Annals of Pure and Applied Logic 45 (1989) 227–246
North-Holland

ON THE COMPLEXITY OF FINDING THE CHROMATIC NUMBER OF A RECURSIVE GRAPH II: THE UNBOUNDED CASE

Richard BEIGEL
Department of Computer Science, The John Hopkins University, Baltimore, MD 21218, USA

William I. GASARCH
Department of Computer Science, Institute for Advanced Study, University of Maryland, College Park, MD 20742, USA

Communicated by A. Nerode
Received July 1988

A recursive graph is a graph whose edge set and vertex set are both recursive. Although the chromatic number of a recursive group G (denoted $\chi(G)$) cannot be determined recursively, it can be determined if queries to the halting set are allowed. We show that the problem of determining the chromatic number of a recursive graph with a minimum number of queries to the halting set, is closely related to the unbounded search problem. In particular if f is a non-decreasing function such that $\sum_{i\geq 0} 2^{-f(i)}$ is effectively computable, then there is an algorithm to determine $\chi(G)$ with $f(\chi(G))$ queries to K iff $\sum_{i\geq 0} 2^{-f(i)} \leq 1$ (i.e., f satisfies Kraft's inequality). We also investigate recursive chromatic numbers (which require queries to a set much harder than the halting set, namely $\emptyset'''$), the effect of allowing queries to a weaker set, and the effect of being able to ask p queries at a time. Most of our results are also true for highly recursive graphs (graphs where one can determine the neighbors of a given node recursively), though there are some interesting differences when queries to K are allowed for free in the computation of a recursive chromatic number.

1. Introduction

We continue the study of the complexity of graph coloring problems initiated in [3]. All the problems we deal with are unsolvable, but are recursive in either K (the halting set), $\emptyset''$ (the jump of the halting set, see [10] or [12]) or $\emptyset'''$ (the jump of the jump of the halting set). We measure the complexity of these problems in two ways: the Turing degree of the oracle and the number of queries to that oracle. In most cases we pin down both quantities exactly. Henceforth 'graph' means 'recursive or highly recursive graph,' terms originally defined in [1]. Most definitions, notations, and conventions not specified are the same as in [3]. References to related work can also be found there.

In [3] we studied several coloring problems where the chromatic number is bounded *a priori* by a constant. Here we consider several coloring problems where we know that the chromatic number exists but we are not given an *a priori* bound on it.

The following definition is needed to state our results.

Definition. Let $\mathbb{Q}$ denote the positive rational numbers. A real number r is *effectively computable* if there exists a recursive function $f:\mathbb{Q}\to\mathbb{Q}$ such that, for all $\epsilon>0$, $|f(\epsilon)-r|<\epsilon$.

In Section 2 we review the following theorem: if $\sum_{i\geq 0}2^{-f(i)}$ is effectively computable, then the unbounded search problem can be solved in $f(n)$ queries (where n is the number being searched for) iff $\sum_{i\geq 0}2^{-f(i)}\leq 1$. In Section 3 we show that finding the chromatic number of a graph is similar to the unbounded search problem in that

(a) if $\sum_{i\geq 0}2^{-f(i)}\leq 1$ and is effectively computable, then $\chi(G)$ can be found with $f(\chi(G))$ queries to K;

(b) if there exists a set X such that $\chi(G)$ can be determined from $f(\chi(G))$ queries to X, then $\sum_{i\geq 0}2^{-f(i)}\leq 1$.

In Section 4 we obtain similar results for recursive chromatic number, with queries to $\emptyset'''$ rather than K.

In Section 5 we investigate the effect of allowing queries to a less powerful set Y, (i.e. $K\not\leq_T Y$ when computing chromatic number, and $\emptyset'''\not\leq_T Y$ when computing recursive chromatic number) in the hope of cutting down on the number of queries to K or $\emptyset'''$. The most substantial savings of queries occurs in the problem of finding the recursive chromatic number of a highly recursive graph. If G is highly recursive, then

(a) $\chi^r(G)$ can be found with an unlimited number of queries to K and $\lceil\log\chi(G)\rceil$ queries to $\emptyset'''$,

(b) if $Y\not\leq_T\emptyset'''$ and there exists a set X such that $\chi^r(G)$ can be determined from an unlimited number of queries to Y and $g(\chi(G))$ queries to X then for all n,

$$g(n)>\lceil\log(n)\rceil-1.$$

In Section 6 we investigate the effect of asking p queries in parallel. Unlike the serial case there are sets such that if they are queried instead of K or $\emptyset'''$ then substantial savings are possible. In Section 7 we investigate the effects of being able to both ask questions to a weaker set, and ask p queries in parallel.

We formally define the class of functions which can be computed by an oracle Turing machine, with oracle A, using a bounded number of queries to A.

Definition. A partial function f is in $\mathrm{FQ}(n, A)$ if $f\leq_T A$ via an oracle Turing machine that, when using oracle A, never makes more than n queries. If B is a set, then f is in $\mathrm{FQ}^B(n, A)$ if $f\leq_T A\oplus B$ via an oracle Turing machine, that, when using oracle $A\oplus B$, never makes more than n queries to A (though it may make many queries to B).

Note. The definition of $\mathrm{FQ}(n, A)$ still makes sense if 'n' is replaced by a function of the input. The statement "$\chi(G)\in\mathrm{FQ}(f(\chi(G)), X)$" will mean that computing the chromatic number of graph G can be computed with $f(\chi(G))$ queries to X, assuming $\chi(G)$ is defined.

Bounded queries are related to the notion of computing a partial function by a set of partial functions.

Definition. Let S be a set of partial functions and f be a partial function. f *is computed by* S if for all x such that $f(x)$ is defined,

$$f(x) \in \{g(x) \mid g \in S \text{ and } g(x)\downarrow\}.$$

The following function will be useful to us.

Definition. Let A be any set and k be any number. The function F_k^A is defined by

$$F_k^A(x_1, \ldots, x_k) = \langle \chi_A(x_1), \ldots, \chi_A(x_k) \rangle$$

where χ_A is the characteristic function of set A.

The following lemmas are proven in [4].

Lemma 1. *If A is a nonrecursive set, then F_n^A cannot be computed by a set of n partial recursive functions.*

The proof of the above lemma easily relativizes to yield

Lemma 2. *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.*

2. The unbounded search problem

The *unbounded search problem* is the following: Player A chooses an arbitrary non-negative integer n. Player B is allowed to ask whether an integer x is less than n. Player B stops when she knows what the number is. The number of questions player B asks depends on n itself. We say that $f(n)$ *questions suffice to solve the unbounded search problem* if there is an algorithm that player B can use such that she will always stop within $f(n)$ questions. Bentley and Yao [5], Knuth [8], and Beigel [2] have studied the unbounded search problem.

Optimal algorithms for unbounded search are related to binary prefix codes and Kraft's inequality.

Definition. 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 no string in the range of the bijection is a prefix of a different string in the range of the bijection.

