View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

ELSEVIER

Theoretical Computer Science 181 (1997) 119–139

Theoretical Computer Science

Binary search and recursive graph problems

William I. Gasarch[a,*], Katia S. Guimarães[b]

[a]*Department of Computer Science and UMIACS, University of Maryland, College Park, MD 20742, USA*

[b]*Departamento de Informática, Univ. Federal de Pernambuco, CP 7851, Recife, PE, 50732-970 Brazil*

Abstract

A graph $G=(V,E)$ is *recursive* if every node of G has a finite number of neighbors, and both V and E are recursive (i.e., decidable). We examine the complexity of identifying the number of connected components of an infinite recursive graph, and other related problems, both when an upper bound to that value is given a priori or not. The problems that we deal with are unsolvable, but are recursive in some level of the arithmetic hierarchy. Our measure of the complexity of these problems is precise in two ways: the Turing degree of the oracle, and the number of queries to that oracle. Although they are in several different levels of the arithmetic hierarchy, all problems addressed have the same upper and lower bounds for the number of queries as the binary search problem, both in the bounded and in the unbounded case.

1. Introduction

A graph $G=(V,E)$ is *recursive* if every node of G has a finite number of neighbors, and both V and E are recursive (i.e., decidable). We examine the complexity of identifying the number of connected components of an infinite recursive graph, and several variations of this problem.

Recursive graph theory can be viewed as part of *Anil Nerode's Recursive Math Program*. He proposes looking at nonconstructive proofs in Recursive Mathematics and either making them constructive, or proving that it cannot be done. His notion of constructive is recursion theoretic. Various people have studied properties of recursive graphs. Bean [1] has studied colorings, Manaster and Rosenstein [12] have studied matchings, and Harel [9] has studied Hamiltonian paths. See [3] for more references.

This work follows the lines of [3, 4], which study the complexity of finding the chromatic number of a recursive graph both, when that number is a priori bounded above by a constant, and when it is not. In the present work we are concerned with the complexity of finding the number of connected components of a recursive graph in both cases.

* Corresponding author. E-mail: gasarch@cs.umd.edu.

0304-3975/97/$17.00 © 1997 — Elsevier Science B.V. All rights reserved
PII S0304-3975(96)00266-6

The problems that we deal with are unsolvable, but are recursive in some level of the arithmetic hierarchy. Our measure of the complexity of these problems is precise in two ways: the Turing degree of the oracle, and the number of queries to that oracle. We show that

1. Finding if a recursive graph has at most c connected components, for a fixed c, requires an oracle of Turing degree $\mathbf{0}''$ (i.e., Σ_2 or Π_2).
2. The number of components can be found with $\lceil \log(c+1) \rceil$ queries to $\mathbf{0}''$, but it cannot be found with $\lceil \log(c+1) \rceil - 1$ queries to *any* oracle, even a more powerful one.
3. Determining if a recursive graph has a finite number of components requires an oracle of Turing degree $\mathbf{0}'''$.
4. The set of graphs with a finite number of infinite components requires an oracle of Turing degree $\mathbf{0}''''$.
5. Allowing free queries to weaker oracles almost always does not lower the number of queries necessary to the more powerful oracle.

We also show that when no bound is set a priori, this problem is related to unbounded search in two ways:

1. If f is a nondecreasing recursive function, and $\sum_{i \geqslant 0} 2^{-f(i)} \leqslant 1$ is effectively computable, then the number of components of a recursive graph G_e, $nC(G_e)$, can be found with $f(nC(G_e))$ queries to $\mathbf{0}''$.
2. If G is an infinite recursive graph and there is a set X such that $nC(G)$ can be computed using $f(nC(G))$ queries to X, then $\sum_{0 \leqslant i} 2^{-f(i)} \leqslant 1$.

Part 2 above can be interpreted as a lower bound for finding $nC(G)$. That result follows from a generalization of Theorem 9 in [4], which allows us to conclude that part 2 also applies to a wide class of problems, including the problems of finding the number of finite components and finding the number of infinite components of an infinite recursive graph.

The rest of this paper is organized as follows. Section 2 presents definitions, notation, and known results. In Section 3 we show that finding the number of connected components of a recursive graph when that number is bounded by a constant requires an oracle of degree $\mathbf{0}''$, and that a binary search algorithm uses the minimal number of queries necessary. This result is tight in two ways: the lower bound on the number of queries holds even if a more powerful oracle is used, and no matter how many queries are used, the oracle must be of degree at least $\mathbf{0}''$. Also in Section 3, we show that the set of recursive graphs which have a finite number of components is Σ_3-complete. In Sections 4 and 5 we investigate the complexity of finding the number of finite components and the number of infinite components, respectively. In Section 6 we study whether or not the number of queries in each case can be reduced if we allow queries to weaker oracles for free. In Section 7 we describe the *unbounded search problem* and some relevant previous results. We also present the analysis of the complexity of identifying the number of connected components of a recursive graph when no upper bound to that number is set a priori. Section 8 is a study of whether or not the lower bound can be reduced if we allow free queries to weaker oracles. Section 9 contains a brief review of the paper.

2. Notation and definitions

All logarithms in this paper are base 2. $M_0, M_1, \ldots$ is an enumeration of all Turing machines, and $M_{e,s}$ denotes machine M_e running for at most s steps (stages). Let W_e denote the domain of M_e, and let $W_{e,s}$ be W_e after s stages, i.e., $W_{e,s} = \{0,1,2,\ldots,s\} \cap \{x \mid M_{e,s}(x)\downarrow\}$. $\mathbf{N}$ represents the set of natural numbers. K represents the halting set. *FIN* represents the set of indices of functions that are only defined finitely often, i.e., $\{e \mid W_e \text{ is finite}\}$. *TOT* represents the set of indices of functions that are defined everywhere, i.e., $\{e \mid W_e = \mathbf{N}\}$. *COF* represents the set of indices of cofinite functions, i.e., $\{e \mid \mathbf{N} - W_e \text{ is finite}\}$. It is shown in [14] that $\overline{K}$ is Π_1-complete, *FIN* is Σ_2-complete, *TOT* is Π_2-complete, *COF* is Σ_3-complete, and $\overline{COF}$ is Π_3-complete. We will use these results later to prove that other sets are in the same classes.

Let $M_0^A, M_1^A, \ldots$ be an enumeration of all oracle Turing machines that are recursive in A (i.e., $M_i^A \leqslant_T A$). Then, $A' = \{e \mid M_e^A(e)\downarrow\}$. $\phi' = K = \{e \mid M_e(e)\downarrow\}$, $\phi'' = \{e \mid M_e^{\phi'}(e)\downarrow\}$, $\phi''' = \{e \mid M_e^{\phi''}(e)\downarrow\}, \ldots, \phi^{(i)} = \{e \mid M_e^{\phi^{(i-1)}}(e)\downarrow\}$. We say that an oracle has Turing degree $\mathbf{0}^{(i)}$ if it is recursive in $\phi^{(i)}$.

Recall that a graph $G = (V,E)$ is *recursive* if every node of G has a finite number of neighbors and both $V \subseteq \mathbf{N}$, and $E \subseteq [\mathbf{N}]^2$ are recursive. A graph $G = (V,E)$ is *highly recursive* if G is recursive and the function that produces all the neighbors of a given node is recursive. Throughout this paper, all graphs are supposed to be undirected.

We represent recursive and highly recursive graphs by the Turing machines that determine their vertex and edge sets. An index for a recursive graph is an ordered pair in which the first and second components are indices for Turing machines which decide the vertex set and the edge set, respectively. We denote a fixed recursive pairing bijection from $\mathbf{N} \times \mathbf{N}$ onto $\mathbf{N}$ by $[e_1, e_2]$, so the symbol '$[x,y]$' is a natural number that corresponds to the ordered pair (x,y). If M_{e_1} and M_{e_2} are total, then the number $e = [e_1, e_2]$ *determines the recursive graph* $G_e^{\mathrm{r}} = (V,E)$, where $V = \{x \mid M_{e_1}(x) = 1\}$, and $E = \{[x,y] \mid x, y \in V \text{ and } M_{e_2}([x,y]) = 1\}$. If M_{e_1} or M_{e_2} is not total, then e does not determine a recursive graph. A number $e = [e_1, e_2]$ *determines a highly recursive graph* if M_{e_1} and M_{e_2} are total, and when M_{e_2} is interpreted as a mapping from $\mathbf{N}$ to finite subsets of $\mathbf{N}$, if $M_{e_2}(x) = Y$ then for all $y \in Y$, $(x,y) \in E$ (i.e., Y is the set of vertices adjacent to x). If e determines a highly recursive graph, then the *highly recursive graph determined by* e is $G_e^{\mathrm{hr}} = (V,E)$, where $V = \{x \mid M_{e_1}(x) = 1\}$, and $E = \{[x,y] \mid x, y \in V \text{ and } x \in M_{e_2}(y)\}$.

