The Computational Complexity of Finding Second-Order Stationary Points

Andreas Kontogiannis [*1,2] **Vasilis Pollatos** [*2] **Sotiris Kanellopoulos** [1,2]
Panayotis Mertikopoulos [3] **Aris Pagourtzis** [1,2] **Ioannis Panageas** [4]

Abstract

Non-convex minimization problems are universally considered hard, and even guaranteeing that a computed solution is locally minimizing is known to be NP-hard. In this general context, our paper focuses on the problem of finding stationary points that satisfy an approximate second-order optimality condition, which serves to exclude strict saddles and other non-minimizing stationary points. Our main result is that the problem of finding approximate second-order stationary points (SOSPs) is PLS-complete, i.e., of the same complexity as the problem of finding first-order stationary points (FOSPs), thus resolving an open question in the field. In particular, our results imply that, under the widely believed complexity conjecture that PLS $\neq$ FNP, finding approximate SOSPs in unconstrained domains is *easier* than in constrained domains, which is known to be NP-hard. This comes in stark contrast with earlier results which implied that, unless PLS = CLS, finding approximate FOSPs in unconstrained domains is *harder* than in constrained domains.

1 Introduction

Background and motivation. The vast majority of machine learning models (from logistic regression to empirical risk minimization and training deep neural networks) ultimately boils down to solving a continuous optimization problem of the form

$$\underset{x\in\mathcal{X}}{\text{minimize}}\ f(x) \tag{Opt}$$

where $\mathcal{X}$, the problem's *search domain*, is a subset of $\mathbb{R}^d$ (often $\mathbb{R}^d$ itself) and $f\colon \mathcal{X} \to \mathbb{R}$ is the problem's *objective function*. In many – if not most – applications, f is high-dimensional and innately non-convex, thus giving rise to a very rich, diverse and challenging optimization landscape, often with an exponential number of saddle points and other spurious solutions.

The central question in the field is the design and analysis of optimization algorithms that can solve (Opt) efficiently. However, since finding even a local minimum of f is well-known to be nondetermistic polynomial time (NP)-hard [27], the focus of most optimization algorithms is to reach a critical point of f, or even an approximation thereof. Here, depending on the gradient information available to the optimizer, standard gradient methods with black-box access to a first-order oracle output an ε-approximate critical point in time which ranges between $\mathcal{O}(1/\varepsilon^2)$ and $\mathcal{O}(1/\varepsilon^4)$ in the deterministic and stochastic cases respectively, and with a polynomial dependence on the dimension, cf. [21] and references therein.

A key obstacle in the above is the existence of stationary points of negative curvature, i.e., points where the Hessian has at least one negative eigenvalue. Such points are commonly referred to as (strict) *saddle points*, and they have long been regarded as a major obstacle for solving non-convex optimization problems in continuous domains. In fact, in many applications and cases of practical interest, it is well known that the number of saddle points significantly exceeds the number of local minimizers [33], and solutions associated with worst-case saddle points are considerably worse than those associated with worst-case local minima [30]. And, even though it has been shown that gradient descent with random initialization or injected random noise can provably avoid saddle points [15, 19, 23], the computational complexity of finding stationary points that do not have negative curvature – i.e., *second-order stationary points* (SOSPs) – remains unknown.

We are thus led to the following natural question, which is the overarching motivation of our work:

What is the computational complexity of finding a second-order stationary point?

[*]Equal contribution [1]National Technical University of Athens, School of Electrical and Computer Engineering, Athens, Greece. [2]Archimedes/Athena RC, Greece. [3]Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France. [4]University of California, Irvine, USA. Correspondence to: Andreas Kontogiannis <a.kontogiannis@athenarc.gr>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

Our contributions in the context of related work. To put this question on a formal footing that also accounts for the bit complexity of function queries, we focus on a white-box description of the problem in terms of polynomial-time Turing machines and arithmetic circuits for representing functions, as in [11, 13, 16]. In this context, the corresponding question for computing (approximate) Karush–Kuhn–Tucker (KKT) points in problems with a compact domain was recently shown to be complete in the class PPAD ∩ PLS [13], which was shown in the same paper to be equal to the class CLS introduced by [10].

Moving to the unconstrained setting, conventional wisdom in optimization would suggest that the problem of finding an approximate first-order stationary point (FOSP) is easier than in the constrained case, because the optimizer does not need to worry about the subtleties of having to satisfy an added set of convex inequalities. However, in a very recent paper, Hollender & Zampetakis [16] showed that finding an approximate FOSP in the unconstrained case is, in fact, PLS-complete, and hence presumably *harder* than the corresponding constrained version of the problem (which, as we mentioned above is CLS-complete, and CLS is widely believed to be strictly included in PLS).

By contrast, the second-order landscape is decidedly different: In particular, as was shown in [31], the computational complexity of finding an approximate SOSP over a compact polyhedron is NP-hard. This is (presumably) a huge jump in complexity, but seeing as many problems of interest in machine learning and data science are formulated as *unconstrained* optimization problems (i.e., $\mathcal{X} = \mathbb{R}^d$), the above result cannot be applied in many cases of practical interest. Accordingly, our paper seeks to fill this gap by determining the computational complexity of finding approximate SOSPs in the unconstrained version of (Opt), a question that was stated as an open problem in [16].

In view of the increase in hardness from constrained to unconstrained problems in the first-order setting – that is, from CLS to PLS – one might expect that finding an approximate SOSP in the unconstrained version of (Opt) must also be NP-hard. However, our first result below show that this intuition is *false:*

Theorem (Informal version of Theorem 1)**.** *Given a twice differentiable function f, bounded, with Lipschitz gradient and Hessian, the problem of computing an approximate second-order stationary point of f lies in PLS.*

This result goes against the first-order computational complexity trend, but it reconnects with the familiar intuition that unconstrained instances of (Opt) are, in general, considerably easier than its constrained instances; as such, the natural question that arises is whether the membership result above is tight. On the one hand, the PLS-completeness result of [16] for finding approximate FOSPs would suggest that the problem of finding an approximate SOSP is at least as hard, in which case our PLS membership result is tight, and the problem is PLS-complete. On the other hand, since we assume second-order smoothness (potentially accessible via a Turing machine), it is not clear if this is a valid comparison: indeed, if f is convex, Monteiro and Svaiter [25] showed that an approximate minimizer of f can be computed in $\mathcal{O}(1/\varepsilon^{2/7})$ oracle calls, improving in this way by almost two orders of magnitude the corresponding $\mathcal{O}(1/\sqrt{\varepsilon})$ rate which is known to be tight for first-order algorithms under first-order smoothness. Thus, when more information and structure is present, and working in a setting that is not complicated by constraint satisfaction issues, the actual hardness of the problem could be – and, in fact, could be *expected* to be – lower than PLS.

Problem	Unconstrained	Constrained
ε-FOSPs	PLS-complete [16]	CLS-complete [13]
ε-SOSPs	**PLS-complete**$^\sharp$	NP-hard [31]

Table 1: The computational complexity results of finding first and second-order stationary points. Recall that CLS ⊆ PLS ⊆ TFNP ⊆ FNP. The inclusions are believed to be strict. (FNP has problems whose decision versions lie in NP). ♯Our work.

Our second result shows that this is not the case either:

Theorem (Informal version of Theorem 2)**.** *Given a twice differentiable function f, bounded, with Lipschitz gradient and Hessian, the problem of computing an approximate second-order stationary point of f is PLS-hard.*

In fact, we have shown a stronger result: The problem of finding an approximate FOSP remains hard for PLS even with the extra assumption of Hessian Lipschitzness and second-order oracles. In view of all this, our main contribution can be summarized as follows:

The problem of finding an approximate SOSP in unconstrained problems is PLS-complete.

We demonstrate the above complexity results in Table 1.

Technical overview. Our work focuses throughout on the computational complexity of finding SOSPs as defined formally in Problem 4) in unconstrained optimization and is split into two parts: First, we show the problem is in PLS (Theorem 1) and, subsequently, we show that the problem is hard for PLS (Theorem 2). In a nutshell, we illustrate our reductions in Fig. 1.

For the membership in PLS, we show that the SOSP problem reduces to LocalOpt (see Problem 1), the problem defining the PLS class. First, we examine the LocalOpt instance used in [16] to prove the PLS membership of the problem of finding first-order stationary points (denoted

Figure 1: The problem reductions used in our work.

by FOSP, see Problem 3). We show that there exist SOSP instances where a solution of LocalOpt is not a solution of SOSP. The cause of this failure is the fact that this LocalOpt instance does not use any second-order information for the function for which we aim to solve SOSP. To overcome this issue, we allow the neighbor function of LocalOpt to alternate between steepest coordinate descent and negative curvature directions (motivated by a standard algorithm for finding SOSPs, see Algorithm 1), both of which are shown to decrease the potential function (see Lemma 1 and Lemma 2). A key point is that LocalOpt can only be defined on a bounded subdomain of the input function of SOSP. To achieve this, we center LocalOpt around zero and exclude from the search space the points that cannot be reached from Algorithm 1 when starting at zero (see Lemma 3).

To prove PLS-hardness, we make a reduction from Iter (a PLS-complete problem, cf. Problem 2) to Hessian–FOSP (Problem 5), a problem identical to FOSP but with the extra requirement for Hessian Lipschitzness. Since Hessian–FOSP trivially reduces to SOSP, we derive the PLS-hardness of SOSP. For the reduction we embed the Iter instance into a 2D-grid and then define a smooth C^2 continuous function over the plane of the grid by interpolating between the values of the grid points. Analogously to [16], the values on the grid points are chosen in such a way that the resulting function could only have stationary points in areas close to the solutions of the embedded Iter instance. However, the standard bicubic interpolation method used in previous works [13, 16] fails to preserve the required second order continuity. We overcome this issue by using a significantly more involved *biquintic* interpolation scheme (Lemma 4), for which we show that no unwanted stationary points are introduced and thus Hessian–FOSP solutions can only occur in the areas of Iter solutions (Lemma 5).

Further related work. Many algorithms have been proposed for computing ε-approximate first-order stationary points for a given f. Those algorithms are allowed to get evaluations from f and its gradients and typically have running time that is polynomial in $1/\varepsilon$ and the dimension d of the domain of f. Probably the most well-known method that runs in time $\mathcal{O}\left(1/\varepsilon^2\right)$ and is dimension-free, as long as f is smooth, is Gradient Descent [28]. With the extra assumption that f has Lipschitz Hessian, faster algorithms have been proposed [1, 6] and achieve running time $\mathcal{O}\left(\log(1/\varepsilon^{7/4})\right)$. These works have been improved further in [5]. Last but not least, if second-order information is used, Nesterov and Polyak [29] designed an algorithm that can achieve even faster rates via cubic regularization (see also [8, 37] for other works). Moreover, a significant corpus of work in the literature has focused on providing guarantees for avoiding spurious, non-minimizing first-order stationary points of f. Depending on the context, these avoidance results may concern deterministic [12, 22, 32], stochastic [9, 17, 18, 24, 34], and even adaptive algorithms (like AdaGrad and its variants) [3, 36]. Last but not least, aiming to find better solutions when second-order information is not enough, other approaches require higher-order derivatives [4, 7].

As for relevant computational complexity results, Daskalakis et. al [11] showed that the linearly constrained min-max optimization problem with nonconvex-nonconcave objectives is PPAD-complete. Fearnley et. al showed that any problem that can be solved by performing Gradient Descent on a bounded convex polytopal domain is CLS-complete [13]. A more recent result is that the problem of finding a KKT point of a quadratic polynomial over box constraints is also CLS-complete [14]. As for higher-order optimization results, the problem of finding fourth-order stationary points has been proven to be NP-hard [2].

2 Preliminaries

Our goal in this section is to describe our blanket assumptions for f and to introduce some basic elements from complexity theory that we will need throughout the sequel. In terms of notational conventions, we will write $[\![a, b]\!]$ for the set of all integers between $a, b \in \mathbb{R}$, $[2^n]$ as a shorthand for $\{0,1\}^n$, and $\text{len}(x)$ for the number of bits in the binary representation of $x \in \mathbb{R}$. Also, given a function $f\colon \mathbb{R}^d \to \mathbb{R}$, we will write $u_1(x)$ for any eigenvector corresponding to the minimum eigenvalue $\lambda_1 \equiv \lambda_{\min}(\nabla^2 f(x))$. Finally, we will write $\|\cdot\|_2$ for the ordinary Euclidean norm on $\mathbb{R}^d$ and the Frobenius norm on $\mathbb{R}^{d\times d}$.

2.1. Blanket assumptions and definitions.

We begin by defining the function class that we will work with in the rest of our paper.

Assumption 1. The objective function $f\colon \mathbb{R}^d \to \mathbb{R}$ of (Opt) satisfies the following conditions for some $B, L, \rho > 0$ and for all $x, y \in \mathbb{R}^d$:

(a) *Boundedness:*

$$|f(x)| \le B \tag{1a}$$

(*b*) *Smoothness:*

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \tag{1b}$$

(*c*) *Second-order smoothness: (Hessian Lipschitzness)*

$$\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq \rho\|x - y\| \tag{1c}$$

These assumptions are the same as in e.g., [16], and they are fairly standard in the field of white-box complexity; for convenience, we will denote this class as $\mathcal{F} \equiv \mathcal{F}(B, L, \rho)$.

Now, to access f and its derivatives, we will assume that the optimizer has access to polynomial-time Turing machines $\mathcal{C}_f$, $\mathcal{C}_{\nabla f}$ and $\mathcal{C}_{\nabla^2 f}$ which, when queried at $x \in \mathbb{R}^d$, respectively return $f(x)$, $\nabla f(x)$ and $\nabla^2 f(x)$. More precisely, given a query point $x \in \mathbb{R}^d$ that is representable in $\text{len}(x) = b$ bits, a Turing machine runs in time that is upper-bounded by some polynomial in b and outputs approximate values for $f(x)$, $\nabla f(x)$ or $\nabla^2 f(x)$, depending on the context. Note that the gradients and Hessians can be estimated with arbitrary accuracy using zero order oracles and the finite difference method, but for simplicity we will assume that they will be directly computed by Turing machines. Finally, as is standard in the white box setting, we will assume that the above is encoded by Boolean circuits $C\colon [2^n] \to [2^n]$ with n inputs and n outputs, and that are allowed to use the logic gates AND, OR and NOT. Otherwise, if the bit complexity is not relevant and the optimizer has access to a mechanism that can output $f(x)$, $\nabla f(x)$ and $\nabla^2 f(x)$ to perfect accuracy in $\mathcal{O}(1)$ time, we will refer to the model as a *black-box setting*.

With all this in hand, the solution concepts that we will focus on are as follows:

Definition 1 (Approximate stationarity)**.** Let $f\colon \mathbb{R}^d \to \mathbb{R}$ be a function in $\mathcal{F}(B, L, \rho)$, and fix some precision target $\varepsilon > 0$. Then a point $x^* \in \mathbb{R}^d$ is said to be an:

1. *ε-first-order stationary point* (ε-FOSP) if $\|\nabla f(x^*)\| \leq \varepsilon$.
2. *ε-second-order stationary point* (ε-SOSP) if it is an ε-FOSP and, in addition, $\lambda_{\min}(\nabla^2 f(x^*)) \geq -\sqrt{\rho\varepsilon}$.

The definition of ε-FOSPs is standard in the literature and, in the unconstrained case, it coincides with the definition of an ε-KKT point, so it is a measure of approximate criticality. The definition of an ε-SOSP implies that the minimum eigenvalue of $\nabla^2 f(x)$ has no arithmetically significant negative part, so it serves to exclude strict saddle points (points with at least one negative Hessian eigenvalue) and, as such, it can be seen as an indicator of local optimality. Historically, this measure was first introduced by [29], and the terminology second-order stationary point seems to be due to [19]. We note however that, even if x^* is an ε-SOSP for all $\varepsilon > 0$, it may still be non-minimizing (for example, if $f(x) = -x^3$ and $x^* = 0$); however, the value of f in the vicinity of x^* will grow at most as $o(\|x - x^*\|^2)$, which is presumed acceptable to second order. In particular, as we will see in Section 3, an important property of SOSPs for a function $f \in \mathcal{F}(B, L, \rho)$ is that (Opt) always admits an ε-SOSP with bit representation $\text{poly}(\log(B, L, \rho, 1/\varepsilon))$.

2.2. Basic notions from complexity theory. By our blanket assumptions, it follows that (Opt) always exists a solution, so the problem belongs to the total search problems that are captured by subclasses of the class TFNP. In particular, TFNP is a subclass of the FNP class, which contains all search problems where all solutions have size polynomial in the size of the instance and any solution can be checked in polynomial time. In other words, FNP contains search problems whose decision version lies in NP. Since a problem in TFNP always has a solution, then its decision version cannot be NP-hard unless NP =co-NP [13, 20].

In this paper, we are interested in the class PLS (Polynomial Local Search), as defined in [20]. In particular, PLS is defined as the set of all TFNP problems that reduce in polynomial time (see Appendix A.6) to the PLS-complete problem LocalOpt below:

Problem 1: LocalOpt

Require: Boolean circuits $N, P\colon [2^n] \to [2^n]$

\# N: neighbor function; P: potential function

1: **Find:** $v \in [2^n]$ such that $P(N(v)) \geq P(v)$

As a problem, LocalOpt embodies local search over the node set $[2^n]$ and captures the problem of unconstrained minimization over a discrete domain. The output of the circuit P represents here a potential value, so our goal is to find a node $v \in [2^n]$ that minimizes $P(v)$. The circuit N represents the search for an improving node in some small (polynomial-size) neighbourhood. The procedure terminates by finding $v \in [2^n]$ such that $P(N(v)) \geq P(v)$.

In the sequel, we will also require another PLS-complete problem known as Iter [26]. This is defined as follows:

Problem 2: Iter

Require: Boolean circuit $C\colon [2^n] \to [2^n]$ with $C(1) > 1$

Find: $v \in [2^n]$ such that either

$C(v) < v$ or $C(v) > v$ and $C(C(v)) = C(v)$.

