A separation of PLS from PPP

Ilario Bonacina*       Neil Thapen†

June 16, 2022

Abstract

Recently it was shown that PLS $\not\subseteq$ PPADS [GHJ†22]. We show that this separation already implies that PLS $\not\subseteq$ PPP. These separations are shown for the decision tree model of TFNP and imply similar separations in the type-2, relativized model.

Important note. This manuscript is based on an early preprint of [GHJ†22] published on ECCC, which did not contain the result that PLS $\not\subseteq$ PPP. Our work is superseded by the revised version of [GHJ†22] of 10 June 2022 which, independently, contains this result. The manuscript has not been edited to reflect this, and in particular references to [GHJ†22] are to the early version. It can be seen as an alternative, slightly more direct, presentation of the result.

1 Introduction

Total NP search problems (TFNP) are total relations of polynomial growth rate, with polynomial-time graphs. They were introduced in [MP91] and since then several classes of TFNP problems have been studied. Such classes in most cases correspond to combinatorial problems. In particular, the class PLS is the class of TFNP relations corresponding to finding an input of a Boolean circuit providing a locally minimal output [JY88]. The class PPP consists of the relations total by virtue of the pigeonhole principle, and the class PPADS consists of the relations whose totality relies on the combinatorial principle saying that a directed graph with a source must have a sink [Pap94].

The different TFNP classes cannot be separated without solving the P = NP problem, but separations are known with respect to oracles. Oracle separations are one of the motivations for Beame et al [BCE+98] to introduce the relativized, type-2 model of TFNP. Separations of classical TFNP classes in this model were shown for instance in [BCE+98, BOM04] and very recently in [GHJ †22].

*Universitat Politecnica de Catalunya bonacina@cs.upc.edu. Partially supported by the grants PID2019-109127GB-C22 and IJC2018-055334-I funded by MCIN/AEI/10.13039/50110011033
†Institute of Mathematics of the Czech Academy of Sciences, thapen@math.cas.cz. Partially supported by GA ČR project 19-05497S and by RVO:67985840.
tend to work with a decision tree model of TFNP and to derive separations from proof complexity lower bounds. We denote TFNP classes in the decision tree model with a superscript $dt$. Separations in the decision tree model immediately imply similar separation between the corresponding classes in the type-2 model of TFNP in which separations are usually stated, which is essentially the uniform, machine model of which decision tree TFNP is the nonuniform version.

We show that the non-reducibility of PLS to PPP, the remaining open relation between these classical classes, follows from the non-reducibility of PLS to PPADS shown in [GHJ+22] together with a standard construction on PPP given by trees, which was used in [BCE+98] to show that PPA is not reducible to PPP, and in [BOM04] to show that PLS is not reducible to PPP by a limited kind of reduction. The separation from [GHJ+22] is the following.

**Proposition 1.1** ([GHJ+22, Corollary 1]). $\text{PLS}^{dt} \not\subseteq \text{PPADS}^{dt}$.

This paper should be read as an extension of [GHJ+22]. However it is also intended to be somewhat self-contained, so we will repeat some definitions and only really use a proof complexity lower bound from [GHJ+22]. Our main result is the following separation.

**Theorem 1.2.** $\text{PLS}^{dt} \not\subseteq \text{PPP}^{dt}$.

**Organization of the paper.** In Section 2 we provide the necessary definitions of the TFNP classes we use, and of the notion of reduction between search problems. Section 3 contains the main technical contribution of this paper and how it implies $\text{PLS}^{dt} \not\subseteq \text{PPP}^{dt}$. In Section 4 we give a more self-contained argument for the separation proven in Section 3, introducing the Sherali-Adams proof system and showing how the Sherali-Adams lower bound in [GHJ+22] implies $\text{PLS}^{dt} \not\subseteq \text{PPP}^{dt}$. Appendix A contains some folklore reductions between total search problems.

**Notation.** In general we try to use similar notation to [GHJ+22]. For $n \in \mathbb{N}$ we let $[n] := \{1, 2, \ldots, n\}$. When writing $\log n$ we always mean $\lceil \log n \rceil$. We write polylog($n$) for the set of functions $f : \mathbb{N} \to \mathbb{N}$ of polylogarithmic growth rate in $n$, that is, such that $f(n) \in O(\log^c n)$ for some $c \in \mathbb{N}$. Similarly we write $\text{qpoly}(n)$ for the functions of quasipolynomial growth rate in $n$, that is, such that $\log f(n) \in \text{polylog}(n)$. We will sometimes identify a tree $T$ with its set of branches, and will write $T[h]$ for the set of branches in $T$ labelled with $h$, and $T[\neg h]$ for the set of branches not labelled with $h$.

2 The decision tree model of TFNP

2.1 Search problems

Most of the definitions in this section are taken from, or modelled after, those in [GHJ+22], with some minor changes to the notation.
Definition 2.1. A total search problem is a sequence $R = (R_n)_{n \in \mathbb{N}}$ of relations $R_n \subseteq \{0,1\}^{m_n} \times O_n$, where $m_n \in \mathbb{N}$, each $O_n$ is a finite set, and for all $x \in \{0,1\}^{m_n}$ there is some $y \in O_n$ such that $(x,y) \in R_n$.

We will also call a single relation $R_n$ in such a sequence a total search problem. In this context the string $x$ is an instance of the problem and any $y$ with $(x,y) \in R_n$ is a solution of the instance. Typically $x$ codes some combinatorial structure for which $n$ is a natural size parameter, such as a function $[n] \to [n]$.

Definition 2.2 (TFNP$^{dt}$). A total search problem is in TFNP$^{dt}$ if $m_n$ is at most quasipolynomial in $n$ and, for each $y \in O_n$, there is a decision tree $T_y$ of depth polylog($n$), querying $x$ and deciding whether $(x,y) \in R_n$.

We introduce the five total search problems we will use in this note. The first is PIGEON, which defines the class PPP$^{dt}$ (see page 5 for what this means).

PIGEON$_n$. An instance $x \in \{0,1\}^{(n+1)\log n}$ encodes a function $F : [n+1] \to [n]$. A solution is a pair $(i,j) \in [n+1] \times [n+1]$ with $i \neq j$ and $F(i) = F(j)$.

The next two problems Sink-of-Line$_n$ and LeftPigeon are equivalent to each other, as we will show, and both define the class PPADS$^{dt}$ (see page 5).

Sink-of-Line$_n$. An instance $x \in \{0,1\}^{2n\log n}$ encodes functions $S : [n] \to [n]$ and $P : [n] \to [n]$, together describing a directed graph on $[n]$ which has an edge $(u,v)$ if $S(u) = v$ and $P(v) = u$. The out-degree of $u$ is 1 if $P(S(u)) = u$ and otherwise 0. The in-degree of $v$ is 1 if $S(P(v)) = v$ and otherwise 0. The solutions are

- 1 if it is not a source, that is, if it has out-degree 0 or in-degree 1
- Any $u \in [2,n]$ if it is a sink, that is, has in-degree 1 and out degree 0.

