Graph Turing Machines

Nathanael L. Ackerman[1] and Cameron E. Freer[2]

[1] Department of Mathematics, Harvard University, Cambridge, MA 02138, USA
nate@math.harvard.edu
[2] Remine, Falls Church, VA 22043, USA
cameron@remine.com

Abstract. We consider *graph Turing machines*, a model of parallel computation on a graph, which provides a natural generalization of several standard computational models, including ordinary Turing machines and cellular automata. In this extended abstract, we give bounds on the computational strength of functions that graph Turing machines can compute. We also begin the study of the relationship between the computational power of a graph Turing machine and structural properties of its underlying graph.

1 Introduction

When studying large networks, it is important to understand what sorts of computations can be performed in a distributed way on a given network. In particular, it is natural to consider the setting where each node acts independently in parallel, and where the network is specified separately from the computation to be performed. In order to study networks whose size is considerably larger than can be held in memory by the computational unit at any single node, it is often useful to model the network as an infinite graph. (For a discussion of modeling large networks via infinite graphs, see, e.g., [Lov09].)

We define a notion of *graph Turing machine* that is meant to capture this setting. This notion generalizes several other well-known models of computation, including ordinary Turing machines, cellular automata, and parallel graph dynamical systems. Each of these models, in turn, occurs straightforwardly as a special case of a graph Turing machine, suggesting that graph Turing machines capture a natural concept of parallel computation on graphs.

A graph Turing machine (henceforth abbreviated as "graph machine") performs computation on a vertex-labeled edge-colored directed multigraph satisfying certain properties. This notion of computation is designed to capture the idea that in each timestep, every vertex performs a limited amount of computation (in parallel, independently of the other vertices), and can only distinguish vertices connected to it when they are connected by different colors of edges.

June 15, 2017

In this paper we study the functions that can be computed using graph machines, which we call *graph computable* functions. As we will see, this parallel notion of computation will yield significantly greater computational strength than ordinary Turing machines. We will see that the computational strength of graph machines is exactly that of $\mathbf{0}^{(\omega)}$, the Turing degree of true arithmetic (thereby providing another natural construction of this degree). We also begin to examine the relationship between various properties of the underlying graph (e.g., finiteness of degree) and the computational strength of the resulting graph machines.

In this extended abstract, we state the main results and provide proofs or proof sketches of several of these results. For detailed proofs and other related results, see the full version at https://arxiv.org/abs/1703.09406.

1.1 Main results and overview of the paper

We begin by introducing the notions of colored graphs, graph machines, and graph computability in Section 2.

Our main results fall into two classes: bounds on the computational power of arbitrary computable graph machines, and bounds among machines with an underlying graph every vertex of which has finite degree (in which case we say the graph is of *finite degree*).

Theorem 3.6 states that every graph computable function is Turing reducible to $\mathbf{0}^{(\omega)}$. In the other direction, we show in Theorem 3.10 that this bound is attained by a single graph Turing machine.

Sitting below $\mathbf{0}^{(\omega)}$ are the arithmetical Turing degrees, i.e., those less than $\mathbf{0}^{(n)}$ for some $n \in \mathbb{N}$, where $\mathbf{0}^{(n)}$ denotes the n-fold iterate of the halting problem. We show in Corollary 3.9 that every arithmetical Turing degree contains a function that is graph Turing computable. (It remains open whether every degree below $\mathbf{0}^{(\omega)}$ can be achieved.)

We next show in Corollary 4.9 that functions determined by graph machines with underlying graph of finite degree are reducible to the halting problem, $\mathbf{0}'$. Further, we show in Corollary 4.10 that if we restrict to graph machines where every vertex has the same (finite) degree, then the resulting graph computable function is computable by an ordinary Turing machine.

We also show in Theorem 4.11 that every Turing degree below $\mathbf{0}'$ is the degree of some graph computable function with underlying graph of finite degree.

In Section 5, we examine how several other models of computation can be viewed as special cases of graph machines, including ordinary Turing machines, cellular automata, and parallel graph dynamical systems. Note that there have been many other attempts (which we do not discuss at length here) to extend Turing machines to operate on graphs, including [AAB+14], [Knu68, pp. 462–463], [KU58], and [Sch80] — the first of which calls its different notion a "graph Turing machine" as well.

1.2 Notation

If $f\colon A \to \prod_{i\leq n} B_i$ and $k \leq n \in \mathbb{N}$, then we let $f_{[k]}\colon A \to B_k$ be the composition of f with the projection map onto the k'th coordinate.

