A-Computable Graphs

Matthew Jura[a], Oscar Levin[b,*], Tyler Markkanen[c]

[a]*Department of Mathematics, Manhattan College*
4513 Manhattan College Parkway
Riverdale, NY 10471, USA
[b]*School of Mathematical Sciences, University of Northern Colorado*
Greeley, CO 80639, USA
[c]*Department of Mathematics, Physics, and Computer Science, Springfield College*
263 Alden Street
Springfield, MA 01109, USA

Abstract

We consider locally finite graphs with vertex set $\mathbb{N}$. A graph G is *computable* if the edge set is computable and *highly computable* if the neighborhood function N_G (which given v outputs all of its adjacent vertices) is computable. Let $\chi(G)$ be the chromatic number of G and $\chi^c(G)$ be the computable chromatic number of G. Bean showed there is a computable graph G with $\chi(G) = 3$ and $\chi^c(G) = \infty$, but if G is highly computable then $\chi^c(G) \leq 2\chi(G)$.

In a computable graph the neighborhood function is Δ^0_2. In highly computable graphs it is computable. It is natural to ask what happens between these extremes. A computable graph G is *A-computable* if $N_G \leq_T A$. Gasarch and Lee showed that if A is c.e. and not computable then there exists an A-computable graph G such that $\chi(G) = 2$ but $\chi^c(G) = \infty$. Hence for A noncomputable and c.e., A-computable graphs behave more like computable graphs than highly computable graphs. We prove analogous results for Euler paths and domatic partitions. Gasarch and Lee left open what happens for other Δ^0_2 sets A. We show that there exists an $\emptyset <_T A <_T \emptyset'$ such that every A-computable graph G with $\chi(G) < \infty$ has $\chi^c(G) < \infty$. Finally, we classify all such A.

Keywords: computability theory, Euler paths, chromatic number, highly computable graphs
2010 MSC: 03D25, 03D28, 03D45

1. Introduction

When applying notions in computability theory to results in graph theory, we often find that classical results are not effectively true. Two early examples of this phenomenon are due to Bean: there is a computable graph with an Euler path but no computable Euler path [2], and there is a computable graph with chromatic number 3 but no computable finite coloring [1] (in fact there is a computable graph with chromatic number 2 with no computable finite coloring if you don't require the graph to be connected [13]). Results of this flavor abound including ones for Hamilton paths [5], perfect matchings [8], edge colorings [7], and domatic partitions [6].

Some of these results (including the original two of Bean) rely heavily on our ability to add neighbors to a given vertex in the construction of the graphs. A *computable graph* is simply a graph for which the edge relation is computable (here and throughout the paper we assume without loss of generality that computable graphs have vertex set $\mathbb{N}$). Thus there is an effective procedure to decide whether two given vertices are in fact adjacent. However, we cannot in general produce the list of all vertices adjacent to a given vertex (or equivalently, determine the degree or valency of a given vertex). Those computable graphs which have a computable neighborhood function, which, when given a vertex, outputs the list of all vertices adjacent to it (i.e., its neighbors), are called *highly computable*. For this definition to make sense, all vertices must have only finitely many neighbors (such graphs are called *locally finite*), and in this paper we will only consider such graphs.

Whenever a construction of a graph requires adding neighbors to vertices arbitrarily late in the construction, we wonder whether this requirement is necessary. Often it is, in that the result that holds for computable graphs fails to hold for highly computable graphs. Indeed, every highly computable graph with chromatic number n has a computable $(2n-1)$-coloring (proved independently by Schmerl [13] and Carstens and Päppinghaus [3]; Schmerl

*Corresponding author

Email addresses: matthew.jura@manhattan.edu (Matthew Jura), oscar.levin@unco.edu (Oscar Levin), tmarkkanen@springfieldcollege.edu (Tyler Markkanen)

proved this bound is tight). Similarly, every highly computable graph containing an Euler path has a computable Euler path [2].

The goal of this paper is to better understand this behavior by investigating graphs that are *between* computable and highly computable. We adopt the approach suggested by Gasarch and Lee [4] and consider *A-computable graphs*[1] for various Δ^0_2 sets A, meaning A computes the neighborhood function. We denote the neighborhood function of G by N_G. Specifically, given a vertex v of G, $N_G(v)$ returns the canonical index for the finite set of vertices that are adjacent to v in G.

Definition 1.1 (Gasarch and Lee)**.** Let A be a set. A locally finite graph $G = (V, E)$ is *A-computable* provided G is computable and $N_G \leq_T A$.

Note that computable graphs always have a neighborhood function computable from the halting problem K, so they are K-computable, while highly computable graphs have computable neighborhood function, making them $\emptyset$-computable.

Since A-computable graphs might be somewhere between computable and highly computable, it is reasonable to wonder whether the complexity of graph-theoretic properties of A-computable graphs might be between those of computable and highly computable graphs. Gasarch and Lee considered this question for vertex colorings and found that at least for noncomputable c.e. sets A, the A-computable graphs behave just like the computable ones. The following is the main result of their paper.

Theorem 1.2 (Gasarch and Lee [4])**.** *Let A be a noncomputable c.e. set. There exists an A-computable graph G such that G is 2-colorable but not computably k-colorable for any natural number k.*

The authors employed the technique of c.e.-permitting in their construction. They asked:

Question 1.3. Can Theorem 1.2 be extended to any set A with $\emptyset <_T A <_T \emptyset'$?