LeftPigeon$_n$. An instance $x \in \{0,1\}^{(2n+1)\log(n+1)}$ encodes two functions $F : [n+1] \to [n]$ and $G : [n] \to [n+1]$. A solution is any $u \in [n+1]$ with $G(F(u)) \neq u$.

LeftPigeon was introduced in [BJ12]. The reason for the name is that $G$ is a kind of left-inverse of $F$. It is total because if $F(u) = F(u')$ for distinct $u, u'$, then either $u$ or $u'$ is a solution to LeftPigeon$_n$.

The last two problems Sink-of-DAG and Iter are equivalent to each other, as we will show, and both define the class PLS$^{dt}$ (see page 5).

Sink-of-DAG$_n$. An instance $x \in \{0,1\}^{n^2\log(n+1)}$ encodes a function $S : [n] \times [n] \to ([n] \cup \{null\})$, which describes a directed graph on a grid with edges from $(i,j)$ to $(i+1, S(i,j))$ (unless $S(i,j) = \text{null}$). A solution is a pair in $[n] \times [n]$ of one of the forms

- $(1,1)$ if $S(1,1) = \text{null}$
- $(n,j)$ if $S(n,j) \neq \text{null}$

3
• \((i, j)\) with \(i < n\) if \(S(i, j) \neq \text{null}\) and \(S(i + 1, S(i, j)) = \text{null}\).

**Iter**. An instance \(x \in \{0, 1\}^{n \log n}\) encodes a function \(S : [n] \to [n]\) such that \(S(u) \geq u\) for every \(u \in [n]\). A solution is

- 0 if \(S(1) = 1\)
- \(u \in [n]\) if \(S(u) > u\) and \(S(S(u)) = S(u)\).

**Iter** was used to characterize PLS in [BOM04]. It was introduced as the *iteration principle* in [BR94], in the context of bounded arithmetic.

### 2.2 Reductions

**Definition 2.3.** Let \(R_m \subseteq \{0, 1\}^r \times \mathcal{O}_R\) and \(S_n \subseteq \{0, 1\}^s \times \mathcal{O}_S\) be two total search problems. We say that \(R_m\) is reducible to \(S_n\) if there exists a pair of functions \(f : \{0, 1\}^r \to \{0, 1\}^s\) and \(g : \{0, 1\}^r \times \mathcal{O}_S \to \mathcal{O}_R\) satisfying
\[
(f(x), y') \in S_n \iff (x, g(x, y')) \in R_m
\]
for all \(x \in \{0, 1\}^r\) and all \(y' \in \mathcal{O}_S\). The reduction has depth \(d\) if

- Each bit of \(f(x)\) is computable by a depth-\(d\) decision tree
- For every \(y' \in \mathcal{O}_S\), the function \(x \mapsto g(x, y')\) is computable by a depth-\(d\) decision tree.

If \(R_m\) is reducible to \(S_n\) by a depth-\(d\) reduction we write \(R_m \leq_d S_n\).

If we omit the subscript and write simply e.g. \(R_n^2 \leq S_n\), this means that the depth of the reduction is polylogarithmic in \(n\), or in some other size parameter which will be clear from the context.

**Proposition 2.4.** If \(R_m \leq_d S_n\) and \(S_n \leq_e T_l\), then \(R_n \leq_{d(e+1)} T_l\). \(\square\)

**Definition 2.5.** Let \(R\) and \(S\) be total search problems in \(\text{TFNP}^{dt}\). We say \(R\) is reducible to \(S\) if for each \(n\) there is \(m \in q\text{poly}(n)\) such that \(R_n\) is reducible to \(S_m\) by a polylog\((n)\)-depth reduction. We write this as \(R \leq S\). If also \(S \leq R\) we say \(R\) and \(S\) are equivalent.

Note that \(R \leq S\) is equivalent to the condition \(S^{dt}(R) = \text{polylog}(n)\) of [GHJ+22], which expresses that for all \(n\) there are reductions \(R_n \leq_d S_m\) for some \(m, d\) satisfying \(m + d \leq \text{polylog}(n)\).

The next proposition recalls some reductions between search problems. The proofs of these are essentially folklore and are postponed to Appendix A.

**Proposition 2.6.** LEFTPigeon is equivalent to Sink-of-Line. Iter is equivalent to Sink-of-DAG. In particular, Sink-of-DAG \(\leq\) Iter\(n^2\).

We can now define the three subclasses of \(\text{TFNP}^{dt}\) that we focus on.
• \( \text{PPP}^{dt} := \{ R \in \text{TFNP}^{dt} : R \leq \text{PIGEON} \} \)
• \( \text{PPADS}^{dt} := \{ R \in \text{TFNP}^{dt} : R \leq \text{Sink-of-Line} \} \)
• \( \text{PLS}^{dt} := \{ R \in \text{TFNP}^{dt} : R \leq \text{Sink-of-DAG} \} \)

We say that \( \text{PIGEON} \) is complete for \( \text{PPP}^{dt} \), and so on. By Propositions 2.6 and 2.4, these classes do not change if we replace \( \text{Sink-of-Line} \) with \( \text{LEFTPIGEON} \) or replace \( \text{Sink-of-DAG} \) with \( \text{ITER} \).

3 From PIGEON to LEFTPIGEON

The main technical contribution of this paper is the following lemma.

**Lemma 3.1.** If \( \text{ITER}_m \leq_d \text{PIGEON}_n \) then \( \text{ITER}_m \leq_e \text{LEFTPIGEON}_n \), where \( e \in \text{poly}(d, \log m, \log n) \).

Before proving the lemma we show how it implies our main result.

**Lemma 3.2.** If \( \text{PLS}^{dt} \subseteq \text{PPP}^{dt} \) then \( \text{PLS}^{dt} \subseteq \text{PPADS}^{dt} \).

**Proof.** Suppose that \( \text{PLS}^{dt} \subseteq \text{PPP}^{dt} \), that is, for all \( n \) there exist \( m \in q\text{poly}(n) \) with the property that \( \text{ITER}_n \leq \text{PIGEON}_m \). By Lemma 3.1 it follows that \( \text{ITER}_n \leq \text{LEFTPIGEON}_m \). Since \( \text{ITER} \) is complete for \( \text{PLS}^{dt} \) and \( \text{LEFTPIGEON} \) is complete for \( \text{PPADS}^{dt} \) this means that \( \text{PLS}^{dt} \subseteq \text{PPADS}^{dt} \).

An immediate corollary of this and Proposition 1.1 is that \( \text{PLS}^{dt} \not\subseteq \text{PPP}^{dt} \). However we will give a more self-contained proof of this separation in the next section, deriving it from a proof-complexity lower bound in [GHJ+22].

The proof of Lemma 3.1 takes up the rest of this section. Fix \( n, m, d \). We will need to talk a lot about partial instances of \( \text{ITER}_m \) of a certain kind, so we introduce a special word for them:

**Definition 3.3.** A structure \( \alpha \) is a partial function \( S : [m] \to [m] \) satisfying \( S(v) \geq v \) wherever \( S(v) \) is defined. If \( S(v) = v \) we say there is a loop at \( v \). If \( S(v) > v \) we say there is a jump at \( v \). The set of solutions of \( \alpha \) is defined by:

• 0 is a solution, if there is a loop at 1
• \( v \in [m] \) is a solution if it is a jump leading to a loop, that is, if \( S(v) > v \) and \( S(S(v)) = S(v) \).