Fix an enumeration of computable partial functions, and for $e \in \mathbb{N}$, let $\{e\}$ be the e'th such function in this list. If X and Y are sets with $0 \in Y$, let $Y^{<X} = \{\eta\colon X \to Y \,:\, |\{a \,:\, \eta(a) \neq 0\}| < \omega\}$, i.e., the collection of functions from X to Y for which all but finitely many inputs yield 0. (Note that by this notation we do *not* mean partial functions from X to Y supported on fewer than $|X|$-many elements.) For a set X, let $\mathfrak{P}(X)$ denote the collection of subsets of X and let $\mathfrak{P}_{<\omega}(X)$ denote the collection of finite subsets of X. Note that the map which takes a subset of X to its characteristic function is a bijection between $\mathfrak{P}_{<\omega}(X)$ and $\{0,1\}^{<X}$.

When working with computable graphs, sometimes the underlying set of the graph will be a finite coproduct of finite sets and $\mathbb{N}^k$ for $k \in \mathbb{N}$. The standard notion of computability for $\mathbb{N}$ transfers naturally to such settings, making implicit use of the computable bijections between $\mathbb{N}^k$ and $\mathbb{N}$, and between $\coprod_{i\leq k} \mathbb{N}$ and $\mathbb{N}$, for $k \in \mathbb{N}$. We will sometimes say *computable set* to refer to some computable subset (with respect to these bijections) of such a finite coproduct X, and *computable function* to refer to a computable function having domain and codomain of that form or of the form $F^{<X}$ for some finite set F.

For sets $X, Y \subseteq \mathbb{N}$, we write $X \leq_{\mathrm{T}} Y$ when X is Turing reducible to Y (and similarly for functions and other computably presented countable objects). We write $X \equiv_{\mathrm{T}} Y$ when $X \leq_{\mathrm{T}} Y$ and $Y \leq_{\mathrm{T}} X$. For more details on results and notation in computability theory, see [Soa87].

2 Graph computing

We now define graph Turing machines and graph computable functions. Note that these definitions differ from those in the full version of this paper, as here we require the sets of labels and colors to be finite. This simplifies the presentation, while losing little generality (since all of our constructions produce graphs with this property).

Definition 2.1. *A* **colored graph** *is a tuple $\mathcal{G}$ of the form $(G, (L, V), (C, E))$ where*

- *G is a set, called the* **underlying set** *or the* **set of vertices**,
- *L is a finite set, called the* **set of labels**, *and $V\colon G \to L$ is called the* **labeling function**, *and*
- *C is a finite set, called the* **set of colors**, *and $E\colon G \times G \to \mathfrak{P}(C)$ is called the* **edge coloring**.

A **computable colored graph** *is a colored graph such that G is a computable set and V and E are computable functions.*

The intuition is that a colored graph is an edge-colored directed multigraph where each vertex is assigned a label, and such that among the edges from one vertex to another, there is at most one of each color. Eventually, we will allow each vertex to do some fixed finite amount of computation, and we will want vertices with the same label to perform the same computations.

For the rest of the paper by a *graph* we will always mean a colored graph, and will generally write the symbol $\mathcal{G}$ to refer to graphs.

Let $\mathcal{G}$ be a graph with underlying set G, and suppose $A \subseteq G$. Define $\mathcal{G}|_A$ to be the graph with underlying set A having the same set of labels and set of colors as $\mathcal{G}$, such that the labeling function and edge coloring function of $\mathcal{G}|_A$ are the respective restrictions to A.

Definition 2.2. *A* **graph Turing machine***, or simply* **graph machine***, is a tuple* $\mathfrak{M} = (\mathcal{G}, (\mathfrak{A}, \{0,1\}), (S, s), T)$ *where the following hold.*

- $\mathcal{G} = (G, (L, V), (C, E))$ *is a graph, called the* **underlying graph***. We will speak of the components of the underlying graph as if they were components of the graph machine itself. For example, we will call G the* underlying set *of $\mathfrak{M}$ as well as of $\mathcal{G}$.*
- $\mathfrak{A}$ *is a finite set, called the* **alphabet***, having distinguished symbols* 0 *and* 1*.*
- S *is a finite set, called the* **collection of states***.*
- s *is a distinguished state, called the* **initial state***.*
- $T\colon L \times \mathfrak{P}(C) \times \mathfrak{A} \times S \to \mathfrak{P}(C) \times \mathfrak{A} \times S$ *is a function, called the* **lookup table***, such that $T(\ell, \emptyset, 0, s) = (\emptyset, 0, s)$ for all $\ell \in L$, i.e., if any vertex is in the initial state, currently displays* 0*, and has received no pulses, then that vertex doesn't do anything in the next step. This lookup table can be thought of as specifying a* transition function*.*

A **$\mathcal{G}$-Turing machine** *(or simply a* **$\mathcal{G}$-machine***) is a graph machine with underlying graph $\mathcal{G}$. A* **computable graph machine** *is a graph machine whose underlying graph is computable.*

If A is a subset of the underlying set of $\mathfrak{M}$, then $\mathfrak{M}|_A$ is the graph machine with underlying graph $\mathcal{G}|_A$ having the same alphabet, states, and lookup table as $\mathfrak{M}$.