Definition. A function f from $\mathbb{N}$ to $\mathbb{N}$ satisfies Krafts inequality if $\sum_{i \geq 0} 2^{-f(i)} \leq 1$.

Kraft's Theorem [7]. *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 $\sum_{i\geq 0} 2^{-|\sigma_i|} \leq 1$.*

Note. The unbounded search problem in the literature is the search for a *positive* integer, not a nonnegative integer. Kraft's inequality is actually $\sum_{i\geq 1} 2^{-f(i)} \leq 1$. We are using a slight modification of the unbounded search problem because we allow graphs to have chromatic number 0. The adjustment to the proofs in the literature is trivial.

We will need the following theorems.

Theorem 3 (Bentley and Yao [5]). *If $f(n)$ questions suffice to solve the unbounded search problem, then $f(n)$ satisfies Kraft's inequality.*

Theorem 4 (Beigel [2]). *Let f be a non-decreasing recursive function such that $\sum_{i\geq 0} 2^{-f(i)} \leq 1$ and is effectively computable. There is an algorithm to solve 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.*

We give examples of slow growing functions that satisfy Krafts's inequality, and examples of slow growing funtions that do not. First we need some definitions.

Definitions.

$$\log_b^{(i)} x = \begin{cases} x & \text{if } i = 0, \\ \log_b \log_b^{(i-1)}(x) & \text{otherwise.} \end{cases}$$

$$\log_b^* x = \min\{t \mid \log_b^{(t)} x \leq 1\}.$$

$$\operatorname{logsum}_b x = \sum_{1 \leq i \leq \log_b^* x} \log_b^{(i)} x.$$

Theorem 5 (Beigel [2]). (a) *The function $f(n) = \lceil \operatorname{logsum}_2(n+1) \rceil + 2$ satisfies Kraft's inequality.*

(b) *If $\epsilon > 0$, then there exist a constant c such that*

$$f(n) = \lceil \operatorname{logsum}_2(n+1) - (\log_2 \log_2(\mathrm{e} - \epsilon)) \log_2^*(n+1) \rceil + c$$

satisfies Kraft's inequality, where e *denotes the base of the natural log.*

(c) *Let c be any natural number. The function*

$$f(n) = \lceil \operatorname{logsum}_2(n+1) - (\log_2 \log_2 \mathrm{e}) \log_2^*(n+1) \rceil + c$$

does not satisfy Kraft's inequality.

3. Computing the chromatic number

In this section we show that if $f(n)$ queries suffice to solve the unbounded search problem, then $f(n)$ queries can be used to find the chromatic number of a recursive graph; and conversely that if $f(n)$ queries suffice to discover that $\chi(G) = n$, then the unbounded search problem can be solved in $f(n)$ queries.

Lemma 6 (Beigel and Gasarch [3]). *Given a recursive graph G and a natural number x, one can determine whether $\chi(G) \leq x$ by making a single query to K.*

Theorem 7. *Let f be a non-decreasing, recursive function such that $\sum_{i\geq 0} 2^{-f(i)} \leq 1$ and is effectively computable. Then $\chi(G)$ is in $\mathrm{FQ}(f(\chi(G)), K)$.*

Proof. By Lemma 6, a question of the form "$\chi(G) \leq x$?" can be phrased as a query to K. Therefore the problem of finding $\chi(G)$ is an unbounded search problem. By Theorem 4 there is an algorithm that solves this problem in $f(\chi(G))$ queries. □

Note. The algorithm in Theorem 7 essentially keeps asking questions of the form "$\chi(G) \leq x$?", with larger and larger values of x, until it receives a YES answer; at which point it will narrow in on the answer. If the input is not an index for a recursive graph, then the algorithm either terminates or asks infinitely many questions. It cannot ask finitely many questions and not terminate.

We prove a converse to Theorem 7.

Lemma 8. *The partial function*

$$\chi_n(G) = \begin{cases} \chi(G) & \text{if } 0 \leq \chi(G) \leq n, \\ \text{undefined} & \text{otherwise} \end{cases}$$

cannot be computed by a set of n partial recursive functions.

Proof. In [3] we showed that $F_n^K(x_1, \ldots, x_n)$ can be computed from $\chi_n(G)$ where G can be constructed from $\{x_1, \ldots, x_n\}$. Hence if $\chi_n(G)$ can be computed by a set of n recursive functions, then F_n^K could be computed by a set of n recursive functions, which violates Lemma 1. □

Theorem 9. *Let X be any set and f be any function. If $x(G)$ is in $\mathrm{FQ}(f(\chi(G)), X)$, then $\sum_{i\geq 0} 2^{-f(i)} \leq 1$.*

Proof. Let $M^{(\ )}$ be the oracle Turing machine such that $M^X(G)$ computes $\chi(G)$ and makes at most $f(\chi(G))$ queries to X, for some function f. Let χ_n be as in the above lemma. We will use the fact that χ_n cannot be computed by a set of n partial recursive functions to obtain a contradiction.

For each sequence $\sigma \in \{0, 1\}^*$ we define a function $c_n^\sigma(G)$ computed as follows: simulate $M^{(\ )}(G)$ using the ith bit of σ to answer the ith query. If during this process any of the following three happen, then diverge:

(a) there is an attempt to make a $(|\sigma| + 1)$th query, or

(b) the computation terminates and the output is not between 0 and n, or

(c) the computation terminates and outputs i where $|\sigma| > f(i)$.

If none of these three happen, then continue simulating the computation, and if it halts, then halt with the same output it gave. Since we can store the values $f(0), \ldots, f(n)$ in a finite table, c_n^σ is a partial recursive function for every n and σ. By the construction of c_n^σ, whenever $0 \leq \chi(G) \leq n$ there exists some σ of length $f(\chi(G))$ or less that represents correct answers to the queries that $M^{(\ )}(G)$ makes to X. That is

$$\chi_n(G) \in \{c_n^\sigma(G): \sigma \in \{0, 1\}^* \text{ and } c_n^0(G)\downarrow\}.$$

Let σ be a prefix of σ'. If $c_n^\sigma(G)$ converges to a value, then $c_n^{\sigma'}(G)$ must converge to the same value. We will use this fact later in order to construct a binary prefix code for the integers 0 through n. By the construction of c_n^σ, if $c_n^\sigma(G)$ converges, then

$$c_n^\sigma(G) \in \{0, \ldots, n\}.$$

Therefore,

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

We claim that

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

We prove this by contradiction. Suppose that

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

Choose such an integer n. Then

$$(\forall G)[|\{c_n^\sigma(G): \sigma \in \{0, 1\}^* \text{ and } c_n^\sigma(G)\downarrow\}| \leq n].$$

For $1 \leq j \leq n$, define a partial recursive function $h_j(G)$, computed as follows: Timeshare $c_n^\sigma(G)$ for all σ until the functions have output j distinct values; output the jth distinct value. Therefore, for all G such that $\chi_n(G)$ is defined

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

Thus the partial function χ_n is computable by a set of n partial recursive functions. This contradicts Lemma 8. This contradiction establishes the claim.