In Iter, the nodes in $[2^n]$ lie on a line from left to right, and the goal is to find any node v that is mapped to the left by C, or any node v that is mapped to the right such that $C(C(v)) = C(v)$; in other words, $C(v)$ is a fixed point of C. Since $C(1) > 1$, it is easy to verify that Iter is total because a solution exists in any instance of Iter.

Finally, we also define FOSP and SOSP, the computational problem of finding ε-FOSPs and ε-SOSP in unconstrained optimization, see below.

Problem 3: FOSP

Require: precision parameter $\varepsilon > 0$
$B, L > 0$ and $f \in \mathcal{F}(B, L, \infty)$
Turing machines representing f and ∇f

Find: $x^* \in \mathbb{R}^d$: $\|\nabla f(x^*)\| \leq \varepsilon$

Problem 4: SOSP

Require: precision parameter $\varepsilon > 0$
$B, L, \rho > 0$ and $f \in \mathcal{F}(B, L, \rho)$
Turing machines representing f, ∇f and $\nabla^2 f$

Find: $x^* \in \mathbb{R}^d$: $\|\nabla f(x^*)\| \leq \varepsilon$, $\lambda_1(\nabla^2 f(x^*)) \geq -\sqrt{\rho\varepsilon}$

As shown in the main results of [16], FOSP is PLS-complete. Finally, SOSP captures the computational complexity problem for finding an ε-SOSP in the unconstrained optimization setting under the assumptions we discussed in Section 2.1, and it will be our primary focal point in the sequel.

3 PLS-membership of Finding ε-SOSPs in Unconstrained Optimization

In this section, we show the membership of SOSP in PLS. We summarize the main result in the following theorem.

Theorem 1 (Membership)**.** *SOSP is in PLS.*

3.1. Warm Up: An example that the LOCALOPT instance used in [16] fails. Before showing the membership of SOSP in PLS, we examine the LOCALOPT instance described in [16], used for showing the membership of FOSP in PLS. We show a counterexample function where a solution of this LOCALOPT instance is an ε-FOSP but not an ε-SOSP of the function.

To show the membership of FOSP in PLS, [16] constructed a LOCALOPT instance within an orthocanonical grid of 2^n discrete points (values of LOCALOPT), centered at $(0, 0)$, with step γ (without loss of generality, consider that $\gamma = 1$), where for any function f: (a) the potential function at a valid point v of the grid equals $f(v)$, and (b) the neighbor function of v equals the immediate neighbor of that point on the grid whose value is less than $f(v)$ and is also the minimum among the values of all candidate immediate neighbors.

We now aim to show that the above LOCALOPT instance is

Algorithm 1: An algorithm for finding an ε-SOSP on an ortho-canonical grid with step γ

Require: f, ∇f and $\nabla^2 f$, $\gamma = \min\left\{\frac{\varepsilon}{2\sqrt{d}(\sqrt{\varepsilon}\sqrt{\rho}+2L)}, \frac{\varepsilon}{\sqrt{d}L}\right\}$, B, $L, \rho, d, r = \frac{3\sqrt{\varepsilon}}{\sqrt{\rho}}$.

$t \leftarrow 0$
while TRUE **do**
  $\lambda_1 \leftarrow \lambda_{min}(\nabla^2 f(x_t))$
  if $\|\nabla f(x_t)\| > \varepsilon$ **then**
    $i \leftarrow \arg\min_{i\in[d], s\in\{0,1\}} \langle \nabla f(x_t), (-1)^s e_i \rangle$
    $x_{t+1} \leftarrow x_t + \gamma \cdot (-1)^s e_i$
  else if $\|\nabla f(x_t)\| \leq \varepsilon$ **and** $\lambda_1 < -\sqrt{4\rho\varepsilon}$ **then**
    $u_1 \leftarrow$ the eigenvector corresponding to λ_1
    $\tilde{x}_t \leftarrow x_t + r u_1$
    $x_{t+1} \leftarrow \text{round}(\tilde{x}_t)$
  else
    return x_t
  end if
  $t \leftarrow t + 1$
end while

a counterexample for the SOSP problem. We define:

$$f(x, y) = \begin{cases} x^2 y - xy & x, y \in [-1, 1] \\ -1 & x, y \in (-\infty, -2] \cup [2, \infty) \\ g(x, y) & \text{otherwise} \end{cases}$$

where $g(x, y)$ is a C^2 function, which is derived by applying *biquintic interpolation* on each grid cell of the defined region. Biquintic interpolation will help us show our hardness theorem, which we will describe in detail in Section 4. We note that biquintic interpolation guarantees that f is also a C^2 function, with Lipschitz gradient and hessian, thus satisfying the input conditions described in Problem 4. We describe and illustrate the above function in Appendix D.

Now, observe that $(0, 0)$ is a solution of FOSP, since $\nabla f(0, 0) = 0$ but not a solution of SOSP because: $\lambda_{min}(\nabla^2(f(0, 0))) = -1$ and the Hessian-Lipschitz constant $\rho = 2\sqrt{2}$. Hence, by letting $\varepsilon = 0.001$, we get $\lambda_{min}(\nabla^2(f(0, 0))) < -\sqrt{\rho\epsilon}$. Now, observe that each neighbor of $(0, 0)$ (i.e. $(0, 1)$, $(0, -1)$, $(1, 0)$ and $(-1, 0)$) has potential 0, which is equal to the potential of $(0, 0)$ and thus $(0, 0)$ is a solution of FOSP, but not a solution of SOSP.

Given the failure of the above LOCALOPT instance, in order to show the membership of SOSP in PLS we need the reduction to construct a new LOCALOPT instance where every solution of that instance is always a solution of SOSP.

3.2. Warm Up: An algorithm that finds an ε-SOSP. First, we present an algorithm (presented in Algorithm 1), indicative of the methodology we follow to show the membership of SOSP in PLS. The algorithm runs on an orthocanonical grid with step γ and finds a grid point that is an ε-SOSP. Depending on the current point x_t, similar to

[8, 19], the algorithm alternates between steepest coordinate descent and negative curvature steps as follows: If the current point x_t is not an ε-FOSP, as in [16], we take a step along the coordinate e_i that guarantees sufficient decrease in the value of f (Steps 4-6). If the algorithm has reached an ε-FOSP, we check the condition for ε-SOSP by computing the minimum eigenvalue (denoted by λ_1) of the Hessian on x_t (Step 8). If the condition is met, the algorithm first follows the direction of the negative curvature on x_t (Steps 9-10) and then performs rounding on the resulted point, so that each coordinate of the point is rounded down to the closest multiple, α, of γ (Step 10). Otherwise, the algorithm has reached an ε-SOSP and terminates (Step 12).

Algorithm 1 will be helpful for proving the membership of SOSP in PLS, as it finds an ε-SOSP in time $\mathcal{O}(1/\varepsilon\sqrt{\varepsilon})$ and also guarantees that both update steps sufficiently decrease the value of f. The latter indicates that the objective function is a potential and that there exists an algorithm that decreases the potential at each step.

3.3. Main Components of the Proof of Theorem 1. In this section, we will define the main components that we need for the proof of Theorem 1.

3.3.1 Descent Lemmas.

We start with the following lemmas that analyse the descent steps of Algorithm 1. These lemmas will later constitute our main arguments for showing that any solution of LocalOpt is a solution of SOSP.

Lemma 1 (Steepest Coordinate Descent). *If $\|\nabla f(x_t)\| > \varepsilon$, then for: $\gamma = \min\left\{\frac{\varepsilon}{2\sqrt{d}(\sqrt{\varepsilon}\sqrt{\rho}+2L)}, \frac{\varepsilon}{\sqrt{d}L}\right\}$, $i, s = \arg\min_{i\in[d],s\in\{0,1\}} \langle \nabla f(v), (-1)^s e_i\rangle$ and $x_{t+1} = x_t + \gamma \cdot (-1)^s e_i$, we have $f(x_{t+1}) \le f(x_t) - \frac{\varepsilon\gamma}{2\sqrt{d}}$.*

Lemma 2 (Negative Curvature Descent). *If $\|\nabla f(x_t)\| \le \varepsilon$, then for: $\lambda_{min}(\nabla^2 f(x_t)) < -\sqrt{4\rho\varepsilon}$, $r = \frac{3\sqrt{\varepsilon}}{\sqrt{\rho}}$, $\tilde{x}_t = x_t + r u_1$ and $x_{t+1} = round(\tilde{x}_t)$, we have $f(x_{t+1}) \le f(x_t) - \frac{3\varepsilon\sqrt{\varepsilon}}{4\sqrt{\rho}}$.*

3.3.2 Construction of the LocalOpt instance.

Now we are ready to construct the LocalOpt instance, where we will next show that any solution of the LocalOpt instance must always be a solution of SOSP.

Grid. First, we define the following quantities: $\mathcal{T}_{NC} = \frac{4B\sqrt{\rho}}{3\varepsilon\sqrt{\varepsilon}}$, $\mathcal{T}_{CD} = \frac{2B\sqrt{d}}{\gamma\varepsilon}$ and $\tilde{R} = \gamma\mathcal{T}_{CD}(\mathcal{T}_{NC}+1) + r\mathcal{T}_{NC}$. Specifically, Algorithm 1 needs at most $\mathcal{T}_{CD}$ steps to find an ε-FOSP starting from an arbitrary non ε-FOSP, and at most $\mathcal{T}_{NC}$ steps to escape from a point that is an ε-FOSP but not an ε-SOSP. Therefore, the algorithm always terminates at a point which is an ε-SOSP and the maximum distance from the starting point x_0 that the algorithm can reach is at most $\tilde{R} = \gamma\mathcal{T}_{CD}(\mathcal{T}_{NC}+1) + r\mathcal{T}_{NC}$.

Let $\gamma = \min\left\{\frac{\varepsilon}{2\sqrt{d}(\sqrt{\varepsilon}\sqrt{\rho}+2L)}, \frac{\varepsilon}{\sqrt{d}L}\right\}$, $m = \left\lceil \frac{R}{\gamma} \right\rceil$ and $R = \max\left\{\frac{10B\sqrt{d}}{\varepsilon}, \frac{10B(r+\gamma\sqrt{d})}{\varepsilon r}, \tilde{R}\right\} = m\gamma$. We define the following grid of $[-R, R]^d$:

$$G_\gamma = \left\{\gamma \cdot a \mid a \in [\![-m, m]\!]^d\right\}$$

More specifically, G_γ is the ortho-canonical grid of $[-R, R]^d$ with step γ between two neighboring vertices of the grid. The following proposition guarantees that the defined grid always contains a solution of SOSP.

Proposition 1 (Feasibility). *G_γ has at least one ε-SOSP.*

From the definition of SOSP we are given a Turing machine $\mathcal{C}_f$ that on input x runs in time $\text{poly}(\text{len}(x))$ and outputs the value of $f(x)$. For every $v \in G_\gamma$ we define $\Gamma(v)$ to be the set of immediate neighbours of v as follows

$$\Gamma(v) = \left\{w \in G_\gamma \mid v \ne w, \|u - w\| = \gamma\right\}.$$

It is clear that $|\Gamma(v)| \le 2d$, where we have exact equality for all the points of the grid in the interior of the box $[-R, R]^d$ and inequality for the points on the boundary.

Valid and Invalid points. We define the set of *valid* points $\mathcal{V}_\gamma$, which is a subset of the grid points G_γ, as follows:

Definition 2 (Valid points). A grid point $v \in G_\gamma$ is valid, i.e. $v \in \mathcal{V}_\gamma$, if:

$$f(v) \le f(0) - \min\left\{\frac{\varepsilon}{2\sqrt{d}}, \frac{\varepsilon r}{4(r+\gamma\sqrt{d})}\right\}\|v\|.$$

where $r = \frac{3\sqrt{\varepsilon}}{\sqrt{p}}$. We also define the set of *invalid* points $\mathcal{I}_\gamma$ to be $\mathcal{I}_\gamma = G_\gamma \backslash \mathcal{V}_\gamma$. Intuitively, *invalid* are the points on the grid which Algorithm 1 cannot reach when starting at zero; otherwise the algorithm would have achieved a total decrease resulting in a value less than $-B$. We use Definition 2 to properly bound the feasible search space.

Proposition 2 (Boundary). *All the points of G_γ that lie on the boundary of $[-R, R]^d$ are invalid. Thus, for every valid point $v \in \mathcal{V}_\gamma$ it holds that $|\Gamma(v)| = 2d$.*

Our next goal is to define the neighbor function $N : G_\gamma \to G_\gamma$ and the potential function $P : G_\gamma \to [-B-1, B+1]$ such that every point $x \in G_\gamma$ with $P(N(x)) \ge P(x)$ is a solution of SOSP for function f.

The neighbor function N. First we define the function $Df(v, s, u) = \langle \nabla f(v), (-1)^s \frac{u-v}{\|u-v\|}\rangle$, with $s \in \{0, 1\}$, which is the directional derivative of f at v in the direction of unit

vector $(-1)^s \frac{u-v}{\|u-v\|}$. Then, we define the neighbor function N on the valid points of the grid, as follows:

$$N(v) = \begin{cases} \arg\min_{u \in \Gamma(v)} \min_{s \in \{0,1\}} Df(v,s,u) & \|\nabla f(v)\| > \varepsilon \\ \text{round}(v + r u_1(v)) & \|\nabla f(v)\| \leq \varepsilon \\ & \lambda_1(v) < -\sqrt{4\rho\varepsilon} \end{cases}$$

and also $N(v) = 0$ if v is invalid. In the above definition, we have used the notation $\lambda_1(v)$ for the minimum eigenvalue of the Hessian of f at v and $u_1(v)$ for the corresponding eigenvector. Note that it is allowed for the neighbor function N to have access to $\nabla f(v)$ and $\nabla^2 f(v)$, as this happens only when v is valid because it is necessary to hold $P(v) = f(v)$.

Lemma 3 (Validity)**.** *If point $v \in \mathcal{V}_\gamma$, then $N(v) \in \mathcal{V}_\gamma$.*

The potential function P. For a valid point v we define $P(v)$ to be the output of the Turing machine $\mathcal{C}_f$ on v, i.e., $P(v) = f(v)$. For any $v \in \mathcal{I}_\gamma$, we define $P(v) = B + 1$.

3.4. Proof Sketch of Theorem 1.

Proof. Using the components we defined in Section 3.3, we will show that that any solution of the LOCALOPT instance with inputs P, N yields a solution of the SOSP problem.

Let $v \in G_\gamma$. We will show that if v is not a solution of SOSP, then $P(N(v)) < P(v)$. Specifically, the following hold:

- If v is invalid then v cannot be a solution of the defined LOCALOPT instance, since $P(v) = B + 1$ and $N(v) = 0$, and thus $P(0) \leq B < B + 1 = P(V)$.
- If v is valid and $\|\nabla f(v)\| > \varepsilon$, then there exist $i \in [d]$ and $s \in \{0, 1\}$ such that $\langle \nabla f(v), (-1)^s e_i \rangle \leq -\varepsilon/\sqrt{d}$, where e_i is the unit vector along the selected coordinate. From Proposition 2, $N(v) \in \Gamma(v)$ and since $N(v) \equiv \arg\min_{u \in \Gamma(v)} \min_{s \in \{0,1\}} Df(v, s, u)$, then we get $N(v) \leq \langle \nabla f(v), (-1)^s e_i \rangle \leq -\varepsilon/\sqrt{d}$. Let $(-1)^{s'} e_{i'}$ be the unit vector along the coordinate of $N(v)$ starting at v, where $s' \in \{0, 1\}$ and $i' \in [d]$. Then we define $w = v + \gamma \cdot (-1)^{s'} e_{i'}$, where by definition $w \equiv N(v)$. From Lemma 3 w is valid and thus $P(w) = f(w)$, and from Lemma 1 we obtain that $P(N(v)) = f(x) < f(v) = P(v)$.
- If v is valid and $\|\nabla f(v)\| \leq \varepsilon$ and $\lambda_{min}(\nabla^2 f(v)) < -\sqrt{4\rho\varepsilon}$, let $w = \text{round}(v + r_0 e_1)$ be the output of $N(v)$. From Lemma 3, w is also valid and thus $P(w) = f(w)$. Using Lemma 2, we get the sufficient decrease $f(x) < f(v)$, which yields $P(N(v)) = f(x) < f(v) = P(v)$.

Therefore, we conclude that v cannot be a solution of LOCALOPT, unless v is a solution of SOSP. ■

4 PLS-hardness of Finding ε-SOSPs in Unconstrained Optimization

In this section we show that the problem SOSP is PLS-hard even when the number of dimensions is $d = 2$. We summarize the main result in the following theorem.

Theorem 2 (Hardness)**.** *SOSP is PLS-hard.*

To show our hardness result, we first define the HESSIAN-FOSP problem. HESSIAN-FOSP is similar to FOSP but differs on the input: It also requires the function f to be ρ-Hessian-Lipschitz. More formally, we define the HESSIAN-FOSP problem as follows:

Problem 5: HESSIAN-FOSP

Require: precision parameter $\varepsilon > 0$
$B, L, \rho > 0$ and $f \in \mathcal{F}(B, L, \rho)$
Turing machines representing f, ∇f and $\nabla^2 f$

Find: $x^* \in \mathbb{R}^d$: $\|\nabla f(x^*)\| \leq \varepsilon$

To prove Theorem 2, we will first show that ITER reduces to HESSIAN-FOSP (which implies that HESSIAN-FOSP is PLS-hard). Moreover, HESSIAN-FOSP trivially reduces to SOSP (because an ε-SOSP is also an ε-FOSP), which implies that SOSP is PLS-hard.

4.1. Finding ε-FOSPs of C^2 continuous functions is also hard for PLS in Unconstrained Optimization.

In this subsection, we will focus on showing the hardness of HESSIAN-FOSP for PLS. We will show that the PLS-hardness holds even for $d = 2$.

Theorem 3. *HESSIAN-FOSP is PLS-hard.*

High-level Proof Sketch of Theorem 3. We will construct the function f, such that f satisfies the input conditions of Problem 5, and we will incorporate ITER instances in the domain of $\mathbb{R}^2$ such that every ε-FOSP of f (i.e. any solution of the HESSIAN-FOSP problem) corresponds to a solution of an ITER instance. The full proof of Theorem 3 can be found in Appendix B.

