The Relative Complexity of NP Search Problems

Paul Beame*
Computer Science and Engineering
University of Washington
Seattle, WA 98195
USA
beame@cs.washington.edu

Stephen Cook†
Computer Science Dept.
University of Toronto
Toronto, Ontario
Canada M5S 1A4
sacook@cs.toronto.edu

Jeff Edmonds‡
I.C.S.I.
Suite 600
1947 Center Street
Berkeley, CA 94704-1198
edmonds@icsi.berkeley.edu

Russell Impagliazzo§
Computer Science and Engineering
UC, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0114
russell@cs.ucsd.edu

Toniann Pitassi¶
Mathematics and Computer Science
University of Pittsburgh
Pittsburgh, PA
toni@cs.pitt.edu

June 23, 1997

*Research supported by NSF grants CCR-8858799 and CCR-9303017

†Research supported by an NSERC operating grant and the Information Technology Research Centre

‡Supported by an NSF postdoctoral fellowship and by a Canadian NSERC postdoctoral fellowship

§Research Supported by NSF YI Award CCR-92-570979, Sloan Research Fellowship BR-3311, grant #93025 of the joint US-Czechoslovak Science and Technology Program, and USA-Israel BSF Grant 92-00043

¶Research supported by an NSF postdoctoral fellowship and by NSF Grant CCR-9457782

Abstract

Papadimitriou introduced several classes of NP search problems based on combinatorial principles which guarantee the existence of solutions to the problems. Many interesting search problems not known to be solvable in polynomial time are contained in these classes, and a number of them are complete problems. We consider the question of the relative complexity of these search problem classes. We prove several separations which show that in a generic relativized world, the search classes are distinct and there is a standard search problem in each of them that is not computationally equivalent to any decision problem. (Naturally, absolute separations would imply that $\mathsf{P} \neq \mathsf{NP}$.) Our separation proofs have interesting combinatorial content and go to the heart of the combinatorial principles on which the classes are based. We derive one result via new lower bounds on the degrees of polynomials asserted to exist by Hilbert's Nullstellensatz over finite fields.

1 Introduction

In the study of computational complexity, there are many problems that are naturally expressed as problems "to find" but are converted into decision problems to fit into standard complexity classes. For example, a more natural problem than determining whether or not a graph is 3-colorable might be that of finding a 3-coloring of the graph if it exists. One can always reduce a search problem to a related decision problem and, as in the reduction of 3-coloring to 3-colorability, this is often by a natural self-reduction which produces a polynomially equivalent decision problem.

However, it may also happen that the related decision problem is not computationally equivalent to the original search problem. This is particularly important in the case when a solution is guaranteed to exist for the search problem. For example, consider the following problems:

1. Given a list $a_1, ..a_n$ of residues mod p, where $n > \log p$, find two distinct subsets $S_1, S_2 \subseteq \{1, ..n\}$ so that $\Pi_{i \in S_1} a_i \bmod p = \Pi_{i \in S_2} a_i \bmod p$. The existence of such sets is guaranteed by the pigeonhole principle, but the search problem is at least as difficult as discrete log modulo p. It arises from the study of cryptographic hash functions.

2. Given a weighted graph G, find a travelling salesperson tour T of G that cannot be improved by swapping the successors of two nodes. This problem arises from a popular heuristic for TSP called 2-OPT. Again, the existence of such a tour is guaranteed, basically because any finite set of numbers has a least element, but no polynomial-time algorithm for this problem is known.

3. Given an undirected graph G where every node has degree exactly 3, and a Hamiltonian circuit H of G find a different Hamiltonian circuit H'. A solution is guaranteed to exist by an interesting combinatorial result called Smith's Lemma. The proof constructs an exponential size graph whose odd degree nodes correspond to circuits of G, and uses the fact that every graph has an even number of odd degree nodes.

In [JPY88, Pap90, Pap91, Pap94, PSY90], an approach is outlined to classify the exact complexity of problems such as these, where every instance has a solution. Of course, one could (and we later will) define the class TFNP of all search problems with this property, but this class is not very nice. In particular, since the reasons for being a member of TFNP seem as diverse as all of mathematics, different combinatorial lemmas being required for different problems, it seems unlikely that TFNP has any complete problem.

As an alternative, the papers above concern themselves with "syntactic" sub-classes of TFNP, where all problems in the sub-class can be presented in a fixed, easily verifiable format. These classes correspond to combinatorial lemmas: for problems in the class, a solution is guaranteed to exist by this lemma. For example, the class PPA is based on the lemma that every graph has an even number of odd-degree nodes; the class PLS is based on the lemma that every directed acyclic graph has a sink; and the class PPP on the pigeonhole principle. The third example above is thus in PPA, the second in PLS and the first in PPP. The class PPAD is a directed version of PPA; the combinatorial lemma here is this: "Every directed graph with an imbalanced node (indegree different from outdegree) must have another imbalanced node." It is shown in [Pap94] that all these classes can be defined in a syntactic way.

As demonstrated in the papers listed above, these classes satisfy the key litmus test for an interesting complexity class: they contain many natural problems, some of which are complete. These problems include computational versions of Sperner's Lemma, Brouwer's Fixed Point Theorem, the Borsuk-Ulam Theorem, and various problems for finding economic equilibria. Thus they provide useful insights into natural computational problems. ¿From a mathematical point of view they are also interesting: they give a natural means of comparison between the "algorithmic power" of combinatorial lemmas. Thus, it is important to classify the inclusions between these classes, both because such classification yields insights into the relative plausibility of efficient algorithms for natural problems, and because such inclusions reveal relationships between mathematical principles.

Many of these problems are more naturally formulated as type 2 computations in which the input, consisting of local information about a large set, is presented by an oracle. Moreover, each of the complexity classes we consider can be defined as the type 1 translation of some natural type 2 problem. We thus consider the relative complexity of these search classes by considering the relationships between their associated type 2 problems. Our main results are several type 2 separations which imply that in a generic relativized world, the type 1 search classes we consider are distinct and there is a standard problem in each of them that is not equivalent to any decision problem. (Naturally, absolute type 1 separations would imply that $\mathsf{P} \neq \mathsf{NP}$.) In fact, our separations are robust enough that they apply also to the Turing closures of the search classes with respect to any generic oracle. Such generic oracle separations are particularly nice because generic oracles provide a single view of the relativized world, much as do probability 1 random oracles, but unlike random oracles, they do not change the fundamental complexity of problems that do not depend on the oracle.

The proofs of our separations have quite interesting combinatorial content. In one example, via a series of reductions using methods similar to those in [BIK+94], we derive our result via new lower bounds on the degrees of polynomials asserted to exist by Hilbert's Nullstellensatz over finite fields. The lower bound we obtain for the degree of these polynomials is $\Omega(n^{1/4})$ where n is the number of variables and this is substantially stronger than the $\Omega(\log^* n)$ bound that was shown (for a somewhat different system) in [BIK+94].

2 The Search Classes

2.1 Type 1 and type 2 problems

A decision problem in NP can be given by a polynomial time relation R and a polynomial p

such that $R(x,c)$ implies $|c| \leq p(|x|)$. The decision problem is "given x, determine whether there exists c such that $R(x,c)$ ". The associated NP search problem is "given x, find c such that $R(x,c)$ holds, if such c exists". We denote the search problem by a multi-valued function Q, where $Q(x) = \{c \mid R(x,c)\}$; that is $Q(x)$ is the set of possible solutions for problem instance x. The problem is *total* if $Q(x)$ is nonempty for all x. FNP denotes the class of all NP search problems, and TFNP denotes the set of all total NP search problems.

The sub-classes of TFNP defined by Papadimitriou all have a similar form. Each input x implicitly determines a structure, like a graph or function, on an exponentially large set of "nodes", in that computing local information about node v (e.g., the value of the function on v or the set of v's neighbors) can be done in polynomial-time given x and v. A solution is a small sub-structure, a node or polynomial size set of nodes, with a property X that can be verified using only local information. The existence of the solution is guaranteed by a lemma "Every structure has a sub-structure satisfying property X." For example, an instance of a problem in the class PPP of problems proved total via the pigeon-hole principle, consists of a $poly(n)$ length description x of a member $f_x = \lambda y.f(x,y)$ of a family of (uniformly) polynomial-time functions from $\{0,1\}^n$ to $\{0,1\}^n - 0^n$. A solution is a pair y_1, y_2 of distinct n bit strings with $f_x(y_1) = f_x(y_2)$, which of course must exist.

It is natural to present such search problems as second order objects $Q(\alpha, x)$, where α is a function ("oracle" input) which, when appropriate, can describe a graph by giving local information (for example $\alpha(v)$ might code the set of neighbors of v). Thus $Q(\alpha, x)$ is a set of strings; the possible solutions for problem instance (α, x). As before we require that solutions be checkable in polynomial time, and the verifying algorithm is allowed access to the oracle α.

Proceeding more formally, we consider strings x over the binary alphabet {0,1}, functions α from strings to strings, and type 2 functions (i.e. operators) F taking a pair (α, x) to a string y. We follow Townsend [Tow90] in defining such an F to be polynomial time computable if it is computable in deterministic time that is polynomial in $|x|$ with calls to α at unit cost. Note that since the time bound depends on $|x|$ and not α, a machine computing F may not have time to read a long value $\alpha(y)$ returned by the oracle. We can define a *type 2 search problem* Q to be a function that associates with each string function α and each string x a set $Q(\alpha, x)$ of strings that are the allowable answers to the problem on inputs α and x. Such a problem Q is in FNP^2 if Q is polynomial-time checkable in the sense that $y \in Q(\alpha, x)$ is a type 2 polynomial-time computable

Class	Name of Q	Instance of Q	Solutions for Q
PPA	*LEAF*	Undirected Graph on $\{0,1\}^{\leq n}$ with degree ≤ 2	any leaf $c \neq 0^n$ 0^n, if 0^n is not a leaf
PPAD	*SOURCE.OR.SINK*	Directed graph on $\{0,1\}^{\leq n}$ with in-degree, out-degree ≤ 1	any source or sink $c \neq 0^n$ 0^n, if 0^n is not a source
PPADS	*SINK*	Directed graph on $\{0,1\}^{\leq n}$ with in-degree, out-degree ≤ 1	any sink $c \neq 0^n$ 0^n, if 0^n is not a source
PPP	*PIGEON*	Function f $f : \{0,1\}^{\leq n} \to \{0,1\}^{\leq n}$	any pair (c, c'), $c \neq c'$ with $f(c) = f(c') \neq 0^n$ any c'' with $f(c'') = 0^n$

Figure 1: Some complexity classes of search problems

predicate, and all elements of $Q(\alpha, x)$ are of length polynomially bounded in $|x|$.

A problem Q is *total* if $Q(\alpha, x)$ is nonempty for all α and x. TFNP^2 is the subclass of total problems in FNP^2. An algorithm A *solves* a total search problem Q if and only if for each function α and string x, $A(\alpha, x) \in Q(\alpha, x)$. FP^2 consists of those problems in TFNP^2 which can be solved by deterministic polynomial time algorithms.

2.2 The classes defined

Each of Papadimitriou's classes can be defined as a set of type 1 problems reducible to a fixed type 2 problem.

We say that a type 2 problem Q_1 is *many-one reducible* to a type 2 problem Q_2 (written $Q_1 \leq_m Q_2$) if there exist type 2 polynomial-time computable functions F, G, and H, such that $H(\alpha, x, y)$ is a solution to Q_1 on input (α, x) for any y that is a solution to Q_2 on input $(G[\alpha, x], F(\alpha, x))$, where $G[\alpha, x] = \lambda z.G(\alpha, x, z)$. It is straightforward to check that this reducibility is transitive. Below we apply this definition to the case in which Q_1 is type 1, which can be done by treating Q_1 as a type 2 problem which ignores its function input α. The definition then becomes: $H(x, y)$ is a solution to Q_1 on input x for any y that is a solution to Q_2 on input $(G[x], F(x))$, where $G[x] = \lambda z.G(x, z)$. If Q_2 is also type 1, then the definition is the same with G and its arguments omitted.