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 $|\sigma_i| \leq f(i)$ and $c_n^{\sigma_i(G)} = i$. As observed above, 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 through n. Therefore

Kraft's Theorem [7] implies that

$$\sum_{0 \leq i \leq n} 2^{-|\sigma_i|} \leq 1.$$

Since $|\sigma_i| \leq f(i)$,

$$\sum_{0 \leq i \leq n} 2^{-f(i)} \leq 1.$$

Letting n approach infinity, we obtain the inequality

$$\sum_{i \geq 0} 2^{-f(i)} \leq 1. \quad \square$$

4. Recursive chromatic number

The *recursive chromatic number* $\chi^r(G)$ of a graph G is the minimum number of colors that suffice in order to color G via an effective algorithm. There are recursive graphs with finite chromatic number that cannot be colored recursively [1] hence we cannot compute $\chi^r(G)$ from $\chi(G)$.

Lemma 10 (Beigel and Gasarch [3]). *Given recursive graph G and a natural number x, one can determine whether $\chi^r(G) \leq x$ by making a single query to $\emptyset'''$.*

Theorem 11. *Let f be a non-decreasing, recursive function such that $\sum_{i \geq 0} 2^{-f(i)} \leq 1$ and is effectively computable. Then $\chi^r(G)$ is in $\mathrm{FQ}(f(\chi^r(G)), \emptyset''')$.*

Proof. Since $\chi^r(G)$ is a positive integer, and since we may determine whether $\chi^r(G) \leq n$ by making a single query to $\emptyset'''$, this follows from Theorem 4. $\square$

Theorem 12. *Let f be a non-decreasing function. If there exists a set X such that $\chi^r(G)$ is in $\mathrm{FQ}(f(\chi^r(G)), X)$, then $\sum_{i \geq 0} 2^{-f(i)} \leq 1$.*

Proof. Similar to the proof of Theorem 9. The key fact needed is that the function χ^r_n, defined analogously to χ_n in Lemma 8, cannot be computed by a set of n partial recursive functions. This is proven in [3]. $\square$

5. Mixed queries

We have seen that if f is such that $f(\chi(G))$ $(f(\chi^r(G)))$ queries to K $(\emptyset''')$ can be used to compute $\chi(G)$ $(\chi^r(G))$, then f satisfies Kraft's inequality, which can be interpreted as a lower bound on the number of queries needed. If we allow queries to a set Y such that $K \not\leq_T Y$ $(\emptyset''' \not\leq_T Y)$, then perhaps the number of queries to K $(\emptyset''')$ can be reduced. In this section we will see that for finding $\chi(G)$,

queries to such a Y do not help; however for finding $\chi^r(G)$ they do. We also exhibit lower bounds on how helpful queries to Y can be. The most substantial savings occur when computing $\chi^r(G)$ for highly recursive G.

We show that for computing $\chi(G)$, queries to any Y such that $K \not\leq_T Y$ do not help.

Lemma 13. *Let Y be any set such that $K \not\leq_T Y$. Let χ_n be the partial function in Lemma 8. Then χ_n cannot be computed by a set of n partial functions that are recursive in Y.*

Proof. The proof of Lemma 8 relativizes, with the help of Lemma 2. □

Theorem 14. *Let Y be any set such that $K \not\leq_T Y$. Let X be any set and f be any function. If $\chi(G)$ is in $\mathrm{FQ}^Y(f(\chi(G)), X)$, then $\sum_{i\geq 0} 2^{-f(i)} \leq 1$.*

Proof. Similar to the proof of Theorem 9, using Lemma 13 instead of Lemma 8. □

We show that for computing $\chi^r(G)$ for recursive graphs, queries to K help cut down on queries to $\emptyset'''$. We have matching upper and lower bounds on how helpful such queries are.

The following theorem holds when applied to both the class of recursive graphs, and to the class of highly recursive graphs; however a stronger version is true for the class of highly recursive graphs.

Theorem 15. *Let f be a nondecreasing recursive function such that $\sum_{i\geq 0} 2^{-f(i)} \leq 1$, and is effectively computable. Then $\chi^r(G)$ is in $\mathrm{FQ}^K(f(\chi^r(G) - \chi(G)), \emptyset''')$.*

Proof. By Theorem 7, $\chi(G)$ can be computed recursively in K. Since $\chi^r(G) \geq \chi(G)$ our unbounded search for $\chi^r(G)$, using Theorem 4 and Lemma 10 applied to f, begins at $\chi(G)$ (instead of at 0). Hence it locates $\chi^r(G)$ in $f(\chi^r(G) - \chi(G))$ queries to $\emptyset'''$. □

If the class of graphs being considered are recursive, then the converse of Theorem 15 is true (Theorem 17). If the class of graphs being considered are highly recursive, then the converse of Theorem 15 is false (Theorem 19). To establish the converse for recursive graphs we need a lemma.

Lemma 16. *Let Y be any set such that $\emptyset''' \not\leq_T Y$. The partial function*

$$\chi^r_{[2,n+2]}(G) = \begin{cases} \chi^r(G) & \text{if } 2 \leq \chi^r(G) \leq n+2 \text{ and } \chi(G) = 2, \\ \text{undefined} & \text{otherwise} \end{cases}$$

cannot be computed by a set of n partial functions recursive in Y.

Proof. In [3] we showed that $F_n^{\emptyset'''}(x_1, \ldots, x_n)$ can be computed from $\chi^{\mathrm{r}}_{[2,n+2]}(G)$ where G can be constructed from $\{x_1, \ldots, x_n\}$. Hence, if $\chi^{\mathrm{r}}_{[2,n+2]}(G)$ can be computed with n functions, then $F_n^{\emptyset'''}$ could be computed with n functions, which violates Lemma 1. □

The proof of the following theorem is similar to the proof of Theorem 9.

Theorem 17 (only for recursive graphs). *Let Y be such that $\emptyset''' \not\leq_{\mathrm{T}} Y$, let X be any set, and let f be any function. If $\chi^{\mathrm{r}}(G)$ is in $\mathrm{FQ}^Y(f(\chi^{\mathrm{r}}(G) - \chi(G)), X)$, then $\sum_{i\geq 0} 2^{-f(i)} \leq 1$.*

Proof. Let $M^{(\ )}$ be the oracle Turing machine such that $M^{X\oplus Y}(G)$ computes $\chi^{\mathrm{r}}(G)$ in $\mathrm{FQ}^Y(f(\chi^{\mathrm{r}}(G) - \chi(G)), X)$, for some function f. Let $\chi^{\mathrm{r}}_{[2,n+2]}(G)$ be as in the above lemma.

