The Complexity of Computing KKT Solutions of Quadratic Programs

John Fearnley
University of Liverpool
United Kingdom
john.fearnley@liverpool.ac.uk

Paul W. Goldberg
University of Oxford
United Kingdom
paul.goldberg@cs.ox.ac.uk

Alexandros Hollender
University of Oxford
United Kingdom
alexandros.hollender@cs.ox.ac.uk

Rahul Savani
Alan Turing Institute and University of Liverpool
United Kingdom
rahul.savani@liverpool.ac.uk

ABSTRACT

It is well known that solving a (non-convex) quadratic program is NP-hard. We show that the problem remains hard even if we are only looking for a Karush-Kuhn-Tucker (KKT) point, instead of a global optimum. Namely, we prove that computing a KKT point of a quadratic polynomial over the domain $[0,1]^n$ is complete for the class CLS = PPAD ∩ PLS.

CCS CONCEPTS

• **Theory of computation → Problems, reductions and completeness**; **Quadratic programming**.

KEYWORDS

Quadratic Programming, KKT, Continuous Local Search

ACM Reference Format:
John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. 2024. The Complexity of Computing KKT Solutions of Quadratic Programs. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649647

1 INTRODUCTION

Quadratic programming (QP) is the problem of optimising a quadratic function of a collection of real variables, subject to linear constraints on those variables. It has widespread applications, numerous software implementations, and an extensive literature on its theoretical analysis, dating back more than 50 years. A fairly standard formulation is the following:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} f(x) &:= x^\top Q x + c^\top x \\ \text{subject to } a_i^\top x &\le b_i, \forall i \in \{1,\dots,m\}\ . \end{aligned} \tag{1}$$

In (1) the matrix Q is usually (and without loss of generality) taken to be symmetric, and there has been much work on restrictions of the problem based on assumed properties of Q, some of which we touch on below. The main result of the present paper is that for QP it is computationally hard to compute a *Karush-Kuhn-Tucker* (KKT) point, an important kind of solution consisting of one that is locally optimal with respect to gradient descent. Moreover, our hardness result applies to a special case of interest known as "box constraints" (e.g., [3, 8]), in which the feasible region (i.e., the region $a_i^\top x \le b_i, \forall i \in \{1,\dots,m\}$) consists of an axis-aligned hypercuboid; here we use $[0,1]^n$. (Indeed, the general version of the problem sometimes assumes that the linear constraints include $x \in [0,1]^n$ (e.g., [5]), which guarantees that the feasible region is compact.)

STOC '24, June 24–28, 2024, Vancouver, BC, Canada
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649647

KKT solutions, and other types of solution. Informally, a KKT point is one that constitutes a local optimum of gradient descent. It may be a point at which the gradient is zero (a stationary point), or one where the gradient is non-zero, but further downhill progress is obstructed by one or more of the boundary constraints. A key feature of KKT solutions of (1) is that they have concise certificates: roughly, the gradient together with the binding constraints at a point of interest. Moreover, if the domain is compact, there is guaranteed to be at least one KKT point, since the global optimum is a KKT point. These two observations indicate that the problem of searching for a KKT point belongs to the complexity class TFNP, search problems in which easily-checkable solutions must exist. In particular, this means that the problem is not expected to be NP-hard, unless NP = co-NP [21].

Apart from KKT points, the other main solution concepts for continuous optimisation problems are the following:

- Global optimum, x for which $f(x) \le f(x')$ for all x' in the feasible region;
- Stationary point (a.k.a. critical point), where $\nabla f(x) = 0$;
- Local minimum, x where for some $\epsilon > 0$ we have $f(x) \le f(x')$ for all x' within ϵ of x; at a *strict* local minimum, we have $f(x) < f(x')$ for all x' in the feasible region within ϵ of x.

For QPs of the form (1), stationary points are not guaranteed to exist, and global optima and local minima are NP-hard to compute. KKT points relate to other solution concepts as follows. Global or local minima are KKT points, thus KKT points are at least as easy to compute as global/local minima. Any stationary point is a KKT point, but stationary points are not guaranteed to exist, even for the box-constrained feasible regions that we consider here. To see that stationary points can be searched for in polynomial time, note that they are given by $\nabla f = 0$, hence define a linear subspace, and checking for points in the subspace that satisfy the boundary constraints $a_i^\top x \leq b_i$ amounts to solving an LP. In the unconstrained case (in which there are no boundary constraints) stationary points are the same as KKT points, so then the problem of searching for either kind of solution is tractable. A stationary point need not be a local minimum: for the problem of minimising $f(x) = -x^2$ over the interval $[0, 1]$, $x = 0$ is a stationary point but not a local minimum.

Hardness results for global/local optima. There has been much work studying the circumstances in which one can efficiently compute a solution of one of the above kinds, also on determining whether a given point x is one of the above solutions.

The global optimisation problem for quadratic programming is amongst the earliest problems to be shown NP-hard [30, 31][1], although containment in NP had to wait until substantially later [33][2]. NP-hardness has also been established for various restrictions of the problem, for example [26] obtain NP-hardness when Q has rank 1 with one negative eigenvector (in a sense the simplest kind of nonconvex program that can be expressed as a QP). A simple reduction from MAX-CLIQUE to QP [2, 22, 28] yields NP-hardness for QPs that are square-free quadratic forms (diagonal entries of the matrix Q in (1) are zero, and there are no linear terms $c^\top x$); the feasible region is a simplex as opposed to a box.

Regarding local optima, there are hardness results known for computing them, as well as for checking whether a given point is locally optimal. It is shown in [2] that it is hard to find an approximation to a local minimum. The problems of checking whether a given point x is a local optimum, or a strict local optimum, are NP-hard [23, 25] (in particular, when the feasible region is the unit box $[0, 1]^n$ and x is the origin). In the unconstrained case ($m = 0$) [1] show that it is possible in polynomial time to determine whether any version of local optimal solution exists.

There are strong hardness results even for computing approximations to the global optimum of QP. [5] obtained hardness of approximation for QP by reducing a two-prover one-round proof (with polylogarithmic communication) to QP in quasi-polynomial time. From this it follows that assuming NP problems are not solvable in quasi-polynomial time, there is no non-trivial constant factor approximation algorithm for QP. [5, Theorem 1.3] also show that for some constant $\mu \in (0, 1)$, QP is NP-hard to approximate within a factor μ; these hardness results even apply assuming numbers are given in unary. Also in the context of two-prover one-round interactive proof systems, [16] show how the search for an optimal strategy for the provers can be expressed as a QP. They study a relaxed version of the QP that corresponds with an upper bound on the value of the game played by the provers (and is poly-time computable); this leads to a general-purpose heuristic for problems in NP, and in turn a general kind of algorithm for diverse problems in P. [15] (Corollary 4) use this to conclude that (unless P = NP) there is no constant-factor approximation algorithm for QP.

1.1 NP Total Search Problems and the Class CLS

As a solution concept, KKT points have two appealing properties: guaranteed existence (provided the feasible region is bounded), and polynomial-time checkability (we can efficiently verify that a point is KKT). These properties mean that the problem of *computing* one belongs to the complexity class TFNP: total (as opposed to partial) functions that belong to NP. Problems that belong to TFNP are classified by various syntactic subclasses associated with the proof principle underlying the existence guarantee. Here, the relevant classes are PLS [19], PPAD [24], and CLS [12], the latter having been shown to be equal to PPAD ∩ PLS [14].