Associated with each type 2 problem Q in TFNP^2 we define the type 1 class $\mathsf{C}Q$ of all problems in TFNP which are many-one reducible to Q. Thus each class $\mathsf{C}Q$ is closed under many-one reducibility within TFNP. We summarize Papadimitriou's classes in this format in Figure 2.1. Each class is of the form $\mathsf{C}Q$ for some $Q \in \mathsf{TFNP}^2$ which we name and briefly describe. The notation $\{0,1\}^{\leq n}$ denotes the set of *nonempty* strings of length n or less. We assume that n is given in unary as the standard part of the input to Q.

For example, in the problem *LEAF* the arguments (α, x) describe a graph $G = G(\alpha, |x|)$ of maximum degree two whose nodes are the nonempty strings of length $|x|$ or less, and $\alpha(u)$ codes the set of 0, 1, or 2 nodes adjacent to u. An edge (u, v) is present in G iff both $\alpha(u)$ and $\alpha(v)$ are proper codes and $\alpha(u)$ contains v and $\alpha(v)$ contains u. A *leaf* is a node of degree one. We want the node $0...0 = 0^n$ to be a leaf (the *standard leaf* in G). The search problem *LEAF* is: 'Given α and x, find a leaf of $G = G(\alpha, |x|)$ other than the standard one, or output 0...0 if it is not a leaf of G'. That is, $LEAF(\alpha, x)$ is the set of nonstandard leaves of $G(\alpha, |x|)$ together with, in case 0...0 is not a leaf of G, the node 0...0.

It should be clear that *LEAF* is a total NP search problem and hence a member of TFNP^2. Further, since the search space has exponential size, a simple adversary argument shows that no deterministic polynomial time algorithm solves *LEAF*. Hence *LEAF* is not in FP^2.

Continuing with this example, we see from Figure 2.1 that Papadimitriou's class PPA is the class of problems in TFNP which are many-one reducible to *LEAF*. Thus a member Q of PPA is presented by a trio of polynomial-time functions F, G, and H. For each input x to Q, $G[x]$ codes a graph of maximum degree 2 whose nodes are the nonempty strings of length $|F(x)|$ or less. For each node u in this graph, $G(x, u)$ is a string encoding the set of nodes adjacent to u. For each nonstandard leaf u of this graph, $H(x, u)$ must be a member of $Q(x)$. Possibly $Q(x)$ contains additional strings not of this form, but since $Q \in \mathsf{TFNP}$, the relation '$y \in Q(x)$' must be recognizable in polynomial-time.

The classes defined from these problems are interesting for more than just the lemmas on which they are based. There are many natural problems in them. Here are some examples in the first order classes PPAD, PPA, and PPP from [Pap94]. Problems in PPAD include, among others: finding a panchromatic simplex asserted to exist by Sperner's Lemma, finding a fixed point of a function asserted to exist by Brouwer's Fixed Point Theorem, and finding the antipodal points on a sphere with equal function values asserted to exist by the Borsuk-Ulam Theorem (where in each case the input structure itself is given implicitly via a polynomial time Turing machine, but could be given by an oracle). Several of these are complete. Problems in PPA not known to be in PPAD include finding a second solution of an underdetermined system of polynomial equations modulo 2 that is asserted to exist by Chevalley's Theorem and finding a second Hamiltonian path in an odd-degree graph given the first. The problem Pigeonhole Circuit is a natural complete problem for PPP.

The class PPADS is called PSK in [Pap90], where it is incorrectly said to be equivalent to PPAD. We note here that a natural problem complete for PPADS is Positive Sperner's Lemma (for dimensions three and above), which is exactly like Sperner's Lemma except that only a panchromatic simplex that is positively oriented is allowed as a solution.

2.3 Relativized classes and Turing reducibility

By an *oracle* A we mean simply a set of strings. We can use our second order setting to define relativized classes by replacing a function argument α by an oracle A, where now we interpret A as a characteristic function: $A(x) = 1$ if $x \in A$ and $A(x) = 0$ otherwise. Thus we define TFNP^A to be the set of all type 1 problems $Q(A, *)$, for $Q \in \mathsf{TFNP}^2$. Note that this is more restrictive than simply requiring Q to be in FNP^2 and $Q(A, *)$ to be total.

Define the relativized class $(CQ)^A$ to be the subclass of TFNP^A consisting of all problems $Q_1(A, *)$, where Q_1 is any problem in TFNP^2 many-one reducible to Q. Equivalently $(CQ)^A$ is the set of all problems in TFNP^A many-one-A reducible to Q, where now the suffix A means that the reduction is allowed to query the oracle A; precisely, A replaces α as arguments to the functions F, G, and H used in the definition of many-one reducibility in the previous subsection. Notice that $(\mathsf{C}Q)^A = \mathsf{C}Q$ when $A \in \mathsf{P}$.

The following theorem shows that the problem of separating relativized NP search classes is equivalent to separating them relative to any generic oracle [BI87], and also equivalent to showing that there is no reduction between the corresponding type 2 problems.

Theorem 1: Let $Q_1, Q_2 \in \mathsf{TFNP}^2$. The following are equivalent: (i) Q_1 is many-one reducible to Q_2; (ii) For all oracles A, $(\mathsf{C}Q_1)^A \subseteq (\mathsf{C}Q_2)^A$; (iii) There exists a generic oracle G such that $(\mathsf{C}Q_1)^G \subseteq (\mathsf{C}Q_2)^G$.

The proof appears in [CIY].

In order to state the full power of our separation results, we now define a more general form of reduction among total search problems: We say Q_1 is *polynomial-time Turing reducible* to Q_2, (or simply Q_1 is *reducible* to Q_2, written $Q_1 \leq Q_2$), if there is some polynomial-time machine M that on input (α, x) and an oracle for Q_2 outputs some $y \in Q_1(\alpha, x)$. (Recall that M's input α is a string function which it accesses via oracle calls.) (See Figures 2 and 3.) For each query to the Q_2 oracle, M must provide some pair (β, z) as input where β is a string function. For M to be viewed as a polynomial-time machine, the β's that M specifies must be computable in polynomial time given the things to which M has access: α, x, and the sequence t of answers that M has received from previous queries to Q_2. We thus view the reduction as a pair of polynomial-time algorithms:

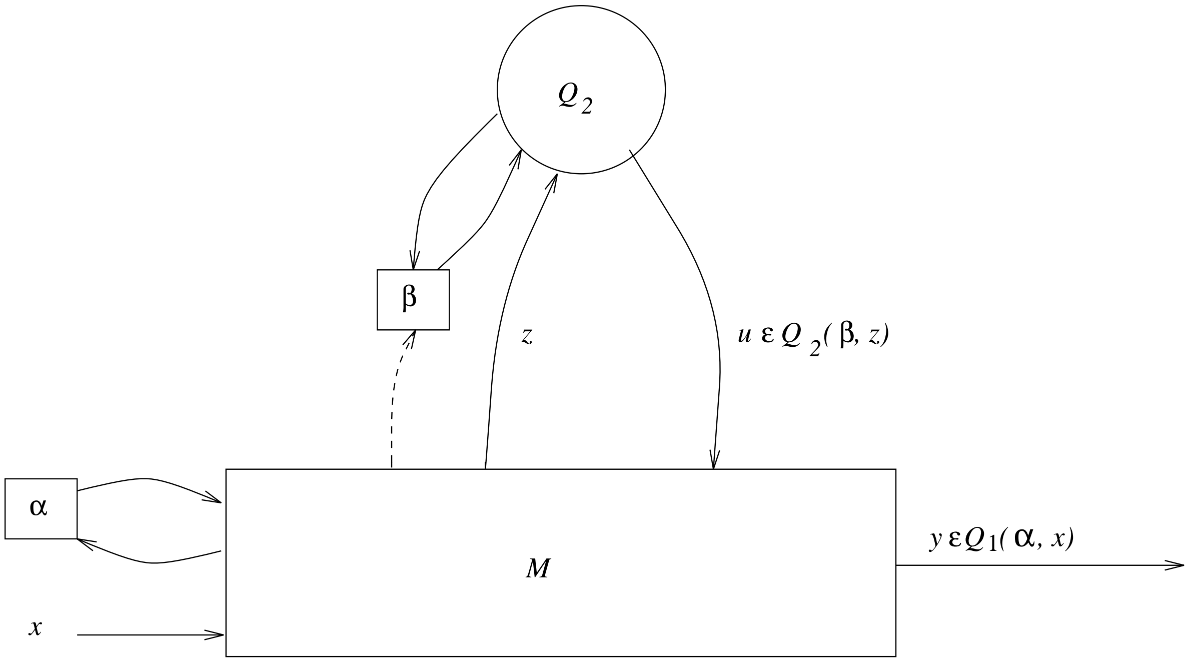

Figure 2: Reducing Q_1 to Q_2

M, and another polynomial-time machine M^* which computes β as a function of α, x, and t. M must produce a correct y for all choices of answers that could be returned by Q_2.

Notice that Q_1 is many-one reducible to Q_2 iff Q_1 reduces to Q_2 as above, but M makes exactly one query to an instance of Q_2.

A statement similar to Theorem 1 holds for the case of Turing reductions with the many-one closures replaced by Turing closures for the type 1 classes. All reductions we exhibit are many-one reductions, so with this theorem they give inclusions or alternative characterizations of the classes defined in [Pap94]. All separations we exhibit hold even against Turing reductions, so they show oracle separations between the Turing closures of the related type 1 search classes and these separations apply to all generic oracles.

2.4 Some simple reductions

It is easy to see that *SOURCE.OR.SINK* $\leq_m$ *LEAF*, by ignoring the direction information on the input graph. Also it is immediate that *SOURCE.OR.SINK* $\leq_m$ *SINK*.

It is not hard to see that *SINK* $\leq_m$ *PIGEON*: Let G be the input graph for *SINK*. The corresponding input function f to *PIGEON* maps nodes of G as follows. If v is a sink of G then let $f(v) = 0...0$; if there is an edge from v to u in G then let $f(v) = u$; and if v is isolated in G, let $f(v) = v$. Then the possible answers to *PIGEON* coincide exactly with the possible answers to *SINK*.

Our main results are that all three of these reductions fail in the reverse direction even when allowing more general Turing reductions. The containments of the corresponding type 1 classes

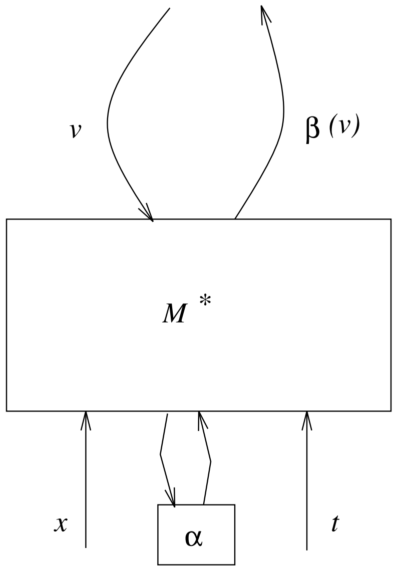

Figure 3: Detail showing β's computation

(with respect to any oracle) are shown in Figure 4.

2.5 Equivalent problems

We say that two problems are *equivalent* if each is reducible (under $\leq$) to the other, and they are *many-one equivalent* if each is many-one reducible (under $\leq_m$) to the other. It is interesting (and also relevant to our separation arguments) that there are several problems many-one equivalent to *LEAF*, based on different versions of the basic combinatorial lemma "every graph has an even number of odd-degree nodes." Strictly speaking, *LEAF* is based on a special case of this lemma, where the graph has degree at most two. A more general problem, denote it *ODD*, is the one in which the degree is not two, but bounded by a polynomial in the length of the input x. That is, $\alpha(v)$ codes a set of polynomially many, as opposed to at most two, nodes, and we are seeking a node $v \neq 0...0$ of odd degree (or 0...0 if that node is not of odd degree).

Another variant of the same lemma is this: "Every graph with an odd number of nodes has a node with even degree." To define a corresponding problem, denoted *EVEN*, we would have $\alpha(v)$ again be a polynomial set of nodes, only now $\alpha(0...0) = \emptyset$. This last condition will essentially leave node 0...0 out of the graph thus rendering the number of nodes odd. We are seeking a node $v \neq 0...0$ of even degree (or 0...0 if that node is not isolated).