The intuition is that a graph machine should consist of a graph where at each timestep, every vertex is assigned a state and an element of the alphabet, which it displays. To figure out how these assignments are updated over time, we apply the transition function determined by the lookup table which tells us, given the label of a vertex, its current state, and its currently displayed symbol, along with the colored *pulses* the vertex has most recently received, what state to set the vertex to, what symbol to display next, and what colored pulses to send to its neighbors.

For the rest of the paper, $\mathfrak{M} = (\mathcal{G}, (\mathfrak{A}, \{0,1\}), (S, s), T)$ will denote a *computable* graph machine whose underlying (computable) graph is $\mathcal{G} = (G, (L, V), (C, E))$.

Definition 2.3. *A* **configuration** *of $\mathfrak{M}$ is a function $f\colon G \to \mathfrak{P}(C) \times \mathfrak{A} \times S$. A configuration f is a* **bounded configuration** *when $|\{v \in G : f_{[2]}(v) \neq 0\}|$ is finite. A bounded configuration f is a* **starting configuration** *when further $f_{[1]}(v) = \emptyset$ and $f_{[3]}(v) = s$ for all $v \in G$.*

In other words, a starting configuration is an assignment in which all vertices are in the initial state s, no pulses have been sent, and only finitely many vertices display a non-zero symbol.

Note that if A is a subset of the underlying set of $\mathfrak{M}$ and f is a starting configuration for $\mathfrak{M}$, then $f|_A$ is a starting configuration for $\mathfrak{M}|_A$.

Definition 2.4. *Given a configuration f for $\mathfrak{M}$, the* **run** *of $\mathfrak{M}$ on f is the function $\langle \mathfrak{M}, f\rangle\colon G \times \mathbb{N} \to \mathfrak{P}(C) \times \mathfrak{A} \times S$ satisfying, for all $v \in G$,*

- $\langle \mathfrak{M}, f\rangle(v, 0) = f(v)$ *and*
- $\langle \mathfrak{M}, f\rangle(v, n+1) = T(V(v), X, z, t)$ *for all $n \in \mathbb{N}$, where*
  - $X = \bigcup_{w \in G} \big(E(w, v) \cap \langle \mathfrak{M}, f\rangle_{[1]}(w, n)\big)$,
  - $z = \langle \mathfrak{M}, f\rangle_{[2]}(v, n)$, *and*
  - $t = \langle \mathfrak{M}, f\rangle_{[3]}(v, n)$.

We say that a run **halts at stage** *n if $\langle \mathfrak{M}, f\rangle(v, n) = \langle \mathfrak{M}, f\rangle(v, n+1)$ for all $v \in G$.*

A run of a graph machine is the function which takes a configuration for the graph machine and a natural number n, and returns the result of letting the graph machine process the configuration for n-many timesteps.

The following lemma is immediate from Definition 2.4.

Lemma 2.5. *Suppose f is a configuration for $\mathfrak{M}$, and for each $n \in \mathbb{N}$, define $f_n := \langle \mathfrak{M}, f\rangle(\,\cdot\,, n)$. Then for all $n, m \in \mathbb{N}$ and $v \in G$, the function f_n is a configuration for $\mathfrak{M}$ such that $f_{n+m}(v) = \langle \mathfrak{M}, f_n\rangle(v, m) = \langle \mathfrak{M}, f\rangle(v, n+m)$.* □

We now describe how a graph machine defines a function.

Definition 2.6. *For $x \in \mathfrak{A}^{<G}$, let $\widehat{x}$ be the configuration such that $\widehat{x}(v) = (\emptyset, x(v), s)$ for all $v \in G$. Let $\{\mathfrak{M}\}\colon \mathfrak{A}^{<G} \to \mathfrak{A}^G$ be the partial function such that*

- *$\{\mathfrak{M}\}(x)\!\uparrow$, i.e., is undefined, if the run $\langle \mathfrak{M}, \widehat{x}\rangle$ does not halt, and*
- *$\{\mathfrak{M}\}(x) = y$ if $\langle T, \widehat{x}\rangle$ halts at stage n and $y(v) = \langle \mathfrak{M}, \widehat{x}\rangle_{[2]}(v, n)$ for all $v \in G$.*

Note that $\{\mathfrak{M}\}(x)$ is well defined as $\widehat{x}$ is always a starting configuration for $\mathfrak{M}$.

The graph machine $\mathfrak{M}$ is **total bounded** *if $\{\mathfrak{M}\}(\widehat{x}) \in \mathfrak{A}^{<G}$ for all $x \in \mathfrak{A}^{<G}$; in particular, the partial function $\{\mathfrak{M}\}$ is total. When $\mathfrak{M}$ is total bounded, we will sometimes write $\{\mathfrak{M}\}\colon \mathfrak{A}^{<G} \to \mathfrak{A}^{<G}$.*

While in general, the output of $\{\mathfrak{M}\}(x)$ might have infinitely many non-zero elements, for purposes of considering which Turing degrees are graph computable, we will mainly be interested in the case of total bounded machines.

