Electronic Colloquium on Computational Complexity, Report No. 63 (2016)

Hardness of Continuous Local Search: Query Complexity and Cryptographic Lower Bounds

Pavel Hubáček* Eylon Yogev*

Abstract

Local search proved to be an extremely useful tool when facing hard optimization problems (e.g., via the simplex algorithm, simulated annealing, or genetic algorithms). Although powerful, it has its limitations: there are functions for which exponentially many queries are needed to find a local optimum. In many contexts the optimization problem is defined by a continuous function, which might offer an advantage when performing the local search. This leads us to study the following natural question: How hard is *continuous* local search? The computational complexity of such search problems is captured by the complexity class **CLS** which is contained in the intersection of **PLS** and **PPAD**, two important subclasses of **TFNP** (the class of **NP** search problems with a guaranteed solution).

In this work, we show the first hardness results for **CLS** (the smallest non-trivial class among the currently defined subclasses of **TFNP**). Our hardness results are in terms of black-box (where only oracle access to the function is given) and white-box (where the function is represented succinctly by a circuit). In the black-box case, we show instances for which any (computationally unbounded) randomized algorithm must perform exponentially many queries in order to find a local optimum. In the white-box case, we show hardness for computationally bounded algorithms under cryptographic assumptions.

As our main technical contribution we introduce a new total search problem which we call End-of-Metered-Line. The special structure of this problem enables us to: (1) show that End-of-Metered-Line is contained in **CLS**, and (2) prove hardness for it both in the black-box and the white-box setting.

*Weizmann Institute of Science, Israel. Email: {pavel.hubacek,eylon.yogev}@weizmann.ac.il. Supported in part by a grant from the I-CORE Program of the Planning and Budgeting Committee, the Israel Science Foundation, BSF and the Israeli Ministry of Science and Technology.

ISSN 1433-8092

Contents

1 Introduction

Local search is a widely applicable tool when facing hard optimization problems. In local search we seek only for a local optimum rather than insisting on a global one. More formally, an instance of the problem is given by two functions, a neighborhood function that maps each point to a set of its local neighbors and a valuation function that gives each point a real value. The goal is to find a point for which none of its neighbors have a (strictly) greater value. Many of the most popular optimization algorithms apply local search; the simplex algorithm, genetic algorithms, steepest ascent hill climbing, and simulated annealing, to name a few.

Although powerful, local search has its limitations. It is known that there exist instances for which finding a local optimum might take exponential time. A natural approach to avoid the shortcomings of local search is to exploit some additional structure of the optimization problem at hand. In particular, both the neighborhood function and the valuation function are often *continuous*, which might offer advantage when trying to solve such *continuous local search* problems. Motivated by these considerations, we ask the following natural question:

*How hard is **continuous** local search?*

There are two common types of hardness results: white-box and black-box. In the white-box setting, the problem at hand is given explicitly and in a succinct manner (e.g., as a polynomially sized circuit). Then, hardness is shown via a reduction from a different problem for which hardness is already established (or assumed). In the black-box setting (also known as the query model, or the decision tree model) the problem is represented via oracle access, and the complexity is measured by the number of performed oracle queries. In this model, it is often possible to prove unconditional lower bounds on the number of queries required to find a solution. In this paper, we show hardness of *continuous* local search in terms of *both white-box and black-box*.

Optimization problems amenable to local search, as well as continuous local search, fall into a vast category of problems for which a solution is guaranteed to exist. Indeed, there is a myriad of important natural problems with a straightforward proof certifying the existence of a solution. One simple example is finding a collision in a (compressing) hash function, where a collision is guaranteed by the pigeonhole principle. Another example is the standard game theoretical notion of equilibrium due to Nash [Nas50]. The existence of a Nash equilibrium is guaranteed in any finite strategic game, whereas the problem of finding one is eluding the algorithmic game theory community with no known general efficient algorithm. Motivated by some of the above observations, Megiddo and Papadimitriou [MP91] proposed the complexity class **TFNP** (for Total Function Nondeterministic Polynomial-time), which is the class of all **NP** search problems with a guaranteed solution. Papadimitriou [Pap94] subsequently introduced a taxonomy of the problems inside **TFNP** that gathers the problems into classes based on the type of combinatorial argument establishing their totality.

The problem of finding a local optimum (not necessarily in the continuous setting) defines its own subclass of **TFNP** called **PLS** (for Polynomial Local Search [JPY88]). The canonical problem for **PLS**, called Local-Search, is given by a pair of circuits N and V that assign to any n-bit string x a polynomial sized neighborhood $\mathsf{N}(x)$ and a non-negative integer value $\mathsf{V}(x)$. The goal is to find a local optimum, i.e., an x such that $\mathsf{V}(x) \geq \mathsf{V}(x')$ for all $x' \in \mathsf{N}(x)$. The totality of this problem is guaranteed as a consequence of the observation that every finite directed acyclic graph has a sink. As for hardness, a series of exponential query complexity lower bounds

for Local-Search [Ald83, LTT89, Aar06, Zha09] established black-box hardness for **PLS** for deterministic, randomized and even quantum algorithms.

What about the problem of finding a local optimum in the *continuous* setting? Daskalakis and Papadimitriou [DP11] introduced the class **CLS** (for Continuous Local Search) to capture the exact computational complexity of continuous local search. The canonical problem for **CLS** is called Continuous-Local-Optimum (CLOpt). Unlike Local-Search, which is a discrete problem over the Boolean hypercube $\{0,1\}^n$, an instance of Continuous-Local-Optimum is given by two functions f and p over the unit cube $[0,1]^3$ which are assumed to be λ-Lipschitz continuous.[1] The function f maps the unit cube to itself and the function p assigns to every point of the unit cube a real value. The goal is to find an ε-approximate local optimum of p with respect to f (i.e., a point x such that $p(f(x)) - p(x) \leq \varepsilon$) or two points that violate the λ-Lipschitz continuity of either f or p. We note that there is no known *combinatorial* complete problem for **CLS** and finding one remains an interesting open problem. Nevertheless, Daskalakis and Papadimitriou showed that **CLS** lies in the intersection of **PLS** and another important subclass of **TFNP** called **PPAD**.

The class **PPAD** (for Polynomial Parity Argument on Directed graphs) has received particular attention in part due to the fact that one of its complete problems is finding Nash equilibra in strategic games [DGP09, CDT09]. The canonical problem for **PPAD**, called End-of-Line (EOL), is given by a successor circuit S and a predecessor circuit P that both map n-bit strings to n-bit strings, and thus implicitly define a graph with 2^n vertices of degree at most two (there is an edge between u and v iff $u = \mathsf{P}(v)$ and $v = \mathsf{S}(u)$). Given a source in this graph, the goal is to find a sink or an alternative source. The totality of this problem can be established based on the handshaking lemma.[2] Similarly to **PLS**, also for **PPAD** there are known black-box hardness results, e.g., exponential query complexity lower bounds for finding Brouwer fixed points (another important complete problem for **PPAD**) [HPV89, Bab14]. Recently, Bitanski, Paneth and Rosen [BPR15] constructed hard instances of End-of-Line under cryptographic assumptions and showed the first white-box hardness result for **PPAD** (the cryptographic assumptions were improved in [GPS15]). An overview of the above subclasses of **TFNP** and the known hardness results is depicted in Figure 1 (see Section 3 for the formal definitions of the various subclasses of **TFNP**).

Given that there are oracle separations showing that in the relativized setting both **PLS** is not reducible to **PPAD** (cf. [BM04]) and **PPAD** is not reducible to **PLS** (cf. [Mor01]), and since **CLS** is contained in the intersection of **PLS** and **PPAD**, it follows that **CLS** is a proper subclass of both **PLS** and **PPAD**. This suggests that continuous local search might be an easier problem than the problems of finding approximate local optima or approximate Brouwer fixed points. Moreover, unlike in the case of **PPAD** and **PLS**, no hardness result is known for **CLS**; neither in terms of query complexity (even in the random oracle model), nor under any cryptographic assumption (including even strong notions of obfuscation used to establish the known white-box hardness results for **PPAD**).

In this work we show the first hardness results for **CLS**. We stress that among the various subclasses currently defined in **TFNP**, the smallest non-trivial[3] subclass is **CLS**. Specifically, we prove an exponential query complexity lower bound, and give hardness from several cryptographic assumptions used in previous works for showing cryptographic hardness for **PPAD**.

[1] A function f is λ-Lipschitz continuous if for any x, x' it holds that $|x - x'| \leq \lambda |f(x) - f(x')|$.

[2] The handshaking lemma states that every finite undirected graph has an even number of vertices with odd degree.

[3] The notion of being "non-trivial" refers to a class that is not known to be solvable in polynomial-time.

1.1 Our Results

In order to study hardness of continuous local search, we introduce a new total search problem we call End-of-Metered-Line (EOML), which is similar in spirit to the End-of-Line problem. An instance of End-of-Metered-Line is given by three circuits S, P, and V. The circuits S and P act exactly like the successor and the predecessor in the End-of-Line problem, i.e., they implicitly define a graph in which every vertex has degree at most two. The goal also remains the same, i.e., given that 0^n is a source, we ask to find a sink or a source different from the trivial one at 0^n. Since 0^n is a source, it follows from the handshaking lemma that there exists a sink with respect to S and P, and thus End-of-Metered-Line is a total problem.

The additional circuit V is intended to aid in solving this task by providing extra information about the structure of the implicit graph. Specifically, V acts as an odometer for the line starting at the initial source 0^n, that is, it reveals how many steps of S it takes to reach any vertex x from 0^n.[4] Recall that the existence of a solution for problems inside **TFNP** is ensured syntactically and not by a promise on the instances. In particular, we cannot make any promise on the behavior of V. Though, we cannot efficiently verify that V indeed answers as expected for every vertex. Therefore, we enforce that the addition of circuit V aids in finding a solution by letting any vertex attesting that V deviates from its expected behavior to constitute an additional solution. First, the initial source 0^n must be the only vertex with value 1. Second, any vertex x with a non-zero value i must be succeeded with a vertex with value $i+1$ and preceded with a vertex with value $i-1$. Any vertex violating either of these two conditions is defined to be a solution. The formal definition (cf. Definition 4.1) together with additional discussion is given in Section 4.

Our first (and the most technical) result shows that the valuation circuit puts End-of-Metered-Line in a much smaller class than the End-of-Line problem. We show that End-of-Metered-Line is reducible to Continuous-Local-Optimum (see Definition 3.5), the canonical problem defining the class **CLS**.

Theorem 5.1. End-of-Metered-Line *is contained in* **CLS**.

Next, we show that continuous local search remains hard by proving hardness for End-of-Metered-Line. Our hardness results are in terms of both black-box and white-box. In the black-box case, we show an exponential query complexity lower bound for End-of-Metered-Line.

Theorem 6.1. *The randomized query complexity of* End-of-Metered-Line *is* $\Omega\left(2^{n/2}/\sqrt{n}\right)$.

By combining Theorem 6.1 with Theorem 5.1, we get as a corollary the black-box hardness of **CLS**:

Corollary 1.1. *In the query model, any randomized algorithm solving the* Continuous-Local-Optimum *problem up to n-digits of precision must perform at least $2^{\Omega(n)}$ queries.*

In the white-box case, where the functions are given via circuits of polynomial size in n, we use cryptographic assumptions, specifically, the notion of secure circuit obfuscation. Our results extend the recent result of Bitanski et al. [BPR15] and Garg et al. [GPS15], who constructed distributions of hard instances of End-of-Line based on indistinguishability obfuscation. We establish that similarly to the End-of-Line, under these cryptographic assumptions, there is a distribution of hard instances for End-of-Metered-Line.

[4] In other words, the circuit V serves the same purpose as milestones placed along the roads throughout the Roman Empire; it reassures the traveler that the correct path is being followed and indicates the number of steps taken from the ultimate origin at the *Milliarium Aureum*, the golden milestone at the Forum Romanum (in our case at 0^n).

Theorem 6.4. *Assume there exist one-way permutations and indistinguishability obfuscation for* **P/Poly**. *Then the* End-of-Metered-Line *problem is hard for polynomial-time algorithms.*

Since our reduction in Theorem 5.1 is polynomial-time computable, we get the following corollary:

Corollary 1.2. *Assume there exist one-way permutations and indistinguishability obfuscation for* **P/Poly**. *Then the* **CLS** *class is hard for polynomial-time algorithms.*

An overview of our results in the context of **TFNP** is depicted in Figure 1.

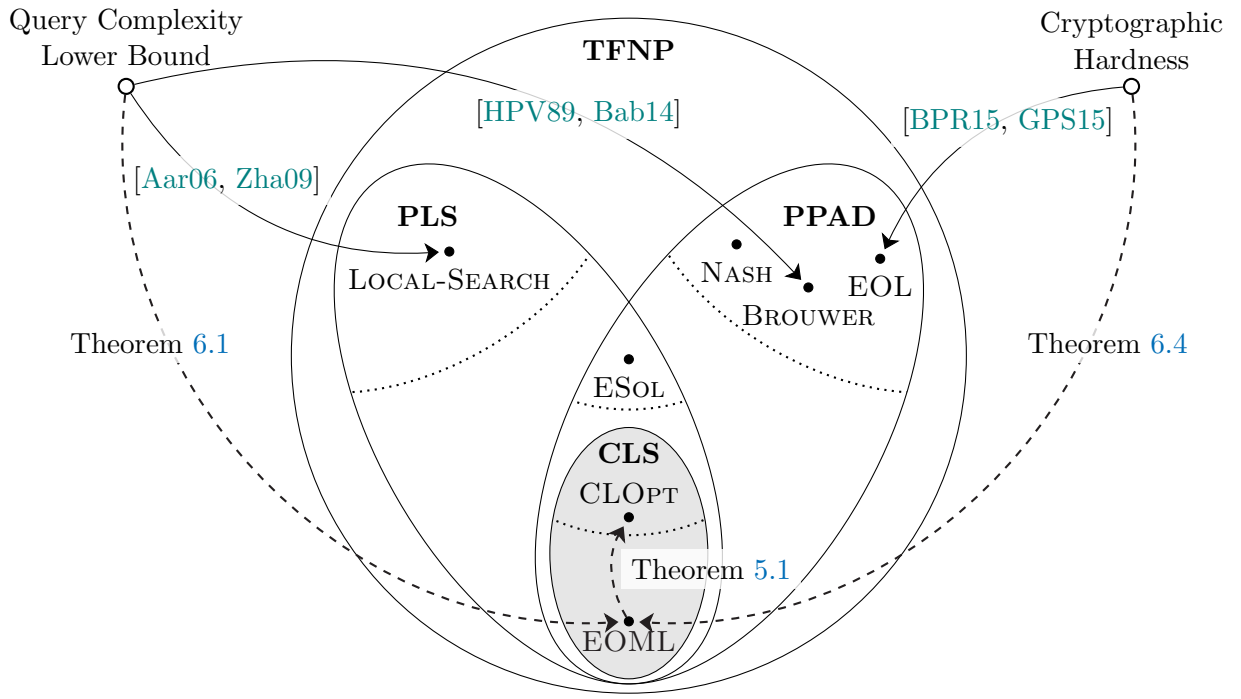


Figure 1: Depiction of our and previous results in the context of **TFNP**. Problems known to be complete for the respective classes are drawn above a dotted line.

2 Our Techniques

In this section we give a high-level description of the main techniques used to establish our results. Formal definitions and complete proofs are provided in the subsequent sections.

2.1 Reducing End-of-Metered-Line to Continuous-Local-Optimum

The core of our hardness results for continuous local search is a reduction showing that End-of-Metered-Line is contained in the class **CLS**. Our reduction takes any instance $(\mathsf{S}, \mathsf{P}, \mathsf{V})$ of End-of-Metered-Line over the domain $\{0,1\}^n$ and builds an instance of Continuous-Local-Optimum consisting of a function f mapping the unit cube to itself, a function p assigning a real value to any point in the unit cube and two constants $\varepsilon, \lambda > 0$, where f and p are λ-Lipschitz, and

such that any local optimum (i.e., a point x such that $p(f(x)) - p(x) \leq \varepsilon$) corresponds to a solution to $(\mathsf{S}, \mathsf{P}, \mathsf{V})$.