In the special case where the graph has maximum degree one, this version of the lemma is "there is no perfect matching of an odd set of nodes." An input pair (α, x) now codes a graph $GM(\alpha, |x|)$ which is a partial matching. The nodes, as before, are the nonempty strings of length $|x|$ or less, and there is an edge between nodes u and v iff (i) $u \neq v$, (ii) $\alpha(v) = u$, (iii) $\alpha(u) = v$, and (iv) neither u nor v is the standard node 0...0. Thus 0...0 is always unmatched, and we are seeking a second unmatched (or *lonely*) node v. This search problem is denoted *LONELY*.

Theorem 2: The problems *LEAF*, *ODD*, *EVEN*, and *LONELY* are all many-one equivalent.

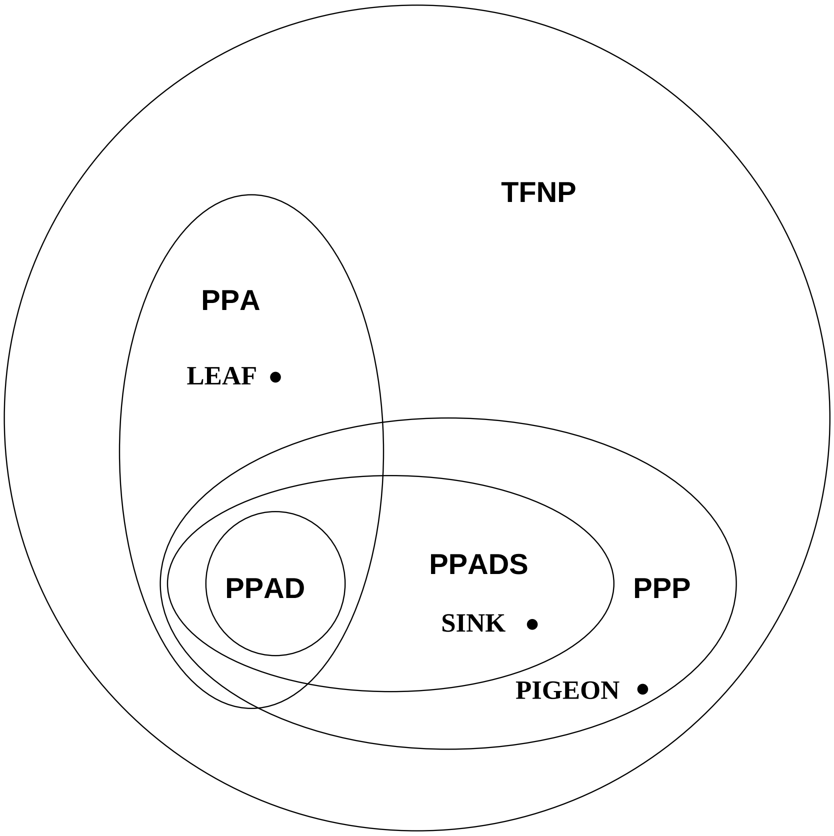

Figure 4: Search class relationships in a generic relativized world

Proof: To show that $\mathit{LEAF} \leq_m \mathit{LONELY}$ consider an input (α, x) to *LEAF*, representing a graph $G = G(\alpha, |x|)$. We transform (α, x) to an input $(\beta, x1)$ to *LONELY*. We describe β implicitly by describing the partial matching $G2 = GM(\beta, |x1|)$. $G2$ has all nodes of G, plus a copy v' of each such node v. We place edges in $G2$ in such a way that the leaves of G are precisely the unmatched nodes in $G2$. For each isolated node v in G there is an edge in $G2$ matching node v and its copy v'. For each edge $\{u, v\}$ in G there is an edge in $G2$ matching one of u or v' to one of v or v'. Which of the node or its copy to use for the edge in $G2$ corresponding to $\{u, v\}$ in G is locally determined as follows. For a node v in G with one incident edge $\{u, v\}$, in graph $G2$ its copy v' is used for the corresponding edge and the node v is left unmatched. For a node v in G with two incident edges, $\{u, v\}$ and $\{v, w\}$, we decide which of v or v' to use for each edge based on the lexicographic ordering of the node names of its neighbors, u and w. If u lexicographically precedes w then in $G2$ the node v will be used in the edge corresponding to $\{u, v\}$ and the node v' will be used in the edge corresponding to $\{v, w\}$. If w lexicographically precedes u then in $G2$ the node v will be used in the edge corresponding to $\{v, w\}$ and the node v' will be used in the edge corresponding to $\{u, v\}$.

Note that for each node v in $G2$, the mate $\beta(v)$ can be determined with at most four calls to α. It is each to verify that, as claimed, the leaves of G are precisely the unmatched nodes in $G2$. Thus $\mathit{LEAF}(\alpha, x) = \mathit{LONELY}(\beta, x1)$.

That $\mathit{LONELY} \leq_m \mathit{EVEN}$ is obvious. To convert any problem in *EVEN* into one in *ODD*, just add to the graph all edges of the form $\{v0, v1\}$ joining nodes with all bits the same except for the last; unless this edge is already present, in which case remove it. This will make 0...0 into the standard leaf, and make all even-degree nodes into odd-degree nodes and vice versa.

Finally, *ODD* $\leq_m$ *LEAF* follows from the "chessplayer algorithm" of [Pap90, Pap94] which makes explicit the local edge-pairing argument that is involved in the standard construction of Euler tours. For completeness we give this construction: Given an input graph G to *ODD* we transform it to an input graph GL to *LEAF*. Let $2d$ be an upper bound on the degree of any node in G. The nodes of GL are pairs (v, i) where v is a node in G and $1 \leq i \leq d$, plus the original nodes of G. Suppose that the neighbors of v in G are $v_1, \ldots, v_m$ in lexicographical order and v is, respectively, the $i_1, \ldots, i_m$-th neighbor of each of them in lexicographical order. Basically, the corresponding edges in GL are $\{(v, \lceil j/2 \rceil), (v_j, \lceil i_j/2 \rceil)\}$ for $j = 1, \ldots, m$. In this way the edges about each node in G are paired up consistently in GL creating a graph of maximum degree 2. It is easy to see that m is odd if and only if the node $(v, \lceil m/2 \rceil)$ is a leaf. We can make the name of the leaf node the same as in the original problem by replacing the node $(v, \lceil m/2 \rceil)$ by the node v if m is odd. The construction may be completed in polynomial time without much difficulty. □

One could give directed versions of *ODD* which would generalize *SOURCE.OR.SINK* to *IMBALANCE* and *SINK* to *EXCESS*, where instead of up to one predecessor and one successor, any polynomial number of predecessors and successors is allowed. In these definitions, the search problem would be to find a nonstandard node with an imbalance of indegree and outdegree (respectively, an excess of indegree over outdegree.) The Euler tour argument given above shows that these new problems are equivalent to the original ones.

3 Separation Results

3.1 PPA^G is not included in PPP^G

Theorem 3: *LONELY* is not reducible to *PIGEON*.

Proof: Suppose to the contrary that *LONELY* $\leq$ *PIGEON*. Let M and M^* be as in the definition of $\leq$ in Section 2.3 (see also Figures 1 and 2). Consider an input (α, x) to *LONELY* and the corresponding graph $G = GM(\alpha, n)$, where $n = |x|$. On input (α, x), the machines M and M^* make queries to the oracles α and *PIGEON* and finally M outputs a lonely node in G. Our task is to find α and x and suitable answers to the queries made to *PIGEON* so that M's output is incorrect.

Fix some large n and some x of length n. Then the nodes of G are the nonempty strings of length n or less, and the edges of G are determined by the values $\alpha(v)$ for v a node of G. For any string v not a node of G we specify $\alpha(v) = \lambda$ (the empty string). (Such values are irrelevant to the graph G and hence to the definition of a correct output.) Also we specify $\alpha(0...0) = \lambda$, since the standard node should be unmatched. For nonstandard nodes v we specify $\alpha(v)$ implicitly by specifying the edges of G. We do this gradually as required to answer queries. The goal is to answer all queries without ever specifying any particular nonstandard node v to be unmatched. In that way M is forced to output a lonely node without knowing one, and we can complete the specification of G so that its answer is incorrect.

In general, after i steps of M's computation, we will have answered all queries made so far by specifying that certain edges are present in G. These edges comprise a partial matching σ_i, where the number of edges in σ_i is bounded by a polynomial in n. Suppose that step $i+1$ is a query v to α. If that query cannot be answered by σ_i and our initial specifications, then we set $\alpha(v) = w$, where $w \neq 0...0$ is any unmatched node, and form σ_{i+1} by adding the edge $\{v, w\}$ to σ_i.

Now suppose step $i+1$ is a query (β, z) to *PIGEON*, specifying a function $f = f_{<\beta,|z|>}$. Here f is the restriction of β to the set of nonempty strings of length $|z|$ or less, except $f(c) = 0...0$ in case $\beta(c)$ is either empty or of length greater than $|z|$. Then we must return either a pair (c, c'), with $c \neq c'$ and $f(c) = f(c') \neq 0...0$, or c'' with $f(c'') = 0...0$. Our task is to show that a possible return value can be determined by adding only polynomially many edges to the partial matching σ_i (i.e. to G), and without specifying that any particular node in G is unmatched.

The value $f(c)$ is determined by the computation of M^* on inputs x, α, c, and t (which codes the answers to the previous queries to *PIGEON*). We have fixed x, part of α (i.e. part of G), and the answers to previous queries, so $f(c)$ depends only on the unspecified part of G. Thus $f(c)$ can be expressed via a decision tree $T'(c)$ whose vertices query the unspecified part of G. Each internal vertex of the tree $T'(c)$ is labelled with a node u in G (representing a query) and each edge in $T'(c)$ leading from a vertex labelled u is labelled either with a node v in G (indicating that u is matched to v in G) or $\emptyset$ (indicating that u is a lonely node in G). If u has already been matched in σ_i or if $u = 0...0$ then we know the answer to the query, so we assume that no such node u appears on the tree, either as an edge label or vertex label. Also we assume that no node u occurs more than once on a path, since this would give either inconsistent or redundant information. Each leaf of $T'(c)$ is labelled by the output string $f(c)$ of M^* under the computation determined by the path to the leaf.

The runtime of M^* is bounded by a polynomial in the lengths of its string inputs, which in turn are bounded by a polynomial in n (since M is time-bounded by a polynomial in the length n of its string input x). This runtime bound on M^*, say k, bounds the height of each tree. If n is sufficiently large, then the number of nodes in G minus the number of nodes in the partial matching σ_i far exceeds k.

For each string c in the domain of f, define $T(c)$ to be the tree $T'(c)$ where all branches with outcome $\emptyset$ on any query are pruned. That is, we shall be interested in the behavior of this decision tree when α evades the answer "lonely node".

Notice that each path from the root to a leaf in a tree $T(c)$ designates a partial matching σ of up to k edges matching up to $2k$ nodes in G. Thus we call each tree $T(c)$ a *matching decision tree*. We call two partial matchings σ and τ *compatible* if $\sigma \cup \tau$ is also a partial matching, i.e. they agree on the mates of all common nodes. Notice that the partial matching designated by any path in $T(c)$ is compatible with the original matching σ_i, since only nodes unmatched by σ_i can appear as labels in $T(c)$.

Case I: Some path p in a tree $T(c)$ leads to a leaf labelled with the standard node $0...0$, indicating that $f(c) = 0...0$. Then we set $\sigma_{i+1} = \sigma_i \cup \sigma$, where σ is the partial matching designated by the path to this leaf. This insures that c is a legitimate answer to our current query to *PIGEON*, and we answer that query with c.

We say that a path p in tree $T(c)$ is *consistent* with a path p' in $T(c')$ if p and p' designate compatible matchings.