When defining a function using a graph machine, it will often be convenient to have extra vertices whose labels don't affect the function being defined, but whose presence allows for a simpler definition. These extra vertices can be thought as "scratch paper" and play the role of extra tapes (beyond the main input/output tape) in a multi-tape Turing machine. We now make this precise.

Definition 2.7. *Let X be an infinite computable subset of G. A map $\zeta\colon \mathfrak{A}^{<X} \to \mathfrak{A}^X$ is* $\langle \mathcal{G}, X\rangle$**-computable** *via $\mathfrak{M}$ if*

(a) $\{\mathfrak{M}\}$ is total,
(b) for $x, y \in \mathfrak{A}^{<G}$, if $x|_X = y|_X$ then $\{\mathfrak{M}\}(x) = \{\mathfrak{M}\}(y)$, and
(c) for all $x \in \mathfrak{A}^{<G}$, for all $v \in G \setminus X$, we have $\{\mathfrak{M}\}(x)(v) = 0$, i.e., when $\{\mathfrak{M}\}(x)$ halts, v displays 0, and
(d) for all $x \in \mathfrak{A}^{<G}$, we have $\{\mathfrak{M}\}(x)|_X = \zeta(x|_X)$.

A function is $\mathcal{G}$**-computable via** $\mathfrak{M}$ *if it is $\langle \mathcal{G}, X\rangle$-computable via $\mathfrak{M}$ for some infinite computable $X \subseteq G$. A function is* $\mathcal{G}$**-computable** *if it is $\mathcal{G}$-computable via $\mathfrak{M}^\circ$ for some computable $\mathcal{G}$-machine $\mathfrak{M}^\circ$. A function is* **graph Turing computable**, *or simply* **graph computable**, *when it is $\mathcal{G}^\circ$-computable for some computable graph $\mathcal{G}^\circ$.*

The following easy lemma captures the sense in which functions that are $\langle \mathcal{G}, X\rangle$-computable via $\mathfrak{M}$ are determined by their restrictions to X.

Lemma 2.8. *Let X be an infinite computable subset of $\mathcal{G}$. There is at most one function $\zeta\colon \mathfrak{A}^{<X} \to \mathfrak{A}^X$ that is $\langle \mathcal{G}, X\rangle$-computable via $\mathfrak{M}$, and it must be Turing equivalent to $\{\mathfrak{M}\}$.* □

3 Arbitrary graphs

In this section, we consider the possible Turing degrees of total graph computable functions. We begin with a bound for finite graphs.

Lemma 3.1. *Suppose G is finite. Let $\mathbf{h}$ be the map which takes a configuration f for $\mathfrak{M}$ and returns $n \in \mathbb{N}$ if $\langle \mathfrak{M}, f\rangle$ halts at stage n (and not earlier), and returns ∞ if $\langle \mathfrak{M}, f\rangle$ doesn't halt. Then $\langle \mathfrak{M}, f\rangle$ is computable and $\mathbf{h}$ is computable.*

Proof. Because G is finite, $\langle \mathfrak{M}, f\rangle$ is computable. Further, there are are only finitely many configurations of $\mathfrak{M}$. Hence there must be some $n, k \in \mathbb{N}$ such that for all vertices v in the underlying set of $\mathfrak{M}$, we have $\langle \mathfrak{M}, f\rangle(v, n) = \langle \mathfrak{M}, f\rangle(v, n+k)$, and the set of such pairs (n, k) is computable. Note that $\langle \mathfrak{M}, f\rangle$ halts if and only if there is some n, less than or equal to the number of configurations for $\mathfrak{M}$, for which this holds for $(n, 1)$. Hence $\mathbf{h}$, which searches for the least such n, is computable. □

We next investigate which Turing degrees are achieved by arbitrary computable graph machines.

3.1 Upper bound

We now show that every graph computable function is computable from $\mathbf{0}^{(\omega)}$.

Definition 3.2. *Let f be a configuration for $\mathfrak{M}$, and let A be a finite subset of G. We say that $(B_i)_{i\leq n}$ is an* n**-approximation of** $\mathfrak{M}$ **and** f **on** A *if*

- $A = B_0$,
- $B_i \subseteq B_{i+1} \subseteq G$ *for all* $i < n$, *and*
- *if* $B_{i+1} \subseteq B \subseteq G$ *then* $\langle \mathfrak{M}|_{B_{i+1}}, f_i|_{B_{i+1}} \rangle(v,1) = \langle \mathfrak{M}|_B, f_i|_B \rangle(v,1)$ *for all* $v \in B_{i+1}$, *where again* $f_i := \langle \mathfrak{M}, f \rangle(\cdot\,, i)$.