A total structure is then the same as an instance of \( \text{ITER}_m \), modulo issues of coding. We could have used simply \( S \) as the name for a structure, rather than distinguishing \( S \) and \( \alpha \), but we find it useful to have the symbol \( S \) as a common language when talking about different structures.

Given two structures \( \alpha \) and \( \beta \), we say \( \alpha \) extends \( \beta \) if \( \beta \subseteq \alpha \). We say that \( \alpha, \beta \) are consistent if they agree on the value of \( S(v) \) wherever they both define it. Note that the presence or absence of a solution does not appear in the definition of consistency.
Suppose \((f, g)\) is the depth-\(d\) reduction of \(\text{Iter}_m\) to \(\text{Pigeon}_n\). That is, for any instance \(x \in \{0, 1\}^m \log(m)\) of \(\text{Iter}_m\), the string \(f(x)\) is an instance of \(\text{Pigeon}_n\) and defines a map \(F : [n + 1] \to [n]\), such that from any collision in \(F\) we can use \(g\) to recover a solution to our instance \(x\) of \(\text{Iter}_m\).

For each pigeon \(p \in [n + 1]\) the value of \(F(p)\) is coded by at most \(\log n\) bits of \(f(x)\), and each bit of \(f(x)\) can be computed by a depth-\(d\) tree querying the bits of \(x\). So from \(\log n\) of these we can construct a single tree \(T_p\) which computes \(F(p)\). However, we construct \(T_p\) so that it does not query individual bits of \(x\), but rather makes queries of the form “\(S(v) = ?\)” for \(v \in [m]\), and is guaranteed an answer \(u \in [m]\) with \(u \geq v\). This tree has depth at most \(d \log n\), that is, along every branch at most \(d \log n\) queries of this form are made. Each branch is labelled with a hole \(h \in [n]\), and each branch can be identified with the structure given by the queries and replies along it.

**Definition 3.4 (i-safe).** For \(i \geq 0\), a structure \(\alpha\) is \(i\)-safe if

1. \(\alpha\) does not contain a solution, and
2. \(S(w)\) is undefined in \(\alpha\) at at least \(i\) many points \(w > v\), where \(v\) is rightmost end of a jump. That is, \(v\) is the maximum \(S(u)\) such that \(u\) is a jump, or \(v\) is 1 if \(\alpha\) contains no jumps.

**Lemma 3.5.** Let \(\alpha\) be a structure. The following are equivalent:

1. \(\alpha\) is \(i\)-safe
2. for any adaptive sequence of queries to \(S\) of length \(i\) or less, there are replies by which we can extend \(\alpha\) such that at the end \(\alpha\) still contains no solution.

*Proof.* To show that 1. implies 2. it is enough to show that, if \(\alpha\) is \(i\)-safe for \(i \geq 1\), then we can extend \(\alpha\) with a reply to any query, such that the result is \((i - 1)\)-safe. Suppose “\(S(u) = ?\)” is queried. If \(u\) is the end of a jump, or \(u = 1\), reply with the smallest \(w > u\) such that \(S(w)\) is not defined; otherwise reply with \(u\), forming a loop at \(u\).

To show that 2. implies 1., consider the sequence of queries that starts at the rightmost end \(v\) of a jump and make \(i\) queries \(S(v), S(S(v)), \ldots, S^{i}(v)\). The replies must be an increasing sequence of \(i\) points in \([m]\). None of these points is a jump in \(\alpha\), since we start at \(v\), and none of them is a loop in \(\alpha\), as otherwise extending \(\alpha\) with these replies would add a solution. Hence we have found \(i\) undefined points above \(v\). \(\square\)

Recall that we use the notation \(T[h]\) to mean the set of branches in \(T\) labelled \(h\), and \(T[\neg h]\) to mean the set of branches not labelled \(h\).

**Lemma 3.6.** Suppose \(\beta \in T_p[h]\) and \(\beta' \in T_{p'}[h]\), for a hole \(h\) and distinct pigeons \(p \neq p'\). Then no \((d + 2)\)-safe structure extends both \(\beta\) and \(\beta'\).
Proof. Suppose for a contradiction that there is a \((d+2)\)-safe structure \(\alpha\) extending both \(\beta\) and \(\beta'\). Then, by the construction of the trees \(T_p\), in any instance \(x\) of \(\text{ITER}_m\) extending \(\alpha\), the pigeons \(p\) and \(p'\) both go to hole \(h\) in the corresponding instance \(f(x)\) of \(\text{PIGEON}_n\). Thus \(g(x, (p, p'))\) is a solution of \(x\), which can be computed by making at most \(d\) queries to bits of \(x\). However since \(\alpha\) is \((d + 2)\)-safe we can construct such an \(x\) in which the output of \(g\) is not a solution, by using Lemma 3.5 to extend \(\alpha\) with \(d\) queries to compute \(g\), then two more queries to check whether the output of \(g\) is a solution (that is, a jump followed by a loop), and finally extending the resulting structure arbitrarily to an instance \(x\) of \(\text{ITER}_m\).

We now extend each tree \(T_p\) to a tree \(T^*_p\) by carrying out the following steps for each branch \(\beta\) in \(T_p\).

1. Let \(v_1 < v_2 < \cdots < v_\ell\) be the elements of the set \(\{S(u) : u\ \text{a jump in } \beta\}\).
2. At the end of \(\beta\), first query \(S(v_i)\) for \(i = 1, \ldots, \ell - 1\).
3. Then, for the largest element \(v_\ell\) query in turn \(S(v_\ell), S(S(v_\ell)), \ldots, S^{d+3}(v_\ell)\).
   If \(\beta\) had no jumps, start these queries at \(1\) instead of \(v_\ell\).
4. If the resulting branch is not \((d + 2)\)-safe, label it with a symbol \text{UNSAFE}.
   Otherwise label it with the same hole \(h\) as \(\beta\) was labelled with.

We call the trees \(T^*_p\) extended pigeon trees. They have depth at most \(2d\log n + d + 3\).

Lemma 3.7. Suppose \(\beta_1^* \in T^*_p[h]\) and \(\beta_2^* \in T^*_p'[h]\), for a hole \(h\) and distinct pigeons \(p \neq p'\). Then \(\beta_1^*\) and \(\beta_2^*\), considered as structures, are inconsistent.

Proof. Let \(\beta_1\) and \(\beta_2\) be the branches in the original trees \(T_{p_1}\) and \(T_{p_2}\) which \(\beta_1^*\) and \(\beta_2^*\) are extensions of. Since \(\beta_1^*\) and \(\beta_2^*\) are labelled with a hole \(h\) rather than with \text{UNSAFE}, all four structures \(\beta_1, \beta_2, \beta_1^*\) and \(\beta_2^*\) are \((d + 2)\)-safe.