Case II: There are consistent paths p and p' in distinct trees $T(c)$ and $T(c')$ such that p and p' have the same leaf label. Then we set $\sigma_{i+1} = \sigma_i \cup \sigma \cup \sigma'$, where σ and σ' are the partial matchings designated by p and p'. This insures that $f(c) = f(c')$, so (c, c') is a legitimate answer to our current query to *PIGEON*, and we answer that query with (c, c').

The lemma below insures that for sufficiently large n, either Case I or Case II must hold. Thus we have described for all cases the partial matching σ_i associated with step i of M's computation. When M completes its computation after, say m steps, and outputs a node y in G, the partial matching σ_m contains only polynomial in n edges, and whenever G extends this partial matching and we answer queries to *PIGEON* as described, the computation of M will be determined and the output will be y. In particular, we can choose a G consistent with σ_m in which y is not a lonely node, so M makes a mistake. $\Box$

Lemma 4: Suppose that the nodes comprising potential queries and answers in the matching decision trees described above come from a set of size K, and each tree has height k. If $K \geq 4k^2$, then either Case I or Case II must hold.

Proof: Suppose to the contrary that neither Case I nor Case II holds. We think of the strings in the domain of f as *pigeons* and the leaf labels as *holes*. If there are N possible pigeons 1,...,N then we have N "pigeon" trees T_1,...,T_N and $N-1$ possible holes 1,...,$N-1$ (recall 0...0 is not a possible leaf label). If the leaf label of path p in tree T_i is j, then pigeon i gets mapped to hole j under any partial matching consistent with p. All trees have height at most k.

We say that a path p *extends* a path p' if the partial matching designated by p extends the partial matching designated by p'.

We will show how to construct a new collection of consistent "hole" matching decision trees $H_1, ..., H_{N-1}$ with possible leaf labels 1,...,N and "unmapped". Intuitively, the decision tree H_j attempts to find a pigeon mapping to hole j or to prove that there is no such pigeon. Formally, H_j will have the following properties: For any path p in H_j with leaf label i, there is a (unique) path p' in T_i with leaf label j so that p extends p'. For any path p in H_j with leaf label "unmapped", p is inconsistent with any path in any T_i with leaf label j. The construction is very similar to an argument due to Riis [Rii93] which is itself similar to the proof that if a Boolean function and its negation both can be written in disjunctive normal form with terms of size $\leq d$, then the function has a Boolean decision tree of height $\leq d^2$. (This last result was implicit in [HH87, HH91], [BI87], [Tar89], and appears explicitly in [IN88].)

Fix $j \leq N-1$ and let P_j be the set of all paths in pigeon trees with leaf label j. Since Case II does not hold, the paths in P_j are mutually inconsistent. We describe H_j implicitly as a strategy for querying the purported matching α. The strategy proceeds in stages, and makes at most $2k$ queries in each stage. Let τ_s represent the set of known edges of G at the beginning of stage s. Then in stage s, if possible, we find a path p_s in P_j consistent with τ_s. If this is impossible, we halt and output "unmapped". If we find a path p_s, we query α for all endpoints of edges of G in p_s that are not contained in τ_s. We update τ_{s+1} to include the newly found edges. If τ_{s+1} includes the edges of some path p with leaf label j from some $T_{i'}$, we halt and output i'; otherwise we begin stage $s+1$.

¿From the above description, it is clear that when H_j halts, the known edges either extend a unique path p in one of the pigeon trees with leaf label j or are inconsistent with every such path. Since each path in P_j has at most k edges, at most $2k$ nodes are queried per stage. To see that there are at most k stages before H_j halts, we show by induction that in stage s every path p in P_j consistent with τ_s has at least s edges in common with τ_s. After k stages any remaining consistent path in P_j must be entirely contained in τ_k in which case the algorithm halts.

To prove the claim, observe that for any stage s, any two paths in P_j consistent with τ_s must match some node not touched by τ_s since they are inconsistent with each other. In particular this means that the set of endpoints of p_s includes at least one node v_p not touched by τ_s from each path p in P_j consistent with τ_s. Since τ_{s+1} contains an edge matching each endpoint of p_s, any path p in P_j that remains consistent with τ_{s+1} will have the additional edge touching v_p in common with τ_{s+1} as required to show the claim.

Since there are at most k stages and at most $2k$ nodes are queried per stage, each path in tree H_j has length at most $2k^2$. We now extend all paths in H_j by adding 'dummy queries' so that each path has length exactly $2k^2$. (The outcome of each dummy query is ignored, and the leaf label of each extended path is the former label of its ancestor.)

Now get new pigeon trees T_i' by first simulating T_i to get a path p in pigeon tree T_i with leaf label j and then simulating H_j, but not asking queries already answered in p, i.e., restrict H_j by p. Along such a path, T_i' still outputs j. Note that any path q in H_j which may be followed in this manner must be consistent with p and thus it must extend some path in P_j by the construction of H_j. Since the paths in P_j are mutually inconsistent, q must extend p itself. This means that the new path in T_i' constructed while following q gives rise to the same partial matching as q does.

Therefore any path p' with leaf label j in the pigeon tree T_i' has the exact same edges as a path p with leaf label i in some hole tree H_j. Thus all paths in both sets of trees have the same length, $2k^2$. Further, since no two paths in T_i' have the exact same edges, this defines a 1-1 mapping from the paths of the T_i''s into the paths of the H_j's. But this is impossible, because there is one more pigeon tree than hole tree, and all trees with the same depth have the same number of paths. □

¿From Theorems 1, 2 and 3 we conclude

Corollary 5: $\mathsf{PPA}^G \not\subseteq \mathsf{PPP}^G$ for any generic oracle G.

3.2 PPP^G is not included in PPADS^G

Using the same technique, we can also show that *PIGEON* is not reducible to *SINK*. Now we construct inputs (α, x) to *PIGEON* in such a way that each can be viewed as a mapping f from $[0, N]$ to $[1, N]$ with the property that the mapping is one-to-one on all but one element of the range. For each query to *SINK*, and for each node c in the directed graph D, the computation of M^* to determine $\beta(c)$ can be expressed via a tree $T(c)$ whose nodes query the function f. The outcome of a query u is the unique element v such that $f(u) = v$. As in the previous proof, the paths in $T(c)$ describe partial matchings from $[0, N]$ into $[1, N]$. (We are only interested in these paths, since they are the ones that evade an answer to the *PIGEON* problem.)

The leaves of $T(c)$ are labelled by the output of M^*. For vertex c, the notation $\{c' \to c,\ c \to c''\}$ means that there is an edge from c' to c, and an edge from c to c'' in the underlying graph D. Either c' or c'' may have the value $\emptyset$, indicating that c is a source, or respectively, sink vertex. Note that because the standard node 0 is a source, all leaves of $T(0)$ are labelled $\{\emptyset \to 0,\ 0 \to c''\}$. We want to show that either the trees $T(c)$ are inconsistent, or that there is some vertex c and some path p in $T(c)$ such that at the leaf label of path p, vertex c is designated as a sink.

For every vertex c, except for the standard source vertex, 0, we will make two copies of $T(c)$; the two copies will be identical except for the leaf labellings. If a path p in $T(c)$ is labelled $\{c' \to c,\ c \to c''\}$, then the path p in the "domain" copy of $T(c)$, $T_1(c)$, will be labelled by $c \to c''$, and the path p in the "range" copy of $T(c)$, $T_2(c)$, will be labelled by $c' \to c$. For vertex 0, there is only one copy, the "domain" copy. Thus, we have one more tree representing "domain" elements than trees representing "range" elements. Assume for the sake of contradiction that all trees are consistent, and that for every path in every domain tree, $T_1(c)$, the leaf label is $c \rightarrow c''$, for some c'' not equal to $\emptyset$. As in the previous argument, we will extend the trees so that: each tree has the same height k, and furthermore, there is a 1-1 mapping from paths in the domain trees to paths in the range trees. This is done by first extending every path p in range tree $T_2(c)$ with leaf label $c' \rightarrow c$, $c' \neq \emptyset$, by the tree $T_1(c')$ restricted by p. Then, all range trees are extended to the same height by adding dummy queries. Finally, every path p in domain tree $T_1(c)$ with leaf label $c \rightarrow c''$, is extended by the tree $T_1(c'')$ restricted by p. But this violates the pigeonhole principle, because there are more domain trees than range trees, and the total number of paths in every tree is the same. Thus, the machine cannot solve *PIGEON*.

3.3 PPADS^G is not included in PPA^G

In section 3.1 we reduced our separation problem to a purely combinatorial question, namely to show that a family of matching decision trees with certain properties could not exist. In this section we again reduce our problem to a similar combinatorial question with a somewhat different kind of decision tree. This question is more difficult than our previous one and we need to apply a new method of attack, introduced in [BIK$^+$94], that is based on lower bounds on the degrees of polynomials given by Hilbert's Nullstellensatz.

More precisely, we show how we can naturally associate an unsatisfiable system of polynomial equations $\{Q_i(\bar{x}) = 0\}$ over GF[2] with each family of decision trees with the specified properties. By Hilbert's Nullstellensatz, the unsatisfiability of these polynomial equations implies the existence of polynomials P_i over GF[2] such that $\sum_i P_i(\bar{x})Q_i(\bar{x}) = 1$. However, our association shows something stronger, namely that if the family of decision trees exists then these coefficient polynomials must also have very small degree ($\log^{O(1)} n$ where n is the number of variables.)

Finally, in the technical heart of the argument, we show that for the family of polynomials we derive, $\mathcal{PHP}_N^{N+s}$, any coefficient polynomials allowing us to generate 1 require large degree, at least $n^{1/4}$. This is an interesting result in its own right since the bound for the coefficients of the system in [BIK$^+$94] was only $\Omega(\log^* n)$. We give the proof of this result in the next section.

Theorem 6: *SINK* is not reducible to *LONELY*.

(The Appendix contains a weaker result with a simpler proof.)

Proof: Suppose to the contrary that $SINK \leq LONELY$. We proceed as in the proof of Theorem 3, except now the reducing machine M takes as input (α, x) which codes a directed graph $G = GD(\alpha, n)$, where $n = |x|$, makes queries to the oracles α and *LONELY* and finally outputs a sink node in G. Our task this time is to find α and x and answers to the queries to *LONELY* so that M's output is incorrect.

We will need a couple of convenient bits of terminology. Recall that G is a directed graph of maximum in-degree and out-degree at most 1. We will call such graphs *1-digraphs*. A *partial 1-digraph* π over a node set V is a partial edge assignment over V. It specifies a collection, $E = E(\pi)$, of edges over V, and a collection $V^{source} \subseteq V$ such that $G(V, E)$ is a 1-digraph and for $v \in V^{source} = V^{source}(\pi)$ there is no edge of the form $u \rightarrow v$ in E. The set E indicates 'included' edges, the set V^{source} indicates 'excluded' edges. The *size* of a partial 1-digraph is $|E \cup V^{source}|$.

Fix some large n and some x of length n. The nodes of G are the non-empty strings of length n or less, and the edges of G are determined by the values of $\alpha(v)$ as before and $\alpha(0...0)$ tells us that $0...0$ is a source. The computation is simulated as in the proof of Theorem 3 except that we build a partial 1-digraph σ_i containing only a polynomial number of edges and we consider queries (β, z) to *LONELY*. In this case we must return a lonely node, c, in the graph $GM = GM(\beta, z)$ ($c = 0...0$ if $0...0$ has a neighbor) where β is defined in the usual way by machine M^*. We will show that a possible value of c can be determined by adding only polynomially many edges to σ_i and without specifying a sink node in G. Again, there is a natural notion of consistency that we can assume holds without loss of generality.

We first obtain a collection of trees in a similar manner to that of the proof of Theorem 3. For node c in graph GM, the computation of M^* can be expressed as a function of the graph G via a tree $T(c)$ whose nodes query the graph G. Without loss of generality, G can be accessed via queries of the form $(pred, v)$, and $(succ, v)$, where v is a node of G. The outcome of a query $(pred, v)$ is an ordered pair $w \to v$ indicating that there is an edge in G from w to v; similarly the outcome of a query $(succ, v)$ is an ordered pair $v \to w$ indicating that there is an edge in G from v to w. In either case, w can be $\emptyset$, indicating that u is a source in the first case, or a sink in the second case. For a given query there is one outcome for each vertex w (or $\emptyset$) except when such a label would violate the rule that the edge labels on a branch, taken together, produce a 1-digraph. Each leaf in the tree $T(c)$ is labelled to indicate the output of M^*, namely an unordered pair $\{c, c'\}$ indicating that node c is adjacent to node c' in the undirected graph GM, or $\emptyset$ indicating that c is lonely. The height of each $T(c)$ is bounded by the runtime of M^*, say ℓ', which is in turn bounded by some polynomial in n.