For each sequence $\sigma \in \{0, 1\}^*$ we define a function $c_n^\sigma(G)$ computed (recursively in Y) as follows: simulate $M^{(\ )}(G)$ using the ith bit of σ to answer the ith query to X, while answering all queries to Y correctly. If during this process any of the following happen, then diverge:

(a) there is an attempt to make a $(|\sigma| + 1)$th query to X, or

(b) the computation terminates and the output is not between 2 and $n + 2$, or

(c) the computation terminates and the output is i where $|\sigma| > f(i - 2)$.

(If $\chi(G) = 2$ and the output is $i = \chi^{\mathrm{r}}(G)$, then these conditions force the number of queries to be $\leq f(i-2) = f(\chi^{\mathrm{r}}(G) - \chi(G))$.)

If none of these three things happen, then continue simulating the computation, and if it halts, then halt with the same output it gave. Since we can store the values $f(0), \ldots, f(n)$ in a finite table, c_n^σ is a partial function that is recursive in Y, for every n and σ.

By the construction of c_n^σ, whenever $2 \leq \chi^{\mathrm{r}}(G) \leq n + 2$ and $\chi(G) = 2$

$$\chi^{\mathrm{r}}_{[2,n+2]}(G) \in \{c_n^\sigma(G)\colon \sigma \in \{0, 1\}^*,\ c_n^\sigma(G)\downarrow,\ \text{and } \chi(G) = 2\}.$$

Let σ be a prefix of σ'. If $c_n^\sigma(G)$ converges to a value, then $c_n^{\sigma'}(G)$ must converge to the same value. We will use this fact later in order to construct a binary prefix code for the integers 2 through $n + 2$. By the construction of c_n^σ, if $c_n^\sigma(G)$ converges, then

$$c_n^\sigma(G) \in \{2, \ldots, n + 2\}.$$

Therefore,

$$(\forall n)(\forall G)[\{c_n^\sigma(G)\colon \sigma \in \{0, 1\}^*,\ c_n^\sigma(G)\downarrow,\ \text{and } \chi(G) = 2\} \subseteq \{2, \ldots, n + 2\}].$$

We claim that

$$(\forall n)(\exists G)[\{c_n^\sigma(G)\colon \sigma \in \{0, 1\}^* \text{ and } c_n^\sigma(G)\downarrow\} = \{2, \ldots, n + 2\} \text{ and } \chi(G) = 2].$$

We prove this by contradiction. Suppose that

$$(\exists n)(\forall G)[\{c_n^\sigma(G)\colon \sigma \in \{0, 1\}^*,\ c_n^\sigma(G)\downarrow,\ \text{and } \chi(G) = 2\} \subsetneq \{2, \ldots, n + 2\}].$$

Choose such an integer n. Then

$$(\forall G)[|\{c_n^\sigma(G)\colon \sigma \in \{0,1\}^*,\ c_n^\sigma(G)\downarrow,\ \text{and } \chi(G)=2\}| \leq n].$$

For $1 \leq j \leq n$, define a partial function $h_j(G)$ that is recursive in Y as follows: Timeshare $c_n^\sigma(G)$ for all σ until the functions have output j distinct values; output the jth distinct value. Therefore, for all G such that $2 \leq \chi^r(G) \leq n+2$ and $\chi(G)=2$

$$\chi^r_{[2,n+2]}(G) \in \{c_n^\sigma(G)\colon \sigma \in \{0,1\}^*,\ c_n^\sigma(G)\downarrow,\ \text{and } \chi(G)=2\} = \{h_j(G)\colon 1 \leq j \leq n\}.$$

Thus the partial function $\chi^r_{[2,n+2]}$ is computable by a set of n partial functions recursive in Y. This contradicts Lemma 16. This contradiction establishes the claim.

For every n, there exists a graph G such that for each i in $\{2, \ldots, n+2\}$, there exists a sequence σ_i of oracle answers such that $c_n^{\sigma_i}(G)=i$. By condition c above we know that $|\sigma_i| \leq f(i-2)$. As observed above, if $i \neq j$, then σ_i is not a prefix of σ_j. Therefore the sequences $\sigma_2, \ldots, \sigma_{n+2}$ form a binary prefix code for the integers 2 through $n+2$. Therefore Kraft's Theorem [7] implies that

$$\sum_{2 \leq i \leq n+2} 2^{-|\sigma_i|} \leq 1.$$

Since $|\sigma_i| \leq f(i-2)$,

$$\sum_{2 \leq i \leq n+2} 2^{-f(i-2)} \leq 1,$$

which implies

$$\sum_{0 \leq i \leq n} 2^{-f(i)} \leq 1.$$

Letting n approach infinity, we obtain the inequality

$$\sum_{i \geq 0} 2^{-f(i)} \leq 1. \qquad \square$$

In highly recursive graphs there is a relationship between chromatic number and recursive chromatic number which will enable us to cut down on queries to $\emptyset'''$ substantially, if we allow unbounded queries to K.

Theorem 18 (Carstens and Pappinghaus [6], and Schmerl [11]). *If G is a highly recursive graph, then*

$$\chi(G) \leq \chi^r(G) \leq 2\chi(G) - 1.$$

Both bounds are optimal [11].

Theorem 19. *$\chi^r(G)$ is in $\mathrm{FQ}^K[\lceil \log(\chi(G)) \rceil, \emptyset''']$.*

Proof. Given G, use the algorithm suggested by Theorem 7 to find $\chi(G)$ recursively in K. Since $\chi(G) \leq \chi^r(G) \leq 2\chi(G) - 1$ a binary search of this interval, using questions to $\emptyset'''$ to determine comparisons, can be used to determine $\chi^r(G)$. This needs at most $\lceil \log((2\chi(G) - 1) - \chi(G) + 1) \rceil = \lceil \log(\chi(G)) \rceil$ queries. □

Theorem 19 is optimal with respect to queries to $\emptyset'''$.

Theorem 20. *Let Y be a set such that $\emptyset''' \not\leq_T Y$. Let X be any set. If $\chi^r(G)$ can be computed with an unlimited number of queries to Y and $g(\chi(G))$ queries to X, then for all $n \geq 2$, $g(n) > \lceil \log n \rceil - 1$.*

Proof. Let b be a fixed constant and Y be as in the hypothesis. Let $\chi^r_{[b,2b-1]}$ be defined by

$$\chi^r_{[b,2b-1]}(G) = \begin{cases} \chi^r(G) & \text{if } b \leq \chi^r(G) \leq 2b-1 \text{ and } \chi(G) = b, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In [3] we showed that the function $\chi^r_{[b,2b-1]}$ is not in $\mathrm{FQ}^Y(\lceil \log b \rceil - 1, X)$.