Suppose for a contradiction that \(\beta_1^*\) and \(\beta_2^*\) are consistent. Then so are \(\beta_1\) and \(\beta_2\). But, by Lemma 3.6, the structure \(\beta_1 \cup \beta_2\) is not \((d + 2)\)-safe. We now consider two cases, based on which part of Definition 3.4 of \("(d + 2)\)-safe\) fails for \(\beta_1 \cup \beta_2\).

Part 1 fails. There is a solution in \(\beta_1 \cup \beta_2\). The solution cannot be a loop at \(1\), since this would be present in either \(\beta_1\) or \(\beta_2\), which are both \((d + 2)\)-safe. Therefore it is some \(u \in [m]\) such that \(S(u) = v > u\) and \(S(v) = v\) in \(\beta_1 \cup \beta_2\). Neither \(\beta_1\) nor \(\beta_2\) contains both parts of this solution so, without loss of generality, in \(\beta_1\) there is a jump \(S(u) = v\) and \(S\) is undefined at \(v\), while in \(\beta_2\) there is a loop \(S(v) = v\) and \(S\) is undefined at \(u\). Thus in \(\beta_1^*\), by construction, we queried \(S(v)\), since \(v\) is the end of a jump. We cannot have \(S(v) = v\) in \(\beta_1^*\), since this would make \(u\) a solution and \(\beta_1^*\) is \((d + 2)\)-safe. Therefore \(\beta_1^*\) and \(\beta_2\) disagree about \(S(v)\) and are inconsistent.

Part 2 fails. Let \(v\) be the rightmost end of a jump in \(\beta_1 \cup \beta_2\), or 1 if there is no jump. Without loss of generality, the same is true of \(v\) in \(\beta_1\). By the failure
of part 2., the instance $\beta_1 \cup \beta_2$ is undefined at at most $d + 1$ points $w$ in the interval $[v + 1, m]$, and by choice of $v$ is a loop everywhere it is defined in this interval. On the other hand $\beta^*_1$ by construction contains a sequence of $d + 3$ jumps, starting at $v$; we know these are jumps, and never hit a loop, because $\beta^*_1$ contains no solution. Thus there must be some point in the interval, the end of one of the first $d + 2$ jumps, which is a jump in $\beta^*_1$ but is a loop in $\beta_1 \cup \beta_2$. Thus $\beta^*_1$ and $\beta_2$ are inconsistent.

From now on we closely follow the construction in the proof of [BCE+98, Lemma 4]. For each hole $h$ we define a hole tree $U_h$, which is intended to find out which pigeon, if any, goes to $h$.

Let $B_h$ be the set of all branches labelled $h$ from among all extended pigeon trees $T^*_p$. These branches all have length at most $t$ and are pairwise inconsistent, automatically if they are from the same tree and by Lemma 3.7 if they are from different trees. We will define $U_h$ by describing the queries made while processing $B_h$ in several steps, at each step removing some elements and shrinking others. After the $i$th step, $B_h$ will consist of pairwise inconsistent structures, each of size at most $t - i$.

Take the first branch $\beta$ in $B_h$ and query $S(v)$ on every $v$ in its domain. Let $\rho_1$ be the structure formed by the replies. For each branch $\gamma$ in $B_h$, if $\gamma$ is inconsistent with $\rho_1$ we remove $\gamma$ from $B_h$. If $\gamma$ is consistent with $\rho_1$, we let $V$ be the set of points on which $\gamma$ and $\beta$ were both defined. $V$ is non-empty, as $\gamma$ and $\beta$ are inconsistent (or identical). Furthermore $\rho_1$ is defined on all $v \in V$, since it has the same domain as $\beta$. Therefore $\rho_1$ agrees with $\gamma$ on all $v \in V$, since they are consistent. We shrink $\gamma$ by removing all points in $V$ from its domain.

After this first step $B_h$ contains a set of (shrunken) branches, all consistent with $\rho_1$ and all of length at most $t - 1$. Furthermore they are still inconsistent with each other, since we only removed edges consistent with $\rho_1$. Thus we may repeat this step, taking the first nonempty branch $\beta$ in the new $B_h$, querying $S(v)$ for each $v$ in its domain, getting some replies $\rho_2$, and again for every branch remaining in $B_h$ either removing or shrinking it; and so on.

After at most $t$ steps, and thus at most $t^2$ queries, we will have exhausted $B_h$ and built a structure $\rho_1 \rho_2 \ldots \rho_t$ which, for every branch in the original $B_h$, either extends it or it is inconsistent with it. If it is inconsistent with all branches in $B_h$ we output the symbol NONE. If it extends a branch then it extends at most one (since all branches in $B_h$ are inconsistent with each other), and we output the pigeon whose tree that one branch came from. This completes the construction of $U_h$, and implies the following proposition.

**Proposition 3.8.** For a pigeon $p \in [n + 1]$ and a hole $h \in [n]$, suppose $\beta \in T^*_p$ and $\rho \in U_h$ are consistent with each other, and $\beta$ is labelled $h$. Then $\rho$ is labelled $p$.  

We can now define a reduction of $\text{Iter}_m$ to $\text{LeftPigeon}_n$, that is, a pair of functions $(f', g')$, computed suitably by trees, such that

$$(f'(x), y) \in \text{LeftPigeon}_n \rightarrow (x, g'(x, y)) \in \text{Iter}_m.$$
To define the instance $f'(x)$ of $\text{LEFTPIGEON}_n$ we need to specify functions $F : [n + 1] \to [n]$ and $G : [n] \to [n + 1]$.

We define $F(p)$ to be the hole computed by $T^*_p$, or to be an arbitrary value, say 1, if we get a branch in $T^*_p$ labelled UNSAFE. We define $G(h)$ to be the pigeon computed by $U_h$, or to be an arbitrary value, say 1, if we get a branch in $U_h$ labelled NONE. Note that to compute $F(p)$ takes at most $t$ queries to $S$, so to compute a bit of $f'(x)$ that corresponds to a bit of $F(p)$ takes at most $t \log m$ queries to the bits of $x$. The calculation for $G$ is similar.

For the function $g'$ translating solutions to solutions, suppose $p$ is a solution to this instance of $\text{LEFTPIGEON}_n$, that is, $F(p) = h$ for some $h$ such that $G(h) = p'$ with $p' \neq p$. Let $\beta$ and $\rho$ be the branches computed in $T^*_p$ and $U_h$. The branch $\rho$ is labelled with $p'$, or possibly with NONE, but is not labelled with $p$. It follows that $\beta$ cannot be labelled with $h$, or this would contradict Proposition 3.8. Therefore $\beta$ is labelled with UNSAFE and thus is not $(d + 2)$-safe. It follows by Lemma 3.5 that we can find a solution to the instance $x \geq 2$ of $\text{ITER}_m$ by making $d + 2$ more queries to $S$.

This completes the proof of Lemma 3.1.