The complexity class CLS (for "continuous local search") was introduced by [12] in an effort to understand the complexity of certain seemingly-hard search problems that belong to both PPAD and PLS. The problems they list include the search for a KKT point of a given polynomial over a domain given by linear constraints. Such problems are unlikely to be complete for either PPAD or PLS, since such a result would indicate that one of PPAD or PLS contains the other. Recently [14] showed that CLS is equal to the intersection of PPAD and PLS, in the process showing that it is CLS-complete to find KKT solutions of piecewise polynomial functions defined by a certain class of arithmetic circuits. Building on these results, [4] showed that computing a (possibly mixed) Nash equilibrium of a congestion game is CLS-complete and furthermore (of more relevance to the present paper) that local optimisation (in the KKT sense) of degree-5 polynomials is also CLS-complete. Since the CLS-completeness results of [4, 14], other problems in game theory have been shown CLS-complete [13, 32] via comparatively direct reductions. The main result of [14] indicates that CLS-complete problems are unlikely to have polynomial-time algorithms. Moreover, the hardness of CLS can also be based on various cryptographic assumptions such as particular versions of indistinguishability obfuscation [17], soundness of Fiat-Shamir [10], or Learning With Errors [18].

Regarding our main result and its significance, we have noted that quadratic programming is a fundamental problem of general interest. Our main result answers an open question raised in [27] (see problem 3) and reiterated in [2]. The problem Polynomial KKT [12] is a generalization of (1) in which f is allowed to be any polynomial, written down as a sum of monomials. [4] showed that the Polynomial KKT problem is CLS-complete for degree-5 polynomials, which naturally raises the question, pointed out in [14], of whether such a result holds for lower degree. Here we identify the lowest degree for which a hardness result holds, since for degree 1 the problem is linear programming. On the other hand, our result does not hold for some versions of interest, such as taking a standard simplex as the feasible region, e.g., [6]. Another important class of QPs that differs from the one studied here involves optimising quadratic functions over a unit sphere, or an intersection of spheres, e.g., [34, 35, 37].

Our main result is for the computation of *exact* (as opposed to approximate) solutions. Fortunately, any problem instance has rational-valued KKT solutions whose bit complexity is polynomial in the bit complexity of numbers appearing in the problem instance[3].

[1] Indeed, even before NP-completeness, [22] reduce MAX CLIQUE to a version of QP in which the feasible region is a simplex. In Sahni's reduction (from SUBSET SUM) the feasible region is a box.

[2] [33] showed in particular that there exist optimal solutions having polynomial bit complexity.

[3] This does not hold for objective functions of degree 3 or more. The distinction is analogous to the distinction between Nash equilibria of 2-player games versus 3-player games.

If we consider natural notions of approximation, computation of exact solutions is polynomial-time equivalent to the computation of ϵ-approximate solutions for inverse-exponential ϵ. [36] gives an algorithm that computes ϵ-KKT points, whose runtime dependence on ϵ is $O((1/\epsilon)\log(1/\epsilon)\log(\log(1/\epsilon)))$ (there is also polynomial dependence on n, and a factor representing the difference between the maximal and minimal objective values). So we give a negative answer to the question of whether a logarithmic dependence on ϵ is possible. Finally, our hardness result also highlights a contrast with convex optimisation, in which KKT points and global optima coincide, and many algorithms are known that find ϵ-approximate solution in time $O(\log(1/\epsilon))$ [7].

Theorem (Main Result). *It is* CLS*-complete to compute KKT solutions of (1), even when the feasible region consists of the unit box* $[0,1]^n$.

1.2 Technical Overview

Our result is proved in two steps. First, we present a reduction from the problem of computing (some sort of) a KKT point of a type of arithmetic circuit to the problem of computing a KKT point of a QP with box constraints. Namely, we consider *linear arithmetic circuits* that compute piecewise linear functions using a single kind of (fairly general, multi-purpose) gate. In the second step, we show that computing a KKT solution of such a circuit is a CLS-hard problem by reducing from a version of the problem for more general circuits, that is known to be CLS-hard [14]. Together, these two reductions establish the CLS-hardness of computing a KKT point of a QP.

While the first step is certainly the most interesting part of this combined reduction, the second step is surprisingly technical and requires a certain number of new ideas, which are likely to be useful in future works. We now present the main challenges in both parts, as well as the new ideas that were needed to overcome them.

Step 1: Reducing from a circuit to a QP. The first challenge in this part is the following main obstacle.

Challenge 1: We can only use terms of degree at most two. Unsurprisingly, the techniques used in [14] to show CLS-hardness of finding a KKT point of a general arithmetic circuit are of no use here, since we are reducing to an explicit polynomial. Rather, just as in [4], we will just use the result of [14] as a starting point for reductions.

The techniques used in [4] to reduce to a degree-5 polynomial are much more relevant here. Their reduction is highly non-trivial and also relies on some older ideas used in the context of proving PLS-hardness of a version of local-max-SAT [20]. However, the restriction here to degree-2 polynomials makes a big difference and we mostly cannot re-use their ideas.[4] We encounter a fundamental obstacle to the use of *guide variables* (called guide players in [4], since their reduction is presented in terms of a game). Very briefly, the role of these guide variables, which were already used in [20] in a somewhat simpler form, is to be able to "deactivate" some interactions between two (or more) other variables. This deactivation is absolutely crucial for the approach of [4], as it already was in [20]. Given that in any reasonable construction of a quadratic polynomial the interaction between two variables would yield a quadratic term (e.g., $x_i x_j$, or perhaps $(x_i - x_j)^2$), the addition of a guide variable on top of that immediately takes us to up degree 3.

The inability to use guide variables, or any of the other involved machinery from [4], forces us to start from the ground up. As a toy example to illustrate our approach, consider a circuit C that takes two inputs $x_1, x_2 \in [0,1]$ and consists of only two gates. The first gate computes $x_3 := x_1 + x_2$, and then the second gate, which is the output of the circuit, computes $x_4 := -2x_3$. Thus, the circuit C simply computes the function $f : [0,1]^2 \to \mathbb{R}, (x_1, x_2) \mapsto -2(x_1 + x_2)$. This is a linear function and so finding a KKT point over the simple domain $[0,1]^2$ is very easy: the only KKT point (for the minimization problem) is at $(1,1)$. Now let us attempt to simulate this circuit by a quadratic polynomial that implements each gate separately.[5] Consider the polynomial

$$p(x_1, x_2, x_3, x_4) := (x_3 - x_1 - x_2)^2 + (x_4 + 2x_3)^2$$

which consists of one squared term for each gate.[6] Intuitively, minimizing p will force $x_3 = x_1 + x_2$ and $x_4 = -2x_3$. The partial derivative of p with respect to x_4 is

$$\frac{\partial p}{\partial x_4} = 2(x_4 + 2x_3) .$$

At a KKT point this must be zero,[7] so we obtain $x_4 = -2x_3$. Next, we have

$$\frac{\partial p}{\partial x_3} = 2(x_3 - x_1 - x_2) + 4(x_4 + 2x_3) .$$

Setting this to zero, and using $x_4 = -2x_3$, we obtain $x_3 = x_1 + x_2$ as desired. Thus, any KKT point (x_1, x_2, x_3, x_4) of p satisfies $x_4 = f(x_1, x_2)$. In other words, we have correctly simulated the evaluation of the circuit. However, this is not enough. We want any KKT point (x_1, x_2, x_3, x_4) of p to yield a KKT point (x_1, x_2) of f, and this is currently not the case. What is missing is that the QP is not "aware" of the fact that it should attempt to minimize the output of the circuit, namely x_4. An initial attempt to fix this by redefining

$$p(x_1, x_2, x_3, x_4) := (x_3 - x_1 - x_2)^2 + (x_4 + 2x_3)^2 + x_4$$

fails because it introduces big errors in the evaluation of the gates. This can be mitigated by using the idea of exponentially-decreasing

[4] One notable exception to this is the idea of simulating the evaluation of a circuit by constructing an objective function that consists of a sum of terms, one for each gate of the circuit, with gates deeper down in the circuit having smaller weights. This idea of exponentially-decreasing weights, already used in [20] in the context of discrete local optimisation, ensures that gates are correctly simulated and that their output is not biased by other gates that use it as an input.