Assume that $\chi^r(G)$ is in $\mathrm{FQ}^Y(g(\chi(G)), X)$. If this algorithm is restricted to graphs G such that $b \leq \chi^r(G) \leq 2b - 1$ and $\chi(G) = b$, then it will only use $g(b)$ queries to X. Hence $g(b) > \lceil \log b \rceil - 1$. Hence, for all n, $g(n) > \lceil \log n \rceil - 1$. □

6. Parallel queries

We now examine what happens if we allow p queries to be asked at once. Each set of p queries is called a round. We place upper and lower bounds on the number of rounds of p queries that are needed to determine the chromatic number and the recursive chromatic number of a graph. Formally, a round of p queries to X is one query to F_p^X. The number p is a fixed constant throughout this section.

Optimal algorithms for unbounded search using rounds of p queries are related to $(p+1)$-ary prefix codes and Kraft's inequality for $(p+1)$-ary-trees.

Definition. Let D be a set of natural numbers. A *$(p+1)$-ary prefix code for D* is a bijection from D onto a subset of $\{0, 1, 2, \ldots, p\}^*$ such that no string in the range of the bijection is a prefix of a different string in the range of the bijection.

Definition. A function f from $\mathbb{N}$ to $\mathbb{N}$ satisfies *Krafts inequality for $(p+1)$-ary-trees* if

$$\sum_{i \geq 0} (p+1)^{-f(i)} \leq 1.$$

Kraft's Theorem for $(p+1)$-ary-trees [7]. *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 $(p+1)$-ary prefix code. Then $\sum_{i \geq 0} (p+1)^{-|\sigma_i|} \leq 1$.*

Theorem 21 (Beigel [2]). *If $f(n)$ rounds of p queries suffice to solve the unbounded search problem, then $f(n)$ satisfies Kraft's inequality for $(p+1)$-ary-trees.*

Theorem 22 (Beigel [2]). *Let f be a non-decreasing recursive function such that*

$$\sum_{i\geq 0}(p+1)^{-f(i)}$$

is effectively computable. There is an algorithm that solves the unbounded search problem by asking $f(n)$ rounds of p questions (where n is the number being searched for) if and only if f satisfies Kraft's inequality for $(p+1)$-ary-trees.

We give examples of slow growing functions that satisfy Krafts' inequality for $(p+1)$-ary-trees, and examples of slow growing functions that do not.

Theorem 23 (Beigel [2]). (1) *The function $f(n)=\lceil \text{logsum}_p(n+1)+2\rceil$ satisfies Kraft's inequality for $(p+1)$-ary-trees.*

(2) *If $\epsilon>0$, then there exists a constant c such that*

$$f(n)=\lceil \text{logsum}_p(n+1)-(\log_p \log_p(\mathrm{e}-\epsilon))\log_p^*(n+1)\rceil + c$$

satisfies Kraft's inequality for $(p+1)$-ary-trees, where e *denotes the base of the natural log.*

(3) *Let c be any natural number. The function*

$$f(n)=\lceil \text{logsum}_p(n+1)-(\log_p \log_p \mathrm{e})\log_p^*(n+1)\rceil + c$$

does not satisfy Kraft's inequality for $(p+1)$-ary-trees.

Theorem 24. *Let f be a non-decreasing, recursive function such that $\sum_{i\geq 0}(p+1)^{-f(i)}\leq 1$ and is effectively computable. $\chi(G)$ is in $\text{FQ}(f(\chi(G)), F_p^K)$.*

Proof. This can be obtained by combining Theorem 22 with Lemma 6. □

We prove a converse to Theorem 24, in a manner similar to the proof of Theorem 9.

Theorem 25. *If $\chi(G)$ is in $\text{FQ}(f(\chi(G)), F_p^K)$, then $\sum_{i\geq 0}(p+1)^{-f(i)}\leq 1$.*

Proof. Let $M^{(\ )}$ be the oracle Turing machine such that $M^K(G)$ computes $\chi(G)$ and makes at most $f(\chi(G))$ queries to F_p^K, for some function f.

Let χ_n be the partial function defined in Lemma 8. By Lemma 8, χ_n cannot be computed by a set of n partial recursive functions. We use this to force f to satisfy Kraft's inequality for $(p+1)$-ary-trees.

Let $\{0,1,2,\ldots,p\}^*$ denote all finite sequences of numbers from the set $\{0,1,\ldots,p\}$. If $\sigma\in\{0,1,2,\ldots,p\}^*$, then let $\sigma(i)$ be the ith element of σ. For

each sequence $\sigma \in \{0, 1, 2, \ldots, p\}^*$, define a function $c_n^\sigma(G)$ computed as follows: Simulate $M^{(\ )}(G)$ until the first round of p queries is reached. Let the questions be $x_1, \ldots, x_p$. Run all the machines $\{x_1\}, \{x_2\}, \ldots, \{x_p\}$ until exactly $\sigma(1)$ halt (this may never happen in which case the machine will diverge). Continue the computation with the oracle query answered by saying YES to all $\sigma(1)$ elements of $x_1, \ldots, x_p$ that halt, and NO to all those that did not. Assuming that the elements of $x_1, \ldots, x_p$ that did not halt are not in K, continue the computation. When the second round of p queries is reached do the same thing only waiting until $\sigma(2)$ of the machines halt. Continue in this manner. If during this process any of the following three things happen, then diverge:

(a) there is an attempt to make a $(|\sigma| + 1)$th query, or

(b) the computation terminates and the output is not between 0 and n, or

(c) the computation terminates and outputs i where $|\sigma| > f(i)$.

If none of these three happen, then continue simulating the computation, and if it halts, then halt with the same output it gave. Since we can store the values $f(0), \ldots, f(n)$ in a finite table, c_n^σ is a partial recursive function for every n and σ. By the construction of c_n^σ, whenever $0 \leq \chi_n(G) \leq n$

$$\chi_n(G) \in \{c_n^\sigma(G): \sigma \in \{0, 1, \ldots, p\}^* \text{ and } c_n^\sigma(G)\downarrow\}.$$

Let σ be a prefix of σ'. If $c_n^\sigma(G)$ converges to a value, then $c_n^{\sigma'}(G)$ must converge to the same value. We will use this fact later in order to construct a $(p+1)$-ary prefix code for the integers 0 through n. By the construction of c_n^σ, if $c_n^\sigma(G)$ converges, then

$$c_n^\sigma(G) \in \{0, \ldots, n\}.$$

Therefore,

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

We prove this by contradiction. Suppose that

$$(\exists n)(\forall G)[\{c_n^\sigma(G): \sigma \in \{0, 1, 2, \ldots, p\}^* \text{ and } c_n^\sigma(G)\downarrow\} \subsetneq \{0, \ldots, n\}].$$

Choose such an integer n. Then

$$(\forall G)[|\{c_n^\sigma(G): \sigma \in \{0, 1, 2, \ldots, p\}^* \text{ and } c_n^\sigma(G)\downarrow\}| \leq n].$$

For $1 \leq j \leq n$, define a partial recursive function $h_j(G)$, computed as follows: Timeshare $c_n^\sigma(G)$ for all σ until the functions have output j distinct values; output the jth distinct value. Therefore, for all G such that $\chi_n(G)$ is defined

$$\chi_n(G) \in \{c_n^\sigma(G): \sigma \in \{0, 1, 2, \ldots, p\}^* \text{ and } c_n^\sigma(G)\downarrow\} = \{h_j(G): 1 \leq j \leq n\}.$$

Thus the partial function χ_n is computable by a set of n partial recursive functions. This contradicts Lemma 8. This contradiction establishes the claim.