Any instance $(\mathsf{S}, \mathsf{P}, \mathsf{V})$ implicitly defines a graph over $\{0,1\}^n$. The graph is defined similarly to the graph corresponding to any End-of-Line instance, i.e., there is an edge between u and v iff $u = \mathsf{P}(v)$ and $v = \mathsf{S}(u)$, and additionally, we discard all vertices with value 0, and all edges $(x, \mathsf{S}(x))$ such that $\mathsf{V}(\mathsf{S}(x)) - \mathsf{V}(x) \neq 1$ (i.e., all the edges where V does not increment correctly). Notice that there are no cycles in this graph since it is impossible to assign incremental values consistently along a cycle. We show how to embed this discrete graph defined by $(\mathsf{S}, \mathsf{P}, \mathsf{V})$ into a continuous function defined over the unit square.[5]

Embedding a Single Line. For simplicity, we begin by considering the graph to be a single line: $0^n \to \mathsf{S}(0^n) \to \mathsf{S}^2(0^n) \to \cdots$. In [CD09], Chen and Deng showed how to embed any End-of-Line instance into a continuous function over the unit square, an instance of the Brouwer problem. That is, they constructed a function f over the unit square such that any (approximate) fixed point of f corresponds to a solution to the End-of-Line instance. Their reduction works by considering a discrete grid on the unit square, and showing how to embed the End-of-Line graph as a directed path onto the lines of this grid. On the grid points, the function f is defined such that it points towards the direction of the path at the points lying on the path, and such that f points towards $(0,0)$ at any point lying off the path. For any intermediate point (off the grid), f is defined by interpolation of its four neighbors on the grid. The resulting function f creates a flow along the path towards its end and a flow towards $(0,0)$ in the remaining area of the unit square.

The main difficulty when extending the above reduction to Continuous-Local-Optimum (instead of to Brouwer) is to construct the additional continuous valuation function p. Ultimately, we would want to define p on the grid (and by interpolation off the grid) to match the behavior of f, and such that any local optimum with respect to f and p corresponds to a solution to the End-of-Metered-Line instance. Specifically, we need to assign increasing values along the different flows defined by f in a locally and efficiently computable way. We achieve this as follows. To each point off the path, we assign a value that corresponds to its distance from the point $(0,0)$. For each point lying on the path, we use V to assign a value corresponding to its distance from the beginning of the path.

The hope is to find appropriate values for p that introduce no local optima except at a fixed point. However, it turns out that we cannot simply apply this idea on top of the reduction of [CD09]. In fact, one can show that any set of values for p on the grid will introduce a new local optimum that does not correspond to a solution for $(\mathsf{S}, \mathsf{P}, \mathsf{V})$. The problem is that there are points on the path where f points in the opposite direction than at a nearby point off the path. The interpolation between such two points then, in the combination with the p values, introduces a local optimum at points near the path (but possibly far from its end). Actually, this issue would introduce exponentially many new local optima from which it is not possible to extract a solution.

In order to circumvent introducing these local optima, we modify the above reduction. We ensure that the change in direction of f is proportional to the change in value of p. This is done by carefully altering the definition of f to create an additional transition layer between the path

[5] In order to match the definition of Continuous-Local-Optimum, we should construct functions over the unit cube and not over the unit square. However, it is easy to extend any Brouwer function f mapping the unit square to itself to a continuous function over the unit cube by simply copying over the third coordinate (i.e., for all $(x, y, z) \in [0,1]^3$, define $f'(x, y, z) = (f(x, y), z)$).

and its surrounding area without introducing new local optima. This enables us to define values for p that do not introduce unnecessary local optima along *most* parts of the path. In particular, if the path traverses in only two directions (say up and right), then no unnecessary local optimum is introduced at all. However, the path in the above reduction of Chen and Deng traverses the unit square in all four directions.

The Staircase Embedding. The natural step to take, at this point, is to change the reduction such that the path traverses the unit square only in two directions, resulting in a path that resembles a "staircase". Note that, in general, it is not possible to embed an End-of-Line instance in such a staircase pattern, as it would show that **PPAD** $\subseteq$ **CLS** (which is unlikely due to known oracle separations [BM04, Mor01]). Indeed, our reduction makes extensive use of the valuation circuit V to create a staircase pattern from the given End-of-Metered-Line graph.

To achieve this, we route the staircase through the unit square so that the coordinates of every point on the staircase encode a vertex in the EOML graph. In particular, we split the unit square into square sub-blocks so that we can identify each block along the side of the square with one of the 2^n non-zero values given out by V. Then, we do the same for each sub-block so that we can identify the small squares along the bottom edge of each sub-block with strings in $\{0,1\}^n$. Thus, in the end we have split the unit square into 2^{4n} small squares. Finally, for every vertex x such that $V(x) = i$ we create a path connecting the point $(x, 0)$ in block (i, i) with the point $(\mathsf{S}(x), 0)$ in the block $(i+1, i+1)$. The points are connected via three line segments going through (x, x) in block (i, i) and $(S(x), x)$ in block $(i+1, i)$ (see Figure 2a). Note that for each line segment, it is possible to identify only from the coordinates of a point whether the line passes through it. For example to test that a point (z_1, z_2) in block (i, i) lies on the first line segment, we check that $\mathsf{V}(z_1) = i$ and that $z_2 \leq z_1$. Testing for the other two line segments is performed similarly.

Now that we have defined the function f so that it results in an embedding of the path as a staircase in the unit square, we can assign the p values to the grid points. To avoid local optima except for endpoints of the path, we assign incremental values for all grid points on the path. That is, for any point on the path, its value is the distance from the beginning of the path. Note that we need to be able to compute this distance locally (so that it can be efficiently computed by the valuation circuit V). Here, we exploit again the structure of our staircase embedding: for any point on the staircase, it holds that the distance from the beginning is exactly its Manhattan distance from $(0, 0)$ (i.e., the sum of its coordinates).

Given the modified version of f that results in a staircase embedding, and the above definition of p, it is readily possible to prove that, in the special case where the End-of-Metered-Line graph is a single line, the only local optimum is exactly at the end of the staircase. Thus, by our construction of the embedding, the coordinates of the unique local optimum can be used to extract a solution to the original End-of-Metered-Line instance $(\mathsf{S}, \mathsf{P}, \mathsf{V})$. We give an example of an embedding of a single line in Figure 5. Next, we show how to handle general graphs which might correspond to more than a single line.

Handling General Graphs. Consider the case where the graph is not a single line, but a collection of lines. The embedding of such a graph will result in a collection of staircases. The problem is that these staircases might intersect, and any such intersection would introduce a new local optimum. Since the intersection could lie at an arbitrary point on two staircases, it is in general not possible to extract from the local optimum a solution to the End-of-Metered-Line

instance. Thus, we use an idea from [CD09] and locally alter the flow of the staircases at any crossing point in a way that eliminates the intersection on one hand and preserves the staircase structure on the other. For any crossing of a horizontal line segment with a vertical line segment, we disconnect the lines so that they do not touch and switch their directions so that afterwards the horizontal line turns up and the vertical line turns right (see Figure 2b). Note that even though this transformation alters the topological structure of the staircases in a major way, it preserves the endpoints of the original staircases and does not introduce any new ones.

In the most general case, an arbitrary End-of-Metered-Line graph can contain a collision of two lines, that is, two lines that at some point merge to one. Notice that this merging point is a solution for $(\mathsf{S}, \mathsf{P}, \mathsf{V})$. In terms of our embedding, any such collision will introduce a local optimum from which we can extract the solution. The full details of the reduction, its proof, and complete discussion of how to extract solutions from any local optimum are given in Section 5. In Figure 6, we give an example of an embedding of the lines near a crossover after the modification. In Figure 7, we give an example of the an embedding of a graph with multiple lines and collisions into the unit square.

2.2 Query Complexity Lower Bound for End-of-Metered-Line

Our query complexity lower bound builds on the techniques used for proving black-box hardness of **PLS**, and in particular, on query complexity lower bounds for the Local-Search problem over the Boolean hypercube. Our starting point is the tight randomized query complexity lower bound of $\Theta(2^{n/2}\sqrt{n})$ for Local-Search on the Boolean hypercube by Zhang [Zha09]. Specifically, Zhang [Zha09] showed how to construct a distribution of self-avoiding random paths over the n-dimensional hypercube, such that any randomized algorithm that finds the endpoint of the path given only oracle access to the path (i.e., it can learn whether a vertex lies on the path or not) must query exponentially many vertices of the hypercube. We show how to exploit the specific structure of Zhang's construction to build a distribution of End-of-Metered-Line instances that require exponential query complexity.

In particular, the label of every vertex $v \in \{0,1\}^n$ on the path constructed in [Zha09] can be parsed as a pair $(x, z) \in \{0,1\}^m \times \{0,1\}^{n-m}$, where the x component is used to perform a random walk and the z component stores a step-counter for the random walk to avoid intersections of the path with itself. Our EOML oracle defines a graph corresponding to a single line chosen according to the above structured random walk, and we use the step-counter to assign incremental values along the path. The oracle answers for every query hitting the path with the respective predecessor vertex, successor vertex, and the distance from the origin of the random walk. We show that any query to the End-of-Metered-Line instance can be implemented by at most n queries to the path of Zhang. Hence, we get an $\Omega(2^{n/2}/\sqrt{n})$ randomized query complexity lower bound for End-of-Metered-Line. See Section 6.1 for a detailed proof.

2.3 Cryptographic Hardness for End-of-Metered-Line

Our cryptographic hardness result for End-of-Metered-Line builds upon previous works on cryptographic hardness for **PPAD**. Bitanski, Paneth and Rosen [BPR15] showed hardness of End-of-Line under the assumption of existence of injective one-way functions and indistinguishability obfuscation (both with sub-exponential security).

A cryptographic obfuscator is a complier that transforms any given circuit to a "scrambled" one, which is functionally equivalent on one hand, but hides its implementation details on the other. The theoretical study of obfuscation was initiated by Barak et al. [BGI+12], which suggested the notion of virtual black-box obfuscation: anything that can be efficiently computed from the obfuscated circuit, can be also computed efficiently from black-box access to the circuit (see Definition D.3). Their main result was that this notion of obfuscation cannot be generally achieved.

As a way to bypass their general impossibility result, they introduced a plausibly weaker notion of indistinguishability obfuscation. An indistinguishability obfuscator is a compiler that guarantees that if two circuits compute the same function, then their obfuscations are computationally indistinguishable (see Definition D.2). Furthermore, since the first candidate construction of indistinguishability obfuscation [GGH+13] was proposed, many other constructions have followed suite [PST14, GLSW14, AB15, BVWW16].

Abbot, Kane and Valiant [AKV04] used the power of secure obfuscation to prove a hardness result for End-of-Line (the canonical complete problem for **PPAD**) assuming virtual black-box obfuscation. However, given that virtual black-box obfuscation is not achievable in general, it remained an obvious important open problem to show hardness for **PPAD** under weaker cryptographic assumptions. Bitanski et al. [BPR15] were the first to solve this open problem. They showed that, assuming one-way functions (see Definition D.1) and indistuinguishability obfuscation (both with subexponential security), there exist hard instance of the End-of-Line problem. Their proof followed two main steps. First, they defined a problem called Sink-of-Verifiable-Line (motivated by the work of Abbott, Kane and Valiant [AKV04]), and they showed that Sink-of-Verifiable-Line is hard under the above cryptographic assumptions. Second, they gave a reduction from Sink-of-Verifiable-Line to End-of-Line yielding the conclusion that End-of-Line is hard under the same assumptions.

Subsequently, Garg et al. [GPS15] showed that the cryptographic assumption for the hardness of Sink-of-Verifiable-Line can be weakened. In particular, they showed hardness of Sink-of-Verifiable-Line under the assumption of existence of one-way permutations and indistinguishability obfuscation both with *polynomial security*, and furthermore they showed an alternative construction of hard instances of Sink-of-Verifiable-Line assuming compact functional encryption (a special kind of public-key encryption which supports restricted secret keys that enable a key holder to learn a specific function of the encrypted data, but learn nothing else), which is a plausibly weaker cryptographic primitive than indistinguishability obfuscation. Using the reduction to End-of-Line of [AKV04, BPR15] their result implies cryptographic hardness for **PPAD**.

Our white-box hardness result for End-of-Metered-Line is achieved by modifying the reduction from Sink-of-Verifiable-Line to End-of-Line to get a reduction from Sink-of-Verifiable-Line to End-of-Metered-Line. This allows us to use any hardness result for Sink-of-Verifiable-Line (i.e., either [BPR15] or[GPS15]) and get a corresponding hardness for End-of-Metered-Line.

An instance of the Sink-of-Verifiable-Line problem consists of a successor circuit S, a target index $1 \leq T \leq 2^n$ and a verification circuit V with the promise the $V(x,i) = 1$ if and only if $\mathsf{S}^{i-1}(0^n) = x$. The goal is to find an x such that $V(x,T) = 1$. In order to reduce any Sink-of-Verifiable-Line instance $(\mathsf{S}, \mathsf{V}, T)$ to an equivalent instance $(\mathsf{S}', \mathsf{P}')$ of End-of-Line, we need to implement the predecessor circuit. [BPR15] use the approach suggested in [AKV04] to make the steps of the successor circuit reversible (leveraging the ideas presented in [Ben89]) which can be performed using the successor circuit S and the verification circuit V given in the Sink-

of-Verifiable-Line instance. We show that the resulting instance $(\mathsf{S}', \mathsf{P}')$ does not come at the expense of the verification circuit V. We extend the reduction of [AKV04, BPR15] with a valuation circuit V' that correctly assigns incremental values along the path starting at the initial source, resulting with a valid instance $(\mathsf{S}', \mathsf{P}', \mathsf{V}')$ of End-of-Metered-Line.

Our work clarifies the implications of the recent hardness results for Sink-of-Verifiable-Line based on cryptographic notions of software obfuscation. In particular, we show that the cryptographic hardness for Sink-of-Verifiable-Line of [GPS15] based on indistinguishability obfuscation and one-way permutations (or compact functional encryption) implies hardness for problems inside the class **CLS**, currently one of the lowest known non-trivial classes inside **TFNP**. Since End-of-Metered-Line is contained in **CLS** which lies in the intersection of **PLS** and **PPAD**, we also get a hardness results of **PLS** for which there were previously no known hardness results based on cryptographic assumptions. See Section 6.2 for the complete construction and the proof.

2.4 Organization of the Paper

The rest of the paper is organized as follows. In Section 3 we give the formal definitions and an overview of related work on complexity of total search problems and their classes. In Section 4 we define the End-of-Metered-Line problem and discuss some of its basic properties. In Section 5 we describe our reduction from End-of-Metered-Line to Continuous-Local-Optimum. In Section 6.1 we show a query complexity lower bound for End-of-Metered-Line. In Section 6.2 we describe a cryptographic hardness result for End-of-Metered-Line.

3 Total Search Problems

Systematic study of total search problems was initiated by Papadimitriou and Megiddo [MP91] who pointed out that the functional analogue of $\mathbf{NP} \cap \mathbf{coNP}$ contains a host of non-trivial problems for which a solution always exists.

Definition 3.1 (Total search problems)**.** *A* search problem $\mathcal{S}$ *is a set of inputs* $I_{\mathcal{S}} \subseteq \Sigma^*$ *on some alphabet* Σ *such that for each* $x \in I_{\mathcal{S}}$ *there is an associated set of solutions* $\mathcal{S}_x \subseteq \Sigma^{|x|^k}$ *for some integer* k*, such that for each* $x \in I_{\mathcal{S}}$ *and* $y \in \Sigma^{|x|^k}$ *it is decidable in polynomial time whether* $y \in \mathcal{S}_x$*. A search problem is* total *if* $\mathcal{S}_x \neq \emptyset$ *for all* $x \in I_{\mathcal{S}}$*. The set of all total search problems is denoted* **TFNP***.*

A *polynomial-time reduction* from total search problem $\mathcal{S}$ to total search problem $\mathcal{T}$ is a pair f, g of polynomial-time computable functions such that, for every input x of $\mathcal{S}$, $f(x)$ is an input of $\mathcal{T}$, and furthermore for every $y \in \mathcal{T}_{f(x)}$, $g(y) \in \mathcal{S}_x$.

Since **TFNP** is a "semantic class", it is unlikely to contain any natural complete problems. Papadimitriou [Pap94] defined various "syntactic" subclasses of **TFNP** with important complete problems based on combinatorial principles used to show their totality.

The class **PPAD** (for Polynomial Parity Arguments on Directed graphs) is defined as all the total search problems reducible to the End-of-Line problem.