[5] Obviously, there is a trivial reduction here that just lets the quadratic polynomial be the linear function f itself. However, we are interested in a construction that implements each gate separately, because we will ultimately need to implement (slightly) more general gates. Indeed, a circuit that only consists of linear gates represents a linear function, and it is easy to find KKT points of such functions.

[6] Because we use such squared terms, our polynomial will not be multilinear. In particular, our result has no implications for games, unlike [4].

[7] To simplify the exposition in this part of the overview we think of x_4 (and all other intermediate circuit variables) as being unconstrained. Thus, we can ignore the fact that if $x_4 \in \{0,1\}$, then the KKT condition is not $\frac{\partial p}{\partial x_4} = 0$, but rather $\frac{\partial p}{\partial x_4} \geq 0$ or $\frac{\partial p}{\partial x_4} \leq 0$. Thinking of x_4 as being unconstrained is actually not completely incorrect, since it is possible to pick a constraint $x_4 \in [a,b]$ for sufficiently small a and sufficiently large b such that x_4 never lies on the boundary at a solution. In any case, we will later revert to $[0,1]$ constraints and these will indeed be used in a very crucial way in our construction.

weights from [20] (which was also heavily used in [4]). Indeed, we can instead define

$$p(x_1, x_2, x_3, x_4) := (x_3 - x_1 - x_2)^2 + \delta(x_4 + 2x_3)^2 + \delta^2 x_4$$

where $\delta > 0$ is small. Performing the same analysis as above, yields $x_4 = -2x_3 - \delta/2$, and then $x_3 = x_1 + x_2 + \delta^2$. So the gates are no longer evaluated exactly, but have some additive error. Fortunately, the error can be made arbitrarily small by making δ sufficiently small.

The interesting observation however is that

$$\frac{\partial p}{\partial x_1} = -2(x_3 - x_1 - x_2) = -2\delta^2$$

and similarly $\frac{\partial p}{\partial x_2} = -2\delta^2$. This forces any KKT point (x_1, x_2, x_3, x_4) of p to satisfy $(x_1, x_2) = (1, 1)$, which is indeed the correct KKT point of the original function f! Moreover, notice that $-2\delta^2$ is equal to $\delta^2 \frac{\partial f}{\partial x_1}$, i.e., it is proportional to the partial derivative of the original function f. In other words, this seemingly arbitrary error term actually carries useful information. This is not a coincidence. The errors, starting from the error in the output gate due to the new term $\delta^2 x_4$, propagate backwards in the circuit evaluation, until they reach the input variables x_1 and x_2. In doing so, every traversed gate modifies the error in a very particular way, until, finally, the signal seen by the input variables corresponds to the gradient of the original function f. Indeed, at every gate the error is modified in a way that corresponds to applying the rules for computing the gradient of a circuit using the *backpropagation* technique. This technique, widely used in machine learning, computes the gradient of a function by starting from the output and repeatedly applying the chain rule for differentiation until the inputs are reached. Indeed, the following can be proved by induction over the depth of the circuit:

Lemma (Linear Backpropagation Lemma). *When p is constructed from a depth-m circuit C with linear gates computing a function f, any KKT point of p satisfies*

$$\frac{\partial p}{\partial x_i} = \delta^m \cdot \frac{\partial f}{\partial x_i}$$

for all input variables x_i.

Challenge 2: Circuits with linear gates are easy. The Linear Backpropagation Lemma implies that any KKT point of p must yield a KKT point of the original function f. Unfortunately, this is not enough to prove that our problem is intractable, because computing a KKT point of a linear function is an easy problem. In order to reduce from existing hard functions [4, 14] we would at least need the circuit C to also consist of multiplication gates $x_k := x_i x_j$. But to implement such a gate we would need terms of the form $(x_k - x_i x_j)^2$, which have degree four.

The crucial observation here is that we have not yet used the boundary of the domain in any way to implement gates. The boundary constraints suggest a natural generalization of linear gates. Indeed, if we consider a term such as $(x_3 - x_1 - x_2)^2$ and now – unlike we did before – also constrain $x_3 \in [0, 1]$, then we see that any KKT point of this term must satisfy

$$x_3 = \mathsf{T}(x_1 + x_2)$$

where $\mathsf{T} : \mathbb{R} \to [0, 1]$ denotes truncation to the $[0, 1]$ interval, i.e., $\mathsf{T}(z) = \min\{1, \max\{0, z\}\}$. More generally, we can simulate any gate of the form $x_k := \mathsf{T}(ax_i + bx_j + c)$ by the term $(x_k - ax_i - bx_j - c)^2$. We call such a gate a *truncated linear gate*. No efficient algorithm for computing KKT points of such circuits seems to be known, so there is hope that we might be able to prove intractability.

Unfortunately, before we can start considering proving such an intractability result, there is a more pressing issue: this reduction only works for a single gate. Although we still have correct (approximate) evaluation of the circuit by picking a sufficiently small δ, the errors no longer correctly simulate backpropagation when truncation occurs. In order to restore the behavior that we observed in the setting without truncation, we need to find a way to simulate truncated linear gates that also works with backpropagation.

We modify the simulation as follows. A truncated linear gate $x_k := \mathsf{T}(ax_i + bx_j + c)$ is now simulated by the term

$$(x_k + z^+ - z^- - ax_i - bx_j - c)^2 + 2z^+ z^- + 2z^+(1 - x_k) + 2z^- x_k$$

where z^+ and z^- are new auxiliary variables. Intuitively, z^+ is here to "pick up the slack" between x_k and $ax_i + bx_j + c$, when the latter is strictly larger than 1. The variable z^- has a similar function when $ax_i + bx_j + c < 0$. Note that the derivative of the new term with respect to x_k is the same as the derivative of $(x_k - ax_i - bx_j - c)^2$. So from the point of view of x_k this new term is the same as the old one; in particular, x_k will again (approximately) take the value $\mathsf{T}(ax_i + bx_j + c)$. The difference is in what the variables x_i and x_j see. The derivative of the old term with respect to x_i was just $-2a(x_k - ax_i - bx_j - c)$, so when truncation occurred this derivative would essentially correspond to the truncation gap, whereas we would want it to be 0, which is the correct backpropagation signal (because a small change in x_i would not change x_k if truncation occurs). On the other hand, the derivative of the new term with respect to x_i is $-2a(x_k + z^+ - z^- - ax_i - bx_j - c)$. In this derivative, the variables z^+ and z^- fill the gap between x_k and $ax_i + bx_j + c$ when truncation occurs, and thus we obtain the correct backpropagation signal 0. If truncation does not occur, then $z^+ = z^- = 0$ and the new term behaves just like the old term.