4 Bounds on Sherali-Adams derivations

4.1 Sherali-Adams

Sherali-Adams is a semi-algebraic proof system in which proofs are polynomial identities of a specific form. It is based on the linear programming hierarchy of Sherali-Adams [SA90] and it can be used to prove the unsatisfiability of a system of polynomial equations and inequalities. The definition of the proof system is modeled upon the definition of the Nullstellensatz proof system introduced in [BIK+94]. In this paper we only need Sherali-Adams to refute systems of equations and hence we give the definition only for this restricted case.

Let $Z = z_1, \ldots, z_m, \bar{z}_1, \ldots, \bar{z}_m$ be formal variables, which we call literals, where the intended meaning of $\bar{z}_i$ is $1 - z_i$. Let $p$ and $p_1, \ldots, p_t$ be polynomials in $\mathbb{Q}[Z]$. A Sherali-Adams derivation, or SA derivation, of $p$ from $p_1, \ldots, p_t$ is a particular way to certify that $p(\alpha) \geq 0$ for every assignment $\alpha \in \{0,1\}^m$ satisfying the equations $p_1 = 0, \ldots, p_t = 0$ (and satisfying $\alpha(\bar{z}_i) = 1 - \alpha(z_i)$ for every $i$). This is done via writing an algebraic identity of the form

\begin{equation}
 p = q + \sum_{j \in [t]} r_j p_j + \sum_{i \in [m]} s_i (z_i^2 - z_i) + \sum_{i \in [m]} s'_i (\bar{z}_i + \bar{z}_i - 1),
\end{equation}

where all polynomials $q, r_i, s_i, s'_i$ are in $\mathbb{Q}[Z]$ and moreover all the coefficients of the polynomial $q$ are non-negative. The polynomials $z_i^2 - z_i, z_i + \bar{z}_i - 1$ are called Boolean axioms\footnote{Notice that we do not need the polynomials $\bar{z}_i^2 - \bar{z}_i$ among the Boolean axioms since $\bar{z}_i^2 - \bar{z}_i = (z_i^2 - z_i) + (z_i + \bar{z}_i - 1)(\bar{z}_i - z_i)$.}. 


Formally, the derivation is the right-hand side of equation (1) considered as a formal sum, and it is a derivation of \( p \) if this sum simplifies to \( p \) after cancellation and rearrangement of terms. The degree of the derivation is the degree of the right-hand side before cancellation, that is, the maximum of \( \deg(q) \), \( \deg(r_j) + \deg(p_j) \), \( \deg(s_i) + 2 \) and \( \deg(s'_i) + 1 \). The size of the derivation is the combined size of all polynomials \( q, r_i, s_i, s'_i \) and \( p_j \), where the size of a polynomial is the number of monomials in it. The magnitude of a proof is the maximum of the magnitudes of these polynomials, where the magnitude of a polynomial is the absolute value of its largest coefficient. Note that this is well-behaved as long as we assume that a polynomial cannot have the same monomial appearing twice (even with different coefficients) and that we do not care here about how, for example, a coefficient is presented as a rational number.

We may also call (1) a derivation of the inequality \( p \geq 0 \), and think of the system of polynomials \( p_i \) we start with as equations \( p_i = 0 \) rather than simply as polynomials. An SA refutation is a derivation of \( -1 \geq 0 \), which is in particular a certificate that this system of equations is unsatisfiable over the assignments \( \alpha \) we consider.

An algebraic identity like the one in (1), after applying the substitution replacing every variable \( \bar{x}_1 \) with \( 1 - x_1 \), is called an SA-derivation without twin variables (here we also no longer need the rightmost sum).

SA can be used as a system for refuting unsatisfiable systems of polynomial equations. It is a small step from this to using it to refute unsatisfiable propositional CNF formulas, via the natural encoding of a clause \( \bigvee_{i \in \ell} z_i \lor \bigvee_{j \in \ell} \neg z_j \) as the polynomial equation \( \prod_{i \in \ell} \bar{z}_i \prod_{j \in \ell} \bar{z}_j = 0 \). As a system for refuting CNFs SA is sound, in that no satisfiable CNF has a refutation, and complete, in that every unsatisfiable CNF has a refutation (but the refutation may be very big).

### 4.2 The left pigeonhole principle

We show that the left pigeonhole principle, or LPHP, a formula associated with LEFTPIGEON, can be refuted efficiently in Sherali-Adams. We will use this in the next section to show essentially that any problem in PPADS\textsuperscript{dc} can be proved total efficiently in Sherali-Adams.

Let \( \bar{T} \) be a sequence \( T_1, \ldots, T_{n+1} \) of binary decision trees, one for each pigeon \( p \in [n+1] \), querying Boolean variables from \( z_1, \ldots, z_m \) and outputting holes in \( [n] \). Let \( \bar{U} \) be a similar sequence \( U_1, \ldots, U_n \) of trees, one for each hole \( h \in [n] \), outputting pigeons in \( [n+1] \). In these trees, outgoing edges from a query to \( z_i \) are labelled with literals \( z_i \) or \( \bar{z}_i \) rather than \( z_i = 1 \) or \( z_i = 0 \). We identify a branch \( b \) in a tree \( T \) with the set of literals occurring along the branch, or with the monomial formed by this set.

We could introduce LPHP\textsubscript{n} as a CNF, but it is more convenient to treat it from the beginning as a set of monomials.

**Definition 4.1.** The formula LPHP\textsubscript{n}(\( \bar{T}, \bar{U} \)) consists of, for each pigeon \( p \) and hole \( h \), for each \( b \in T_p[h] \) and \( c \in U_h[\neg p] \) such that \( b \) and \( c \) are consistent, the monomial \( bc \).
We will call these monomials $bc$ the axioms of LPHP$_n$. Recall that each should be semantically understood as the equation $bc = 0$, expressing that some literal in $b$ or $c$ is 0. The formula expresses that if $F : [n+1] \rightarrow [n]$ and $G : [n] \rightarrow [n+1]$ are the functions specified by $\overline{T}$ and $\overline{U}$, then $G(F(p)) = p$ for all $p \in [n+1]$. To confirm that it expresses what it is supposed to, let $p$ be any pigeon and suppose we have an assignment $x$ to $Z$ in which $p$ goes to $h$, that is, some branch $b$ in $T_p[h]$ evaluates to 1. Then the axioms of LPHP$_n$ guarantee that every branch $c$ in $U_h[¬p]$ evaluates to 0, so the (unique) branch of $U_h$ which evaluates to 1 in $x$ must be labelled $p$. Note that LPHP$_n$ is not symmetrical between pigeons and holes.

A useful fact about any tree $T$ querying Boolean variables is that its branches sum to 1 modulo the Boolean axioms. Precisely, if $T$ has depth $d$, then by induction on $d$ we get that $\sum_{b \in T} b = 1 + q_T$ where $q_T$ is a “remainder” polynomial of the form $\sum_{i \in [m]} s_i(z_i + \overline{z}_i - 1)$, of degree $d$ and of magnitude and size at most $O(2^d)$. In particular, we always have

$$\sum_{b \in T[h]} b + \sum_{b \in T[¬h]} b = 1 + q_T. \quad (2)$$

Another useful fact is that given inconsistent monomials $b$ and $c$, then $bc$ is 0 modulo the Boolean axioms. Precisely, $bc = mz_i\overline{z}_i$ for some monomial $m$ and variable $z_i$ and thus

$$bc = mz_i(z_i + \overline{z}_i - 1) - m(z_i^2 - z_i). \quad (3)$$

**Lemma 4.2.** LPHP$_n(\overline{T}, \overline{U})$ has an SA refutation of degree $2d$ and magnitude and size $\text{poly}(n, 2^d)$, where $d$ is a bound on the depth of all trees in $\overline{T}, \overline{U}$.