They remarked that the construction would be more difficult without permitting, and suggested that it might be easier first to consider the case when

[1] Gasarch and Lee used the term *A-recursive* to denote the same concept.

A is 2-c.e. Indeed, there is a version of permitting with Δ^0_2 sets, aptly called Δ^0_2-permitting. (See [9] for a nice exposition of this method.) Unfortunately, the method of Δ^0_2-permitting does not seem to apply when constructing an A-computable graph. The reason is that, while the neighborhood function of an A-computable graph is necessarily Δ^0_2, we cannot remove an edge from an A-computable graph once one has been enumerated; after all, the graph (and therefore the edge relation) needs to be computable. One would need to be able to "undo" any change made to the neighborhood of any given vertex in the graph in order to be able to use Δ^0_2-permitting, because the set A might give permission to make a change to the neighborhood function, and then later withdraw that permission because an element leaves A in the computable approximation being used (which is what would require us to remove edges).

Our main result is to answer Question 1.3 in the negative. In fact, we show that there exists a set $\emptyset <_T A \leq_T \emptyset'$ such that every A-computable graph is highly computable. We define such sets to be *low for graph neighborhood*. Further we show that for any Δ^0_2 set A, either the set is low for graph neighborhood or there is an A-computable graph which is 2-colorable but not computably k-colorable for any k. Thus, so far as vertex coloring goes, A-computable graphs are never strictly between computable and highly computable.

Before building our set A, we start in Section 2 by applying the c.e.-permitting argument of Gasarch and Lee to the question of computable Euler paths and domatic partitions. Then in Section 3 we construct a Δ^0_2 set A for which every A-computable graph with finite chromatic number has finite computable chromatic number. We say how to modify our construction to build sets which are low for graph neighborhood allowing our result to extend to the functions on graphs considered in Section 2. Finally, in Section 4 we establish that there are no Δ^0_2 sets A for which A-computable graphs act any differently than either computable or highly computable graphs, at least with respect to the graph functions considered.

2. *A*-Computable Graphs and C.E.-Permitting

To familiarize ourselves with A-computable graphs, we present a modification of the c.e.-permitting construction used by Gasarch and Lee, applied to Euler paths instead of coloring. Recall an Euler path is simply a sequence of vertices $v_0, v_1, \ldots$ such that for all $i \in \mathbb{N}$, $\{v_i, v_{i+1}\}$ is an edge in the

graph and every edge in the graph appears (as consecutive vertices in the sequence) exactly once.[2] A *computable* Euler path is a computable function f satisfying $f(n) = v_n$ for all $n \in \mathbb{N}$. In [2], Bean proved that there is a computable graph with an Euler path but no computable Euler path, and that every highly computable graph with an Euler path has a computable Euler path. As in the case for coloring, if A is a noncomputable c.e. set, then A-computable graphs behave as computable graphs do.

Theorem 2.1. *For any noncomputable c.e. set A, there exists an A-computable graph which has a one-way Euler path but no computable one-way Euler path.*

Proof. Fix a computable enumeration A_s of A with $A_{s+1} \setminus A_s = \{a_s\}$. We build G in stages; at stage s we will have $G_s = (V_s, E_s)$ and then will possibly add some vertices and edges to form $G_{s+1} \supseteq G_s$. We will have $G = \bigcup_s G_s$ with $V = \bigcup_s V_s = \mathbb{N}$. By the construction, G will be A-computable. When complete, G will consist of an infinite one-way path made up of the even vertices, together with "triangles" coming off of some of the even vertices consisting of the odd vertices. See Figure 1 for one possible start of the graph.

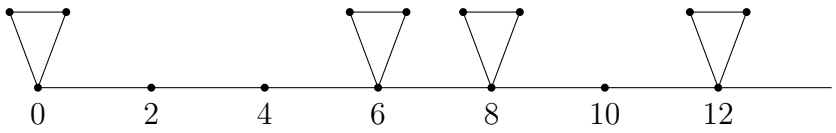

Figure 1: A possible beginning of the graph G. The unlabeled vertices of triangles are each some odd number.

We diagonalize against each φ_e by satisfying for all e the requirement

$\mathcal{R}_e$: φ_e is not a one-way Euler path.

To accomplish this, partition the even natural numbers into infinitely many infinite sets $T_0, T_1, \ldots$. Until the requirement $\mathcal{R}_e$ is *satisfied*, it will work on the set T_e of even numbers by looking for the vertex enumerated by φ_e directly after v is enumerated for the first time, for each $v \in T_e$.

[2]When we speak of Euler paths, we mean one-way Euler paths. Results for two-way Euler paths can be found analogously

Specifically, we wait until a stage s by which we find the least k_v such that $\varphi_e(k_v) = v$ (by searching) *and* $\varphi_e(k_v + 1) = v + 2$. If this ever happens, we say that $\mathcal{R}_e$ *needs attention on* v *at stage* s.[3] If in addition $a_s \leq v$, we say that $\mathcal{R}_e$ *deserves attention on* v *at stage* s. The idea is that A grants permission to give attention to $\mathcal{R}_e$ by enumerating a small element (relative to v) into A.

Construction: At stage $s = 0$, define G_0 by setting $V_0 = \{0\}$ and $E_0 = \emptyset$. Wait for a_0 to appear in A.