The following proposition (in the case where $\ell = n - n'$) states that if $(B_i)_{i \leq n}$ is an n-approximation of $\mathfrak{M}$ and f on A, then as long as we are only running $\mathfrak{M}$ with starting configuration f for ℓ-many steps, and are only considering the states of elements within $B_{n'}$, then it suffices to restrict $\mathfrak{M}$ to B_n. It follows by a straightforward though technical induction.

Proposition 3.3. *The following claim holds for every* $n \in \mathbb{N}$*: For every configuration* f *for* $\mathfrak{M}$, *and finite* $A \subseteq G$,

- *there is an* n*-approximation of* $\mathfrak{M}$ *and* f *on* A, *and*
- *if* $(B_i)_{i \leq n}$ *is such an approximation, then*

$$(\forall n' < n)(\forall \ell \leq n - n')(\forall v \in B_{n'}) \langle \mathfrak{M}|_{B_{n'+\ell}}, f|_{B_{n'+\ell}} \rangle(v, \ell) = \langle \mathfrak{M}, f \rangle(v, \ell). \quad \square$$

We now analyze the computability of approximations and of runs.

Proposition 3.4. *Let* $n \in \mathbb{N}$. *For all computable graph machines* $\mathfrak{M}$ *and configurations* f *for* $\mathfrak{M}$, *the following are* $\mathbf{f}^{(n)}$*-computable, uniformly in* n, *where* $\mathbf{f}$ *is the Turing degree of* f.

- *The collection* $P_n(f) := \{(A, (B_i)_{i \leq n}) : A \subseteq G$ *is finite and* $(B_i)_{i \leq n}$ *is an* n*-approximation of* $\mathfrak{M}$ *and* f *on* $A\}$.
- *The function* $f_n := \langle \mathfrak{M}, f \rangle(\cdot\,, n)$. $\square$

Corollary 3.5. *If* f *is a configuration for* $\mathfrak{M}$, *then* f_n *is* $\mathbf{f}^{(n)}$*-computable and so* $\langle \mathfrak{M}, f \rangle$ *is* $\mathbf{f}^{(\omega)}$*-computable, where* $\mathbf{f}$ *is the Turing degree of* f.

Proof. By Proposition 3.3, for each $v \in G$ (the underlying set of $\mathfrak{M}$) and each $n \in \mathbb{N}$, there is an approximation of $\mathfrak{M}$ for f and $\{v\}$ up to n. Further, by Proposition 3.4 we can $\mathbf{f}^{(n)}$-compute such an approximation, uniformly in v and n. But if $(B_i^v)_{i \leq n}$ is an approximation of $\mathfrak{M}$ for f and $\{v\}$ up to n then $\langle \mathfrak{M}|_{B_n^v}, f|_{B_n^v} \rangle(v, n) = \langle \mathfrak{M}, f \rangle(v, n)$. So $f_n = \langle \mathfrak{M}, f \rangle(\cdot\,, n)$ is $\mathbf{f}^{(n)}$-computable, uniformly in n. Hence $\langle \mathfrak{M}, f \rangle$ is $\mathbf{f}^{(\omega)}$-computable. $\square$

Theorem 3.6. *Suppose that* $\{\mathfrak{M}\}$ *is a total function. Then* $\{\mathfrak{M}\}$ *is computable from* $\mathbf{0}^{(\omega)}$.

Proof. Let f be any starting configuration of $\mathfrak{M}$. Then f is computable. Hence by Corollary 3.5, $\langle \mathfrak{M}, f \rangle(v, n+1)$ is $\mathbf{0}^{(n+1)}$-computable. This then implies that the function determining whether or not $\{\mathfrak{M}\}(x)$ halts after n steps is $\mathbf{0}^{(n+2)}$-computable.

But by assumption, $\{\mathfrak{M}\}(x)$ halts for every $x \in \mathfrak{A}^{<G}$, and so $\{\mathfrak{M}\}$ is $\mathbf{0}^{(\omega)}$-computable. $\square$

3.2 Lower bound

We have seen that every graph computable function is computable from $\mathbf{0}^{(\omega)}$. In this subsection, we will see that this bound can be obtained. We begin by showing that every arithmetical Turing degree has an element that is graph computable. From this we then deduce that there is a graph computable function Turing equivalent to $\mathbf{0}^{(\omega)}$.

We first recall the following standard result from computability theory (see [Soa87, III.3.3]).

Lemma 3.7. *Suppose $n \in \mathbb{N}$ and $X \subseteq \mathbb{N}$. Then the following are equivalent.*

- $X \leq_{\mathrm{T}} \mathbf{0}^{(n)}$.
- *There is a computable function $g\colon \mathbb{N}^{n+1} \to \mathbb{N}$ such that*
  - $h(\,\cdot\,) := \lim_{x_0 \to \infty} \cdots \lim_{x_{n-1} \to \infty} g(x_0, \ldots, x_{n-1}, \,\cdot\,)$ *is total.*
  - $h \equiv_{\mathrm{T}} X$. □

We now sketch the following construction.