Let $e = [e_1, e_2]$ be a number that determines a recursive graph. We define the *approximation to* G_e^{r} *by stage* s ($G_{e,s}^{\mathrm{r}}$) to be the subgraph of G_e^{r} formed by taking all nodes in the set $\{0,1,2,\ldots,s\}$ that are in the graph and connecting them as they are connected in the graph. Formally, $G_{e,s}^{\mathrm{r}} = (V_s, E_s)$, where $V_s = \{0,1,2,\ldots,s\} \cap \{x \mid M_{e_1}(x) = 1\}$, and $E_s = [V_s]^2 \cap \{[x,y] \mid M_{e_2}([x,y]) = 1\}$.

The *approximation to* G_e^{hr} *by stage* s ($G_{e,s}^{\mathrm{hr}}$) is defined inductively. $G_{e,0}^{\mathrm{hr}} = (\{0\} \cap V, \emptyset)$. For $s > 0$, $G_{e,s}^{\mathrm{hr}}$ is defined as the subgraph of G_e^{hr} formed by $G_{e,s-1}^{\mathrm{hr}}$ and all of its neighbors (together with the corresponding edges), plus vertex s, if it is in G_e^{hr}.

Formally, if $G^{hr}_{e,s-1} = (V_{s-1}, E_{s-1})$, then $G^{hr}_{e,s} = (V_s, E_s)$, where $V_s = V_{s-1} \cup \{x \mid x \in M_{e_2}(y)$, for some $y \in V_{s-1}\} \cup (\{s\} \cap V)$, and $E_s = [V_s]^2 \cap \{[x,y] \mid x \in M_{e_2}(y)\}$.

We denote the existence of a path between nodes x_i and x_j in a graph by $x_i \bowtie x_j$.

Let $G_{e,s}$ have connected components $c_{(1,s)}, c_{(2,s)}, \ldots, c_{(m,s)}$. Then, let $|c_{(1,s)}|, |c_{(2,s)}|, \ldots, |c_{(m,s)}|$ represent the number of nodes in $c_{(1,s)}, c_{(2,s)}, \ldots, c_{(m,s)}$, respectively. For each i, $1 \leqslant i \leqslant m$, and some $t \geqslant s$, $c_{(c_i,t)}$ represents the component of $G_{e,t}$ which contains all nodes of $c_{(i,s)}$. (Hence, $|c_{(c_i,t)}|$ represents the number of nodes in the component of $G_{e,t}$ which contains all nodes of $c_{(i,s)}$.)

Let $nC(G_e)$, $fC(G_e)$, and $iC(G_e)$ denote, respectively, the number of connected components, the number of finite components, and the number of infinite components of a recursive graph G_e. For any $c \geqslant 1$, we define three functions:

$$nC_c(G) = \begin{cases} nC(G) & \text{if } 0 \leqslant nC(G) \leqslant c, \\ c & \text{otherwise,} \end{cases}$$

$$fC_c(G) = \begin{cases} fC(G) & \text{if } 0 \leqslant fC(G) \leqslant c, \\ c & \text{otherwise,} \end{cases}$$

$$iC_c(G) = \begin{cases} iC(G) & \text{if } 0 \leqslant iC(G) \leqslant c, \\ c & \text{otherwise.} \end{cases}$$

Let $\gamma(G)$ be a function from graphs into the naturals, such as nC, fC, or iC. Then the partial function $\gamma_n(G)$ is defined as follows.

$$\gamma_n(G) = \begin{cases} \gamma(G) & \text{if } 0 \leqslant \gamma(G) \leqslant n, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

If A and B are sets, then $A \oplus B$ is the set $\{2x \mid x \in A\} \cup \{2x+1 \mid x \in B\}$. An oracle machine using oracle $A \oplus B$ can ask questions to either A or B. When an even number is queried, we say that a query to A has been made, and when an odd number is queried, we say that a query to B has been made. If f and g are functions, $f \leqslant_T g$ means that f is Turing-reducible to g. Let g be a total function and $n \geqslant 0$ be a number. A partial function f is in $FQ(n,g)$ if $f \leqslant_T g$ via an oracle Turing machine which uses oracle g, and never makes more than n queries. If g is the characteristic function of a set A, then we use the notation $FQ(n,A)$. If B is a set, then f is in $FQ^B(n,A)$ if $f \leqslant_T A \oplus B$ via an oracle Turing machine that, when using oracle $A \oplus B$, never asks more than n queries to A (although it may ask many queries to B).

Remark. The definition of $FQ(n,A)$ still makes sense if 'n' is replaced by a function of the input. The statement '$nC(G) \in FQ(f(nC(G)),X)$' will mean that computing the number of components of graph G can be done with $f(nC(G))$ queries to X, assuming $nC(G)$ is defined.

Let A be a set of natural numbers. The function χ_A, is the characteristic function of A. We identify a set with its characteristic function.

Let A be any set and $n \geqslant 1$ be a number. We define two functions:

$$\#_n^A(x_1, x_2, \ldots, x_n) = |\{i \mid x_i \in A\}|,$$

$$F_n^A(x_1, x_2, \ldots, x_n) = [\chi_A(x_1), \chi_A(x_2), \ldots, \chi_A(x_n)] .$$

A real number r is *effectively computable* if there is a fixed algorithm that takes a rational number y as input and determines if $x < y$.

Let D be a set of natural numbers. A *binary prefix code* for D is a bijection from D onto a subset of $\{0,1\}^*$ such that for any two strings x and y in the range of the bijection, x is not a prefix of y.

A function f from N to N *satisfies Kraft's inequality* if $\sum_{i \geqslant 0} 2^{-f(i)} \leqslant 1$.

In this paper we are not concerned with the problem of determining if a number is an index of a recursive graph. We implicitly assume that the indices are valid. Finding out if e determines either a recursive or a highly recursive graph is Π_2-complete. A *promise problem* is a set A and a function f, where domain$(f) = A$. A *solution* to a promise problem (A, f) is a function g such that $\forall x \in A$, $g(x) = f(x)$. A promise problem (A, f) *is in class* $\mathscr{A}$ if it has a solution g, and $g \in \mathscr{A}$. $X \leqslant_{\mathrm{m}} (A, f)$ if for *all* solutions g to (A, f), $X \leqslant_{\mathrm{m}} g$. Throughout this paper we deal with promise problems with respect to indices.

3. Number of connected components

In this section we show that finding if a recursive graph has at most c connected components, for a fixed constant c, requires an oracle of Turing degree $\mathbf{0}''$. We also show that $\lceil \log(c+1) \rceil$ queries is a tight bound on the number of queries necessary to solve the problem, even if a more powerful oracle is used. We finally show that determining whether a recursive graph has a finite number of components requires an oracle of Turing degree $\mathbf{0}'''$. All results in this section hold for recursive and highly recursive graphs.

Theorem 1. *For any natural* $k \geqslant 1$, $NC_k = \{e \mid G_e$ *has at most* k *connected components*$\}$ *is* Π_2*-complete.*

Proof. We can rewrite NC_k as $NC_k = \{e \mid \forall x_1, x_2, \ldots, x_{k+1}\ \exists s, i, j\ [x_i \bowtie x_j \text{ in } G_{e,s}]\}$.

The function that, given e and s, checks whether $x_i \bowtie x_j$ in $G_{e,s}$ is recursive, and is defined when G_e is recursive. Hence, NC_k is in Π_2.

We show that NC_k is Π_2-hard by showing that $TOT \leqslant_{\mathrm{m}} NC_k$ (i.e., given x, we construct a recursive graph $G(x) = G$ such that $G \in NC_k$ iff M_x is total). The idea is to make several infinite components grow simultaneously, and, at every stage s, to connect the components corresponding to elements i and $i+1$ iff all numbers $j \leqslant i$ are in $W_{x,s}$. The construction proceeds in stages. G_s is the graph at the end of stage s. G is the limit graph $\lim_{s \to \infty} G_s$. The vertices of G are identified by pairs of naturals.

Construction

Stage 0. $G_0 = (\{(0,0)\}, \emptyset)$.

Stage $s+1$. Let $G_s = (V_s, E_s)$.