For each node c, we first prune the tree $T(c)$ defined above by removing all branches with outcome $u \to \emptyset$ on any query. That is, we restrict our interest to situations in which the oracle α evades the answer "u is a sink vertex". The rest of the argument of this section shows that, because of the consistency condition on M^*, there is some node c such that tree $T(c)$ must have a leaf designating that c is a lonely node. This will complete the proof: Suppose there is some branch σ with leaf label $\emptyset$ in some tree $T(c)$ with $c \neq 0...0$. It follows that $\sigma_{i+1} = \sigma_i \cup \sigma$ forces c to be a lonely node of GM. This allows us to fix the computation of the reduction in the $i+1$-st step and by induction we can force the reduction to make an error as in the proof of Theorem 3.

We now argue by contradiction that such a branch must exist in some $T(c)$ with $c \neq 0...0$. Assume that none of the leaves of $T(c)$ for any $c \neq 0...0$ have label $\emptyset$. Let N be the number of nodes in G minus 1 (for $0...0$) minus the size of σ_i. Let $s = |V^{source}(\sigma_i)| + 1$. Thus there are $N + s$ nodes that can appear in internal labels on the trees, s of which are guaranteed to be sources. The set of edge labels along any branch of $T(c)$ forms a partial 1-digraph of size at most ℓ' on these $N + s$ nodes. Thus we call each such tree $T(c)$ a *1-digraph decision tree*. Let $\mathcal{T}$ be the collection of trees $T(c)$ for all nodes c in GM. We identify a branch in a 1-digraph decision tree T with the partial 1-digraph determined by its edge labels and define $\mathrm{br}(T)$ to be the set of branches of T.

We call two partial 1-digraphs σ and τ *compatible* if $\sigma \cup \tau$ is also a partial 1-digraph. Notice that since β is consistent, the collection $\mathcal{T}$ is also *consistent*: That is, if σ is a branch of $T(c)$ with leaf label $\{c, c'\}$ then all branches τ in $T(c')$ that are compatible with σ must have leaf label $\{c, c'\}$.

Given a consistent collection $\mathcal{T}$, we can define a new collection of 1-digraph decision trees $\mathcal{T}^* = \{T^*(c) \mid c \neq 0...0\}$ that satisfies an even stronger consistency condition:

For each node c, define $T^*(c)$ to be the result of the following operation: For each c' and each branch σ of $T(c)$ with leaf label $\{c, c'\}$ append the tree $T(c')$ rooted at the leaf of σ and simplify the resulting tree. Remove all branches inconsistent with σ and collapse any branches that are consistent with σ. (For example, if σ contains the edge $u \to v$, and an internal node of $T(c')$ is labelled with the query $(succ, u)$ or $(pred, v)$, then we replace that query node by the subtree reached by the edge labelled $u \to v$.) Note that since the original collection $\mathcal{T}$ was consistent, all new leaves added below a leaf labelled $\{c, c'\}$ will be correctly labelled $\{c, c'\}$. Furthermore, if τ is a branch in $T^*(c)$ with leaf label $\{c, c'\}$, then τ is also a branch in $T^*(c')$ with leaf label $\{c, c'\}$. Note that all the trees in $\mathcal{T}^*$ now have height at most $\ell = 2\ell'$ and that $M = |\mathcal{T}^*|$ is odd. Such a collection $\mathcal{T}^*$ is very similar to the *generic systems* considered in [BIK+94]. The rest of the proof is devoted to showing that such a collection cannot exist.

Reducing the combinatorial problem to a degree lower bound

Given the partial 1-digraph σ_i, we can rename the nodes of the oracle graph G as follows: Remove all cycles in $E(\sigma_i)$ from G; remove all internal nodes on any path in $E(\sigma_i)$ and identify the beginning and end vertices of any such path; rename all source nodes as $N+1, \ldots, N+s$ with the standard source as $N+1$; rename all remaining non-source nodes to $1, \ldots, N$. We assume from now on that the internal labels of the trees of $\mathcal{T}^*$ have been renamed in this manner.

We will now show that if this collection of 1-digraph decision trees $\mathcal{T}^*$ exists then there is a particular unsatisfiable system of polynomial equations whose Nullstellensatz witnessing polynomials have small degree. This system is the natural expression of the sink counting principle for 1-digraphs that guarantees the totality of *SINK*.

DEFINITION 3.1: Let $\mathcal{S}_N^{N+s}$ be the following system of polynomial equations in variables $x_{i,j}$ with $i \in [0, N+s]$, $j \in [1, N]$:

$$(\sum_{j \in [1,N]} x_{i,j}) - 1 = 0$$

one for each $i \in [1, N+s]$, and

$$(\sum_{i \in [0,N+s]} x_{i,j}) - 1 = 0$$

one for each $j \in [1, N]$, and

$$x_{i,j} \cdot x_{i,k} = 0$$

one for each $i \in [1, N+s]$, $j \neq k$, $j, k \in [1, N]$, and

$$x_{i,k} \cdot x_{j,k} = 0$$

one for each $i \neq j$, $i, j \in [0, N+s]$, $k \in [1, N]$.

The variables $x_{i,j}$ describe a directed graph on vertices $[1, N+s]$ with vertices $[N+1, N+s]$ guaranteed to be source vertices. The variable $x_{i,j}$, $i \neq 0$, describes whether or not there is an edge from i to j. The variable $x_{0,k}$ indicates whether or not vertex k is a source vertex. A solution to the above equations would imply that there is a 1-digraph with source vertices but no sink vertex. Since this is impossible, there cannot exist a solution to $\mathcal{S}_N^{N+s}$.

Write $\mathcal{S}_N^{N+s} = \{Q_i'(\bar{x}) = 0\}_i$. We call any expression of the form $\sum_i P_i'(\bar{x})Q_i'(\bar{x})$ where the $P_i'(\bar{x})$ are polynomials a *linear combination* of the Q_i'. The degree of such a linear combination is the maximum of the degrees of the P_i' polynomials. (We say that the polynomial 0 has degree -1.) We now show that if the collection $\mathcal{T}^*$ exists then there is a linear combination of the Q_i''s over GF[2] that equals 1 and has degree at most $\ell - 1$. (Such a result, without the degree bound, would follow directly from Hilbert's Nullstellensatz.)

Given a partial 1-digraph π over $[1, N+s]$ with $[N+1, N+s]$ as source vertices, the monomial

$$X_\pi = \Big(\prod_{i \to j \in E(\pi)} x_{i,j}\Big) \cdot \Big(\prod_{j \in V^{source}(\pi)} x_{0,j}\Big)$$

is the natural translation of π into the polynomial realm ($X_\pi = 1$ if π is empty.)

Lemma 7: Let T be a 1-digraph decision tree of height at most ℓ over $[1, N+s]$ with $[N+1, N+s]$ as source vertices and suppose that $2\ell < N$. Then the polynomial $P_T(\bar{x}) = \sum_{\pi \in \mathrm{br}(T)} X_\pi \; - 1$ can be expressed as a linear combination of degree at most $\ell - 1$.

Proof: The proof proceeds by induction on the number of internal vertices of T. If T has no internal vertices then it has one branch of height 0, $P_T(\bar{x}) = 0$, and all coefficient polynomials in the linear combination are 0 which is of degree -1. Thus the lemma holds in this case.

Suppose now that T has at least one internal vertex and has height ℓ. Then it has some internal vertex v all of whose children are leaves. Let π be the partial 1-digraph that labels the path from the root of the tree to v and let T' be the 1-digraph decision tree with the children of v removed (the leaf label of v in T' will be immaterial.) Applying the inductive hypothesis to T' which has one fewer internal vertex than T, we get that $P_{T'}(\bar{x})$ is some linear combination of the Q_i' of degree at most $\ell - 1$.

The difference between $P_T(\bar{x})$ and $P_{T'}(\bar{x})$ is that we have removed the monomial for the branch π in T' and replaced it by the sum of the monomials for all branches in T extending π. Note also that X_π has degree at most the depth of v which is at most $\ell - 1$.

We have two cases to consider. If v is labelled with the query $(pred, j)$ for some $j \in [1, N]$ then j has no predecessors in $E(\pi)$, $j \notin V^{source}(\pi)$, and

$$P_T(\bar{x}) = P_{T'}(\bar{x}) + X_\pi \cdot \Big(\sum_{i \in \{0\} \cup S} x_{i,j} \; - 1\Big)$$

where S is the set of all $i \in [1, N+s]$ that have no successors in $E(\pi)$. It is easy to see that for any $i \in [1, N+s] \setminus S$, $X_\pi \cdot x_{i,j}$ is a multiple of some $x_{i,k} \cdot x_{i,j}$ (with $k \neq j$) of degree at most $\ell - 2$ so $X_\pi \cdot \sum_{i \in [1,N+s] \setminus S} x_{i,j}$ is a linear combination of degree at most $\ell - 2$. Then

$$\begin{aligned} & X_\pi \cdot \Big(\sum_{i \in \{0\} \cup S} x_{i,j} \; - 1\Big) \\ &= \; X_\pi \cdot \Big(\sum_{i \in [0, N+s]} x_{i,j} \; - 1\Big) - X_\pi \cdot \sum_{i \in [1, N+s] \setminus S} x_{i,j} \end{aligned}$$

is a linear combination of degree at most $\ell - 1$ since $\sum_{i \in [0,N+s]} x_{i,j} \; - 1$ is one of the Q' polynomials. Thus $P_T(\bar{x})$ also is a linear combination of degree at most $\ell - 1$.

Similarly, if v is labelled with the query $(succ, i)$ for some $i \in [1, N+s]$ then i has no successors in $E(\pi)$ and

$$P_T(\bar{x}) = P_{T'}(\bar{x}) + X_\pi \; \Big(\sum_{j \in S'} x_{i,j} \; - 1\Big)$$

where S' is the set of all $j \in [1, N]$ that have no predecessors in $E(\pi)$ and are not in $V^{source}(\pi)$. Again $X_\pi \cdot \sum_{j \in [1,N]-S'} x_{i,j}$ is a linear combination of degree at most $\ell - 2$ and

$$X_\pi \cdot (\sum_{j \in S'} x_{i,j} \ - 1) = X_\pi \cdot (\sum_{j \in [1,N]} x_{i,j} \ - 1) - X_\pi \cdot \sum_{j \in [1,N] \setminus S'} x_{i,j}$$

is a linear combination of degree at most $\ell - 1$ since $\sum_{j \in [1,N]} x_{i,j} \ - 1$ is one of the Q' polynomials. Again it follows that $P_T(\bar{x})$ is a linear combination of degree at most $\ell - 1$.

The lemma follows by induction. □

Lemma 8: Suppose that $\mathcal{T}^*$ exists as defined above. Then $\sum_{T \in \mathcal{T}^*} \sum_{\pi \in \mathrm{br}(T)} X_\pi = 0$ over GF[2].

Proof: Thus over GF[2] the sum is 0. □

Lemma 9: If $\mathcal{T}^*$ exists as defined above then, over GF[2], there are $P'_i(\bar{x})$ of degree at most $\ell - 1$ such that $\sum_i P'_i(\bar{x}) Q'_i(\bar{x}) = 1$.

Proof: Defining $P_T(\bar{x})$ as in the statement of Lemma 7 we have

$$\begin{aligned} \sum_{T \in \mathcal{T}^*} P_T(\bar{x}) &= \sum_{T \in \mathcal{T}^*} (\sum_{\pi \in \mathrm{br}(T)} X_\pi \ - 1) \\ &= (\sum_{T \in \mathcal{T}^*} \sum_{\pi \in \mathrm{br}(T)} X_\pi) - |\mathcal{T}^*| \\ &= (\sum_{T \in \mathcal{T}^*} \sum_{\pi \in \mathrm{br}(T)} X_\pi) + 1 \end{aligned}$$

over GF[2] since $|\mathcal{T}^*|$ is odd.