At stage $s > 0$, begin by setting $V_s' = V_{s-1} \cup \{2s\}$ and $E_s' = E_s \cup \{\{2s - 2, 2s\}\}$. Wait for a_s to appear in A. At the same time, run the procedure outlined above to decide if any unsatisfied $\mathcal{R}_e$ (with $e \leq s$) needs attention at stage s. If any do, decide whether the vertex on which they need attention is greater than a_s. That is, decide whether any of the requirements that need attention actually *deserve* attention. For each requirement $\mathcal{R}_e$ that needs and deserves attention on (say) v_e at stage s, set $V_{s+1} = V_s' \cup \{i_e, j_e\}$ and $E_{s+1} = E_s \cup \{\{v_e, i_e\}, \{i_e, j_e\}, \{j_e, v_e\}\}$, where the i_e, j_e are the least odd numbers not already in V_s. Declare all $\mathcal{R}_e$ for which this process was completed as *satisfied*.

This ends the construction.

Verification: By the construction, G is computable and has an Euler path. We must check that G is in fact A-computable and that each $\mathcal{R}_e$ is satisfied.

Here is how A can compute N_G. We need only concern ourselves with even vertices, as the neighbors of odd vertices are determined completely when they first appear in the construction (so $N_G(v)$ is computable for all odd v). To find the neighbors of v for v even, we run the enumeration of A until a stage s for which $A_s \upharpoonright v = A \upharpoonright v$. Now run the construction of G through stage s. We have $N_{G_s}(v) = N_G(v)$, since the only way to add a new neighbor of v is for a requirement to deserve attention on v at some stage t. But after stage s this can never happen, because nothing enters A below v after stage s. Therefore G is A-computable.

Finally we argue that each $\mathcal{R}_e$ is satisfied. (Some will be declared satisfied during the construction, others in the limit because φ_e is not total or doesn't even enumerate a path, etc.) Suppose this was not the case. Then for some e, each $v \in T_e$ is enumerated by φ_e for the first time immediately before

[3] At this point in Bean's original construction, we would enumerate the triangle adjacent to v thus satisfying $\mathcal{R}_e$.

it enumerates $v+2$. As soon as we see φ_e enumerate $v+2$ after v, the requirement $\mathcal{R}_e$ needs attention. However, we will never add the "triangle" at v, since φ_e is an Euler path by assumption. This means that $\mathcal{R}_e$ will never deserve attention on v at a later stage. That is, nothing below v will enter A at a later stage. But this means we can compute A: to decide whether $n \in A$, we run the construction above until a stage s at which $\mathcal{R}_e$ requires attention on some $v > n$ with $v \in T_e$. At this stage we know that $A_s \upharpoonright v = A \upharpoonright v$ (else at a later stage $\mathcal{R}_e$ would deserve attention on v), so we simply check whether $n \in A_s$. This is a contradiction, as we assumed that A is noncomputable.

This completes the verification and the proof. □

Using a similar c.e.-permitting technique (and a construction closer to that used by Gasarch and Lee [4] to prove Theorem 1.2), we can prove an analogous result for domatic partitions. A *domatic k-partition* of a graph G is a partition of the vertices into k *dominating* sets, where a set is dominating if every vertex not in the set is adjacent to a vertex in the set. For each $k \geq 3$ there is a computable graph with a domatic k-partition but no computable domatic 3-partition [6]. This results extends to A-computable graphs.

Theorem 2.2. *Let $k \geq 3$. For any noncomputable c.e. set A, there is an A-computable graph $G = (V, E)$ such that G has a domatic k-partition, but no computable domatic 3-partition.*

Proof sketch. The construction for computable graphs diagonalizes against all φ_e using a single copy of $K_{3(k-2)+1}$ (the complete graph on $3(k-2)+1$ vertices) targeted for each φ_e. We wait for φ_e to provide a domatic 3-partition on its gadget, at which point we find at least $k-1$ vertices partitioned identically (by the pigeon hole principle) and add a new vertex adjacent to all of these. This allows for the augmented gadget to still have a domatic k-partition, but makes φ_e's partition incorrect.

For the A-computable case being considered here, we use the same diagonalization gadgets, but this time have infinitely many gadgets targeted for each φ_e. We run φ_e on all of its gadgets, but only act (add a vertex) when A gives us permission by enumerating an element into A smaller than the largest vertex in the gadget.

By waiting for A to enumerate a small (relative to the vertices in the gadget) element, we ensure that the graph is A-computable: A tells us when it is done enumerating below the vertices of a gadget, at which point we know the vertices will gain no new neighbors.

By giving φ_e infinitely many opportunities to diagonalize, we ensure that at least one gadget will accomplish this task. For if none did, we could compute A (a contradiction) by waiting for φ_e to require attention at larger and larger vertices. □

In the introduction, we mentioned that there are computable graphs with Hamilton paths but no computable Hamilton path. Not surprisingly, this extends to A-computable graphs as above.

Proposition 2.3. *For any noncomputable c.e. set A, there is an A-computable graph G such that G has a one-way Hamilton path, but no computable one-way Hamilton path.*

We omit the proof, as it is so similar to the proofs above. In fact, this result is trivial since there exist *highly* computable graphs with Hamilton paths but no computable Hamilton path [2] and every highly computable graph is A-computable for all Δ^0_2 sets A. However, the construction for highly computable graphs is considerably more complicated than the construction for computable graphs. It is possible to take the simpler graph-theoretic mechanism used to diagonalize in the computable graph case and apply it to the A-computable case within the c.e.-permitting framework.