$V_{s+1} = V_s \cup \{(i, s+1) \mid 0 \leqslant i \leqslant s\} \cup \{(s+1, i) \mid 0 \leqslant i \leqslant s+1\}$.

Let f be the first natural such that $f \notin W_{x,s}$.

$$E_{s+1} = E_s \cup \{[(i,s),(i,s+1)] \mid 0 \leqslant i \leqslant s\}$$
$$\cup \{[(s+1,i),(s+1,i+1)] \mid 0 \leqslant i \leqslant s\}$$
$$\cup \{[(i,s+1),(i+1,s+1)] \mid 0 \leqslant i \leqslant f\}.$$

End of Construction

Suppose M_x is total. Then, for each element i, there is a stage s_o when $j \in W_{x,s_o}\ \forall j \leqslant i$. Hence, the components corresponding to the first i elements will be interconnected by edges of type $[(i,s+1),(i+1,s+1)]$. Thus, at the limit, graph G will have exactly one component. Now, suppose that M_x is not total, and let i_o be the first element such that $i_o \notin W_x$. Then, the components corresponding to elements i_o+1, $i_o+2, \ldots$ will represent distinct connected components. Thus, at the limit, graph G will have an infinite number of components. □

Theorem 1 shows that determining the number of components of a recursive graph requires an oracle of degree at least $\mathbf{0}''$. The next theorem gives an exact bound on how many queries to ϕ'' are required to actually find $nC(G_e)$, if a bound to that number is given. We use the following results, which were proved in [5].

Lemma 2. *If A and X are sets, A is nonrecursive, and n is any number, then $F_{2^n}^A \notin FQ(n,X)$. (i.e., membership in A for 2^n elements cannot be decided using n (or less) queries to any oracle.)*

Lemma 3. *For any numbers $x_1, \ldots, x_n$, given the value of $|K \cap \{x_1, \ldots, x_n\}|$, the value of $F_n^K(x_1, \ldots, x_n)$ can be computed.*

Theorem 4. *For any $c \geqslant 1$, function nC_c is in $FQ(\lceil \log(c+1) \rceil, \phi'')$, but for any set X, $nC_c \notin FQ(\lceil \log(c+1) \rceil - 1, X)$.*

Proof. Using theorem 1 and a binary search on $[0,c]$ for the proper number of components, we obtain that $nC_c \in FQ(\lceil \log(c+1) \rceil, \phi'')$.

Let X be any set. To establish that $nC_c \notin FQ(\lceil \log(c+1) \rceil - 1, X)$, we show that otherwise we have $F_{2^n}^K \in FQ(n,X)$ (where $n = \lceil \log(c+1) \rceil - 1$), which contradicts Lemma 2. We describe an algorithm to determine $F_{2^n}^K(x_1, \ldots, x_{2^n})$ that will use only one call to the function nC_c; hence, if nC_c is in $FQ(n,X)$, then the function $F_{2^n}^K$ is in $FQ(n,X)$.

For $i = 1, \ldots, 2^n$, let

$$G_i = \begin{cases} (\{t\}, \emptyset) & \text{if } x_i \in K, \text{ where } x_i \notin W_{x_i,t-1},\ x_i \in W_{x_i,t}, \\ (\emptyset, \emptyset) & \text{if } x_i \notin K. \end{cases}$$

Let G_e be the disjoint union (union in a way so that all vertices are distinct) of $G_1, G_2, \ldots, G_{2^n}$. Then $nC(G_e) = |K \cap \{x_1, x_2, \ldots, x_{2^n}\}| \leqslant 2^n \leqslant c$. By Lemma 3, $F_{2^n}^K$ can be computed from a single query to nC_c. □

We now show that the set of recursive graphs that have a finite number of components is Σ_3-complete.

Theorem 5. *$NC_f = \{e \mid G_e$ has a finite number of components$\}$ is Σ_3-complete.*

Proof. We can rewrite NC_f as $NC_f = \{e \mid \exists k\ \forall x_1, x_2, \ldots, x_{k+1}\ \exists s, i, j\ [x_i \bowtie x_j \text{ in } G_{e,s}]\}$.

We prove that NC_f is Σ_3-hard by showing that $COF \leqslant_m NC_f$. The idea is to make several infinite components grow simultaneously, and at every stage, if a new element i is added to W_x, we connect the component corresponding to i to the next $(i+1)$. As we did before, the construction proceeds in stages. G_s is the graph at the end of stage s. G is the graph $\lim_{s\to\infty} G_s$.

Construction

Stage 0. Let $G_0 = (\{(0,0)\}, \emptyset)$.

Stage $s+1$. Let $G_s = (V_s, E_s)$.

$V_{s+1} = V_s \cup \{(i, s+1) \mid 0 \leqslant i \leqslant s\} \cup \{(s+1, i) \mid 0 \leqslant i \leqslant s+1\}$.

$E_{s+1} = E_s \cup \{[(i,s),(i,s+1)] \mid 0 \leqslant i \leqslant s\}$
$\cup \{[(s+1,i),(s+1,i+1)] \mid 0 \leqslant i \leqslant s\}$
$\cup \{[(i,s+1),(i+1,s+1)] \mid i \in W_{x,s}\}$.

End of Construction

We can easily show that W_x is cofinite iff G has a finite number of components. □

4. Number of finite components

In this section we show that determining if the number of finite components of a recursive graph is within a given upper bound requires an oracle of Turing degree $\mathbf{0}'$ if the graph is highly recursive, and requires an oracle of Turing degree $\mathbf{0}''$ if the graph is recursive. We show that $\lceil \log(c+1) \rceil$ queries is a tight bound on the number of queries necessary to solve the problem, even if a more powerful oracle is used. We finally show that determining whether a recursive graph has a finite number of finite components requires an oracle of Turing degree $\mathbf{0}''$, if the graph is highly recursive, and of Turing degree $\mathbf{0}'''$, if the graph is recursive.

Theorem 6. *For any natural number $k \geqslant 0$, the set $NFC_k = \{e \mid G_e$ has at most k finite components$\}$ is Π_1-complete for highly recursive graphs, and is Π_2-complete for recursive graphs.*

Proof. To show that NFC_k^{hr} is in Π_1, and that NFC_k^{r} is in Π_2, we rewrite them as: $NFC_k^{hr} = \{e \mid \forall s$ at most k components c_i, $1 \leqslant i \leqslant k$, will not have $|c_{(i,s)}| < |c_{(c_i,s+1)}|\}$, and $NFC_k^{r} = \{e \mid \forall s \exists t$ at most k components c_i, $1 \leqslant i \leqslant k$, will not have $|c_{(i,s)}| < |c_{(c_i,t)}|\}$.

To prove that NFC_k^{hr} is Π_1-hard we show that $\overline{K} \leqslant_{\mathrm{m}} NFC_k^{\mathrm{hr}}$. The idea is to add a new vertex to the graph at each stage s, and to connect it to the previous graph only if $M_x(x)$ has not halted at stage s.

Construction
Stage 0. Let $G_0 = (\{0\}, \phi)$.
Stage $s+1$. Let $G_s = (V_s, E_s)$. $V_{s+1} = V_s \cup \{s+1\}$.
There are two cases:
1. If $x \notin W_{x,s}$, then $E_{s+1} = E_s \cup \{(s, s+1)\}$
2. If $x \in W_{x,s}$, then $E_{s+1} = E_s$.

End of Construction

It is easy to see that if $x \notin W_x$, then G has no finite components, and if $x \in W_x$, then G has an infinite number of finite components.

Finally, to prove that NFC_k^{r} is Π_2-hard we show that $TOT \leqslant_{\mathrm{m}} NFC_k^{\mathrm{r}}$. The idea is to always add a new node to the graph, but to connect a node only when the corresponding element and all of its predecessors are already in W_x.

Construction
Stage 0. Let $G_0 = (\{0\}, \phi)$. Set $z := 0$.
Stage $s+1$. Let $G_s = (V_s, E_s)$. $V_{s+1} = V_s \cup \{s+1\}$.
Let z be the highest numbered vertex such that $\forall i \leqslant z$, i is connected to vertex 0 in G_s, and let w be the highest numbered element such that $\forall j \leqslant w$, $j \in W_{x,s}$.
There are two cases:
1. If $w+1 \notin W_{x,s+1}$, then $E_{s+1} = E_s$.
2. If $w+1 \in W_{x,s+1}$, then $E_{s+1} = E_s \cup \{(z, s+1), (s+1, z+1)\})$.