Definition 3.2 (End-of-Line)**.** *Given two circuits* $\mathsf{S}, \mathsf{P} : \{0,1\}^n \to \{0,1\}^n$*, such that* $\mathsf{P}(0^n) = 0^n \neq \mathsf{S}(0^n)$*, find a string* $x \in \{0,1\}^n$ *such that* $\mathsf{P}(\mathsf{S}(x)) \neq x$ *or* $\mathsf{S}(\mathsf{P}(x)) \neq x \neq 0^n$*.*

The main appeal of the class **PPAD** is that it has many important complete problems related to algorithmic game theory, such as finding a Nash equilibrium in bimatrix games [DGP09, CDT09] or in constant degree graphical games [Rub15]. The black-box hardness of problems related to **PPAD** was extensively studied, and we point the interested reader to the works mentioned in Section 1 and the following works [SS06, CD08, FISV09] for additional results.

Johnson, Papadimitriou and Yannakakis [JPY88] defined and studied a subclass of **TFNP** that captures the complexity of local search problems. The class **PLS** (for Polynomial Local Search) is defined as all the total search problems reducible to the Local-Search problem. In the following we denote by $[n]$ the set of numbers $\{1, 2, \ldots, n\}$:

Definition 3.3 (Local-Search)**.** *Given two circuits* $\mathsf{S}\colon \{0,1\}^n \to \{0,1\}^n$ *and* $\mathsf{V}\colon \{0,1\}^n \to [2^n] \cup \{0\}$*, find a string* $x \in \{0,1\}^n$ *such that* $\mathsf{V}(\mathsf{S}(x)) \leq \mathsf{V}(x)$*.*

Note that the above definition is equivalent to an alternative definition where the problem is given by circuits $\mathsf{N}\colon \{0,1\}^n \to \{0,1\}^{p(n)}$ and $\mathsf{V}\colon \{0,1\}^n \to [2^n] \cup \{0\}$ such that for each vertex $x \in \{0,1\}^n$, the circuit N outputs a polynomial number $p(n)$ of x's neighbors. First, every instance (S, V) can be viewed as an instance (N, V) of the second type with a neighborhood function $\mathsf{N} = \mathsf{S}$ outputting only singleton neighborhoods. Second, any instance (N, V) can be transformed into an equivalent "hill-climbing" instance (S, V) by making the circuit S output a neighbor with the highest value under V.

Additional black-box hardness results for local search can be found in [SS09, SY09]. The relation between the query complexity of local search and approximate fixed point computation was studied by Chen and Teng [CT07].

Continuous Local Search. Even though the Local-Search problem and the End-of-Line problem that define **PLS** and **PPAD** respectively are discrete and presented in terms of Boolean circuits, both classes have equivalent definitions in terms of continuous functions over the unit cube. The notion of continuity used in this context is the *Lipschitz continuity*.

Definition 3.4. *A function* $f\colon \mathbb{R}^n \to \mathbb{R}^m$ *is* λ*-Lipschitz for some* $\lambda > 0$ *if for all* $x, x' \in \mathbb{R}^n$ *it holds that* $|f(x) - f(x')|_\infty \leq \lambda \cdot |x - x'|_\infty$*, where* $|\cdot|_\infty$ *denotes the maximum norm.*

The continuous complete problem for **PPAD** is finding a Brouwer fixed point. An instance is given by an $\varepsilon > 0$ and a λ-Lipschitz function f mapping the unit cube to itself. The goal is to find an ε-approximate fixed point of f, i.e., some $x \in [0,1]^3$ such that $|f(x) - x| < \varepsilon$. The continuous complete problem for **PLS** is finding a local optimum in a continuous valuation function. An instance is given by a function f mapping the unit cube to itself (not necessarily Lipschitz) and a λ-Lipschitz function p assigning a real value between zero and one to every point of the unit cube. The goal is to find an ε-approximate local optimum of p under f, i.e., a point $x \in [0,1]^3$ such that $p(f(x)) - p(x) < \varepsilon$.

Daskalakis and Papadimitriou [DP11] suggested to combine the above two definitions in a synergetic way and defined the class **CLS** (for Continuous Local Search) to contain all the total search problems reducible to the following Continuous-Local-Optimum problem, where both the transition and the valuation functions are λ-Lipshitz.

Definition 3.5 (Continuous-Local-Optimum)**.** *Given two arithmetic circuits* $f\colon [0,1]^3 \to [0,1]^3$ *and* $p\colon [0,1]^3 \to [0,1]$*, and two real constants* $\varepsilon, \lambda > 0$*, find either a point* $x \in [0,1]^3$ *such that* $p(f(x)) \leq p(x) + \varepsilon$ *or a pair of points* $x, x' \in [0,1]^3$ *certifying that either* p *or* f *is not* λ*-Lipschitz.*

We already discussed that the class **CLS** lies in the intersection of **PLS** and **PPAD**. Moreover, as shown in [DP11], it contains many important computational problems related to algorithmic game theory with no known polynomial time algorithm, such as finding Nash equilibria in congestion games [Ros73], finding Brouwer fixed points of contraction maps [Ban22], or solving simple stochastic games of Shapley and Condon [Sha53, Con92].

4 End-of-Metered-Line

In this section we define a new total search problem, which we call End-of-Metered-Line, that will serve as a basis of our hardness results for **CLS**. Our new problem extends End-of-Line, given by a successor circuit S and a predecessor circuit P, by including an additional valuation circuit V. The purpose of the valuation circuit is to act as an odometer for line starting at the initial source 0^n, that is, it reveals how many steps of S it takes to reach any vertex x from 0^n. In particular, we would like to have the promise that $\mathsf{V}(x) = i$ if and only if $\mathsf{S}^{i-1}(0^n) = x$. Of course, it is not possible to efficiently verify that indeed V answers as expected for every vertex. Moreover, **TFNP** (see Definition 3.1) is a class of problems where a solution must exist, without additional promises. Thus, we introduce a compromise between certifying correctness and having a promise: We make no promises on the behavior of V but let any vertex attesting that V deviates from its expected behavior to be an additional solution. The formal definition is given below.

Definition 4.1 (End-of-Metered-Line)**.** *Given circuits* $\mathsf{S}, \mathsf{P} \colon \{0,1\}^n \to \{0,1\}^n$*, and* $\mathsf{V} \colon \{0,1\}^n \to [2^n] \cup \{0\}$ *such that* $\mathsf{P}(0^n) = 0^n \neq \mathsf{S}(0^n)$ *and* $\mathsf{V}(0^n) = 1$*, find a string* $x \in \{0,1\}^n$ *satisfying one of the following:*

1. *either* $\mathsf{P}(\mathsf{S}(x)) \neq x$ *or* $\mathsf{S}(\mathsf{P}(x)) \neq x \neq 0^n$*,*
2. $x \neq 0^n$ *and* $\mathsf{V}(x) = 1$*,*
3. *either* $\mathsf{V}(x) > 0$ *and* $\mathsf{V}(\mathsf{S}(x)) - \mathsf{V}(x) \neq 1$ *or* $\mathsf{V}(x) > 1$ *and* $\mathsf{V}(x) - \mathsf{V}(\mathsf{P}(x)) \neq 1$*.*

Given an End-of-Metered-Line instance $(\mathsf{S}, \mathsf{P}, \mathsf{V})$, we say that a vertex x is a solution of the first, the second, or the third type if it satisfies the corresponding item in the above definition. Notice that solutions of the *first type* are also solutions to the End-of-Line instance (S, P) defined by the successor and the predecessor circuit of the End-of-Metered-Line instance. The solutions of the second and the third type correspond to some indication that V is "untruthful", that is, V is not answering according to the distance of the vertex from 0^n. In particular, solutions of the *second type* indicate that V is giving values to a different line other than the one starting from 0^n. Solutions of the *third type* indicate that V is reporting a "miscount" at some vertex x.

The Intuition Behind our Definition. There are several ways how to define the "expected behavior" of V, where for each we would obtain a possibly different set of solutions, and thus a variation of the problem which, in general, is not equivalent to Definition 4.1. In our definition, we require V to give additional information only for vertices on the line, and thus we allow V to set the zero value to vertices off the line starting at 0^n. That is, a vertex x such that $\mathsf{V}(x) = 0$ and $\mathsf{V}(\mathsf{S}(x)) = 0$ is *not* considered as a solution. In particular, V can set the zero value to all vertices on any cycle without introducing new solutions. This property is crucial for our hardness results (cf. Section 6.1 and Section 6.2). In both cases, we show hardness by a reduction from

a certain problem to End-of-Metered-Line. To maintain the hardness, it is crucial for the valuation functions constructed in our reductions that we are able to assign the zero value without introducing trivial solutions.

Moreover, we do not require V to distinguish whether a vertex lies on the line or not. Consider two vertices $x \neq x'$ such that $\mathsf{V}(x) = \mathsf{V}(x') \neq 0$. It is clear that both cannot be at the same distance from 0^n, and thus they are a witness of V deviating from its expected behavior. However, we do not consider such pairs to be a solution, as there is no way to distinguish which vertex is not on the line originating at 0^n. It is possible to add an additional type of solution corresponding to such x and x'. We note that all of our results naturally extend to this variant of End-of-Metered-Line.

Warm-Up: EOML lies in PPAD ∩ PLS. We show that End-of-Metered-Line lies at the intersection of **PLS** (see Definition 3.3) and **PPAD** (see Definition 3.2). As pointed out by Daskalakis and Papadimitriou [DP11], proving that a problem lies in the intersection of **PLS** and **PPAD** is equivalent to reducing the problem both to a complete problem in **PLS** and to a complete problem in **PPAD**. They showed that Either-Solution, i.e., given an instance of Local-Search and an instance of End-of-Line to find a solution to either one of them, is a complete problem for **PLS** ∩ **PPAD**. We have already shown that End-of-Metered-Line is contained in **PPAD** since for any instance $(\mathsf{S}, \mathsf{P}, \mathsf{V})$ of End-of-Metered-Line, the circuits (S, P) constitute an instance of End-of-Line. Additionally, End-of-Metered-Line can also be reduced to Local-Search, which together with the above straightforward reduction to End-of-Line completes the reduction to Either-Solution. Thus, End-of-Metered-Line is contained in **PLS** ∩ **PPAD**.

Lemma 4.2. End-of-Metered-Line *is reducible to* Local-Search.

Proof. Given an End-of-Metered-Line instance $(\mathsf{S}, \mathsf{P}, \mathsf{V})$ we define a Local-Search instance $(\mathsf{S}', \mathsf{V})$, where $\mathsf{S}'(x)$ outputs 0^n if $\mathsf{V}(x) = 0$ and otherwise outputs $\mathsf{S}(x)$. Let x be any solution to $(\mathsf{S}', \mathsf{V})$. Note that $\mathsf{V}(x) \neq 0$, since every vertex x such that $\mathsf{V}(x) = 0$ is under S' followed by 0^n and it cannot be a local optimum (recall that $\mathsf{V}(0^n) = 1$). Moreover, $\mathsf{V}(x) \geq \mathsf{V}(\mathsf{S}'(x)) = \mathsf{V}(\mathsf{S}(x))$ which implies that $\mathsf{V}(x) \neq \mathsf{V}(\mathsf{S}(x)) - 1$. Hence, x is a solution of the third type to the original End-of-Metered-Line instance $(\mathsf{S}, \mathsf{P}, \mathsf{V})$. □

5 End-of-Metered-Line is in CLS

In this section we show that End-of-Metered-Line is contained in **CLS** (see Section 3). In particular, we give a reduction from any instance $(\mathsf{S}, \mathsf{P}, \mathsf{V})$ of End-of-Metered-Line (cf. Definition 4.1) to an equivalent instance $(f, p, \varepsilon, \lambda)$ of Continuous-Local-Optimum (cf. Definition 3.5). Our reduction takes the discrete graph implicitly defined by any End-of-Metered-Line instance and embeds it inside the unit square in a continuous manner. In particular, our embedding defines the function f for the new Continuous-Local-Optimum instance that resembles a staircase (or a collection of staircases in case the valuation circuit V outputs non-zero values on more than one line). Additionally, we define a continuous valuation function p for all the points of the unit square with the property that from any local optimum of p under f it is possible to reconstruct a solution to the original EOML instance.

Theorem 5.1. End-of-Metered-Line *is reducible to* Continuous-Local-Optimum.

Our reduction takes any instance $(\mathsf{S}, \mathsf{P}, \mathsf{V})$ of End-of-Metered-Line and produces an instance of Continuous-Local-Optimum, defined by functions $f\colon [0,1]^3 \to [0,1]^3$, and $p\colon [0,1]^3 \to [0,1]$ and constants $\varepsilon, \lambda > 0$. We begin by describing the functions $f\colon [0,L]^2 \to [0,L]^2$ and $p\colon [0,L]^2 \to [0,4L]$, where $L \in \mathbb{N}$ will be defined later. Afterwards, we show how to convert f and p to the appropriate domains and we define the constants ε and λ.

Consider a tessellation of the domain $[0,L]^2$ with $L \cdot L$ unit squares. Similarly to some previous works [HPV89, Rub15], we call the border area of the domain the frame, and the area inside the frame the picture. We set the frame to be of width 5 unit squares. That is, the picture covers exactly the area $[5, L-5]^2$. The picture is comprised of square blocks, where the side of each block is of length $10 \cdot 2^n$ unit squares. The picture contains 2^n such blocks. Thus, we get that $L = 10(2^{2n}+1)$, where $10 \cdot 2^{2n}$ is the number of squares along the side of the picture and 10 is added to account for the width of the frame on both sides of the picture.

The coordinates of any point $(X, Y) \in [5, L-5]^2$ within the picture can be parsed as a pair of triplets: $X = (b_x, s_x, x)$ and $Y = (b_y, s_y, y)$. First, $(b_x, b_y) \in [2^n] \times [2^n]$ are the coordinates of the square block containing (X, Y). Second, $(s_x, s_y) \in [10 \cdot 2^n] \times [10 \cdot 2^n]$ are the coordinates of the unit square within this block. Third, $(x, y) \in [0,1]^2$ are the coordinates of (X, Y) within this square. Formally, we have that

$$X = 5 + (10 \cdot 2^n)(b_x - 1) + (s_x - 1) + x$$

and

$$Y = 5 + (10 \cdot 2^n)(b_y - 1) + (s_y - 1) + y.$$

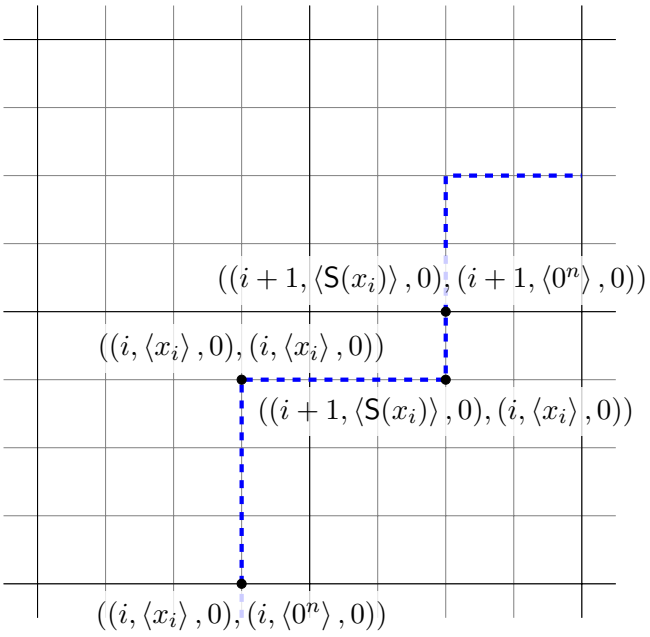
We embed the line defined by the End-of-Metered-Line instance into the picture. The line starts at the bottom left corner of the picture and traverses the borders of the unit squares inside the picture in a staircase-like pattern. Below we use a mapping from $\{0,1\}^n$ to the set $[10 \cdot 2^n]$, and for any $x = (x_n, \ldots, x_1) \in \{0,1\}^n$ denote by $\langle x \rangle = 10\left(\sum_{i=1}^{n} x_i 2^{i-1}\right) + 1$ its corresponding value (however, when clear from the context we simply write x). The line passes through points $((i, \langle x_i \rangle, 0), (i, \langle 0^n \rangle, 0))$ for all $(x_i, i) \in \{0,1\}^n \times [2^n]$ such that $\mathsf{V}(x_i) = i$. We connect all pairs of points $((i, \langle x_i \rangle, 0), (i, \langle 0^n \rangle, 0))$ and $((i+1, \langle \mathsf{S}(x_i) \rangle, 0), (i+1, \langle 0^n \rangle, 0))$ via three line segments:

$$\begin{aligned}
&((i, \langle x_i \rangle, 0), (i, \langle 0^n \rangle, 0)) \longleftrightarrow \\
&((i, \langle x_i \rangle, 0), (i, \langle x_i \rangle, 0)) \longleftrightarrow \\
&((i+1, \langle \mathsf{S}(x_i) \rangle, 0), (i, \langle x_i \rangle, 0)) \longleftrightarrow \\
&((i+1, \langle \mathsf{S}(x_i) \rangle, 0), (i+1, \langle 0^n \rangle, 0)).
\end{aligned}$$

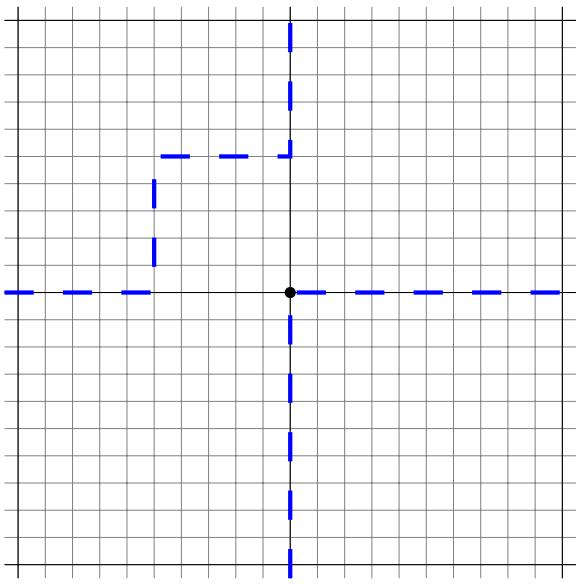
See Figure 2a for an illustration.

