Electronic Colloquium on Computational Complexity, Report No. 120 (2018)

The Complexity of Multi-source Variants of the End-of-Line Problem, and the Concise Mutilated Chessboard

Alexandros Hollender[1] and Paul W. Goldberg[2]

[1]Department of Computer Science, University of Oxford
alexandros.hollender@cs.ox.ac.uk
[2]Department of Computer Science, University of Oxford
Paul.Goldberg@cs.ox.ac.uk

June 21, 2018

Abstract

The complexity class PPAD is usually defined in terms of the End-of-Line problem, in which we are given a concise representation of a large directed graph having indegree and outdegree at most 1, and a known source, and we seek some other degree-1 vertex. We show that variants where we are given multiple sources and seek one solution or multiple solutions are PPAD-complete. Our proof also shows that a multiple source Sink problem where we look for multiple sinks instead of one is equivalent to Sink (i.e. PPADS-complete). Using our result, we provide a full proof of PPAD-completeness of Imbalance. Finally, we use these results together with earlier work on the 2D-Brouwer problem to investigate the complexity of a new total search problem inspired by the mutilated chessboard tiling puzzle.

1 Introduction

The complexity class TFNP denotes the set of *total search* problems belonging to FNP: a problem in this class has the property that every instance has a solution whose correctness may be checked in polynomial time. The best-known example of such a problem is Factoring: given a number, compute its prime factorisation. Every number has a prime factorisation, and a prime factorisation can be efficiently checked for correctness (multiply the factors to check that we obtain the input number, and run the AKS primality test [1] to check that they are indeed prime). Factoring is one of a number of seemingly-hard problems in TFNP, and these problems are of enduring interest since they lie at the frontier of P and NP: they cannot be NP-hard unless NP=co-NP [26], so alternative notions of hardness have to be investigated.

TFNP is a "semantic" complexity class: it does not seem to have complete problems. The syntactic subclass PLS was introduced by [22], and subclasses PPAD, PPADS, PPA, and PPP by [29], in order to classify the complexity of various important TFNP problems. These are classes of problems whose totality is proved via certain combinatorial principles corresponding to well-known elementary non-constructive existence proofs:

- PPP (embodying the pigeonhole principle);

ISSN 1433-8092

- PPA (embodying the principle "every graph with an odd-degree node must have another");
- PPAD ("every directed graph with an unbalanced node must have another")[1];
- PPADS (same as PPAD, except we are looking for an *oppositely unbalanced* node), and
- PLS ("every dag has a sink").

These complexity classes are usually defined in terms of the corresponding search problem on an exponentially-large graph represented by circuits. For example, PPAD is defined as the set of problems that reduce in polynomial time to End-of-Line, in which a directed graph on $[2^n]$ with indegree/outdegree at most 1 is presented via circuits which identify the neighbour(s) of a given vertex. If some given vertex is a source in the graph, the problem is to find some other degree-1 vertex.

These classes have been very successful in capturing the complexity of interesting TFNP problems. Many local optimisation problems are PLS-complete [22, 30, 24, 15, 14]. PPAD has turned out to be important in capturing the complexity of various "natural" computational challenges, including Nash equilibrium computation and others [12, 8, 10, 5, 9, 23]. PPA captures the complexity of the consensus-halving problem [18], necklace-splitting, and ham sandwich bisection [17]. On the other hand, various problems continue to defy this taxonomy, including indeed Factoring, which has so far just been partially related to PPA and PPP [21].

A question of general interest is what sort of problems can be classified using the above classes, in view of the existence of problems that are not known to be complete for any of them. See [19, 20] for a more detailed discussion of this general question, and examples of other "rogue problems". In this paper we consider combinatorial principles related to PPAD and PPADS, leading to the following problems, on graphs with indegree/outdegree at most 1:

- given k sources and $\ell \neq k$ sinks, find another degree-1 vertex
- given k sources and $\ell < k$ sinks, find $k - \ell$ other sinks
- given k sources, find k sinks (or alternatively one sink, or indeed some intermediate number of sinks, possibly with the option of outputting some larger number of sources instead...)

We show that these variants can be classified in terms of the original problems. Our results also yield a full proof of PPAD-completeness of the problem Imbalance, in which we are given a directed graph and an unbalanced node, and have to find another unbalanced node. Although this result was previously stated in [4], the proof idea that was provided only seems to work if the imbalance in the given node is exactly 1.

Finally, we apply the above results to a problem that we call "concise mutilated chessboard". This problem is inspired by a well-known puzzle in which we consider a chessboard that has had two diagonally opposite corner squares removed, and we are asked to prove that it is impossible to tile the resulting chessboard with dominoes (where a domino may occupy two adjacent squares). This impossibility is proved by noting that a domino

[1]We verify that this informal characterisation is indeed correct by proving that Imbalance is equivalent to End-of-Line. The previous proof only worked if the given node had unbalance exactly 1, see subsection 3.4 for more details.

occupies one black and one white square, and the mutilated chessboard is missing two squares of the same colour. The computational problem envisages a domino-tiling of an exponentially-large mutilated chessboard, presented via a circuit that identifies for any square, which adjacent square it shares with a domino. Clearly there should be an error in this tiling, and the computational challenge is to find it. In Section 5 we show that variants of this problem are PPAD-complete.

Overview of our results. Section 3 presents Theorem 1: given k sources, it is PPAD-complete to find any other degree-1 vertex (k may be any constant, or a quantity polynomial in n). In reducing to End-of-Line, notice that we cannot simply ignore all but one of the sources, since a solver for End-of-Line would be allowed to return one of those sources. We present a more sophisticated proof of the equivalence. Section 4 considers versions where multiple degree-1 vertices (other than the known ones) are sought. Theorem 2 shows that if k sinks are sought, the problem is straightforwardly equivalent to Sink, i.e. is PPADS-complete. Theorem 3 shows that if k degree-1 vertices are sought, the problem is PPAD-complete. The proof builds on Theorem 1 and is more complex. Again, k may be polynomial in n. We conclude Section 4 by noting that if either k sinks or some polynomially larger number of sources are sought, then the problem is still PPAD-complete. Thus, PPADS-completeness only arises when we insist on obtaining at least one sink; all other versions are PPAD-complete.

In Section 5 we apply these results (along with results of Chen and Deng on the two-dimensional Brouwer problem [7]) to obtain PPAD-completeness for variants of the concise mutilated chessboard problem. The "original" version with two squares removed, reduces most naturally to the 2-source End-of-Line problem, and versions with k squares removed, reduce to k-source End-of-Line.

2 Definitions and Notation

A computational search problem is a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, interpreted as follows: $y \in \{0,1\}^*$ is a solution to instance $x \in \{0,1\}^*$, if and only if $(x, y) \in R$. The search problem R is in FNP (*Functions in NP*), if R is polynomial-time computable (i.e. $(x, y) \in R$ can be decided in polynomial time) and there exists some polynomial p such that $(x, y) \in R \implies |y| \leq p(|x|)$. Here $\{0,1\}^*$ denotes all finite length bit-strings and $|x|$ is the number of bits in bit-string x.

The class TFNP (*Total Functions in NP* [26]) contains all search problems R that are in FNP and are *total*, i.e. every instance has at least one solution. Formally, this corresponds to requiring that for every $x \in \{0,1\}^*$ there exists $y \in \{0,1\}^*$ such that $(x, y) \in R$.

Let R and S be total search problems in TFNP. We say that R (many-one) reduces to S, if there exist polynomial-time computable functions f, g such that

$$(f(x), y) \in S \implies (x, g(x, y)) \in R.$$

Note that if S is polynomial-time solvable, then so is R. We say that two problems R and S are (polynomial-time) equivalent, if R reduces to S and S reduces to R.

The study of the complexity of problems in TFNP mainly relies on classifying them with respect to its subclasses: PPA, PPAD, PPADS, PPP, PLS. Each of these subclasses can be defined using a problem that is representative of what the class encompasses. PPA is defined as the class of all total search problems that reduce to the problem Leaf [29]. PPAD and PPADS are defined as the classes of all problems that reduce to End-of-Line and Sink [29, 4], respectively. Similarly, PPP and PLS can be defined using PigeonholeCircuit [29] and Iter [22, 27].

An important question is whether there are other such subclasses that contain interesting problems. In particular, one might wonder if slightly tweaking some aspect of the defining problem of a class, also changes the class it defines. For example, if we change the End-of-Line problem and only accept a sink as a solution (not another source), then we obtain a (likely) different class: PPADS. In the next sections, we show that End-of-Line is quite robust with respect to modifications. We consider various natural variants of the problem and show that they are equivalent to the original problem.

At this point we should note that we work in the so-called *white-box* model, i.e. functions that are part of the input are given as Boolean circuits. However, our reductions do not make use of the ability to "analyse" circuits by investigating their logical gates and inner circuitry. As a result our reductions are of the *black-box* type and also apply in the corresponding oracle model (see [4]).

3 End-of-Line Variants

3.1 The End-of-Line Problem

The End-of-Line problem is informally defined as follows: given a directed graph where each vertex has in- and out-degree at most 1 and given a known source of this graph, find a sink or another source. The problem is computationally challenging, because the graph is not given explicitly in the input. Instead, we are given an implicit concise representation of the graph through circuits that compute the predecessor and successor of a vertex in the graph. In what follows, we interpret the input and output of the circuits, which are elements in $\{0,1\}^n$, as the numbers $\{0,1,\dots,2^n-1\}$.

Definition 1 (End-of-Line)**.** The End-of-Line problem is defined as: given Boolean circuits S, P with n input bits and n output bits and such that $P(0) = 0 \neq S(0)$, find x such that $P(S(x)) \neq x$ or $S(P(x)) \neq x \neq 0$.