Proposition 3.8. *Let $n \in \mathbb{N}$ and suppose $g\colon \mathbb{N}^{n+1} \to \mathbb{N}$ is computable such that*

$$h(\,\cdot\,) := \lim_{x_0 \to \infty} \cdots \lim_{x_{n-1} \to \infty} g(x_0, \ldots, x_{n-1}, \,\cdot\,)$$

is total. Then h is graph computable via a graph machine whose lookup table does not depend on n or g.

Proof sketch. We define a "subroutine" graph machine that, on its own, computes the limit of a computable binary sequence. We then embed n repetitions of this subroutine into a single graph machine that computes the n-fold limit of the $(n+1)$-dimensional array given by g.

The subroutine graph machine has a countably infinite sequence (with one special vertex) as its underlying graph, in that every vertex is connected to all previous vertices (and all are connected to the special vertex). Each vertex first activates itself, setting its displayed symbol to the appropriate term in the sequence whose limit is being computed. Each vertex sends a pulse to every previous vertex, signaling its displayed state. Any vertex which receives both 0 and 1 from vertices later in the sequence knows that the sequence alternates at some later index. Finally, any vertex which only receives a 0 or 1 pulse, but not both, sends a pulse corresponding to the one it receives to the special vertex. This special vertex then knows the limiting value of the sequence. □

This technical construction allows us to conclude the following.

Corollary 3.9. *Suppose $X \subseteq \mathbb{N}$ is such that $X \leq_{\mathrm{T}} \mathbf{0}^{(n)}$ for some $n \in \mathbb{N}$. Then X is Turing-equivalent to some graph computable function.*

Proof. By Lemma 3.7, X is Turing equivalent to the n-fold limit of some computable function. By Proposition 3.8, this n-fold limit is graph computable. □

Not only are all graph computable functions Turing reducible to $\mathbf{0}^{(\omega)}$, but this bound can be achieved.

Theorem 3.10. *There is a graph computable function that is Turing equivalent to* $\mathbf{0}^{(\omega)}$. □

4 Finite degree graphs

We have seen that every arithmetical function is graph computable. However, as we will see in this section, if we instead limit ourselves to graphs where each vertex has finite degree, then not only is every graph computable function computable from $\mathbf{0}'$, but also we can obtain more fine-grained control over the Turing degree of the function by studying the degree structure of the graph.

4.1 Upper bound

Before we move to the specific case of graphs of finite degree (defined below), there is an important general result concerning bounds on graph computability and approximations to computations.

Definition 4.1. *Let* $\Theta\colon \mathfrak{P}_{<\omega}(G) \to \mathfrak{P}_{<\omega}(G)$. *We say that* Θ *is a* **uniform approximation** *of* $\mathfrak{M}$ *if for all finite subsets* $A \subseteq G$, *we have* $A \subseteq \Theta(A)$, *and for any configuration* f *for* $\mathfrak{M}$, *the pair* $(A, \Theta(A))$ *is a* 1*-approximation of* $\mathfrak{M}$ *and* f *on* A.

The following is an easy induction on n.

Lemma 4.2. *Let* $\Theta(A)$ *be a uniform approximation of* $\mathfrak{M}$. *Then for any finite subset* A *of* G, *any configuration* f *for* $\mathfrak{M}$, *and any* $n \in \mathbb{N}$, *the tuple* $(A, \Theta(A), \Theta^2(A), \ldots, \Theta^n(A))$ *is an* n*-approximation of* $\mathfrak{M}$ *and* f *on* A. □

Note that while we will be able to get even better bounds in the case of finite degree graphs, we do have the following bound on computability, as a straightforward consequence of Lemma 4.2.

Lemma 4.3. *Let* Θ *be a uniform approximation of* $\mathfrak{M}$. *Then for any configuration* f, *the function* $\langle \mathfrak{M}, f \rangle$ *is computable from* Θ *and* f *(uniformly in* f*), and if* $\{\mathfrak{M}\}$ *is total, then* $\{\mathfrak{M}\} \leq_{\mathrm{T}} \Theta'$. □

We now introduce the *degree function* of a graph.

Definition 4.4. *For* $v \in G$, *define the* **degree** *of* v *to be the number of vertices incident with it, i.e.,* $\deg_{\mathcal{G}}(v) := |\{w : E(v,w) \cup E(w,v) \neq \emptyset\}|$, *and call* $\deg_{\mathcal{G}}(\,\cdot\,)\colon G \to \mathbb{N} \cup \{\infty\}$ *the* **degree function** *of* $\mathcal{G}$.

We say that $\mathcal{G}$ *has* **finite degree** *when* $\mathrm{rng}(\deg_{\mathcal{G}}) \subseteq \mathbb{N}$, *and say that* $\mathcal{G}$ *has* **constant degree** *when* $\deg_{\mathcal{G}}$ *is constant.*

We will see that for a graph $\mathcal{G}$ of finite degree, its degree function bounds the computability of $\mathcal{G}$-computable functions.