It is thus tempting to try to establish the following lemma.

Desired Lemma (Ideal Backpropagation Lemma). *When p is constructed from a depth-m circuit C with truncated linear gates computing a function f, any KKT point of p satisfies*

$$\left(\frac{\partial p}{\partial x_1}(x_1, x_2), \frac{\partial p}{\partial x_2}(x_1, x_2)\right) \in \delta^m \cdot \partial f(x_1, x_2)$$

where x_1 and x_2 are the input variables, and ∂f is the generalized gradient[8] of the (almost everywhere differentiable) function f.

Challenge 3: The Ideal Backpropagation Lemma does not hold. Unfortunately, this lemma fails to hold. The reason for this is quite fundamental: backpropagation does not really work for such circuits. Consider the following simple example: the circuit C has a single input x_1, and outputs the value $\mathsf{T}[2x_1]/2 + \mathsf{T}[x_1 - 1/2]$. This can easily be implemented by using three truncated linear gates. Note that the circuit computes the (linear!) function $[0, 1] \to$

[8] At a point where f is differentiable, the generalized gradient is the singleton set consisting of the gradient; where f is not differentiable, it is the set of convex combinations of well-defined gradients close to that point.

$[0,1], x_1 \mapsto x_1$, which has derivative 1 everywhere. Now consider performing backpropagation when the input is $x_1 = 1/2$. Since this is the threshold for both truncations, the value 0 is a valid derivative for both of those gates. As a result, the gradient computed by backpropagation could theoretically output 0. In Section 3 we provide a slightly more involved example (Example 3.3) where this indeed happens in our QP construction: the correct gradient value is $1/2$ everywhere, but at $x_1 \approx 1/2$ we have $\frac{\partial p}{\partial x_1}(x_1) = 0$. In particular, our construction would incorrectly output $x_1 \approx 1/2$ as a KKT point. This shows that the Ideal Backpropagation Lemma cannot hold, even in some approximate version where we would also consider points in the vicinity of (x_1, x_2).

However, the example suggests that a weaker statement might hold. Indeed, the issue occurs because both truncations have a threshold at $1/2$. If we were to slightly perturb these thresholds, then we would indeed see a derivative of 0 appear. In other words, it seems reasonable to think that the backpropagation that occurs computes some convex combination of gradients of various perturbed versions of the circuit C. To be more precise, in a perturbed version of C, every gate $x_k := \mathsf{T}(ax_i + bx_j + c)$ is replaced by a gate $x_k := \mathsf{T}(ax_i + bx_j + c + \pi_k)$ for some $\pi_k \in \mathbb{R}$. By picking δ sufficiently small, we can ensure that all π_k are as small as required. Indeed, we can prove the following result.

Lemma (Backpropagation Lemma (informal)). *When p is constructed from a depth-m circuit C with truncated linear gates, any KKT point of p satisfies*

$$\left(\frac{\partial p}{\partial x_1}(x_1,x_2), \frac{\partial p}{\partial x_2}(x_1,x_2)\right) \in \delta^m \cdot \operatorname{conv}\Big\{\nabla \tilde{f}(x_1,x_2) : \tilde{f} \text{ computed by small perturbation of } C\Big\}$$

where x_1 and x_2 are the input variables.

This leads us to define the new notion of a generalized gradient *of a linear arithmetic circuit* to capture this behavior. See the subsequent preliminaries section for more details on this.

With the Backpropagation Lemma in hand, we can now leave quadratic polynomials behind us and focus on circuits with truncated linear gates. We will also refer to these by the simpler name *linear arithmetic circuits* (which is usually used to refer to circuits with $+, -, c, \times c, \min, \max$ gates [14]), since our circuits can easily simulate such circuits and vice-versa. In the next step, we construct a class of such circuits that is robust to perturbations, and for which it is CLS-hard to find a KKT point (with respect to the new definition of generalized gradient).

Step 2(a): Designing a robust function: the mesa construction. In order to show CLS-hardness of this problem, we have to reduce from an existing CLS-hard problem. We reduce from the problem of computing an approximate KKT point of a smooth function defined on the two-dimensional grid $[0,1]^2$, which is known to be CLS-complete when the function is represented by an arithmetic circuit with more general gates [14]. Indeed, being able to work on a two-dimensional domain (as opposed to a high-dimensional one if we used [4] instead) allows us to avoid having to unnecessarily complicate the construction.

At a high level, given such a smooth function defined on $[0,1]^2$, we would like to construct a piecewise linear function that has (approximately) the same KKT solutions as the original function. Importantly, our piecewise linear function must be represented by a linear arithmetic circuit. This is already challenging, even if we ignore the perturbations.

Challenge 1: Existing linear circuit constructions give no guarantees about the gradient. Interpolations of continuous functions by linear arithmetic circuits have been given in prior work [9, 14, 29] and indeed these have been pivotal in proving important results in this field. However, these constructions only aim to obtain a piecewise linear function that closely approximates the original function in terms of *function value*. The usage of the *averaging trick* [11], which is common to all of these, means that the (generalized) gradient of the piecewise linear function can be wildly different from the original gradient and introduce spurious KKT solutions.

We are thus forced to move away from this type of construction. Putting circuits aside for a bit, there is a standard interpolation by a piecewise linear function that does (approximately) maintain the gradient. Simply pick a sufficiently fine standard triangulation of $[0,1]^2$, define the value of the function at the vertices of the triangulation to agree with the original function, and interpolate linearly within each triangle. This interpolation would be sufficient for our purposes, because it is not hard to show that any KKT point of the new function must correspond to an approximate KKT point of the original function.

The "catch" is that we would have to construct a linear arithmetic circuit that represents this piecewise linear interpolation. Existing techniques, which all use the averaging trick, are unable to achieve this. However, it turns out that using some new ideas (most of which we end up using in our final construction and which are highlighted below) it is in fact possible to construct a linear arithmetic circuit that *exactly* computes this piecewise linear interpolation. At this point, if the Ideal Backpropagation Lemma stated earlier was true, we would be done. Unfortunately, this is not the case, and we also have to argue about what happens to the circuit when gates are slightly perturbed.

Challenge 2: The standard piecewise linear interpolation is not robust to perturbations. Unfortunately, this circuit is not robust to perturbations, i.e., perturbed versions of the circuit would introduce new solutions that did not appear in the original function. In order to illustrate the issues that can occur, as well as explain how they can be overcome, it is useful to take a step back and think about what would happen if the domain was one-dimensional, instead of two-dimensional.

In the one-dimensional case, with domain $[0,1]$, the standard interpolation is indeed very simple. Given some sufficiently fine discretization of $[0,1]$, we simply interpolate linearly between adjacent points. Implementing this using a linear arithmetic circuit is still non-trivial (because the discretization is exponentially fine), but it is possible using the new ideas hinted at above. Without going into too much detail, this piecewise linear function is constructed by taking the maximum of an exponential[9] number of simple functions. Every simple function implements a single linear segment

[9] Of course this would not be efficient, so the actual construction has to do something more clever. Nevertheless, this (incomplete) description is sufficient for this part of the technical overview.

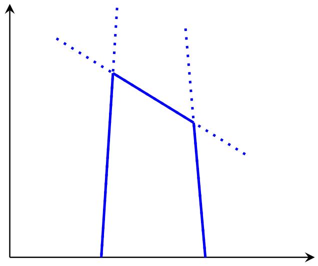

Figure 1: Creating a one-dimensional mesa as the minimum of three lines.

of the interpolation and then very quickly decreases in value as soon as we leave the segment. Each of the simple functions can be constructed by taking the minimum of the three linear functions that they each consist of, see Figure 1. Taking the maximum of all these simple functions then indeed correctly yields the desired interpolation.