The circuits define a graph as follows. There is a directed edge from vertex x to y ($x \neq y$), if and only if $S(x) = y$ and $P(y) = x$. Note that any badly defined edge, i.e. $S(x) = y$ and $P(y) \neq x$, or $P(y) = x$ and $S(x) \neq y$, qualifies as a solution of End-of-Line as defined above (because $P(S(x)) \neq x$ or $S(P(x)) \neq x$ respectively). Note that 0 is a source of the graph, unless $P(S(0)) \neq 0$, in which case 0 is a valid solution to the problem as stated above.

It is easy to check that this formal definition of the problem is computationally equivalent to the informal description given above. It is well known that End-of-Line is PPAD-complete [29]. Furthermore, reduction from End-of-Line is a very common technique to show PPAD-hardness (e.g. [12, 7]).

Given the importance of End-of-Line, it seems interesting to investigate how modifying the problem affects its complexity. For now, we have only provided a purely theoretical motivation for doing this. Fortunately, as we will see in section 5, modified End-of-Line problems can indeed be useful to investigate the complexity of other problems.

In the next few sections we show that various modified versions of End-of-Line are equivalent to the original problem. Hardness will be easy to show for those variants, but it is the inclusion in PPAD (i.e. reduction to End-of-Line) that is more challenging.

3.2 Multiple Sources: One Source to Rule Them All

The most natural modification to consider is perhaps the following: what if, instead of one, we are given two sources of the End-of-Line graph. The objective remains the same: find a sink or another source. Intuitively, the problem might seem easier, because the existence of two sources implies the existence of at least two sinks, hence more potential solutions. However, at the same time, by having more known sources, we have eliminated some potential solutions, because known sources are not acceptable solutions.

Formally, we consider the following variant of End-of-Line, which corresponds to the case where we are given k known sources, for some constant k.

Definition 2 (kS-EoL)**.** Let $k \in \mathbb{N}$. The k-source End-of-Line problem, abbreviated kS-EoL, is defined as: given circuits S, P with n inputs and n outputs and such that $P(z) = z \neq S(z)$ for all $z < k$, find x such that $P(S(x)) \neq x$ or $x \geq k$ such that $S(P(x)) \neq x$.

Note that we are still only looking for a single sink or a single other source. It is easy to see that End-of-Line reduces to kS-EoL: simply take the disjoint union of k separate copies of the original graph. In this new graph, we know exactly k sources and any sink or other source is a solution to the original problem. Reducing in the other direction is much harder. If we interpret a kS-EoL instance as an End-of-Line instance, then any of the other $k-1$ known sources are valid End-of-Line solutions, but not valid solutions of the original kS-EoL instance. Thus, a more complicated reduction seems necessary to ensure that we go from k known sources to only 1.

Undirected case. In passing, let us note that in the undirected case this kind of generalisation is trivial. The undirected analogue of End-of-Line is Leaf: given an undirected graph where every vertex has degree at most 2 and given a vertex of degree 1, find another vertex of degree 1, i.e. another leaf. Assume that we know k leaves instead of just one. If k is even, then the problem is not even in TFNP. If k is odd, then we can add edges between known leaves until exactly one is left. Thus, the problem is equivalent to Leaf.

This kind of reduction does not work for the directed case. Fortunately, it turns out that k-source End-of-Line is indeed equivalent to the standard End-of-Line. Furthermore, this holds even if k is not constant, but is bounded by some polynomial in n. Below we prove the result for constant k. See Remark 1 and Definition 3 for more details on the case where k is not constant.

Theorem 1. *For any $k \in \mathbb{N}$, kS-EoL is equivalent to End-of-Line.*

Proof Idea. Before giving a formal proof, we briefly present the proof idea for $k = 2$. Let $G = (V, E)$ be the graph corresponding to some 2S-EoL instance. Consider constructing the graph G' where the vertices are of the form $\{x, y\}$, where $x \neq y$ are vertices of G. The perhaps most natural way to define the successor circuit is to let $S'(\{x,y\}) = \{S(x), S(y)\}$. However, if $S(x) = x$ or $S(y) = y$, then this is not a good idea, because two vertices would end up having the same successor (e.g. $\{x,y\}$ and $\{P(x), y\}$, if $S(x) = x$ and $P(x) \neq x$). Thus, we let $S'(\{x,y\}) = \{x,y\}$ if $S(x) = x$ or $S(y) = y$. We define the predecessor circuit P' analogously.

However, there is still a problem: for any x with $P(x) \neq x \neq S(x)$, $\{0,x\}$ and $\{1,x\}$ are sources of G', but they don't yield a solution to the 2S-EoL instance. Thus, we need to "eliminate" all those sources. We achieve this by adding a path connecting $\{t_0, x\}$ to $\{0, x\}$, where t_0 is the sink at the end of the path starting at source 0. Thus, in a sense we "sacrifice" some sinks to eliminate "bad" sources. We do this for all sources of the type $\{s, x\}$, where s is a source of G and $P(x) \neq x \neq S(x)$. Of course, we would prefer to only do it for $s = 0$ and $s = 1$, but we have to do it for all sources of G, because given a vertex of G, we cannot efficiently decide whether it lies on a path that started from 0 or 1 or some other source. The generalisation to larger k is slightly more complicated, but follows the same central idea.

Proof. Reduction from End-of-Line to kS-EoL works by taking k copies of the original graph. Note that any sink or unknown source of the new graph immediately provides a solution of the original problem. The challenging step is the reduction from kS-EoL to End-of-Line.

Let $k, n \in \mathbb{N}$ and (S, P) be an instance of kS-EoL. For simplicity, we are going to assume that $P(S(z)) = z$ for all $z < k$. We also assume that for all x we have $P(S(x)) = x$ unless $S(x) = x$, and $S(P(x)) = x$ unless $P(x) = x$. The first assumption corresponds to requiring that the first k vertices indeed be sources. Note that we can check this using $O(k)$ evaluations of the circuits, i.e. in polynomial time in $|S|+|P|$, and any "false" source is a solution to the kS-EoL instance. Here, $|S| + |P|$ denotes the size of the input, i.e. the sum of the sizes of the two circuits given in the input. The second assumption corresponds to requiring that the graph is in some sense well-defined. This requirement can be enforced by a slight modification of the circuits that can be done in time polynomial in $|S| + |P|$. Any solution of the modified instance is a solution of the original instance.

Let V^s, V^t, V^p be the following subsets of $\{0, \dots, 2^n - 1\}$:

- $V^s = \{x : P(x) = x, S(x) \neq x\}$. This corresponds to all the sources of the graph.
- $V^t = \{x : S(x) = x, P(x) \neq x\}$. This corresponds to all the sinks of the graph.
- $V^p = \{x : P(x) \neq x, S(x) \neq x\}$. This corresponds to all the vertices that are not isolated, but are neither sources nor sinks. We call those *path vertices.*

Note that members of all those subsets are recognisable in polynomial time in $|S| + |P|$. Let $V = V_P \cup V_S \cup V_I$. Note that isolated vertices are not contained in V.

Let $G = (V, E)$ be the graph represented by circuits S, P (without isolated vertices). Below we will give an inductive construction of the graph $G_k = (V_k, E_k)$ that will have the following properties:

- The sources of G_k are all the sets of the form $\{s_1, \dots, s_k\}$, where $s_1, \dots, s_k \in V^s$ are distinct sources of the original graph.
- The sinks of G_k are all the sets of the form $\{t_1, \dots, t_k\}$, where $t_1, \dots, t_k \in V^t$ are distinct sinks of the original graph.

- Every vertex of G_k can be represented using at most $kn + k^2$ bits. There exists a polynomial algorithm that for each such bit-string decides whether it represents a vertex of G_k or not.
- The successor and predecessor circuits S_k, P_k have polynomial size with respect to $|S| + |P|$ and can be constructed in polynomial time.

Note that the only known source of G_k is $\{0, 1, \ldots, k-1\}$. Any other source or any sink of G_k contains at least one unknown source or sink of the original graph. Thus, if we can construct (in polynomial time in $|S| + |P|$) circuits P_k, S_k that represent this graph, then we have a polynomial reduction from kS-EoL to End-of-Line.

We now give a formal inductive construction of G_ℓ. For $\ell = 1$, G itself already satisfies these properties (if we interpret any vertex x as $\{x\}$). Let $\ell \geq 2$. Assume that we know how to construct G_i for all $1 \leq i \leq \ell - 1$. We then construct G_ℓ as follows.

For any set X and any $j \in \mathbb{N}$, let $\text{Subsets}(X, j) := \{A \subseteq X : |A| = j\}$, i.e. the set of all subsets of X with cardinality exactly j.

The set of vertices of G_ℓ is defined as

$$V_\ell = \text{Subsets}(V, \ell) \cup \bigcup_{i=1}^{\ell-1} \left(\text{Subsets}(V^p, i) \times V_{\ell-i}\right).$$

Let us investigate the number of bits needed to represent an element in V_ℓ. We need $n \cdot i$ bits to represent an element in $\text{Subsets}(V^p, i)$. By induction hypothesis, $(\ell - i)n + (\ell - i)^2$ bits suffice to represent an element in $V_{\ell-i}$. Thus, for any $i \in \{1, \ldots, \ell - 1\}$, at most $n + (\ell - 1)n + (\ell - 1)^2$ bits suffice to represent an element in $V_{\ell-i}$. We add another $\lceil \log_2(\ell - 1) \rceil \leq \ell - 1$ bits to explicitly store the value of i. Thus, we can represent any element in $\bigcup_{i=1}^{\ell-1} \left(\text{Subsets}(V^p, i) \times V_{\ell-i}\right)$ using at most $\ell n + (\ell - 1)^2 + \ell - 1$ bits. We add one more bit to decide whether the element is in $\text{Subsets}(V, \ell)$ or not. We only need ℓn bits to represent an element in $\text{Subsets}(V, \ell)$. Putting everything together, we get an upper bound on the number of bits needed to represent an element in V_ℓ

$$1 + \max\{\ell n, \ell n + (\ell - 1)^2 + \ell - 1\} = \ell n + (\ell - 1)^2 + \ell \leq \ell n + \ell^2.$$

Furthermore, given a bit-string, it is easy to "decode" it and decide whether it is a vertex of G_ℓ or not (and if it is not, then treat it as an isolated vertex).

