem sechsfarbenproblem von g. ringel. *Abh. Math. Semin. Univ. Hambg.*, pages 41–52, 1983. doi:10.1007/BF02941309.

7 Franz J. Brandenburg. Recognizing optimal 1-planar graphs in linear time. *Algorithmica*, 80(1):1–28, 2018. doi:10.1007/s00453-016-0226-8.

8 Július Czap and Dávid Hudák. On drawings and decompositions of 1-planar graphs. *Electron. J. Combin.*, 20(2):Paper 54, 8, 2013. doi:10.37236/2392.

9 Emilio Di Giacomo, Walter Didimo, William S. Evans, Giuseppe Liotta, Henk Meijer, Fabrizio Montecchiani, and Stephen K. Wismath. New results on edge partitions of 1-plane graphs. *Theoretical Computer Science*, 713:78–84, 2018. doi:10.1016/j.tcs.2017.12.024.

10 Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, fifth edition, 2018. doi:10.1007/978-3-662-53622-3.

11 Harold N. Gabow and Herbert H. Westermann. Forests, frames, and games: algorithms for matroid sums and applications. *Algorithmica*, 7(5-6):465–497, 1992. doi:10.1007/BF01758774.

12 Alexander Grigoriev and Hans L. Bodlaender. Algorithms for graphs embeddable with few crossings per edge. *Algorithmica*, 49(1):1–11, 2007. doi:10.1007/s00453-007-0010-x.

13 Shai Gutner and Michael Tarsi. Some results on (a:b)-choosability. *Discrete Mathematics*, 309(8):2260–2270, 2009. doi:10.1016/j.disc.2008.04.061.

14 Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki Łącki, Eva Rotenberg, and Piotr Sankowski. Contracting a planar graph efficiently. In *25th European Symposium on Algorithms*, volume 87 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 50, 15. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2017.

15 Seok-Hee Hong and Takeshi Tokuyama. *Beyond Planar Graphs.* Springer Nature, Singapore, 2020.

16 Dmitri V. Karpov. An upper bound on the number of edges in an almost planar bipartite graph. *J. Math Sci.*, 196:737–746, 2014. doi:10.1007/s10958-014-1690-9.

17 Jon Kleinberg and Éva Tardos. *Algorithm Design.* Addison-Wesley Longman Publishing Co., Inc., USA, 2005.

**18** Stephen G. Kobourov, Giuseppe Liotta, and Fabrizio Montecchiani. An annotated bibliography on 1-planarity. *Comput. Sci. Rev.*, 25:49–67, 2017. doi:10.1016/j.cosrev.2017.06.002.

**19** William J. Lenhart, Giuseppe Liotta, and Fabrizio Montecchiani. On partitioning the edges of 1-plane graphs. *Theor. Comput. Sci.*, 662:59–65, 2017. doi:10.1016/j.tcs.2016.12.004.

**20** C. St. J. A. Nash-Williams. Decomposition of finite graphs into forests. *J. London Math. Soc.*, 39:12, 1964. doi:10.1112/jlms/s1-39.1.12.

**21** Gerhard Ringel. Ein Sechsfarbenproblem auf der Kugel. *Abh. Math. Sem. Univ. Hamburg*, 29:107–117, 1965. doi:10.1007/BF02996313.

**22** Gerhard Ringel. Two trees in maximal planar bipartite graphs. *J. Graph Theory*, 17(6):755–758, 1993. doi:10.1002/jgt.3190170610.

**23** Marcus Schaefer. The graph crossing number and its variants: A survey. *The Electronic Journal of Combinatorics [electronic only]*, 20, April 2013. doi:10.37236/2713.

**24** Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, pages 138–148, USA, 1990. Society for Industrial and Applied Mathematics.