For every n, there exists a graph G such that for each i in $\{0, \ldots, n\}$, there exists a sequence $\sigma_i \in \{0, 1, 2, \ldots, p\}^*$ such that $|\sigma_i| \leq f(i)$ and $c_n^\sigma(G) = i$. As

observed above, if $i \neq j$, then σ_i is not a prefix of σ_j. Therefore the sequences $\sigma_0, \ldots, \sigma_n$ form a $(p+1)$-ary prefix code for the integers 1 through n. Therefore Kraft's Theorem [7] implies that

$$\sum_{0 \leq i \leq n} (p+1)^{-|\sigma_i|} \leq 1.$$

Since $|\sigma_i| \leq f(i)$,

$$\sum_{0 \leq i \leq n} (p+1)^{-f(i)} \leq 1.$$

Letting n approach infinity, we obtain the inequality

$$\sum_{i \geq 0} (p+1)^{-f(i)} \leq 1. \quad \square$$

If an oracle other than K is used, then we can decrease the number of queries substantially.

Theorem 26. *Let f be a non-decreasing, recursive function such that $\sum_{i \geq 0} 2^{-pf(i)} \leq 1$ and is effectively computable. There exists a set A, $A \equiv_T K$, such that $\chi(G)$ is in $\mathrm{FQ}(f(\chi(G)), F_p^A)$.*

Proof. Note that the function $p \times f(n)$ satisfies the conditions of Theorem 7. Hence there is an algorithm that computes $\chi(G)$ using $p \times f(\chi(G))$ queries to K. Let $M^{(\,)}$ be the machine that computes that algorithm. By the note following Theorem 7 the algorithm in Theorem 7 only diverges by asking infinitely many questions. Hence the following set A is Turing equivalent to K:

$$A = \{\langle e, i\rangle \mid \text{the } i\text{th query made in the } M^K(e) \text{ computation answers YES}\}.$$

To compute $\chi(G)$, first query

$$F_p^A(\langle e, 1\rangle, \langle e, 2\rangle, \langle e, 3\rangle, \ldots, \langle e, p\rangle)$$

and then do the computation $M^{(\,)}(e)$ knowing the first p answers to queries that will be asked. If the computation does not terminate, then query

$$F_p^A(\langle e, p+1\rangle, \langle e, p+2\rangle, \langle e, p+3\rangle, \ldots, \langle e, 2p\rangle)$$

and continue the computation. Keep up this process until the computation ends. In the worst case the last query to F_p^A is

$$F_p^A(\langle e, (f(\chi(G))-1)p+1\rangle, \ldots, \langle e, f(\chi(G)) \times p\rangle)$$

and the number of queries to F_p^A is $(f(\chi(G)) \times p)/p = f(\chi(G))$. $\square$

Theorem 27. *Let X be any set and f be any function. If $\chi(G)$ is in $\mathrm{FQ}(f(\chi(G)), F_p^X)$, then $\sum_{i \geq 0} 2^{-pf(i)} \leq 1$.*

Proof. If $\chi(G)$ is in $\mathrm{FQ}(f(\chi(G)), F_p^X)$, then $\chi(G)$ is in $\mathrm{FQ}(pf(\chi(G)), X)$. By Theorem 9, $\sum_{i \geq 0} 2^{-pf(i)} \leq 1$. $\square$

Theorem 28. *Let f be a non-decreasing, recursive function such that $\sum_{i\geq 0}(p+1)^{-f(i)} \leq 1$ and is effectively computable. Then $\chi^r(G)$ is in $\mathrm{FQ}(f(\chi^r(G)), F_p^{\emptyset'''})$.*

Proof. This can be obtained by combining Theorem 22 with Lemma 10. □

We have not been able to obtain a converse for Theorem 28. There are different partial converses for recursive and highly recursive graphs.

Theorem 29 (only for recursive graphs). *If $\chi^r(G)$ is in $\mathrm{FQ}(f(\chi^r(G)-\chi(G)), F_p^{\emptyset'''})$, then $\sum_{i\geq 0}(p+1)^{-f(i)} \leq 1$.*

Proof. This is the $Y=\emptyset$ case of Theorem 35. □

Theorem 30 (only for highly recursive graphs). *If $\chi^r(G)$ is in $\mathrm{FQ}(g(\chi(G)), \emptyset''')$, then for all n,*

$$g(n) > \left\lceil \frac{\log(n)}{\log(p+1)} \right\rceil - 1$$

Proof. This is the $Y=\emptyset$ case of Theorem 39. □

Theorem 31. *Let f be a non-decreasing, recursive function such that $\sum_{i\geq 0} 2^{-pf(i)} \leq 1$ and is effectively computable. There exists an oracle $A \equiv_T \emptyset'''$ such that $\chi^r(G)$ is in $\mathrm{FQ}(f(\chi^r(G)), F_p^A)$.*

Proof. Similar to the proof of Theorem 26. □

Theorem 32. *Let X be any set and f be any function. If $\chi^r(G)$ is in $\mathrm{FQ}(f(\chi^r(G)), F_p^X)$ then $\sum_{i\geq 0} 2^{-pf(n)} \leq 1$.*

Proof. If $\chi^r(G) \in \mathrm{FQ}(f(\chi^r(G)), F_p^X)$, then $\chi^r(G) \in \mathrm{FQ}(pf(\chi^r(G)), X)$. By Theorem 12, $\sum_{i\geq 0} 2^{-pf(n)} \leq 1$. □

7. Parallel and mixed queries

In this section we explore the questions raised in Section 5 in a parallel setting. Most of the proofs use a combination of techniques from the last two sections and hence will be omitted.

Theorem 33. *Let Y be any set such that $K \not\leq_T Y$. Let X be any set. If $\chi(G)$ is in $\mathrm{FQ}^Y(f(\chi(G)), F_p^X)$, then $\sum_{i\geq 0}(p+1)^{-f(i)} \leq 1$.*

Proof. Similar to the proof of Theorem 25, except that Lemma 13 is used instead of Lemma 8. □

Theorem 34. *Let f be a nondecreasing recursive function such that $\sum_{i\geq 0}(p+1)^{-f(i)}\leq 1$, and is effectively computable. Then $\chi^r(G)$ is in $\mathrm{FQ}^K(f(\chi^r(G)-\chi(G)), F_p^{\emptyset'''})$.*

Proof. Combine the techniques used in the proofs of Theorems 15 and 28. □

The above theorem is optimal if recursive graphs are considered.