Combining Theorem 3 (and more formally, Theorem 4, see Appendix E) with the PLS-completeness of finding ε-FOSPs [16], we get the following corollary.

Corollary 1. *There is an efficient polynomial-time reduction from finding approximate FOSPs of C^2 continuous functions to finding approximate FOSPs of C^1 continuous functions in unconstrained optimization. The inverse also holds.*

The above corollary is interesting because it implies that even if we add access to an extra Turing machine, i.e. $\mathcal{C}_{\nabla^2 f}$ that computes $\nabla^2 f$, the white box problem of finding ε-FOSPs in unconstrained optimization remains hard for PLS.

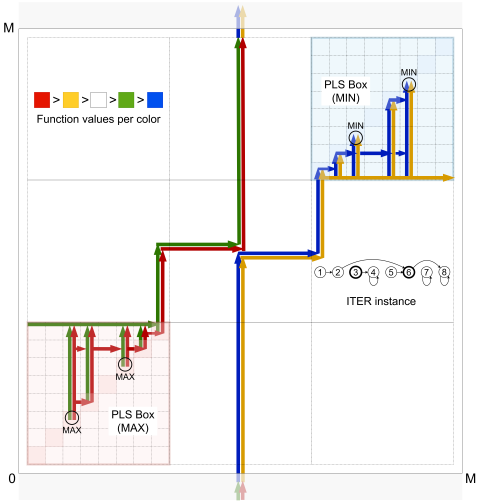


Figure 2: PLS-Hardness construction overview

One would might expect that when given access to the second-order oracles, the problem would be easier.

4.1.1 Main Components of the Proof of Theorem 2

Overview of the construction of f. We will construct f so as to be a periodic function, as follows: We define the function g over a square $[0, M]^2$, where $M = \mathcal{O}(2^n)$ and $[2^n]$ represents the domain of the values of an ITER instance. We denote this square by *tile*. Then, f is obtained by repeating copies of g in the whole $\mathbb{R}^2$ plane. If g satisfies some necessary boundary properties then repeating g this way yields a function f that is bounded, smooth and Hessian-Lipschitz. So, the main question is how to construct the function g in order to obtain f.

Tile and Small Boxes. We illustrate a tile in Fig. 2. The tile contains a 2-dimensional discrete grid, namely $[\![0, M]\!]$, which consists of *small boxes* of length 1. In order to define g, we first define the discrete functions $\Phi, \Phi_x, \Phi_y : [\![0, M]\!] \to \mathbb{R}$. The above discrete functions are defined on the four corners of the small boxes contained inside the tile (see an example of a small box in Appendix B, Fig. 5). We want the following to hold at the four corners of each small box: $g = \Phi$, $g_x = \Phi_x$, $g_y = \Phi_y$. Fig. 2 shows the values of g and the direction of its negative gradients at the corners of the small boxes of the tile. Specifically, at the corners of the small boxes, the values of g are organized in five ordered colors and the corresponding values of $\nabla g = [\Phi_x, \Phi_y]^T$ can be either $[-1/2, 0]^T$ or $[0, -1/2]^T$.

Given the fact that g is a periodic C^2 continuous function, the function must attain at least one local minimum and one local maximum in each tile. Similar to [16], we construct the corners of the small boxes such that their colors and the negative gradients will ensure that the function g attains local optima in specific regions of the tile. At the areas where the optima can be formed, we locate two arbitrary ITER instances inside PLS "gadgets" (as in [13, 16]); namely the MIN-PLS Box (for the local minima) and the MAX-PLS Box (for the local maxima). Intuitively, we want the function g to allow the following: (a) If we begin at any point in the background (i.e. regions in the tile colored in white) and we are looking for a local minimum then by following the direction of the negative gradients we will necessarily move to the MIN-PLS Box, and (b) If we begin at any point in the background and we are looking for a local maximum then by following the direction of the gradients we will necessarily move to the MAX-PLS Box.

The challenge now is how to construct the function g, so that it is bounded, C^2 continuous, smooth and Hessian-Lipschitz, while allowing the stationary points to be formed only in places where an ITER solution inside a PLS Box exists.

Biquintic Interpolation. To address the above question, we will construct one polynomial for each small box based on the specified function and gradient values of g on the four corners of the small box. To ensure C^2 over the whole tile, apart from the corner constraints, the polynomials should satisfy C^2 continuity constraints over the common edges between adjacent small boxes.

Standard approaches, such as [13, 16], leverage the *bicubic interpolation* schema [35]. However, in our case, bicubic interpolation does not allow the function g (and thus f) to satisfy the input requirements of the HESSIAN-FOSP problem, because the number of free parameters of the polynomial is less than the number of constraints on the corners. In particular, for small box $sm(a, b)$[1] there are 12 constraints for the target function and gradient values on the 4 corners and 9 constraints for the Hessian values on the three corners that are common either with $sm(a-1, b)$ or with $sm(a, b-1)$. This gives a total of 21 independent constraints, while bicubic interpolation has 16 free parameters.

To overcome this issue, we need higher degree interpolation schemes. It turns out that the minimal degree required for variable x and variable y is 5. We can see this by considering the marginalisation of the polynomial on the edges of the box, which is a one variable polynomial of x or y, depending on the edge. The marginal polynomial on a box edge should satisfy 6 constraints, corresponding to its value, first and second derivative on the two corners of the edge. Polynomials of degree less than 5 could not satisfy these constraints, thus marginals should have degree at least 5.

[1] We denote by $sm(a, b)$ the small box which has: the point (a, b) as the bottom left corner, the point $(a, b+1)$ as the top left corner, the point $(a+1, b)$ as the bottom right corner and the point $(a+1, b+1)$ as the top right corner.

In doing so, we utilize the *biquintic interpolation* schema. The general form of the biquintic interpolation on $sm(a,b)$ is the following:

$$g^{a,b}(x,y) = \sum_{i=0}^{5}\sum_{j=0}^{5} c_{i,j}^{a,b}(x-a)^i(y-b)^j$$

for every $a, b \in [\![0, M-1]\!]$, where we used $g^{a,b}$ to denote the function g defined on $sm(a,b)$. Now, to determine the unknown coefficients $c_{i,j}^{a,b}$ based on the corner values and derivatives we consider the matrix $V^{a,b}$, containing the target function and derivative values on the corners. More specifically, the box $V^{a,b}_{i:i+1,j:j+1}$, for $i = 0,2,4, j = 0,2,4$, contains the target values of $\frac{\partial^{(i+j)/2} g^{a,b}(x,y)}{\partial x^{i/2}, \partial y^{j/2}}$ on the four corners. For this construction we set all higher order derivatives equal to zero on the corners.

The corner constraints for polynomial $g^{a,b}(x,y)$ give a system of equations which can be written in matrix form as: $A \cdot C^{a,b} \cdot A^T = V^{a,b}$, where $C^{a,b}$ is the matrix of the unknown coefficients $c_{i,j}^{a,b}$ and A is a constant invertible matrix. Since A is invertible the system is solvable and the coefficients are calculated using the formula $C^{a,b} = A^{-1} \cdot V^{a,b} \cdot (A^{-1})^T$. We will now focus on the continuity and boundedness constraints.

Lemma 4. *If we apply biquintic interpolation on each small box of the tile, the function f, defined on the whole $\mathbb{R}^2$, is $2^{15}M$-bounded, $2^{22}M$-smooth and $2^{23}M$-Hessian-Lipschitz.*

The key idea of the reduction is to construct the continuous function f in such a way that it can only have ε-SOSPs in areas that correspond to solutions of the ITER. It suffices to show that small boxes that do not lie close to an ITER solution do not have any ε-FOSPs (i.e. solutions of HESSIAN-FOSP). Since the number of these boxes scales with the dimension of the ITER instance, we want to reduce the number of cases that we need to check. Thus, we consider some general conditions based on the colors and the gradients on the four corners of a small box and show that these conditions imply the inexistence of ε-FOSPs in the box. We also show that all small boxes that do not lie close to an ITER solution satisfy these conditions.

Lemma 5. *Biquintic interpolation does not introduce any 0.001-FOSP in areas where an* ITER *solution does not exist.*

The above lemma is the final component for our proof and implies that through biquintic interpolation a solution of HESSIAN-FOSP inside the tile can exist only in the areas where there exists a solution of ITER.

4.2. Query Complexity: Lower Bound for $d = 2$. Let $\mathcal{T}(\mathcal{A}; (x_0, y_0); f)$ be the query complexity of the algorithm A using (x_0, y_0) and f as inputs. From the construction of f we used to prove our hardness result, we obtain the following corollary, which indicates the lower bound query complexity for finding an ε-SOSP for $d = 2$ in the unconstrained optimization setting.

Corollary 2. *For any deterministic algorithm $\mathcal{A}$ that computes ε-SOSPs and any starting point $(x_0, y_0) \in \mathbb{R}^2$, it holds that there exists a 1-bounded, 1-Lipschitz, 1-smooth and 1-Hessian-Lipschitz function $f : \mathbb{R}^2 \to \mathbb{R}$, such that $\mathcal{T}(\mathcal{A}; (x_0, y_0); f) \geq \Omega(1/\varepsilon)$.*

It is worth noting that for ϵ-FOSPs in the unconstrained setting, Hollender & Zampetakis [16] showed that the query lower bound is $\Omega(1/\epsilon)$ which matches the query lower bound for ϵ-SOSPs.

5 Conclusion

In this paper, we showed that the problem of finding approximate second-order stationary points is PLS-complete, answering an open question asked in [16]. There are many possible open questions for future consideration: (a) What is the computational complexity of third-order stationary points in the white box and black box models? (b) Is there a relaxed notion of second-order stationarity in restricted domains that is not NP-hard (as in [31]) but belongs in PLS or CLS instead?

Impact Statement

The paper deals with the computational complexity of approximating second-order stationary points and hence it is of theoretical nature. The authors do not foresee any societal or ethical implications from this work.

Acknowledgements

This research was supported in part by project MIS 5154714 of the National Recovery and Resilience Plan Greece 2.0 funded by the European Union under the NextGenerationEU Program, and the French National Research Agency (ANR) in the framework of the PEPR IA FOUNDRY project (ANR-23-PEIA-0003), the "Investissements d'avenir" program (ANR-15-IDEX-02), the LabEx PERSYVAL (ANR-11-LABX-0025-01), MIAI@Grenoble Alpes (ANR-19-P3IA-0003). IP would like to acknowledge startup grant from UCI and UCI ICS Research Award. Part of this work was conducted while PM and IP were visiting Archimedes Research Unit.

References

[1] Agarwal, N., Allen-Zhu, Z., Bullins, B., Hazan, E., and Ma, T. Finding approximate local minima faster than gradient descent. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1195–1199, 2017.

[2] Anandkumar, A. and Ge, R. Efficient approaches for escap-

ing higher order saddle points in non-convex optimization. In *Conference on learning theory*, pp. 81–102. PMLR, 2016.

[3] Antonakopoulos, K., Mertikopoulos, P., Piliouras, G., and Wang, X. AdaGrad avoids saddle points. In *ICML '22: Proceedings of the 39th International Conference on Machine Learning*, 2022.

[4] Arjevani, Y., Shamir, O., and Shiff, R. Oracle complexity of second-order methods for smooth convex optimization. *Mathematical Programming*, 178(1):327–360, 2019.

[5] Carmon, Y., Duchi, J. C., Hinder, O., and Sidford, A. "convex until proven guilty": Dimension-free acceleration of gradient descent on non-convex functions. In *International conference on machine learning*, pp. 654–663. PMLR, 2017.

[6] Carmon, Y., Duchi, J. C., Hinder, O., and Sidford, A. Accelerated methods for nonconvex optimization. *SIAM Journal on Optimization*, 28(2):1751–1772, 2018.

[7] Cartis, C., Gould, N. I., and Toint, P. L. Second-order optimality and beyond: Characterization and evaluation complexity in convexly constrained nonlinear optimization. *Foundations of Computational Mathematics*, 18:1073–1107, 2018.

[8] Curtis, F. E. and Robinson, D. P. Exploiting negative curvature in deterministic and stochastic optimization. *Mathematical Programming*, 176:69–94, 2019.

[9] Daneshmand, H., Kohler, J., Lucchi, A., and Hofmann, T. Escaping saddles with stochastic gradients. In *ICML '18: Proceedings of the 35th International Conference on Machine Learning*, 2018.

[10] Daskalakis, C. and Papadimitriou, C. Continuous local search. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pp. 790–804. SIAM, 2011.

[11] Daskalakis, C., Skoulakis, S., and Zampetakis, M. The complexity of constrained min-max optimization. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1466–1478, 2021.

[12] Du, S. S., Jin, C., Lee, J. D., Jordan, M. I., Póczos, B., and Singh, A. Gradient descent can take exponential time to escape saddle points. In *NIPS '17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

[13] Fearnley, J., Goldberg, P., Hollender, A., and Savani, R. The complexity of gradient descent: Cls= ppad∩ pls. *Journal of the ACM*, 70(1):1–74, 2022.

[14] Fearnley, J., Goldberg, P. W., Hollender, A., and Savani, R. The complexity of computing kkt solutions of quadratic programs. *arXiv preprint arXiv:2311.13738*, 2023.

[15] Ge, R., Huang, F., Jin, C., and Yuan, Y. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on learning theory*, pp. 797–842. PMLR, 2015.

[16] Hollender, A. and Zampetakis, E. The computational complexity of finding stationary points in non-convex optimization. In *The Thirty Sixth Annual Conference on Learning Theory*, pp. 5571–5572. PMLR, 2023.

[17] Hsieh, Y.-P., Mertikopoulos, P., and Cevher, V. The limits of min-max optimization algorithms: Convergence to spurious non-critical sets. In *ICML '21: Proceedings of the 38th International Conference on Machine Learning*, 2021.

[18] Hsieh, Y.-P., Karimi, M. R., Krause, A., and Mertikopoulos, P. Riemannian stochastic optimization methods avoid strict saddle points. In *NeurIPS '23: Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2023.

[19] Jin, C., Ge, R., Netrapalli, P., Kakade, S. M., and Jordan, M. I. How to escape saddle points efficiently. In *International conference on machine learning*, pp. 1724–1732. PMLR, 2017.

[20] Johnson, D. S., Papadimitriou, C. H., and Yannakakis, M. How easy is local search? *Journal of computer and system sciences*, 37(1):79–100, 1988.

[21] Lan, G. *First-order and Stochastic Optimization Methods for Machine Learning*. Springer, 2020.

[22] Lee, J. D., Simchowitz, M., Jordan, M. I., and Recht, B. Gradient descent only converges to minimizers. In *COLT '16: Proceedings of the 29th Annual Conference on Learning Theory*, 2016.

[23] Lee, J. D., Panageas, I., Piliouras, G., Simchowitz, M., Jordan, M. I., and Recht, B. First-order methods almost always avoid strict saddle points. *Mathematical programming*, 176: 311–337, 2019.

[24] Mertikopoulos, P., Hallak, N., Kavis, A., and Cevher, V. On the almost sure convergence of stochastic gradient descent in non-convex problems. In *NeurIPS '20: Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

[25] Monteiro, R. D. and Svaiter, B. F. An accelerated hybrid proximal extragradient method for convex optimization and its implications to second-order methods. *SIAM Journal on Optimization*, 23(2):1092–1125, 2013.

[26] Morioka, T. *Classification of search problems and their definability in bounded arithmetic*. PhD thesis, 2001.

[27] Murty, K. G. and Kabadi, S. N. Some np-complete problems in quadratic and nonlinear programming. *Mathematical Programming: Series A and B*, 39(2):117–129, 1987.

[28] Nesterov, Y. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[29] Nesterov, Y. and Polyak, B. T. Cubic regularization of newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

[30] Nesterov, Y. et al. *Lectures on convex optimization*, volume 137. Springer, 2018.

[31] Nouiehed, M., Lee, J. D., and Razaviyayn, M. Convergence to second-order stationarity for constrained non-convex optimization. *arXiv preprint arXiv:1810.02024*, 2018.

[32] Panageas, I. and Piliouras, G. Gradient descent only converges to minimizers: Non-isolated critical points and invariant regions. In *ITCS '17: Proceedings of the 8th Conference on Innovations in Theoretical Computer Science*, 2017.

[33] Pascanu, R., Dauphin, Y. N., Ganguli, S., and Bengio, Y. On the saddle point problem for non-convex optimization. *arXiv preprint arXiv:1405.4604*, 2014.

[34] Pemantle, R. Nonconvergence to unstable points in urn models and stochastic aproximations. *Annals of Probability*, 18(2):698–712, April 1990.

[35] Russell, W. S. Polynomial interpolation schemes for internal derivative distributions on structured grids. *Applied Numerical Mathematics*, 17(2):129–171, 1995.

[36] Staib, M., Reddi, S., Kale, S., Kumar, S., and Sra, S. Escaping saddle points with adaptive gradient methods. In *ICML*

'19: Proceedings of the 36th International Conference on Machine Learning, 2019.

[37] Tripuraneni, N., Stern, M., Jin, C., Regier, J., and Jordan, M. I. Stochastic cubic regularization for fast nonconvex optimization. *Advances in neural information processing systems*, 31, 2018.

A Missing Proofs of Section 3

A.1. Proof of Lemma 1.

Proof. Using the Taylor's theorem on the smoothness of f we get:

$$\begin{aligned} f(x_{t+1}) &\leq f(x_t) + \langle \nabla f(x_t), x_{t+1} - x_t \rangle + \frac{L}{2}\|x_{t+1} - x_t\|^2 \\ &= f(x_t) + \gamma \langle \nabla f(x_t), (-1)^s e_i \rangle + \frac{L\gamma^2}{2}\|(-1)^s e_i\|^2 \\ &\leq f(x_t) - \frac{\varepsilon\gamma}{\sqrt{d}} + \frac{L\gamma^2}{2} \\ &\leq f(x_t) - \frac{\varepsilon\gamma}{2\sqrt{d}} \end{aligned}$$

where we used the fact that $\gamma \leq \frac{\varepsilon}{\sqrt{d}L}$.