Now, let us see what happens when we allow small perturbations at gates of the circuit. Of course, it is hard to think about all the ways in which perturbations can interfere with a construction (especially since we have not given many details here), but a good starting point, and in a certain sense, the absolute minimum requirement is: if we slightly perturb each of the linear functions that are used to construct each of the simple functions by some small additive term, no new solutions should appear.

Unfortunately, the construction already fails this simple test, as can be seen in the example in Figure 2. Furthermore, note that the issue appears as soon as we allow non-zero perturbations; requiring the perturbations to be very small does not help. However, this example suggests a somewhat different approach: instead of always trying to faithfully approximate the gradient of the original function, we can relax this requirement as long as we do not introduce any new KKT points. Indeed, we can avoid the issue by using the following "halved-gradient" trick: for each linear segment, halve its slope while keeping the same value at the middle of the segment. As long as the perturbations are kept sufficiently small, this simple trick ensures that linear pieces with "bad" gradients are no longer visible, i.e., they disappear when we take the maximum over all simple functions. See Figure 2 for an example. The only case where such a "bad" piece might appear is if the slope of the linear segment is very flat. But in that case, the original function must have an approximate KKT point there.

This approach indeed works for the one-dimensional setting. It turns out that the easiest way to generalize this to two dimensions is not to try to apply this to a triangulation of the unit square, but rather to a grid over the unit square. For any point on the grid, we construct a corresponding square segment with a gradient that is half the gradient of the original function at that point. When we leave that square, the function value decreases very quickly. We call this simple function a mesa due to its shape which is reminiscent of a flat-topped hill with steep sides, see Figure 3. The final function is then obtained by taking the maximum of (an exponential number of such) mesa functions, one at each grid point.

We show that this mesa construction does not introduce any new KKT points, except in the vicinity of approximate KKT points of the original function. Importantly, this continues to hold even if we add an arbitrary, but small, perturbation to each linear piece of each mesa.

In the last part of this technical overview, we give some details about how the mesa construction can be implemented by using only the gates available in a linear arithmetic circuit (namely, truncated linear gates, as well as other gates that can be simulated by them, such as max and min). Furthermore, we need to make sure that the perturbations, which appear in any gate, can only impact the construction in the way described above (i.e., perturbing each linear piece of each mesa), and not in any other way.

Step 2(b): Robustly implementing the mesa construction with a linear circuit. From Step 2(a) we get a grid of points G covering $[0,1]^2$, and Boolean circuits that, for each point $y \in G$ define a mesa centered at y. If $m(x, y)$ is the height of the mesa centered at y at the point x, then we must implement a linear circuit that computes

$$f(x) = \max_{y \in G} \; m(x, y).$$

Since G contains exponentially many points, it is clearly infeasible to compute f directly. Instead we first perform a bit extraction on x to obtain binary encodings of a small set $S \subseteq G$ of nearby grid points, and we then evaluate $f(x) = \max_{y \in S} m(x, y)$ instead. This works because each mesa can only achieve the maximum used in f in a small radius around its center, so we can disregard the mesas that are far from x when computing f.

While the technique of extracting bits from x to succinctly compute some function $f(x)$ has been used before, we must overcome several challenges to make this work for our setting.

Challenge 1: Dealing with bit extraction failures. The bit extraction process requires us to implement an inherently discontinuous function, and since linear circuits can compute only continuous functions, we are forced to rely on bit extractors that can fail for a small subset of the inputs. This is a well-known problem, and prior work has addressed it through the use of the "averaging trick", in which one extracts bits for x and also a large number of points that are close to x. By arranging the process such that only a small number of the bit extractions can fail, then the results over all of the points can be averaged, giving us a function $\widetilde{f}$ with $|\widetilde{f}(x) - f(x)| \leq \varepsilon$ for some small ε.

For prior work, which was usually interested only in ensuring that $\widetilde{f}(x)$ was far from zero whenever $f(x)$ was also far from zero, this approach was good enough. However for our purposes, any deviation in the desired output, no matter how small, may fatally undermine the construction by introducing new gradients, which could create new solutions.

To overcome this, we apply averaging in a new way that ensures that any errors arising from bit extraction failures do not make their way into the output. Specifically, we divide the points in the grid into four sets $S_1, S_2, S_3, S_4 \subseteq G$ where each set contains points from a grid of double the width of the original. The four sets are shown in four different colours on the left side of Figure 4. For each set S_i we build a function $g_i(x) = \max_{y \in S_i} m(x, y)$ which takes the

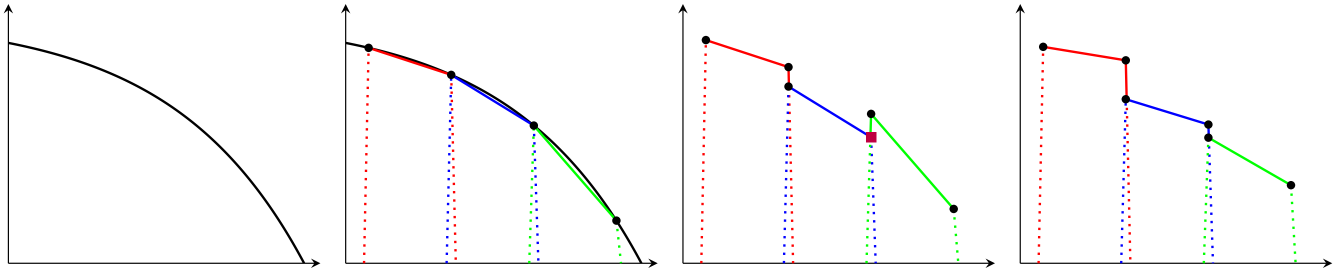

Figure 2: Left to right: smooth function we want to interpolate (with no stationary points); standard interpolation as the maximum of adjacent mesas without perturbations (still no stationary points); additive perturbations introduce an unwanted stationary point (indicated as a brown square); this unwanted stationary point does not arise when the halved-gradient trick is applied.

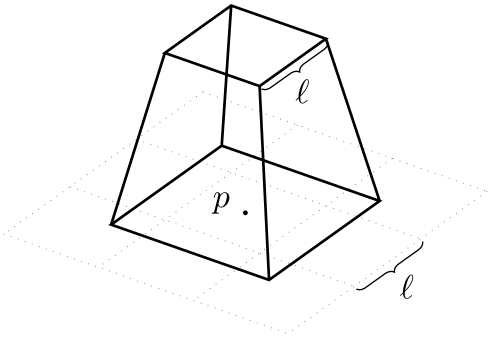

Figure 3: An illustration of a two-dimensional mesa function centered at point p.

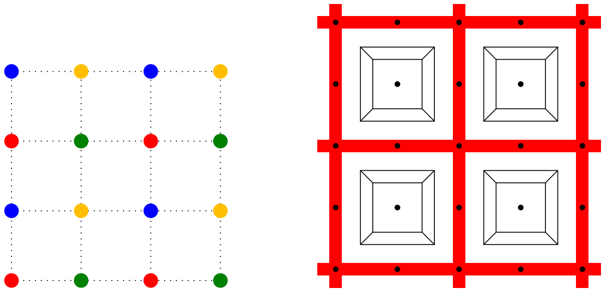

Figure 4: Left: the division of the grid into four sets. Right: one of the four g_i functions.

maximum only over the mesas whose centers are in S_i. We then set $f(x) = \max(g_1(x), g_2(x), g_3(x), g_4(x))$.