Note that in general it could be the case that $\mathsf{V}(x) = \mathsf{V}(x') = i$ for some $x \neq x'$ and there are crossing line segments inside the blocks (i, i) and $(i+1, i)$. To avoid such crossing of lines, we alter the behavior of the line in the vicinity of a crossing point as illustrated in Figure 2b. In particular, the neighborhood of any crossing point is changed according to a fixed crossing gadget inspired by the work of Chen and Deng [CD09]. Even though the lines are altered after applying the crossover section, the most important property of this transformation is that it neither removes existing ends of line nor it introduces additional ones.

The function f is defined to create a "flow" along the above staircase. Formally, the function f is defined by a different displacement at each point. Each unit square inside $[0,L]^2$ is given a "color" corresponding to the displacement at the center of the square. For any other point z, the displacement is defined as the Cartesian interpolation of the f-values of the four centers surrounding



(a) A staircase segment connecting points representing x_i and $\mathsf{S}(x_i)$.



(b) The fixed template for avoiding crossing at a point that lies on two lines.

Figure 2: Details of our embedding of an End-of-Metered-Line instance into the unit square.

z. That is, f is of the form $f(z) = z + \delta d(z)$ where $\delta > 0$ is some constant and $d(z) \in \mathbb{R}^2$ is the displacement function, which we describe next.

There are five basic displacements (depicted in Figure 3): blue $(-1, -1)$, green $(0, -1)$, purple $(-1, 0)$, yellow $(1, 0)$, and red $(0, 1)$. By default the color of each square is blue, which corresponds

(a) blue: $(-1, -1)$ (b) green: $(0, -1)$ (c) purple: $(-1, 0)$ (d) yellow: $(1, 0)$ (e) red: $(0, 1)$

Figure 3: The five basic displacements.

to the displacement $(-1, -1)$ (i.e., pointing down and left). The squares on the frame are colored according to a fixed assignment of colors. First, the squares in the leftmost column get yellow color and the squares in the second left column get the green color. Second, the squares in the bottom row get red color and the square in the row above it get purple color. Then, we color the bottom left corner of the frame according to Figure 4. Finally, we color the squares inside and below the picture corresponding to their distance from the line. Any square touching the line from left or above is colored yellow. Any square touching the line from right or below is colored red. Any square that is distance one from the line from left or above is colored green. Any square that is distance one from the line from right or below is colored purple.

The function $p : [0, L]^2 \to [0, 4L]$ is defined similarly to f. Each square is assigned an integer value corresponding to the value of p at the middle of the square, and the value of p at any other point is the Cartesian interpolation of the p-values at the four centers surrounding it. We number

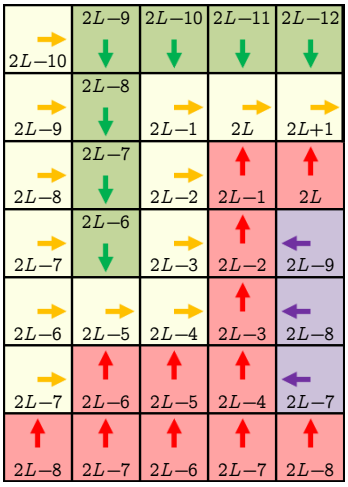


Figure 4: The f-values and the p-values of the squares close to the origin. The bottom left square is aligned at the bottom left corner of the frame.

every unit square in the tiling of $[0, L]^2$ by coordinates in $[L] \times [L]$, where the bottom left unit square gets coordinates $(1, 1)$ and the top left unit square gets coordinates (L, L). We assign the p-values to each square according to its color under f. Every square $(i, j) \in [L] \times [L]$ that is green, blue or purple gets value $2L - i - j$ (that is, the top right square gets value 0 and the bottom left square gets value $2L - 2$). Now we assign the p-values of the yellow and red squares. First, any square $(1, j)$ in the leftmost column gets value $2L - j - 2$. Second, any square $(i, 1)$ in the bottom row gets value $2L - i - 2$. Then, we assign the p-values in the bottom left corner of the frame according to Figure 4.

To assign the values to the remaining yellow and red squares, we begin by assigning a value corresponding to each step on the line inside the picture. The start of the line gets value $M = 2L+1$, and after performing ℓ steps of unit length the value is $M + \ell$. Note that all the remaining yellow and red squares are adjacent to the line. If the square is below the line (i.e., it has red color) then it gets the value of its top left corner. If the square is above the line (i.e., it has yellow color) then it gets the value of its bottom right corner. For concrete example of a specific embedding with the corresponding values for f and p see Figure 5.

We note that the evaluation of f and p on any point can be performed locally. In particular, the values follow a simple fixed pattern on the frame. The values inside the picture are computed based on the proximity of the staircase that embeds the End-of-Metered-Line instance. Note that coordinates of any point on the staircase are sufficient to verify that the point indeed lies on the line. We give the complete description of procedure that decides whether a point lies on the staircase in Algorithm 4. Given this procedure it is easy to decide whether a square touches the staircase from above or below. We can simply check whether the square touches the staircase with its bottom right corner or its top left corner, which splits the squares adjacent to the staircase to those that lie above and those that lie below (see Algorithm 5). Knowing that a square lies above or below the line allows us to immediately assign the yellow and red colors. Otherwise, we test the neighbours of the square for being above or below to assign the green and purple color; in case all of these tests fail, the square gets the default blue color (see Algorithm 6).

The p-values are easy to compute based on the f-colors. Note that the value of any blue, green,

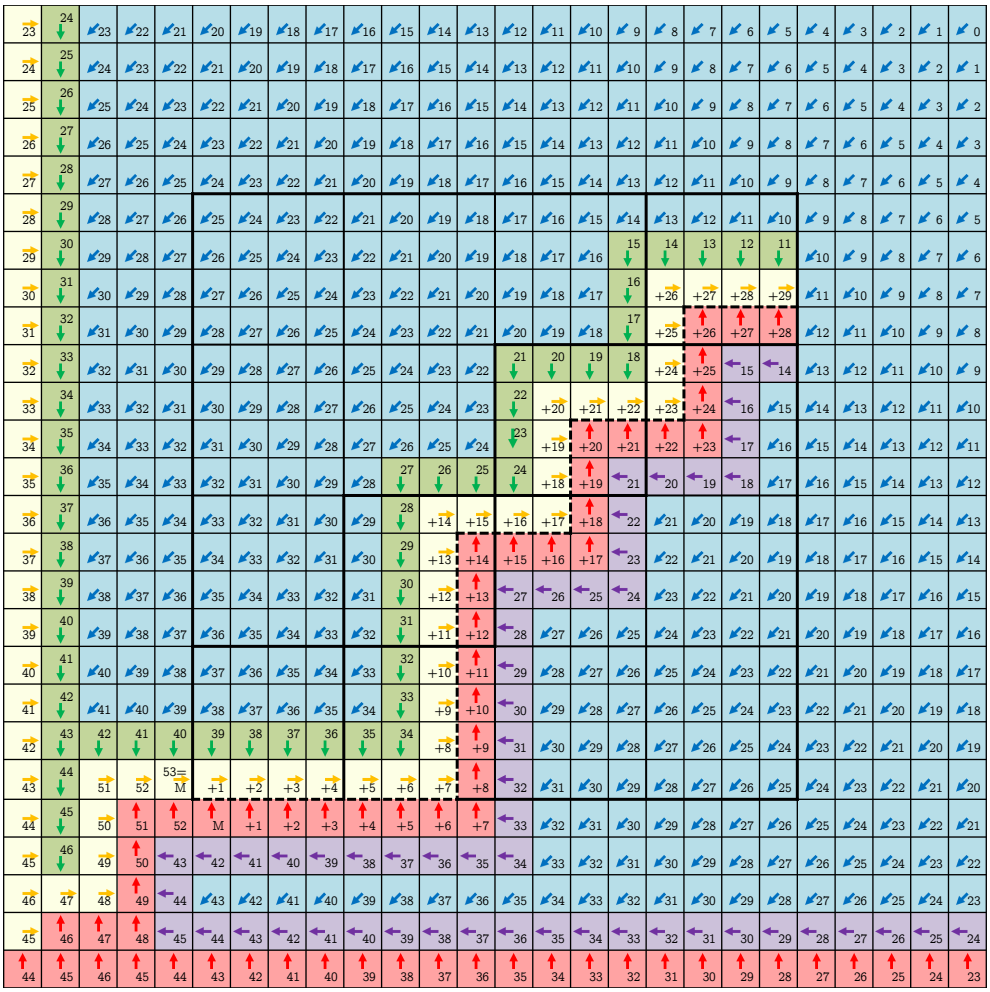


Figure 5: A simplified depiction (omitting the ten unit square buffer inside the blocks) of an instance of Continuous-Local-Optimum obtained by applying our reduction on an instance of End-of-Metered-Line. The dashed line is the staircase corresponding to embedding a single line $00 \to 11 \to 10 \to 01$ implicitly defined by the successor circuit S of the End-of-Metered-Line instance. The arrows indicate the displacement at the center of each square and the numbers indicate the corresponding p-values ("$+j$" is a shorthand for "$M + j$").

or purple square depends only on its Manhattan distance d from the bottom right corner of the picture, i.e., the value is $2L - 10 - d$. Moreover, the value of any yellow or red square depends only on the number of steps from the base of the staircase up to that square. Hence, it is independent of the exact shape of the staircase and it can also be expressed in terms of the Manhattan distance from the bottom left corner of the picture, i.e., the value is $M - 1 + d$ (see Algorithm 7).

Finally, we set the parameter according to the construction. The actual values are determined by the analysis in the proof. In terms of correctness, we only need to verify that the parameter can be described in $\mathsf{poly}(n)$ bits. Concretely, we set $\lambda = 2^{3n}$, $\delta = \frac{1}{10\lambda n}$, and $\varepsilon = \frac{\delta}{10}$ (and recall that $L = 10(2^{2n} + 1)$).

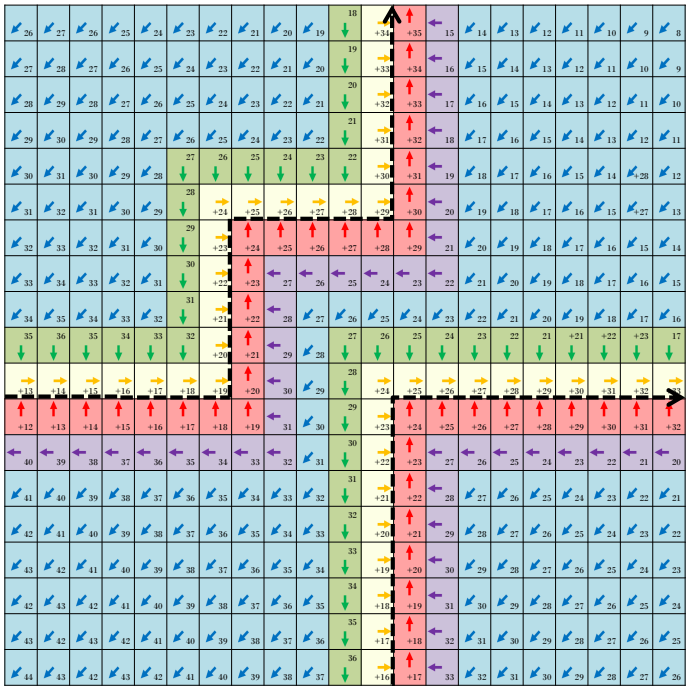


Figure 6: The embedding of the line near a crossover.

5.1 Proof of Theorem 5.1

We prove the correctness of the above reduction. First, we show that the result of the reduction is a valid instance of Continuous-Local-Optimum. That is, we show that f is 4-Lipschitz and p is 2^{3n}-Lipschitz. Second, we show that any local optimum of the resulting instance of Continuous-Local-Optimum can efficiently be transformed to a solution for the EOML instance.

The proof that f and p are Lipschitz follows from the way they are defined, i.e., by Cartesian interpolation. The proof is standard and rather technical, and thus we defer it to the appendix (see Lemma A.1 and Lemma A.2 in Appendix A.1). The core of our proof is to demonstrate that we did not introduce any local optima far from any end of a line. That is, we show that any unit square that contains a local optimum can be efficiently transformed into a solution of the EOML instance.

We consider the picture to be divided into square templates, each of size 1×1 such that its corners are at the middle of the squares in the picture. The different types of templates correspond to the different colors assigned to the four squares covering the template. Chen and Deng [CD09] showed that no template composed of only two colors contains a fixed point. We show a stronger statement: such templates do not contain a local optimum either.

Lemma 5.2. *There is no ε-local optimum in any template that contains only two colors.*

Proof. For any point z we define $\Delta = \Delta(z) = p(f(z)) - p(z)$ to be the improvement in p-value after taking a single step in f originating from z. For each template we prove that for any point z inside

this template the improvement is greater than ε, i.e.,

$$\Delta(z) = p(f(z)) - p(z) > \varepsilon.$$

Given a point in a specific template, we consider its coordinates relative to the template (the x axis is positioned along the bottom of the square template, so that the origin is at the bottom-left corner, and the top-right corner of the template has coordinates (1,1)). We prove that $\Delta(z) > \varepsilon$ for all $z = (x, y) \in [0, 1]^2$. Let $a, b, c, d \in \mathbb{N}$ be the values of p on the top-left, top-right, bottom-left, and bottom-right corners respectively. We define $p_\perp(x) = xd + (1-x)c$, and $p_\top(x) = xb + (1-x)a$. Then, we can express p in terms of $p_\perp$ and $p_\top$ as

$$\begin{aligned} p(x, y) &= yp_\top(x) + (1 - y)p_\perp(x) \\ &= (-a + b + c - d)xy + (a - c)y + (-c + d)x + c \\ &= a'xy + b'y + c'x + d', \end{aligned}$$

where $a' = -a + b + c - d$, $b' = a - c$, $c' = -c + d$, and $d' = c$. Recall that $f(z) = f(x, y) = z + \delta d(x, y) = (x + \delta d_x(x, y), y + \delta d_y(x, y))$, where d_x (respectively d_y) is the x component (respectively the y component) of the displacement function d. Thus, we can express $\Delta(z)$ as

$$\begin{aligned} \Delta(x, y) &= p(f(x, y)) - p(x, y) \\ &= p(x + \delta d_x(z), y + \delta d_y(z)) - p(x, y) \\ &= \delta(c'd_x(z) + b'd_y(z) + a'(\delta d_x(z)d_y(z) + d_y(z)x + d_x(z)y)). \end{aligned}$$

Since f is also defined by Cartesian interpolation, we can plug in the specific values for p and for f at the four corners of the template to the above expression, and we get an explicit formula for $\Delta(x, y)$ in the template.

We begin by considering all the points $z \in [0, 1]^2$ such that $f(z) \in [0, 1]^2$, i.e., points such that their image under f lands in the same template. In this case we use the explicit formula for $\Delta(z)$ to directly argue that $\Delta(z) > \varepsilon$. In the rest of this part of the proof we go over all the possible templates that can occur in our reduction. For each template we plug in the corner values of f and p for the template, derive the explicit formula for $\Delta(x, y)$, and prove that it is always larger than $\varepsilon' = 2\varepsilon$. The complete analysis of all the different templates is given in Appendix A.2.