We now give the construction of the predecessor and successor circuits. Let $\{x_1, \ldots, x_\ell\} \in \text{Subsets}(V, \ell)$. Assume that we have reordered the elements in the set such that for some $i, j \in \{0, \ldots, \ell\}$ we have $x_1, \ldots, x_i \in V^p$, $x_{i+1}, \ldots, x_j \in V^s$ and $x_{j+1}, \ldots, x_\ell \in V^t$.

- If $j = \ell$ (i.e. only sources and path vertices), then we define

$$S_\ell(\{x_1, \ldots, x_\ell\}) = \{S(x_1), \ldots, S(x_\ell)\}$$

  Furthermore, if we also have $1 \leq i \leq \ell - 1$ (i.e. at least one path vertex and source), then we define

$$P_\ell(\{x_1, \ldots, x_\ell\}) = (\{x_1, \ldots, x_i\}, \{x_{i+1}, \ldots, x_\ell\}) \in \text{Subsets}(V^p, i) \times V_{\ell-i}$$

- If $i = j$ and $j < \ell$ (i.e. only path vertices and sinks), then we define

$$P_\ell(\{x_1, \ldots, x_\ell\}) = \{P(x_1), \ldots, P(x_\ell)\}$$

Furthermore, if we also have $i \geq 1$ (i.e. at least one path vertex and sink), then we define

$$S_\ell(\{x_1, \dots, x_\ell\}) = (\{x_1, \dots, x_j\}, \{x_{j+1}, \dots, x_\ell\}) \in \text{Subsets}(V^p, j) \times V_{\ell-j}$$

(recall that we are in the case $i = j$).

- If $i < j$ and $j < \ell$ (i.e. both sources and sinks, as well as path vertices potentially), then $\{x_1, \dots, x_\ell\}$ is an isolated vertex.

Let $i \in \{1, \dots, \ell - 1\}$. Let $(\{x_1, \dots, x_i\}, z) \in \text{Subsets}(V^p, i) \times V_{\ell-i}$.

- We define $P_\ell((\{x_1, \dots, x_i\}, z)) = (\{x_1, \dots, x_i\}, S_{\ell-i}(z))$, except if $S_{\ell-i}(z) = z$ and $P_{\ell-i}(z) \neq z$, in which case $z \in \text{Subsets}(V^t, \ell - i)$ (by induction hypothesis) and we then define $P_\ell((\{x_1, \dots, x_i\}, z)) = \{x_1, \dots, x_i\} \cup z \in \text{Subsets}(V, \ell)$. Note that the two sets in this union are disjoint, because $x_1, \dots, x_i$ are path vertices of G, whereas z only contains sinks of G.
- We define $S_\ell((\{x_1, \dots, x_i\}, z)) = (\{x_1, \dots, x_i\}, P_{\ell-i}(z))$, except if $P_{\ell-i}(z) = z$ and $S_{\ell-i}(z) \neq z$, in which case $z \in \text{Subsets}(V^s, \ell - i)$ (by induction hypothesis) and we then define $S_\ell((\{x_1, \dots, x_i\}, z)) = \{x_1, \dots, x_i\} \cup z \in \text{Subsets}(V, \ell)$.

It is straightforward to check that in the graph represented by S_ℓ, P_ℓ every vertex has in- and out-degree at most 1. Furthermore, by construction we also get that the sources of G_ℓ are exactly the vertices in $\text{Subsets}(V^s, \ell)$ and the sinks of G_ℓ are exactly the vertices in $\text{Subsets}(V^t, \ell)$.

By induction it follows that we can construct (in polynomial time in $|S| + |P|$) circuits S_k, P_k that represent G_k.

□

Remark 1. Note that the proof above also works if k is not a constant, but is polynomial in the size of the input. In particular, if we consider a different version of the problem where the sources are explicitly given as part of the input, then it also reduces to End-of-Line. We include the definition of this problem below.

Definition 3 (MS-EoL)**.** The Multiple-Source End-of-Line problem, abbreviated MS-EoL, is defined as: given circuits S, P with n inputs and n outputs and $s_1, \dots, s_m \in \{0, \dots, 2^n - 1\}$ such that $P(s_i) = s_i \neq S(s_i)$ for all i, find x such that $P(S(x)) \neq x$ or $x \notin \{s_1, \dots, s_m\}$ such that $S(P(x)) \neq x$.

The proof above can be used to show that MS-EoL is equivalent to End-of-Line.

3.3 Multiple Sources and Sinks

A natural generalisation of Multiple-Source End-of-Line is the following problem: given an End-of-Line instance with k known sources and m known sinks, find another source or sink. Note that for this problem to be in TFNP, we must require $k \neq m$. Otherwise, there might not be any other source or sink.

It is easy to see that this problem is equivalent to End-of-Line, assuming that k and m are polynomial in n. If $k > m$, then we can add an edge from each of the m known sinks to some corresponding known source and obtain an instance with $k - m$ known sources and no known sinks. Similarly, if $k < m$, then we can first reverse all directed edges in the graph and then apply the same trick as above to obtain an instance with $m - k$ known sources.

3.4 Higher Degree Graphs: The Imbalance Problem

Up to this point, we have only considered graphs where every vertex has in- and out-degree at most 1. However, the principle that guarantees the existence of a solution in an End-of-Line graph can be generalised to higher degree graphs. If we are given a directed graph and an "unbalanced" vertex, i.e. a vertex with in-degree $\neq$ out-degree, then there must exist another unbalanced vertex.

Imbalance. Beame et al. [4] defined the corresponding problem Imbalance, which is seemingly more general than End-of-Line. In this problem, every vertex is not constrained to have in- and out-degree at most 1. Instead, in- and out-degree are bounded by some polynomial of the input length[2]. We are given a vertex that is not balanced, i.e. its in- and out-degree are not the same, and have to find another unbalanced vertex (which is guaranteed to exist). Similarly, Beame et al. also defined a generalisation of Sink, namely the problem Excess[3]: given a vertex that is in *deficiency* (i.e. in-degree < out-degree), find a vertex that is in *excess* (i.e. in-degree > out-degree).

Undirected Case. Consider the analogous generalisation in the undirected case. The analogue of End-of-Line is Leaf: given an undirected graph where every vertex has degree at most 2 and given a vertex of degree 1, find another vertex of degree 1, i.e. another leaf. The generalisation of Leaf is called Odd: given an undirected graph and a vertex of odd degree, find another vertex of odd degree. It is quite straightforward to show that Odd is equivalent to Leaf. First of all, Leaf trivially reduces to Odd.

To show that Odd reduces to Leaf, Papadimitriou [28, 29] and Beame et al. [4] use the "chessplayer algorithm" (which is based on an Euler tour argument). Let $G = (V, E)$ be an instance of Odd and $2d$ be an upper bound on the maximum degree of any vertex in G. The graph G' consists of the vertices (v, j), where $v \in V$ and $j \in \{1, \dots, d\}$. For any edge $\{u, v\} \in E$, we have a corresponding edge in G', namely $\{(u, \lceil \#_u(v)/2 \rceil), (v, \lceil \#_v(u)/2 \rceil)\}$. Here, $\#_u(v)$ is the index of v in the neighbour list of u, ordered in lexicographic order. It is easy to see that if a vertex v has even degree in G, then all its versions $(v, \cdot)$ in G' have either degree two or are isolated. Similarly, if a vertex has odd degree in G, then exactly one of its versions in G' has degree one, and all other version have degree two or are isolated. Thus, G' is a valid input to Leaf and any solution yields an odd-degree vertex of G. It is easy to check that the construction of G' (i.e. the circuit returning the (at most) two neighbours of each vertex) can be constructed in polynomial time from G (i.e. given the circuit returning the neighbour list of each vertex in G).

Directed Case. As claimed by Papadimitriou [28] and Beame et al. [4], this construction can also be applied to the directed case. Let $G = (V, E)$ be a directed graph and d be

[2]Note that this trivially holds, if the input consists of circuits that explicitly output the predecessor and successor list.

[3]The same problem was also defined by Papadimitriou [28] as the basis for the class "PQS", which he showed is equal to "PSK" (now called PPADS).

an upper bound on the in- and out-degree of any vertex in G. The graph G' consists of the vertices (v, j), where $v \in V$ and $j \in \{1, \ldots, d\}$. For any edge $(u, v) \in E$, we have a corresponding edge in G', namely $((u, \#_u^S(v)), (v, \#_v^P(u)))$. Here, $\#_u^S(v)$ corresponds to the index of v in the successor list of u, ordered lexicographically. Similarly, $\#_v^P(u)$ is the index of u in the predecessor list of v, ordered lexicographically. Again, it is easy to see that for any balanced vertex v of G, all of its versions $(v, \cdot)$ in G' will either have in- and out-degree one, or be isolated. For any unbalanced vertex v of G with in-degree p and out-degree q, there are two cases. If $p < q$, then there are $q - p$ versions of v in G' that are sources (i.e. in-degree 0 and out-degree 1) and the rest are balanced or isolated. If $p > q$, then there are $p - q$ versions of v in G' that are sinks (i.e. in-degree 1 and out-degree 0) and the rest are balanced or isolated.

Beame et al. [4] claim that this construction suffices to show that Imbalance and Excess are equivalent to End-of-Line and Sink respectively. For Excess this is easy to see (see also Papadimitriou [28]). The known vertex with deficiency in G immediately yields a source in G', and finding any sink in G' immediately yields a vertex with excess in G. Note that knowing additional sources is not a problem here, because we are only interested in finding sinks.