The function g_i is shown on the right of Figure 4. The red lines in the figure show the locations at which a bit extraction might fail. Importantly, these regions occur only in places at which $m(x, y) \leq 0$ for all mesa centers $y \in S_i$. We ensure that at most two bit extractions can fail, and that when they do fail they result in an output that is at most 1. So by averaging over 12 points near x we can ensure that $g_i(x) \leq 1/6$ whenever a bit extraction failure occurs.

However, we also ensure that $f(y) \geq 1/3$ for all points y. Therefore, the function g_j that is responsible for the mesa that defines $f(x)$ will satisfy $g_j(x) \geq 1/3$. Since f is defined to be the maximum of the g functions, this means that any errors arising from failed bit decodings in g_i will be masked completely by another function g_j that has correctly decoded its input. Ultimately, this means that any spurious gradient arising from a failed bit extraction is well below the value of $f(x)$, and so these gradients can never make their way into the output.

Challenge 2: Multiplying two variables. Now we are given Boolean circuits that define the mesas that we must output. This means that a Boolean circuit will tell us to output an affine function at $x = (x_1, x_2)$ with gradient $g = (g_1, g_2)$ and additive value a, in which case we will need to output

$$x_1 \cdot g_1 + x_2 \cdot g_2 + a.$$

This is problematic however, because x and g are both variables, and a linear circuit does not allow us to multiply two variables together.

We circumvent this by showing that it is possible to compute $x \cdot y$ when x is a continuous variable and y is a variable encoded in binary. So we represent gradients in binary in our circuit, and this allows us to precisely control the gradients that we output.

Challenge 3: Perturbations. It is not enough to create a linear circuit that implements f, because our linear circuit must also be robust to perturbations. These perturbations will slightly alter the values that are outputted by min, max, and truncation operations, and we need to ensure that they do not alter the gradients of the mesas that appear in the output of the function.

We are able to show that evaluating a Boolean circuit and decoding a binary value can be carried out exactly with no errors even in the presence of perturbations, while introducing perturbations in the bit extraction process only slightly increases the region in which the bit extraction will fail. Then we show that each mesa can be computed with gradients that are exactly correct, but where each of the five pieces might be additively perturbed by a small amount. Finally, we show that the step of splitting the points into the g_i functions and then maximizing over them to produce f only adds an additional small perturbation, while not altering the gradients of any of the mesas.

2 PRELIMINARIES

All numbers appearing as inputs in our problem are assumed to be rational. A rational number x is represented by its numerator and denominator (in binary) of the irreducible fraction for x. We let $\text{size}(x)$ denote the number of bits needed to represent x in this way. We also extend this notation in the natural way to the case where x is a vector with rational entries.

Definition 2.1. The QUADRATIC-KKT problem is defined as follows. We are given a degree-2 polynomial p over n variables, and the goal is to compute a KKT point of the following optimization problem:

$$\begin{aligned} \min \quad & p(x) \\ \text{s.t.} \quad & 0 \leq x_i \leq 1 \quad \forall i \in [n] \end{aligned} \tag{2}$$

A point $x \in [0,1]^n$ is a KKT point of (2) if, for all $i \in [n]$,

- if $x_i > 0$, then $\frac{\partial p}{\partial x_i}(x) \leq 0$, and
- if $x_i < 1$, then $\frac{\partial p}{\partial x_i}(x) \geq 0$.

The QUADRATIC-KKT problem lies in the class CLS, even for more general domains,[10] namely any non-empty compact domain given by linear inequalities [14]. This is because the problem can be solved (inefficiently) by gradient descent. The problem is guaranteed to always admit at least one rational solution with polynomially bounded bit complexity; see the full version for a proof sketch of this fact. Our main result is the following theorem.

THEOREM 2.2. *The* QUADRATIC-KKT *problem is* CLS*-complete.*

The problem remains CLS-complete even if we only ask for an ε-KKT point, where $\varepsilon > 0$ is allowed to be exponentially small (i.e., is given in binary). Indeed, by standard arguments, finding an exact solution reduces to finding an approximate solution; see the full version. For the definition of ε-KKT points, we just replace "≤ 0" and "≥ 0" by "$\leq \varepsilon$" and "$\geq -\varepsilon$" (respectively) in the definition of QUADRATIC-KKT.

We prove Theorem 2.2 by reducing from the 2D-LINEAR-KKT problem, which we introduce below and for which we prove CLS-hardness. The reduction from 2D-LINEAR-KKT to QUADRATIC-KKT is presented in Section 3.

KKT Points of Linear Arithmetic Circuits. A linear arithmetic circuit is a circuit that consists of gates implementing piecewise linear operations. As a result, the function represented by a linear circuit is a piecewise linear function. Such circuits can be evaluated in polynomial time [14].

In this paper, we consider linear arithmetic circuits that consist of a single[11] type of gate: truncated linear gates. A truncated linear gate is defined by rational parameters $a, b, c \in \mathbb{Q}$. The gate takes as input two variables x_i, x_j of the circuit and outputs $x_k := \mathsf{T}(ax_i + bx_j + c)$, where $\mathsf{T} : \mathbb{R} \to [0,1]$ denotes truncation (i.e., projection) to the $[0,1]$ interval.

Generalized gradients. Let C be a linear arithmetic circuit with n inputs and one output. We let $f : \mathbb{R}^n \to \mathbb{R}$ denote the function computed by the circuit C. This is a piecewise linear function that is almost everywhere differentiable. The generalized gradient of f at point y can be defined as

$$\partial f(y) := \text{conv}\Big\{ \lim_{t\to\infty} \nabla f(y_t) : (y_t)_t \text{ converging to } y \text{ such that } \\ f \text{ differentiable at } y_t \text{ and } \nabla f(y_t) \text{ also converges} \Big\}.$$

For our purposes we have to introduce a new more general notion of generalized gradient *of a circuit*. Let C be a linear arithmetic circuit consisting of m truncated linear gates. For any $\pi = (\pi_i)_{i\in[m]} \in \mathbb{R}^m$, we let C^π denote the circuit C perturbed by π, namely, for each $i \in [m]$ the ith gate $x_i := \mathsf{T}(ax_j + bx_k + c)$ is replaced by $x_i := \mathsf{T}(ax_j + bx_k + c + \pi_i)$. We let $f^\pi : \mathbb{R}^n \to \mathbb{R}$ denote the function represented by the perturbed circuit C^π.

For any $\delta > 0$ and any such circuit C, the δ-generalized circuit gradient of C at point $y \in \mathbb{R}^n$ is defined as

$$\widetilde{\partial}_\delta C(y) := \text{conv}\big\{ \nabla f^\pi(y) : \pi \in [-\delta, \delta]^m \text{ such that } \\ f^\pi \text{ is differentiable at } y \big\}.$$

It can be shown that $\partial f(y) \subseteq \widetilde{\partial}_\delta C(y)$ for all $\delta > 0$. Although it is tempting to think that $\widetilde{\partial}_\delta C(y) \to \partial f(y)$ as $\delta \to 0$, this is not the case. Indeed, Example 3.3 together with the Backpropagation Lemma (Lemma 3.2) provide a counter-example.

We can now define the intermediate computational problem which will act as a bridge between existing CLS-hard problems and QUADRATIC-KKT.