It remains to show that also the points that are brought by f outside their template do not constitute local optima. Notice that any such point must be in a distance of at most δ from the border of the template since f makes steps of length at most δ (in the maximum norm). Recall that by considering the points that under f stay inside the template, we have already shown that all the points that lie inside the template at least δ-far from its border are not local optima. We use this fact together with the following claim to prove that also none of the points δ-close to the boundary of the template constitute a local optimum.

Claim 5.3. *Let $f\colon \mathbb{R}^3 \to \mathbb{R}^3$ and $p\colon \mathbb{R}^3 \to \mathbb{R}$ be λ_f-Lipschitz and λ_p-Lipschitz respectively, and let $\varepsilon, \gamma > 0$. Let $R \subset \mathbb{R}^3$ be such that for all $z \in R$ it holds that*

$$p(f(z)) - p(z) > \varepsilon.$$

Then for all $z \in R_\gamma = \{x : \exists y \in R, |x - y| \leq \gamma\}$ it holds that

$$p(f(z)) - p(z) > \varepsilon - \lambda_p\lambda_f\gamma - \lambda_p\gamma.$$

Proof. Let $z \in R_\gamma$. By the definition of R_γ, there exists some $z' \in R$ at most γ-far from z, i.e., such that $|z - z'| \leq \gamma$. First, we derive the following bound on the difference in the p-values at the images of z and z' under f:

$$|p(f(z)) - p(f(z'))| \leq \lambda_p |f(z) - f(z')| \leq \lambda_p \lambda_f |z - z'| \leq \lambda_p \lambda_f \gamma,$$

where we used the λ_p-Lipschitz continuity of p, the λ_f-Lipschitz continuity of f, and the bound on the distance of z and z'. Since no $z' \in R$ can be an ε-approximate local optimum, we get that

$$p(f(z)) \geq p(f(z')) - \lambda_p \lambda_f \gamma > p(z') + \varepsilon - \lambda_p \lambda_f \gamma. \tag{5.1}$$

Additionally, from the λ_p-Lipschitz continuity of p and the bound on distance of z and z' we get that $|p(z) - p(z')| \leq \lambda_p |z - z'| \leq \lambda_p \gamma$. Thus, $p(z') \geq p(z) - \lambda_p \gamma$, which we can plug into Equation (5.1) and we get that

$$p(f(z)) > p(z') + \varepsilon - \lambda_p \lambda_f \gamma \geq p(z) - \lambda_p \gamma + \varepsilon - \lambda_p \lambda_f \gamma,$$

as claimed. □

We have proved that any point z within a template that is at least δ-far from the border satisfies $\Delta(z) > \varepsilon' = 2\varepsilon$. Thus, we can define R to be all such points that are at least δ-far from the border of the template, and using Claim 5.3 we get that for *all* points z in the template it holds that

$$\Delta(z) > \varepsilon' - \lambda_p \lambda_f \delta - \lambda_p \delta \geq 2\varepsilon - 5\lambda_p \delta \geq 2\varepsilon - 1/n \geq \varepsilon.$$

Which concludes the proof of Lemma 5.2. □

Extracting a Solution. To finish the proof of Theorem 5.1, we show how to translate any solution to the resulting Continuous-Local-Optimum instance to a solution for original the End-of-Metered-Line instance. Suppose that we are given an arbitrary local optimum $(X, Y) \in [0, L]^2$ inside the picture (there are no local optima in the frame by our construction). We parse the coordinates of (X, Y) as $((b_x, s_x, x), (b_y, s_y, y))$, where $(b_x, b_y) \in [2^n] \times [2^n]$ is the block, $(s_x, s_y) \in [10 \cdot 2^n] \times [10 \cdot 2^n]$ is the square inside this block and $(x, y) \in [0, 1]^2$ is the relative position of (X, Y) within this square. We have proved that all templates containing only two colors contain no local optimum (Lemma 5.2). Thus, any local optimum must be inside a template in the vicinity of the beginning or the end of some line. Recall that each block contains a number of squares that is a multiple of 10, and thus let $s'_x = \lfloor s_x/10 \rfloor$ and $s'_y = \lfloor s_y/10 \rfloor$. Our main claim is that if (X, Y) is a local optimum in the Continuous-Local-Optimum instance then either s'_x or s'_y is a solution for the End-of-Metered-Line instance. Since verifying a solution can be done efficiently, given that the claim is true, extracting a solution is easy. We elaborate more on why the claim is true.

First, consider the simplest case where the implicit graph defined by S and P from the End-of-Metered-Line instance contains a single line, and some additional cycles. Moreover, V gives the correct incremental values for vertices on the line, and it assigns the zero value to any other vertex. Let w be the sink of this line, i.e., the only solution to the End-of-Metered-Line instance. In this case, the reduction results in an instance with a single continuous line (cf. Figure 5) with exactly one local optimum that occurs at block $(\mathsf{V}(w), \mathsf{V}(w) + 1)$ at square $(10 \cdot \mathsf{S}(w), 10 \cdot w)$, and thus the claim follows.

Now consider a graph with an additional line (or many additional lines), where V outputs the values according to the distance of the vertices on the additional line with respect to its actual source. Then we will have two (or more) staircases traveling through the blocks. Here comes the importance of the crossover template (cf. Figure 6) that ensures two staircases never cross (which would create a local optimum at the crossing point). The only local optima are in the vicinity of the beginning/end of these staircases, and thus we get the sink/source of the corresponding line, which is a solution to the End-of-Metered-Line instance.

In a more general case, it might be that V gives inconsistent values for vertices on a line. Again, any deviation from the "correct" value (i.e., the exact distance from the source) will cut the line embedded at that block. Let w be the last vertex on which V answers consistently. Then, there will be a local optimum at block $(\mathsf{V}(w), \mathsf{V}(w)+1)$ at square $(10 \cdot \mathsf{S}(w), 10 \cdot w)$.

In the most general case, we might have collisions in S, collisions in P and have V output inconsistent values for many vertices. Suppose that $\mathsf{S}(x) = \mathsf{S}(x') = y$ and $\mathsf{P}(y) = \mathsf{S}(x)$. Then, x' is a solution (a sink), and indeed we will have a local optimum at block $(\mathsf{V}(x), \mathsf{V}(x)+1)$ at square $(10 \cdot y, 10 \cdot x')$ which can be used to extract x'. Now suppose that $\mathsf{P}(y) = \mathsf{P}(y') = x$ and $\mathsf{S}(x) = y$. Then y' is a solution (a source), and indeed we will have a local optimum at block $(\mathsf{V}(y'), \mathsf{V}(y')-1)$ at square $(10 \cdot y', 10 \cdot x)$. Moreover, if $\mathsf{V}(x) = 1$ and $x \neq 0^n$ then x is a solution and we will have a local optimum at block $(1,1)$ at square $(10 \cdot x, 0)$. In general, the End-of-Metered-Line instance might have many different solutions which will yield many different local optima. However, from coordinates of any such local optimum it is easy to recover the original solution. To illustrate the result of the embedding we give an example for a specific End-of-Metered-Line instance in Figure 7.

Normalizing the Domain. Finally, we show how to convert f and p to the appropriate domain and range. First, we scale the domain of the two functions to be the unit square, and we define $f'(z) = \frac{1}{T} \cdot f(Tz)$ and $p'(z) = \frac{1}{T} \cdot p(Tz)$ for $T = 4L$.

Claim 5.4. *Let $f\colon \mathbb{R}^n \to \mathbb{R}^m$ be a λ-Lipschitz function. Then for any $T > 0$ the function $f'(z) = \frac{1}{T} \cdot f(Tz)$ is λ-Lipschitz.*

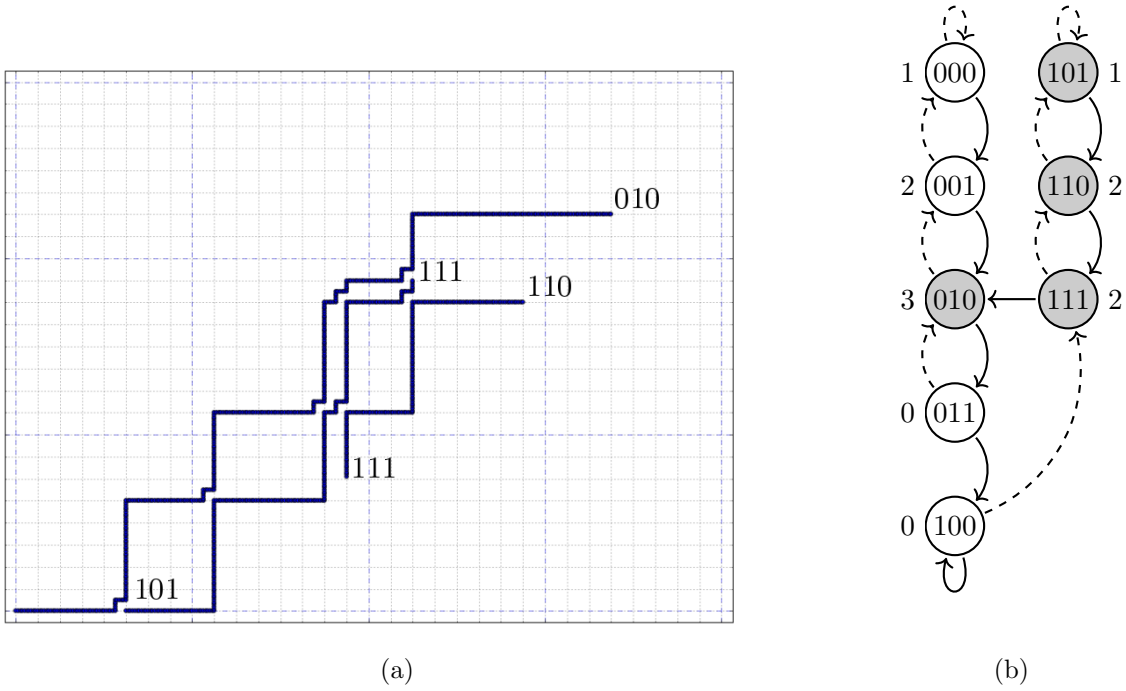
Proof. We get by a simple calculation that

$$|f'(z) - f'(z')| = \left| \frac{1}{T} \cdot f\left(Tz\right) - \frac{1}{T} \cdot f\left(Tz'\right) \right| = \frac{1}{T} \left| f\left(Tz\right) - f\left(Tz'\right) \right| \leq \frac{1}{T} \lambda \left| Tz - Tz' \right| = \lambda |z - z'|,$$

as claimed. □

Recall that $L = 10(2^{2n}+1)$. Since f is 4-Lipschitz, by Claim 5.4, we get that f' is 4-Lipschitz. Similarly, since p is 2^{3n}-Lipschitz we get that p' is 2^{3n}-Lipschitz. Thus, for the final construction we set $\lambda' = 2^{3n}$, $\delta' = \delta/T$ and $\varepsilon' = \varepsilon/T$, such that any ε-local optimum with respect to f and p is an ε'-local optimum with respect to f' and p'.

Finally, notice that f' and p' are defined over $[0,1]^2$ and not over $[0,1]^3$ as in the definition of Continuous-Local-Optimum. They can be easily extended to $[0,1]^3$ by copying over the third dimension. Namely, define $f''(x,y,z) = (f'(x,y), z)$ and $p''(x,y,z) = p'(x,y)$. It is easy to see that this transformation does not change the Lipschitz constants of the functions. This concludes the proof of Theorem 5.1.



(a) (b)

Figure 7: (a) A collection of staircases obtained by our reduction from a specific EOML instance. Each end point of a staircase is marked by the label of the corresponding vertex that constitutes a solution in the EOML instance. (b) The EOML instance. A solid (respectively dashed) arrow from i to j denotes that j is a successor (respectively predecessor) of i. The value next to a vertex denotes its valuation under V. The vertices with dark background are the solutions obtained via our reduction: 101 is a solution of the second type ($\mathsf{V}(101) = 1$), 110 and 010 are solutions of the third type since the value of their successors is not incremented, 111 is a solution of the third type since the value of its predecessor is not decremented, and additionally also a solution of the first type because it constitutes a sink in the graph ($\mathsf{P}(\mathsf{S}(111)) = 001 \neq 111$).

6 On the Hardness of End-of-Metered-Line

6.1 Query Complexity Lower Bound

In this section, we show an exponential query complexity lower bound for End-of-Metered-Line. This, in turn, shows a query complexity lower bound for **CLS**. In more general terms, we show that finding a local optimum even in a continuous domain is exponentially hard.

The Query Model. Let $\Pi(n)$ be a search problem defined by a function parametrized by n. In the query model, an algorithm is given oracle access to the function, and we measure the number of oracle queries performed by the algorithm, whereas its running time may be unbounded. We denote by $\mathrm{QC}_p(\Pi(n))$ the number of queries required for any randomized algorithms to solve $\Pi(n)$ with probability at least p, and we denote $\mathrm{QC}(\Pi(n)) = \mathrm{QC}_{2/3}(\Pi(n))$.

We consider the End-of-Metered-Line in the query model. Formally, let $\mathrm{EOML}(n)$ be given by an oracle that on query $v \in \{0,1\}^n$ outputs a triple consisting of the predecessor of v, the

successor of v, and the value of v. That is, the EOML oracle will answer for any vertex $v \in \{0,1\}^n$ with the triple $(\mathsf{S}(v), \mathsf{P}(v), \mathsf{V}(v)) \in \{0,1\}^n \times \{0,1\}^n \times \mathbb{N}$. The goal is to find a vertex v that satisfies one of the three conditions for a solution of End-of-Metered-Line (cf. Definition 4.1). We show that the randomized query complexity of EOML(n) is exponential in n.

Theorem 6.1. $\mathrm{QC}(\mathrm{EOML}(n)) = \Omega\left(2^{n/2}/\sqrt{n}\right)$.

In order to show a lower bound for probabilistic algorithms, by Yao's minmax principle it is enough to show that there exists a distribution over instances such that every *deterministic* algorithm fails with high probability.

For the problem Local-Search, i.e., finding a (non-continuous) local optimum on the n-dimensional Boolean hypercube, Zhang [Zha09] gave a tight query complexity lower bound of $\Theta(2^{n/2}\sqrt{n})$. In this section we show how to adapt his techniques to get a matching lower bound for End-of-Metered-Line. In particular, Zhang [Zha09] defined a query complexity problem Path, and showed that $\mathrm{QC}(\textsc{Path}) \leq 2\mathrm{QC}(\textsc{Local-Search})$. Finally, to argue the tight query complexity lower bound for Local-Search, Zhang proved the following lower bound for Path (matching upper bound for Local-Search was given by Aldous [Ald83]):

Claim 6.2 ([Zha09])**.** $\mathrm{QC}(\textsc{Path}) = \Omega\left(2^{n/2} \cdot \sqrt{n}\right)$.

The Path Game. We begin by describing the Path problem that asks to find an endpoint on a random self-avoiding path over the Boolean hypercube. In particular, consider a decomposition of the Boolean hypercube $\{0,1\}^n$ as $\{0,1\}^m \times \{0,1\}^{n-m}$, where $m = \lfloor (n + \log n)/2 \rfloor$. We construct a self-avoiding path over $\{0,1\}^n$ that corresponds to following a random walk on the first component $\{0,1\}^m$, while using the second component $\{0,1\}^{n-m}$ to keep a step-counter for the random walk in order to avoid self-loops.

Formally, fix any Hamiltonian path of length $2^{n-m} - 1 = 2T + 1$ over $\{0,1\}^{n-m}$. We denote the vertices on the Hamiltonian path as

$$(z_{0,0}, z_{1,0}, z_{1,1}, z_{2,1}, z_{2,2}, \ldots, z_{T,T}).$$

Next, we build a random walk over $\{0,1\}^m$ that starts at an arbitrary vertex x_0 (e.g., $x_0 = 0^m$), and proceeds for T steps as follows. For all $i \in [T]$, we choose $x_i \in \{0,1\}^m$ by flipping a random coordinate in x_{i-1}. Finally, given the random walk $(x_0, x_1, \ldots, x_T)$ and the Hamiltonian path $(z_{0,0}, z_{1,0}, \ldots, z_{T,T})$, we define a path X over $\{0,1\}^n = \{0,1\}^m \times \{0,1\}^{n-m}$ by making a step in the random walk and two subsequent steps on the Hamiltonian path, i.e.,

$$X = ((x_0, z_{0,0}), (x_1, z_{0,0}), (x_1, z_{1,0}), (x_1, z_{1,1}), \ldots, (x_T, z_{T-1,T-1}), (x_T, z_{T,T-1}), (x_T, z_{T,T})).$$

The Path problem is given by an oracle access to the above path X, i.e., the Path oracle answers 1 for any vertex $v \in \{0,1\}^n$ that lies on the path X and 0 for any point that lies off the path X. The goal is to find the endpoint of the path X, i.e., the vertex $v = (x_T, z_{T,T})$.