End of Construction

We can easily show that if M_x is total, then G has exactly 1 (infinite) component, and if M_x is not total, then G has an infinite number of finite components. □

Theorem 7. *Let* $c \geqslant 0$ *be any number.*
- $fC_c(G_e^{\mathrm{r}}) \in FQ(\lceil \log(c+1) \rceil, \phi'')$, *and* $fC_c(G_e^{\mathrm{hr}}) \in FQ(\lceil \log(c+1) \rceil, \phi')$.
- *For any set* X, $fC_c \notin FQ(\lceil \log(c+1) \rceil - 1, X)$.

Proof. Theorem 6 and binary search can easily give us the upper bounds. The same proof that we used for the lower bound in Theorem 4 applies here. □

The following theorem shows that determining whether or not a recursive graph has a finite number of finite components requires an oracle of Turing degree $\mathbf{0}''$ or $\mathbf{0}'''$, depending on the kind of recursive graph at hand.

Theorem 8. $NFC_{\mathrm{f}} = \{e \mid G_e$ *has a finite number of finite components*$\}$ *is* Σ_2*-complete for highly recursive graphs, and is* Σ_3*-complete for recursive graphs.*

Proof. We can rewrite NFC_f^{hr} and NFC_f^{r} as $NFC_f^{hr} = \{e \mid \exists k\, \forall s$ at most k components c_i, $1 \leqslant i \leqslant k$, will not have $|c_{(i,s)}| < |c_{(c_i,s+1)}|\}$, and $NFC_f^{r} = \{e \mid \exists k \forall s\, \exists t$ at most k components c_i, $1 \leqslant i \leqslant k$, will not have $|c_{(i,s)}| < |c_{(c_i,t)}|\}$.

To show that NFC_f^{hr} is Σ_2-hard we show that $FIN \leqslant_m NFC_f^{hr}$. The idea is to make several infinite components grow simultaneously, and if at stage s element i is added to W_x, then stop augmenting the component which corresponds to i.

Construction

Stage 0. Let $G_0 = (\{(0,0)\}, \emptyset)$.

Stage $s+1$. Let $G_s = (V_s, E_s)$.

$V_{s+1} = V_s \cup \{(i, s+1), \forall i,\ 0 \leqslant i \leqslant s,\ i \notin W_{x,s}\} \cup \{(s+1, i), \forall i,\ 0 \leqslant i \leqslant s+1\}$.

$E_{s+1} = E_s \cup \{[(i,s),(i,s+1)]$, such that $0 \leqslant i \leqslant s,\ i \notin W_{x,s}\}$
$\cup \{[(s+1,i),(s+1,i+1)]\ \forall i,\ 0 \leqslant i \leqslant s\}$.

End of Construction

It is easy to see that W_x is finite iff G has a finite number of finite components.

To show that NFC_f^{r} is Σ_3-hard we show that $COF \leqslant_m NFC_f^{r}$. The idea is to create new components, but to keep each component i finite until stage s when $i \in W_{x,s}$. Then, let component i grow forever.

Construction

Stage 0. Let $G_0 = (\{(0,0)\}, \emptyset)$.

Stage $s+1$. Let $G_s = (V_s, E_s)$.

$V_{s+1} = V_s \cup \{(s+1, 0)\} \cup \{(i, s+1),\ \forall\, 0 \leqslant i \leqslant s+1,\ i \in W_{x,s+1}\}$.

$E_{s+1} = E_s \cup \{[(i,s),(i,s+1)]$, such that $i \in W_{x,s}\}$
$\cup \{[(i,0),(i,s+1)]$, such that $i \in W_{x,s+1}$ but $i \notin W_{x,s}\}$.

End of Construction

It is easy to see that W_x is cofinite iff G has a finite number of finite components. □

5. Number of infinite components

We now look into the problem of determining the number of infinite components of a recursive graph. We first show that determining if the number of infinite components of a recursive graph is within a given upper bound requires an oracle of Turing degree $\mathbf{0}''$ if the graph is highly recursive, and requires an oracle of Turing degree $\mathbf{0}'''$ if the graph is recursive. We show that $\lceil \log(c+1) \rceil$ queries is a tight bound on the number of queries necessary to solve the problem, even if a more powerful oracle is used. We finally show that determining whether a recursive graph has a finite number of infinite components requires an oracle of Turing degree $\mathbf{0}'''$ if the graph is highly recursive, and of Turing degree $\mathbf{0}''''$ if the graph is recursive.

Theorem 9. *For any natural number $k \geqslant 0$, the set $NIC_k = \{e \mid G_e$ has at most k infinite components$\}$ is Π_2-complete for highly recursive graphs, and is Π_3-complete for recursive graphs.*

Proof. We can rewrite NIC_k^{hr} and NIC_k^{r} as:

$$NIC_k^{\mathrm{hr}} = \{e \mid \forall s\, \exists t\, [t > s, \text{ and at most } k \text{ of the components } c_i,\ 1 \leqslant i \leqslant k, \text{ of } G_{e,s} \text{ will have } |c_{(c_i,t+1)}| > |c_{(c_i,t)}|]\},$$

and

$$NIC_k^{\mathrm{r}} = \{e \mid \forall s\, \exists t_1\, \forall t_2 [t_2 > t_1 \Rightarrow \text{at most } k \text{ of the components } c_i,\ 1 \leqslant i \leqslant k, \text{ of } G_{e,s} \text{ will have } |c_{(c_i,t_2)}| > |c_{(c_i,t_1)}|]\}.$$

The proof used in Theorem 1 to show that $TOT \leqslant_{\mathrm{m}} NC_k$ can also be used here to show that $TOT \leqslant_{\mathrm{m}} NIC_k^{\mathrm{hr}}$, and hence that NIC_k^{hr} is Π_2-hard. To show that NIC_k^{r} is Π_3-hard, we show that $\overline{COF} \leqslant_{\mathrm{m}} NIC_k^{\mathrm{r}}$. The idea is to create one finite component for each element i. In time, component i will have added to it as many vertices as the number of consecutive subsequent elements that are in W_x.

Construction
Stage 0. Let $G_0 = (\{(0,0)\}, \emptyset)$.
Stage $s+1$. Let $G_s = (V_s, E_s)$.
$V_{s+1} = V_s \cup \{(i, s+1), \forall i,\ 0 \leqslant i \leqslant s\} \cup \{(s+1, i), \forall i,\ 0 \leqslant i \leqslant s+1\}$.
$E_{s+1} = \{[(k,i),(k,i+1)],$ such that $0 \leqslant i,\ k \leqslant s,\ i \in W_{x,s}\}$.
End of Construction

If $x \in \overline{COF}$, then for each element i there is an element $j > i$ such that $j \notin W_x$. Hence, the component corresponding to i will be finite. Overall there will be no infinite components in the graph. If $x \notin \overline{COF}$, then there is an element i_0 such that for all $j > i_0$, $j \in W_x$. Hence, all components which correspond to elements greater than i_0 will be infinite. □

Theorem 10. *Let $c \geqslant 0$ be any number.*
- *$iC_c(G_e^{\mathrm{r}}) \in FQ(\lceil \log(c+1) \rceil, \phi''')$, and $iC_c(G_e^{\mathrm{hr}}) \in FQ(\lceil \log(c+1) \rceil, \phi'')$.*
- *For any set X, $iC_c \notin FQ(\lceil \log(c+i) \rceil - 1, X)$.*

Proof. Again, binary search with the help of Theorem 9 will give us the upper bound. The proof of the lower bound is similar to the one in Theorem 4. The algorithm here is as follows.

For $i = 1, 2, \ldots, 2^n$, let

$$G_i = \begin{cases} (\{j \mid j \geqslant t\}, \{(j, j+1) \mid j \geqslant t\}) & \text{if } x_i \notin W_{x_i,t-1} \text{ but } x_i \in W_{x_i,t}, \\ (\emptyset, \emptyset) & \text{if } x_i \notin W_{x_i}. \end{cases}$$

Let G_e be the disjoint union of $G_1,\ldots,G_{2^n}$. Then $iC(G_e)=|K\cap\{x_1,\ldots,x_{2^n}\}|\leqslant 2^n\leqslant c$. Again by Lemma 3, $F_{2^n}^K$ can be computed from a single query to iC_c. □

The following theorem shows that determining whether or not a recursive graph has a finite number of infinite components requires an oracle of Turing degree $\mathbf{0}'''$ or $\mathbf{0}''''$, depending on the kind of recursive graph at hand.

