Computing Planarity in Computable Planar Graphs

Oscar Levin and Taylor McMillan

2015

Abstract

We use methods from computability theory to answer questions about infinite planar graphs. A graph is *computable* if there is an algorithm which decides whether given vertices are adjacent. Having a procedure for deciding the edge set might not help compute other properties or features of the graph, however. The goal of this paper is to investigate the extent to which features related to the *planarity* of a graph might or might not be computable. We propose three definitions for what it might mean for a computable graph to be computably planar and for each build a computable planar graph which fails to be computably planar. We also consider these definitions in the context of *highly computable graphs*, those for which there is an algorithm which computes the degree of a given vertex.

1 Introduction

Many results in graph theory that are true of finite graphs can be extended to (countably) infinite graphs. For example, the four color theorem for finite graphs implies that every infinite planar graph is 4-colorable [4]. We can also extend the study of complexity to infinite graphs by using tools from computability theory (which until recently has been called *recursion* theory). For example, in the finite case we ask for the complexity of finding a proper vertex coloring by measuring the time or space required to produce the coloring. For infinite graphs, we measure the complexity in part by asking whether there even *exists* a computable coloring. An early result in this area is due to Bean: there is a computable planar graph which is not computably k-colorable for any natural number k [1]. Thus while the four color theorem extends to infinite graphs, it does not do so *effectively*.

Results of this flavor abound. It is possible to build infinite computable graphs which have Euler paths but no computable Euler paths [2], Hamilton paths but no computable Hamilton paths [7], domatic k-partitions but no computable domatic k-partitions [8], matchings but no computable matchings [11], edge colorings but no computable edge colorings [9], and so on (see [6] for a survey). In each of these examples, while the results deal with infinite graphs, there are implications to finite graph theory, especially if you consider the proofs of the results. To be able to construct a computable graph with property P but no computable P, you defeat a potential computation of P at some finite stage. This says that to be able to compute P, you must look at the entire graph. This is possible for finite graphs, but suggests high complexity for the calculation.

For the computability theory results about vertex coloring and Euler paths, the computable graphs constructed happen to be planar. In both cases, the way we know the graphs are planar is that from the construction it is clear how you could draw the graph in the plane. We wonder whether working with planar graphs in computability theory would always be this straight forward. In fact, could we get results like those mentioned above about the *planarity* itself of a graph? Before we can have any hope of achieving this, we must first decide what it means for the planarity of a graph to be computable.

In this paper we will consider three potential definitions for what it means for a graph to be *computably* planar. In each case, we will construct a graph which is planar but not computably planar. We then further refine our results by considering highly computable graphs and by investigating how noncomputable the planarity could be. After dealing with some preliminaries in Section 2, we start in Section 3 by considering computable planar embeddings. This is the most obvious way to discuss the planarity of a graph, but it is not straight forward to describe this effectively. When given a particular planar embedding, we can ask whether two vertices border the same face. We establish that this question cannot always be answered effectively in Section 4. An alternative relation on planar embeddings is to ask whether two adjacent edges are *radially* adjacent in the embedding, meaning you can rotate around their common vertex from one to the other without encountering another edge between them. We show that this too need not be computable for computable graphs in Section 5. We conclude with some ideas for further research in Section 6.

2 Preliminaries

Throughout this paper we will take a rather intuitive view of planar graphs and planar graph embeddings (or drawings). We will discuss *faces* as regions in the plane bounded by edges and vertices, and talk about regions *inside* or *outside* of cycles. For a more rigorous (topological) basis for these concepts, see Chapter 4 of [3].

There are a few additional details that should be considered when working with *infinite* planar graphs. Since we wish to consider computable graphs, all our graphs will be countable (in fact, we will use $\mathbb{N} = \{0, 1, 2, \ldots\}$ for the vertex set). As we do for finite graphs, we say that an infinite graph is *planar* if there is an embedding of the vertices into the real plane and the edges, represented as Jordan curves, never cross (that is, two edges intersect at no more than one point, the vertex to which they are both incident). To simplify matters, we will insist that our edges are actually straight lines. There is no loss of generality here as an infinite graph is planar if and only if there is a straight line representation (with a bounded vertex set, in fact). This is due to Thomassen [13], an extension of the finite Fáry's theorem [5]. In Section 4 we will consider *faces* of a planar graph. For infinite graphs, this can be tricky: there can be multiple unbounded faces, and of course faces bounded by infinitely many edges. In particular, it is not entirely clear how to define a face in the presence of an *accumulation point* of vertices (a point in the plane for which every open neighborhood contains vertices); could such a point separate two faces? Our approach is to avoid these messy situations as much as possible.