There is no reason to think the same basic argument wouldn't work for other graph-theoretic properties. The key step seems to be the ability to replicate the diagonalization infinitely often, so that instead of finding one witness for each φ_e, we can potentially find infinitely many (by using the fact that the set is unbounded). This allows for the contradiction argument in the verification that if there were some φ_e that did the right thing, we could compute A. A subtle difference between the two constructions outlined above is worth pointing out. For Euler paths, the vertices of the graph were the range of φ_e, while for domatic partitions, the vertices were the *domain*. The condition by which A grants permission to defeat φ_e is always that a small element enters A relative to the *vertices*, be they in the domain or range. This is necessary for the sake of ensuring that the graph is actually A-computable.

All of this is to say that it appears we get results of the form, "For any noncomputable c.e. set A, there is an A-computable graph G such that G has a _____ but no computable _____," no matter what graph-theoretic relation we put in both blanks. All that is required is that the corresponding result for computable graphs holds. So we end this section with a somewhat abstract and, admittedly, informal conjecture.

Conjecture 2.4. *Let P be the set of properties of a given kind of graph-theoretic relation. For any relation R, we say that R is a* P-relation *if R satisfies all the properties in P. If R is also computable, we call it a* computable P-relation*. If there exists a computable graph that has a P-relation but no computable P-relation, then for every noncomputable c.e. set A, there is an A-computable graph that has a P-relation but no computable P-relation.*

We leave as an open problem how to make this conjecture rigorous, and to prove it is true.

3. Constructing a Counterexample

We now return to Question 1.3. In order to answer this question in the negative, we build a noncomputable Δ^0_2 set A with the property that any A-computable graph that is n-colorable has finite computable chromatic number. In our construction, we will use a coloring procedure similar to the one used in the following theorem due independently to Schmerl [13] and Carstens and Päppinghaus [3].

Theorem 3.1. *If G is highly computable and n-colorable, then G is computably $(2n-1)$-colorable.*

A proof of the above theorem uses a coloring procedure that alternates between two sets of colors: $\{1,\ldots,n-1\}$ and $\{n+1,\ldots,2n-1\}$. The color n is then used as a kind of buffer between the two sets. In our construction, it simplifies matters to dispense with the buffer and merely alternate between two sets of colors that are side by side. This increases the computable chromatic number χ^c by 1 (i.e., from $2n-1$ to $2n$), but since there will be other parts of the construction that increase χ^c, potentially beyond this, we might as well accept this loss of efficiency. Indeed, in our construction there will be no bound on the finite number of colors needed for the computable coloring, but for each graph with finite chromatic number, the computable chromatic number will be finite.

Theorem 3.2. *There exists a noncomputable Δ^0_2 set A such that every A-computable graph with finite chromatic number has finite computable chromatic number.*

Before beginning the proof, here is the main idea of the construction. As we build A, we use our approximation as an oracle to compute the neighborhood function for the graph we attempt to color. If this oracle computation

is always "correct," then we are essentially coloring a highly computable graph and can do so as in the proof of Theorem 3.1. Of course, since A will change (as we diagonalize to ensure A is not computable), there will be instances where we find a vertex adjacent to earlier vertices we have already colored. If this happens only finitely often, we can just use a finite number of new colors to take care of these vertices (but this might add an arbitrarily large finite number of colors). Alternatively, we realize that these new vertices are only problematic because A previously claimed that they were not neighbors of earlier vertices. If we can fix A in this previous "incorrect" state, then we will no longer need to produce a computable coloring, because the graph will no longer be A-computable. Our proof only involves finite injury. After all higher priority requirements have acted for the last time, we will either change A back once more and be done (since the graph will not be A-computable) or never change A and actually produce an acceptable coloring.

In the proof, we will use two sorts of colorings. The *alternating coloring* alternatively uses the colors $\{1, \ldots, n\}$ and $\{n+1, \ldots, 2n\}$ to color finite subgraphs (which is possible by searching, if the graph has chromatic number n). This will be used as long as A is "correctly" computing the neighborhood function. When we find A in error, we use an *online coloring*, which simply assigns to a vertex the least color greater than $2n$ not among the vertex's already colored neighbors.

Proof of Theorem 3.2. We build the set A while diagonalizing against all partial computable functions φ_e to ensure that A is not computable, but build A so that it will be Δ^0_2 (by ensuring that A is limit computable). At the same time, for each partial computable function ψ_i, if it looks like $G = \psi_i$ is an n-colorable computable graph, and for each Turing functional Φ^A_e, if it looks like Φ^A_e gives the neighborhood function on G, we attempt to give a computable finite proper coloring of the graph, or to ensure that A is such that Φ^A_e is wrong about the neighbors of at least one vertex of G.

Construction. We satisfy the following requirements, for all e and i:

$\mathcal{P}_e$: $A \neq \varphi_e$

$\mathcal{R}_{\langle e,i,n\rangle}$: If $G = \psi_i$ is an n-colorable A-computable graph with neighborhood function Φ^A_e, then there is a computable proper finite coloring Δ of G.

Effectively arrange the requirements in a priority listing $\mathcal{P}_0 \prec \mathcal{R}_0 \prec \mathcal{P}_1 \prec \cdots$, alternating between the two types. (This gives the priority ordering type

ω.) The lower the requirement is in the ordering, the higher its priority.

Strategy for $\mathcal{P}_e$. Pick a large (above any restraint on A) unused witness x, targeted for A, and wait for $\varphi_e(x)\downarrow = 0$. If this ever occurs, enumerate x into A, place restraint on A up to x and cease work on the strategy, unless the requirement is *injured*. The requirement can be injured by higher priority requirements in two ways. Either a higher priority $\mathcal{R}_j$ removes x after A has enumerated it into A or, at the point that $\varphi_e(x)\downarrow$, we discover that a higher priority requirement has already placed restraint on A above x. In either case, we restart the strategy for $\mathcal{P}_e$ using a new large unused witness.