■

A.2. Proof of Lemma 2.

Proof. Using the Taylor's theorem on the Lipschitzness of $\nabla^2 f$ [29] we get:

$$\begin{aligned} f(\tilde{x}_t) &\leq f(x_t) + \langle \nabla f(x_t), \tilde{x}_t - x_t \rangle + \frac{1}{2}\langle \nabla^2 f(x_t)(\tilde{x}_t - x_t), \tilde{x}_t - x_t \rangle + \frac{\rho}{6}\|\tilde{x}_t - x_t\|^3 \\ &\leq f(x_t) + r\langle \nabla f(x_t), u_1 \rangle + \frac{r^2\lambda_1}{2} + \frac{r^3\rho}{6} \\ &< f(x_t) + r\varepsilon - r^2\sqrt{\rho\varepsilon} + \frac{r^3\rho}{6} \end{aligned}$$

where we used the fact that $\tilde{x}_t = x_t + ru_1$, with u_1 being the unit eigenvector of $\nabla^2 f(x_t)$ corresponding to the smallest eigenvalue (denoted by λ_1), and $\|\nabla f(x_t)\| \leq \varepsilon$. Next by letting $\xi(r) = r\varepsilon - r^2\sqrt{\rho\varepsilon} + \frac{r^3\rho}{6}$, we get $f(\tilde{x}_t) \leq f(x_t) + \xi(r)$. By assigning $r = r_0 = \frac{3\sqrt{\varepsilon}}{\sqrt{\rho}}$, we get the sufficient decrease $\xi(r_0) = -\frac{3\varepsilon\sqrt{\varepsilon}}{2\sqrt{\rho}}$.

Using the Taylor's theorem now on the smoothness of f we get:

$$\begin{aligned} f(x_{t+1}) &\leq f(\tilde{x}_t) + \langle \nabla f(\tilde{x}_t), x_{t+1} - \tilde{x}_t \rangle + \frac{L}{2}\|x_{t+1} - \tilde{x}_t\|^2 \\ &= f(\tilde{x}_t) + \|\nabla f(\tilde{x}_t)\| \cdot \sqrt{d} \cdot \gamma + \frac{Ld\gamma^2}{2} \\ &\leq f(\tilde{x}_t) + (\varepsilon + Lr)\sqrt{d}\gamma + \frac{\varepsilon\sqrt{d}\gamma}{2} \\ &= f(\tilde{x}_t) + \left(\frac{3}{2}\varepsilon\sqrt{d} + \frac{3L\sqrt{\varepsilon}\sqrt{d}}{\sqrt{\rho}}\right)\gamma \end{aligned}$$

where we used the fact that $\gamma \leq \frac{\varepsilon}{\sqrt{d}L}$ and $||\nabla f(x_t)|| \leq \varepsilon + L\rho$.

Let $\xi' = \left(\frac{3}{2}\varepsilon\sqrt{d} + \frac{3L\sqrt{\varepsilon}\sqrt{d}}{\sqrt{\rho}}\right)\gamma$. Now we can guarantee sufficient decrease in $f(x_{t+1})$, since $\xi' - |\xi(r_0)| \leq -\frac{3}{4}\frac{\varepsilon\sqrt{\varepsilon}}{\sqrt{\rho}}$, which implies that $f(\tilde{x}_t) \leq f(x_t) - \frac{3}{4}\frac{\varepsilon\sqrt{\varepsilon}}{\sqrt{\rho}}$.

■

A.3. Proof of Lemma 3.

Proof. Let v be a valid of G_γ. We examine the two following cases:

Let v be a valid of G_γ. Let $w = N(v)$. We examine the two following cases:

- If $\|\nabla f(v)\| > \varepsilon$, from Lemma 1 we get:

$$\begin{aligned} f(w) &\leq f(v) - \frac{\varepsilon\gamma}{2\sqrt{d}} \\ &= f(v) - \frac{\varepsilon}{2\sqrt{d}}\|w - v\| \\ &\leq f(0) - \min\left\{\frac{\varepsilon}{2\sqrt{d}}, \frac{\varepsilon r}{4(r+\gamma\sqrt{d})}\right\}\|v\| - \frac{\varepsilon}{2\sqrt{d}}\|w - v\| \\ &\leq f(0) - \min\left\{\frac{\varepsilon}{2\sqrt{d}}, \frac{\varepsilon r}{4(r+\gamma\sqrt{d})}\right\}(\|v\| - \|w - v\|) \\ &\leq f(0) - \min\left\{\frac{\varepsilon}{2\sqrt{d}}, \frac{\varepsilon r}{4(r+\gamma\sqrt{d})}\right\}\|w\| \end{aligned}$$

  where we have used the fact that $\gamma = \|w - v\|$, $\frac{\varepsilon}{2\sqrt{d}} \geq \min\left\{\frac{\varepsilon}{2\sqrt{d}}, \frac{\varepsilon r}{4(r+\gamma\sqrt{d})}\right\}$ and $\|w - v\| \geq \|w\| - \|v\|$.

- Following similar steps as above, if $\|\nabla f(v)\| \leq \varepsilon$ and $\lambda_{min}(\nabla^2 f(v)) < -\sqrt{4\rho\varepsilon}$, using Lemma 2 we get:

$$\begin{aligned} f(w) &\leq f(v) - \frac{3\varepsilon\sqrt{\varepsilon}}{4\sqrt{\rho}} \\ &= f(v) - \frac{\varepsilon r}{4} \\ &\leq f(0) - \min\left\{\frac{\varepsilon}{2\sqrt{d}}, \frac{\varepsilon r}{4(r+\gamma\sqrt{d})}\right\}\|v\| - \frac{\varepsilon r}{4} \\ &\leq f(0) - \min\left\{\frac{\varepsilon}{2\sqrt{d}}, \frac{\varepsilon r}{4(r+\gamma\sqrt{d})}\right\}\|v\| - \frac{\varepsilon r\|v - w\|}{4(r+\gamma\sqrt{d})} \\ &\leq f(0) - \min\left\{\frac{\varepsilon}{2\sqrt{d}}, \frac{\varepsilon r}{4(r+\gamma\sqrt{d})}\right\}\|w\| \end{aligned}$$

Therefore, from the above we conclude that w is always valid in either case.

■

A.4. Proof of Proposition 1.

Proof. It suffices to show that the maximum distance from starting point $x_0 = 0$ that Algorithm 1 can reach is bounded by $\tilde{R}$; i.e $\|x_0 - x^*\| \leq \tilde{R}$. It is easy to see that Algorithm 1 terminates at a point x^* which is an ε-SOSP.

We define Steps 4-6 as the FIND-FOSP routine and Steps 7-10 as the ESCAPE routine. From Lemma 1, we get that the algorithm needs at most $\mathcal{T}_{GD}$ steps to find an ε-FOSP starting from an arbitrary non ε-FOSP (i.e. it needs at most $\mathcal{T}_{GD}$ steps to complete a FIND-FOSP routine). From Lemma 2, we get that the algorithm needs at most $\mathcal{T}_{NC}$ steps to escape an ε-FOSP but not ε-SOSP and find an ε-SOSP. Since the ESCAPE routine may only happen after the termination of a FIND-FOSP routine, the algorithm may run the ESCAPE routine at most $\mathcal{T}_{NC}$ times and the FIND-FOSP routine at most

$\mathcal{T}_{NC}+1$ times. Moreover, a FOSP routine can be completed within a radius of at most $\gamma\mathcal{T}_{CD}$, while a single update of the ESCAPE routine uses a constant radius of r. Therefore, the algorithm terminates at a point which is an ε-SOSP and the maximum distance from starting point x_0 that the algorithm can reach is at most $\tilde{R}=\gamma\mathcal{T}_{CD}(\mathcal{T}_{NC}+1)+r\mathcal{T}_{NC}$.

■

A.5. Missing Proof of Proposition 2.

Proof. To show this claim we use arguments similar to [16]. More specifically, any point w on the boundary of $[-R,R]^d$ satisfies that $\|w\|\geq R$. For the sake of contradiction we assume that a point w on the boundary is valid. Then it should hold that $f(w)\leq f(0)-\min\left\{\frac{\varepsilon}{2\sqrt{d}},\frac{\varepsilon r}{4(r+\gamma\sqrt{d})}\right\}\|w\|$. Since $|f(0)|\leq B$, we get that $f(w)\leq B-\min\left\{\frac{\varepsilon}{2\sqrt{d}},\frac{\varepsilon r}{4(r+\gamma\sqrt{d})}\right\}R$. But also we have that $R=\max\left\{\frac{10B\sqrt{d}}{\varepsilon},\frac{10B(r+\gamma\sqrt{d})}{\varepsilon r},\tilde{R}\right\}$, which implies that $f(w)<-B$. This is a contradiction, due to the boundedness assumption $|f(w)|\leq -B$. Thus, we conclude that each point w on the boundary is invalid.

■

A.6. The reduction is polynomial-time in the input.

Lemma A.1 (Transformation to boolean circuits)**.** *The defined neighbor function N and potential function P can be transformed from Turing machines to boolean circuits and their domain and range to $[2^n]$ for some $n\in\mathbb{N}$.*

Proof. We use arguments similar to [16]. Specifically, first, observe that the binary representation of the points in $v\in G_\gamma$ is $\text{len}(v)\leq d\cdot\log(R/\gamma)$. Hence, the output $f(v)$ for any $v\in G_\gamma$ has $\text{len}(f(v))=q(\text{len}(v))$ where q is some polynomial that is specified together with the description of $\mathcal{C}_f$. We can then pick $n=q(\text{len}(v))$ and we can map the set $[2^n]$ to describe both G_γ and the possible outputs of the Turing machine P that we described above. This way we can make N,P to be mappings from $[2^n]$ to itself and have running time $r(n)$ for some polynomial r. We note that this is feasible because finding the eigenvalues and eigenvectors of Hessian requires polynomial time in d (e.g. via spectral decomposition). Finally, we utilize classical transformations of Turing machines with running time $r(n)$ to boolean circuits with size $r(n)\log(r(n))$. ■

To finish the proof of Theorem 1, we use Lemma A.1 to transform the Turing machines of N and P into boolean circuits, as required in Problem 1. This completes the reduction of SOSP to LocalOpt. Based on the above, it is easy to see that the reduction is polynomial-time in the input, i.e., the size of the Turing machines $\mathcal{C}_f,\mathcal{C}_{\nabla f},\mathcal{C}_{\nabla^2 f}$, and polylogarithmic in the parameters B,L,ρ, and $1/\varepsilon$.

Remark A.1. Observe that to decide whether a point $v\in G_\gamma$ is valid, we need time $\text{poly}(\text{len}(x))$. If it were not for that requirement, we could have simply defined as valid only the points that Algorithm 1 traverses starting at 0, all the remaining points of the grid as invalid and grid's radius $R=\tilde{R}$ (i.e. equal to the maximum distance that Algorithm 1 may cover to find an ε-SOSP). This would guarantee that the LocalOpt instance has exactly one solution (that is the the point that Algorithm 1 reached), but it would require to decide whether a point is valid in time $\text{poly}(1/\varepsilon)$ which is prohibitive for polynomial-time Turing machines.

B Hardness: Proof of Theorem 3

Our goal is to construct a function f that satisfies the input conditions of HESSIAN-FOSP (Problem 5), while incorporating Iter instances in the domain of $\mathbb{R}^2$ in a way that every ε-FOSP of f corresponds to a solution of an Iter instance. This reduces Iter to HESSIAN-FOSP, thus proving the PLS-hardness.

Our construction in Subsections B.1 and B.2 follows the one in [16]. The main technical challenge we address here, which is not covered by the techniques in [16], is the interpolation step which guarantees the C^2-continuity of f, along with boundedness, smoothness and Hessian-Lipschitzness.

B.1. Defining the function on the grid. In this subsection we will define a function $g:[0,M]^2\to\mathbb{R}$, which is later used as a repeated "tile" for the construction of the function f over $\mathbb{R}^2$ (see Appendix B.5). Since Iter is a combinatorial problem, it is easier to define g on a grid $[[0,M]]^2$ and then extend it to a continuous tile via interpolation (see Appendix B.3).

Let $N = 2^{n+3}$, $M = 3N + 4$ and $K = M/2$, where n is given by the ITER instance, i.e. the ITER instance contains 2^n nodes. In the next subsection it will become clear why we chose this value for N.

The definition of g on $[[0, M]]^2$ contains colored lines (whose color corresponds to the function's values), two PLS Boxes (in which the ITER instance is encoded) and a white background. See Fig. 2 for the complete definition of g or Fig. 4 for the definition of g outside of the PLS Boxes. Note that the PLS Boxes have width N and the thin white strips between the border and the PLS Boxes have width 2.

At this point we will describe the position, the function values $\Phi(a, b)$ and the gradients $(\Phi_x(a, b), \Phi_y(a, b))$ of each region. The two piecewise functions below summarize all of this information. See Fig. 2 for the relationship between function values and colors, as well as Fig. 4 for the direction of the negative gradient at each grid point.

1. **PLS Boxes:** These are squares of width N, where the ITER instance is encoded. MIN-PLS Box contains g's minima and MAX-PLS Box contains g's maxima. Their exact description will be given in Appendix B.2. For now, we consider them as black boxes with function values $\Phi_{MIN-PLS\ BOX}(a, b)$ and $\Phi_{MAX-PLS\ BOX}(a, b)$ respectively and gradients $\Phi_{xy,MIN-PLS\ BOX}(a, b)$ and $\Phi_{xy,MAX-PLS\ BOX}(a, b)$ respectively. Note that MIN-PLS Box includes all points in $[[2N + 2, 3N + 2]]^2$ and MAX-PLS Box includes all points in $[[2, N + 2]]^2$.

2. **Colored Lines:** There are lines of four different colors: blue, green, orange and red. Note that function values increase per color, in the aforementioned order. We denote the colored lines by $\mathcal{B}, \mathcal{G}, \mathcal{O}, \mathcal{R}$ respectively. We also use $\mathcal{TB}, \mathcal{TO}$ for the short blue and orange lines at the top of the grid. All colored lines have width 2. We now define the following:

   - **Line Positions:** As can be seen in Fig. 4, a red and a green line exit the MAX-PLS Box and, after curving first by $-90°$ and then by $90°$, they reach the middle of the top of the grid. Note that at the top of the grid there are two short lines (orange and blue), starting at the end of the red and green lines respectively. These short lines will "connect" with the orange and blue lines of the next repetition of g above. The orange and blue lines begin at the middle of the bottom of the grid and, after curving first by $-90°$ and then by $90°$, they enter the MIN-PLS Box. Note that the blue and red lines touch at the middle of the grid. All of the above information is summarized in the first piecewise function below, where the exact locations of the colored lines are also shown.

   - **Function Values:** We define the function values for each of the six lines as shown below. Note that we treat the top blue and orange lines as regular blue and orange lines because their values are very similar and therefore they function the same for the purpose of our proofs.

     - $\Phi_{\mathcal{B}}(a, b) = -x - y - 6M$
     - $\Phi_{\mathcal{G}}(a, b) = -x - y - 3M$
     - $\Phi_{\mathcal{O}}(a, b) = -x - y + 3M$
     - $\Phi_{\mathcal{R}}(a, b) = -x - y + 6M$
     - $\Phi_{\mathcal{TB}}(a, b) = -x - (y - M - 1) - 6M$
     - $\Phi_{\mathcal{TO}}(a, b) = -x - (y - M - 1) + 3M$

   - **Gradients:** For every colored point, its gradient is defined as either $(-1/2, 0)$ or $(0, -1/2)$. Generally, a colored point has gradient $(-1/2, 0)$ if it is adjacent to a white point, otherwise it has gradient $(0, -1/2)$. This rule has a few exceptions (see Fig. 4), therefore we define the gradient values more strictly in the second piecewise function below, $(\Phi_x(a, b), \Phi_y(a, b))$.

3. **White Background:** This region includes every point not mentioned in the above, i.e. the white points. Note that white points have larger values than blue and green points, but smaller than orange and red points. We denote white points by $\mathcal{W}$. For the function values of the background we use $\Phi_W(x, y) = -x + M \cdot \mathbb{1}\{x \geq K\} + M$. The gradients of the background are always $(-1/2, 0)$.