**Proof.** Let us introduce the notation $\sum S$ for the sum of a set $S$ of branches. We define degree-$d$ polynomials

$$F(p, h) := \sum T_p[h] \quad \text{and} \quad G(h, p) := \sum U_h[p].$$

Then $F(p, h)$ evaluates to 1 precisely if $F(p) = h$, and $G(h, p)$ evaluates to 1 precisely if $G(h) = p$. Thus the LPHP$_n$ axioms semantically imply that $F(p, h) \leq G(h, p)$. We show that this inequality has a small SA derivation.

Shortening $F(p, h)$ to $F$ and $G(h, p)$ to $G$, we have

$$F - FG = F(1 - \sum U_h[p]) = F(\sum U_h[¬p] - q_{U_h})$$

$$= (\sum T_p[h])(\sum U_h[¬p]) - Fq_{U_h}$$

where we used (2) for the second equality. We similarly have

$$G - FG = G(1 - \sum T_p[h]) = (\sum U_h[p])(\sum T_p[¬h]) - Gq_{T_p}.$$

Thus

$$G - F = (\sum U_h[p])(\sum T_p[¬h]) - (\sum T_p[h])(\sum U_h[¬p]) + Fq_{U_h} - Gq_{T_p}.$$
This has the form of an SA derivation of $G(h, p) - F(p, h) \geq 0$, as follows. The first product expands out to a sum of monomials with positive coefficients. Any term in the second product has the form $bc$ for $b \in T_p[\bar{h}]$ and $c \in U_h[-p]$, where either $b$ and $c$ are consistent, in which case $bc$ is an axiom of LPHP, or they are inconsistent, in which case $bc$ is a combination of Boolean axioms by (3). Finally $F_{q_U}$ and $G_{q_T}$ are combinations of Boolean axioms. The whole expression on the right has degree at most $2d$. Let us call it $J(p, h)$.

Now consider the expression

$$
\sum_{h \in [n]} \sum_{p \in [n+1]} G(h, p) - \sum_{p \in [n+1]} \sum_{h \in [n]} F(p, h).
$$

On the one hand, this is equal to $\sum_{p \in [n+1]} \sum_{h \in [n]} J(p, h)$. On the other hand, $\sum_{p \in [n+1]} G(h, p)$ is now the sum $\sum U_h$ of all branches of $U_h$, and thus equals $1 + q_U$, by (2), and similarly $\sum_{h \in [n]} F(p, h) = 1 + q_T$. Combining all the above and observing that $\sum_{h \in [n]} 1 - \sum_{p \in [n+1]} 1 = -1$, we get

$$
-1 = \sum_{p \in [n+1]} \sum_{h \in [n]} J(p, h) - \sum_{h \in [n]} q_U + \sum_{p \in [n+1]} q_T
$$

which is the desired SA refutation. \hfill \square

### 4.3 PLS\textsuperscript{dt} \not\subseteq PPP\textsuperscript{dt}

In Section 3 we showed how PLS\textsuperscript{dt} \not\subseteq PPP\textsuperscript{dt} follows from the result PLS\textsuperscript{dt} \not\subseteq PPADS\textsuperscript{dt} of [GHJ+22] together with our main technical Lemma 3.1. In this section we give a more self-contained argument, deriving PLS\textsuperscript{dt} \not\subseteq PPP\textsuperscript{dt} directly from the proof-complexity lower-bound shown in [GHJ+22] on SA refutations of a CNF formula associated with Sink-or-DAG. Our proof is similar to that in [GHJ+22], except that we use LPHP where they use a formula associated with Sink-or-Line. Similar TFNP separations based on proof-complexity bounds are shown in [BCE+98, BOM04].

**Definition 4.3.** Suppose $(R_n)_{n \in \mathbb{N}}$ is a search problem in TFNP\textsuperscript{dt}, where for each $n$ we have $R_n \subseteq \{0, 1\}^{m_n \times \mathcal{O}_n}$, with $(x, y) \in R_n$ decided by a decision tree $T_y$ of depth $\text{polylog}(n)$, querying bits of $x$. The associated CNF is

$$
\text{CNF}(R_n) := \bigwedge_{y \in \mathcal{O}_n} \bigwedge_{b \in T_y[1]} \neg b,
$$

where $\neg b$ is the clause saying that “some edge in $b$ is false”. Precisely, CNF$(R_n)$ uses literals from $Z$ (as $x$ is reserved as a name for a binary string). We let $B$ be the conjunction of \{ $z_i : x_i = 1$ is an edge in $b$ \} and \{ $\bar{z}_i : x_i = 0$ is an edge in $b$ \} and take $\neg b$ to be the negation of $B$.

By the bounds in the definition of a TFNP\textsuperscript{dt} problem, CNF$(R_n)$ has $q\text{poly}(n)$ many variables and width $\text{polylog}(n)$. It expresses that the instance $x \in$
\{0,1\}^m_n \ given \ by \ the \ variables \ Z \ is \ such \ that \ T_y(x) = 0 \ for \ all \ y \in \mathcal{O}_n; \ in
other \ words, \ that \ it \ has \ no \ solution. \ Since \ R_n \ is \ total, \ it \ is \ unsatisfiable.

As \ before, \ we \ will \ also \ treat \ this \ CNF \ as \ a \ system \ of \ polynomial \ equations. \ Let \ CNF^*(R_n) \ be \ this \ system \ of \ equations \ after \ making \ the \ substitution \ \tilde{z}_i \mapsto (1 - z_i) \ for \ each \ literal \ z_i. \ Now \ we \ can \ state \ the \ lower \ bound
from \ [GHJ+22] \ that \ we \ will \ use.

**Proposition 4.4** ([GHJ+22, Lemma 3]). Any refutation of \(\text{CNF}^*(\text{Sink-of-DAG}_{n^2})\) \ in \(\text{SA}\) without twin variables \ in \ degree \ \(n^{o(1)}\) \ must \ have \ magnitude \ \(2^{\Omega(n)}\).

**Corollary 4.5.** Any refutation of \(\text{CNF}(\text{Sink-of-DAG}_{n^2})\) \ in \(\text{SA}\) \ (with twin variables) \ in \ degree \ \(\text{polylog}(n)\) \ must \ have \ magnitude \ \(2^{\Omega(n)}\).