However, this construction is not enough to show the equivalence of Imbalance and End-of-Line. Clearly, End-of-Line reduces to Imbalance. Now consider an instance G of Imbalance. First of all, we can make sure that the known unbalanced vertex is in deficiency and not in excess, simply by inverting the direction of all edges in G. However, if we then use the chessplayer algorithm construction, we might obtain multiple known sources in G'. Indeed, if the known unbalanced vertex satisfies in-degree $+k =$ out-degree, then we will obtain k known sources.

Using our results, namely Theorem 1 and Remark 1, we immediately obtain that Imbalance reduces to End-of-Line. However, our results are not trivial to prove (as far as we know), but necessary to show that Imbalance is indeed in PPAD.

4 Looking for Multiple Solutions

In the previous section we considered the case where we are given multiple sources of an End-of-Line graph, but are only looking for a single solution. However, if we are given k distinct sources, then we know that there must exist at least k sinks. Thus, it makes sense to investigate generalisations where we are looking for more than one solution.

Using the construction from the proof of Theorem 1, we immediately obtain a result about a multiple sink problem. With some additional effort, we show that k-source End-of-Line where we are looking for k solutions, is also equivalent to End-of-Line.

4.1 Sink: Looking for Multiple Sinks

Inspired by the results in the previous sections, it is natural to consider similar variations of the problem Sink, which is PPADS-complete [28, 4]. Sink is identical to End-of-Line, except that we only accept a sink as a solution and are not interested in other sources.

Definition 4 (Sink)**.** The Sink problem is defined as: given circuits S, P with n inputs and n outputs and such that $P(0) = 0 \neq S(0)$, find x such that $P(S(x)) \neq x$.

Consider the problem kS-Sink, where we are given a graph and k sources and are looking to find a sink. It is easy to see that this problem is equivalent to Sink. A reduction from Sink to kS-Sink is given by simply taking k copies of the original Sink instance graph. The reduction in the other direction is trivial. Indeed, we can just ignore the extra $k-1$ sources we know, because we are only interested in sinks.

The fact that this kS-Sink problem is not very interesting, motivates us to define a new problem, which seems harder. We are given k sources and have to find k sinks, which are guaranteed to exist.

Definition 5 (k-Sink)**.** Let $k \in \mathbb{N}$. The k-source k-Sink problem, abbreviated k-Sink, is defined as: given circuits S, P with n inputs and n outputs and such that $P(z) = z \neq S(z)$ for all $z < k$, find distinct $x_1, \dots, x_k$ such that $P(S(x_i)) \neq x_i$ for all i.

Using the reduction presented in the proof of Theorem 1, we immediately obtain the following result:

Theorem 2. *For any $k \in \mathbb{N}$, k-Sink is equivalent to* Sink.

Note that this result does not follow from the PPADS-completeness of Excess. k-Sink does not trivially reduce to Excess, because a solution to Excess would only yield a single sink.

Proof. A reduction from Sink to k-Sink is easily obtained by taking k copies of the input graph.

A reduction from k-Sink to Sink is given by the reduction used in the proof of Theorem 1. Note that in the graph G_k, the sinks are exactly the elements in Subsets(V^t, k), i.e. all sets of k distinct sinks of the original graph. As a result, solving the Sink problem on G_k yields k distinct sinks of the original graph, i.e. a solution of the original k-Sink instance. $\square$

Remark 2. Just as noted in Remark 1, this result also holds if there is a polynomial number of sources, in particular if they are given explicitly in the input.

4.2 Looking for Multiple End-of-Line Solutions

The analogous generalisation of End-of-Line would probably be: given k sources, find k distinct vertices that are sinks or other sources. But, we can also consider an even (seemingly) harder problem: given k sources, find k sinks or k other sources. Clearly, the former reduces to the latter. Thus, we will focus on the latter, because reducing it to End-of-Line will immediately give us the result for the former.

Definition 6 (k-EoL)**.** Let $k \in \mathbb{N}$. The k-source k-End-of-Line problem, abbreviated k-EoL, is defined as: given circuits S, P with n inputs and n outputs and such that $P(z) = z \neq S(z)$ for all $z < k$, find distinct $x_1, \dots, x_k$ such that $P(S(x_i)) \neq x_i$ for all i or $S(P(x_i)) \neq x_i \geq k$ for all i.

Note that this problem lies in TFNP, because at least k sinks are guaranteed to exist. Clearly, this problem is at least as hard as End-of-Line, but, at first sight, it is not obvious whether it is actually harder or not. Intuitively, this problem seems harder than

kS-EoL, because we are given the same amount of sources, but are now looking for many more solutions. Fortunately, we are able to show that this problem is equivalent to End-of-Line.

Theorem 3. *For any $k \in \mathbb{N}$, k-EoL is equivalent to* End-of-Line.

Note that it is quite straightforward to reduce k-EoL to End-of-Line if we consider Turing reductions. By Theorem 1 we can efficiently simulate an oracle for ℓS-EoL, $\ell \in \{1, \dots, 2k-1\}$, using an oracle for End-of-Line. Furthermore, we can solve an instance of k-EoL by making repeated calls to ℓS-EoL oracles, where ℓ is the number of currently known sources. If the oracle call returns a new source, then we will use the $(\ell+1)$S-EoL oracle next. If the oracle returns a sink, then we add an edge from this sink to one of the known sources, before making the next oracle call to $(\ell-1)$S-EoL. It is easy to see that after at most $2k$ oracle calls we will have obtained at least k sinks or at least k new sources. While the Turing reduction is pretty simple, finding a many-one reduction is, to the best of our knowledge, much more challenging.

Proof. As before, End-of-Line reduces to k-EoL by taking k copies of the original graph.

It remains to show that k-EoL reduces to End-of-Line. Note that our reduction from kS-EoL to End-of-Line (proof of Theorem 1) is not enough. Even though every source of the constructed graph contains k sources of the original graph, an unknown source of the constructed graph can contain multiple known sources of the original graph. Thus, we need to extend our construction.

Let (S, P) be an instance of k-EoL and let G denote the graph described by (S, P). For any $\ell \in \mathbb{N}$, let G_ℓ denote the corresponding graph constructed in the proof of Theorem 1. Recall the following properties of G_ℓ:

- The sources of G_ℓ are all the sets of the form $\{s_1, \dots, s_\ell\}$, where $s_1, \dots, s_\ell \in V^s$ are distinct sources of the original graph.
- The sinks of G_ℓ are all the sets of the form $\{t_1, \dots, t_\ell\}$, where $t_1, \dots, t_\ell \in V^t$ are distinct sinks of the original graph.
- Every vertex of G_ℓ can be represented using at most $\ell n + \ell^2$ bits. There exists a polynomial algorithm that for each such bit-string decides whether it represents a vertex of G_ℓ or not.
- The successor and predecessor circuits S_ℓ, P_ℓ have polynomial size with respect to $|S| + |P|$ and can be constructed in polynomial time.

Let $\overline{G}_\ell$ denote the reverse graph of G_ℓ, i.e. where all directed edges are reversed in the opposite direction. This is easy to achieve by switching S_ℓ and P_ℓ. For any $\ell \in \{0, 1, 2, \dots, k-1\}$, we define $\tilde{G}_{k+\ell}$ as follows:

- if ℓ is even, then $\tilde{G}_{k+\ell}$ is constructed by taking $\binom{k+\ell-1}{\ell}$ copies of $G_{k+\ell}$
- if ℓ is odd, then $\tilde{G}_{k+\ell}$ is constructed by taking $\binom{k+\ell-1}{\ell}$ copies of $\overline{G}_{k+\ell}$.

The vertices of $\tilde{G}_{k+\ell}$ are of the form $(v, (i_1, \dots, i_\ell))$, where v is a vertex of $G_{k+\ell}$ (or $\overline{G}_{k+\ell}$ if ℓ is odd), and $i_1, \dots, i_\ell \in \{0, \dots, k-1\}$ are such that $i_1 \le i_2 \le \dots \le i_\ell$. Note that there are exactly $\binom{k+\ell-1}{\ell}$ such ℓ-tuples $(i_1, \dots, i_\ell)$[4]. Thus, the tuple uniquely identifies which copy of v this is. Note that in particular, for $\ell = 0$, $\tilde{G}_k$ is simply constructed by taking

[4]Such tuples exactly correspond to ℓ-combinations with repetitions using k possible numbers.

one copy of G_k. Thus, elements of $\tilde{G}_k$ are of the form $(v, ())$, where $()$ denotes the empty tuple.

Now we can finally define the final graph $\hat{G}_k$, which is constructed by taking the (disjoint) union of $\tilde{G}_k, \tilde{G}_{k+1}, \dots, \tilde{G}_{2k-1}$. Any vertex in $\tilde{G}_{k+\ell}$ can be represented using at most $(k+\ell)n+(k+\ell)^2+\ell\log_2 k$ bits. Thus, it follows that any vertex in $\hat{G}_k$ can be represented using a polynomial (in n and k) number of bits. Furthermore, it is easy to construct polynomial size (in $|S|+|P|$) circuits that encode and decode such a bit-string representation (i.e. decide if the given bit-string is a valid vertex and in which part of the graph is is contained).

$\hat{G}_k$ has two kinds of sources and sinks:

- vertices of the form
$$u = (\{t_1, \dots, t_{k+\ell}\}, (i_1, \dots, i_\ell))$$
where $i_1 \leq \dots \leq i_\ell$, $\ell \in \{0, 1, 2, \dots, k-1\}$, and $t_1, \dots, t_{k+\ell}$ are distinct sinks of the original graph G. Note that u is a sink of $\hat{G}_k$ if ℓ is even, and a source if ℓ is odd. In any case, u leads to a valid solution of the original k-EoL instance, because we can extract from it at least k distinct sinks of the original graph in polynomial time.