To discuss whether notions of planarity are effective, we must present our infinite graphs nicely. Intuitively, a graph should be *computable* provided there is some algorithm which decides if vertices are adjacent. We could certainly make this more formal by using Turing machines or another formal definition for computable functions, but by the Church-Turing thesis, all these models of computation are equivalent, so we will simply consider informal algorithms. The important piece is that our algorithms themselves are finite, and for specific inputs, when they halt they do so in finite time using finite space (although both of these can be unbounded as we range over all inputs). In general, an algorithm φ does not need to halt on all inputs, and so is called a *partial computable function*. If φ does halt on all inputs we say the function φ is *total computable* or simply *computable*. Technically our algorithms always accept a single input and give a single output (if they halt at all), but by using computable pairing functions we can also input or output finite tuples, as we need to for graphs. Given a pair of vertices, the computable function should output either 0 or 1 to specify whether the pair of vertices is adjacent.

There is an effective list of all partial computable functions (think of listing all algorithms in alphabetical order). We will denote this $\varphi_0, \varphi_1, \ldots$. We write $\varphi(x)\downarrow$ and say that φ *halts on input* x if the algorithm does produce an output for input x, otherwise we write $\varphi(x)\uparrow$. There is no way to know in advance whether a particular algorithm in this list will halt on a given input. This is the *halting set*: $K = \{e \in \mathbb{N} : \varphi_e(e)\downarrow\}$. This set is not computable (that is, the characteristic function for K is not a computable function), but is *computably enumerable*—there is an algorithm that lists out every element in K eventually (and never lists an element not in K). The halting set is the most complex c.e. set in that it can compute every other c.e. set (either by using K as an oracle or in a many-one reduction). For more details on these basics of computability theory, we recommend the standard [12], whose notation we follow.

In the upcoming sections we will prove that there are computable graphs which have properties that are not computable. There are essentially two ways to do this. First, we could *diagonalize* against all partial computable functions by finding one or more inputs for each φ_e that witness that φ_e is "wrong." The alternative approach is to *code in* K, so that if the property were computable, we would then be able to compute K, a contradiction. The latter method is stronger since this says the property is as complicated as the halting set (there are noncomputable functions where are strictly less complicated than K).

Regardless of which method we adopt, we will *construct* the computable graphs to have the required properties. We build the graphs by adding new vertices at each stage. Since the graphs need to be computable, once we enumerate a vertex we must immediately say to which previously enumerated vertices they are adjacent. This is also sufficient: the algorithm to decide whether two vertices are adjacent can simply run our construction until both vertices are mentioned, and at that point say whether they are adjacent. Notice that we do not need to specify *all* vertices adjacent to a vertex at this point, just those which we have already enumerated. This will be extremely helpful. If we use diagonalization, our constructions will build the graph while waiting for each φ_e to halt on some fixed set of vertices, which can happen arbitrarily late. At that point, we can add a new neighbor to one of the fixed "early" vertices to defeat φ_e. If we try to code in K, there too we wait to see a particular number enumerated into K, which can happen arbitrarily late.

It is interesting to consider what might happen if this mechanic was not permitted. To add such a restriction, we consider *highly computable* graphs. These are locally finite computable graphs which in addition have computable degree function. Note that from the degree of a vertex, we can effectively find a list of all adjacent vertices by searching: keep asking whether vertices are adjacent until you have found them all. When constructing highly computable graphs, we will need to specify all adjacent vertices as soon as a vertex is enumerated in the construction. The added information present in highly computable graphs often is enough to help compute properties of the graph. For example, while not all computable Eulerian graphs have a computable Euler path, each highly computable graph does [2].

3 Computable Planar Embeddings

The most straightforward way one might try to define "computably planar" would be to insist that the graph have a planar embedding that was itself computable. Since we insist on straight line embeddings (i.e., all edges are lines), all we need to do is say where the vertices are in the plane, and make sure that edges do not intersect. Note that given two line segments, it is computable to determine whether they intersect. We insist that vertices are mapped to rational points on the plane since those are easier to represent computably.

Definition 3.1. A graph G has a *computable planar embedding* provided there is a computable injective function $\varphi : G \to \mathbb{Q}\times\mathbb{Q}$ such that if $\{u_1, v_1\}, \{u_2, v_2\} \in E$ then the line segment from $\varphi(u_1)$ to $\varphi(v_1)$ does not intersect the line segment from $\varphi(u_2)$ to $\varphi(v_2)$.

Theorem 3.2. *There is a connected computable graph G which is planar but has no computable planar embedding.*

Proof. We will build the desired planar graph G in stages, while simultaneously diagonalizing against all φ_e, where φ_e is supposedly a computable planar embedding. The graph will consist of infinitely many copies of K_4 (the complete graph on four vertices), connected by a single edge from each K_4 to the next. Additionally, some pairs of K_4's will have one of each of their vertices connected to a "new" common vertex. A possible start of the graph is shown in Fig. 1.