Definition 2.3. The 2D-LINEAR-KKT problem is defined as follows. We are given $\varepsilon, \delta > 0$ and a linear arithmetic circuit C with two inputs and one output, and consisting only of truncated linear gates. The goal is to find a point $y \in [0,1]^2$ that satisfies the ε-KKT conditions with respect to the δ-generalized circuit gradient of C, i.e., such that there exists $u \in \widetilde{\partial}_\delta C(y)$ satisfying

- if $y_i > 0$, then $u_i \leq \varepsilon$
- if $y_i < 1$, then $u_i \geq -\varepsilon$

for $i = 1, 2$.

We note that it is not clear whether 2D-LINEAR-KKT lies in TFNP, because it is not clear whether we can efficiently check if some given y is a solution. Nevertheless, we establish that this problem is CLS-hard, which is all we require from this intermediate problem.

PROPOSITION 2.4. *The* 2D-LINEAR-KKT *problem is* CLS*-hard.*

Due to space constraints, the proof of this proposition, which is quite involved as explained in the introduction, is omitted. It can be found in the full version of the paper. The rest of this paper focuses on the reduction from 2D-LINEAR-KKT to our problem of interest.

3 REDUCTION FROM 2D-LINEAR-KKT TO QUADRATIC-KKT

The main result of this section is the following.

PROPOSITION 3.1. *There is a polynomial-time reduction from* 2D-LINEAR-KKT *to* QUADRATIC-KKT.

[10] We omit the definition of a KKT point for more general domains; see, e.g., [14].

[11] Of course, various other types of gates can be simulated using truncated linear gates and we will use this later in the paper.

The remainder of this section proves this result. We begin with the detailed construction of the quadratic polynomial and a statement of the Backpropagation Lemma (Lemma 3.2), and explain why it implies Proposition 3.1. Then, we prove some simple properties of the construction, before the technical culmination of this section, namely the proof of the Backpropagation Lemma.

3.1 Construction and Backpropagation Lemma

Let C be a linear arithmetic circuit that has two inputs and one output, and that consists only of truncated linear gates.

Let n denote the number of variables in the circuit C. We use x_i to denote the ith variable in the circuit, and assume that $x_1, \dots, x_n$ are ordered such that the gate computing x_i uses inputs $x_{\ell(i)}, x_{r(i)}$ with $\ell(i) < i$ and $r(i) < i$. In particular, x_1 and x_2 are the input variables, and x_n is the output variable. For every $i \in [n] \setminus \{1, 2\}$, the ith gate of C is the gate computing x_i. It will be more convenient to write the ith gate's function $x_i = \mathsf{T}(a_i x_{\ell(i)} + b_i x_{r(i)} + c_i)$ as $x_i = \mathsf{T}(\sum_{j=1}^{i-1} a_{ij} x_j + c_i)$, where

$$a_{ij} = \begin{cases} a_i & \text{if } j = \ell(i) \\ b_i & \text{if } j = r(i) \\ 0 & \text{otherwise} \end{cases}$$

Let also $K \geq 1$ be such that $K \geq \max_{i \in [n] \setminus \{1,2\}} (\sum_{j=1}^{i-1} |a_{ij}| + |c_i|)$.

We now construct a polynomial p on $n + 2(n-2) = 3n - 4$ variables. In more detail, the polynomial will have the following variables:

- For each $i \in [n]$, a variable y_i, corresponding to each variable x_i of C.
- For each $i \in [n] \setminus \{1, 2\}$, two auxiliary variables z_i^+ and z_i^- to help with the implementation of the ith gate, which computes x_i.

For each gate $i \in [n] \setminus \{1, 2\}$ we construct a polynomial q_i on variables $y = (y_1, \dots, y_n)$, $z = (z_3^+, z_3^-, \dots, z_n^+, z_n^-)$

$$q_i(y, z) := \left(y_i + K z_i^+ - K z_i^- - \sum_{j=1}^{i-1} a_{ij} y_j - c_i \right)^2 + 2K^2 z_i^+ z_i^- + 2K z_i^+ (1 - y_i) + 2K z_i^- y_i.$$

For a given $\delta \in (0, 1)$, the final polynomial p is then constructed as follows

$$p(y, z) := \delta^{n+1} y_n + \sum_{i=3}^{n} \delta^i q_i(y, z).$$

QUADRATIC-KKT instance. The instance of QUADRATIC-KKT we consider is thus

$$\begin{aligned} \min \quad & p(y, z) \\ \text{s.t.} \quad & (y, z) \in [0, 1]^{3n-4} \end{aligned} \tag{3}$$

We are now ready to state the main technical lemma of this section.

Lemma 3.2 (Backpropagation Lemma). *Let (y, z) be a KKT point of the constructed QP (3), for some $\delta \in (0, 1/16K^2)$. Then we have*

$$\frac{1}{\delta^{n+1}} \cdot \left(\frac{\partial p}{\partial y_1}(y, z), \frac{\partial p}{\partial y_2}(y, z) \right) \in \widetilde{\partial}_{\delta'} C(y_1, y_2)$$

where $\delta' = 8K^2 \delta$.

Let us see how Proposition 3.1 follows from this lemma. Let ε', δ', and C be the inputs to a 2D-LINEAR-KKT instance. We construct the polynomial p described above with $\delta := \min\{\delta'/8K^2, 1/32K^2\}$. Clearly, this can be done in polynomial time. Now, consider any KKT point (y, z) of the resulting QP (3). We claim that (y_1, y_2) must be a solution to the original 2D-LINEAR-KKT instance. Indeed, let

$$u := \frac{1}{\delta^{n+1}} \cdot \left(\frac{\partial p}{\partial y_1}(y, z), \frac{\partial p}{\partial y_2}(y, z) \right).$$

By the Backpropagation Lemma, we have that $u \in \widetilde{\partial}_{\delta'} C(y_1, y_2)$. Furthermore, since (y, z) is a KKT point of (3), we in particular have for $i = 1, 2$

- if $y_i > 0$, then $\frac{\partial p}{\partial y_i}(y, z) \leq 0$, and thus $u_i \leq 0$
- if $y_i < 1$, then $\frac{\partial p}{\partial y_i}(y, z) \geq 0$, and thus $u_i \geq 0$.

In other words, (y_1, y_2) satisfies the KKT conditions (and thus, in particular, the ε'-KKT conditions) with respect to the δ'-generalized circuit gradient of C.

Before proceeding with the proof of the Backpropagation Lemma, we present an example showing that a stronger version of the lemma – where we ask for the generalized gradient of f at (y_1, y_2) (or even of some point in the vicinity) to be zero – fails.

Example 3.3. Consider the circuit C that has one single input x_1 and computes $x_2 := \mathsf{T}(2x_1)$, $x_3 := \mathsf{T}(x_1 - 1/2)$, and outputs $x_4 := \mathsf{T}(x_2/2 + x_3 - x_1/2)$. It is easy to see that this circuit computes the function $f : [0, 1] \to [0, 1], x_1 \mapsto x_1/2$. Thus, the only KKT point of f is at $x_1 = 0$. However, it can be checked that if we construct the polynomial p as described above from C, then, for any sufficiently small $\delta > 0$, the QP (3) will have a KKT point at $y_1 = 1/2 + \delta^2/4$ (and where we have $y_2 = 1$, $y_3 = 0$, and $y_4 = 1/4 - \delta^2/8 - \delta/2$). In particular, this means that the backpropagation computes gradient 0 at that point, even though the actual gradient of f is always $1/2$. As a result, no general backpropagation result can be proved for this kind of circuit without taking into account perturbed versions of the circuit.