- vertices of the form
$$w = (\{s_1, \dots, s_{k+\ell}\}, (i_1, \dots, i_\ell))$$
where $i_1 \leq \dots \leq i_\ell$, $\ell \in \{0, 1, 2, \dots, k-1\}$, and $s_1, \dots, s_{k+\ell}$ are distinct sources of the original graph G. Note that w is a source of $\hat{G}_k$ if ℓ is even, and a sink if ℓ is odd. However, unlike the previous case, w does not necessarily lead to a solution of the original k-EoL instance, because it might contain strictly less than k *unknown* sources. If this is the case, then we call w a *bad* source/sink.

The only known source/sink of $\hat{G}_k$ is $(\{0, 1, \dots, k-1\}, ())$ (which comes from $\tilde{G}_k$, that contains a single copy of G_k). Note that this vertex is indeed a source of $\hat{G}_k$ and no other source or sink of this graph is known.

However, there remains one difficulty to overcome. As noted above, some sources/sinks of the second type, namely the so-called bad ones, do not always lead to a solution of the original problem. Thus, we need to make sure that they are no longer solutions, by adding additional edges to $\hat{G}_k$.

For any $\ell \in \{1, 2, \dots, k-1\}$, let $w = (\{s_1, \dots, s_{k+\ell}\}, (i_1, \dots, i_\ell))$, $i_1 \leq \dots \leq i_\ell$, be a source/sink of $\hat{G}_k$, where $s_1, \dots, s_{k+\ell}$ are distinct sources of the original graph G. We can write $\{s_1, \dots, s_{k+\ell}\} = K \cup U$, where $K \subseteq \{0, \dots, k-1\}$ are the known sources and U are the unknown sources (i.e. not in $\{0, \dots, k-1\}$). Note that K and U are disjoint and we have $|K| + |U| = k + \ell$. If $|U| \geq k$, then w leads to a solution of the original problem. Thus, we only need to consider the case where $|U| < k$, which corresponds to the case where w is a bad source/sink.

Let $\overline{K} = \{0, 1, \dots, k-1\} \setminus K$. Also, let $\max \emptyset := -\infty$. We now describe the edges that we introduce.

If $i_\ell > \max \overline{K}$, then we introduce an edge between $w = (K \cup U, (i_1, \dots, i_\ell))$ and $v = ((K \setminus \{i_\ell\}) \cup U, (i_1, \dots, i_{\ell-1}))$. If ℓ is odd, then the edge is directed from w to v. If ℓ is even, then the edge is directed from v to w. Note that in the first case w was a sink and v a source (before adding the edge), whereas in the second the roles are reversed. Thus, the new edge we add always correctly connects a sink to a source.

No other edges are needed. Consider any bad source/sink $w = (K \cup U, (i_1, \dots, i_\ell))$ for some $\ell \in \{1, 2, \dots, k-1\}$. If $i_\ell > \max \overline{K}$, then we have introduced an edge between w and v as seen above. Otherwise, we must have $i_\ell \leq \max \overline{K} =: j$. However, in this case we have introduced an edge between $v' = ((K \cup \{j\}) \cup U, (i_1, \dots, i_\ell, j))$ and w, because $j > \max \overline{K \cup \{j\}}$. Note that v' is a valid vertex of $\hat{G}_k$.

It follows that for $\ell \in \{1, 2, \dots, k-1\}$, all the bad sources/sinks are eliminated. If $\ell = 0$, then $w = (K \cup U, ())$ is connected to $v' = ((K \cup \{j\}) \cup U, (j))$ as above (recall $j = \max \overline{K}$), but only if $|U| \geq 1$. If $|U| = 0$, then $w = (\{0, 1, \dots, k-1\}, ())$, which corresponds to the only known source. Note that this is the only bad source/sink to which we did not add an edge.

It follows that in $\hat{G}_k$ with the new added edges, any End-of-Line solution leads to a solution of the original k-EoL instance. It is easy to check that for any vertex in $\hat{G}_k$, one can compute the predecessor and successor in time polynomial in $|S| + |P|$.

□

Remark 3. Similarly to the points made in Remarks 1 and 2, the reduction also works if k is polynomial in n. In particular, it also works if the sources are given explicitly in the input.

Surprisingly, a natural extension of the construction in the proof of Theorem 3, also yields PPAD-completeness for the following problem. Fix some polynomial p. Consider the following problem: given k sources, find k sinks or $p(n)$ sources. This seems quite surprising, as one might have expected this problem to be closer to PPADS.

The proof sketch is as follows. Instead of considering $\hat{G}_k$ which is the union of $\tilde{G}_k$, $\tilde{G}_{k+1}$, $\dots$, $\tilde{G}_{2k-1}$, we consider $\hat{G}_k^p$ which is the union of $\tilde{G}_k$, $\tilde{G}_{k+1}$, $\dots$, $\tilde{G}_{k+p(n)-1}$. Note that any vertex in this graph can still be represented using a polynomial number of bits (in n and k). The edges of $\hat{G}_k^p$ are constructed using the same "rule" as for $\hat{G}_k$. It follows by inspection that all End-of-Line solutions in $\hat{G}_k^p$ either contain at least k sinks of the original graph, or at least $p(n)$ unknown sources.

5 Concise Mutilated Chessboard Problems

In this section we provide an application of our results. We consider a computational problem inspired from the mutilated chessboard puzzle. PPAD-hardness for the problem is shown using a result of Chen and Deng [7] on 2-dimensional fixed point problems. Inclusion in PPAD is obtained using Theorem 1.

5.1 The Mutilated Chessboard Puzzle

Consider an 8×8 chessboard to be tiled with dominoes. Every domino covers exactly two (horizontally or vertically) adjacent squares of the chessboard. Clearly, we can tile the chessboard using 32 dominoes.

The *Mutilated Chessboard* puzzle is the following: if we mutilate the chessboard by removing two diagonally opposite squares (say the bottom left and the top right one as in Figure 1), can we still tile the remaining chessboard with dominoes?

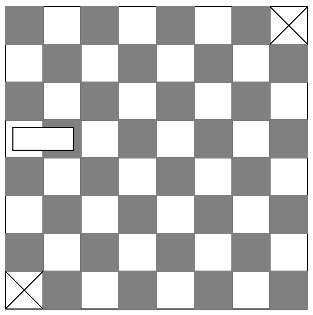


Figure 1: The standard mutilated chessboard and a domino covering two squares.

The answer is that this is impossible. Every domino covers one white square and one black square. However, the mutilation has removed two squares that have the same colour. Thus, it is not possible to tile the remaining chessboard.

According to Robinson [32], this problem was first proposed by Max Black. The problem has been studied from the standpoint of proof complexity. Exponential lower bounds for the length of resolution proofs have been proved [2, 11]. There is also some work in the field of automated reasoning [3, 31], where the problem was first proposed by McCarthy [25].

5.2 The Mutilated Chessboard as a Total Search Problem

In this paper, we consider this problem from the standpoint of computational complexity. The computational problem we define is the following: given an alleged tiling of the mutilated chessboard, find a mistake, i.e. a square that is uncovered or two dominoes overlapping. For this to make sense as a computational problem, the chessboard must have variable size depending on some input parameter n. However, if the chessboard has size $2n \times 2n$, the problem is easy to solve in polynomial time. Thus, we consider the case where the chessboard is exponentially large, i.e. $2^n \times 2^n$, but the alleged tiling is concisely represented using a circuit.

Formalised this way, the problem becomes a total search problem and the subclass-structure of TFNP can be used to classify its computational complexity.

Let $n \in \mathbb{N}$ and $B_n = \{0, \dots, 2^n - 1\} \times \{0, \dots, 2^n - 1\}$.

Consider a boolean circuit C with $2n$ inputs and $2n$ outputs. We can consider this circuit as a function $C : B_n \to B_n$ with the following interpretation: if for some $x \in B_n$, we have $C(C(x)) = x$ and $x - C(x) \in \{(1,0), (0,1), (-1,0), (0,-1)\}$, then there is a domino covering x and $C(x)$. For any subset $S \subseteq B_n$ we say that C yields a valid domino tiling of S, if for all $x \in S$ we have $C(x) \in S$, $C(C(x)) = x$ and $x - C(x) \in \{(1,0), (0,1), (-1,0), (0,-1)\}$.

As seen above, if we take $S = B_n \setminus \{(0,0), (2^n - 1, 2^n - 1)\}$, then there exists no circuit $C : B_n \to B_n$ that yields a valid domino tiling of S.

We now define some computational problems inspired by this mathematical problem.

Definition 7 (2-MC)**.** The problem Mutilated Chessboard with two squares missing, abbreviated 2-MC, is defined as: given a circuit C with $2n$ inputs and $2n$ outputs such that $C(0,0) = (0,0)$ and $C(2^n - 1, 2^n - 1) = (2^n - 1, 2^n - 1)$, find an example that shows that C is not a valid domino tiling of $B_n \setminus \{(0,0),(2^n-1,2^n-1)\}$. Namely, find $x \in B_n \setminus \{(0,0),(2^n-1,2^n-1)\}$ such that $C(C(x)) \neq x$ or $x - C(x) \notin \{(1,0),(0,1),(-1,0),(0,-1)\}$.

Definition 8 (1-MC)**.** The problem Mutilated Chessboard with one square missing, abbreviated 1-MC, is defined as: given a circuit C with $2n$ inputs and $2n$ outputs such that $C(0,0) = (0,0)$, find an example that shows that C is not a valid domino tiling of $B_n \setminus \{(0,0)\}$. Namely, find $x \neq (0,0)$ such that $C(C(x)) \neq x$ or $x - C(x) \notin \{(1,0),(0,1),(-1,0),(0,-1)\}$.

The totality of 2-MC follows from the argument in the previous section. The totality of 1-MC is easy to see, because the chessboard with one square missing has an odd number of squares and is thus impossible to tile with dominoes covering exactly two squares each. Both problems are in FNP, because any solution is easy to check in polynomial time (with respect to the size of the input circuit). It follows that both 2-MC and 1-MC are in TFNP.