Theorem 11. *$NIC_{\mathrm{f}}=\{e\mid G_e$ has a finite number of infinite components$\}$ is Σ_3-complete for highly recursive graphs, and is Σ_4-complete for recursive graphs.*

Proof. We can rewrite $NIC_{\mathrm{f}}^{\mathrm{hr}}$ and $NIC_{\mathrm{f}}^{\mathrm{r}}$ as:

$$NIC_{\mathrm{f}}^{\mathrm{hr}}=\{e\mid \exists k\,\forall s\,\exists t\,[t>s,\text{ and at most } k \text{ components } c_i,\ 1\leqslant i\leqslant k,\text{ of } G_{e,s} \text{ will have } |c_{(i,t)}|<|c_{(c_i,t+1)}|]\},$$

and

$$NIC_{\mathrm{f}}^{\mathrm{r}}=\{e\mid \exists k\,\forall s\,\exists t_1\,\forall t_2\,[t_2>t_1\Rightarrow \text{ at most } k \text{ of the components } c_i,\ 1\leqslant i\leqslant k,\text{ of } G_{e,s} \text{ will have } |c_{(i,t_1)}|<|c_{(c_i,t_2)}|]\}.$$

The proof of the lower bound in Theorem 5 can be used to show that $COF\leqslant_{\mathrm{m}} NIC_{\mathrm{f}}^{\mathrm{hr}}$, hence $NIC_{\mathrm{f}}^{\mathrm{hr}}$ is Σ_3-hard. To prove that $NIC_{\mathrm{f}}^{\mathrm{r}}$ is Σ_4-hard we use the following claim.

Claim. $S=\{e\mid W_e\not\in COF\}$ is Σ_4-hard.

Proof. Let A be any set in Σ_4. Assume that $A=\{a\mid \exists b, R^{\Pi_3}(a,b)\}$, for some property $R^{\Pi_3}(a,b)$ in Π_3. Since $\overline{COF}$ is Π_3-complete, there is a recursive function f_R such that $R^{\Pi_3}(a,b)$ holds iff $f_R(a,b)\in\overline{COF}$. We use f_R to construct an algorithm for A.

Algorithm for A

1. Input (a);
2. Create a Turing machine to do the following:
   1. Input (x);
   2. For $b=1,2,3,\ldots$ do:
      If $f_R(a,b)=x$ then HALT;
3. Let e be an index for the machine constructed in step 2. Return $(M_S(e))$.

End of Algorithm

The Turing machine M_e created in step 2 of the algorithm halts precisely on inputs x for which there is a b such that $x=f_R(a,b)$.

$a\in A\Rightarrow \exists b,\ R^{\Pi_3}(a,b)$ holds $\Rightarrow \exists b, f_R(a,b)\in\overline{COF}$. Since $f_R(a,b)\in W_e$, $e\in S$.

$a\notin A\Rightarrow \forall b,\ R^{\Pi_3}(a,b)$ does not hold $\Rightarrow \forall b,\ f_R(a,b)\in COF\Rightarrow W_e\subseteq COF\Rightarrow e\notin S$.

End of Proof of Claim

Proof of Theorem 11 (*Conclusion*). The last thing we need to show is that $S \leqslant_m NIC_f^r$. Given an input e, we construct a graph G such that $G \in NIC_f^r$ iff $e \in S$. The idea is to grow one infinite component for each element until some element is accepted in W_e. (If W_e is empty, the process continues forever, and in the limit G will have an infinite number of infinite components.) Then, for each element z accepted in W_e and each stage s, we create a new set of vertices and edges (which we call *subgraph* $G(z,s)$), in which vertices representing stages $i = 1,\ldots,s$, (z,s,i), are connected to the next neighbor, $(z,s,i+1)$, iff $i \in W_{z',s}$ for all $z' \in W_{e,s}$, with $z' < z$. The formal construction follows.

Construction
Stage 0. Let $G_0 = (\{(0,0)\}, \emptyset)$.
Stage $s+1$. Let $G_s = (V_s, E_s)$.
If $W_{e,s+1}$ is empty, then
$V_{s+1} = V_s \cup \{(i, s+1), \forall i,\ 0 \leqslant i \leqslant s\} \cup \{(s+1, i), \forall i,\ 0 \leqslant i \leqslant s+1\}$;
$E_{s+1} = \{[(k,i),\ (k,i+1)],$ such that $0 \leqslant i,\ k \leqslant s\}$
else
$V_{s+1} = V_s \cup \{(z, s+1, i),\ \forall i,\ 0 \leqslant i \leqslant s+1,\ z \in W_{e,s+1},$ and $z \leqslant s\}$;
$E_{s+1} = \{[(z,s+1,i),(z,s+1,i+1)],$ such that $z \in W_{e,s+1}, i \in \bigcap_{z' \in W_{e,s}, z' < z} W_{z',s}\}$.
End of Construction

If W_e is empty, then ($e \notin S$) the '*then*' part of the construction will always be followed, generating one infinite component for each natural number.

If W_e is not empty, then eventually the construction will start following the '*else*' part. If there is some $x \in W_e$ with $x \notin COF$, then let t be such that $x \in W_{e,t}$. For all $z' > z$, and all stages $s > t$, the subgraphs $G(z',s)$ of G will have no infinite components. Hence, the number of infinite components of G is finite. On the other hand, if for all $x \in W_e$, $x \in COF$, then for all x there is an element i_x such that $i > i_x \Rightarrow i \in W_x$. Hence, for every $z \in W_{e,s}$ and all $i \geqslant I_z$, where I_z is the maximum over all i_z for $z \in (W_e \cap \{1,\ldots,z\})$, edge $[(z,s,i),(z,s,i+1)] \in G$. Hence, every $z \in W_e$ will in the limit generate an infinite number of subgraphs $G(z,s)$ each of which containing an infinite component. □

6. Lower bounds on mixed queries

We have seen that $\lceil \log(c+1) \rceil$ queries are required to compute nC_c, fC_c, and iC_c. One could ask if perhaps that number could be reduced if we allowed some help from weaker oracles. In this section we show that in most cases free queries to weaker oracles do not help, and when they do help the gain is very small.

Throughout this section we will use the following lemma, proven in [11].

Lemma 12 (Kummer [11]). *For any sets* A, Y, $\#_{c-1}^A \in FQ^Y(\lceil \log c \rceil - 1, X) \Rightarrow A \leqslant_T Y$.

6.1. Number of components

We have shown before (Theorem 4) that finding the value of $nC_c(G_e)$ requires exactly $\lceil \log(c+1) \rceil$ queries to ϕ''. Next we show that if queries to a weaker oracle are allowed for free, then the number of queries to ϕ'' can be slightly reduced.

Theorem 13. *For any* $c \geqslant 0$, $nC_c \in FQ^K(\lceil \log c \rceil, \phi'')$. *For any sets* X, Y, $nC_c \notin FQ^Y(\lceil \log c \rceil - 1, X)$, *unless* $\phi'' \leqslant_T Y$.

Proof. Consider a Turing machine M_e that inputs $G = (V,E)$, then asks for $v = 1,2,3,\ldots$ whether or not v is in V, and halts when it gets a positive answer. Since G is recursive, with one query to K, asking whether e is in K, we eliminate the case of an empty graph (0 components). Binary search between $[1,c]$ using Theorem 1 will find the proper value for nC_c in $\lceil \log c \rceil$ queries to ϕ''.

Assume that $nC_c \in FQ^Y(\lceil \log c \rceil - 1, X)$, for some set X. We show that we also have $\#_{c-1}^{\phi''} \in FQ(1, nC_c)$. (Hence, $\#_{c-1}^{\phi''} \in FQ^Y(\lceil \log c \rceil - 1, X)$, and by Lemma 12 we have that $\phi'' \leqslant_T Y$.) Since $\phi'' \leqslant_T TOT$, for this lower bound we use TOT instead of ϕ''. We describe an algorithm for $\#_{c-1}^{TOT}$ that asks only one query to nC_c. The idea is to construct one highly recursive graph G_i, $1 \leqslant i \leqslant c-1$, corresponding to each of the $c-1$ input machines in a way such that graph G_i will have 1 component if $M_{x_i} \in TOT$, and 2 components if $M_{x_i} \notin TOT$. This can be done by keeping in each graph G_i two components which will be connected only when the next consecutive element is accepted by M_{x_i}. In the limit, G_i will have only one component iff M_{x_i} is in TOT. Let graph G_e be obtained from the disjoint union of all G_i's. Notice that G_e has $c-1 \leqslant nC(G_e) \leqslant 2(c-1)$ components. Let $G_{e'}$ be obtained from G_e by connecting a new vertex v to one vertex in each of the old graphs G_i. Now we have $1 \leqslant nC(G_{e'}) \leqslant c$, and by construction of $G_{e'}$, $\#_{c-1}^{TOT} = c - nC(G_{e'})$.