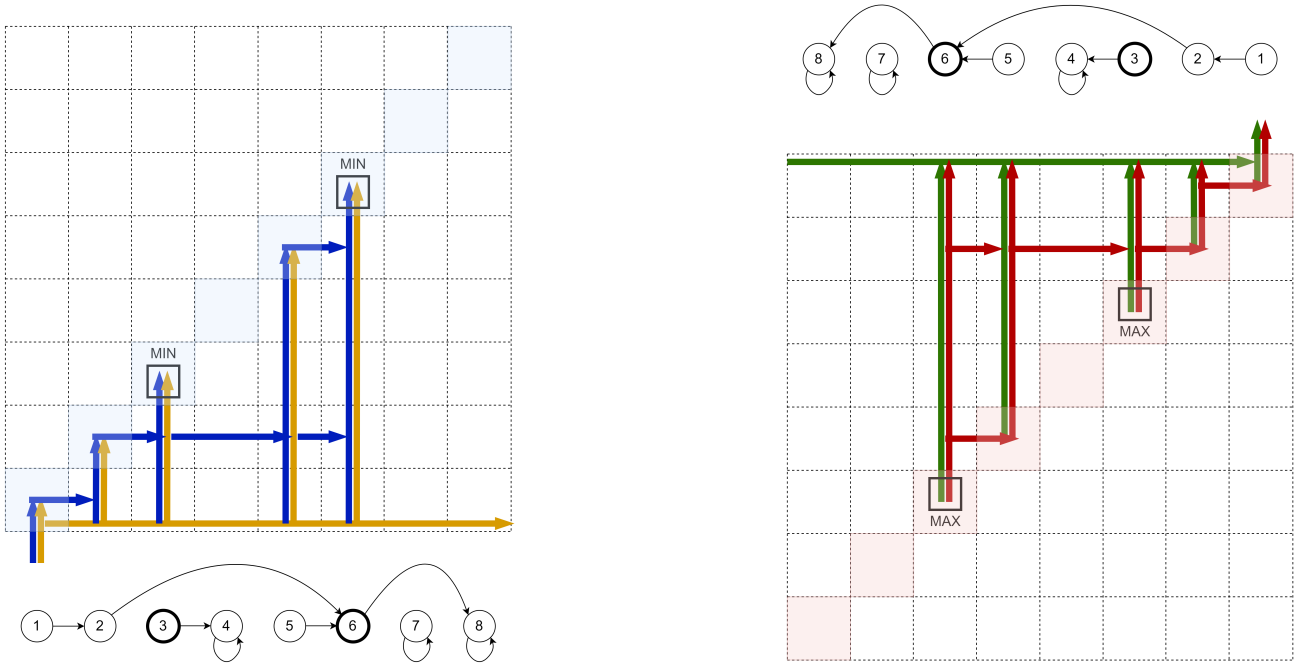
$$\Phi(a,b) = \begin{cases} \Phi_{\mathcal{B}}(a,b), & \text{for } K-2 \le a \le K \text{ and } 0 \le b \le K, \\ & \quad K-2 \le a \le 2N+4 \text{ and } K-2 \le b \le K, \\ & \quad 2N+2 \le a \le 2N+4 \text{ and } K \le b \le 2N+2 \\ \Phi_{\mathcal{O}}(a,b), & \text{for } K \le a \le K+2 \text{ and } 0 \le b \le K-2, \\ & \quad K \le a \le 2N+4 \text{ and } K \le b \le K+2, \\ & \quad 2N+4 \le a \le 2N+6 \text{ and } K-4 \le b \le 2N+2 \\ \Phi_{\mathcal{G}}(a,b), & \text{for } K-2 \le a \le K \text{ and } K+2 \le b \le M-2, \\ & \quad N-2 \le a \le K \text{ and } K+2 \le b \le K+4, \\ & \quad N-2 \le a \le N \text{ and } N+2 \le b \le K+4 \\ \Phi_{\mathcal{R}}(a,b), & \text{for } K \le a \le K+2 \text{ and } K \le b \le M-2, \\ & \quad N \le a \le K+2 \text{ and } K \le b \le K+2, \\ & \quad N \le a \le N+2 \text{ and } N+2 \le b \le K+2 \\ \Phi_{\mathcal{TB}}(a,b), & \text{for } K-2 \le a \le K \text{ and } M-1 \le b \le M \\ \Phi_{\mathcal{TO}}(a,b), & \text{for } K \le a \le K+2 \text{ and } M-1 \le b \le M \\ \Phi_{MAX\text{-}PLS\ BOX}(a,b), & \text{for } 2 \le a \le N+2 \text{ and } 2 \le b \le N+2 \\ \Phi_{MIN\text{-}PLS\ BOX}(a,b), & \text{for } 2N+2 \le a \le 3N+2 \text{ and } 2N+2 \le b \le 3N+2 \\ \Phi_{\mathcal{W}}(a,b), & \text{elsewhere} \end{cases}$$

$$(\Phi_x(a,b),\Phi_y(a,b)) = \begin{cases} \left(-\frac{1}{2},0\right), & \text{for } a = K-3/2 \text{ and } 0 \le b \le K \quad (\mathcal{B}),\\ & K-2 \le a \le 2N+3 \text{ and } b = K-1/2 \quad (\mathcal{B}),\\ & a = 2N+5/2 \text{ and } K \le b \le 2N+2 \quad (\mathcal{B}),\\ & a = K+3/2 \text{ and } 0 \le b \le K-3 \quad (\mathcal{O}),\\ & K+1 \le a \le 2N+5 \text{ and } b = K-7/2 \quad (\mathcal{O}),\\ & a = 2N+11/2 \text{ and } K-4 \le b \le 2N+2 \quad (\mathcal{O}),\\ & a = K-3/2 \text{ and } K+3 \le b \le M \quad (\mathcal{G},\mathcal{TB}),\\ & N-2 \le a \le K-1 \text{ and } b = K+7/2 \quad (\mathcal{G}),\\ & a = N-3/2 \text{ and } N+2 \le b \le K+4 \quad (\mathcal{G}),\\ & a = K+3/2 \text{ and } K \le b \le M \quad (\mathcal{R},\mathcal{TO}),\\ & N+1 \le a \le K+2 \text{ and } b = K+1/2 \quad (\mathcal{R}),\\ & a = N+3/2 \text{ and } N+2 \le b \le K+1 \quad (\mathcal{R})\\ \left(0,-\frac{1}{2}\right), & \text{for } a = K-1/2 \text{ and } 0 \le b \le K-1 \quad (\mathcal{B}),\\ & K-1 \le a \le 2N+4 \text{ and } b = K-3/2 \quad (\mathcal{B}),\\ & a = 2N+7/2 \text{ and } K-1 \le b \le 2N+2 \quad (\mathcal{B}),\\ & a = K+1/2 \text{ and } 0 \le b \le K-2 \quad (\mathcal{O}),\\ & K \le a \le 2N+5 \text{ and } b = K-5/2 \quad (\mathcal{O}),\\ & a = 2N+9/2 \text{ and } K-3 \le b \le 2N+2 \quad (\mathcal{O}),\\ & a = K-1/2 \text{ and } K+2 \le b \le M \quad (\mathcal{G},\mathcal{TB}),\\ & N-1 \le a \le K \text{ and } b = K+5/2 \quad (\mathcal{G}),\\ & a = N-1/2 \text{ and } N+2 \le b \le K+1 \quad (\mathcal{G}),\\ & a = K+1/2 \text{ and } K+1 \le b \le M \quad (\mathcal{R},\mathcal{TO}),\\ & N \le a \le K+1 \text{ and } b = K+3/2 \quad (\mathcal{R}),\\ & a = N+1/2 \text{ and } N+2 \le b \le K+2 \quad (\mathcal{R})\\ \Phi_{xy,MAX-PLS\ BOX}\ (a,b), & \text{for } 2 \le a \le N+2 \text{ and } 2 \le b \le N+2\\ \Phi_{xy,MIN-PLS\ BOX}\ (a,b), & \text{for } 2N+2 \le a \le 3N+2 \text{ and } 2N+2 \le b \le 3N+2\\ \left(-\frac{1}{2},0\right), & \text{elsewhere} \quad (\mathcal{W}) \end{cases}$$

B.2. PLS Boxes. In order to complete the description of g, we will explain how the PLS Boxes are constructed. We will show how to map an ITER instance to the two PLS Boxes included in each repetition of g. As an example, we will use the ITER instance shown in Fig. 3a and Fig. 3b. Note that in these figures each column corresponds to the node that is aligned with it in the accompanying ITER instance.

The purpose of this construction is to allow the optima of g to appear only in the regions corresponding to ITER solutions. This is essential for proving Theorem 2. Note that these regions are marked with a black rectangle labeled "MIN" or "MAX" in Fig. 3a and Fig. 3b.

We will first describe the MIN-PLS Box. Recall that the input for ITER is a boolean circuit $C : [2^n] \to [2^n]$ with $C(1) > 1$, for some natural number n. We want to partition the PLS Box into regions corresponding to each of the ITER nodes. We partition the grid into 2^n horizontal and 2^n vertical strips of width 8, each corresponding to a node of the ITER input (recall that each PLS Box has width $N = 2^{n+3}$). Hence, the grid is split into $2^n \times 2^n$ *medium boxes* of size 8×8. We refer to



(a) MIN-PLS BOX **(b)** MAX-PLS BOX

Figure 3: PLS Boxes

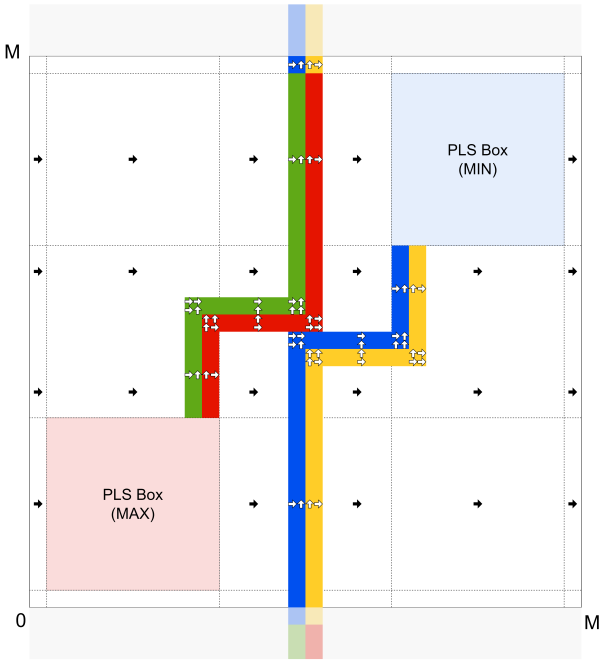


Figure 4: PLS-Hardness construction overview

medium boxes by using coordinates (i, j), indicating that the box is the intersection of horizontal strip i and vertical strip j (with $(1, 1)$ being the lower left box).

Note that the $i = j$ diagonal contains 2^n medium boxes, each of which will contain a minimum if and only if the corresponding ITER node is a solution. Our construction must guarantee that minima can only appear in this diagonal, which is colored light blue in Fig. 3a.

For simplicity, we will partition the PLS box into the regions described below:

1. **Base:** This region is independent of the ITER instance. It consists of a long horizontal orange line.

   As can be seen in Fig. 4, a blue and an orange line enter the MIN-PLS Box from the middle of the medium box $(1, 1)$. From that point, we draw a horizontal orange line, all the way to the medium box $(1, 2^n)$. Like all other lines, it has width 2, lying on the bottom two rows of the first horizontal strip. This line ends at the 7th column of the medium box $(1, 2^n)$, leaving the last column empty (white).
2. **Nodes:** Each node will have a corresponding region, which will contain a minimum if and only if the node is a solution (nodes $3, 6$ in our example).
   - For each node i we draw two vertical lines (a blue and an orange one) starting from the base, lying on the 4 middle columns of the i-th vertical strip. The lines go up to the middle of the medium box (i, i), i.e. to the 4th row of the medium box. Note that for the first node this means extending the two lines that enter the box, up to the middle of the box $(1, 1)$.
   - The above construction is only used for nodes that do not have a self-loop. For nodes with a self-loop, we leave the respective vertical strip empty. In our example, these are the nodes $4, 7, 8$.
3. **Node Connections:** These regions correspond to the edges of the ITER graph.
   - For each node i with $C(i) = j > i$ and $C(j) > j$, we draw a horizontal blue line of width 2, starting right above the blue line corresponding to the node i (thus lying the 5th and 6th rows of the i-th horizontal strip). This line is terminated when it reaches the vertical blue line in the medium box (i, j), as can be seen, e.g., with the line connecting the regions of nodes 5 and 6 of our example in Fig. 3a.
   - If this horizontal blue line encounters some pair of vertical blue-orange lines belonging to a node other than j, then it is overwritten by the vertical line in the intersection. This happens with the line corresponding to the edge $(2, 6)$ in our example, where it is overwritten by the vertical lines belonging to nodes 3 and 5.
4. **Background:** All other grid points are defined in the same way as the "white" points outside of the PLS Boxes (see Appendix B.1).

The function values of the grid points are defined exactly as in Appendix B.1 for each color. It remains to describe the gradient values for each region.

1. **Base orange line:** For each node i with $C(i) > i$, the gradient of the two middle points (4th and 5th) of the the 2nd row of the medium box $(1, i)$ (i.e. the upper half of the orange line) is $(0, -1/2)$. Everywhere else, it is $(-1/2, 0)$.
2. **Vertical lines:** The gradients in the "outer" half of each colored line (i.e. the half that touches the white background) are $(-1/2, 0)$. The gradients in the inner half are $(0, -1/2)$.
3. **Blue horizontal lines:** For each node i with $C(i) = j > i$ and $C(j) > j$, the gradient of the two middle points (4th and 5th) of the 5th row of the medium box (i, i) (i.e. the lower half of the blue line) is $(0, -1/2)$. Everywhere else, it is $(-1/2, 0)$.
4. **Background:** The gradient is $(-1/2, 0)$ for every grid point in the background.

The MAX-PLS Box is the same as the MIN-PLS Box, rotated by 180°. Its construction is almost identical, with the following minor differences:

- Orange lines are now green and blue lines are red.
- The green/red lines now enter the box from the upper right, therefore the numbering of the medium boxes begins from the upper right corner and increases while going down or left. This is why the ITER instance is written from right to left in Fig. 3a.

- Gradients are unaffected by the 180° rotation, as shown in Fig. 3a, where the red and green arrows still point up or to the right, despite the shape's rotation. The purpose of this is to make the box contain a maximum instead of a minimum.

Non-grid points inside the PLS Boxes are defined by the biquintic interpolation shown in Appendix B.3. This completes the construction of the PLS Boxes.

B.3. Biquintic Interpolation. Given the function and gradient values on the points of the $M \times M$ grid, we want to construct a C^2 continuous function on the whole tile that takes the specified values on the grid points. For this purpose, similar to [13, 16], we will use polynomial interpolation. In particular, we will construct one polynomial for each small box based on the specified function and gradient values on the 4 corners of the box. Apart from the corner constraints, the polynomials should satisfy C^2 continuity constraints over the common edges between adjacent small boxes. Let $g^{a,b}(x,y)$ be the polynomial function inside the box with corners $(a,b),(a+1,b),(a+1,b+1),(a,b+1)$, which we denote as $sm(a,b)$. The continuity equations for a box in the interior of the grid are the following:

- Zero order continuity:
  - $g^{a,b}(a,y) = g^{a-1,b}(a,y)$
  - $g^{a,b}(a+1,y) = g^{a+1,b}(a+1,y)$
  - $g^{a,b}(x,b) = g^{a,b-1}(x,b)$
  - $g^{a,b}(x,b+1) = g^{a,b+1}(x,b+1)$
- First order continuity:
  - $g_x^{a,b}(a,y) = g_x^{a-1,b}(a,y), \quad g_y^{a,b}(a,y) = g_y^{a-1,b}(a,y)$
  - $g_x^{a,b}(a+1,y) = g_x^{a+1,b}(a+1,y), \quad g_y^{a,b}(a+1,y) = g_y^{a+1,b}(a+1,y)$
  - $g_x^{a,b}(x,b) = g_x^{a,b-1}(x,b), \quad g_y^{a,b}(x,b) = g_y^{a,b-1}(x,b)$
  - $g_x^{a,b}(x,b+1) = g_x^{a,b+1}(x,b+1), \quad g_y^{a,b}(x,b+1) = g_y^{a,b+1}(x,b+1)$
- Second order continuity:
  - $g_{xx}^{a,b}(a,y) = g_{xx}^{a-1,b}(a,y),\ g_{xy}^{a,b}(a,y) = g_{xy}^{a-1,b}(a,y),\ g_{yy}^{a,b}(a,y) = g_{yy}^{a-1,b}(a,y)$
  - $g_{xx}^{a,b}(a+1,y) = g_{xx}^{a+1,b}(a+1,y),\ g_{xy}^{a,b}(a+1,y) = g_{xy}^{a+1,b}(a+1,y),\ g_{yy}^{a,b}(a+1,y) = g_{yy}^{a+1,b}(a+1,y)$
  - $g_{xx}^{a,b}(x,b) = g_{xx}^{a,b-1}(x,b),\ g_{xy}^{a,b}(x,b) = g_{xy}^{a,b-1}(x,b),\ g_{yy}^{a,b}(x,b) = g_{yy}^{a,b-1}(x,b)$
  - $g_{xx}^{a,b}(x,b+1) = g_{xx}^{a,b+1}(x,b+1),\ g_{xy}^{a,b}(x,b+1) = g_{xy}^{a,b+1}(x,b+1),\ g_{yy}^{a,b}(x,b+1) = g_{yy}^{a,b+1}(x,b+1)$

We observe that it suffices to guarantee the continuity constraints on the edge between boxes $sm(a,b)$ and $sm(a-1,b)$ and the edge between boxes $sm(a,b)$ and $sm(a,b-1)$ for all $(a,b) \in [\![0, M-1]\!]^2$ (see Fig. 5). This way, inductively, all constraints will be satisfied. The resulting set of constraints is minimal, as all constraints are independent.