Theorem 35 (only for recursive graphs). *Let Y be any set such that $\emptyset''' \not\leq_T Y$. If $\chi^r(G)$ is in $\mathrm{FQ}^Y(f(\chi^r(G)-\chi(G)), F_p^{\emptyset'''})$, then $\sum_{i\geq 0}(p+1)^{-f(i)}\leq 1$.*

Proof. Combine the techniques used in the proofs of Theorems 17 and 25. □

Theorem 36. *Let f be a nondecreasing recursive function such that $\sum_{i\geq 0}2^{-pf(i)}\leq 1$, and is effectively computable. There exists an oracle $A\equiv_T\emptyset'''$ such that $\chi^r(G)$ is in $\mathrm{FQ}^K(f(\chi^r(G)-\chi(G)), F_p^A)$.*

Proof. Combine the techniques used in the proofs of Theorem 15 and 26. □

The above theorem is optimal if recursive graphs are considered.

Theorem 37 (only for recursive graphs). *Let Y be any set such that $\emptyset''' \not\leq_T Y$. Let X be any set. If $\chi^r(G)$ is in $\mathrm{FQ}^Y(f(\chi(G)), F_p^X)$, then $\sum_{i\geq 0}2^{-pf(n)}\leq 1$.*

Proof. If $\chi^r(G)$ is in $\mathrm{FQ}^Y(f(\chi(G)), F_p^X)$, then $\chi^r(G)$ is in $\mathrm{FQ}^Y(pf(\chi(G)), X)$. By Theorem 14, $\sum_{i\geq 0}2^{-pf(n)}\leq 1$. □

If only highly recursive graphs are considered, then we can cut down on the number of queries substantially.

Theorem 38 (only for highly recursive graphs). *$\chi^r(G)$ is in*

$$\mathrm{FQ}^K\left(\left\lceil\frac{\log(\chi(G)}{\log(p+1)}\right\rceil, F_p^{\emptyset'''}\right).$$

Proof. Combine the technique used in the proof of Theorem 19 with $(p+1)$-ary search [9]. □

The above theorem is optimal in terms of queries to $F_p^{\emptyset'''}$.

Theorem 39 (only for highly recursive graphs). *Let Y be any set such that $\emptyset''' \not\leq_T Y\oplus\emptyset''$. If $\chi^r(G)$ is in $\mathrm{FQ}^Y(g(\chi(G)), \emptyset''')$, then for all n,*

$$g(n) > \left\lceil\frac{\log(n)}{\log(p+1)}\right\rceil - 1.$$

Proof. Similar to the proof of Thereom 20. The key fact needed is that if $\emptyset''' \not\leq_T Y \oplus \emptyset''$, then $\chi^r_{[2,2b-1]}$ is not in

$$\mathrm{FQ}^Y\left(\left\lceil \frac{\log(b)}{\log(p+1)} \right\rceil - 1, F_p^{\emptyset'''}\right).$$

This is proven in [3]. □

If we allow unlimited access to K and use an oracle other than $\emptyset'''$, then we can cut down on the number of queries to that oracle.

Theorem 40 (only for highly recursive graphs). *Let f be a nondecreasing recursive function such that $\sum_{i\geq 0} 2^{-pf(i)} \leq 1$ and is effectively computable. There exists an oracle $A \equiv_T \emptyset'''$ such that $\chi^r(G)$ is in*

$$\mathrm{FQ}^K\left(\left\lceil \frac{\log(\chi(G))}{p} \right\rceil, F_p^A\right).$$

Proof. Combine the techniques used in the proofs of Theorems 19 and 26. □

The above theorem is optimal in terms of the number of queries to be more powerful oracle A.

Theorem 41. *Let Y be any set such that $\emptyset''' \not\leq_T Y$, X be any set, and g be any function. If $\chi^r(G)$ is in $\mathrm{FQ}^Y(g(\chi(G)), F_p^x)$, then for all n,*

$$g(n) > \left\lceil \frac{\log(n)}{p} \right\rceil - 1.$$

Proof. If $\chi^r(G) \in \mathrm{FQ}^Y(f(\chi^r(G)), F_p^X)$, then $\chi^r(G) \in \mathrm{FQ}^Y(pg(\chi^r(G)), X)$. By Theorem 20, for all n, $pg(n) > \lceil \log n \rceil - 1$, from which the theorem follows. □

8. Summary and open problems

We summarize our results in the following table. Let $p \geq 1$ be a fixed natural number. The function χ returns the chromatic number of a graph. The function χ^r returns the recursive chromatic number of a graph. Unless otherwise specified, a result holds for both recursive and highly recursive graphs. If X is used in a statement of a result, then that result holds when X is replaced by any set. If Y is used in a statement about chromatic number, then the intention is that the statement holds for any Y such that $K \not\leq_T Y$. If Y is used in a statement about recursive chromatic number, then the intention is that the statement holds for any Y such that $\emptyset''' \not\leq_T Y$; unless it is a statement about parallel queries to $\emptyset'''$ in which case the intention is that the statement holds for all Y such that $\emptyset''' \not\leq_T \emptyset'' \oplus Y$. If A

is used in a statement about chromatic number, then we are saying that a set A, $A \equiv_T K$, exists; if A is used in a statement about recursive chromatic number, then we are saying that a set A, $A \equiv_T \emptyset'''$, exists.

Throughout this section

(1) f represents any function such that $\sum_{i \geq 0} 2^{-f(i)} \leq 1$ and is effectively computable,

(2) g represents any function such that $\sum_{i \geq 0} 2^{-g(i)} > 1$,

(3) f_p represents any function such that $\sum_{i \geq 0} (p+1)^{-f_p(i)} \leq 1$ and is effectively computable,

(4) g_p represents any function such that $\sum_{i \geq 0} (p+1)^{-g_p(i)} > 1$.

In some cases our lower bounds do not (numerically) match our upper bounds. These lower bounds are marked with $**$. We conjecture that the lower bounds can be improved to match the upper bounds. In some cases we have the condition $\emptyset''' \not\leq_T Y \oplus \emptyset''$ instead of $\emptyset''' \not\leq_T Y$. These cases are marked with $*$. We conjecture that the lower bound with the condition $\emptyset''' \not\leq_T Y$ can be obtained. The following conjecture would imply that $\emptyset''' \not\leq_T Y$ would suffice: "If $\#_n^{\emptyset'''} \in \mathrm{FQ}^Y(1, F_{n-1}^{\emptyset'''})$, then $\emptyset''' \not\leq_T Y$."