A more abstract interpretation of this problem is as follows. The chessboard defines a bipartite graph with the black squares on one side and the white squares on the other side. There is an edge between two squares if one domino can cover them (i.e. the two squares have different colour and are adjacent). The alleged tiling can then be interpreted as yielding an alleged perfect matching on this graph. Since one or two vertices have been removed from one side of the bipartite graph, no perfect matching can exist. This interpretation will be useful to show that the problems lie in PPAD (Theorem 6).

We are now ready to prove that 2-MC and 1-MC are PPAD-complete. The proof is divided in three parts. First, we prove that 1-MC reduces to 2-MC. Then, we prove that 2-MC lies in PPAD. Finally, we prove that 1-MC is PPAD-hard.

Theorem 4. 1-MC *and* 2-MC *are PPAD-complete.*

Proof. Follows from Theorems 5, 6 and 8 (see below). □

5.3 1-MC Reduces to 2-MC

Intuitively, it seems that 1-MC should be at most as hard as (or even easier than) 2-MC, because the proof of 1-MC's totality is slightly less involved than the proof of 2-MC's totality. Indeed, we now show that 1-MC reduces to 2-MC.

Theorem 5. 1-MC *reduces to* 2-MC *in polynomial time.*

Proof. Let $n \in \mathbb{N}$ and $C : B_n \to B_n$ be any instance of 1-MC. Construct $C' : B_{n+1} \to B_{n+1}$ as follows. For any $x \in B_{n+1}$ let

- $C'(x) = C(x)$ if $x \in \{0,\dots,2^n-1\} \times \{0,\dots,2^n-1\}$
- $C'(x) = (2^{n+1}-1, 2^{n+1}-1) - C((2^{n+1}-1, 2^{n+1}-1) - x)$ if $x \in \{2^n,\dots,2^{n+1}-1\} \times \{2^n,\dots,2^{n+1}-1\}$
- $C'(x) = x + (1 - 2(x_1 \mod 2), 0)$ for $x \in \{0,\dots,2^n-1\} \times \{2^n,\dots,2^{n+1}-1\}$ or $x \in \{2^n,\dots,2^{n+1}-1\} \times \{0,\dots,2^n-1\}$ $(x = (x_1,x_2))$

C' is an instance of 2-MC that can be constructed from C in polynomial time (with respect to the size of C). C' yields a valid domino tiling on the two quadrants that are constructed without using C. Note that any solution to the 2-MC problem with instance

C', yields a solution to the 1-MC problem with instance C, because a mistake can only occur in one of the quadrants constructed from C and such a mistake immediately leads to a mistake in C.

An illustration of the proof idea is given in Figure 2. □

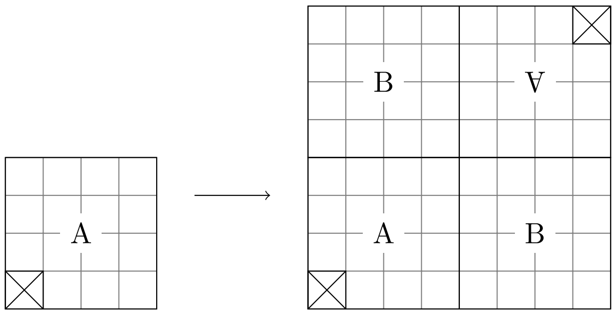


Figure 2: An illustration of the proof idea for Theorem 5. Here A is the original 1-MC instance and B is a fully tiled non-mutilated chessboard.

5.4 2-MC lies in PPAD

In this part we will show that 2-MC lies in PPAD by using our result about 2-source End-of-Line.

Theorem 6. 2-MC *lies in PPAD.*

Proof. Let $n \in \mathbb{N}$ and $C : B_n \to B_n$ be an instance of 2-MC, i.e. $C(0,0) = (0,0)$ and $C(2^n-1, 2^n-1) = (2^n-1, 2^n-1)$. Without loss of generality, we assume that we have $x - C(x) \in \{(1,0),(0,1),(-1,0),(0,-1)\}$ for all $x \in B_n \setminus \{(0,0),(2^n-1,2^n-1)\}$. This can be ensured by modifying the circuit C into C' that satisfies this property and such that any solution of 2-MC with input C' yields a solution of 2-MC with input C.

Partition B_n into $B_E = \{(x_1,x_2) \in B_n : x_1 + x_2 = 0 \mod 2\}$ and $B_O = \{(x_1,x_2) \in B_n : x_1 + x_2 = 1 \mod 2\}$. These subsets correspond to black and white squares on a chessboard. Every valid domino covers one square in B_E and one square in B_O. From the fact that $x - C(x) \in \{(1,0),(0,1),(-1,0),(0,-1)\}$ for all $x \in B_n \setminus \{(0,0),(2^n-1,2^n-1)\}$, it follows that $C(B_O) \subseteq B_E$ and $C(B_E \setminus \{(0,0),(2^n-1,2^n-1)\}) \subseteq B_O$.

Let $g : B_E \to B_O$ be any polynomial-time bijection between B_E and B_O, say $g(x_1,x_2) = (x_1 + 1 \mod 2^n, x_2)$. We define a bipartite graph on vertices (B_O, B_E) by constructing successor and predecessor circuits S, P as follows:

- for any $x \in B_O$, set $S(x) = C(x)$ and $P(x) = g^{-1}(x)$
- for any $x \in B_E$, set $S(x) = g(x)$ and $P(x) = C(x)$

S, P is an instance of 2-source End-of-Line with sources $(0,0)$ and $(2^n-1, 2^n-1)$. We have $P(0,0) = (0,0) \neq S(0,0)$ and $P(2^n-1,2^n-1) = (2^n-1,2^n-1) \neq S(2^n-1,2^n-1)$ by construction. Any 2S-EoL solution yields a 2-MC solution. If $P(S(x)) \neq x$, then it must be that $x \in B_O$ and $C(C(x)) \neq x$. If $S(P(x)) \neq x \notin \{(0,0),(2^n-1,2^n-1)\}$, then it must be that $x \in B_E \setminus \{(0,0),(2^n-1,2^n-1)\}$ and $C(C(x)) \neq x$. It follows that 2-MC reduces to 2S-EoL and thus 2-MC lies in PPAD by Theorem 1. □

5.5 1-MC is PPAD-hard

We will need the following problem for the proof of our result.

Definition 9 (2D-EoL [6])**.** The 2-dimensional End-of-Line problem, abbreviated 2D-EoL, is defined as: given circuits S, P with $2n$ inputs and $2n$ outputs and such that $P(0,0) = (0,0) \neq S(0,0)$, find $x \in B_n$ such that one of the following properties holds

- $P(S(x)) \neq x$
- $S(P(x)) \neq x \neq (0,0)$
- $x - S(x) \notin \{(0,0),(1,0),(0,1),(-1,0),(0,-1)\}$
- $x - P(x) \notin \{(0,0),(1,0),(0,1),(-1,0),(0,-1)\}$.

This problem is the same as the End-of-Line problem, but with the additional constraint that the graph should be embedded in the 2-dimensional grid. Surprisingly, this problem is known to be as hard as End-of-Line.

Theorem 7 ([7, 6])**.** 2D-EoL *is PPAD-complete.*

Proof. 2D-EoL corresponds to a special subset of instances of End-of-Line and thus it is clear that 2D-EoL reduces to End-of-Line. A reduction from End-of-Line to 2D-EoL was provided by Chen and Deng in [7], where they used this result to prove the PPAD-completeness of 2D-Sperner and 2D-Brouwer (they gave a reduction from End-of-Line to a problem called RLEAFD, which is easily seen to be a special case of 2D-EoL). Chen and Deng also state this result more explicitly in [6], where the problem is called 2D-AEL (2-dimensional Another-End-Of-Lines). □

Using this strong result, we can now prove the following:

Theorem 8. 1-MC *is PPAD-hard.*

Proof. Let (S, P) be an instance of 2D-EoL for some $n \in \mathbb{N}$. We now construct an instance of 1-MC. More precisely, we will construct a circuit $C : B_{n+1} \to B_{n+1}$ such that every solution of 1-MC with instance C leads to a solution of the original 2D-EoL instance.

Intuitively, the reduction works as follows. We double the size of the 2D-EoL-grid in both directions and interpret it as a chessboard. This means that every vertex of the original grid now corresponds to 4 squares of the chessboard. Then we construct a domino tiling of this chessboard such that any mistake in the tiling leads to a solution of the original 2D-EoL problem. For any isolated vertex in the 2D-EoL instance, we tile the corresponding 4 squares on the chessboard with two dominoes. Thus, no mistake can occur there. For any vertex of in- and out-degree exactly 1, we determine the predecessor and successor and then tile the 4 squares corresponding to this vertex in such a way that one of the dominoes also covers one of the 4 squares of the predecessor and one other domino also covers one of the 4 squares of the successor. Thus, this yields a valid domino tiling locally. Finally, any 4 squares that correspond to a solution of 2D-EoL are not tiled correctly, i.e. we leave at least one square uncovered. Figure 3 shows a few examples illustrating how a tiling is constructed from a path.

We now give a formal proof of this argument. Let $v = (v_1, v_2) \in B_n$ be any vertex of the original 2D-EoL-grid. The corresponding 4 squares in the chessboard are $u_0(v) = (2v_1, 2v_2)$, $u_1(v) = (2v_1 + 1, 2v_2)$, $u_2(v) = (2v_1, 2v_2 + 1)$ and $u_3(v) = (2v_1 + 1, 2v_2 + 1)$. Assume that we colour in black every square where the sum of coordinates is even and in white every square where the sum of coordinates is odd. Then, we always have that $u_0(v)$ and $u_3(v)$ are black and $u_1(v)$ and $u_2(v)$ are white.