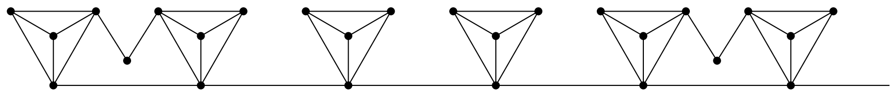

Figure 1: The possible start of G. The pairs connected to an additional vertex will be determined by the construction.

At each stage s we will have built a finite graph G_s, where $G_s \subseteq G_{s+1}$. Our final graph will be $G = \bigcup_{s\in\mathbb{N}} G_s$ which is an infinite planar graph. To diagonalize against each φ_e, we will run φ_e on two of the copies of K_4 and if φ_e embeds them in the plane in a planar way, we add a vertex adjacent to the "inside" vertices of each K_4, making φ_e's embedding non-planar.

To keep track of vertices, we use T_i for the ith copy of K_4, and call its vertices $t_i^0, t_i^1, t_i^2, t_i^3$. Whenever in the construction we "add vertices" we mean to take the least natural number not yet in G_s as the new vertex.

<u>Construction</u>: At stage 0, we add T_0 and T_1, each a complete graph on 4 vertices, as well as an edge from t_0^0 to t_1^0. Initially declare all φ_e as *not defeated*.

At stage $s \geq 1$, begin by adding T_{2s} and T_{2s+1} to the graph along with edges $\{t_{2s-1}^0, t_{2s}^0\}$ and $\{t_{2s}^0, t_{2s+1}^0\}$.

For each $e \leq s$ with φ_e still not defeated, run φ_e on the vertices of $T_{2e} \cup T_{2e+1}$ for s steps. If $\varphi_{e,s}(v)\downarrow$ for all $v \in T_{2e} \cup T_{2e+1}$, then

1. Find the vertex of T_{2e} inside the convex hull of T_{2e} and the vertex of T_{2e+1} inside the convex hull of T_{2e+1}. Call these vertices t_{2e}^{in} and t_{2e+1}^{in}. This can be done effectively, provided φ_e gives a planar embedding of T_{2e} and T_{2e+1}.
2. Unless no "inside" vertices were found, add a new vertex v to the graph and add edges $\{t_{2e}^{\text{in}}, v\}$ and $\{v, t_{2e+1}^{\text{in}}\}$. See Fig. 2.

For those e for which we completed the above, declare φ_e defeated.

This completes the construction.

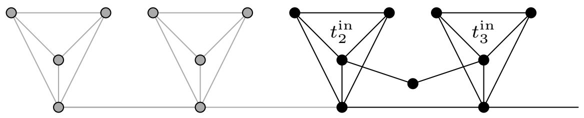

Figure 2: Example of G_s if $\varphi_1\downarrow$.

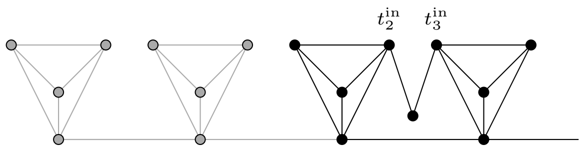

Figure 3: Note that G_s is still planar.

Verification: First, note that G is a planar graph. A (noncomputable) planar embedding consists of a graph much like that of Fig. 1, where the vertices found to be "inside" by any φ_e are placed "outside" as in Fig. 3. Also, G is clearly computable: to decide whether two vertices are adjacent, run the construction above until the end of a stage s where both have been mentioned. The vertices will be adjacent in G if and only if they are adjacent at stage $s + 1$.

Now for each e, the partial computable function φ_e is not a computable planar embedding. This could occur in one of two ways. Either φ_e is not the sort of partial computable function which claims to be a planar embedding because it does not converge on $T_{2e} \cup T_{2e+1}$ or does so but does not give outputs in $\mathbb{Q} \times \mathbb{Q}$ or does so but does not embed each of T_{2e} and T_{2e+1} as triangular convex hulls with a single vertex in the interior. In any of these cases, we never act to defeat φ_e, but we don't need to (we win for free). On the other hand, if φ_e *does* appear to be a planar embedding on $T_{2e} \cup T_{2e+1}$, we immediately add a new vertex to the graph. No matter how φ_e embeds this new vertex, the result will have an edge crossing, as the new vertex cannot be in the interior of *both* convex hulls. Either way, we see that φ_e is not a planar embedding.