**Proof.** For \ each \ polynomial \(r\) \ in \ such \ a \ refutation, \ let \(r^*\) \ be \(r\) \ after \ applying \ all \ substitutions \ \(z_i \mapsto (1 - z_i)\) \ and \ simplifying \ the \ result \ so \ that \ the \ polynomial \ is \ again \ written \ as \ a \ sum \ of \ monomials, \ none \ of \ which \ occurs \ twice, \ even \ with \ different \ coefficients. \ This \ operation \ does \ not \ increase \ degree, \ but \ it \ may \ increase \ magnitude; \ for \ example \ if \(r\) \ is \(z_i - \tilde{z}_i\) \ then \(r^*\) \ is \(2\tilde{z}_i - 1.\) \ However \ it \ increases \ the \ magnitude \ by \ at \ most \ \(2^\deg(r),\) \ since \ this \ is \ the \ number \ of \ monomials \ in \(r\) \ that \ can \ contribute \ to \ a \ single \ monomial \ in \(r^*\). \ \(\square\)

We \ introduce \ one \ more \ search \ problem, \ that \ of \ finding \ a \ falsified \ clause \ in \ an \ unsatisfiable \ CNF, \ given \ an \ assignment. \ This \ is \ essentially \ the \ inverse \ of \ the \ operation \ \(\text{CNF}(R_n)\) \ above \ that \ turns \ a \ search \ problem \ into \ a \ CNF.

**Definition 4.6** (Search(\(\varphi_n\))). Let \(\varphi_n\) \ be \ an \ unsatisfiable \ CNF \ \(C_1 \land \cdots \land C_L\), \ over \ \(m_n\) \ variables. \ The \ falsified \ clause \ search \ problem \ for \ \(\varphi\) \ is \ the \ relation \ \(\text{Search}(\varphi_n) \subseteq \{0,1\}^{m_n} \times [\ell]\) \ where \ an \ instance \ \(x \in \{0,1\}^{m_n}\) \ \(\text{encodes} \ \) a \ Boolean \ assignment \ to \ the \ variables \ of \ \(\varphi,\) \ and \ \((x,i) \in \text{Search}(\varphi_n)\) \ if \ and \ only \ if \ the \ clause \ \(C_i\) \ is \ falsified \ by \ \(x.\)

**Proposition 4.7.** Suppose \(\text{CNF}(R_n)\) \ is \ a \ search \ problem \ in \ \text{TFNP}^{dt}. \ Then \ \text{Search}(\text{CNF}(R_n)) \leq R_n \ \text{for} \ each \ \(n.\)

**Proof.** For \ an \ instance \ of \ \(\text{Search}(\text{CNF}(R_n))\), \ suppose \ \((x,y) \in R_n.\) \ We \ run \ the \ tree \ \(T_y\) \ verifying \ \((x,y) \in R_n\) \ to \ find \ which \ branch \ is \ true \ in \ \(x,\) \ and \ this \ tells \ us \ which \ clause \ of \ \text{CNF}(R_n) \ is \ false. \ \(\square\)

The \ next \ result \ is \ an \ alternative \ way \ to \ prove \ the \ result \ in \ [GHJ+22, \ Lemma 7]. \ That \ is, \ if \ a \ problem \ is \ in \ \text{PPADS}^{dt} \ then \ its \ \text{CNF} \ has \ small \ \text{SA} \ refutations.

**Lemma 4.8.** Suppose \(\varphi\) \ is \ a \ \(r\)-\text{CNF} \ and \ \text{Search}(\varphi) \leq_d \text{LEFTPigeon}_{m_r}, \ where \ \(r, d \in \text{polylog}(m).\) \ Then \ \(\varphi\) \ has \ a \ \text{SA} \ refutation \ of \ degree \ \text{polylog}(m) \ and \ magnitude \ \text{qpoly}(m).\)

Before \ proving \ the \ lemma \ we \ derive \ our \ main \ result.

**Restated Theorem 1.2.** \(\text{PLS}^{dt} \not\subseteq \text{PPP}^{dt}.\)
Proof. Suppose that $\text{PLS}^{dt} \subseteq \text{PPP}^{dt}$. Then in particular $\text{ITER} \leq \text{Pigeon}$. Thus for all $n$ there exists $m \in \text{qpoly}(n)$ such that $\text{ITER}_n \leq \text{Pigeon}_m$. By Lemma 3.1, in fact $\text{ITER}_n \leq \text{LEFTPigeon}_m$.

Proposition 2.6 tells us that $\text{Sink-of-DAG}_{n^2} \leq \text{ITER}_n$, which implies that $\text{Sink-of-DAG}_{n^2} \leq \text{LEFTPigeon}_m$. Thus by Proposition 4.7 we conclude

$$\text{Search(CNF}(\text{Sink-of-DAG}_{n^2})) \leq \text{LEFTPigeon}_m.$$ 

We can now apply Lemma 4.8, which tells us that $\text{CNF}(\text{Sink-of-DAG}_{n^2})$ has polylog$(n)$-degree, qpoly$(n)$-magnitude SA refutations. This contradicts Corollary 4.5. $\square$

We now prove Lemma 4.8.

Proof of Lemma 4.8. Suppose $\varphi$ is an $r$-CNF such that $\text{Search}(\varphi)$ is reducible to $\text{LEFTPigeon}_m$ by some depth-$d$ reduction $(f, g)$. We will use this reduction, together with the SA refutation of $\text{LPHP}_m$ in Lemma 4.2, to construct an efficient SA refutation of $\varphi$.

For any assignment $x$ to the variables of $\varphi$, the reduction gives us $f(x)$ defining an instance $F, G$ of $\text{LEFTPigeon}_m$. To find $F(p)$ we need to read $\log n$ bits of $f(x)$, each computed by a tree of depth $d$ querying $x$. Therefore we can compute $F(p)$ by such a tree of depth $d \log m$. Let $T_p$ be this tree, and let $U_h$ be a similar tree computing $G(h)$.

We extend $T_p$ (but not $U_h$) as follows. At the end of each branch in $T_p$ we run the tree for $g(x, p)$, getting some output $i$ naming a clause $C_i$ of $\varphi$. Then we query $x$ at the location of every variable in $C_i$. At the end we output whichever hole labelled the original branch in $T_p$, so the tree still computes $F(p)$. Its depth is now at most $e := d \log m + d + r \in \text{polylog}(m)$.

We make a final change to each $T_p$ and $U_h$, which is to replace the labels $x_i = 1$ or $x_i = 0$ on the edges with literals $z_i$ or $\bar{z}_i$. By Lemma 4.2 the formula $\text{LPHP}_m(T, \bar{U})$ has an SA refutation of degree $2e$ and magnitude $\text{poly}(m, 2^e)$. We will show that every axiom of $\text{LPHP}_m(T, \bar{U})$ is a weakening of some clause of $\varphi$. The lemma follows immediately.

Consider any such axiom. It has the form $bc = 0$, for $b \in T_p[h]$ and $c \in U_h[-p]$, where $b$ and $c$ are consistent. Let $x$ be any total assignment setting $bc = 1$. Since $T_p$ computes $F(p)$ and $U_h$ computes $G(h)$ it follows that, in the corresponding instance $f(x)$ of $\text{LEFTPigeon}_m$, we have $F(p) = h$ but $G(h) \neq p$. Thus $p$ is a solution of $f(x)$, so $g(p, x)$ is the index of a clause $C_i$ in $\varphi$ which is false in $x$. Suppose we are evaluating the tree $T_p$ on $x$. We go along the branch $b$. By construction as we go we query every literal in $C_i$, and since $C_i$ is false in $x$, all these literals are recorded as false along $b$. Thus $b$ is a weakening of the monomial corresponding to the clause $C_i$. $\square$
References

[BCE+98] Paul Beame, Stephen Cook, Jeff Edmonds, Russell Impagliazzo, and
Toniann Pitassi. The relative complexity of NP search problems.