Now by the definition of $\mathcal{T}^*$, for $T = T^*(c) \in \mathcal{T}^*$ any $\pi \in \mathrm{br}(T)$ has some leaf label $\{c, c'\}$ such that we also have $\pi \in \mathrm{br}(T^*(c'))$ with leaf label $\{c, c'\}$. This association pairs two copies of every branch in $\mathcal{T}^*$ so every X_π appears an even number of times in $\sum_{T \in \mathcal{T}^*} \sum_{\pi \in \mathrm{br}(T)} X_\pi$. Therefore this sum equals 0 over GF[2] and thus $\sum_{T \in \mathcal{T}^*} P_T(\bar{x}) = 1$ over GF[2].

By Lemma 7, $\sum_{T \in \mathcal{T}^*} P_T(\bar{x})$ is a linear combination of degree at most $\ell - 1$ and we obtain our desired result. □

It remains to show that there cannot exist small degree P'_i such that $\sum_i P'_i Q'_i = 1$ over GF[2]. We first argue that there is a simpler subset of the equations in $\mathcal{S}_N^{N+s}$, $\mathcal{PHP}_N^{N+s} = \{Q_i(\bar{x}) = 0\}$, such that for any $d \geq 1$, any linear combination of the Q'_i of degree at most d that equals 1 can be transformed into a linear combination of the Q_i of degree at most d that equals 1. We then argue our degree lower bound in terms of the Q_i. The equations in $\mathcal{PHP}_N^{N+s}$ are the natural encoding of the the pigeonhole principle stating that there is no function from a set of size $N + s$ to a set of size N.

Definition 3.2: $\mathcal{PHP}_N^{N+s}$ is the following system of polynomial equations in variables $x_{i,j}$ with $i \in [1, N+s]$, $j \in [1, N]$:

$$(\sum_{j \in [1,N]} x_{i,j}) - 1 = 0$$

one for each $i \in [1, N+s]$, and

$$x_{i,j} \cdot x_{i,k} = 0$$

one for each $i \in [1, N+s]$, $j \neq k$, $j, k \in [1, N]$, and

$$x_{i,k} \cdot x_{j,k} = 0$$

one for each $i \neq j$, $i, j \in [1, N+s]$, $k \in [1, N]$.

Lemma 10: Write $\mathcal{S}_N^{N+s} = \{Q_i'(\bar{x}) = 0\}$ and $\mathcal{PHP}_N^{N+s} = \{Q_i(\bar{x}) = 0\}$. For any $d \geq 1$, there is a linear combination of the Q_i' of degree at most d that equals 1 if and only if there is a linear combination of the Q_i of degree at most d that equals 1.

Proof: One direction is immediate. For the other direction, assume there exist polynomials P_i' of degree at most $d \geq 1$ such that $\sum_i P_i'(\bar{x})Q_i'(\bar{x}) = 1$. Now apply the substitution $x_{0,i} = 1 - (x_{1,i} + \ldots x_{N+s,i})$ to this linear combination. First notice that it doesn't change the degree of any coefficient monomials. There are two types of polynomials among the Q_i' that are not explicitly present among the Q_i: The first type is any 'range polynomial', i.e., $x_{0,i} + x_{1,i} + \ldots + x_{N+s,i} - 1$. But this becomes 0 under the substitution. The second type is of the form $x_{0,i} \cdot x_{k,i}$, for $k > 0$. However, under the substitution, the resulting combination is of degree 1 over the reduced system: $[1 - (x_{1,i} + \ldots + x_{N+s,i})] \cdot x_{k,i}$ is equal to $x_{k,i} - x_{k,i}^2$ plus a degree 0 combination of $x_{j,i} \cdot x_{k,i}$ for $0 < j \neq k$. Now $x_{k,i} - x_{k,i}^2$ is a degree 1 combination of the domain polynomial for k in the reduced system and some of the other polynomials since $-x_{k,i}(x_{k,1} + x_{k,2} + \ldots + x_{k,n} - 1)$ equals $x_{k,i} - x_{k,i}^2$ plus a degree 0 combination of $x_{k,j} \cdot x_{k,i}$ for $j \neq i$. Thus the degree of the combination in the reduced system is at most d. $\square$

By Theorem 12 proven in the next section we can now complete the proof of Theorem 6. Combining Theorem 12 with Lemma 9 and Lemma 10 we have that the existence of $\mathcal{T}^*$ implies that $\ell \geq \sqrt{2N}$. However, ℓ is also polynomial in $n < \log N$ which contradicts $\ell \geq \sqrt{2N}$ for n sufficiently large. Thus the collection $\mathcal{T}^*$ as defined above cannot exist.

$\square$

Corollary 11: $\mathsf{PPADS}^G \not\subseteq \mathsf{PPA}^G$ for any generic oracle G.

4 A Nullstellensatz degree lower bound for $\mathcal{PHP}_N^{N+s}$

In this section we prove the following theorem which is of independent interest.

Theorem 12: Write $\mathcal{PHP}_N^{N+s} = \{Q_i(\bar{x}) = 0\}$. Over GF[2], if $\sum_i P_i(\bar{x})Q_i(\bar{x}) = 1$ for polynomials P_i then one of them must have degree at least $\sqrt{2N} - 1$.

Let $P_i(\bar{x})$ be polynomials over GF[2] of degree at most d. We consider the class of assignments to the variables $\bar{x}$ that correspond to bi-partite matchings in $U_N^{N+s} = [1, N+s] \times [1, N]$, and examine the behavior of $\sum_i P_i(\bar{x})Q_i(\bar{x})$ under such assignments.

Given a bi-partite matching $M = \{\langle i_1, j_1 \rangle, \ldots, \langle i_m, j_m \rangle\} \subset U_N^{N+s}$ we naturally obtain the monomial $X_M = \prod_{\langle i,j \rangle \in M} x_{i,j}$ as well as the assignment such that $x_{i,j} \leftarrow 1$ if and only if $\langle i, j \rangle \in M$. (If

$M = \emptyset$ then $X_M = 1$.) Any monomial that is not of the form X_M for some bi-partite matching M will be 0 under all assignments we consider so we ignore such terms without loss of generality. In particular, we will not need to consider the Q_j that give the degree 2 equations in $\mathcal{PHP}_N^{N+s}$. Therefore, we can assume that we have the polynomial $\sum_{i=1}^{N+s} P_i(\bar{x})Q_i(\bar{x})$ where $Q_i(\bar{x}) = \sum_{j=1}^{N} x_{i,j} \ - 1$ and all monomials not of the form X_M for some matching M have been removed. Let the coefficient in P_i of the monomial X_M corresponding to matching M be a_M^i.

DEFINITION 4.1: Matching M *matches* i if $\langle i,j \rangle \in M$ for some $j \in [1, N]$. We write this formally as $i \in M$. If $i \in M$, we write $M - i$ for the matching $M - \{\langle i,j \rangle\}$ where j is the unique value such that $\langle i,j \rangle \in M$. Let $\mathrm{dom}(M) = \{i \in [1, N+s] \mid i \in M\}$ be the projection of M onto the first co-ordinate.

Since we only consider assignments over GF[2], we can assume that $a_M^i = 0$ if $i \in M$. The reason is that if $M = \{\langle i,k \rangle\} \cup (M - i)$, then $X_M = X_{M-i} \cdot x_{i,k}$ and

$$\begin{aligned} X_M \cdot Q_i &= X_{M-i} \cdot x_{i,k} \cdot (\sum_{j \in [1,N]} x_{i,j} \ - 1) \\ &= X_{M-i} \cdot (x_{i,k}^2 - x_{i,k}) = 0 \end{aligned}$$

since $x^2 - x = 0$ for all $x \in \mathrm{GF}[2]$.

By considering assignments corresponding to each bipartite matching M of size up to $d+1$ in turn, we inductively obtain an equation over GF[2] for the coefficient of X_M in $\sum_{i=1}^{N+s} P_i \cdot Q_i$ so that the combination equals 1 over GF[2]:

(1) $-\sum_{i \in [1,N+s]} a_\emptyset^i = 1$

(2) $\sum_{i \in M} a_{M-i}^i - \sum_{i \notin M} a_M^i = 0$, for all matchings $M \neq \emptyset$ on U_N^{N+s} with $|M| \leq d$

(3) $\sum_{i \in M} a_{M-i}^i = 0$, for all matchings M on U_N^{N+s} with $|M| = d+1$.

We will now show that the above system of equations (1)–(3) has a solution over GF[2] if and only if there does not exist a particular combinatorial design.

DEFINITION 4.2: Let $\mathcal{M}$ be a collection of matchings on U_N^{N+s} so that all matchings $M \in \mathcal{M}$ match $i \in [1, N+s]$. Define $\mathcal{M} - i$ to be the set of matchings $\bigoplus_{M \in \mathcal{M}} \{M - i\}$ where $\bigoplus$ operates like $\cup$ except that it only includes elements that appear in an odd number of its arguments.

DEFINITION 4.3: A *k-design for (1)–(3)* is a collection of matchings, $\mathcal{M}$, on U_N^{N+s} such that each matching in $\mathcal{M}$ has size at most k and such that the following conditions hold.

(a) The empty matching $M = \emptyset$ is in $\mathcal{M}$.

(b) The sets $\mathcal{M}_S = \{M \in \mathcal{M} \mid \mathrm{dom}(M) = S\}$ for $S \subset [1, N+s]$, $|S| \leq k$, satisfy $\mathcal{M}_{S - \{i\}} = \mathcal{M}_S - i$.

Lemma 13: Equations (1)–(3) have a solution over GF[2] if and only if there does not exist a $(d+1)$-design for (1)–(3).

Proof: We give the proof of the above lemma in the direction that we will need, although using basic linear algebra the converse direction can also be proven.

Suppose we have a $(d+1)$-design $\mathcal{M}$ for (1)–(3) and a solution for equations (1)–(3). We view the matchings $M \in \mathcal{M}$ as selecting a subset of the equations in (1)–(3), since there is one equation for each matching on U_N^{N+s} of size at most $d+1$. We consider the GF[2] sum of the selected equations. Condition (a) in the definition of a $(d+1)$-design requires that equation (1) is selected so the right-hand side of the sum is 1.

We will show that condition (b) in the definition of a $(d+1)$-design implies that the left-hand side of this sum is 0 which is a contradiction. Consider the coefficient of a_M^i in the sum. It occurs once (with coefficient -1) if $M \in \mathcal{M}$. It also occurs once (with coefficient $+1$) for each j such that $M \cup \{\langle i,j\rangle\} \in \mathcal{M}$. We rewrite this in terms of $S = \mathrm{dom}(M)$: There is a contribution of -1 if $M \in \mathcal{M}_S$ and a contribution of $+1$ if there are an odd number of j such that $M \cup \{\langle i,j\rangle\} \in \mathcal{M}_{S\cup\{i\}}$. The latter is true if and only if $M \in \mathcal{M}_{S\cup\{i\}} - i$. By condition (b) of the definition of a $(d+1)$-design, $\mathcal{M}_S = \mathcal{M}_{S\cup\{i\}} - i$ so the net coefficient of a_i^M is 0. $\square$

We now state the conditions under which we can produce designs.

Theorem 14: For any d such that $N \geq \binom{d+2}{2}$ there exists a $(d+1)$-design for (1)–(3).

By Theorem 14 if $N \geq \binom{d+2}{2} = (d+1)(d+2)/2$, there is a $(d+1)$-design for (1)–(3) and thus by Lemma 13 there is no solution to equations (1)–(3) and no polynomials P_i of degree d such that $\sum_i P_i \cdot Q_i = 1$. This proves Theorem 12. $\square$

The proof of Theorem 14 occupies the remainder of this section.