The standard *bicubic interpolation* scheme, which has been used in other similar constructions [13, 16] fails to

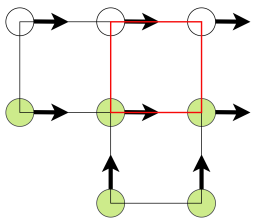


Figure 5: A small box and its neighbours on the left and at the bottom

preserve Hessian-Lipschitzness, because the number of free parameters is less than the number of constraints on the corners. In particular, for $sm(a,b)$ there are 12 constraints for the target function and gradient values on the 4 corners and 9 constraints for the second order derivatives on the three corners that are common either with box $sm(a-1,b)$ or $sm(a,b-1)$. This gives a total of 21 independent constraints, while bicubic interpolation has 16 free parameters. Thus, we will need higher degree interpolation schemes. The minimal degree required for variable x and variable y is 5. We can

see this by considering the marginalisation of the polynomial on the edges of the box. Such a marginalisation is a one variable polynomial of x or y, depending on the edge. The marginal polynomial on a box edge should satisfy 6 constraints, corresponding to its value, first and second derivative on the two corners of the edge. Polynomials of degree less than 5 could not satisfy any arbitrary value of such constraints, thus marginals wrt x and y should have degree at least 5. It turns out that degree 5 for x and y suffices to satisfy all corner and edge constraints. This scheme is called *biquintic interpolation*. Next, we will show the exact formulae that we used. The general form of the polynomial is the following:

$$g^{a,b}(x,y) = \sum_{i=0}^{5}\sum_{j=0}^{5} c_{i,j}^{a,b}(x-a)^i(y-b)^j \quad \forall a \in [\![0, M-1]\!], b \in [\![0, M-1]\!]$$

To determine the unknown coefficients $c_{i,j}^{a,b}$ based on the corner values and derivatives, we apply the following:
We consider matrix A as follows:

$$A := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6 & 12 & 20 \end{bmatrix}$$

Also, we consider matrix $V^{a,b}$ as follows:

$$V^{a,b} := \begin{bmatrix} \Phi(a,b) & \Phi(a,b+1) & \Phi_y(a,b) & \Phi_y(a,b+1) & 0 & 0 \\ \Phi(a+1,b) & \Phi(a+1,b+1) & \Phi_y(a+1,b) & \Phi_y(a+1,b+1) & 0 & 0 \\ \Phi_x(a,b) & \Phi_x(a,b+1) & 0 & 0 & 0 & 0 \\ \Phi_x(a+1,b) & \Phi_x(a+1,b+1) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

with $V^{a,b}$ containing the target function and derivative values on the corners. In general, box $V^{a,b}_{i:i+1,j:j+1}$, for $i = 0, 2, 4$, $j = 0, 2, 4$, contains the target values of $\frac{\partial^{(i+j)/2} g^{a,b}(x,y)}{\partial x^{i/2} \partial y^{j/2}}$ on the four corners. For this construction we set all higher order derivatives equal to zero on the corners. Finally we write the set of unknown polynomial coefficients in matrix form:

$$C^{a,b} := \begin{bmatrix} c_{0,0}^{a,b} & c_{0,1}^{a,b} & c_{0,2}^{a,b} & c_{0,3}^{a,b} & c_{0,4}^{a,b} & c_{0,5}^{a,b} \\ c_{1,0}^{a,b} & c_{1,1}^{a,b} & c_{1,2}^{a,b} & c_{1,3}^{a,b} & c_{1,4}^{a,b} & c_{1,5}^{a,b} \\ c_{2,0}^{a,b} & c_{2,1}^{a,b} & c_{2,2}^{a,b} & c_{2,3}^{a,b} & c_{2,4}^{a,b} & c_{2,5}^{a,b} \\ c_{3,0}^{a,b} & c_{3,1}^{a,b} & c_{3,2}^{a,b} & c_{3,3}^{a,b} & c_{3,4}^{a,b} & c_{3,5}^{a,b} \\ c_{4,0}^{a,b} & c_{4,1}^{a,b} & c_{4,2}^{a,b} & c_{4,3}^{a,b} & c_{4,4}^{a,b} & c_{4,5}^{a,b} \\ c_{5,0}^{a,b} & c_{5,1}^{a,b} & c_{5,2}^{a,b} & c_{5,3}^{a,b} & c_{5,4}^{a,b} & c_{5,5}^{a,b} \end{bmatrix}$$

The corner constraints for polynomial $g^{a,b}(x,y)$ give a system of equations which can be written in matrix form as follows:

$$A \cdot C^{a,b} \cdot A^T = V^{a,b}$$

Matrix A is invertible and thus the system is solvable and the coefficients are calculated using the formula:

$$C^{a,b} = A^{-1} \cdot V^{a,b} \cdot (A^{-1})^T$$

It is easy to validate that this interpolation scheme satisfies the edge constraints that we described earlier. The idea is that the marginalisation of $g^{a,b}(x,y)$ and its derivatives on box edges are one variable polynomials, the corners of the edge impose $d+1$ (independent) constraints on the marginal polynomial, where d is its degree, and these constraints are the same in the two adjacent boxes. Thus, the resulting polynomials of two adjacent boxes on the common edge are uniquely determined by the corner constraints and are equal, since the constraints are the same. We will give some illustrative examples of this argument, focusing on the edge $y = b, a \le x \le a+1$, which is shared by boxes (a,b) and $(a,b-1)$:

- First, let's study the zero-order continuity condition. We want to show that $g^{a,b}(x, y = b) = g^{a,b-1}(x, y = b)$ for all x. Both of these functions are polynomials of x of degree at most 5. The 6 constraints are $g^{a,b}(a, b) = g^{a,b-1}(a, b) = \Phi(a, b)$, $g^{a,b}(a + 1, b) = g^{a,b-1}(a + 1, b) = \Phi(a + 1, b)$, $g^{a,b}_x(a, b) = g^{a,b-1}_x(a, b) = \Phi_x(a, b)$, $g^{a,b}_x(a + 1, b) = g^{a,b-1}_x(a + 1, b) = \Phi_x(a + 1, b)$, $g^{a,b}_{xx}(a, b) = g^{a,b-1}_{xx}(a, b) = \Phi_{xx}(a, b)$ and $g^{a,b}_{xx}(a + 1, b) = g^{a,b-1}_{xx}(a + 1, b) = \Phi_{xx}(a + 1, b)$. The two polynomials of degree at most 5 satisfy the same 6 independent constraints, thus they are equal.
- Now let's consider the continuity of the derivative w.r.t. y. We want to show that $g^{a,b}_y(x, y = b) = g^{a,b-1}_y(x, y = b)$ for all x. The 6 constraints are $g^{a,b}_y(a, b) = g^{a,b-1}_y(a, b) = \Phi_y(a, b)$, $g^{a,b}_y(a + 1, b) = g^{a,b-1}_y(a + 1, b) = \Phi_y(a + 1, b)$, $g^{a,b}_{xy}(a, b) = g^{a,b-1}_{xy}(a, b) = 0$, $g^{a,b}_{xy}(a + 1, b) = g^{a,b-1}_{xy}(a + 1, b) = 0$, $g^{a,b}_{xxy}(a, b) = g^{a,b-1}_{xxy}(a, b) = 0$ and $g^{a,b}_{xxy}(a + 1, b) = g^{a,b-1}_{xxy}(a + 1, b) = 0$. Again, the two polynomials have degree at most 5 and they satisfy the same 6 independent constraints, thus they are equal.
- Finally, let's consider the continuity of the second derivative w.r.t. x. We want to show that $g^{a,b}_{xx}(x, y = b) = g^{a,b-1}_{xx}(x, y = b)$ for all x. The two polynomials have degree at most 4 and the 4 constraints are:
  - $g^{a,b}_{xx}(a, b) = g^{a,b-1}_{xx}(a, b) = 0$
  - $g^{a,b}_{xx}(a + 1, b) = g^{a,b-1}_{xx}(a + 1, b) = 0$
  - $\int_a^{a+1} g^{a,b}_{xx}(x, b)\, dx = \int_a^{a+1} g^{a,b-1}_{xx}(x, b)\, dx = \Phi_x(a + 1, b) - \Phi_x(a, b)$
  - $\int_a^{a+1} \left(\int_a^x g^{a,b}_{xx}(x', b)\, dx' + \Phi_x(a, b) \right) dx = \int_a^{a+1} \left(\int_a^x g^{a,b-1}_{xx}(x', b)\, dx' + \Phi_x(a, b) \right) dx =$ $\Phi(a + 1, b) - \Phi(a, b)$

B.4. Biquintic Interpolation does not introduce stationary points outside the areas of ITER solutions. The main idea of the PLS-hardness construction is that approximate second order stationary points of the continuous function $g(x, y)$, which is constructed by stitching together the small box polynomials $g^{a,b}(x, y)$, can only occur in areas that correspond to solutions of the embedded ITER problem.

A brute-force approach to proving this would be to consider each small box individually, based on the exact formula of its polynomial. We adopt a more efficient approach, similar to [16], where we assign the small boxes that do not lie in an ITER solution region into 4 different groups (see figure...). Moreover, we define a fifth group, group 0, where we assign all the boxes that lie close to an ITER solution. Boxes that belong to the same group have similar behaviour, regardless of their exact position in the tile. For groups $1 - 4$, the only information used to define a group is the direction of the gradients on the four corners and (partial) information about the colors of the corners. The step allowing us to drop the full information about the colours and only use a small set of conditions on the corner values is the use of symmetries w.r.t a set of transformations. In particular, for small boxes that do not lie in an ITER solution region we need to show that each possible combination of color and gradient direction on the corners creates a small box that is symmetric to the representative of one of the groups $1 - 4$. This means that applying a number of the transformations on the small box makes it fit to the conditions of the group. This way it is possible to classify each small box that appears in the construction into one of the 5 groups. At this point the only things we need to check is that:

(a) For each one of the groups $1 - 4$ small boxes that satisfy the conditions of the group could not contain any stationary points.

(b) The transformations we consider do not introduce spurious stationary points when applied to the small boxes.

First, we will focus on the transformations. Similarly to [16], the transformations we consider are the following:

1. Reflection with respect to the $y = b + 1/2$-axis. Applying this transformation moves the top two corners of the box at the bottom (and vice-versa) and flips the sign of the y-coordinate of each arrow.
2. Reflection with respect to the $x = a + 1/2$-axis. Applying this transformation moves the left two corners of the box to the right (and vice-versa) and flips the sign of the x-coordinate of each arrow.
3. Reflection with respect to the axis $y = x - a + b$. Applying this transformation swaps the corners $(a, b + 1)$ and $(a + 1, b)$ of the box and the x - and y-coordinate of the arrows at all four corners.
4. Reflection with respect to the axis $y = -x + a + 1 + b$. Applying this transformation swaps the corners (a, b) and $(a + 1, b + 1)$ of the box and the x - and y-coordinate of the arrows at all four corners and flips the sign of the $x-$ and $y-$ coordinates of each arrow.

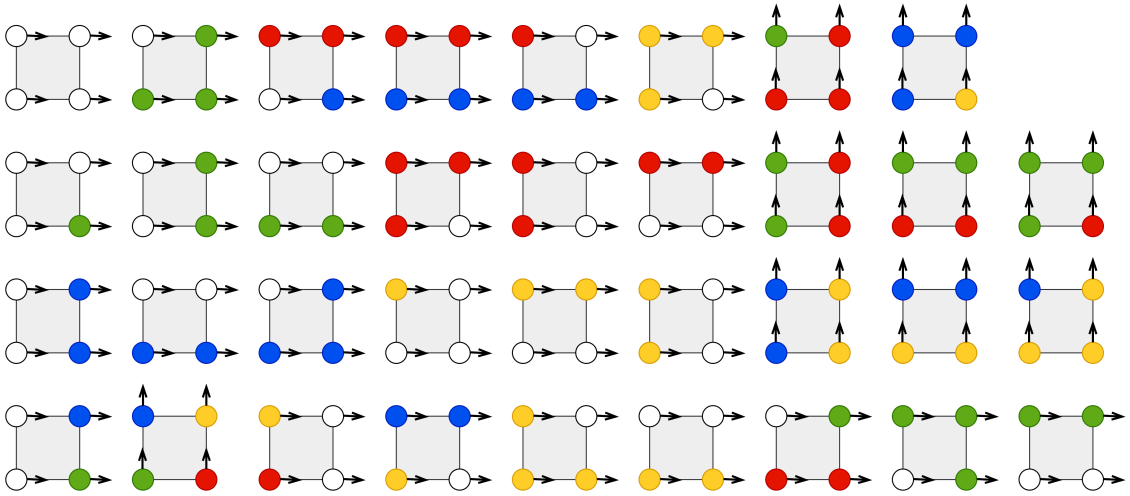


Figure 6: Group 1

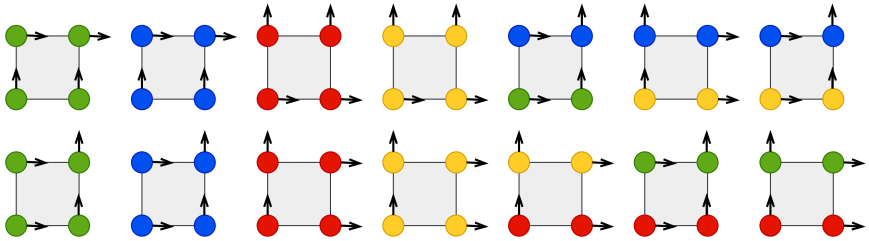


Figure 7: Group 2

5. Negation. Applying this transformation flips the sign of the values and the sign of the $x-$ and $y-$ coordinates of the arrows at the four corners.

It is easy to validate that the above transformations "commute" with biquintic interpolation. Without loss of generality we consider a box centered at $(0, 0)$. Applying transformation 1 to the box and then taking the biquintic interpolation of the reflected box yields the same result as taking the interpolation of the original box and then applying the reflection to the interpolated function (which corresponds to considering $(x, y) \rightarrow g^{a,b}(x, 1 - y)$). Similarly, transformation 2 has the same effect as taking $g^{a,b}(1 - x, y)$, transformation 3 the same as taking $g^{a,b}(y, x)$, transformation 4 the same as taking $g^{a,b}(1 - y, 1 - x)$ and transformation 5 the same as taking $-g^{a,b}(x, y)$. Obviously, if the interpolation polynomial does not have any ε-stationary points, then applying any reflection or taking the negation cannot change that property. Thus, applying combinations of the above transformations does not introduce spurious stationary points.

Now it remains to classify all the small boxes that do not lie close to the ITER solutions into the four groups and show that the conditions of each such group imply the inexistence of stationary points. The clustering of the boxes into the four groups is given in Fig. 6, Fig. 7, Fig. 8 and Fig. 9. We have to ensure that all small boxes that belong to Groups 1-4 do not contain any approximate SOSP. Note that it suffices to focus on SOSPs of a fixed constant accuracy ε_0, e.g. $\varepsilon_0 = 0.001$, because this would imply that the construction works for all $\varepsilon \leq \varepsilon_0$ and thus SOSP is PLS-hard for these values

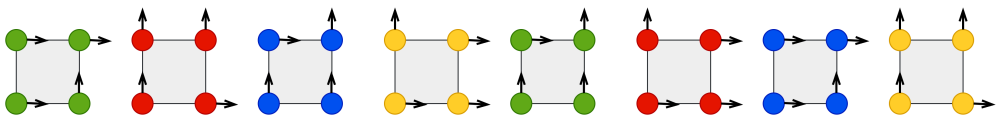


Figure 8: Group 3

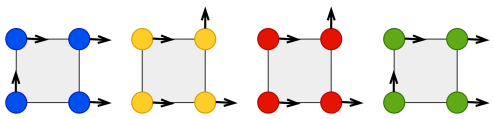


Figure 9: Group 4

of ε. Moreover, we observe that it suffices to show that boxes in Groups 1-4 do not contain any ε_0-FOSPs, as this would imply that they also do not contain any ε_0-SOSPs. This is exactly what we prove in the following Lemma.

Lemma B.1. *The polynomials $g^{a,b}(x,y)$ of small boxes that belong to Groups 1-4 do not contain any* 0.001*-FOSPs.*

Proof. Let $\varepsilon_0 := 0.001$. We will examine one by one the groups 1-4. For each group we calculate the expression of the polynomial, parameterised by the values a, b, c, d on the corners. The gradients on the corners are fixed for each group (see Fig. 10, Fig. 11, Fig. 12 and Fig. 13). Moreover, for groups 2,3,4, equality constraints hold for the polynomial values on the corners. In this case, the number of free parameters (a, b, c, d) is reduced, simplifying the analysis.
For group 1 the analysis is straightforward. We prove that the partial derivative w.r.t. x is always less than ε_0. For the other groups we need to divide the small box into two kinds of subregions: the ones where the derivative w.r.t. x is provably less than ε_0 and the ones where the derivative w.r.t. y is less than ε_0.

The boxes in Group 1 are symmetric (w.r.t. transformations $1-5$) to a box of the following form:

Conditions: $a \geq b+1, \quad c \geq d+1$

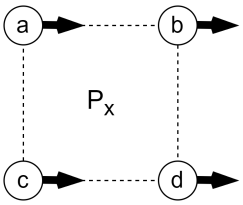


Figure 10: Group 1 characteristic image

The expression of the polynomial in the small box is the following:

$$\begin{aligned}P(x,y) = {} & c + x^5y^5(-36a+36b+36c-36d) + x^5y^4(90a-90b-90c+90d) + x^5y^3(-60a+60b+60c-60d)\\ & + x^5(-6c+6d+3) + x^4y^5(90a-90b-90c+90d) + x^4y^4(-225a+225b+225c-225d)\\ & + x^4y^3(150a-150b-150c+150d) + x^4(15c-15d-15/2) + x^3y^5(-60a+60b+60c-60d)\\ & + x^3y^4(150a-150b-150c+150d) + x^3y^3(-100a+100b+100c-100d) + x^3(-10c+10d+5)\\ & - x/2 + y^5(6a-6c) + y^4(-15a+15c) + y^3(10a-10c)\end{aligned}$$

The expression for the derivative (after simplification) is the following:

$$\begin{aligned}P_x(x,y) &= (1-x)^2x^2((-180a+180b+180c-180d)y^5 + (450a-450b-450c+450d)y^4\\ &\quad + (-300a+300b+300c-300d)y^3 - 30c+30d+15) - 1/2\\ &= (1-x)^2x^2((b-a)(180y^5-450y^4+300y^3) + (d-c)(-180y^5+450y^4-300y^3+30)+15) - 1/2\end{aligned}$$

For $0 \leq y \leq 1$ we have:

- $-180y^5+450y^4-300y^3+30 \geq 0 \Rightarrow$
  $(d-c)(-180y^5+450y^4-300y^3+30) \leq -(-180y^5+450y^4-300y^3+30)$
- $180y^5-450y^4+300y^3 \geq 0 \Rightarrow (b-a)(180y^5-450y^4+300y^3) \leq -(180y^5-450y^4+300y^3)$
- $(1-x)^2x^2 \geq 0$

Thus we obtain:

$$\begin{aligned} P_x(x,y) &\leq (1-x)^2x^2(-(180y^5-450y^4+300y^3)-(-180y^5+450y^4-300y^3+30)+15)-1/2 \\ &= -15(1-x)^2x^2-1/2 \\ &\leq -1/2 < -\varepsilon_0 \end{aligned}$$

The boxes in Group 2 are symmetric to a box of the following form:

Conditions: $c = a+1, \quad d = b+1, \quad c \geq d+1$
The expression of the polynomial in the small box is the following:

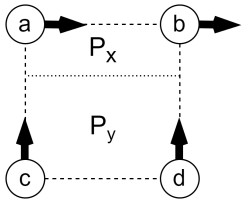


Figure 11: Group 2 characteristic image with areas of gradient activation

$$\begin{aligned} P(x,y) =& c + x^5y^5(-36a+36b+36c-36d+18) + x^5y^4(90a-90b-90c+90d-45) \\ &+x^5y^3(-60a+60b+60c-60d+30) + x^5(-6c+6d) + x^4y^5(90a-90b-90c+90d-45) \\ &+x^4y^4(-225a+225b+225c-225d+225/2) + x^4y^3(150a-150b-150c+150d-75) + x^4(15c-15d) \\ &+x^3y^5(-60a+60b+60c-60d+30) + x^3y^4(150a-150b-150c+150d-75) \\ &+x^3y^3(-100a+100b+100c-100d+50) + x^3(-10c+10d) - 3xy^5 + 15xy^4/2 - 5xy^3 \\ &+y^5(6a-6c+3/2) + y^4(-15a+15c-4) + y^3(10a-10c+3) - y/2 \end{aligned}$$

The expressions for the derivatives (after simplification) is the following:

$$P_x(x,y) = -(1-x)^2x^2(30c-30d-90y^5+225y^4-150y^3) - 3y^5 + 15y^4/2 - 5y^3$$

$$P_y(x,y) = (1-y)\left((90x^5-225x^4+150x^3-15x-43/2)(y^2-y^3) + y^3 - \frac{1}{2}y - \frac{1}{2}\right)$$

For $y < \frac{2}{3}$ we have:

- $90x^5-225x^4+150x^3-15x-43/2 < 0$ and $y^3 < y^2$, thus $(90x^5-225x^4+150x^3-15x-43/2)(y^2-y^3) < 0$
- $y^3 - \frac{1}{2}y - \frac{1}{2} < \frac{8}{27} - \frac{1}{2} = -\frac{11}{54}$
- $1 - y > \frac{1}{3}$

Putting it all together we have $P_y < -\frac{11}{162} < -\varepsilon_0$

For $y > \frac{2}{3}$ we have:

- $-(1-x)^2x^2 \leq 0$
- $30c - 30d \geq 30$ and $-90y^5+225y^4-150y^3 \geq -15$, thus $30c-30d-90y^5+225y^4-150y^3 > 15 > 0$
- $-3y^5+15y^4/2-5y^3 \leq -\frac{32}{81}$

Thus we have $P_x < -\frac{32}{81} < -\varepsilon_0$

The boxes in Group 3 are symmetric to a box of the following form:

Conditions: $d = c - 1, b = a - 1, a = c - 1$
The expression of the polynomial in the small box is the following:

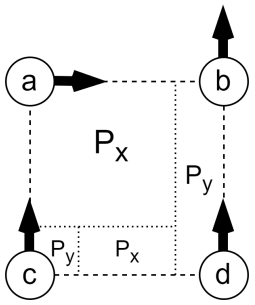


Figure 12: Group 3 characteristic image with areas of gradient activation

$$\begin{aligned}P(x,y) =& c + x^5y^5(-36a + 36b + 36c - 36d + 18) + x^5y^4(90a - 90b - 90c + 90d - 87/2)\\ &+x^5y^3(-60a + 60b + 60c - 60d + 27) + x^5(-6c + 6d) + x^4y^5(90a - 90b - 90c + 90d - 93/2)\\ &+x^4y^4(-225a + 225b + 225c - 225d + 225/2) + x^4y^3(150a - 150b - 150c + 150d - 70) + x^4(15c - 15d)\\ &+x^3y^5(-60a + 60b + 60c - 60d + 33) + x^3y^4(150a - 150b - 150c + 150d - 80)\\ &+x^3y^3(-100a + 100b + 100c - 100d + 50) + x^3(-10c + 10d) - 3xy^5 + 15xy^4/2 - 5xy^3\\ &+y^5(6a - 6c + 3/2) + y^4(-15a + 15c - 4) + y^3(10a - 10c + 3) - y/2\end{aligned}$$

The expressions for the derivatives (after simplification) is the following:

$P_x(x,y) = -(1-x)^2((-90y^5 + 435y^4/2 - 135y^3 + 30)x^2 + y^3(6y^2 - 30y/2 + 10)(x + 1/2))$
$P_y(x,y) = (1-x)y^2((-90x^4 + 285x^3/2 - 45x^2/2 - 45x/2 - 15/2)y^2 +$
$(174x^4 - 276x^3 + 44x^2 + 44x + 14)y - 81x^4 + 129x^3 - 21x^2 - 21x - 6) - 15y^2(y-1)^2 - 1/2$

One can validate (e.g. using Wolfram Mathematica) that the following inequalities hold:

- $-90x^4 + 285x^3/2 - 45x^2/2 - 45x/2 - 15/2 \le 0$ for $x \in [0,1]$
- $174x^4 - 276x^3 + 44x^2 + 44x + 14 < 30$, for $x \in [0,1]$
- $-81x^4 + 129x^3 - 21x^2 - 21x - 6 \le 0$, for $x \in [0,1]$
- $-90y^5 + 435y^4/2 - 135y^3 + 30 > 20$ for all $y \in [0,1]$.
- $6y^2 - 30y/2 + 10 \ge 1$ for all $y \in [0,1]$.

Thus, the partial derivatives are upper-bounded as follows:

$$P_x \le -(1-x)^2(20x^2 + y^3(x + 1/2))$$
$$P_y \le 30(1-x)y^3 - 15y^2(y-1)^2 - 1/2$$

For $0 \le x \le \frac{1}{61}$ and $0 \le y \le 1/5$ we have $P_y \le 30(1/5)^3 - 1/2 < -\varepsilon_0$.

For $1 - \frac{1}{61} \le x \le 1$ and $0 \le y \le 1$ we have $P_y \le 30/61 - 1/2 < -\varepsilon_0$.

For $0 \le x \le 1 - \frac{1}{61}$ and $1/5 \le y \le 1$ we have $P_x \le -(1-x)^2(20x^2 + 5^{-3}(x + 1/2)) < -0.003 < -\varepsilon_0$.

For $\frac{1}{61} \le x \le 1 - \frac{1}{61}$ and $0 \le y \le 1/5$ we have $P_x \le -20x^2(1-x)^2 \le -20\left(\frac{1}{61}\right)^2\left(1 - \frac{1}{61}\right)^2 < -\varepsilon_0$.

The boxes in Group 4 are symmetric to a box of the following form:

Conditions: $c - a = 1, d - b = 1, c - d = 1, a - b = 1$
The expression of the polynomial in the small box is the following:

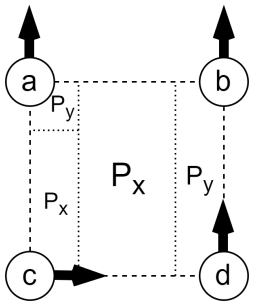


Figure 13: Group 4 characteristic image with areas of gradient activation

$$\begin{aligned}P(x,y) =& c + 36x^5y^5(-a+b+c-d) + 3x^5y^4(60a-60b-60c+60d-1)/2 + 3x^5y^3(-20a+20b+20c-20d+1)\\ &-3x^5y + 3x^5(-4c+4d+1)/2 + 3x^4y^5(60a-60b-60c+60d+1)/2 + 225x^4y^4(-a+b+c-d)\\ &+5x^4y^3(30a-30b-30c+30d-1) + 15x^4y/2 + x^4(15c-15d-4) + 3x^3y^5(-20a+20b+20c-20d-1)\\ &+5x^3y^4(30a-30b-30c+30d+1) + 100x^3y^3(-a+b+c-d) - 5x^3y + x^3(-10c+10d+3) + 3xy^5\\ &-15xy^4/2 + 5xy^3 - x/2 + 3y^5(4a-4c+1)/2 + y^4(-30a+30c-7)/2 + 2y^3(5a-5c+1)\end{aligned}$$

The expressions for the derivatives (after simplification) is the following:

$$P_x(x,y) = -(1-x)^2((15y^4/2 - 15y^3 + 15y + 45/2)x^2 + (1-y)^3(6y^2+3y+1)(x+1/2))$$

$$P_y(x,y) = (1-x)^3(1-y)^2\left(-\frac{15}{2}(x+1)y^2 + (6x^2+3x+1)(y+1/2)\right) - 15y^2(y-1)^2 - 1/2$$

- $15y^4/2 - 15y^3 + 15y + 45/2 \geq 45/2$ for $y \in [0,1]$
- $(6x^2+3x+1)(y+1/2) \leq (6+3+1)(1+1/2) = 15$ for $x \in [0,1]$ and $y \in [0,1]$

$P_x \leq -(1-x)^2\left(\frac{45}{2}x^2 + (1-y)^3(x+1/2)\right)$
$P_y \leq 15(1-x)^3(1-y)^2 - 15y^2(y-1)^2 - 1/2$

For $x \in [0.1, 0.9]$ and $y \in [0,1]$ we have $P_x \leq -\frac{45}{2}(1-x)^2x^2 < -0.1 < -\varepsilon_0$
For $x \in [0.9,1]$ and $y \in [0,1]$ we have $P_y \leq 15(1-x)^3 - 1/2 < \frac{15}{1000} - 1/2 < -\varepsilon_0$
For $x \in [0, 0.1]$ and $y \in [0, 0.85]$ we have $P_x \leq -\frac{1}{2}(1-x)^2(1-y)^3 \leq -\frac{1}{2}(0.9)^2(0.15)^3 < -\varepsilon_0$
For $x \in [0, 0.1]$ and $y \in [0.85, 1]$ we have $P_y \leq 15(1-y)^2 - 1/2 < 15 \cdot (0.15)^2 - 1/2 < -\varepsilon_0$

■

B.5. Periodic Structure of the function f. We want our function f to be bounded, even though it is defined over the plane $\mathbb{R}^2$. A simple solution for this is to make f periodic. We choose to construct f by repeating tiles of the function $g : [0, M] \to \mathbb{R}^2$ defined in Appendix B.1, Appendix B.2 and Appendix B.3. Therefore f is defined as:

$$f(x,y) = g\left(x - M \cdot \left\lfloor \frac{x}{M} \right\rfloor, y - M \cdot \left\lfloor \frac{y}{M} \right\rfloor\right)$$

By the definition of g, it follows that the equations below hold:

$$g(x,0) = g(x,M) \quad \text{and} \quad g(0,y) = g(M,y) \quad \text{for all } x,y \in [0,M]$$
$$\nabla g(x,0) = \nabla g(x,M) \quad \text{and} \quad \nabla g(0,y) = \nabla g(M,y) \quad \text{for all } x,y \in [0,M]$$
$$\nabla^2 g(x,0) = \nabla^2 g(x,M) = 0 \quad \text{and} \quad \nabla^2 g(0,y) = \nabla^2 g(M,y) = 0 \quad \text{for all } x,y \in [0,M]$$

More specifically, the construction of the grid guarantees the continuity between the right and the left side of the grid and the continuity between the top and the bottom side of the grid. That is, the first two lines of the above equations hold for the points of the grid. The continuity between the points along the sides of the tiles is guaranteed due to the continuity properties of the biquintic interpolation described in Appendix B.3. In particular, if the values and gradients of two adjacent small boxes are the same on their two common corners, then the polynomials as well as their gradients and Hessians are also the same along their common side.

Thus, we ensure the C^2 continuity of f.

B.6. Lipschitz Constants and Boundedness.

Lemma B.2. *The constructed function f is B-bounded, L-smooth and ρ-Hessian Lipschitz in $\mathbb{R}^2$, with $B \leq 2^{15}M$, $L \leq 2^{22}M$ and $\rho \leq 2^{23}M$.*

First, we bound the absolute values of the polynomial coefficients. As we saw earlier, the matrix containing them is calculated by the formula

$$C^{a,b} = A^{-1} \cdot V^{a,b} \cdot (A^{-1})^T$$

We can bound the Frobenius norm of this matrix as follows:

$$\|C^{a,b}\| \leq \|A^{-1}\| \|V^{a,b}\| \|(A^{-1})^T\|$$

One can easily verify that $\|A^{-1}\| = \|(A^{-1})^T\| < 2^5$. For the Frobenius norm of $V^{a,b}$ we have $\|V^{a,b}\| = \sqrt{4\left(\frac{1}{2}\right)^2 + \Phi^2(a,b) + \Phi^2(a+1,b) + \Phi^2(a,b+1) + \Phi^2(a+1,b+1)} \leq \sqrt{4\left(\frac{1}{2}\right)^2 + 4(8M)^2} < 16M+1$, where we used the fact that $|\Phi(x,y)|$ is upper bounded by $8M$ in the $M \times M$ tile. Putting it all together we have that $\|C^{a,b}\| < 2^{10}(16M+1)$. Next we will show that the polynomials inside each small box are C^2 continuous and we will upper bound their smoothness constant and Hessian Lipschitz constant. For the smoothness constant L we have

$$\begin{aligned}
L &= \max_{x \in [a,a+1], y \in [b,b+1]} \{\lambda_{max}(\Phi(g^{a,b}(x,y)))\} \\
&\leq \max_{x \in [a,a+1], y \in [b,b+1]} \{\|\Phi(g^{a,b}(x,y))\|\} \\
&\leq \max_{x \in [a,a+1], y \in [b,b+1]} \{|g^{a,b}_{xx}(x,y)| + |g^{a,b}_{yy}(x,y)| + 2|g^{a,b}_{xy}(x,y)|\} \\
&\leq (20 + 20 + 2 \cdot 25)\|C^{a,b}\| \\
&= 90\|C^{a,b}\|
\end{aligned}$$

For the last inequality we used the fact that for $x \in [a, a+1]$ and $y \in [b, b+1]$:

- $|g^{a,b}_{xx}(x,y)| \leq \sum_{i=0}^{5}\sum_{j=0}^{5} |c^{a,b}_{i,j}| i(i-1)(x-a)^{i-2}(y-b)^j \leq \sum_{i=0}^{5}\sum_{j=0}^{5} 20|c^{a,b}_{i,j}| = 20\|C^{a,b}\|$
- $|g^{a,b}_{yy}(x,y)| \leq \sum_{i=0}^{5}\sum_{j=0}^{5} |c^{a,b}_{i,j}| j(j-1)(x-a)^{i}(y-b)^{j-2} \leq \sum_{i=0}^{5}\sum_{j=0}^{5} 20|c^{a,b}_{i,j}| = 20\|C^{a,b}\|$
- $|g^{a,b}_{xy}(x,y)| \leq \sum_{i=0}^{5}\sum_{j=0}^{5} |c^{a,b}_{i,j}| ij(x-a)^{i-1}(y-b)^{j-1} \leq \sum_{i=0}^{5}\sum_{j=0}^{5} 25|c^{a,b}_{i,j}| = 25\|C^{a,b}\|$

Thus we have that the polynomial in each small box is L-smooth with $L \leq 90\|C^{a,b}\| < 2^{22}M$

Next we will bound the Hessian-Lipschitz constant of the polynomials in each small box. For this purpose we bound the Lipschitz constant of each of the entries of the Hessian. For $g^{a,b}_{xx}$ we have:

$$L_{xx,ab} = \max_{x \in [a,a+1], y \in [b,b+1]} \|\nabla g^{a,b}_{xx}(x,y)\|$$

$$
\begin{aligned}
&\leq \max_{x\in[a,a+1],y\in[b,b+1]} \{|g^{a,b}_{xxx}(x,y)| + |g^{a,b}_{xxy}(x,y)|\} \\
&\leq \max_{x\in[a,a+1],y\in[b,b+1]} \{\sum_{i=0}^{5}\sum_{j=0}^{5} |c^{a,b}_{i,j}| i(i-1)(i-2)(x-a)^{i-3}(y-b)^{j} + \sum_{i=0}^{5}\sum_{j=0}^{5} |c^{a,b}_{i,j}| i(i-1)j(x-a)^{i-2}(y-b)^{j-1}\} \\
&\leq (5\cdot 4\cdot 3 + 5\cdot 4\cdot 5)\|C^{a,b}\| \\
&= 160\|C^{a,b}\|
\end{aligned}
$$

Similarly we obtain $L_{yy,ab} = \max_{x\in[a,a+1],y\in[b,b+1]} \|\nabla g^{a,b}_{yy}(x,y)\| \leq 160\|C^{a,b}\|$ and $L_{xy,ab} = \max_{x\in[a,a+1],y\in[b,b+1]} \|\nabla g^{a,b}_{xy}(x,y)\| \leq 200\|C^{a,b}\|$.

For the Hessian Lipschitzness we have:

$$
\begin{aligned}
\|\Phi(g^{a,b}(x_1,y_1)) - \Phi(g^{a,b}(x_2,y_2))\| &\leq \sqrt{(L^2_{xx,ab} + L^2_{yy,ab} + 2L^2_{xy,ab})\left((x_1-x_2)^2 + (y_1-y_2)^2\right)} \\
&\leq \|C^{a,b}\|\sqrt{2(160^2+200^2)}\sqrt{(x_1-x_2)^2+(y_1-y_2)^2} \\
&\leq 400\|C^{a,b}\|\sqrt{(x_1-x_2)^2+(y_1-y_2)^2} \\
&\leq 2^{11}100(16M+1)\|\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}
\end{aligned}
$$

From the above we can conclude that $\rho = 2^{23}M$ is a Lipschitz constant of the Hessian for all small+ boxes.

Finally we will show that the boundedness assumption is satisfied for the polynomial in each small box. For $x \in [a, a+1]$ and $y \in [b, b+1]$ we have:

$$
|g^{a,b}(x,y)| \leq \sum_{i=0}^{5}\sum_{j=0}^{5} |c^{a,b}_{i,j}|(x-a)^i(y-b)^j \leq \sum_{i=0}^{5}\sum_{j=0}^{5} |c^{a,b}_{i,j}| = \|C^{a,b}\| < 2^{10}(16M+1) < 2^{15}M
$$

Thus the polynomials in each small box are B bounded with $B < 2^{15}M$.

It remains to show that the boundedness and smoothness properties also hold for f. Since f is constructed by tiling small boxes and B-boundedness holds for all small boxes it trivially holds for f as well. The smoothness and Hessian Lipschitzness constants L and ρ are also passed on from the small boxes to f due to a simple argument that can be found in the proof of Lemma 4.2 in [13]. The argument is originally stated for the smoothness constant L but can it can be also shown for ρ following similar steps.

B.7. Turing machines that compute $f, \nabla f, \nabla^2 f$. The last part of the proof is to prove the following lemma.