Lower bounds on Hilbert’s Nullstellensatz and propositional proofs.
In Proceedings 35th Annual Symposium on Foundations of Computer

[BJ12] Samuel R. Buss and Alan S. Johnson. Propositional proofs and re-
ductions between NP search problems. Annals of Pure and Applied

[BK94] Samuel R. Buss and Jan Krajíček. An application of Boolean com-
plexity to separation problems in bounded arithmetic. Proceedings

[BOM04] J. Buresh-Oppenheim and T. Morioka. Relativized NP search pro-
blems and propositional proof systems. In Proceedings. 19th IEEE
Annual Conference on Computational Complexity, pages 54–67, Los

[GHJ+22] Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre,
William Pires, Robert Robere, and Ran Tao. Separations in proof

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yann-
akakis. How easy is local search? Journal of Computer and System

[MP91] Nimrod Megiddo and Christos H. Papadimitriou. On total func-
tions, existence theorems and computational complexity. Theoretical

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument
and other inefficient proofs of existence. Journal of Computer and

between the continuous and convex hull representations for zero-one
programming problems. SIAM Journal on Discrete Mathematics,

A Postponed reductions

Proposition A.1. $\text{ITER}_n \leq \text{Sink-of-DAG}_n \leq \text{ITER}_{n^2}$. 
Proof. Given an instance $x$ of $\text{ITER}_n$, that is, a function $S : [n] \to [n]$ with $S(i) \geq i$, we define an instance for $\text{SINK-OF-DAG}_n$ as the following function $S' : [n] \times [n] \to ([n] \cup \{\text{null}\})$:

$$S'(i, j) = \begin{cases} S(j) & \text{if } j \geq i \text{ and } S(j) > j \\ \text{null} & \text{otherwise} \end{cases}$$

The function $f$ maps $x$ to an encoding of $S'$. To describe the function $g$ mapping solutions of $f(x)$ to solutions of $x$, we consider the three kinds of solutions to $\text{SINK-OF-DAG}_n$. If $(1, 1)$ is a solution with $S'(1, 1) = \text{null}$, then $S(1) = 1$ and $0$ is a solution of the $\text{ITER}_n$ instance. If $(n, j)$ is a solution with $S'(n, j) \neq \text{null}$, then $j = n$ and $S(n) > n$, which is impossible. Lastly if $(i, j)$ is a solution with $i < n$, $S'(i, j) \neq \text{null}$ and $S'(i + 1, S'(i, j)) = \text{null}$, then $j \geq i$ and $S'(i, j) = S(j) > j$, but either $S(j) < i + 1$, which is impossible, or $S(S(j)) = S(j)$, in which case $j$ is a solution of the $\text{ITER}_n$ instance.

For the other direction, given an instance of $\text{SINK-OF-DAG}_n$, that is, a function $S : [n] \times [n] \to ([n] \cup \{\text{null}\})$, we use the mapping $(i, j) \mapsto n(i - 1) + j$ that maps the rows of grid $[n] \times [n]$ into successive blocks along a single row $[n^2]$. We define an instance of $\text{ITER}_{n^2}$ as the following function $S' : [n^2] \to [n^2]$:

$$S'(u) = \begin{cases} S(i, j) & \text{if } u = n(i - 1) + j \text{ and } S(i, j) \neq \text{null} \\ u & \text{otherwise} \end{cases}$$

If $u$ is a solution of the $\text{ITER}_{n^2}$ instance then the corresponding $(i, j)$ is a solution of the $\text{SINK-OF-DAG}_n$ instance. \qed

**Proposition A.2.** $\text{LEFTPIGEON}_n \leq \text{SINK-OF-LINE}_{n+1} \leq \text{LEFTPIGEON}_n$.

Proof. For the purpose of these reductions we will make a cosmetic change to $\text{LEFTPIGEON}_n$, defining the set of holes to be $\{2, \ldots, n+1\}$ rather than $[n]$. The set of pigeons remains as $[n+1]$.

The reduction of $\text{LEFTPIGEON}_n$ to $\text{SINK-OF-LINE}_{n+1}$ is trivial. Given an instance $F$, $G$ of $\text{LEFTPIGEON}_n$, we define an instance $S$, $P$ of $\text{SINK-OF-LINE}_{n+1}$ by $S(u) = F(u)$ and $P(u) = G(u)$ (and $P(1) = 1$). Suppose $u$ is a solution to this new instance. If $G(F(u)) \neq u$ then $u$ is already a solution to the $\text{LEFTPIGEON}_n$ instance. Otherwise, $u$ has out-degree 1. Since $u$ is a solution to $\text{SINK-OF-LINE}_{n+1}$, it must be that $u = 1$ and $1$ has in-degree 1, and in particular $F(w) = 1$ for some $w$. But this is impossible.

For the other direction, given an instance $S$, $P$ of $\text{SINK-OF-LINE}_{n+1}$, we put

$$F(u) = \begin{cases} S(u) & \text{if } P(S(u)) = u \\ u & \text{otherwise} \end{cases} \quad G(v) = \begin{cases} P(v) & \text{if } S(P(v)) = v \\ v & \text{otherwise} \end{cases}$$

Formally $F$ should only take values in the interval $\{2, \ldots, n+1\}$, to match our definition of $\text{LEFTPIGEON}_n$. Note that if $F(u) = 1$ then either $S(u) = 1$ and $P(1) = u$, so 1 has in-degree 1, or $u = 1$ and $P(S(1)) \neq 1$, so 1 has out-degree 0.
In either case, 1 is a solution to the \text{Sink-of-Line}_{n+1} instance. To meet the formal requirement replace $F$ with $F'$, defined to be 2 wherever $F$ is 1.

Suppose we have a solution to this instance of \text{LeftPigeon}_n. That is, $u, v, u'$ with $u' \neq u$ and $F'(u) = v$, $G(v) = u'$. Then either 1 is a solution to the \text{Sink-of-Line}_{n+1} instance, or we can forget about $F'$ and assume we have $F(u) = v$.

We must have $P(S(u)) \neq u$, since otherwise we are in the first case of the definition of $F(u)$, with $v = S(u)$ and thus $P(v) = u$. This implies that $S(P(v)) = v$, so we are in the first case of the definition of $G(v)$, with $G(v) = P(v) = u$, contradicting that $G(v) = u' \neq u$.

It follows from $P(S(u)) \neq u$ that $u$ has out-degree 0 and that $v = u$, so $G(u) = G(v) = u' \neq u$. Thus we must be in the first case of the definition of $G(u)$, which means that $u$ has in-degree 1. Therefore we have found a sink. \hfill \qed