We show that any query complexity lower bound for the Path problem implies a query complexity lower bound for End-of-Metered-Line.

Claim 6.3. $\mathrm{QC}(\textsc{Path}) \leq n\mathrm{QC}(\mathrm{EOML})$.

Proof. For any Path oracle, we implement an equivalent EOML oracle.

EOML-Oracle(v)**:**

1. Query $\textsc{Path}(v)$ to learn whether v lies on the path X.
2. If v lies off the path X, output $(v, v, 0)$.
3. If $v = (x, z_{T,T})$ then output $((x, z_{T,T-1}), (x, z_{T,T}), 3T + 1)$.
4. If $v = (x, z_{k,k})$ for some $k \in [T] \cup \{0\}$ and $\textsc{Path}(x, z_{k+1,k}) = 0$ then:
   (a) For all x' that differ from x in a single coordinate, query $\textsc{Path}(x', z_{k,k})$ to find $(x_{k+1}, z_{k,k})$, the next vertex on the path X.
   (b) If $k = 0$ then output $((x, z_{0,0}), (x_1, z_{0,0}), 1)$.
   (c) Otherwise output $((x, z_{k,k-1}), (x_{k+1}, z_{k,k}), 3k + 1)$.
5. If $v = (x, z_{k,k})$ for some $k \in [T] \cup \{0\}$ and $\textsc{Path}(x, z_{k,k-1}) = 0$ then:
   (a) For all x' that differ from x in a single coordinate, query $\textsc{Path}(x', z_{k,k})$ to find $(x_{k-1}, z_{k,k})$, the previous vertex on the path X.
   (b) Output $((x_{k-1}, z_{k,k}), (x, z_{k+1,k}), 3k - 1)$.
6. If $v = (x, z_{k,k-1})$ for some $k \in [T]$ then output $((x, z_{k-1,k-1}), (x, z_{k,k}), 3k)$.

The unique solution to the new EOML instance is the local optimum at the vertex $v = (x_T, z_{T,T})$, which constitutes also a solution to the original Path problem. Notice that to answer each End-of-Metered-Line query we perform at most $m + 2 \leq n$ queries to the Path oracle. Thus, we get that $\mathrm{QC}(\textsc{Path}) \leq n\mathrm{QC}(\mathrm{EOML})$, as claimed. □

By combining Claim 6.3 with Claim 6.2 we get the following query complexity lower bound for End-of-Metered-Line:

$$\mathrm{QC}(\mathrm{EOML}) \geq 2^{n/2}/\sqrt{n},$$

as claimed in Theorem 6.1.

Query Complexity of Continuous-Local-Optimum. To get the exponential query complexity lower bound for Continuous-Local-Optimum, we combine the above lower bound for End-of-Metered-Line with our reduction from Theorem 5.1. It follows that one must perform $2^{n/2}/\sqrt{n}$ queries to get accuracy ε, where in the reduction we set $\varepsilon \approx 2^{-5n}$ (up to a polynomial factor). In other words, to solve Continuous-Local-Optimum with n-digits of precious (i.e., up to accuracy $\varepsilon = 2^{-n}$) one must perform approximately $2^{n/10} = 2^{\Omega(n)}$ queries.

Alternative Approach Based on Lower Bounds for PPAD. We note that we could have directly proved exponential query complexity lower bound for **CLS** by combining our reduction from End-of-Metered-Line to Continuous-Local-Optimum from Theorem 5.1 with the previous works on query complexity of computing approximate Brouwer fixed points (e.g., [HPV89, LNNW95]). However, our query complexity lower bound for End-of-Metered-Line presented in Theorem 6.1 is a stronger result, since End-of-Metered-Line is not complete for

CLS, and hence we give a lower bound for potentially easier problem than continuous local search. For completeness, we sketch below the direct approach for showing black-box hardness for **CLS**.

In the case of Brouwer fixed points for functions over the unit square, Hirsch, Papadimitriou and Vavasis [HPV89] showed an exponential query complexity lower bound for any deterministic algorithm treating the function as a black-box. Their work was based on reducing the problem of finding an end of a staircase traversing square grid with N^2 points to finding a fixed point of a continuous function over the unit square. Specifically, they showed that any query complexity lower bound for the staircase problem with N^2 points implies a matching query complexity lower bound for finding $\frac{1}{N}$-approximate Brouwer fixed points. They showed that the deterministic query complexity of the staircase problem is $\Theta(N)$ (Lovász et al. [LNNW95] showed a randomized query complexity lower bound of $\Omega(N^{\frac{1}{3}})$), which implies an exponential query complexity lower bound for finding approximate Brouwer fixed points and thus for **PPAD**.

Similarly to [HPV89], our reduction from End-of-Metered-Line to Continuous-Local-Optimum also provides an embedding of any staircase into the unit square resulting in a continuous function f that has a fixed point only at the end of the staircase. The additional property of our construction is that we are able to define a valuation function p assigning to any point of the unit square a real value such that any local optimum of p under f is in the vicinity of the fixed point of f. Moreover, the p-value at any point x is computed based solely on its coordinates and its f-value. In other words, any oracle call to p does not provide any additional information since it can be simulated based on oracle calls to f. Any algorithm that can solve our continuous local optimum instance given oracle access to f and p can solve it only given the oracle access to f. Hence, our staircase embedding can readily be used in place of the embedding of Hirsch et al. [HPV89] to obtain a lower bound for **CLS** from any query complexity lower bound for the above grid problem.

6.2 Cryptographic Hardness of End-of-Metered-Line

In this section, we show that under cryptographic assumptions End-of-Metered-Line (EOML) is hard. Formally, we prove the following theorem.

Theorem 6.4. *Assume there exist one-way permutations and indistinguishability obfuscation for* **P/Poly**. *Then the* End-of-Metered-Line *problem is hard for polynomial-time algorithms.*

Our cryptographic hardness result for End-of-Metered-Line builds upon previous works on cryptographic hardness for **PPAD**. Recently, Bitanski et al. [BPR15] were able to show that assuming one-way functions (see Definition D.1) and indistuinguishability obfuscation (both with subexponential security) the End-of-Line problem is hard. Their proof followed two main steps. First, they defined a problem called Sink-of-Verifiable-Line (motivated by the work of Abbott, Kane and Valiant [AKV04]), and they showed that Sink-of-Verifiable-Line is hard under the above cryptographic assumptions. Second, they gave a reduction from Sink-of-Verifiable-Line to End-of-Line yielding the conclusion that End-of-Line is hard under the same assumptions.

We start by giving an overview of the original reduction to End-of-Line and then describe our modifications. The following formal definition of Sink-of-Verifiable-Line was given in Bitanski et al. [BPR15].

Definition 6.5 (Sink-of-Verifiable-Line [BPR15])**.** *An instance* $(\mathsf{S}, \mathsf{V}, x_s, T)$ *consists of a source* $x_s \in \{0,1\}^n$*, a target index* $T \in [2^n]$*, and a pair of circuits* $\mathsf{S}\colon \{0,1\}^n \to \{0,1\}^n$*,* $\mathsf{V}\colon \{0,1\}^n \times [T] \to \{0,1\}$*, with the guarantee that, for all* $(x,i) \in \{0,1\}^n \times [T]$*, it holds that* $\mathsf{V}(x,i) = 1$ *iff* $x = x_i := \mathsf{S}^{i-1}(x_s)$*, where* $x_1 := x_s$*. A string* $w \in \{0,1\}^n$ *is a valid witness iff* $\mathsf{V}(w,T) = 1$.

As discussed in the previous works [AKV04, BPR15, GPS15], the above problem is not necessarily total without the promise about the behavior of the verification circuit V. In particular, V might just for all $x \in \{0,1\}^n$ reject any pair (x, T) and such behavior cannot be efficiently checked.[6] Notice that there is no need for an explicit source vertex x_s in the above definition. The source can be without loss of generality labeled 0^n, and we use this convention from now on as it is standard in definitions of other problems inside **TFNP** (see Section 3).

Warm-up Reduction to End-of-Line. Consider a Sink-of-Verifiable-Line instance denoted $(\mathsf{S}, \mathsf{V}, T)$. In order to reduce it to an End-of-Line instance it is necessary to implement the predecessor circuit P'. Notice that it is easy to construct the predecessor circuit in an *inefficient* way. One can simply modify the labels of the vertices to contain the entire history of the previous steps on the line. That is, we construct circuits S' and P' such that if $0^n \to x_2 \to x_3 \to \cdots \to x_T$ is a line according to S then there is a line according to S' of the form

$$0^n \to (0^n, x_2) \to (0^n, x_2, x_3) \to (0^n, x_2, x_3, x_4) \to \cdots \to (0^n, x_2, \ldots, x_T) \ .$$

Given these labels, implementing the predecessor is easy: simply remove the last element from the label. However, the obvious issue of this transformation is that the size of the labels becomes eventually exponentially large which prevents it from being a polynomial reduction. To reduce the size of the labels to give an efficient construction of the predecessor circuit, [AKV04, BPR15] utilized techniques used for implementing *reversible computation* [Ben89] where only a small number of states is stored in order to be able to revert previous steps in the computation. The general approach can be explained via a simple *pebbling game* that we describe next.

The Pebbling Game. There are n pebbles that can be placed on positions indexed by positive integers. The rules of the game are as follows: a pebble can be placed in or removed from position i if and only if either there is a pebble in position $i-1$ or $i = 1$. The goal of the game is to place a pebble in position $2^n - 1$.

As shown by Chung, Diaconis and Graham [CDG01], the optimal efficient strategy achieves the goal of the game in a recursive manner. The main idea is that since the rules for placing and removing pebbles are symmetric, it is always possible to reverse any sequence of moves. Suppose then there is a way to get to $2^{n-1} - 1$ using $n-1$ pebbles. Then, place a pebble at 2^{n-1}. Next, free the first $n-1$ pebbles by reversing the original sequence of moves performed in the first part. Finally, perform the same sequence starting from 2^{n-1}. This strategy will end with a pebble at position $2^n - 1$.

The predecessor circuit in the reduction from Sink-of-Verifiable-Line to End-of-Line is implemented by simulating the optimal strategy in the pebbling game. Each vertex has a label representing the states of the n pebbles. The efficient strategy demonstrates that by storing only n intermediate states we can implement S' and P' that can traverse an exponential number of steps. The resulting End-of-Line instance $(\mathsf{S}', \mathsf{P}')$ corresponds to a graph with a single line traversing the sequence of all the configurations visited by the optimal pebbling strategy. In particular, every vertex corresponding to an intermediate state of the pebbling strategy is followed by the subsequent state, and the final step of the pebbling strategy is a self-loop under S'. Any state

[6] In fact, the hardness results of [BPR15, GPS15] strongly exploit this fact. Both works show that under cryptographic assumptions there exist instances of Sink-of-Verifiable-Line that are computationally indistinguishable from instances that do not obey the promise on behavior of V and have no solutions.

describing an illegal configuration of the pebbling game is defined to be a self loop both under S' and P'. Therefore, the resulting instance has a unique solution, a sink that identifies a solution to the original Sink-of-Verifiable-Line instance. For completeness, the pseudocode of S' and P' are given in Appendix C.

Our Reduction. In order to give a reduction to End-of-Metered-Line, we must provide not only the predecessor circuit P' but also the valuation circuit V' that meters the line starting at 0^n. Recall the inefficient construction of the predecessor circuit that uses the labels of the vertices to store the complete history of line. Given these labels the implementation of V' is simple: the distance of a vertex from the start of the line is exactly the number of elements in its label. Our crucial observation is that even in the efficient reduction based on simulating the pebbling game the predecessor circuit does not come at the expense of the verification circuit V from the Sink-of-Verifiable-Line instance.

Since the label of each vertex corresponds to a configuration of the pebbling game, the value we assign to each vertex is the number of steps performed in the optimal strategy until reaching the specific configuration corresponding to its label. To construct the valuation circuit V' we need to show that, given a configuration, it is possible to efficiently compute where exactly in the pebbling strategy we are without simulating the entire game. We start by computing the total number of steps in the efficient pebbling strategy.

Claim 6.6. *Let $g(n)$ be the number of steps performed in the pebbling game by the optimal strategy until a pebble is put in position $2^n - 1$. Then the following hold:*

1. $g(n) = 3g(n-1) + 1$,
2. $g(n) = \frac{3^n - 1}{2}$.

Proof. We begin by proving the first item. By the definition of the pebbling strategy, we first perform $g(n-1)$ steps to put a pebble at position $2^{n-1} - 1$. Then we put the n^{th} pebble at position 2^{n-1}. Then, we reverse the initial $g(n-1)$ moves to free all the first $n-1$ pebble. Finally, we use the same strategy again starting from $2^{n-1} + 1$, and perform another $g(n-1)$ steps to get to position $2^n - 1$. Hence, the total number of steps is $3g(n-1) + 1$.

The second item follows by the recursive definition given in the first item. We have that

$$g(n) = 3g(n-1) + 1 = 3^2 g(n-2) + 3 + 1 = \ldots = 3^{n-1} + 3^{n-2} \ldots + 3 + 1 = \frac{3^n - 1}{2},$$

as claimed. □

The circuit V' can be built based on the recursive expression for $g(n)$ from Claim 6.6. Note that $g(n)$ was computed as an addition of three phases: (1) getting to the half, (2) reversing the first half, and (3) completing the second half. To check if phase 1 has finished, we check if position 2^{n-1} is occupied. This will indicate that we have already performed at least the initial $g(n-1)+1$ steps. Otherwise, phase 1 has not be finished and we return a recursive call with the first half of the board. Next, to check if phase 2 (reversing the computation) we check whether all the pebbles are to the right of position 2^{n-1}. If there is even 1 pebble of the left, then we are still in phase 2. The number of steps taken in this phase is $g(n-1)$ minus a recursive call with $n-1$ on the first half of the board. If we are in phase 3 then we have performed $2g(n-1) + 1 = 3^n$ steps plus the

number of steps currently in the phase which is computed by a recursive call to the second half of the board.

For completeness, any vertex with label describing an illegal configuration of the pebbling game gets a default value 0. The formal description of this procedure is given in Algorithm 1.

Correctness of our Reduction. The key point of our reduction is that the resulting End-of-Metered-Line instance $(\mathsf{S}', \mathsf{P}', \mathsf{V}')$ has a single line corresponding to the line of the SVL instance, and all other nodes are self-loops. This restricts the set of possible solutions of $(\mathsf{S}', \mathsf{P}', \mathsf{V}')$ as follows. There is a unique solution of the first type corresponding to the sink at the last state of the pebbling game. There is no solution of the second type, since the only vertex that is assigned value 1 by V' is the initial source. The unique solution of the first type is also the unique solution of the third type, since the only vertex satisfying $0 \neq \mathsf{V}'(x) \neq \mathsf{V}'(\mathsf{S}'(x)) - 1$ is the sink corresponding to the last state of the pebbling game. Moreover, from the unique solution it is easy to extract a vertex x that is a solution to the original SVL instance. In particular, one of the pebbles at the final configuration of the pebbling strategy is exactly in a position corresponding to an x such that $\mathsf{V}(x, T) = 1$.

```
1: function V'(N = (u_1, ..., u_n))
2:     if all u_1, ..., u_n are valid states then
3:         return 1+V'_n(1, 2^n − 1, n, N)
4:     else
5:         return 0
6:     end if
7: end function
8:
9: function V'_j(start, end, N = (u_1, ..., u_n))
10:    if j = 0 then return 0
11:    end if
12:    mid ← start + 2^{j−1} − 1
13:    if position mid is free then
14:        return V'_{j−1}(start, mid − 1, N)
15:    else if there is a pebble at position i ∈ [start, mid − 1] then
16:        return 3^{j−1} − V'_{j−1}(start, mid − 1, N)
17:    else return 3^{j−1} + V'_{j−1}(mid + 1, end, N)
18:    end if
19: end function
```

Algorithm 1: The algorithm V'.

Acknowledgements

We are grateful to Karthik C. S. for introducing us to the class **CLS**, and for many helpful discussions about **TFNP** in general. We also wish to thank Uriel Feige, Moni Naor, Roei Tell, and Margarita Vald for their invaluable help and comments.