Strategy for $\mathcal{R}_{\langle e,i,n\rangle}$. We attempt to satisfy $\mathcal{R}_{\langle e,i,n\rangle}$ either by producing a finite coloring or by changing A so that ψ_i is not A-computable via Φ_e^A. We describe the strategy in terms of stages, but note that each of these stages might span several stages in the overall construction of A. In what follows, we will use t to denote the stage of the strategy, and s to denote the stage of the overall construction.

Define V_t to be the set of vertices mentioned by the end of stage $t-1$ in the strategy. Initially $V_0 = \emptyset$. To ensure that all vertices are eventually investigated, we insist that $\{0,\ldots,t\} \subseteq V_t$ (by adding t to V_t at the start of stage t if necessary). We also define for each $v \in V_t$ the set

$$N_{t,s}(v) = \{u \in V_t \cup \Phi_e^{A_s}(v) : \{u,v\} \in \psi_i\}$$

as long as $\Phi_e^{A_s}(v)\downarrow$ and $\psi_i(u,v)\downarrow$ for all u (otherwise leave the set undefined). In other words, $N_{t,s}(v)$ is the approximation of the set of neighbors of v at stage t of the strategy and stage s of the construction. We include $u \in \Phi_e^{A_s}(v)$ since these vertices are not necessarily in V_t (although they will be in V_{t+1}) but are being reported as neighbors by the approximation of A. Note that $N_{t,s}(v) \subseteq N_{\psi_i}(v)$ (the true neighbors of v in the graph ψ_i) but we might have $N_{t,s}(v) \neq \Phi_e^{A_s}(v)$.

At stage $t \geq 0$ of the strategy, we begin by adding t to V_t (if not already present) and then proceed to compute $N_{t,s}(v)$ for each $v \in V_t$. This computation might span multiple values of s, and we continue until some stage s' at which $N_{t,s'}(v)$ is computed for all $v \in V_t$. Let U_t be the set of vertices in $V_t \cup \bigcup_{v\in V_t} N_{t,s'}(v)$ which are currently uncolored by Δ. There are two cases.

1. It is possible to apply the alternating coloring to properly color the vertices of U_t. Specifically we can use either colors $\{1,\ldots,n\}$ or colors $\{n+1,\ldots 2n\}$ (whichever set we did not use the last stage we were in this case) to color all vertices in U_t so that $V_t \cup U_t$ is properly colored

(with no adjacent vertices colored identically). In this case, apply the alternating coloring. Set $V_{t+1} = V_t \cup U_t$ and move to stage $t+1$.

2. It is not possible to apply the alternating coloring. This might be because we have found a finite subgraph which is not n-colorable, in which case our requirement is automatically satisfied. Otherwise, this can only be because one or more vertices in U_t are adjacent to colored vertices in V_t, but previously were not believed to be neighbors of these colored vertices. So search for a vertex $v \in V_t$ and stages $t_0 < t$ and $s_0 < s'$ such that $N_{t_0,s_0}(v) \neq N_{t,s'}(v)$, and for which it is possible to *rewind* $A_{s'}$ to A_{s_0} without violating the restraint of any higher priority requirement. If this search is successful, set $A_{s'+1} = A_{s_0}$ (which might add or remove elements from A) and place restraint on A up to the use of $\Phi_e^{A_{s_0}}(v)$. Now the premise of the requirement is false, so cease work on the strategy (unless the requirement is later injured). On the other hand, if no v, t_0, s_0 are found, then color the vertices of U_t using the online coloring, set $V_{t+1} = V_t \cup U_t$, and move to stage $t+1$.

The requirement can be *injured* if a higher priority requirement changes A below the restraint placed by $\mathcal{R}_{\langle e,i,n\rangle}$ (which only happens if we thought we had met the requirement by making the graph not A-computable). If this happens, we simply restart the strategy where we left off.

Another way other requirements can interact with the strategy for $\mathcal{R}_{\langle e,i,n\rangle}$ is to change A while we are running the computation of $N_{t,s}(v)$ (since we have no guarantee that $N_{t,s}(v)$ will exist; $\Phi_e^{A_s}(v)$ or $\psi_i(u,v)$ might take a long time to converge, or might not at all). If this change in A occurs below the use of $\Phi_e^{A_s}(v)$ (while waiting for ψ_i to halt) or before $\Phi_e^{A_s}(v)$ has halted, we say that $\mathcal{R}_{\langle e,i,n\rangle}$ has been *disturbed* and just restart the computation of $N_{t,s}(v)$.

Verification. We will argue that each requirement is injured only finitely often and as such is satisfied. We must also verify that A is Δ_2^0. To this end, suppose for contradiction that some $x \in \mathbb{N}$ enters and leaves A infinitely often. Let $\mathcal{P}_e$ be the requirement that first puts x into A (this is the only way an element can enter A for the first time). Only the requirements $\mathcal{R}_j$ of priority higher than $\mathcal{P}_e$ can remove x or put x back in from now on, and every time this happens, the strategy which changes the status of x puts restraint on A above x. So we conclude that there must be some requirement of priority higher than $\mathcal{P}_e$ that issues restraint infinitely often. Let $\mathcal{Q}$ be the least such requirement. Consider the last stage at which any requirement higher than $\mathcal{Q}$ issues restraint. The next stage at which $\mathcal{Q}$ issues restraint

will satisfy $\mathcal{Q}$ (because the only time either kind of strategy issues restraint is when the requirement is, until injured, satisfied), so in fact $\mathcal{Q}$ will never issue restraint again, a contradiction. Thus A is limit computable and as such Δ^0_2.