This completes the verification and the proof. □

Notice that the way we diagonalized against φ_e was to add a new neighbor to a pair of vertices which were cut off from each other in the proposed planar embedding. This is acceptable for computable graphs, but would not be possible if the graph was highly computable. In that case, φ_e could wait to see all the neighbors of the vertices in question before it said how to embed them. This suggests that perhaps highly computable planar graphs might always have computable planar embeddings. This turns out to not be the case.

Theorem 3.3. *There exist a highly computable graph G that is planar but has no computable planar embedding.*

Proof. The graph we build will consist of infinitely many triangles (copies of K_3). In each triangle, one vertex will be adjacent to a single vertex of the next triangle (creating an infinite chain of triangles). Off of each of the other two vertices of each triangle we will build a (possibly infinite) path of vertices. See Fig. 4.

Each φ_e will be assigned two consecutive triangles in the chain. While we wait for φ_e to compute the positions in $\mathbb{Q} \times \mathbb{Q}$ of the 6 vertices in question, we

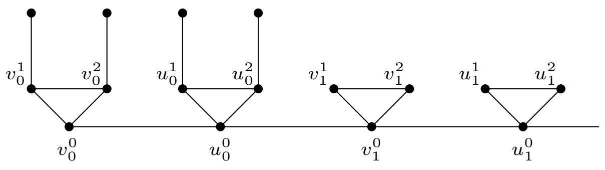

Figure 4: Possible start of the graph for Theorem 3.3.

continue to extend the four paths emanating from the triangles. Specifically, at each stage of the construction, if φ_e has not yet halted on its 6 inputs, add a vertex adjacent to the end of each of these paths. If φ_e never halts or does so but embeds the vertices in a way that creates an edge crossing (so is not a planar embedding), these paths will be extended forever. However, if φ_e halts and looks like it could be a planar embedding, we connect the four paths in such a way that no matter how φ_e would embed the vertices of the paths, there would always be at least one edge crossing.

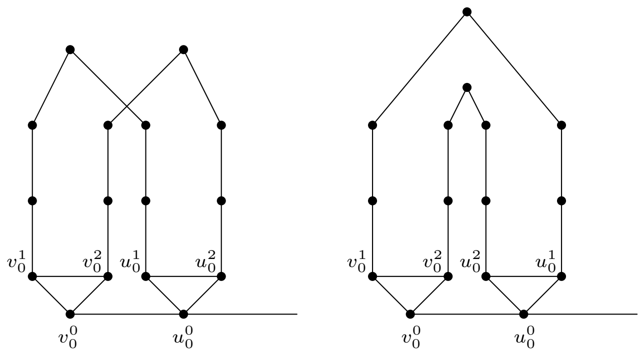

Figure 5: After φ_0 halts, we ensure that φ_0 is "wrong" (left) even though the graph is still planar (right).

To accomplish this last feat, notice that from an embedding of the triangles, we can determine which two paths will be on the "inside" and which will be on the "outside." Specifically, let the two triangles consist of v_0, v_1, v_2 and u_0, u_1, u_2 and assume v_0 is adjacent to u_0 (the connecting edge which is part of the chain of triangles). If we rotate around the first triangle clockwise, we might travel do so in the order v_0, v_1, v_2, or in the order v_0, v_2, v_1. The order depends on the embedding, but once we have the embedding, we can effectively decide which case we are in. Without loss of generality, assume the two triangles are embedded in the (clockwise) order v_0, v_1, v_2 and u_0, u_1, u_2. We say that the paths starting at v_2 and u_1 are the "inside" pair, while the other two paths are the "outside" pair. Once we have determined which pairs of paths φ_e has chosen to be inside and outside, we connect the outside path from the first triangle to the inside path from the second, and the inside path from the first triangle to the outside path of the second. This creates a crossing of the paths as in Fig. 5, which can only be avoided if φ_e had picked different paths to be inside and outside by embedding one of the triangles in the opposite order. Luckily for us, φ_e cannot change its mind.

The details of the construction are left to the reader, but we should point out that the graph we built really is planar (just orient the triangles "correctly") and in fact highly computable. This is because as soon as we enumerate a vertex into our graph in the construction, we say exactly what vertices previously mentioned it is adjacent to (so the graph is computable) and we know exactly what the degree of each vertex will be (vertices in the paths will always have degree 2, vertices in triangles will have degree 3 or 4 depending if they start an infinite path or are the vertices we use to connect all the triangles). The graph will not have a computable planar embedding, since no φ_e can correctly embed its six assigned vertices.