References

[Aar06] Scott Aaronson. Lower bounds for local search by quantum arguments. *SIAM J. Comput.*, 35(4):804–824, 2006.

[AB15] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 528–556, 2015.

[AKV04] Tim Abbott, Daniel Kane, and Paul Valiant. On algorithms for Nash equilibria. http://web.mit.edu/tabbott/Public/final.pdf, 2004. unpublished manuscript.

[Ald83] David Aldous. Minimization algorithms and random walk on the d-cube. *The Annals of Probability*, 11(2):403–413, 1983.

[Bab14] Yakov Babichenko. Query complexity of approximate Nash equilibria. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 535–544, 2014.

[Ban22] Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. Math*, 3(1):133–181, 1922.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, 1989.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *Journal of the ACM*, 59(2):6, 2012. Preliminary version [BGI+01].

[BM04] Josh Buresh-Oppenheim and Tsuyoshi Morioka. Relativized NP search problems and propositional proof systems. In *19th Annual IEEE Conference on Computational Complexity (CCC 2004), 21-24 June 2004, Amherst, MA, USA*, pages 54–67, 2004.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In *56th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, October 18-20, 2015*, pages 1480–1498, 2015.

[BVWW16] Zvika Brakerski, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Obfuscating conjunctions under entropic ring LWE. In *Proceedings of the 2016 ACM Conference ITCS, Cambridge, MA, USA, January 14-16, 2016*, pages 147–156, 2016.

[CD08] Xi Chen and Xiaotie Deng. Matching algorithmic bounds for finding a brouwer fixed point. *J. ACM*, 55(3), 2008.

[CD09] Xi Chen and Xiaotie Deng. On the complexity of 2D discrete fixed point problem. *Theor. Comput. Sci.*, 410(44):4448–4456, 2009.

[CDG01] Fan Chung, Persi Diaconis, and Ronald Graham. Combinatorics for the east model. *Advances in Applied Mathematics*, 27(1):192–206, 2001.

[CDT09] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *J. ACM*, 56(3), 2009.

[Con92] Anne Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.

[CT07] Xi Chen and Shang-Hua Teng. Paths beyond local search: A tight bound for randomized fixed-point computation. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 124–134, 2007.

[DGP09] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM J. Comput.*, 39(1):195–259, 2009.

[DP11] Constantinos Daskalakis and Christos H. Papadimitriou. Continuous local search. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 790–804, 2011.

[FISV09] Katalin Friedl, Gábor Ivanyos, Miklos Santha, and Yves F. Verhoeven. On the black-box complexity of Sperner's lemma. *Theory Comput. Syst.*, 45(3):629–646, 2009.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

[GLSW14] Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.

[GPS15] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. On the exact cryptographic hardness of finding a Nash equilibrium. *IACR Cryptology ePrint Archive*, 2015:1078, 2015.

[HPV89] Michael D. Hirsch, Christos H. Papadimitriou, and Stephen A. Vavasis. Exponential lower bounds for finding Brouwer fixed points. *J. Complexity*, 5(4):379–416, 1989.

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

[LNNW95] László Lovász, Moni Naor, Ilan Newman, and Avi Wigderson. Search problems in the decision tree model. *SIAM J. Discrete Math.*, 8(1):119–132, 1995.

[LTT89] Donna Crystal Llewellyn, Craig Tovey, and Michael Trick. Local optimization on graphs. *Discrete Applied Mathematics*, 23(2):157–178, 1989.

[Mor01] Tsuyoshi Morioka. Classification of search problems and their definability in bounded arithmetic. *Electronic Colloquium on Computational Complexity (ECCC)*, (082), 2001.

[MP91] Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theor. Comput. Sci.*, 81(2):317–324, 1991.

[Nas50] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36:48–49, 1950.

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO (1)*, volume 8616 of *Lecture Notes in Computer Science*, pages 500–517. Springer, 2014.

[Ros73] Robert W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

[Rub15] Aviad Rubinstein. Inapproximability of Nash equilibrium. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 409–418, 2015.

[Sha53] Lloyd S Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953.

[SS06] Rahul Savani and Bernhard Stengel. Hard-to-solve bimatrix games. *Econometrica*, 74(2):397–429, 2006.

[SS09] Miklos Santha and Mario Szegedy. Quantum and classical query complexities of local search are polynomially related. *Algorithmica*, 55(3):557–575, 2009.

[SY09] Xiaoming Sun and Andrew Chi-Chih Yao. On the quantum query complexity of local search in two and three dimensions. *Algorithmica*, 55(3):576–600, 2009.

[Zha09] Shengyu Zhang. Tight bounds for randomized and quantum local search. *SIAM J. Comput.*, 39(3):948–977, 2009.

A Proofs deferred from Section 5.1

A.1 Proof of f and p being Lipschitz

Lemma A.1. *The function f described in the reduction in Section 5 is λ_f-Lipschitz, for $\lambda_f = 4$.*

Proof. By the definition of f we have that $f(z) = z + \delta d(z)$, where $d(z)$ is the displacement function defined by interpolation of the different displacements at the centers of squares surrounding z. First, we show that $d(z)$ is λ_d-Lipschitz for $\lambda_d = 4$. Consider $d(z)$ on a single dimension, that is, fix one of the coordinates to any constant. Suppose that $x \in [a, b]$ where a, b are two adjacent center points

with displacement values d_a, d_b. Then the function $d_{a,b}(x) = (x-a)d_b + (b-x)d_a$ is the function $d(z)$ on this region over one dimension. Suppose that $x, x' \in [a,b]$. Then we get that

$$\begin{aligned}|d_{a,b}(x) - d_{a,b}(x')| &= |(x-a)d_b + (b-x)d_a - (x'-a)d_b - (b-x')d_a| \\ &= |(x-x')d_b + (x'-x)d_a| \\ &\leq |d_b - d_a| \cdot |x-x'| \\ &\leq 2|x-x'|,\end{aligned}$$

where we used that the maximum norm of both displacement values d_a and d_b is one. Thus, $d_{a,b}(x)$ is λ_d-Lipschitz on $[a,b]$ for $\lambda_d = 2$. Now suppose that $x \in [b,c]$ and $x' \in [a,b]$. Then we get that:

$$\begin{aligned}|d_{b,c}(x) - d_{a,b}(x')| &\leq |d_{b,c}(x) - d_{b,c}(b)| + |d_{a,b}(b) - d_{a,b}(x')| \\ &\leq 2\delta|x-b| + 2|b-x'| \\ &= 2|x-x'|.\end{aligned}$$

Now suppose that $x \in [a,b]$ and $x' \in [c,d]$ where $b < c$. Then we get that

$$|d_{a,b}(x) - d_{c,d}(x')| \leq 2 \leq 2|x-x'|.$$

Altogether, we got that for any a, b it holds that $d_{a,b}$ is 2-Lipschitz. Going back to two dimensions we get that for the four center points a, b, c, d representing the top-left, top-right, bottom-left, and bottom-right corners of a template respectively we have

$$d(x,y) = d_{a,b,c,d}(x,y) = (y - c_y)d_{c,d}(x) + (a_y - y)d_{a,b}(x).$$

Thus, we get that

$$\begin{aligned}|d(x,y) - d(x',y')| &\leq |d(x,y) - d(x,y')| + |d(x,y') - d(x',y')| \\ &\leq 2|y-y'| + 2\delta|x-x'| \\ &\leq 4\max\left\{|x-x'|, |y-y'|\right\} \\ &\leq 4|(x,y) - (x',y')|.\end{aligned}$$

We can now compute the Lipschitz constant of f:

$$\begin{aligned}|f(z) - f(z')| &= |z + \delta d(z) - z' - \delta d(z')| \\ &\leq |z-z'| + \delta|d(z) - d(z')| \\ &\leq |z-z'| + 4\delta|z-z'| \\ &\leq 4|z-z'|,\end{aligned}$$

which concludes the proof. □

We now establish the Lipschitz continuity of p.

Lemma A.2. *The function p described in the reduction in Section 5 is λ_p-Lipschitz, for $\lambda_p = 2^{3n}$.*

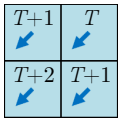
Proof. The analysis of p is simpler. The square with the highest value is at the end of the line, and its value is at most $4L$. The square with the smallest value is 0. The distance between 2 square is 1, and between the square we p is defined by linear interpolation. Therefore, is suffices to bound $|p_\perp(x) - p_\perp(x')|$ for $x, x' \in [0,1]$ such that $p_\perp(0) = 0$ and $p_\perp(1) = 4L$. For these values we get that

$$|p_\perp(x) - p_\perp(x')| \leq |x \cdot 4L - x' \cdot 4L| = 4L|x-x'| \leq 2^{3n}|x-x'|,$$

where we use the fact that $L = 10(2^{2n}+1)$. □

A.2 Analysis of Templates from Lemma 5.2

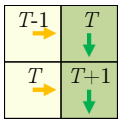
Case 1. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we get that

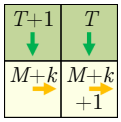
$$\Delta = (1 + 3x(1 - y))\delta \geq \delta > \varepsilon'.$$

Case 2. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = \delta > \varepsilon'$.

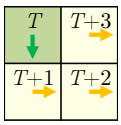
Case 3. The template is of the form:



for some $T, k \in \mathbb{N}$ such that $M + k > T + 1$. It suffice to prove for the case where $M + k = T + 2$. For these values we get that

$$\begin{aligned}
\Delta &= \delta\left(1 + 2y^2(1 - \delta) + 2y(-1 + x + \delta)\right) \\
&= \delta(1 + 2y(y(1 - \delta) + (-1 + x + \delta))) \\
&\geq \delta(1 + 2y((1 - \delta) + (-1 + x + \delta))) \\
&= \delta + 2xy\delta \geq \delta > \varepsilon'.
\end{aligned}$$

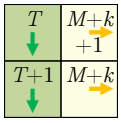
Case 4. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we that

$$\begin{aligned}
\Delta &= \delta\left(1 + 2y\left(1 + x^2 + x(-1 + \delta) - \delta\right) + 2(-1 + x)y^2(1 + (-1 + x)\delta)\right) \\
&= \delta\left(1 + 2y\left(1 + x^2 + x(-1 + \delta) - \delta + (-1 + x)y(1 + (-1 + x)\delta)\right)\right) \\
&\geq \delta\left(1 - 2xy\delta + 2x^2y(1 + \delta)\right) \\
&\geq \delta\left(1 - 2\delta\right)) \geq \delta/2 > \varepsilon'.
\end{aligned}$$

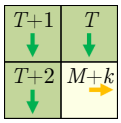
Case 5. The template is of the form:



for some $T, k \in \mathbb{N}$ such that $M + k > T + 1$. It suffice to prove for the case where $M + k = T + 2$. For these values we get that

$$\begin{aligned}\Delta &= \delta\left(1 + 2x(-1 + y - \delta) + 2x^2(1 + \delta)\right)\\ &= \delta(1 + 2x(-1 + x + y - \delta + x\delta))\\ &\geq \delta(1 + 2x(-1 + x - \delta))\\ &\geq \delta(3/4 - 2x + 2x^2)\\ &\geq \delta(3/4 - 1/2) = \delta/4 > \varepsilon'.\end{aligned}$$

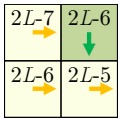
Case 6. The template is of the form:



for some $T, k \in \mathbb{N}$ such that $M + k > T + 2$. It suffice to prove for the case where $M + k = T + 3$. For these values we get that

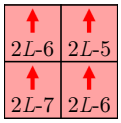
$$\begin{aligned}\Delta &= \delta\left(1 + 2x^2(1 - y)(-1 + (-1 + y)\delta) + 2x\left(1 + y^2 + \delta - y(1 + \delta)\right)\right)\\ &\geq \delta\left(1 + 2x\left((1 - y)(-1 + (-1 + y)\delta) + \left(1 + y^2 + \delta - y(1 + \delta)\right)\right)\right)\\ &= \delta(1 + 2xy(y + \delta - y\delta))\\ &\geq \delta > \varepsilon'.\end{aligned}$$

Case 7. The template is of the form:



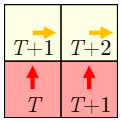
Such a template can be at exactly one position (in the bottom left position of the frame). For these values we directly get that $\Delta = \delta > \varepsilon'$.

Case 8. The template is of the form:



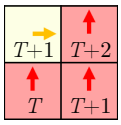
Such a template can be at exactly one position (in the bottom left position of the frame). For these values we get directly that $\Delta = \delta > \varepsilon'$.

Case 9. The template is of the form:



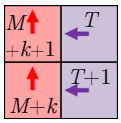
for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = \delta > \varepsilon'$.

Case 10. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = \delta > \varepsilon'$.

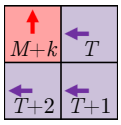
Case 11. The template is of the form:



for some $T, k \in \mathbb{N}$ such that $M + k > T + 2$. It suffice to prove for the case where $M + k = T + 3$. For these values we get that

$$\begin{aligned}\Delta &= \delta\left(1 - 2x^2(-1+\delta) + 2x(-1+y+\delta)\right)\\ &\geq \delta(1 + 2x(-1+x+y+\delta-x\delta))\\ &\geq \delta(1+2x(-1+x))\\ &= \delta(1-2x+2x^2) \geq \delta/2 > \varepsilon'.\end{aligned}$$

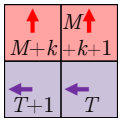
Case 12. The template is of the form:



for some $T, k \in \mathbb{N}$ such that $M + k > T + 2$. It suffice to prove for the case where $M + k = T + 3$. For these values we get that

$$\begin{aligned}\Delta &= \delta\left(1 - 2(-1+x)y^2(-1+(-1+x)\delta) + 2y\left(1+x^2+\delta-x(1+\delta)\right)\right)\\ &= \delta\left(1 + 2y\left(1+x^2+\delta-x(1+\delta)+(1-x)y(-1+(-1+x)\delta)\right)\right)\\ &\geq \delta\left(1 + 2y\left(1+x^2+\delta-x(1+\delta)+(1-x)(-1+(-1+x)\delta)\right)\right)\\ &= \delta\left(1 + 2x^2y(1-\delta) + 2xy\delta\right) \geq \delta > \varepsilon'.\end{aligned}$$

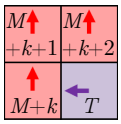
Case 13. The template is of the form:



for some $T, k \in \mathbb{N}$ such that $M + k > T + 1$. It suffice to prove for the case where $M + k = T + 2$. For these values we get that

$$\begin{aligned}\Delta &= \delta\left(1 + 2y(-1 + x - \delta) + 2y^2(1 + \delta)\right)\\ &= \delta(1 + 2y((-1 + x - \delta) + y(1 + \delta)))\\ &\geq \delta(1 + 2y((-1 + x - \delta) + (1 + \delta)))\\ &= \delta + 2xy\delta \geq \delta > \varepsilon'.\end{aligned}$$

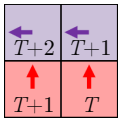
Case 14. The template is of the form:



for some $T, k \in \mathbb{N}$ such that $M + k > T$. It suffice to prove for the case where $M + k = T + 1$. For these values we get that

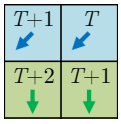
$$\begin{aligned}\Delta &= \delta\left(1 + 2x\left(1 + y^2 + y(-1 + \delta) - \delta\right) + 2x^2(-1 + y)(1 + (-1 + y)\delta)\right)\\ &= \delta\left(1 + 2x\left(\left(1 + y^2 + y(-1 + \delta) - \delta\right) + x(-1 + y)(1 + (-1 + y)\delta)\right)\right)\\ &\geq \delta\left(1 + 2x\left(\left(1 + y^2 + y(-1 + \delta) - \delta\right) + (-1 + y)(1 + (-1 + y)\delta)\right)\right)\\ &= \delta(1 + 2xy(y - \delta + y\delta)) \geq \delta > \varepsilon'.\end{aligned}$$

Case 15. The template is of the form:



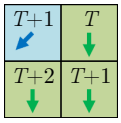
for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = \delta > \varepsilon'$.

Case 16. The template is of the form:



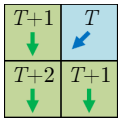
for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = (1 + y)\delta \geq \delta > \varepsilon'$.

Case 17. The template is of the form:



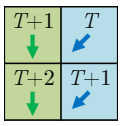
for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = (1+y-xy)\delta \geq \delta > \varepsilon'$.