We show that for all e, i, and n, requirements $\mathcal{P}_e$ and $\mathcal{R}_{\langle e,i,n\rangle}$ are satisfied. Inductively assume all requirements of priority higher than $\mathcal{Q}$ are satisfied. We consider the two possibilities.

Case 1. $\mathcal{Q} = \mathcal{P}_e$ for some e. We will have $\mathcal{P}_e$ automatically satisfied if φ_e is not total or not a characteristic function, so assume $\varphi_e(x)\downarrow \in \{0,1\}$ for all $x \in \mathbb{N}$. Since no satisfied requirement can issue restraint infinitely often (as both kinds of requirements cease computation as soon as they issue restraint and only continue computation if injured), there will be some finite stage $\hat{s}$ at which the strategy for $\mathcal{P}_e$ picks its last witness, call it x. At some later stage, $\varphi_e(x)\downarrow$. If $\varphi_e(x) = 0$, then we put $x \in A$ and restrain A up to x. No higher priority requirement ever changes A again (since they are done issuing restraint), so $\mathcal{P}_e$ will be and always remain satisfied. If $\varphi_e(x) = 1$, then we do not put $x \in A$. While we do not issue restraint in this case, we do not need to, as only the requirements $\mathcal{P}_i$ can put elements into A for the first time. Note that in either case, $\mathcal{P}_e$ only changes A finitely often (and as such, only injures other requirements finitely often).

Case 2. $\mathcal{Q} = \mathcal{R}_{\langle e,i,n\rangle}$ for some e, i, and n. Assume that ψ_i is a computable, locally finite graph with chromatic number n and whose neighborhood function is Φ^A_e. Let $\hat{s}$ be a stage at which all higher priority requirements have issued restraint for the last time. Say that we are at stage $\hat{t}$ of the strategy at this point in the construction. At this stage we might still need to use the online coloring, but starting with stage $\hat{t}+1$, we will always be able to use the alternating coloring. This is because if we ever needed to use the online coloring, we would first try to rewind A to a previous state that did not correctly compute the neighborhood function on some vertex. If $A_{\hat{s}}$ or any future version of A are incorrect about the neighbors of any of the vertices they are asked about, we would rewind back to that incorrect version of A and Φ^A_e would not be the neighborhood function. But assuming it is, we see that after stage $\hat{s}$ we will only use the alternating coloring. Since by then we have used the online coloring only finitely often, on only finitely many vertices, we will have a coloring in finitely many colors.

All that remains is to verify that we are able to complete each stage t of the strategy. This is not obvious because to get to the part of stage t where we color vertices, we must first compute $N_{t,s'}(v)$ for all $v \in V_t$, and

requirements of any priority (higher or lower) can disturb this process. To compute $N_{t,s}(v)$ we must first find $\Phi_e^{A_s}(v)$, and then check a bunch of edges using ψ_i. Eventually, there will be a stage at which restraint is placed on A above the use of $\Phi_e^A(v)$. (Note the oracle here is A, not A_s.) So $\mathcal{R}_{\langle e,i,n\rangle}$ will not have been disturbed at such a stage, and we will be able to carry out the computation of $N_{t,s}(v)$. □

As a brief analysis of the construction in the above proof, notice that there were two major features of the given A-computable graph ψ_i at play: ψ_i's neighborhood function and the coloring of ψ_i's vertices. If we disregard one of these features, namely, the coloring, and only worry about the neighborhood function, then we obtain a stronger result as follows.

Theorem 3.3. *There exists a noncomputable Δ_2^0 set A such that every A-computable graph is highly computable.*

Proof. The construction for the proof of this theorem is a modification of the construction for the proof of Theorem 3.2. The following shows how to modify the strategy used to satisfy each $R_{\langle e,i,n\rangle}$. Begin the strategy the same way, but instead of defining Δ to be the finite computable coloring, we now define it to be the neighborhood function of ψ_i (we simply ignore the n). However, this time we completely redefine Δ each time $R_{\langle e,i,n\rangle}$ is injured. If ψ_i is an A-computable graph via Φ_e^A, then there will be some finite stage at which we redefine Δ for the last time. Therefore Δ is total and yields the neighborhood function of ψ_i (in our new construction). □

Observe that the above theorem allows us to specify the computable chromatic number of the A-computable graphs mentioned in Theorem 3.2. Specifically, given n, there is a noncomputable Δ_2^0 set A such that every A-computable graph that is n-colorable is computably $(2n-1)$-colorable (by Theorems 3.3 and 3.1). However, we lose the uniformity of the coloring in Theorem 3.2.

The sets like those constructed above will be of interest in the next section so let's give them a special name.

Definition 3.4. We say that a set A is *low for graph neighborhood* if every A-computable graph is actually highly computable (that is, if G is a computable graph such that $N_G \leq_T A$, then N_G is computable).

From Theorem 3.3, we immediately get:

Corollary 3.5. *There exists a noncomputable Δ^0_2 set which is low for graph neighborhood.*

On the other hand, from Theorems 1.2 and 3.1, we have:

Corollary 3.6. *No noncomputable c.e. set is low for graph neighborhood.*

4. Classifying Sets Low for Graph Neighborhood

We have an explicit construction of a noncomputable Δ^0_2 set A that prohibits Theorem 1.2 from being extended to Δ^0_2 sets in general. However, by using a known result from computability theory (namely, the existence of minimal Δ^0_2 sets), we can actually give a purely existential argument of such a set A. To do this, we begin with the following observation about A-computable graphs for any Δ^0_2 set A.