$\square$

4 Computable Facewise Adjacency

A planar embedding tells us exactly where to place vertices of the graph. Perhaps this is *too* precise. In attempting to effectively describe how we draw a graph, we really only care about the general location of vertices, relative to other vertices. We might want something along the lines of, "after drawing these three vertices as a triangle, you put the next vertex inside the triangle, and the one after that outside of the triangle." Such a description would ensure that these two newly placed vertices are in some sense far away from each other in the drawing, as they belong to different faces of some subgraph. The following is an attempt to describe this notion of "closeness."

For finite planar graphs, each planar embedding divides the plane into regions (called *faces*) separated by the edges and vertices. This leads to a (possibly) different method for describing a planar embedding of a graph: for each pair of vertices, we can say whether or not they are *facewise* adjacent in that embedding. Intuitively, facewise adjacent vertices are those which border the same face in the embedding. For infinite graphs we must be careful here, as faces might be bordered by infinitely many vertices, converging to an accumulation point. For example, consider the graph in Fig. 6.

To avoid these messy cases, we will say that two vertices are *facewise adjacent* provided they are not separated by a *finite* cycle. Now it becomes an easy matter to define the computable analog.

Definition 4.1. In a particular planar embedding of a graph G, vertices u and v are *facewise adjacent* provided there is no (finite) cycle for which u is on the "inside" and v is on the "outside" (or visa-versa).

Definition 4.2. A computable graph G has a *computable facewise adjacency*

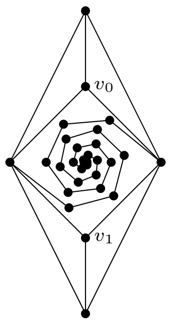

Figure 6: Is v_0 facewise adjacent to v_1 if there are two infinite paths spiraling to an accumulation point?

(relative to a particular planar embedding) provided there is a computable function φ such that $\varphi(u, v) = 1$ if u is facewise adjacent to v and 0 otherwise.

Note that pairs of vertices might be facewise adjacent in one planar embedding but not in another (which is what we want, since we are looking for ways to describe particular planar representations), although there are some graphs in which facewise adjacency is preserved in *any* planar representation. For a graph to have its facewise adjacency relation computable relative to a particular planar embedding, we require that the graph be computable but not that there exist a computable planar embedding *a priori* (it could turn out that we could compute the planar embedding from the facewise adjacency relation, but we leave this as an open question). It would be reasonable to assume that a noncomputable planar embedding would yield a noncomputable facewise adjacency. We can actually do a good deal better: there is a computable graph with computable planar embedding for which no facewise adjacency relation (relative to any planar embedding) is computable.

We could prove this result by again diagonalizing against all partial computable functions. Instead, for variety, we will show how to code the halting set K into the facewise adjacency relation. This will establish the stronger result that there is a computable graph for which every facewise adjacency relation computes the halting set. (This is the best result possible, since the halting set can check for the existance of a separating cycle, and thus decide whether given vertices are facewise adjacent.)

To accomplish this, we will have a particular subgraph assigned to each $n \in N$ to which we add vertices to make a pair no longer facewise adjacent precisely when $n \in K$. Before giving the proof, we describe the "gadgets" used in the construction and prove a purely graph-theoretic result about their planar embeddings.

Definition 4.3. Define D to be a copy of C_4 (a 4-cycle), with vertices v_0, v_1, v_2, v_3. We will define $\widehat{D} \supset D$ to be the result of adding the vertices v_4, v_5 and edges shown in Fig. 7.

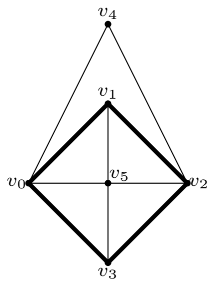

Figure 7: The resulting graph, $\widehat{D}$, after adding new vertices and edges.

Note that v_1 and v_3 are facewise adjacent in D (in every planar embedding), but not in $\widehat{D}$, at least not in the embedding drawn above. It turns out that v_1 and v_3 are not facewise adjacent in *any* planar embedding of $\widehat{D}$.

Lemma 4.4. *There is no embedding of $\widehat{D}$ such that v_1 is facewise adjacent to v_3.*

Proof. If v_1 and v_3 were facewise adjacent in a particular embedding of $\widehat{D}$, then we could add the edge $\{v_1, v_3\} \in E$ and still have a planar graph (this is a finite graph, so we need not worry about accumulation points). But this new graph would be a subdivision of K_5, so it would not be planar. Therefore, there is no planar embedding of $\widehat{D}$ such that v_1 is facewise adjacent to v_3. □

Theorem 4.5. *There is a computable planar graph for which the facewise adjacency relation in every planar embedding computes the halting set.*

Proof. Fix an effective enumeration K_s of the halting set K. While we run this enumeration, we build G as a sequence $D_0, D_1, \ldots$ of copies of the graph D from Definition 4.3, chained together with edges from v_2 of D_i to v_0 of D_{i+1}. Whenever we see a number n appear in K (that is, when $n \in K_s \setminus K_{s-1}$) we add vertices to the gadget D_n to form $\widehat{D}_n$.