I. Serial queries without help

$$\chi \in \mathrm{FQ}(f(\chi(G)), K)$$
$$\chi \notin \mathrm{FQ}(g(\chi(G)), X)$$
$$\chi^r \in \mathrm{FQ}(f(\chi^r(G)), \emptyset''')$$
$$\chi^r \notin \mathrm{FQ}(g(\chi^r(G)), X)$$

II. Serial queries with help

(a) *Recursive graphs*

$$\chi \in \mathrm{FQ}(f(\chi(G)), K)$$
$$\chi \notin \mathrm{FQ}^Y(g(\chi(G)), X)$$
$$\chi^r \in \mathrm{FQ}^K(f(\chi^r(G) - \chi(G)), \emptyset''')$$
$$\chi^r \notin \mathrm{FQ}^Y(g(\chi^r(G) - \chi(G)), X)$$

(b) *Highly recursive graphs*

$$\chi \in \mathrm{FQ}(f(\chi(G)), K)$$
$$\chi \notin \mathrm{FQ}^Y(g(\chi(G)), X)$$
$$\chi^r \in \mathrm{FQ}^K(\lceil \log(\chi(G)) \rceil, \emptyset''')$$
$$\chi^r \notin \mathrm{FQ}^Y(\lceil \log(\chi(G)) \rceil - 1, X)$$

III. Parallel queries without help

(a) Using queries to $F_p^K(F_p^{\emptyset'''})$ to compute $\chi(\chi^r)$

$$\chi \in \mathrm{FQ}(f_p(\chi(G)), F_p^K)$$
$$\chi \notin \mathrm{FQ}(g_p(\chi(G)), F_p^K)$$
$$\chi^r \in \mathrm{FQ}(f_p(\chi(G)), F_p^{\emptyset'''})$$
$$\chi^r \notin \mathrm{FQ}(g_p(\chi(G)), F_p^{\emptyset'''}) \quad \text{(for recursive graphs)}$$
$$\chi^r \notin \mathrm{FQ}(\lceil \log(\chi(G)) \rceil - 1, F_p^{\emptyset'''}) ** \quad \text{(for highly recursive graphs}$$

(b) Using queries to F_p^A, where A is any oracle, to compute $\chi(\chi^r)$

$$\chi \in \mathrm{FQ}\left(\frac{f(\chi(G))}{p}, F_p^A\right)$$
$$\chi \notin \mathrm{FQ}\left(\frac{g(\chi(G))}{p}, F_p^X\right)$$
$$\chi^r \in \mathrm{FQ}\left(\frac{f(\chi^r(G))}{p}, F_p^A\right)$$
$$\chi^r \notin \mathrm{FQ}\left(\frac{g(\chi^r(G))}{p}, F_p^X\right)$$

IV. Parallel queries with help

(a) Using queries to $F_p^K(F_p^{\emptyset'''})$ to compute $\chi(\chi^r)$, but allowing unlimited queries to a set Y where $K \not\leq_T Y$ ($\emptyset''' \not\leq_T Y$ or $\emptyset''' \not\leq_T Y \oplus \emptyset''$ when noted)

(i) *Recursive graphs,*

$$\chi \in \mathrm{FQ}(f(\chi(G)), F_p^K)$$
$$\chi \notin \mathrm{FQ}^Y(g(\chi(G)), F_p^K)$$
$$\chi^r \in \mathrm{FQ}^K(f(\chi^r(G) - \chi(G)), F_p^{\emptyset'''})$$
$$\chi^r \notin \mathrm{FQ}^Y(g(\chi^r(G) - \chi(G)), F_p^{\emptyset'''}) * \quad (\emptyset''' \not\leq_T Y \oplus \emptyset'')$$

(ii) *Highly recursive graphs,*

$$\chi \in \mathrm{FQ}(f(\chi(G)), F_p^K)$$
$$\chi \notin \mathrm{FQ}^Y(g(\chi(G)), F_p^K)$$
$$\chi^r \in \mathrm{FQ}^K\left(\left\lceil \frac{\log(\chi(G)}{\log(p+1)} \right\rceil, F_p^{\emptyset'''}\right)$$
$$\chi^r \notin \mathrm{FQ}^Y\left(\left\lceil \frac{\log(\chi(G)}{\log(p+1)} \right\rceil - 1, F_p^{\emptyset'''}\right) * \quad (\emptyset''' \not\leq_T Y \oplus \emptyset'')$$

(b) For this subsection (IV.b) we look at computing χ (χ^r) with a limited number of calls to F_p^A (where A is any oracle) and an unlimited number of calls to some Y such that $K \not\leq_T Y$ ($\emptyset''' \not\leq_T T$).

(i) *Recursive graphs*

$$\chi \in \mathrm{FQ}\left(\frac{f(\chi(G))}{p}, F_p^A\right)$$

$$\chi \notin \mathrm{FQ}^Y\left(\frac{g(\chi(G))}{p}, F_p^X\right)$$

$$\chi^r \in \mathrm{FQ}^K\left(\frac{f(\chi^r(G)-\chi(G))}{p}, F_p^A\right)$$

$$\chi^r \notin \mathrm{FQ}^Y\left(\frac{g(\chi^r(G)-\chi(G))}{p}, F_p^X\right)$$

(ii) *Highly recursive graphs*

$$\chi \in \mathrm{FQ}\left(\frac{f(\chi(G))}{p}, F_p^A\right)$$

$$\chi \notin \mathrm{FQ}^Y\left(\frac{g(\chi(G))}{p}, F_p^X\right)$$

$$\chi^r \in \mathrm{FQ}^K\left(\left\lceil\frac{\log(\chi(G))}{p}\right\rceil, F_p^A\right)$$

$$\chi^r \notin \mathrm{FQ}^Y\left(\left\lceil\frac{\log(\chi(G))}{p}\right\rceil - 1, F_p^X\right)$$

References

[1] D.R. Bean, Effective coloration, J. Symbolic Logic 41 (1976) 469–480.

[2] R.J. Beigel, Unbounded searching algorithms, SIAM J. Comput., to appear.

[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, W.I. Gasarch, J.T. Gill and J. Owings, Terse, superterse, and verbose sets, Univ. of Maryland at College Park, Dept. of Computer Science, TR-1806 (1987).

[5] J.L. Bentley and A.C.C. Yao, An almost algorithm for unbounded searching, Inform. Process. Lett. 5 (August 1976) 82–87.

[6] H.G. Carstens and P. Pappinghaus, Recursive coloration of countable graphs, Ann. Pure Appl. Logic 25 (1983) 19–45.

[7] R.G. Gallagher, Information Theory and Reliable Communication (Wiley, New York, 1968).

[8] D.E. Knuth, Supernatural numbers, in: D.A. Klarner, ed., The Mathematical Gardener (1981) 310–325.

[9] C.P. Kruskal, Searching, merging, and sorting in parallel computation, IEEE Trans. Computers 32 (October 1983) 942–946.

[10] H. Rogers, Jr., Theory of Recursive Functions and Effective Computability (McGraw-Hill, New York, 1967).

[11] J.M. Schmerl, Recursive colorings of graphs. Canad. J. Math. 32 (1980) 821–830.

[12] R.I. Soare, Recursively Enumerable Sets and Degrees, Omega Series (Springer, Berlin and New York, 1987).