Lemma 4.1. *For any Δ^0_2 set A and any A-computable graph G, there is a c.e. set B such that $B \leq_T A$ and G is B-computable.*

Proof. Let A be a Δ^0_2 set, and let G be an A-computable graph. Fix a computable approximation A_s of the set A. By the recursion theorem, we may recursively know an index for the Turing reduction that witnesses G being A-computable, so let's say that the reduction is Ψ_i (and so for all x, $\Psi_i^A(x)\downarrow = N_G(x)$). We also know an index for the graph, say $G = \varphi_e$.

The c.e. set B will have elements of the form $\langle v, s\rangle$, which will code the stage s by which $\Psi_i^A(v)$ converges to its limit. For a fixed vertex v in the graph G, we do the following (dovetailing this process through all vertices of G).

Construction. At stage s in the construction of B, we compute A_s, and run $\Psi_{i,s}^{A_s}(v)$. If $\Psi_{i,s}^{A_s}(v)\uparrow$, then we do nothing and go to stage $s+1$.

If $\Psi_{i,s}^{A_s}(v)\downarrow$, then we check whether the value of $\Psi_{i,s}^{A_s}(v)\downarrow$ is consistent with G by running φ_e on all pairs of the form (v, x), where x is no more than the largest vertex mentioned so far in the construction, to see if the vertices that $\Psi_{i,s}^{A_s}(v)$ claims are neighbors of v are actually neighbors of v. Note that φ_e will converge on all such pairs since G is computable. If it turns out that $\Psi_{i,s}^{A_s}(v)$ is incorrect, or appears to be correct but had previously given the same (apparently correct) approximation, then do nothing and go to stage $s+1$. But if $\Psi_{i,s}^{A_s}(v)$ is consistent with G as defined above and we see that all previous approximations are incorrect, then we enumerate each number $\langle v, r\rangle$, for $r \leq s$, (not already in B) into B, and go to stage $s+1$.

This ends the construction.

Verification. Note that B is c.e. by the construction.

We first verify that the graph G is B-computable. Indeed, B will compute the neighborhood of v in G by finding the first $\langle v, s\rangle$ not in B, computing A_s and $\Psi_{i,s}^{A_s}(v)$, which will be the true neighborhood of v. For if $\Psi_{i,s}^{A_s}(v) \neq N_G(v)$, then at stage $s-1$ of our construction of B, it appeared that $\Psi_{i,s-1}^{A_{s-1}}(v)$ was correct but all previous approximations were wrong (and actually were wrong, according to G). But since $\Psi_i^A(v) = N_G(v)$, there must be a later stage $t > s$ at which $\Psi_{i,t}^{A_t}(v) = N_G(v)$, and at the first such stage we would enumerate $\langle v, r\rangle$ into B for all $r \leq t$, contradicting our assumption that $\langle v, s\rangle$ was not in B.

Next, we claim $B \leq_T A$. To decide whether $\langle v, s\rangle \in B$, run $\Psi_{i,t}^{A_t}(v)$ for all $t < s$ and compare this with $\Psi_i^A(v)$ (which must converge by assumption). If $\Psi_{i,t}^{A_t}(v) = \Psi_i^A(v)$ for some $t < s$, then we will never put any $\langle v, r\rangle$ into B for $r > t$, so we know that $\langle v, s\rangle \notin B$. If $\Psi_{i,t}^{A_t}(v) \neq \Psi_i^A(v)$ for any $t < s$, then $\langle v, s\rangle \in B$ because there will be some later stage of our construction at which our approximation does converge (correctly for the first time) and at that stage we will put $\langle v, s\rangle$ into B.

This completes the verification and the proof. □

Using this lemma, we can classify which sets are low for graph neighborhood and summarize the results of this paper.

Theorem 4.2. *Let A be a noncomputable Δ_2^0 set. The following are equivalent.*

1. *A is low for graph neighborhood.*
2. *Every c.e. set $B \leq_T A$ is computable.*
3. *Every A-computable graph with finite chromatic number has finite computable chromatic number.*
4. *Every A-computable graph with an Euler path has a computable Euler path.*

Proof. If A is low for graph neighborhood, then every A-computable graph is actually highly computable. Using the constructions in [13] and [2] we see that both $(1 \to 3)$ and $(1 \to 4)$, respectively.

$(1 \to 2)$ Suppose A is low for graph neighborhood, and let $B \leq_T A$ be c.e. Every B-computable graph is also A-computable (because if B computes

N_G then so does A), and as such highly computable, so B is low for graph neighborhood as well. Thus B is computable by Corollary 3.6.

$(2 \to 1)$ Suppose A is not low for graph neighborhood. Then there is an A-computable graph G such that G is not highly computable. By Lemma 4.1, there is a c.e. set B such that G is B-computable. Since G is not highly computable, we must have that B is noncomputable.

$(3 \to 2)$ Suppose (2) fails. Then there exists a noncomputable c.e. set $B \leq_T A$. So, by Theorem 1.2, there exists a B-computable graph G that has finite chromatic number but does not have finite computable chromatic number. Since G is also A-computable, (3) fails.