This is enough. To decide whether $n \in K$, we simply ask whether the vertices v_1 and v_3 in D_n are facewise adjacent. If the answer is yes, we must not have added vertices to D_n, so $n \notin K$. On the other hand, if the answer is no, then we definitely did add the vertices, so $n \in K$. Therefore, if we are able to compute the halting set from a computable facewise adjacency then there is no computable planar graph G which has a computable facewise adjacency. □

Again, our proof relied on our ability to add edges to a vertex arbitrarily late. And again, it did not need to.

Theorem 4.6. *There is a highly computable graph for which the facewise adjacency relation in every planar embedding computes the halting set.*

Proof. The proof is almost identical to that of Theorem 4.5. However, now when initializing D_n we begin adding two paths off of each of the vertices v_0, and v_1 and also a path off of each of v_2 and v_3. At each stage for which n does not appear in K_s, add a new vertex to the end of each path. If n never enters K, we continue to build these paths forever. If n does enter K, we add vertices v_4 and v_5, as before, but now make them adjacent to the ends of the paths, creating a subdivision of $\widehat{D_n}$.

Note that if v_1 and v_3 are facewise adjacent (in any particular embedding) then we must not have connected the ends of the paths, so $n \notin K$. On the other hand, if v_1 and v_3 are not facewise adjacent (in a particular embedding) then we must have connected the paths, so $n \in K$. □

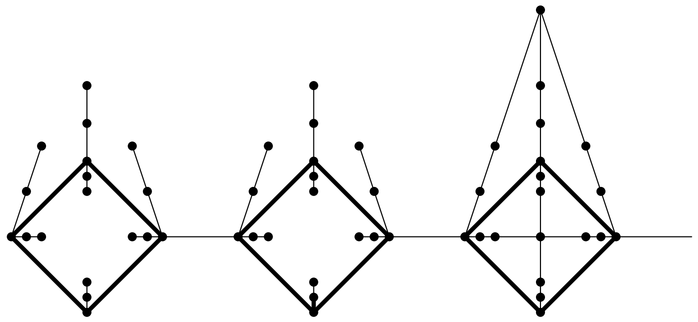

Figure 8: Possible start of G. In this case, we saw 2 enter K at stage 3 so we formed a subdivision of $\widehat{D_2}$, but 0 and 1 have not yet appeared in K.

5 Computable Radial Adjacency

Yet another approach to describing the planarity of a graph is to specify which edges radiate from a given vertex in which order. That is, if you have a planar embedding and look at a given edge e incident to a given vertex v, we should be able to say which edge e' incident to v is the next edge rotating around counterclockwise from e (an idea inspired by the method used by Lipton and Tarjan to represent a planar embedding in [10]). We will call such pairs e and e' *radially adjacent.* For computable graphs, we can ask whether a radial adjacency relation is computable.

Definition 5.1. In a particular planar embedding of a graph G, adjacent edges $\{u, v\}$ and $\{u, w\}$ are *radially adjacent* provided when you rotate counterclockwise from $\{u, v\}$, centered at u, the first edge you encounter is $\{u, w\}$.

Definition 5.2. A computable graph has a *computable radial adjacency* (relative to a particular planar embedding) provided there is a computable function

φ, such that $\varphi(e_1, e_2) = 1$ if e_1 and e_2 are edges and are radially adjacent and 0 otherwise.

Note that the computable function φ accepts *edges*, which means it actually accepts a 4-tuple of vertices, and interprets the first two as an edge, and the last two as an edge, then checks whether these really are edges, checks whether they have a vertex in common, and then checks whether they are radially adjacent. As with facewise adjacency, whether edges are radially adjacent can depend on the planar embedding. Still, there are computable planar graphs for which all radial adjacency relations (relative to all planar embeddings) are noncomputable.

Theorem 5.3. *There is a computable planar graph for which radial adjacency relative to any planar embedding computes the halting set.*

The proof will be similar to that of Theorem 4.5, but using different gadgets.

Definition 5.4. Define the graph H where $v_0, v_1, v_2 \in V$ and $\{v_0, v_1\}, \{v_0, v_2\} \in E$. Let $\widehat{H}$ be defined as $H \subset \widehat{H}$ where we add the vertices v_3, v_4, v_5 and edges as in Fig. 9.

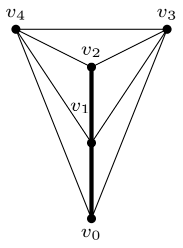

Figure 9: The graph $\widehat{H}$.