Definition 4.4: Let $[N]^{(k)} \subset [N]^k$ denote the set of k-tuples from $[1,N]$ that do not contain any repeated elements. For any set $S \subset [1, N+s]$, we can define a set of matchings $\mathcal{M}_S$ by giving an associated set $\mathcal{V}_S \subseteq [N]^{(|S|)}$ with the interpretation that if $S = \{i_1, \ldots, i_{|S|}\}$ where $i_1 < i_2 < \cdots < i_{|S|}$ then

$$\mathcal{M}_S = \{(\langle i_1, j_1\rangle, \ldots, \langle i_{|S|}, j_{|S|}\rangle) \mid (j_1, j_2, \ldots, j_S) \in \mathcal{V}_S\}$$

We use the notation $\mathcal{M}_S = M(S, \mathcal{V}_S)$.

The design that we produce will be symmetric in the following sense. For any two sets $S,\ S' \subset [1, N+s]$ with $|S| = |S'|$ we will have $\mathcal{V}_S = \mathcal{V}_{S'}$. We will use the notation $\mathcal{V}_k$ to denote $\mathcal{V}_S$ for $|S| = k$. In order to describe our design it will be convenient to define the following somewhat bizarre operation.

Definition 4.5: Let $v \in [N]^{(k)}$ and $I \subseteq [1,k]$, $I = \{i_1, \ldots i_{|I|}\}$ such that $i_1 < i_2 < \ldots < i_{|I|}$. Let $A \subseteq [N]^{(|I|)}$ be such that no element of v appears in any element of A. Define

$$\begin{aligned} v \bigotimes\nolimits_I A &= \{x \in [N]^{(k)} \mid \exists w \in A. \forall j \leq |I|.\ x_{i_j} = w_j \\ &\qquad \text{and } \forall i \in [1,k] - I.\ x_i = v_i\} \end{aligned}$$

This operation creates the set of tuples made by 'spreading out' some tuple in A into the positions indexed by I and filling the remaining positions with the corresponding entries from v. Note that if $I = \emptyset$ then $v \bigotimes_I A = \{v\}$ and if $I = [1,k]$ then $v \bigotimes_I A = A$.

DEFINITION 4.6: Let $\mathcal{V}_0 = \{()\}$, the set containing the empty tuple. For $k > 0$ let $v_k = (\binom{k}{2} + 1, \ldots, \binom{k+1}{2})$ and define

$$\mathcal{V}_k = \bigcup_{I \subset [1,k]} v_k \otimes_I \mathcal{V}_{|I|}$$

In order to understand this definition it will be convenient to represent each set $\mathcal{V}_k$ as an array, each of whose columns is a tuple in $\mathcal{V}_k$, and listed so that the columns are in order of decreasing size of the set I used in their construction. Using this representation, we have

$$\begin{aligned}
\mathcal{V}_0 &= () \\
\mathcal{V}_1 &= (1) \\
\mathcal{V}_2 &= \begin{pmatrix} 2 & 1 & 2 \\ 1 & 3 & 3 \end{pmatrix} \\
\mathcal{V}_3 &= \begin{pmatrix} 4 & 4 & 4 & 2 & 1 & 2 & 2 & 1 & 2 & 4 & 4 & 1 & 4 \\ 2 & 1 & 2 & 5 & 5 & 5 & 1 & 3 & 3 & 5 & 1 & 5 & 5 \\ 1 & 3 & 3 & 1 & 3 & 3 & 6 & 6 & 6 & 1 & 6 & 6 & 6 \end{pmatrix}
\end{aligned}$$

and so on.

DEFINITION 4.7: Let $A \subseteq [N]^{(k)}$ and $1 \leq i \leq k$. We define $A - i$ to be the projection of A onto the $k-1$ co-ordinates other than i where we cancel repeated tuples in pairs. That is

$$\begin{aligned}
A - i \;&=\; \{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k) \in [N]^{(k-1)} \mid \\
&\qquad \#\{y \in A \;:\; \forall j \neq i.\ y_j = x_j\} \text{ is odd}\}
\end{aligned}$$

By the definition, if A is the disjoint union of sets $A_1, \ldots, A_r$ then $A - i = \bigoplus_{j=1}^{r}(A_j - i)$.

The following is the key property of the sets $\mathcal{V}_k$.

Lemma 15: For $k \geq 1$ and any $i \in [1,k]$, $\mathcal{V}_k - i = \mathcal{V}_{k-1}$.

Proof: The proof is by induction on k. For the base case, $\mathcal{V}_1 = (1)$ so $\mathcal{V}_1 - 1$ is $\{()\}$ which equals $\mathcal{V}_0$.

Now suppose that $\mathcal{V}_l - i = \mathcal{V}_{l-1}$ for all $1 \leq l < k$ and $i \in [1,l]$. Consider $\mathcal{V}_k - i$ where $i \in [1,k]$. It is clear that the union in the definition of $\mathcal{V}_k$ is a disjoint union so

$$\mathcal{V}_k - i = \bigoplus_{I \subset [1,k]} [(v_k \otimes_I \mathcal{V}_{|I|}) - i] \tag{1}$$

Claim: Suppose that $i \notin I$ and $I \cup \{i\} \subset [1,k]$. Then $(v_k \otimes_I \mathcal{V}_{|I|}) - i = (v_k \otimes_{I \cup \{i\}} \mathcal{V}_{|I|+1}) - i$

Before proving the claim we first see that it is sufficient to complete the induction. Consider the natural pairing between the subsets $I \subseteq [1,k]$ that do not contain i and those subsets that do contain i, namely I is paired with $I \cup \{i\}$. Equation (1) has terms for both elements of every pair except for the pair with $I = [1,k] - \{i\}$ since there is no term for $I = [1,k]$. By the claim, the contributions to $\mathcal{V}_k - i$ from the elements of any of these pairs cancel each other out so we have $\mathcal{V}_k - i = (v_k \otimes_{[1,k]-\{i\}} \mathcal{V}_{k-1}) - i = \mathcal{V}_{k-1}$ which is what we needed to show.

Now to prove the claim, define v_k^i to be v_k with its i-th component removed. Also, for $i \notin I$, define

$$I|_i = \{j \mid j \in I, j < i\} \cup \{j-1 \mid j \in I, j > i\}.$$

Since $i \notin I$, by the definition of $\bigotimes_I$ we have $(v_k \bigotimes_I \mathcal{V}_{|I|}) - i = v_k^i \bigotimes_{I|_i} \mathcal{V}_{|I|}$ because all tuples in $v_k \bigotimes_I \mathcal{V}_{|I|}$ have the same i-th component, namely the i-th component of v_k. On the other hand, by the definition of $\bigotimes_{I\cup\{i\}}$ we have $(v_k \bigotimes_{I\cup\{i\}} \mathcal{V}_{|I|+1}) - i = v_k^i \bigotimes_{I|_i} (\mathcal{V}_{|I|+1} - j)$ where i is the j-th element of $I \cup \{i\}$. This follows because we are first inserting the j-th component of each tuple in $\mathcal{V}_{|I|+1}$ into the i-th component of our new tuples (ignoring the i-th component of v_k) and then removing that i-th component. (All duplicates created in this process must be from tuples in $\mathcal{V}_{|I|+1}$ that disagree on the j-th component but agree everywhere else.)

Since $i \notin I$ and $I \cup \{i\} \subset [1,k]$, we have $|I| + 1 < k$. Therefore, by the inductive hypothesis, $\mathcal{V}_{|I|+1} - j = \mathcal{V}_{|I|}$ and thus

$$\begin{aligned}
(v_k \textstyle\bigotimes_{I\cup\{i\}} \mathcal{V}_{|I|+1}) - i &= v_k^i \textstyle\bigotimes_{I|_i} (\mathcal{V}_{|I|+1} - j) \\
&= v_k^i \textstyle\bigotimes_{I|_i} \mathcal{V}_{|I|} \\
&= (v_k \textstyle\bigotimes_I \mathcal{V}_{|I|}) - i
\end{aligned}$$

which proves the claim. $\square$

Lemma 16: Assume that $N \geq \binom{d+2}{2}$. For every $S \subset [N+s]$ with $|S| \leq d+1$, define $\mathcal{M}_S = M(S, \mathcal{V}_{|S|})$. Then $\mathcal{M} = \cup_S \mathcal{M}_S$ is a $(d+1)$-design for (1)–(3).

Proof: We first observe that for any k, $\mathcal{V}_k$ contains entries from $[1, \binom{k+1}{2}]$ so $N \geq \binom{d+2}{2}$ implies that $\mathcal{V}_k$ is well defined for $k \leq d+1$.

For condition (a) of the definition of a $(d+1)$-design for (1)–(3), observe that $\mathcal{M}_\emptyset = M(\emptyset, \mathcal{V}_0) = M(\emptyset, \{()\}) = \{\emptyset\}$, where $\emptyset$ is the empty matching and so $\emptyset \in \mathcal{M}$.

Let $S \subset [N+s]$, $|S| \leq d+1$ and $i \in S$. Write $S = \{i_1, \ldots, i_k\}$ for $k \leq d+1$, where $i_i < i_2 < \cdots < i_k$ and suppose that $i = i_j$. Interpreting the definitions and applying Lemma 15 we have,

$$\begin{aligned}
\mathcal{M}_S - i &= M(S, \mathcal{V}_k) - i = M(S - \{i\}, \mathcal{V}_k - j) \\
&= M(S - \{i\}, \mathcal{V}_{k-1}) = \mathcal{M}_{S-\{i\}}
\end{aligned}$$

where the second equality follows because both the definitions $\mathcal{M} - i$ and $\mathcal{V} - j$ use the same $\bigoplus$ operator. Thus condition (b) of the definition of a $(d+1)$-design holds and the lemma follows. $\square$

This proves Theorem 14.

5 Search vs decision

We now show that our focus on search problems as opposed to decision problems is necessary. We say that two problems are *computationally equivalent* if each is reducible to the other. It is well-known that the problem *SAT-SEARCH* (find a satisfying assignment to a set of clauses, if one exists) is computationally equivalent to the decision problem *SAT* (determine whether a given set of clauses has a satisfying assignment).

Although a *total* search problem does not always have an obvious decision problem equivalent to it, nevertheless every *single-valued* total NP search problem is computationally equivalent to the decision problem "is the i-th bit of the unique answer equal to one?". An interesting example comes from the Fellows and Koblitz paper [FK92], which shows how to provide every prime number with a unique certificate that can be used to verify in polynomial-time that the number is prime. (The certificates provided by Pratt [Pra75] are not unique.) The single-valued NP search problem coming from Fellows and Koblitz is: Given a number m, list its prime divisors in order, together with their unique certificates.

The theorem below shows that none of the type 2 search problems introduced in Section 2 is computationally equivalent to any decision problem. In fact, from the proof, one can see this will be true for basically any non-trivial problem in TFNP^2. It follows that the same will be true relative to a generic oracle for any complete problem for the corresponding search classes.

Theorem 17: None of the problems *SOURCE.OR.SINK*, *SINK*, *LEAF*, or *PIGEON* is polynomial-time Turing equivalent to any decision problem.

Proof: Define NP^2 and coNP^2 to be the type 2 analogs of NP and coNP (in the same way that FNP^2 is the type 2 analog of FNP.) It is easy to see that if a decision problem D is polynomial-time Turing reducible to some Q in TFNP^2 then one can guess and verify answers to the oracle queries to Q made by the reducing machine, so D is in $\mathsf{NP}^2 \cap \mathsf{coNP}^2$. Therefore, to show that a problem in TFNP^2 is not equivalent to any decision problem, it suffices to show that it is not reducible to a problem in $\mathsf{NP}^2 \cap \mathsf{coNP}^2$.

Lemma 18: None of the problems *SOURCE.OR.SINK*, *SINK*, *LEAF*, or *PIGEON* is polynomial-time Turing reducible to any decision problem in $\mathsf{NP}^2 \cap \mathsf{coNP}^2$.