The following procedure shows how to compute C on the squares corresponding to v, i.e. on $u_0(v)$, $u_1(v)$, $u_2(v)$ and $u_3(v)$. For readability we write $u_i := u_i(v)$, i.e. we omit the argument for the squares corresponding to v.

Let $p = P(v)$ and $s = S(v)$.

1. If $v = (0, 0)$, then set $C(u_0) = u_0$ (this corresponds to the missing square).
2. If $p = s = v$, then set $C(u_0) = u_1$, $C(u_1) = u_0$, $C(u_2) = u_3$ and $C(u_3) = u_2$ (this corresponds to an isolated vertex).
3. a. If $v = p + (1, 0)$, then set $C(u_0) = u_1(p)$.
   b. If $v = p + (0, 1)$, then set $C(u_0) = u_2(p)$.
   c. If $v = p + (-1, 0)$, then set $C(u_3) = u_2(p)$.
   d. If $v = p + (0, -1)$, then set $C(u_3) = u_1(p)$.
4. a. If $s = v + (1, 0)$, then set $C(u_1) = u_0(s)$.
   b. If $s = v + (0, 1)$, then set $C(u_2) = u_0(s)$.
   c. If $s = v + (-1, 0)$, then set $C(u_2) = u_3(s)$.
   d. If $s = v + (0, -1)$, then set $C(u_1) = u_3(s)$.
5. Pick $u_i \in \{u_0, u_3\}$ and $u_j \in \{u_1, u_2\}$ such that $C(u_i)$ and $C(u_j)$ are still undefined (this is always possible). Set $C(u_i) = u_j$ and $C(u_j) = u_i$.
6. For all $u_k \in \{u_0, u_1, u_2, u_3\}$ such that $C(u_k)$ is still undefined, set $C(u_k) = u_k$.

Clearly, this procedure handles isolated vertices correctly, because the corresponding 4 squares are covered by 2 dominoes without interfering with any neighbouring squares. Now consider vertices v and w that are adjacent on the 2D-EoL-grid and such that $S(v) = w$ and $P(w) = v$. Then this procedure will place a domino that covers one of the two white squares of v (i.e. $u_1(v)$ or $u_2(v)$) and one of the two black squares of w (i.e. $u_0(w)$ or $u_3(w)$). This means that for any vertex v, the predecessor of v determines which of the two black squares of v is covered and the successor determines which of the two white squares of v is covered. Thus, if v has a valid predecessor and successor, then exactly one white and one black square of v remain uncovered and they can always be covered by one single domino (which is what the procedure does).

If v does not have a valid predecessor and/or successor, then the procedure makes sure that a tiling mistake is introduced. If the predecessor or successor of v is not adjacent on the grid, then at least one square of v will be left uncovered. If an edge is not valid, i.e. $P(S(v)) \neq v$ or $S(P(v)) \neq v$, then one of the squares of v will be such that $C(C(u_i)) \neq u_i$. For the special case of $v = (0, 0)$, we make sure that $C(u_0) = u_0$ and thus there is no tiling mistake in the 4 squares corresponding to $v = (0, 0)$, unless the successor of $(0, 0)$ is not valid, in which case $(0, 0)$ is a solution to the 2D-EoL instance.

It follows that from any solution of the 1-MC instance C, we can efficiently deduce a solution of the 2D-EoL instance (S, P). Furthermore, the construction of C is done in polynomial time with respect to $|S| + |P|$. □

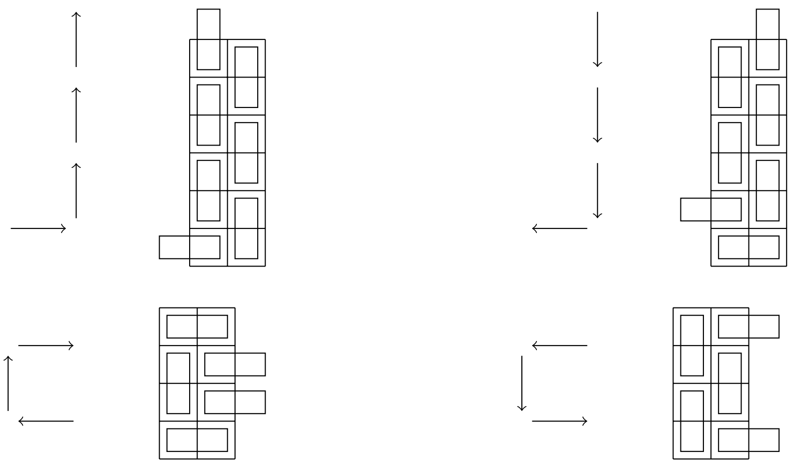


Figure 3: Four examples illustrating how a path is turned into a domino tiling in the proof of Theorem 8.

5.6 Generalising the Mutilated Chessboard Problem

The following is a natural way to generalise the Mutilated Chessboard problem. Recall that $B_n = \{0, \dots, 2^n - 1\} \times \{0, \dots, 2^n - 1\}$ for $n \in \mathbb{N}$. Let $B_n^E = \{(x_1, x_2) \in B_n | x_1 + x_2 = 0 \mod 2\}$ and $B_n^O = \{(x_1, x_2) \in B_n | x_1 + x_2 = 1 \mod 2\}$, which represent the partition of the chessboard according to colour of squares.

Definition 10 (GMC)**.** The Generalised Mutilated Chessboard problem, abbreviated GMC, is defined as: given sets $M_E \subset B_n^E$ and $M_O \subset B_n^O$ with $|M_O| \neq |M_E|$, and a circuit C with $2n$ inputs and outputs such that $C(z) = z$ for all $z \in M_E \cup M_O$, find a mistake in the domino tiling C of $B_n \setminus (M_E \cup M_O)$. Namely, find $x \in B_n \setminus (M_E \cup M_O)$ such that $C(C(x)) \neq x$ or $x - C(x) \notin \{(1, 0), (0, 1), (-1, 0), (0, -1)\}$.

Informally, we are given a chessboard with a different number of black and white squares missing. As a result, the mutilated chessboard is impossible to tile perfectly and the problem lies in TFNP. The sets of missing squares M_E and M_O are given explicitly in the input, so we can assume that their size is polynomial in n (adding this restriction yields a polynomially equivalent problem, because the two versions reduce to each other).

Using our results, we immediately obtain:

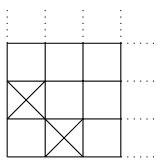
Theorem 9. GMC *is PPAD-complete.*

Proof. PPAD-hardness follows from Theorem 8 and the observation that all 1-MC instances are also GMC instances.

Using the same reduction as in the proof of Theorem 6, one can reduce GMC to a multiple source and multiple sink End-of-Line instance. The arguments given in subsection 3.3 show that the problem is thus in PPAD. □

This result is satisfying, mostly because of the inclusion in PPAD. It shows that any reasonable mutilation of the chessboard will result in a problem that lies in PPAD. However, the PPAD-hardness result is not very satisfying. The hardness of the problem only relies on the 1-MC (or 2-MC) instances and does not give any information about what happens if we remove more squares, or even just squares that do not lie in a corner.

As an illustration consider the following example where two squares are missing:



Clearly, a mistake in any domino tiling of this chessboard is very easy to find, because the square $(0, 0)$ cannot be tiled correctly. Increasing the size of the chessboard does not make the problem harder. Thus, the restriction of GMC where those two squares are missing is polynomial-time solvable. The underlying reason for this is that, in this case, the problem can be shown total by using a local argument involving only the first few squares. Unlike the general case, no parity argument is necessary.

Intuitively, we believe the following holds in general: if the reason for totality is not local, but global, then the problem is PPAD-hard.

As an example, we consider the following restriction of GMC.

Definition 11 (k-Standard MC)**.** Let $k \in \mathbb{N}$. The k-Standard Mutilated Chessboard problem, abbreviated kS-MC, is defined as: given a circuit C with $2n$ inputs and outputs such that $C(z) = z$ for all $z = (2i, 0)$, $i = 0, \dots, k-1$, find a mistake in the domino tiling.

This problem corresponds to the special case where we remove k identically-coloured squares along the bottom edge of the chessboard, starting from the origin. Note that the problem is only well-defined if n is large enough, namely $2^n \geq 2k$. Otherwise, the bottom edge of the chessboard is not long enough to accommodate all the missing squares.

Theorem 10. *For any $k \in \mathbb{N}$, kS-MC is PPAD-complete.*

Proof. Inclusion in PPAD follows from Theorem 9. To show PPAD-hardness we reduce from 1-MC, which is PPAD-hard by Theorem 8.

Let $k, n \in \mathbb{N}$ such that $2^n \geq k$. Consider any 1-MC instance on the chessboard of size $2^n \times 2^n$ with circuit C. Construct an instance of kS-MC on the chessboard of size $2^{2n} \times 2^{2n}$ with circuit C' defined as follows (see Figure 4 for an illustration):

- Include k copies of the 1-MC instance starting from the top-left corner and going towards the bottom-right as follows: the first copy is located in the square $[0, 2^n - 1] \times [2^{2n} - 2^n, 2^{2n} - 1]$, the second copy is in $[2, 2^n + 1] \times [2^{2n} - 2 \cdot 2^n, 2^{2n} - 2^n - 1]$, etc, i.e. the ith copy is in $[2i, 2^n - 1 + 2i] \times [2^{2n} - (i + 1) \cdot 2^n, 2^{2n} - i \cdot 2^n - 1]$ for $i = 0, 1, \dots, k - 1$.
- Using the same technique as in the proof of Theorem 8 create a path from the first missing square (i.e. the origin $(0, 0)$) to the missing square of the first 1-MC instance in the top-left. Note that the path is two squares wide, goes towards the top in a straight line and the last domino of this path will cover the missing square of the 1-MC instance. Create such a path from each of the remaining $k-1$ missing squares

to the corresponding 1-MC instance. All the paths just go straight upwards and thus do not intersect.