Case 18. The template is of the form:



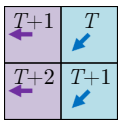
for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = \delta(1+xy) \geq \delta > \varepsilon'$.

Case 19. The template is of the form:



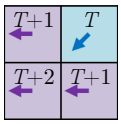
for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = (1+x)\delta \geq \delta > \varepsilon'$.

Case 20. The template is of the form:



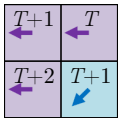
for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = (1+x)\delta \geq \delta > \varepsilon'$.

Case 21. The template is of the form:



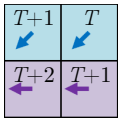
for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = \delta(1+xy) \geq \delta > \varepsilon'$.

Case 22. The template is of the form:



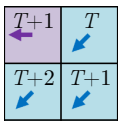
for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = (1+x-xy)\delta \geq \delta > \varepsilon'$.

Case 23. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = (1+y)\delta \geq \delta > \varepsilon'$.

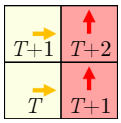
Case 24. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that

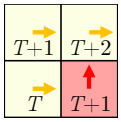
$$\Delta = (2 + (-1 + x)y)\delta \geq \delta > \varepsilon'.$$

Case 25. The template is of the form:



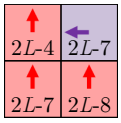
for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = \delta > \varepsilon'$.

Case 26. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that $\Delta = \delta > \varepsilon'$.

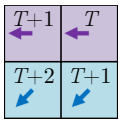
Case 27. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that

$$\begin{aligned}\Delta &= \delta\left(3 + 2x\left(-1 + y^2 + y(-1+\delta)\right) - 2x^2y(-1+y\delta)\right)\\ &\geq \delta\left(3 + 2x\left(-1 + y^2 - y\right)\right)\\ &\geq \delta\left(3 - 2.5x\right)\\ &\geq \delta/2 > \varepsilon'.\end{aligned}$$

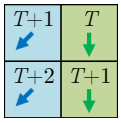
Case 28. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that

$$\Delta = (2-y)\delta \geq \delta > \varepsilon'.$$

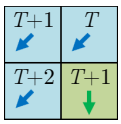
Case 29. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that

$$\Delta = (2-x)\delta \geq \delta > \varepsilon'.$$

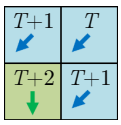
Case 30. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that

$$\Delta = (2 + x(-1+y))\delta \geq \delta > \varepsilon'.$$

Case 31. The template is of the form:



for some $T \in \mathbb{N}$. Plugging these values for the formula for Δ we directly get that

$$\Delta = (1 + x + y - xy)\delta \geq \delta > \varepsilon'.$$

B Algorithms' Pseudocode

```
1: function IsPoint(b_x, b_y, c_x, c_y, q_x, q_y)
2:     if q_x > 1 and q_y > 1 then return No
3:     if b_x = b_y then
4:         if V(c_x) = b_x then
5:             if 10c_y + q_y ≤ 10c_x + 1 and q_x = 1 return Yes
6:         else if V(c_y) = b_x then
7:             if 10c_x + q_x > 10c_y + 1 and q_y = 1 then return Yes
8:         end if
9:     else if b_x − 1 = b_y then
10:        if V(c_y) = b_y then
11:            if 10c_x + q_x ≤ 10S(c_y) + 1 and q_y = 1 then return Yes
12:        else if V(c_x) = b_x then
13:            if 10c_y + q_y > 10P(c_x) + 1 and q_x = 1 then return Yes
14:        end if
15:    else return No
16:    end if
17: end function
```

Algorithm 2: Identifies whether the bottom left corner point of square $(10(c_x-1)+q_x, 10(c_y-1)+q_y)$ in block (b_x, b_y) lies on the line.

```
1: function IsCross(b_x, b_y, c_x, c_y, q_x, q_y)
2:     if IsPoint(b_x, b_y, c_x, c_y, q_x, q_y) then
3:         if IsPoint(b_x, b_y, c_x, c_y, q_x − 1, q_y) and IsPoint(b_x, b_y, c_x, c_y, q_x, q_y + 1) then
4:             if IsPoint(b_x, b_y, c_x, c_y, q_x + 1, q_y) then return Yes
5:             if IsPoint(b_x, b_y, c_x, c_y, q_x, q_y − 1) then return Yes
6:         end if
7:     else return No
8:     end if
9: end function
```

Algorithm 3: Identifies whether the bottom left corner point of square $(10(c_x-1)+q_x, 10(c_y-1)+q_y)$ in block (b_x, b_y) lies on a crossing of two lines.

```
1: function OnTheLine(b_x, b_y, s_x, s_y)
2:     c_x ← 1 + (s_x div 10)
3:     c_y ← 1 + (s_y div 10)
4:     q_x ← 1 + (s_x mod 10)
5:     q_y ← 1 + (s_y mod 10)
6:     if q_x = q_y = 1 then
7:         return IsPoint(b_x, b_y, c_x, c_y, q_x, q_y)
8:     else if IsCross(b_x, b_y, c_x + 1, c_y, 1, 1) then
9:         if q_y = 1 and q_x ≤ 6 then return Yes
10:        if q_x = 6 and q_y ≤ 6 then return Yes
11:        if q_y = 6 and q_x ≥ 6 then return Yes
12:        if q_x = 1 and q_y ≥ 6 and IsCross(b_x, b_y, c_x, c_y, 1, 1) then return Yes
13:        else return No
14:    else if IsCross(b_x, b_y, c_x, c_y, 1, 1) then
15:        if q_x = 1 and q_y ≥ 6 then return Yes
16:        if q_x = 1 and q_y ≤ 5 then return No
17:        if q_y = 1 then return IsPoint(b_x, b_y, c_x, c_y, q_x, q_y)
18:    else return IsPoint(b_x, b_y, c_x, c_y, q_x, q_y)
19:    end if
20: end function
```

Algorithm 4: Identifies whether the bottom left corner point of square (s_x, s_y) in block (b_x, b_y), lies on the line.

```
1: function AboveOrBelow(b_x, b_y, s_x, s_y)
2:     if OnTheLine(b_x, b_y, s_x + 1, s_y) then return Above
3:     else if OnTheLine(b_x, b_y, s_x, s_y + 1) then return Below
4:     else return ⊥
5:     end if
6: end function
```

Algorithm 5: Identifies whether a square (s_x, s_y) in block (b_x, b_y) touches the line from above or below.

```
function Color(b_x, b_y, s_x, s_y)
    if AboveOrBelow(b_x, b_y, s_x, s_y) ≠ ⊥ then
        if AboveOrBelow(b_x, b_y, s_x, s_y) = Above then return Yellow
        else if AboveOrBelow(b_x, b_y, s_x, s_y) = Below then return Red
        end if
    else if AboveOrBelow(b_x, b_y, s_x, s_y − 1)= Above then return Green
    else if AboveOrBelow(b_x, b_y, s_x + 1, s_y − 1)= Above then return Green
    else if AboveOrBelow(b_x, b_y, s_x, s_y + 1)= Below then return Purple
    else if AboveOrBelow(b_x, b_y, s_x − 1, s_y + 1)= Below then return Purple
    else return Blue
    end if
end function
```

Algorithm 6: Assigns the f-color to any square (s_x, s_y) in block (b_x, b_y).

```
function Value(b_x, b_y, s_x, s_y)
    d ← 2^n(b_x − 1 + b_y − 1) + s_x + s_y.
    if Color(b_x, b_y, s_x, s_y) ∈ {Yellow,Red} then return M − 1 + d
    else return 2L − 10 − d.
    end if
end function
```

Algorithm 7: Assigns the p-value to any square (s_x, s_y) in block (b_x, b_y).

C Pseudocode for S' and P' from [BPR15]

For completeness, we provide the pseudocode of S' and P' as given by [BPR15]. Let $(\mathsf{S}, \mathsf{V}, T)$ be an SVL instance and let $t = \lceil \log(T+1) \rceil$. We construct an instance $(\mathsf{S}', \mathsf{P}')$ for the EOL problem where $m = t \cdot (n+t)$. We interpret every node in $\{0,1\}^m$ as a sequence $(u_1, \ldots, u_t)$ of t *states* where, for every $j \in [t]$, the state u_j is of the form $(x, i) \in \{0,1\}^n \times [T]$. We say that a state (x, i) is *valid* if $\mathsf{V}(x, i) = 1$ and denote the i-th valid state $(\mathsf{S}^{i-1}(0^n), i)$ by $v(i)$. Given $u = (x, i)$, we abuse notation (overloading the function S) and denote $\mathsf{S}(u) := (\mathsf{S}(x), i+1)$. Given $u = (x, i)$ and $u' = (x', i')$, we say that $u < u'$ if $i < i'$.

The functions S' and P' are defined below.

```
1: function S′(N = (u_1, ..., u_t))
2:     return S_t((0^n, 1), N)
3: end function
4:
5: function S_1(u_b = (x, i), N = (u_1, ..., u_t))
6:     if N contains an invalid state then
7:         return N unchanged
8:     end if
9:     if u_j is free then
10:        Set u_1 ← S(u_b)
11:        return (u_1, ..., u_t)
12:    else
13:        return N unchanged
14:    end if
15: end function
16:
17: function S_j(u_b = (x, i), N = (u_1, ..., u_t))
18:    if N contains an invalid state then
19:        return N unchanged
20:    end if
21:    if u_j is free then
22:        N′ ← S_{j−1}(u_b, N)
23:        if N′ ≠ N then
24:            return N′
25:        else if for all k ∈ [j − 1], u_k = v(i + 2^{j−1} − 2^{k−1}) then
26:            Set u_j ← S(u_1)
27:            return (u_1, ..., u_t)
28:        end if
29:        return N unchanged
30:    else if u_j = v(i + 2^{j−1}) then
31:        if for every k ∈ [j − 1], u_k is either free or u_k < u_j then
32:            N′ ← P_{j−1}(u_b, N)
33:            if N′ ≠ N then
34:                return N′
35:            else if for all k ∈ [j − 1], u_k is free then
36:                return S_{j−1}(u_j, N)
37:            end if
38:            return N unchanged
39:        else if for every k ∈ [j − 1], u_k is either free or u_k > u_j then
40:            return S_{j−1}(u_j, N)
41:        end if
42:    end if
43:    return N unchanged
44: end function
```

Algorithm 8: The function S'.

```
function P'(N = (u_1, ..., u_t))
    return P_t((0^n, 1), N)
end function

function P_1(u_b = (x, i), N = (u_1, ..., u_t))
    if N contains an invalid state then
        return N unchanged
    end if
    if u_j = v(i + 1) then
        Set u_1 ← v(1)
        return (u_1, ..., u_t)
    else
        return N unchanged
    end if
end function

function P_j(u_b = (x, i), N = (u_1, ..., u_t))
    if N contains an invalid state then
        return N unchanged
    end if
    if u_j is free then
        return P_{j-1}(u_b, N)
    else if u_j = v(i + 2^{j-1}) then
        if for every k ∈ [j − 1], u_k is either free or u_k < u_j then
            N' ← S_{j-1}(u_b, N)
            if N' ≠ N then
                return N'
            else if for all k ∈ [j − 1], u_k = v(i + 2^{j-1} − 2^{k-1}) then
                Set u_j ← v(1)
                return (u_1, ..., u_t)
            end if
            return N unchanged
        else if for every k ∈ [j − 1], u_k is either free or u_k > u_j then
            N' ← P_{j-1}(u_j, N)
            if N' ≠ N then
                return N'
            else if for all k ∈ [j − 1], u_k is free then
                return S_{j-1}(u_b, N)
            end if
        end if
    end if
    return N unchanged
end function
```

Algorithm 9: The pseudocode for circuit P'.

D Cryptographic Definitions

D.1 One-Way Functions

Definition D.1 (One-Way Functions)**.** *A function f is said to be* one-way *if the following two conditions hold:*

1. *There exists a polynomial-time algorithm A such that $A(x) = f(x)$ for every $x \in \{0,1\}^*$.*
2. *For every probabilistic polynomial-time algorithm A and all sufficiently large n,*

$$\Pr[A'(1^n, f(x)) \in f^{-1}(f(x))] < \mathsf{neg}(n),$$

where the probability is taken uniformly over all possible $x \in \{0,1\}^n$ and the internal randomness of A'.

A one-way function f is said to be a one-way permutation if it is also a permutation.

D.2 Obfuscation

We say that two circuits C and C' are *equivalent* and denote it by $C \equiv C'$ if they compute the same function (i.e., $\forall x : C(x) = C'(x)$).

Indistinguishability Obfuscation

Definition D.2 (Perfect/Imperfect Indistinguishability Obfuscator)**.** *Let $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ be a class of polynomial-size circuits, where $\mathcal{C}_n$ is a set of circuits operating on inputs of length n. A uniform algorithm $i\mathcal{O}$ is called an (*imperfect*) indistinguishability obfuscator for the class $\mathcal{C}$ if it takes as input a security parameter and a circuit in $\mathcal{C}$ and outputs a new circuit so that following properties are satisfied:*

1. *(Perfect/Imperfect) Preserving Functionality:*

   There exists a negligible function α such that for any input length $n \in \mathbb{N}$, any λ and any $C \in \mathcal{C}_n$ it holds that

$$\Pr_{i\mathcal{O}}\left[C \equiv i\mathcal{O}(1^\lambda, C)\right] \geq 1 - \alpha(\lambda),$$

   where the probability is over the internal randomness of $i\mathcal{O}$. If $\alpha(\cdot) = 0$, then we say that $i\mathcal{O}$ is perfect*.*

2. *Polynomial Slowdown:*

   There exists a polynomial $p(\cdot)$ such that: For any input length $n \in \mathbb{N}$, any λ and any circuit $C \in \mathcal{C}_n$ it holds that $\left|i\mathcal{O}(1^\lambda, C)\right| \leq p(|C|)$.

3. *Indistinguishable Obfuscation:*

   For any probabilistic polynomial-time algorithm D, any $n \in \mathbb{N}$, any two equivalent *circuits $C_1, C_2 \in \mathcal{C}_n$ of the same size and large enough λ, it holds that*

$$\left|\Pr_{i\mathcal{O},D}\left[D\left(i\mathcal{O}\left(1^\lambda, C_1\right)\right) = 1\right] - \Pr_{i\mathcal{O},D}\left[D\left(i\mathcal{O}\left(1^\lambda, C_2\right)\right) = 1\right]\right| \leq \mathsf{neg}(\lambda).$$

We say that $i\mathcal{O}$ is efficient *if it runs in polynomial-time.*

Virtual Black-Box Obfuscation

Definition D.3 (Perfect/Imperfect VBB Obfuscator)**.** *Let $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ be a class of polynomial-size circuits, where $\mathcal{C}_n$ is a set of circuits operating on inputs of length n. A uniform algorithm $\mathcal{O}$ is called an (*imperfect*) VBB obfuscator for the class $\mathcal{C}$ if it takes as input a security parameter and a circuit in $\mathcal{C}$ and outputs a new circuit so that following properties are satisfied:*

1. *(Perfect/Imperfect) Preserving Functionality:*

   There exists a negligible function α such that for any input length $n \in \mathbb{N}$, any λ and any $C \in \mathcal{C}_n$ it holds that

   $$\Pr_{\mathcal{O}}\left[C \equiv \mathcal{O}(1^\lambda, C)\right] \geq 1 - \alpha(\lambda),$$

   where the probability is over the internal randomness of $\mathcal{O}$. If $\alpha(\cdot) = 0$, then we say that $\mathcal{O}$ is perfect*.*

2. *Polynomial Slowdown:*

   There exists a polynomial $p(\cdot)$ such that: For any input length $n \in \mathbb{N}$, any λ and any circuit $C \in \mathcal{C}_n$ it holds that $\left|\mathcal{O}(1^\lambda, C)\right| \leq p(|C|)$.

3. *Virtual Black-Box:*

   For any probabilistic polynomial-time algorithm D, any predicate $\pi : \mathcal{C}_n \to \{0,1\}$, any $n \in \mathbb{N}$ and any circuit $C \in \mathcal{C}_n$, there is a polynomial-size simulator S such that for large enough λ it holds that

   $$\left|\Pr_{\mathcal{O},D}\left[D\left(\mathcal{O}\left(1^\lambda, C\right)\right) = \pi(C)\right] - \Pr_{S}\left[D\left(S^C\left(1^\lambda\right)\right) = \pi(C)\right]\right| \leq \mathsf{neg}(\lambda).$$

We say that $\mathcal{O}$ is efficient if it runs in polynomial-time.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de