The following easy lemma (using the fact that each vertex of a computable graph has a computably enumerable set of neighbors) allows us to provide a computation bound on graph Turing machines all vertices of whose underlying graph have finite degree.

Lemma 4.5. *Suppose that $\mathcal{G}$ has finite degree. Then $\deg_{\mathcal{G}} \leq_{\mathrm{T}} \mathbf{0}'$.* □

We next need the following definition.

Definition 4.6. *For each $A \subseteq G$ and $n \in \mathbb{N}$, the n-**neighborhood** of A, written $\mathbf{N}_n(A)$, is defined by induction as follows.*
Case 1: The 1-neighborhood of A is $\mathbf{N}_1(A) := A \cup \{v \in G : (\exists a \in A)\ E(v,a) \cup E(a,v) \neq \emptyset\}$.
Case $k+1$: The $k+1$-neighborhood of A is $\mathbf{N}_{k+1}(A) = \mathbf{N}_1(\mathbf{N}_k(A))$.

Lemma 4.7. *Suppose that $\mathcal{G}$ has finite degree. Then*

(a) the 1-neighborhood map $\mathbf{N}_1$ is computable from $\deg_{\mathcal{G}}$, and
(b) for any $\mathcal{G}$-machine $\mathfrak{M}$, the map $\mathbf{N}_1$ is a uniform approximation to $\mathfrak{M}$.

Proof. Clause (a) follows from the fact that given the degree of a vertex one can search for all of its neighbors, as this set is computably enumerable (uniformly in the vertex) and of a known finite size.

Clause (b) follows from the fact that if a vertex receives a pulse, it must have come from some element of its 1-neighborhood. □

The following more precise upper bound on the computability of a graph computable function holds when the underlying graph has finite degree.

Theorem 4.8. *Suppose that $\mathcal{G}$ has finite degree and $\{\mathfrak{M}\}$ is total. Then $\mathfrak{M}$ is bounded and $\{\mathfrak{M}\}$ is computable from $\deg_{\mathcal{G}}$.* □

The following important corollaries are immediate from Lemma 4.5, Theorem 4.8, and the fact that if $\deg_{\mathcal{G}}$ is constant, it is computable.

Corollary 4.9. *Suppose that $\mathcal{G}$ has finite degree. Then any $\mathcal{G}$-computable function is $\mathbf{0}'$-computable.* □

Corollary 4.10. *Suppose that $\mathcal{G}$ has constant degree. Then any $\mathcal{G}$-computable function is computable (in the ordinary sense).* □

4.2 Lower bound

Finally we consider the possible Turing degrees of graph computable functions where the underlying graph has finite degree. In particular, we show that every Turing degree below $\mathbf{0}'$ is the degree of some total graph computable function where the underlying graph has finite degree.

Recall from Lemma 3.7 (for $n = 1$) that a set $X \subseteq \mathbb{N}$ satisfies $X \leq_{\mathrm{T}} \mathbf{0}'$ when the characteristic function of X is the limit of a 2-parameter computable function.

Theorem 4.11. *For every $X : \mathbb{N} \to \{0,1\}$ such that $X \leq_{\mathrm{T}} \mathbf{0}'$ there is a graph machine $\mathcal{N}_X$ such that every vertex of its underlying graph has degree at most 3, and $\{\mathcal{N}_X\}$ is total and Turing equivalent to X.* □

5 Representations of other computational models via graph machines

Many other models of computation can be viewed as special cases of graph machines, providing further evidence for graph machines being a universal model of computation on graphs.

5.1 Ordinary Turing machines

An ordinary Turing machine can be simulated by a graph Turing machine, where the tape of the Turing machine is encoded by an underlying graph that is a $\mathbb{Z}$-chain.

The doubly-infinite one-dimensional read/write tape of an ordinary Turing machine has cells indexed by $\mathbb{Z}$, the free group on one generator, and in each timestep the head moves according to the generator or its inverse. This interpretation of a Turing machine as a $\mathbb{Z}$-machine has been generalized to H-machines for arbitrary finitely generated groups H by [ABS17], and the simulation mentioned above extends straightforwardly to this setting as well.

One might next consider how cleanly one might embed various extensions of Turing machines where the tape is replaced by a graph, such as Kolmogorov–Uspensky machines [KU58], Knuth's pointer machines [Knu68, pp. 462–463], and Schönhage's storage modification machines [Sch80]. For a further discussion of these and their relation to sequential abstract state machines, see [Gur93] and [Gur00].

5.2 Cellular automata

Cellular automata (which we take to be finite-dimensional, finite-radius and with finitely-many states) can be naturally simulated by graph machines, moreover of constant degree. In particular, Corollary 4.10 applies to this embedding. For background on cellular automata, see, e.g., the book [TM87].