We now formalize the above intuitive description. In the formal construction, the vertices of the graphs G_i are represented by triplets, where the first coordinate identifies the corresponding graph. The only use for that is to allow for an easy identification of vertices in distinct components of G_e. Formally, the algorithm is as follows.

Algorithm for $\#_{c-1}^{TOT}$

1. Input $(x_1, x_2, \ldots, x_{c-1})$;
2. For $i = 1,2,\ldots,c-1$, let G_i be constructed in stages:

Stage 0. Let $G_{i,0} := (\{(i,0,0),(i,1,0)\}, \emptyset)$; $j := 0$;

Stage $s+1$. Let $G_{i,s} = (V_{i,s}, E_{i,s})$.

If $j \in W_{x_i, s+1}$ then begin

$V_{i,s+1} = V_{i,s} \cup \{(i,j+1,s+1),(i,j+2,s+1)\}$;

$E_{i,s+1} = E_{i,s} \cup \{[(i,j,s),(i,j+1,s)],[(i,j+1,s),(i,j+1,s+1)]\}$;

$j := j+1$

end

else begin

$V_{i,s+1} = V_{i,s} \cup \{(i,j,s+1),(i,j+1,s+1)\}$;
$E_{i,s+1} = E_{i,s} \cup \{[(i,j,s),(i,j,s+1)], [(i,j+1,s),(i,j+1,s+1)]\}$
end;

3. Let G_e be constructed by the union over all the G_i's plus one extra vertex $(0,0,0)$, and $c-1$ new edges $[(0,0,0)(i,0,0)]$, $1 \leqslant i \leqslant c-1$;
4. Return $(c - nC_c(G_e))$.
End of Algorithm

Notice that graph G_e is highly recursive, hence the proof also applies to recursive graphs. □

6.2. *Number of finite components*

We have shown before (Theorem 7) that $fC_c(G_e)$ can be found with $\lceil \log(c+1) \rceil$ queries to ϕ'' if G_e is recursive, or with $\lceil \log(c+1) \rceil$ queries to ϕ' if G_e is highly recursive. Next we show that even if queries to weaker oracles are allowed for free, we still need the same number of queries to ϕ'' (ϕ') to find the number of finite components in a recursive graph.

Theorem 14. *Let $c \geqslant 0$ be any number. For any sets X, Y,*
- $fC_c(G_e^{\mathrm{r}}) \in FQ^Y(\lceil \log(c+1) \rceil - 1, X) \Rightarrow \phi'' \leqslant_{\mathrm{T}} Y$.
- $fC_c(G_e^{\mathrm{hr}}) \in FQ^Y(\lceil \log(c+1) \rceil - 1, X) \Rightarrow \phi' \leqslant_{\mathrm{T}} Y$.

Proof. Let X be any set. Since $\phi'' \leqslant_{\mathrm{T}} TOT$ and $\phi' \leqslant_{\mathrm{T}} \overline{K}$, we use TOT instead of ϕ'', and $\overline{K}$ instead of ϕ' to prove the lower bounds. To establish that $fC_c(G_e^{\mathrm{r}}) \notin FQ^Y(\lceil \log(c+1) \rceil - 1, X)$, unless $\phi'' \leqslant_{\mathrm{T}} Y$, we show that $\#_c^{TOT} \in FQ(1, fC_c)$. Then by the hypothesis, $\#_c^{TOT} \in FQ^Y(\lceil \log(c+1) \rceil - 1, X)$, and by Lemma 12 we have $TOT \leqslant_{\mathrm{T}} Y$.

We describe an algorithm for $\#_c^{TOT}$ that will use only one query to function $fC_c(G_e^{\mathrm{r}})$. The idea is to construct a graph G_i corresponding to each input x_i, in a way such that G_i has 0 finite components if $x_i \in TOT$, and has 1 finite component otherwise. To obtain this, we add a new vertex to graph G_i subject to the next consecutive element being accepted by M_{x_i}. Let G_e be the graph obtained from the disjoint union of the G_i's. G_e clearly has $c - fC_c(G_e)$ finite connected components.

Algorithm for $\#_c^{TOT}$
1. Input $(x_1, x_2, \ldots, x_c)$;
2. For $i = 1, 2, \ldots, c$, let G_i be constructed in stages:
Stage 0. Let $G_{i,0} := (\{0\}, \emptyset)$; $j := 0$; $z := 0$;
Stage $s+1$. Let $G_{i,s} = (V_{i,s}, E_{i,s})$.
If $j \in W_{x_i,s+1}$ then begin
$V_{i,s+1} = V_{i,s} \cup \{s+1\}$;
$E_{i,s+1} = E_{i,s} \cup \{[z, s+1]\}$;

$j := j + 1$; $z := s + 1$
end;

3. Take the disjoint union of $G_1, G_2, \ldots, G_c$, generating G_e^{r};
4. Return $(c - fC_c(G_e^{\mathrm{r}}))$.

End of Algorithm

Now, to establish that $fC_c(G_e^{\mathrm{hr}}) \notin FQ^Y(\lceil \log(c+1) \rceil - 1, X)$, unless $\phi' \leqslant_{\mathrm{T}} Y$, we show that $\#_c^{\overline{K}} \in FQ(1, fC_c)$. Then by the hypothesis, $\#_c^{\overline{K}} \in FQ^Y(\lceil \log(c+1) \rceil - 1, X)$, and by Lemma 12 we have $\overline{K} \leqslant_{\mathrm{T}} Y$. Since $\phi' \leqslant_{\mathrm{T}} \overline{K}$, we also have $\phi' \leqslant_{\mathrm{T}} Y$. We describe an algorithm for $\#_c^{\overline{K}}$ that uses only one query to function $fC_c(G_e^{\mathrm{hr}})$.

Algorithm for $\#_c^{\overline{K}}$

1. For $i = 1, 2, \ldots, c$, let G_i be as follows:

$$G_i = \begin{cases} (\{t\}, \emptyset) & \text{if } M_{x_i,t-1}(x_i)\uparrow \text{ and } M_{x_i,t}(x_i)\downarrow \\ (\emptyset, \emptyset) & \text{if } M_{x_i} \notin K. \end{cases}$$

2. Take the disjoint union of the G_i's, generating G_e^{hr}.
3. Return $(c - fC_c(G_e^{\mathrm{hr}}))$.

End of Algorithm

It is easy to see that if $x_i \in \overline{K}$ then G_i will have 0 finite components, and if $x_i \notin \overline{K}$ then G_i will have 1 finite component. Let j be the number of machines among $M_{x_1}, M_{x_2}, \ldots, M_{x_c}$ that are in $\overline{K}$. Consider G_e^{hr}, the disjoint union of $G_1, G_2, \ldots, G_c$. Then $fC_c(G_e^{\mathrm{hr}}) = c - j$. □

6.3. *Number of infinite components*

We have shown before (Theorem 10) that $iC_c(G_e)$ can be found with $\lceil \log(c+1) \rceil$ queries to ϕ''' if the graph is recursive, and with $\lceil \log(c+1) \rceil$ queries to ϕ'' if the graph is highly recursive. Next we show that even if queries to weaker oracles are allowed for free, we still need the same number of queries to ϕ''' (ϕ'') to find the number of infinite components in a recursive graph.

Theorem 15. *Let $c \geqslant 0$ be any number. For any sets X, Y,*

- $iC_c(G_e^{\mathrm{r}}) \in FQ^Y(\lceil \log(c+1) \rceil - 1, X) \Rightarrow \phi''' \leqslant_{\mathrm{T}} Y$.
- $iC_c(G_e^{\mathrm{hr}}) \in FQ^Y(\lceil \log(c+1) \rceil - 1, X) \Rightarrow \phi'' \leqslant_{\mathrm{T}} Y$.

Proof. Let X be any set. To establish that help from a weaker oracle does not allow a smaller number of queries to ϕ''', we use COF instead of ϕ''' and show that $\#_c^{COF} \in FQ(1, iC_c)$. Then by the hypothesis, $\#_c^{COF} \in FQ^Y(\lceil \log(c+1) \rceil - 1, X)$, so by Lemma 12 we have $COF \leqslant_{\mathrm{T}} Y$. We describe an algorithm for $\#_c^{COF}$ that will use only one query to function iC_c. The idea is to create graphs with components representing sequences of consecutive elements in the corresponding W_{x_i}. If $M_{x_i} \in COF$, then $iC(G_e^{\mathrm{r}}) = 1$, otherwise, $iC(G_e^{\mathrm{r}}) = 0$.