$(4 \to 2)$ The same proof works by replacing Theorem 1.2 with Theorem 2.1. □

Notice that this theorem gives us a shortcut answer to Question 1.3. There are minimal Δ^0_2 sets [11], which in particular don't compute any noncomputable c.e. sets. All minimal Δ^0_2 sets are therefore low for graph neighborhood. However, low for graph neighborhood is not equivalent to minimal Δ^0_2. The class of 1-generic sets also never compute noncomputable c.e. sets (see exercises VI.3.8 in [14]), so they too (those that are Δ^0_2) are low for graph neighborhood. Note also that low for graph neighborhood is not equivalent to a set being low (in the sense that its jump is equivalent to the halting problem). There are low sets which are low for graph neighborhood (since there are low minimal degrees [15]), low sets which are not low for graph neighborhood (since there are low c.e. sets), and non-low sets low for graph neighborhood (since there are non-low minimal Δ^0_2 degrees [12]).

5. Conclusion

We have shown how the c.e.-permitting technique used in [4] can be applied to other areas where it might be interesting to investigate A-computable graphs, such as Euler paths or domatic partitions (Gasarch and Lee investigated the graph colorings in A-computable graphs). In the case of Euler paths, there is always an algorithm to find an Euler path in a highly computable graph which has an Euler path, but a computable counterexample exists for computable graphs which have Euler paths (analogously to the case of graph colorings). When A is a noncomputable c.e. set, we have shown that A-computable graphs behave the same as computable graphs for both Euler paths and domatic partitions. It appears that the technique is general

enough to apply to a whole host of graph-theoretic properties, although we leave the statement and proof of this general principle as an open problem.

In [4], Gasarch and Lee asked whether their main result about graph colorings in A-computable graphs (with A noncomputable c.e.) can be extended to any noncomputable Δ^0_2 set A. We have answered this question in the negative. First, we directly constructed such a set A in Theorem 3.2. Then in Lemma 4.1, we showed that for any noncomputable Δ^0_2 set A and any A-computable graph G, there is a c.e. set $B \leq_T A$ such that G is also B-computable. So if A is a noncomputable Δ^0_2 set that does not compute any noncomputable c.e. set, then A also yields a negative answer to the question of Gasarch and Lee. We have called such sets *low for graph neighborhood*, which we have classified in Theorem 4.2. We know there are many examples of sets which are low for graph neighborhood, since neither minimal Δ^0_2 sets nor 1-generics can compute any noncomputable c.e. sets.

Finally, an idea to extend this work. Nies and others have investigated various lowness notions (especially with respect to algorithmic randomness as in [10]). From various notions of lowness, such as "low for random," they have defined weak reducibilities, such as the LR reducibility $\leq_{LR}$. We could try to define a reducibility with the notion of "low for graph neighborhood," but unfortunately it would not define a weak reducibility, but only because it does not respect the jump operator. Perhaps we could fix this by relativizing appropriately. A future direction would be to investigate whether we could find the appropriate relativization, and then prove properties about this new reducibility and how it relates to other notions of lowness.

Acknowledgements

The authors would like to thank the anonymous reviewer for several helpful suggestions.

References

[1] Dwight R. Bean, *Effective coloration*, J. Symbolic Logic **41** (1976), no. 2, 469–480. MR0416889 (54 #4952)

[2] ———, *Recursive Euler and Hamilton paths*, Proc. Amer. Math. Soc. **55** (1976), no. 2, 385–394. MR0416888 (54 #4951)

[3] Hans-Georg Carstens and Peter Päppinghaus, *Recursive coloration of countable graphs*, Ann. Pure Appl. Logic **25** (1983), no. 1, 19–45. MR722167 (85h:03045)

[4] William I. Gasarch and Andrew C. Y. Lee, *On the finiteness of the recursive chromatic number*, Ann. Pure Appl. Logic **93** (1998), no. 1-3, 73–81. Computability theory. MR1635600 (2000a:03070)

[5] David Harel, *Hamiltonian paths in infinite graphs*, Israel J. Math. **76** (1991), no. 3, 317–336. MR1177348 (93d:68023)

[6] Matthew Jura, Oscar Levin, and Tyler Markkanen, *Domatic partitions of computable graphs*, Arch. Math. Logic **53** (2014), no. 1-2, 137–155. MR3151402

[7] Henry A. Kierstead, *Recursive colorings of highly recursive graphs*, Canad. J. Math. **33** (1981), no. 6, 1279–1290. MR645224 (84b:05045)

[8] 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. MR0314610 (47 #3161)

[9] Russell Miller, *The Δ^0_2-spectrum of a linear order*, J. Symbolic Logic **66** (2001), no. 2, 470–486. MR1833459 (2002e:03065)

[10] André Nies, *Computability and randomness*, Oxford Logic Guides, vol. 51, Oxford University Press, Oxford, 2009. MR2548883 (2011i:03003)

[11] Gerald E. Sacks, *Degrees of unsolvability*, Princeton University Press, Princeton, N.J., 1963. MR0186554 (32 #4013)

[12] Leonard P. Sasso Jr., *A minimal degree not realizing least possible jump*, J. Symbolic Logic **39** (1974), 571–574. MR0360242 (50 #12692)

[13] James H. Schmerl, *Recursive colorings of graphs*, Canad. J. Math. **32** (1980), no. 4, 821–830. MR590647 (81m:03054)

[14] 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. MR882921 (88m:03003)

[15] C. E. M. Yates, *Initial segments of the degrees of unsolvability. II. Minimal degrees.*, J. Symbolic Logic **35** (1970), 243–266. MR0274288 (43 #53)