Cells of the automata are taken to be the vertices of the graph, and cells are connected to its "neighbors" (other cells within the given radius) by a collection of edges (of the graph) whose labels encode their relative position (e.g., "1 to the left of") and all possible cellular automaton states. (In particular, every vertex has the same finite degree.) The displayed symbol of each vertex encodes the cellular automaton state of that cell. The rule of the cellular automaton is encoded in the lookup table so as to provide a bisimulation between the original cellular automaton and the graph machine built based on it.

Not only is the evolution of every such cellular automata computable (moreover via this embedding as a graph machine), but there are particular automata whose

evolution encodes the behavior of a universal Turing machine [Coo04], [WN09]. Several researchers have also considered the possibility of expressing intermediate Turing degrees via this evolution; see [Bal04], [Coh02], and [Sut03]. Analogously, one might ask which Turing degrees can be expressed in the evolution of graph machines.

5.3 Parallel graph dynamical systems

Parallel graph dynamical systems [AMV15b] can be viewed as essentially equivalent to the finite case of graph Turing machines, as we now describe. Finite cellular automata can also be viewed as a special case of parallel graph dynamical systems, as can finite boolean networks, as noted in [AMV15b, §2.2]. For more on parallel graph dynamical systems, see [AMV15a], [AMV15b], and [BCZ04].

In contrast, it is not immediately clear how best to encode an arbitrary (parallel) abstract state machine [BG03] as a graph Turing machine (due to the higher arity relations of the ASM).

Acknowledgements

The authors would like to thank Tomislav Petrović, Linda Brown Westrick, and the anonymous referees of earlier versions for helpful comments.

Bibliography

[AAB+14] D. Angluin, J. Aspnes, R. A. Bazzi, J. Chen, D. Eisenstat, and G. Konjevod, *Effective storage capacity of labeled graphs*, Inform. Comput. **234** (2014), 44–56.

[ABS17] N. Aubrun, S. Barbieri, and M. Sablik, *A notion of effectiveness for subshifts on finitely generated groups*, Theoret. Comput. Sci. **661** (2017), 35–55.

[AMV15a] J. A. Aledo, S. Martinez, and J. C. Valverde, *Graph dynamical systems with general Boolean states*, Appl. Math. Inf. Sci. **9** (2015), no. 4, 1803–1808.

[AMV15b] ———, *Parallel dynamical systems over graphs and related topics: a survey*, J. Appl. Math. (2015), no. 594294.

[Bal04] J. Baldwin, *Review of* A New Kind of Science *by Stephen Wolfram*, Bull. Symb. Logic **10** (2004), no. 1, 112–114.

[BCZ04] C. L. Barrett, W. Y. C. Chen, and M. J. Zheng, *Discrete dynamical systems on graphs and Boolean functions*, Math. Comput. Simulation **66** (2004), no. 6, 487–497.

[BG03] A. Blass and Y. Gurevich, *Abstract state machines capture parallel algorithms*, ACM Trans. Comput. Log. **4** (2003), no. 4, 578–651.

[Coh02] H. Cohn, *Review of* A New Kind of Science *by Stephen Wolfram*, MAA Reviews (2002).

[Coo04] M. Cook, *Universality in elementary cellular automata*, Complex Syst. **15** (2004), no. 1, 1–40.

[Gur93] Y. Gurevich, *Kolmogorov machines and related issues*, Current Trends in Theoretical Computer Science, World Scientific Series in Computer Science, vol. 40, World Scientific, 1993, pp. 225–234.

[Gur00] ———, *Sequential abstract-state machines capture sequential algorithms*, ACM Trans. Comput. Log. **1** (2000), no. 1, 77–111.

[Knu68] D. E. Knuth, *The art of computer programming. Vol. 1: Fundamental algorithms*, Addison-Wesley, 1968.

[KU58] A. N. Kolmogorov and V. A. Uspensky, *On the definition of an algorithm*, Uspekhi Mat. Nauk **13** (1958), no. 4, 3–28.

[Lov09] L. Lovász, *Very large graphs*, Current developments in mathematics, 2008, Int. Press, Somerville, MA, 2009, pp. 67–128.

[Sch80] A. Schönhage, *Storage modification machines*, SIAM J. Comput. **9** (1980), no. 3, 490–508.

[Soa87] R. I. Soare, *Recursively enumerable sets and degrees*, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1987.

[Sut03] K. Sutner, *Cellular automata and intermediate degrees*, Theoret. Comput. Sci. **296** (2003), no. 2, 365–375.

[TM87] T. Toffoli and N. Margolus, *Cellular automata machines: a new environment for modeling*, MIT Press, Cambridge, MA, 1987.

[WN09] D. Woods and T. Neary, *The complexity of small universal Turing machines: a survey*, Theoret. Comput. Sci. **410** (2009), no. 4–5, 443–450.