Algorithm for $\#_c^{COF}$

1. Input $(x_1, x_2, \ldots, x_c)$;
2. For $i = 1, 2, \ldots, c$, let G_i be constructed in stages:
Stage 0. Let $G_{i,0} := (\{0\}, \emptyset)$;
Stage $s+1$. $V_{i,s+1} = V_{i,s} \cup \{s+1\}$; $E_{i,s+1} = \{[j, j+1] \mid j \in W_{x_i, s+1}\}$;
3. Take the disjoint union of $G_1, G_2, \ldots, G_c$, generating G_e^r;
4. Return $(iC_c(G_e^r))$.

End of Algorithm

It is easy to show that if $x_i \in COF$ then G_i has 1 infinite component, and if $x_i \notin COF$ then G_i has no infinite components. Let j be the number of machines among $M_{x_1}, M_{x_2}, \ldots, M_{x_c}$ that are in COF. Consider G_e^r, the disjoint union of $G_1, G_2, \ldots, G_c$. Then $iC_c(G_e^r) = j$.

A slight modification in the size of the input in the proof of Theorem 13 will show the second part of Theorem 15. □

7. Unbounded recursive graph problems

We now turn our attention to the case when no bound is set *a priori*. We show that the problem of finding the number of connected components of an infinite recursive graph in this case is related to unbounded search in two ways:

1. If f is a nondecreasing recursive function, and $\sum_{i \geqslant 0} 2^{-f(i)} \leqslant 1$ is effectively computable, then the number of components of a recursive graph G_e, $nC(G_e)$, can be found with $f(nC(G_e))$ queries to ϕ'', and
2. If G is an infinite recursive graph and there is a set X such that $nC(G)$ can be computed using $f(nC(G))$ queries to X, then $\sum_{0 \leqslant i} 2^{-f(i)} \leqslant 1$.

Part 2 above can be interpreted as a lower bound for finding $nC(G)$. That result follows from a generalization of Theorem 9 in [4], which allows us to conclude that part 2 also applies to a wide class of problems, including the problems of finding the number of finite components and finding the number of infinite components of an infinite recursive graph.

In this section we introduce the *unbounded* search problem, and we study the complexity of finding the number of components of a recursive graph when it is known that the number of components is finite but no bound to it is given. We also study the problems of determining the number of finite and the number of infinite components.

7.1. The unbounded search problem

In this subsection we introduce the unbounded search problem and some relevant results.

The *unbounded search problem* is the following: The first player chooses an arbitrary number $n \geqslant 0$. The second player is allowed to ask queries of the type: '$x \leqslant n$?'. The latter player stops when she knows what number n is. The number of questions the

second player can ask depends on n itself. We say that $f(n)$ *questions suffice to solve the unbounded search problem* if there is an algorithm that the second player can use to guarantee that she knows the number n within $f(n)$ questions.

Optimal algorithms for unbounded search are related to binary prefix codes and Kraft's inequality [2, 6, 10].

The following are relevant previous results.

Lemma 16 (Beigel et al. [5]). *If A is a nonrecursive set, then F_n^A cannot be computed by a set of n partial recursive functions.*

Lemma 17 (Beigel et al. [5]). *If A and Y are sets such that $A \not\leq_T Y$, then F_n^A cannot be computed by a set of n partial functions that are recursive in Y.*

Theorem 18 (Bentley and Yao [6]). *If $f(n)$ questions suffice to solve the unbounded search problem, then f satisfies Kraft's inequality.*

Theorem 19 (Beigel [2]). *Let f be a nondecreasing recursive function such that $\sum_{i\geq 0} 2^{-f(i)} \leq 1$ and is effectively computable. There is an algorithm that solves the unbounded search problem by asking $f(n)$ questions (where n is the number being searched for) if and only if f satisfies Kraft's inequality.*

Theorem 20 (Gallagher [7, Kraft's Theorem]). *Let $\sigma_0, \sigma_1, \sigma_2, \ldots$ be an infinite sequence of elements from $\{0,1\}^*$ such that the bijection that maps i to σ_i is a binary prefix code. Then $\sum_{i\geq 0} 2^{-|\sigma_i|} \leq 1$.*

Remark. In the literature, the *unbounded search problem* is the search for a *positive* integer (not a *nonnegative* integer as we need), and Kraft's inequality is actually $\sum_{i\geq 1} 2^{-|\sigma_i|} \leq 1$. Since we can have empty graphs (with no components), we need a slight modification of what is found in the literature, but the modifications that are required in the proofs involved are trivial.

7.2. Computing the number of connected components

In this subsection we relate the complexity of finding the number of components, finite components, and infinite components of a recursive graph with the *unbounded search problem* in two ways:

1. If f be a nondecreasing recursive function, and $\sum_{i\geq 0} 2^{-f(i)} \leq 1$ is effectively computable, then $nC(G_e)$, $fC(G_e)$, and $iC(G_e)$ can be found with $f(nC(G_e))$, $f(fC(G_e))$, and $f(iC(G_e))$ queries to ϕ'', respectively.
2. If G is an infinite recursive graph and there is a set X such that $nC(G)$ can be computed using $f(nC(G))$ queries to X, then f satisfies Kraft's inequality.

The second part above can be interpreted as a lower bound for the problem, and is obtained through a more general result, which can also be used to derive similar lower bounds for $fC(G_e)$, and $iC(G_e)$.

Theorem 21. *Let f be a nondecreasing recursive function. If $\sum_{i\geqslant 0} 2^{-f(i)} \leqslant 1$ and is effectively computable, then $nC(G_e) \in FQ(f(nC(G_e)), \phi'')$.*

Proof. The proof of Theorem 1 shows that we can ask one single query to ϕ'' to get the answer to: '$nC(G_e) \leqslant k$?'. Hence, we can find $nC(G_e)$ by asking that type of query to ϕ'' as in an unbounded search algorithm. □

Theorem 21 together with Theorem 19 imply the existence of an algorithm that finds $nC(G_e)$ with $f(nC(G_e))$ queries to ϕ'', thus establishing an upper bound for that problem. By similar reasoning, one can show that we can find $fC(G_e)$ $(iC(G_e))$ by asking that type of query to ϕ' (ϕ'') or to ϕ'' (ϕ'''), depending on whether the graph is highly recursive or not.

Next we prove a theorem that implies lower bounds to those problems. Recall that $\gamma_n(G)$ is a partial function (presented in Section 2) which is undefined if $\gamma(G) > n$.

Lemma 22. *Let $\gamma(G)$ be any of $nC(G)$, $fC(G)$, or $iC(G)$. For any $n \geqslant 1$, the partial function $\gamma_n(G)$ cannot be computed by a set of n partial recursive functions.*

Proof. In the proof of Theorem 4 (respectively, 7 and 10), we showed that for all n, $F_n^K(x_1, x_2, \ldots, x_n)$ can be computed from one single use of $\gamma_n(G_e)$, where G_e can be constructed from $\{x_1, \ldots, x_n\}$. Hence, if $\gamma_n(G)$ could be computed by a set of n partial recursive functions, then so could F_n^K, which violates Lemma 16. □

The following theorem is a generalization of Theorem 9 in [4].

Theorem 23. *Let X be any set and f be any function. If a function $\gamma(G)$ is in $FQ(f(\gamma(G)), X)$, and $\gamma_n(G)$ cannot be computed by a set of n partial recursive functions, then f satisfies Kraft's inequality.*

Proof. Let $M^{(\ )}$ be the oracle Turing machine such that $M^X(G)$ computes $\gamma(G)$ with at most $f(\gamma(G))$ queries to X, for some function f. We use the fact that γ_n cannot be computed by a set of n partial recursive functions to obtain a contradiction.

For every natural n and sequence $\sigma \in \{0,1\}^*$, we define a partial recursive function $c_n^\sigma(G)$, constructed by simulating $M^{(\ )}(G)$ using the ith bit of σ to answer the ith query, in a way such that the machine either diverges or does the following:

(a) It makes at most $|\sigma|$ queries.

(b) The output x is between 0 and n, and $|\sigma| \leqslant f(|x|)$.

Notice that if σ is a prefix of σ' and $c_n^\sigma(G)$ converges to a value, then $c_n^{\sigma'}(G)$ converges to the same value.