Lemma 5.5. *The edges $\{v_1, v_0\}$ and $\{v_1, v_2\}$ in $\widehat{H}$ are not radially adjacent in any planar embedding.*

Proof. Suppose there were a planar embedding of $\widehat{H}$ in which $\{v_1, v_0\} \in E$ and $\{v_1, v_2\} \in E$ were radially adjacent. Then we can add the edge $\{v_0, v_2\}$ and not intersect any other edge of $\widehat{H}$. But this would give a copy of K_5, a contradiction. Therefore, there is no planar embedding of $\widehat{H}$ where the edges $\{v_1, v_0\}$ and $\{v_1, v_2\}$ radially adjacent. □

Now we can prove the computability result.

Proof of Theorem 5.3. We build G while enumerating K. The graph will consist of a sequence $H_0, H_1, \ldots$ of copies of the graph H from Definition 5.4, chained together with edges along the base (as in Fig. 10). At each stage, we extend this sequence and wait for a number n to enter K. For this n, we add vertices to H_n to form $\widehat{H}_n$. This completes the construction.

To determine if $n \in K$, we simply ask whether the two edges $\{v_0, v_1\}$ and $\{v_1, v_2\}$ in H_n are radial adjacent. If the answer is yes, then we must not have added vertices to H_n to form $\widehat{H}_n$, so $n \notin K$. On the other hand, if the answer is no then we must have added the vertices to H_n, so $n \in K$. □

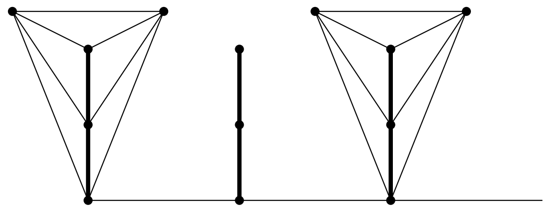
Figure 10: Possible start of G. In this case, we have seen 0 and 2 enter K, but not 1 (yet).

The next natural question to ask is whether there exists a highly computable planar graph with noncomputable radial adjacency. This sort of extension worked for computable planar embeddings and for facewise adjacency, but it is not nearly as straight forward for radial adjacency. The way we forced two edges to not be radially adjacent is by adding a new edge to their common vertex. This can happen arbitrarily late in the construction, so there is no hope to replicate this for highly computable graphs. In the previous results, we overcame this by adding ever-lengthening paths instead of single edges, and only connecting them (arbitrarily late) when needed. Here, as soon as we add an edge (even if it is the beginning of a path) we have created a graph which has at least one planar embedding for which the target edges are no longer radially adjacent.

In particular, suppose the computable graph also has a computable planar embedding. This is not enough to say that radially adjacency is computable in general, but if the graph is also highly computable, then it is. Given two edges e and e' with common vertex u, we find all the neighbors of u (using the computable degree function) and then see if any of them occur between e and e' (using the computable planar embedding).

If the graph does not have any computable planar embeddings, then we can also ensure that the radial adjacency relation is noncomputable. To do this we would replicate the construction of Theorem 3.3, but instead of waiting for φ_e to embed vertices, we wait for φ_e to say which of the base edges are radially adjacent. After φ_e has committed, we link the paths to ensure φ_e is wrong. This construction is not enough to code in the halting set however, and we leave that possibility as an open question.

6 Conclusion

We have given three potential definitions for what it might mean for a computable graph to be *computably planar*. These are motivated by what we "should" be able to do with a nicely presented planar graph. We should be able to say where vertices should be embedding in the plane so that the resulting graph embedding has no edges crossing. It should also be possible to decide whether two vertices border the same face in a particular (or any) planar embedding. It also should be possible to decide the order in which edges radiate from a vertex in a particular (or any) planar embedding. We called these *computable planar embeddings*, *computable facewise adjacency*, and *computable radial adjacency* respectively. In each case, there are computable graphs that are planar but not computably planar.

We leave open the question of which of these definitions should be adopted as *the* definition. Whatever the choice, the results about the other two notions would be theorems. For example, if we decide that a computable planar graph is computably planar provided there is a computable planar embedding, then we have the theorem, "There is a computably planar graph with noncomputable facewise adjacency relation" from Theorem 4.5.