Lemma B.3. *The exist polynomial-time Turing machines $\mathcal{C}_f$, $\mathcal{C}_{\nabla f}$, $\mathcal{C}_{\nabla^2 f}$ such that given two numbers $x, y \in [0,1]$ with bit complexity $b \geq \text{len}(x)$ and $b \geq \text{len}(y)$ we can compute the value, the gradient of f and the hessian of f at any point in time that is polynomial in b and in the size of the boolean circuit C of* ITER.

Proof. It is easy to see that the computation of modulo $z \mapsto z - M \cdot \left\lfloor \frac{z}{M} \right\rfloor$ can be done in time polynomial in $\text{len}(z)$ and $\text{len}(M)$. $\text{len}(z)$ will be bounded by b and M is a natural number such that $M = \mathcal{O}(2^n)$. This gives $\text{len}(M) = \mathcal{O}(n)$. Since C is a boolean circuit with n inputs and n outputs, we get that $\mathcal{O}(n)$ is certainly polynomial in the size of C. Now it suffices to show that we can compute $g(x,y)$ in polynomial time. To achieve this we need to identify the type of the small box where (x,y) belongs. To do this outside the PLS boxes, we need time linear in b. Inside the PLS boxes, on the other hand, we need to evaluate the circuit C with input u that corresponds to the column of the medium box that (x,y) belongs to, and with input v that corresponds to the row of the medium box that (x,y) belongs to. Therefore, we need to evaluate $C(u), C(v)$ as well as $C(C(u)), C(C(v))$. If we have these values then we can identify the type of the small box that (x,y) belongs to. Thus we need to evaluate C four times, which takes linear time in the size of C. Finally, once we identify the small box, we need to compute the biquintic interpolation which involves solving a linear system and computing a five degree polynomial with numbers that use at most $\max\{b, \text{len}(M)\}$ bits. Note that both of these can be done in time polynomial in the description of the number and thus we conclude that there exists an efficient Turing machine that computes g which implies an efficient Turing machine that compute f. In a similar manner, once we get the interpolation polynomial at the

small box it is easy to also compute the gradient and the hessian of this polynomial, and thus we can get the polynomial-time Turing machines that compute ∇f and $\nabla^2 f$.

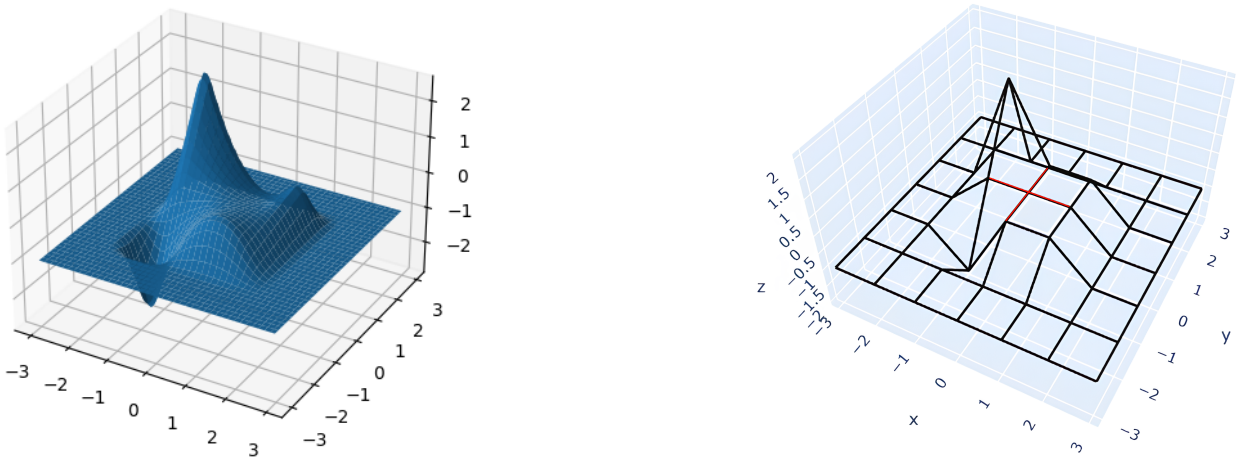
■

C Proof of Theorem 2

Proof. From Theorem 3, we get that HESSIAN-FOSP is PLS-hard. Now, consider any SOSP instance. Let x^* be a solution to that SOSP instance. Since x^* is an ε-SOSP, then automatically is an ε-FOSP, as well. Now we can construct a HESSIAN-FOSP instance using the same inputs as in the SOSP instance above. Then, x^* is trivially a solution to the FOSP instance. Thus, HESSIAN-FOSP (polynomially) reduces to SOSP which implies that SOSP is PLS-hard.

■

D Example where simple coordinate descent fails



(a) Continuous plot of $f(x, y)$

(b) $f(x, y)$ on the LocalOpt grid

Figure 14: An example where simple coordinate descent fails to show the membership of SOSP in PLS: In the subplot (a) we illustrate the continuous function derived after applying the biquintic interpolation. One can easily verify that the function is C^2 continuous. In the subplot (b) we illustrate the derived function on the grid of the LocalOpt instance. The red "cross" demonstrates that the function has equal (zero) values at $(0, 0)$ and its immediate neighbors. Thus, coordinate-descent based LocalOpt has a solution at $(0, 0)$, a point that is FOSP but not SOSP.

The exact function used in the counterexample is the following:

$$f(x,y) = \begin{cases} xy(x-1) & \text{if } x,y \in [-1,1] \times [-1,1] \\ P_1(x,y) & \text{if } x,y \in [-2,-1]^2 \\ P_2(x,y) & \text{if } x,y \in [-2,-1] \times [-1,0] \\ P_3(x,y) & \text{if } x,y \in [-2,-1] \times [0,1] \\ P_4(x,y) & \text{if } x,y \in [-2,-1] \times [1,2] \\ P_5(x,y) & \text{if } x,y \in [-1,0] \times [-2,-1] \\ P_6(x,y) & \text{if } x,y \in [-1,0] \times [1,2] \\ P_7(x,y) & \text{if } x,y \in [0,1] \times [-2,-1] \\ P_8(x,y) & \text{if } x,y \in [0,1] \times [1,2] \\ P_9(x,y) & \text{if } x,y \in [1,2] \times [-2,-1] \\ P_{10}(x,y) & \text{if } x,y \in [1,2] \times [-1,0] \\ P_{11}(x,y) & \text{if } x,y \in [1,2] \times [0,1] \\ P_{12}(x,y) & \text{if } x,y \in [1,2] \times [1,2] \\ -1 & \text{otherwise} \end{cases}$$

where

- $P_1(x,y) = -162(x+2)^5(y+2)^5 + 394(x+2)^5(y+2)^4 - 248(x+2)^5(y+2)^3 + 387(x+2)^4(y+2)^5 - 941(x+2)^4(y+2)^4 + 592(x+2)^4(y+2)^3 - 237(x+2)^3(y+2)^5 + 576(x+2)^3(y+2)^4 - 362(x+2)^3(y+2)^3 - 1$

- $P_2(x,y) = 22y(x+2)^5 - 53y(x+2)^4 + 33y(x+2)^3 + 6(x+2)^5 - 15(x+2)^4 + 10(x+2)^3 - 1$

- $P_3(x,y) = 22y(x+2)^5 - 53y(x+2)^4 + 33y(x+2)^3 + 6(x+2)^5 - 15(x+2)^4 + 10(x+2)^3 - 1$

- $P_4(x,y) = 22y(x+2)^5 - 53y(x+2)^4 + 33y(x+2)^3 - 234(x+2)^5(y-1)^5 + 596(x+2)^5(y-1)^4$
  $- 412(x+2)^5(y-1)^3 + 6(x+2)^5 + 567(x+2)^4(y-1)^5 - 1444(x+2)^4(y-1)^4$
  $+ 998(x+2)^4(y-1)^3 - 15(x+2)^4 - 357(x+2)^3(y-1)^5 + 909(x+2)^3(y-1)^4$
  $- 628(x+2)^3(y-1)^3 + 10(x+2)^3 - 1$

- $P_5(x,y) = 27x(y+2)^5 - 66x(y+2)^4 + 42x(y+2)^3 - 9(x+1)^2(y+2)^5 + 22(x+1)^2(y+2)^4$
  $- 14(x+1)^2(y+2)^3 + 15(y+2)^5 - 37(y+2)^4 + 24(y+2)^3 - 1$

- $P_6(x,y) = -9x^2y^5 + 68x^2y^4 - 198x^2y^3 + 276x^2y^2 - 184x^2y + 48x^2 + 9xy^5 - 68xy^4 + 198xy^3$
  $- 276xy^2 + 184xy - 48x - 6y^5 + 45y^4 - 130y^3 + 180y^2 - 120y + 31$

- $P_7(x,y) = -9x^2(y+2)^5 + 22x^2(y+2)^4 - 14x^2(y+2)^3 + 9x(y+2)^5 - 22x(y+2)^4$
  $+ 14x(y+2)^3 + 6(y+2)^5 - 15(y+2)^4 + 10(y+2)^3 - 1$

- $P_8(x,y) = x^2y - 9x^2(y-1)^5 + 23x^2(y-1)^4 - 16x^2(y-1)^3 - xy + 9x(y-1)^5 - 23x(y-1)^4$
  $+ 16x(y-1)^3 - 6(y-1)^5 + 15(y-1)^4 - 10(y-1)^3$

- $P_9(x,y) = -9x(y+2)^5 + 22x(y+2)^4 - 14x(y+2)^3 + 2(x-1)^5(y+2)^4 - 4(x-1)^5(y+2)^3$
  $- 9(x-1)^4(y+2)^5 + 17(x-1)^4(y+2)^4 - 4(x-1)^4(y+2)^3 + 21(x-1)^3(y+2)^5$
  $- 48(x-1)^3(y+2)^4 + 26(x-1)^3(y+2)^3 - 9(x-1)^2(y+2)^5 + 22(x-1)^2(y+2)^4$
  $- 14(x-1)^2(y+2)^3 + 15(y+2)^5 - 37(y+2)^4 + 24(y+2)^3 - 1$

- $P_{10}(x,y) = -4x^5y - 6x^5 + 31x^4y + 45x^4 - 93x^3y - 130x^3 + 134x^2y + 180x^2 - 92xy - 120x + 24y + 31$

- $P_{11}(x,y) = (x-1)(-4y(x-1)^4 + 11y(x-1)^3 - 9y(x-1)^2 + y(x-1) + y - 6(x-1)^4 + 15(x-1)^3 - 10(x-1)^2)$

- $P_{12}(x,y) = 72x^5y^5 - 542x^5y^4 + 1572x^5y^3 - 2184x^5y^2 + 1456x^5y - 384x^5 - 549x^4y^5 + 4133x^4y^4 - 11988x^4y^3$
  $+ 16656x^4y^2 - 11104x^4y + 2928x^4 + 1617x^3y^5 - 12174x^3y^4 + 35314x^3y^3 - 49068x^3y^2 + 32712x^3y$
  $- 8624x^3 - 2286x^2y^5 + 17212x^2y^4 - 49932x^2y^3 + 69384x^2y^2 - 46256x^2y + 12192x^2 + 1548xy^5$
  $- 11656xy^4 + 33816xy^3 - 46992xy^2 + 31328xy - 8256x - 408y^5 + 3072y^4 - 8912y^3 + 12384y^2$
  $- 8256y + 2175$

The above function is also illustrated in Fig. 14. It is C^2 continuous, as it was constructed with biquintic interpolation and tiling and as we have showed earlier, biquintic interpolation preserves second order continuity in tiling.

E PLS-completeness of Finding Approximate FOSPs of C^2 continuous functions in Unconstrained Optimization

In Theorem 3 we have shown that the HESSIAN-SOSP problem is PLS-hard. We proceed with the following theorem that shows that the problem is complete in PLS.

Theorem 4. *HESSIAN-FOSP is PLS-complete.*

Proof. From Theorem 3, we get that HESSIAN-SOSP problem is PLS-hard. Now it remains to show that the membership of the problem in PLS: The membership of the HESSIAN-FOSP problem in PLS is easily derived as a consequence of the membership of FOSP in PLS (proved in [16]), using the fact that the requirement for f to be a C^2 continuous function is stronger than the requirement for f to be a C^1 continuous function.

■

F Proof of Corollary 2

For the query complexity we will consider the same construction of a C^2 function f that we used to show the PLS-Hardness. Given an algorithm that calculates ϵ-SOSPs, we embed an appropriate instance of the ITER that forces the algorithm into making a number of queries exponential in the size of the ITER. The first step for this argument is that the black box version of ITER has a query lower bound of 2^n.

Lemma F.1. *For any algorithm $\mathcal{A}$ that solves the black box version of the* ITER *there exist an* ITER *instance of size n that requires 2^n queries by the algorithm $\mathcal{A}$ to be solved.*

The proof of this lemma can be found in the appendix section D of [16].

In Section Appendix B we construct a function f, such that any 0.001-FOSP of f reveals a solution to ITER with input C. Thus, any algorithm that finds a 0.001-FOSP of f could be used as an algorithm for solving ITER. Combining this property with Lemma F.1, we conclude that any algorithm that finds a 0.001-FOSP of f will in the worst-case make at least $2^n/N_c$ queries to f, where N_c is the maximal number of calls to C per query. In our construction N_c is equal to 4.

Next, we will establish a connection between the complexity 2^{n-2} and the accuracy parameter ε of approximate FOSP. The parameter M that we use to construct f satisfies $M = \Theta(2^n)$, which means that any algorithm that finds a 0.001-stationary point in functions that we construct will in the worst-case take time at least $\Omega(M)$. We consider the normalised function $\tilde{f}(x) = \frac{1}{M} \cdot f(x)$. Finding a 0.001-stationary point for f is equivalent to finding a $\frac{1}{1000M}$-stationary point of $\tilde{f}$ so the new accuracy parameter ε is $\frac{1}{1000M}$. $\tilde{f}$ is $B-$bounded with $B = O(1)$, and has smoothness and Hessian Lipschitzness constants $L = O(1)$ and $\rho = O(1)$. Still, any algorithm would need $\Omega(M)$ time to find a $\frac{1}{1000M}$-stationary point. Hence, any algorithm that finds an ε-FOSP for $\tilde{f}$ will need in the worst case $\Omega(1/\varepsilon)$ queries. This query complexity continuous to hold for calculating ε-SOSPs, since any approximate SOSP is also an approximate FOSP. This completes the proof of Corollary 2.

G Extra Definitions

G.1. Polynomial-time Reductions.

Definition 3 (Polynomial-time Reduction)**.** A search problem P_1 is polynomial-time reducible to P_2 if there exist polynomial-time computable functions $f : \{0,1\}^* \to \{0,1\}^*$ and $g : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ with the following properties: (a) if x is an input to P_1, then $f(x)$ is an input to P_2, and (b) if y is a solution to P_2 on input $f(x)$, then $g(x;y)$ is a solution to P_1 on input x.

H Promise/Violation White Box Problem Definitions

There is no known way of syntactically enforcing a Turing machine to be Lipschitz-continuous, Hessian-continouous and/or bounded. There are two ways to handle this issue: (a) consider the **promise version** of the problem (as we did in the main body of the paper), where we restrict our attention to instances that satisfy these conditions, or (b) introduce **violation** versions of the problem, following the work of Daskalakis and Papadimitriou [10], i.e. allow as a solution a witness of the fact that one of the conditions is not satisfied. The first option is more natural, but the second option ensures that the problem is formally in TFNP.

Problem 6: FOSP (Violation version)

Require: precision parameter $\varepsilon > 0$

$B, L > 0$ and $f \in \mathcal{F}(B, L, \infty)$

Turing machines representing f and ∇f

Find: $x^* \in \mathbb{R}^d$: $\|\nabla f(x^*)\| \leq \varepsilon$

Alternatively, we also accept one of the following violations as a solution:

- $x, y \in \mathbb{R}^d$ that violate the smoothness of f.
- $x \in \mathbb{R}^d$ such that $|f(x)| > B$,
- $x, y \in \mathbb{R}^d$ that certify that $\mathcal{C}_{\nabla f}$ computes ∇f incorrectly.

Problem 7: HESSIAN-FOSP (Violation version)

Require: precision parameter $\varepsilon > 0$

$B, L, \rho > 0$ and $f \in \mathcal{F}(B, L, \rho)$

Turing machines representing f, ∇f and $\nabla^2 f$

Find: $x^* \in \mathbb{R}^d$: $\|\nabla f(x^*)\| \leq \varepsilon$

Alternatively, we also accept one of the following violations as a solution:

- $x, y \in \mathbb{R}^d$ that violate the smoothness of f.
- $x \in \mathbb{R}^d$ such that $|f(x)| > B$,
- $x, y \in \mathbb{R}^d$ that certify that $\mathcal{C}_{\nabla f}$ computes ∇f incorrectly.
- $x, y \in \mathbb{R}^d$ that certify that $\mathcal{C}_{\nabla^2 f}$ computes $\nabla^2 f$ incorrectly.

Problem 8: SOSP (Violation version)

Require: precision parameter $\varepsilon > 0$
$B, L, \rho > 0$ and $f \in \mathcal{F}(B, L, \rho)$
Turing machines representing f, ∇f and $\nabla^2 f$

Find: $x^* \in \mathbb{R}^d$: $\|\nabla f(x^*)\| \leq \varepsilon$, $\lambda_{min}(\nabla^2 f(x^*)) \geq -\sqrt{\rho\varepsilon}$

Alternatively, we also accept one of the following violations as a solution:

- $x, y \in \mathbb{R}^d$ that violate the smoothness or the Hessian-Lipschitzness of f.
- $x \in \mathbb{R}^d$ such that $|f(x)| > B$,
- $x, y \in \mathbb{R}^d$ that certify that $\mathcal{C}_{\nabla f}$ computes ∇f incorrectly.
- $x, y \in \mathbb{R}^d$ that certify that $\mathcal{C}_{\nabla^2 f}$ computes $\nabla^2 f$ incorrectly.

Remark H.1. Our reductions also apply both to the promise and violation versions of SOSP and HESSIAN-FOSP.