By construction of $c_n^\sigma(G)$, we have that:

$$\forall n\ \forall G\ [\{c_n^\sigma(G) \mid \sigma \in \{0,1\}^* \text{ and } c_n^\sigma(G)\downarrow\} \subseteq \{0, \ldots, n\}].$$

We proceed to show by contradiction that

$$\forall n\ \exists G\ [\{c_n^\sigma(G) \mid \sigma \in \{0,1\}^* \text{ and } c_n^\sigma(G)\downarrow\} = \{0, \ldots, n\}].$$

To prove that the $\supseteq$ part also holds, we assume otherwise, and choose some integer n for which $\forall G\,[\{c_n^\sigma(G) \mid \sigma \in \{0,1\}^*$ and $c_n^\sigma(G)\downarrow\} \not\supseteq \{0,\ldots,n\}]$.

Then, $\forall G\,[\,|\{c_n^\sigma(G)\colon \sigma \in \{0,1\}^*$ and $c_n^\sigma(G)\downarrow\}| \leqslant n]$.

For each j, $1 \leqslant j \leqslant n$, we define partial recursive functions $h_j(G)$, which are computed by timesharing $c_n^\sigma(G)$ for all σ until the functions have output j distinct values, and outputting the jth distinct value. Therefore, for all G such that γ_n is defined,

$$\gamma_n(G) \in \{c_n^\sigma(G)\colon \sigma \in \{0,1\}^* \text{ and } c_n^\sigma(G)\downarrow\} = \{h_j(G) : 1 \leqslant j \leqslant n\}.$$

We conclude that the partial function γ_n is computable by a set of n partial recursive functions, which contradicts the hypothesis.

Hence we have that for every n, there exists a graph G such that for each $i \in \{0,\ldots,n\}$, there exists a sequence σ_i of oracle answers such that $|\sigma_i| \leqslant f(i)$ and $c_n^{\sigma_i}(G) = i$. Moreover, if $i \neq j$, then σ_i is not a prefix of σ_j. Therefore the sequences $\sigma_0,\ldots,\sigma_n$ form a binary prefix code for the integers 0–n, and by Kraft's Theorem (Theorem 20) we have $\sum_{0 \leqslant i \leqslant n} 2^{-|\sigma_i|} \leqslant 1$. Since $|\sigma_i| \leqslant f(i)$, $\sum_{0 \leqslant i \leqslant n} 2^{-f(i)} \leqslant 1$. Letting n approach infinity, we obtain $\sum_{0 \leqslant i} 2^{-f(i)} \leqslant 1$. □

Theorem 23 can be used to derive relationships of several problems in recursive graphs to Kraft's inequality, thereby establishing lower bounds for those problems. The next corollary illustrates some of them.

Corollary 24. *For any set X and function f, if*

(a) $nC(G) \in FQ(f(nC(G)),X)$, *or*

(b) $fC(G) \in FQ(f(fC(G)),X)$, *or*

(c) $iC(G) \in FQ(f(iC(G)),X)$, *then f satisfies Kraft's inequality.*

Proof. Lemma 22 shows that $nC(G)$, $fC(G)$, and $iC(G)$ satisfy the condition necessary to apply Theorem 23. □

8. Mixed queries in the unbounded case

In the previous section we have shown that if f is such that $nC(G)$ is in $FQ(f(nC(G)),X)$, then f satisfies Kraft's inequality, which can be interpreted as the number of queries needed to solve the problem. But it may be the case that if we allow queries to an oracle Y such that $\phi'' \not\leqslant_T Y$ the number of queries to ϕ'' can be reduced.

It turns out that the lower bound in the previous section is optimal with respect to queries to ϕ'', as we will show next.

Lemma 25. *Let Y be a set such that $\phi'' \not\leqslant_T Y$, and let $\gamma(G)$ represent any of $nC(G)$, $fC(G)$, or $iC(G)$. Then $\gamma_n(G)$ cannot be computed by a set of n partial functions that are recursive in Y.*

Proof. The proof of Lemma 22 relativizes, using Lemma 17 instead of Lemma 16, to accomplish the contradiction. □

Theorem 26. *Let Y be a set such that $\phi'' \not\leq_T Y$. Let X be any set and f be any function. If function $nC(G)$ is in $FQ^Y(f(nC(G)),X)$, then f satisfies Kraft's inequality.*

Proof. This proof is similar to the proofs of Theorem 23 and Corollary 24, but using Lemma 25 instead of Lemma 22 where appropriate. □

9. Final comments

We have classified the difficulty of determining the number of components, finite components, and infinite components of a recursive (highly recursive) graph, given a fixed upper bound. These results are tight in two ways: the lower bound on the number of queries holds even if a more powerful oracle is used, and no matter how many queries are used, the oracle must be of the degree established.

We have also studied the complexity of deciding if a recursive (highly recursive) graph has a finite number of components, finite components, and infinite components, and have shown that if we allow queries to weaker oracles for free, it may help just slightly, but in most cases it is of no use.

In the case when no bound is given a priori, we have shown that the problem of finding the number of connected components of an infinite recursive graph is related to unbounded search in two ways:

1. If f is a nondecreasing recursive function, and $\sum_{i\geq 0} 2^{-f(i)} \leq 1$ is effectively computable, then the number of components of a recursive graph G_e, $nC(G_e)$, can be found with $f(nC(G_e))$ queries to ϕ'';
2. If G is an infinite recursive graph and there is a set X such that $nC(G)$ can be computed using $f(nC(G))$ queries to X, then $\sum_{0\leq i} 2^{-f(i)} \leq 1$.

Part 2 above (which can be interpreted as a lower bound for finding $nC(G)$) also applies to a wide class of problems, including the problems of finding the number of finite components and finding the number of infinite components of an infinite recursive graph. That lower bound is optimal, even if we allow free queries to weaker oracles.

It is interesting to observe that, regardless of the Turing degree of the oracle involved, the optimal number of queries to solve each of the problems addressed is the same as the one for the binary search problem in both cases.

Acknowledgements

The authors would like to thank Steven Lempp for helpful discussions, and the anonymous referees for useful observations.

References

[1] D.R. Bean, Effective coloration, *J. Symbolic Logic* **41** (1976) 469–480.
[2] R.J. Beigel, Unbounded searching algorithms, *SIAM J. Comput.* **19** (1990) 522–537.
[3] R.J. Beigel and W.I. Gasarch, On the complexity of finding the chromatic number of a recursive graph I: The bounded case, *Ann. Pure Appl. Logic* **45** (1989) 1–38.
[4] R.J. Beigel and W.I. Gasarch, On the complexity of finding the chromatic number of a recursive graph II: The unbounded case, *Ann. Pure Appl. Logic* **45** (1989) 227–246.
[5] R.J. Beigel, W.I. Gasarch, J.T. Gill and J. Owings, Terse, superterse, and verbose sets, *Inform. and Comput.* **103** (1993) 68–85. Also Tech. Report 1806, Department of Computer Science, University of Maryland, 1987.
[6] J.L. Bentley and A.C.C. Yao, An almost optimal algorithm for unbounded searching, *Inform. Process. Lett.* **5** (August 1976) 82–87.
[7] R.G. Gallagher, Information Theory and Reliable Communication (Wiley, New York, 1968).
[8] W.I. Gasarch and K.S. Guimarães, in: Imre Simon, ed., On the number of components of a recursive graph, Lecture Notes in Computer Science, Vol. 583 (Springer, Berlin, 1992) 177–190.
[9] D. Harel, Hamiltonian paths in infinite graphs, *Israel J. Math.* **76** (1991) 317–336. A shorter version appeared in *STOC* (1991) 220–229.
[10] D.E. Knuth, Supernatural numbers, in: D.A. Klarner, ed., *The Mathematical Gardner* (Wadsworth, Belmont, CA, 1981) 310–325.
[11] M. Kummer, A proof of Beigel's cardinality conjecture, Universität Karlsruhe, Fakultät für Informatik, 1991, 5, Postfach 6980, D-7500 Kalsruhe 1, FRG. *J. Symbolic Logic* **57** (1992) 677–681.
[12] A. Manaster and J. Rosenstein, Effective matchmaking and k-chromatic graphs, *Proc. Amer. Math. Soc.* **39** (1973) 371–378.
[13] H. Rogers Jr., *Theory of Recursive Functions and Effective Computability* (McGraw-Hill, New York, 1967).
[14] R.I. Soare, *Recursively Enumerable Sets and Degrees*, Omega Series (Springer, Berlin, 1987).