Since *SOURCE.OR.SINK* reduces to all of the other problems mentioned in the statement of the theorem, it suffices to show this for *SOURCE.OR.SINK*. A slightly weaker version of the following proposition is implicit in [HH87], [BI87], [Tar89]; the proposition as stated is implicit in [IN88]:

Proposition 19: $\mathsf{NP}^2 \cap \mathsf{coNP}^2 \subseteq (\mathsf{P}^2)^{\mathsf{TFNP}}$

Thus, if *SOURCE.OR.SINK* were reducible to a problem in $\mathsf{NP}^2 \cap \mathsf{coNP}^2$, it would be in $(\mathsf{FP}^2)^A$ for some type 1 oracle A (moreover, A could be a search problem in TFNP, but this is not important for the argument.) Thus, there would be a polynomial time oracle machine which asks queries to A and to the underlying directed graph and which returns a source or a sink other than 0 of the directed graph. This machine would yield a decision tree making predecessor/successor queries of depth poly-logarithmic in the number of nodes in the directed graph, which finds a source or a sink of the graph. For sufficiently large sizes of n, the number of queries asked is smaller than $2^{n-1} - 2$, and each query fixes the predecessor or successor of at most 2 nodes. Thus, any consistent path in this tree leaves at least 3 nodes whose predecessors and successors are not yet fixed. If the path produces a node c as output, two of these three nodes (removing c if c is one of the three) can be used to consistently define the value of c's predecessor and/or successor, if they have not been fixed by the path. Thus, there is a graph consistent with p where c is neither a source nor a sink, a contradiction to the assumed correctness of the decision tree. □

The above argument holds for any problem in TFNP^2 that does not have a poly-logarithmic depth decision tree that solves it. The above outline was used in the proof of [IN88] (Proposition

4.2) which shows that for a generic oracle G, TFNP^G is not contained in FP^G. There, the problem in TFNP^2 without poly-log depth decision trees was to find either a logarithmic-size clique or anti-clique in an undirected graph given as the type 2 input, the existence of a solution being guaranteed by Ramsey's Theorem.

Acknowledgements

The authors would like to thank Christos Papadimitriou for sharing his insights on these problems and for a number of discussions that led to this work, and Steven Rudich for helpful discussions.

References

[BI87] Manuel Blum and Russell Impagliazzo. Generic oracles and oracle classes. In *28th Annual Symposium on Foundations of Computer Science*, pages 118–126, Los Angeles, CA, October 1987. IEEE.

[BIK+94] Paul W. Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, and Pavel Pudlák. Lower bounds on Hilbert's Nullstellensatz and propositional proofs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 794–806, Santa Fe, NM, November 1994. IEEE.

[CIY] S. A. Cook, R. Impagliazzo, and T. Yamakami. A tight relationship between generic oracles and type-2 complexity theory. Manuscript.

[FK92] M. Fellows and N Koblitz. Self-witnessing polynomial-time complexity and prime factorization. In *Proceedings, Structure in Complexity Theory, Seventh Annual Conference*, pages 107–110, Boston, M A, June 1992. IEEE.

[HH87] Juris Hartmanis and Lane A. Hemachandra. One-way functions, robustness, and non-isomorphism of *NP*-complete sets. In *Proceedings, Structure in Complexity Theory, Second Annual Conference*, pages 160–174, Cornell University, Ithaca, NY, June 1987. IEEE.

[HH91] J. Hartmanis and L. Hemachandra. One-way functions, robustness, and non-isomorphism of NP-complete sets. *Theoretical Computer Science*, 81:155–163, 1991.

[IN88] R. Impagliazzo and M. Naor. Decision trees and downward closures. In *Proceedings, Structure in Complexity Theory, Third Annual Conference*, pages 29–38, Washingto n, D.C., June 1988. IEEE.

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988.

[Pap90] Christos H. Papadimitriou. On graph-theoretic lemmata and complexity classes. In *Proceedings 31st Annual Symposium on Foundations of Computer Science*, pages 794–801, St. Louis, MO, October 1990.

[Pap91] Christos H. Papadimitriou. On inefficient proofs of existence and complexity classes. In *Proceedings of the 4th Czechoslovakian Symposium on Combinatorics*, 1991.

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, pages 498–532, 1994.

[Pra75] Vaughan R. Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4:214–220, 1975.

[PSY90] Christos H. Papadimitriou, Alejandro A. Schäffer, and Mihalis Yannakakis. On the complexity of local search. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, pages 438–445, Baltimore, MD, May 1990. (Extended Abstract).

[Rii93] Søren Riis. *Independence in Bounded Arithmetic*. PhD thesis, Oxford University, 1993.

[Tar89] G. Tardos. Query complexity, or why is it difficult to separate $NP^A \bigcap coNP^A$ by a random oracle A? *Combinatorica*, 9:385–392, 1989.

[Tow90] M. Townsend. Complexity of type-2 relations. *Notre Dame J. Formal Logic*, 31:241–262, 1990.

Appendix: A Weak Separation

Theorem 6, showing that *SINK* is not reducible to *LONELY* (equivalently, not to *LEAF*) has a difficult proof involving the Nullstellensatz degree bound. Here we present a simpler proof, based on a probabilistic argument, of a weaker result which applies only to "strong" reductions.

We say that problem Q_1 is *strongly reducible* to problem Q_2 if there exist type 2 polynomial-time computable functions F and G such that $Q_2(G[\alpha, x], F(\alpha, x)) \subseteq Q_1(\alpha, x)$, for all α and x, where $G[\alpha, x] = \lambda z.G(\alpha, x, z)$. This is the same as the definition of many-one reducibility given in section 2.2, with the restriction that the function H, which maps solutions for Q_2 to solutions for Q_1, is required to be trivial (i.e. $H(\alpha, x, y) \equiv y$). It turns out that all of the many-one reductions given in sections 2.4 and 2.5 are in fact strong reductions. The proof techniques below could easily be strengthened to apply to the case of many-one reductions in which H satisfies the restriction that for all α, x, and z, at most polynomially many (in $|x|$) different strings y satisfy $H(\alpha, x, y) = z$.

Theorem 20: *SINK* is not strongly reducible to *LEAF*.

Proof: Suppose to the contrary that *SINK* is strongly reducible to *LEAF* using functions F and G'. We proceed as in the proof of Theorem 3, except now the reducing machine M has a very simple form. It takes (α, x) coding a directed graph GD as input, makes a single query to $LEAF^*$, and the query answer must be a sink in GD. (The last is because we only consider input graphs GD in which the nonstandard node 0...0 is a source.) The query is made to $LEAF^* = LEAF(\beta, z)$, where (β, z) describes (using functions F and G') an undirected graph G of maximum degree 2. Since only a single query is made, we can ignore z and assume that β is computed by a polynomial time machine M_β with inputs α, x, and c, where c is a node in G. As before, we fix a long string x, and represent the computation of M_β on input c by a decision tree $T(c)$ whose nodes query the input graph GD. The outcome of each query of node u is the ordered pair $< v, w >$, indicating that there is an edge in GD from v to u and one from u to w. Here either v or w can be empty, in case u is a source or sink. Each leaf in the tree $T(c)$ is labelled with the information coded by the output of M_β, namely an unordered pair $\{c', c''\}$ of configurations indicating that c' and c'' are the neighbors of c in G. Once again, either or both of c' and c'' can be empty.

We denote the standard source 0...0 in GD by 0.

The decision trees represent a transformation from a directed input graph GD_i to an undirected output graph G_i. Our task is to construct GD_0 such that the nonstandard leaves of the corresponding graph G_0 do not form a subset of the sinks of GD_0. We will construct GD_0 to consist of two chains; one from 0 to a sink s and one from a nonstandard source to a sink t, such that (assuming the decision trees behave properly for certain other inputs GD_i) the resulting graph G_0 has both s and t as leaves. It follows that G_0 must have a third leaf (other than the standard leaf). Since GD_0 has only the two sinks s and t, we will have achieved our goal.

We begin the construction of GD_0 by giving a random process for constructing an input graph GD_1 consisting of a single chain from 0 to a random node s (see Figure 5). We observe that if the corresponding undirected graph G_1 is correct, it must have s as a leaf (since s is the only sink in GD_1). Therefore the path determined by GD_1 in the tree $T(s)$ must lead to a leaf labelled $\{u, \epsilon\}$ for some node u in G_1. Next we construct an oracle GD_2 consisting of a rearrangement of GD_1 to form a chain from 0 to a new random node t. As before, the path determined by GD_2 in the tree $T(t)$ must lead to a leaf with a label of the form $\{v, \epsilon\}$, indicating that t is a leaf in G_2. Finally we construct GD_0 by referring to both GD_1 and GD_2 so that GD_0 has as sinks precisely the two nodes s and t, and further both of the trees $T(s)$ and $T(t)$ follow the same paths under GD_0 as they did before respectively under GD_1 and GD_2. Hence both s and t are leaves in G_0, as required to achieve our goal.

Recall that the nodes of GD_i consist of all nonempty strings of length $n = |x|$ or less. We now describe the random process for constructing GD_1 and GD_2.

(1) Pick five random distinct nodes r, r', s, t, t', all distinct from the standard node 0, and let GD_1 consist of a random chain (uniformly distributed) from 0 to s, subject to the constraints that r' is the successor of r, t' is the successor of t, and r' precedes t. That is GD_1 has the form: $\langle w_{0,r}, w_{r',t}, w_{t',s}\rangle$, where $w_{i,j}$ is a chain of nodes beginning with i and ending with j.

(2) Let GD_2 consist of GD_1 with the second and third segments transposed, as shown in Figure 5, so that GD_2 is a chain from 0 to t. That is, GD_2 has the form: $\langle w_{0,r}, w_{t',s}, w_{r',t}\rangle$.

(3) If the path determined by GD_1 in the tree $T(s)$ queries any of the nodes r, r', t, t', then FAIL.

(4) If the path determined by GD_2 in the tree $T(t)$ queries any of the nodes r, r',s, t', then FAIL.

(5) Let GD_0 consist of segments of GD_1 rearranged into two chains, with sinks s and t respectively, where the first chain is $\langle w_{0,r}, w_{t',s}\rangle$, and the second chain is $\langle w_{r',t}\rangle$. See Figure 5.

Note that if the process survives steps (3) and (4), then indeed both of the trees $T(s)$ and $T(t)$ follow the same paths under GD_0 as they did before respectively under GD_1 and GD_2, as required above. Hence we are done if we can argue that the probability of failure in steps (3) and (4) is small.

To argue the case for (3), note that an equivalent process modelling (3) would be to choose s at random and a random chain from 0 to s. This determines a path p in $T(s)$. Now choose r and t

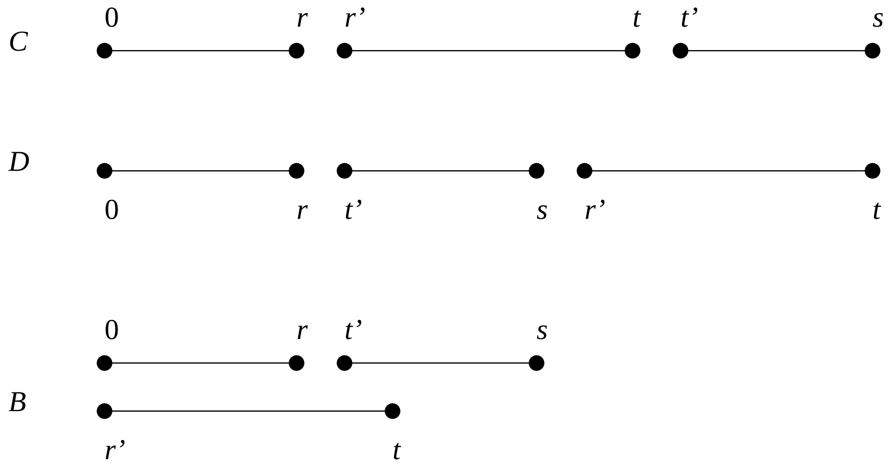

Figure 5: Oracle graphs GD_1, GD_2, and GD_0

at random, let r' and t' be their successors, and FAIL if the path p queries any of these four nodes. Since p queries only a tiny fraction of the 2^n possible nodes, the probability of failure is tiny.

The probability of failure in (4) is exactly the same as in (3). This is because there is an obvious one-one correspondence (namely transpose segments) between chains and nodes generated according to the process for GD_1 and chains and nodes generated according to the process for GD_2. The process preserves the failure set. $\square$