- The squares that remain untiled can easily be grouped in clusters of 4 (i.e. squares of side-length 2), because of the way the paths are constructed. Thus, the remaining squares can also be tiled efficiently.

We have thus constructed an instance of kS-MC such that mistakes in the domino tiling can only appear in one of the copies of the 1-MC instance, which immediately yield a mistake in the original 1-MC instance. □

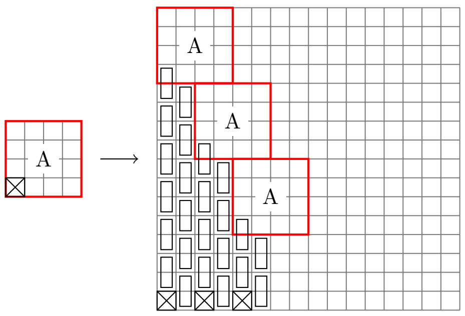


Figure 4: An illustration of the proof idea of Theorem 10 for the case $k = 3$.

Remark 4. Note that the proof still works if k is not a constant, but bounded by some polynomial in n.

The way we defined kS-MC is quite arbitrary. Instead, we could have removed squares along some other edge or along one of the diagonals. In all of these cases, slight modifications of the same proof technique yield PPAD-hardness. We close this section by providing an incomplete characterisation of PPAD-hard special cases of GMC.

Locally Tileable. Consider an instance of GMC on a chessboard of size $2^n \times 2^n$ with m missing squares. Let $M = 2m^2 + 2m - 1$. We say that this instance is *locally tileable* if every rectangular subregion R of the chessboard of size at most $M \times M$ is *internally* tileable, i.e. there exists a domino tiling that completely covers R, but where we allow dominoes overlapping the boundary of R (except on sides of R that are adjacent to the boundary of the chessboard). Note that given such a rectangular region, one can check in polynomial time whether it is internally tileable or not (e.g. by formulating it as a max-flow problem).

Necessary Condition. Being locally tileable is a necessary condition for a GMC instance to be hard. If the instance is not locally tileable, then there exists a polynomial-size rectangular region that must contain a tiling error. Furthermore, such a region can be found in polynomial time in m and n, because it needs to contain or be adjacent to at least one missing square. Thus, instances that are not locally tileable are always easy to solve.

Sufficient Condition. Let RESTRICTEDGMC be any restriction of GMC, where we have restricted which squares can be missing in any instance of size n (but all possible input tilings are allowed). If there exists some $N \in \mathbb{N}$ such that for all $n \geq N$, RestrictedGMC has an instance of size n that is locally tileable (and is easy to find), then RestrictedGMC is PPAD-hard. The proof idea goes as follows. Given any 1-MC instance, find a sufficiently large locally tileable instance of RestrictedGMC. In this instance, find a set of (at most m) rectangles of size at most $M \times M$, such that they contain all missing squares, such that the pairwise distance (in max-norm) between any two rectangles is at least $2m$ and such that every rectangle either touches the boundary of the chessboard or is at distance at least $2m$ away from it. This is guaranteed to exist by the choice of M and can be computed in polynomial time in m and n. Using the fact that each of those rectangles is internally tileable and the techniques presented above, we can then embed the 1-MC instance in an instance of RestrictedGMC. In particular, every domino "sticking out" of the internal tiling of a rectangle will be connected to a copy of the 1-MC instance through a path of dominoes. Note that the value $2m$ is chosen such that the at most m "paths" (of width 2) that connect these "sticking-out" dominoes to the 1-MC instances can easily be constructed without intersection or overlap.

6 Conclusion and Further Work

Our results proving that End-of-Line variants are PPAD-complete show that the definition of PPAD based on End-of-Line is very robust. This robustness of PPAD is also one of the reasons why this class captures the complexity of so many problems. In particular, we note the perhaps surprising result that if we are given one source and seek one sink or n^{100} other sources, then the problem is still in PPAD. Our results on PPAD and PPADS also provide further evidence suggesting that the number of (most likely) non-trivial interesting subclasses of TFNP is rather small. Indeed, despite extensive investigation of problems in TFNP in the last 20 years, most of them seem to end up being complete for one of the already known subclasses. This is also the case for the new computational problem we introduce in this paper, the concise mutilated chessboard, which, it turns out, is PPAD-complete.

In terms of future work, we believe it would be interesting to extend our End-of-Line results to a similar problem: End-of-Potential-Line. This problem, introduced in [16], considers an End-of-Line instance augmented with a *potential* that assigns a value to every vertex. The potential is required to increase when we follow a directed edge. Solutions of this problem correspond to any End-of-Line solution or any "mistake" in the potential. This problem was used to define the class EOPL [16], a subclass of CLS [13], which contains the P-LCP problem in particular. A small modification of our construction in the proof of Theorem 1 can be used to show that multiple-source End-of-Potential-Line is equivalent to End-of-Potential-Line. However, new ideas seem necessary to extend these results to the case where multiple solutions are sought.

Another research direction is looking at End-of-Line variants where we have a super-polynomial number of given sources. In this case the sources would have to be provided implicitly, e.g. by saying that all vertices smaller than some super-polynomial value $k(n) \in [2^n]$ are sources. To solve the problem one would have to provide a new source, a sink, or a vertex in $[k]$ that is actually not a source. If the number of sources is a polynomial fraction of the total number of vertices, then the problem can be solved efficiently using a straightforward randomised algorithm. However, there remains a gap between those two extreme cases, where the complexity of the problem is not known.

References

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, pages 781–793, 2004.

[2] M. Alekhnovich. Mutilated chessboard problem is exponentially hard for resolution. *Theoretical Computer Science*, 310(1-3):513–525, 2004.

[3] G. Bancerek. The mutilated chessboard problem-checked by Mizar. In *The QED Workshop II*, pages 25–26, 1995.

[4] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences*, 57(1):3–19, 1998.

[5] X. Chen, D. Dai, Y. Du, and S. Teng. Settling the complexity of Arrow-Debreu equilibria in markets with additively separable utilities. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 273–282, 2009.

[6] X. Chen and X. Deng. Recent development in computational complexity characterization of Nash equilibrium. *Computer Science Review*, 1(2):88–99, 2007.

[7] X. Chen and X. Deng. On the complexity of 2d discrete fixed point problem. *Theoretical Computer Science*, 410(44):4448–4456, 2009.

[8] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3):1–57, 2009.

[9] X. Chen, D. Paparas, and M. Yannakakis. The complexity of non-monotone markets. *Journal of the ACM*, 64(3):20:1–20:56, June 2017.

[10] B. Codenotti, A. Saberi, K. Varadarajan, and Y. Ye. The complexity of equilibria: Hardness results for economies via a correspondence with games. *Theor. Comput. Sci.*, 408(2-3):188–198, 2008.

[11] S. Dantchev and S. Riis. “Planar” tautologies hard for resolution. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 220–229. IEEE, 2001.

[12] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[13] C. Daskalakis and C. H. Papadimitriou. Continuous local search. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 790–804. Society for Industrial and Applied Mathematics, 2011.

[14] D. Dumrauf, B. Monien, and K. Tiemann. Multiprocessor scheduling is PLS-complete. In *HICSS'09. 42nd Hawaii International Conference on System Sciences*, pages 1–10. IEEE, 2009.

[15] A. Fabrikant, C. H. Papadimitriou, and K. Talwar. The complexity of pure Nash equilibria. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 604–612. ACM, 2004.

[16] J. Fearnley, S. Gordon, R. Mehta, and R. Savani. End of potential line. *arXiv preprint arXiv:1804.03450*, 2018.

[17] A. Filos-Ratsikas and P. W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. *arXiv preprint arXiv:1805.12559*, 2018.

[18] A. Filos-Ratsikas and P. W. Goldberg. Consensus halving is PPA-complete. In *50th Annual Symposium on the Theory of Computing (STOC)*, pages 51–64. ACM, 2018.

[19] P. W. Goldberg and C. H. Papadimitriou. TFNP: An update. In *International Conference on Algorithms and Complexity*, pages 3–9. Springer, 2017.

[20] P. W. Goldberg and C. H. Papadimitriou. Towards a unified complexity theory of total functions. *Journal of Computer and System Sciences*, 94:167–192, 2018. Journal of Computer and System Science: 50 years of celebration. In memory of Professor Edward Blum.

[21] E. Jeřábek. Integer factoring and modular square roots. *Journal of Computer and System Sciences*, 82(2):380–394, 2016.

[22] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988.

[23] S. Kintali, L. J. Poplawski, R. Rajaraman, R. Sundaram, and S. Teng. Reducibility among fractional stability problems. *SIAM J. Comput.*, 42(6):2063–2113, 2013.

[24] M. Krentel. Structure in locally optimal solutions. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 216–221. IEEE, 1989.

[25] J. McCarthy. *A tough nut for proof procedures.* Stanford University Stanford, CA, USA, 1964.

[26] N. Megiddo and C. H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991.

[27] T. Morioka. Classification of search problems and their definability in bounded arithmetic. Master's thesis, University of Toronto. National Library of Canada, Bibliothèque nationale du Canada, 2001.

[28] C. H. Papadimitriou. On graph-theoretic lemmata and complexity classes. In *31st Annual IEEE Symposium on Foundations of Computer Science*, pages 794–801. IEEE, 1990.

[29] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

[30] C. H. Papadimitriou, A. A. Schäffer, and M. Yannakakis. On the complexity of local search. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 438–445. ACM, 1990.

[31] L. C. Paulson. A simple formalization and proof for the mutilated chess board. *Logic Journal of IGPL*, 9(3):475–485, 2001.

[32] J. A. Robinson. Formal and informal proofs. In *Automated Reasoning*, pages 267–282. Springer, 1991.