This raises the more general question of how our three notions of computable planarity relate to each other. For any pair, we can ask whether one implies the other. For example, it might be that every planar graph with computable radial adjacency also has a computable planar embedding. That is, perhaps knowing a particular radial adjacency relation would actually tell us how to embed the graph in the plane without crossings. Of the 6 possible implications, we have negative answers to three:

Corollary 6.1 (to Theorem 4.5)**.** *There is a computable graph G with computable planar embedding which does not have computable facewise adjacency. Thus computable planar embeddings do not help compute facewise adjacency.*

Corollary 6.2 (to Theorem 5.3)**.** *There is a computable graph G with computable planar embedding which does not have computable radial adjacency. Thus computable planar embeddings do not help compute radial adjacency.*

Corollary 6.3. *There is a computable graph G with computable radial adjacency which does not have computable facewise adjacency. Thus computable radial adjacency does not help compute facewise adjacency.*

Proof. The graph constructed in the proof of Theorem 4.6 has computable radial adjacency. □

Question 6.4. Do any of the other three implications hold? That is, if a graph has a computable facewise adjacency, does it have computable radial adjacency or a computable embedding, and if a graph has computable radial adjacency, does it have a computable embedding?

Another direction for future study is to consider just how noncomputable planar embeddings, facewise adjacency and radial adjacency can be. By Theorem 4.5, we know there are computable graphs with facewise adjacency at least as complicated as the halting set. Could they have even higher complexity? In this case it is fairly easy to see the answer is no, as long as we consider a particular planar embedding.

Theorem 6.5. *Let G be a computable planar graph with computable planar embedding φ. Then K can compute the facewise adjacency in G and K can compute the radial adjacency in G.*

Proof. To decide whether vertices v and u are facewise adjacent, we simply ask whether there is a finite cycle which separates v and u. K can answer this existential question, so can compute facewise adjacency.

Similarly, K can compute the radial adjacency of any graph with particular computable planar embedding, since K can compute the degree of a vertex, and from this (and a computable planar embedding) we can compute radial adjacency. □

The main open question in this direction is what the complexity of a planar embedding might be.

Question 6.6. Is there a computable planar graph for which every planar embedding has complexity strictly larger than K? Alternatively, does every computable planar graph have a planar embedding a complexity strictly less than K?

Finally, we think another interesting direction would be to investigate how these notions of planarity relate to other questions in graph theory. Bean's result [1] that there is a computable planar graph with infinite computable chromatic number essentially produces a planar embedding (that is how we know the constructed graph is planar). However, it is not immediately clear whether the graph produced has computable facewise adjacency or radial adjacency.

Question 6.7. Is there a computable planar graph with computable facewise (radial) adjacency with computable chromatic number greater than 4?

References

[1] Dwight R. Bean, *Effective coloration*, J. Symbolic Logic **41** (1976), no. 2, 469–480. MR 0416889 (54 #4952)

[2] ———, *Recursive Euler and Hamilton paths*, Proc. Amer. Math. Soc. **55** (1976), no. 2, 385–394. MR 0416888 (54 #4951)

[3] Reinhard Diestel, *Graph theory*, fourth ed., Graduate Texts in Mathematics, vol. 173, Springer, Heidelberg, 2010. MR 2744811 (2011m:05002)

[4] P. Erdös, *Some remarks on set theory*, Proc. Amer. Math. Soc. **1** (1950), 127–141. MR 0035809 (12,14c)

[5] István Fáry, *On straight line representation of planar graphs*, Acta Univ. Szeged. Sect. Sci. Math. **11** (1948), 229–233. MR 0026311 (10,136f)

[6] W. Gasarch, *A survey of recursive combinatorics*, Handbook of recursive mathematics, Vol. 2, Stud. Logic Found. Math., vol. 139, North-Holland, Amsterdam, 1998, pp. 1041–1176. MR 1673598 (2000f:03127)

[7] David Harel, *Hamiltonian paths in infinite graphs*, Israel J. Math. **76** (1991), no. 3, 317–336. MR 1177348 (93d:68023)

[8] Matthew Jura, Oscar Levin, and Tyler Markkanen, *Domatic partitions of computable graphs*, Arch. Math. Logic **53** (2014), no. 1-2, 137–155. MR 3151402

[9] Henry A. Kierstead, *Recursive colorings of highly recursive graphs*, Canad. J. Math. **33** (1981), no. 6, 1279–1290. MR 645224 (84b:05045)

[10] Richard J. Lipton and Robert Endre Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math. **36** (1979), no. 2, 177–189. MR 524495 (80k:68050)

[11] Alfred B. Manaster and Joseph G. Rosenstein, *Effective matchmaking (recursion theoretic aspects of a theorem of Philip Hall)*, Proc. London Math. Soc. (3) **25** (1972), 615–654. MR 0314610 (47 #3161)

[12] Robert I. Soare, *Recursively enumerable sets and degrees*, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1987, A study of computable functions and computably generated sets. MR 882921 (88m:03003)

[13] Carsten Thomassen, *Straight line representations of infinite planar graphs*, J. London Math. Soc. (2) **16** (1977), no. 3, 411–423. MR 479743 (80i:05039)