3.2 Properties of KKT Points

In this section we prove some simple properties that are satisfied by any KKT point of the QUADRATIC-KKT instance (3). Recall that a point $(y, z) \in [0, 1]^{3n-4}$ is a KKT point of (3) if, for all $i \in [n]$,

- if $y_i > 0$, then $\frac{\partial p}{\partial y_i}(y, z) \leq 0$, and
- if $y_i < 1$, then $\frac{\partial p}{\partial y_i}(y, z) \geq 0$,

and similarly for the other variables z_i^+ and z_i^- for all $i \in [n] \setminus \{1, 2\}$.

Truncation. The following lemma 3.4 states that, at any KKT point, the auxiliary variables z enforce truncation, in a certain sense.

Lemma 3.4. *Let (y, z) be a KKT point of (3). Then for all $i \in [n] \setminus \{1, 2\}$*

$$\mathsf{T}\left(\sum_{j=1}^{i-1} a_{ij} y_j + c_i \right) = \sum_{j=1}^{i-1} a_{ij} y_j + c_i - K z_i^+ + K z_i^-.$$

Proof. We show the following stronger fact, namely that

$$K z_i^+ = \max \left\{ 0, \left(\sum_{j=1}^{i-1} a_{ij} y_j + c_i \right) - \mathsf{T}\left(\sum_{j=1}^{i-1} a_{ij} y_j + c_i \right) \right\} \tag{4}$$

and

$$Kz_i^- = \max\left\{0, \mathsf{T}\left(\sum_{j=1}^{i-1} a_{ij}y_j + c_i\right) - \left(\sum_{j=1}^{i-1} a_{ij}y_j + c_i\right)\right\}. \tag{5}$$

In order to prove (4), note that the variable z_i^+ only appears in q_i, and thus $\frac{\partial p}{\partial z_i^+} = \delta^i \frac{\partial q_i}{\partial z_i^+}$ and

$$\begin{aligned}\frac{\partial q_i}{\partial z_i^+}(y,z) &= 2K\left(y_i + Kz_i^+ - Kz_i^- - \sum_{j=1}^{i-1} a_{ij}y_j - c_i\right)\\ &\quad + 2K^2 z_i^- + 2K(1-y_i)\\ &= 2K\left(1 + Kz_i^+ - \sum_{j=1}^{i-1} a_{ij}y_j - c_i\right).\end{aligned}$$

We now consider two cases. If $\sum_{j=1}^{i-1} a_{ij}y_j + c_i \leq \mathsf{T}(\sum_{j=1}^{i-1} a_{ij}y_j + c_i)$, then it must be that $\sum_{j=1}^{i-1} a_{ij}y_j + c_i \leq 1$. As a result, $\frac{\partial p}{\partial z_i^+}(y,z) = \delta^i \frac{\partial q_i}{\partial z_i^+}(y,z) \geq \delta^i \cdot 2K^2 z_i^+$. By the KKT conditions it follows that $z_i^+ = 0$. Indeed, if $z_i^+ > 0$, then we would have $\frac{\partial p}{\partial z_i^+}(y,z) > 0$, which contradicts the KKT conditions.

If, on the other hand, $\sum_{j=1}^{i-1} a_{ij}y_j + c_i > \mathsf{T}(\sum_{j=1}^{i-1} a_{ij}y_j + c_i)$, then it must be that $\sum_{j=1}^{i-1} a_{ij}y_j + c_i > 1$. As a result, $\frac{\partial p}{\partial z_i^+}(y,z) = \delta^i \frac{\partial q_i}{\partial z_i^+}(y,z) < \delta^i \cdot 2K^2 z_i^+$. In particular, we cannot have $z_i^+ = 0$, since that would imply $\frac{\partial p}{\partial z_i^+}(y,z) < 0$, which is not allowed by the KKT conditions at $z_i^+ = 0$. We also cannot have $z_i^+ = 1$. Indeed, by the KKT conditions, that would imply that $\frac{\partial p}{\partial z_i^+}(y,z) \leq 0$, which translates to

$$\delta^i \cdot 2K\left(1 + K - \sum_{j=1}^{i-1} a_{ij}y_j - c_i\right) \leq 0$$

which is impossible, since $K \geq 1$ was chosen such that $K \geq \sum_{j=1}^{i-1} |a_{ij}| + |c_i| \geq \sum_{j=1}^{i-1} a_{ij}y_j + c_i$. As a result, we must have $z_i^+ \in (0,1)$, which implies that the KKT condition is $\frac{\partial p}{\partial z_i^+}(y,z) = 0$. This yields

$$Kz_i^+ = \sum_{j=1}^{i-1} a_{ij}y_j + c_i - 1 = \sum_{j=1}^{i-1} a_{ij}y_j + c_i - \mathsf{T}\left(\sum_{j=1}^{i-1} a_{ij}y_j + c_i\right)$$

as desired. We have thus shown that (4) always holds at a KKT point.

In order to prove (5), we again note that $\frac{\partial p}{\partial z_i^-} = \delta^i \frac{\partial q_i}{\partial z_i^-}$ and

$$\begin{aligned}\frac{\partial q_i}{\partial z_i^-}(y,z) &= -2K\left(y_i + Kz_i^+ - Kz_i^- - \sum_{j=1}^{i-1} a_{ij}y_j - c_i\right)\\ &\quad + 2K^2 z_i^+ + 2Ky_i\\ &= 2K\left(Kz_i^- + \sum_{j=1}^{i-1} a_{ij}y_j + c_i\right).\end{aligned}$$

and then perform a similar case analysis. □

Approximate evaluation. The next lemma 3.5 states that the gates of the circuit are correctly simulated at a KKT point of (3), up to some small additive error depending on the parameter δ. The lemma also gives a precise expression for the value of each variable y_i at a KKT point, which will be useful for the next section. In order to state this precise expression, we first have to introduce some additional notation. We define, for any $i \in [n] \setminus \{1\}$,

$$p_i(y,z) := \delta^{n+1} y_n + \sum_{\ell=i+1}^{n} \delta^\ell q_\ell(y,z).$$

In particular, $p_2 = p$, and $p_n(y,z) = \delta^{n+1} y_n$. We are now ready to state the lemma.

Lemma 3.5. *Let (y,z) be a KKT point of (3). Then for any $i \in [n] \setminus \{1,2\}$*

$$\begin{aligned}y_i &= \mathsf{T}\left(\sum_{j=1}^{i-1} a_{ij}y_j + c_i - \frac{1}{2\delta^i} \cdot \frac{\partial p_i}{\partial y_i}(y,z)\right)\\ &= \mathsf{T}\left(\sum_{j=1}^{i-1} a_{ij}y_j + c_i\right) \pm (2K\delta)^{n+1-i}.\end{aligned}$$

Proof. Due to space constraints, we only include the proof of the first equality; the second equality is proved in the full version. Note that

$$\begin{aligned}\frac{\partial p}{\partial y_i}(y,z) &= \delta^i \frac{\partial q_i}{\partial y_i}(y,z) + \frac{\partial p_i}{\partial y_i}(y,z)\\ &= 2\delta^i\left(\left(y_i + Kz_i^+ - Kz_i^- - \sum_{j=1}^{i-1} a_{ij}y_j - c_i\right) - Kz_i^+ + Kz_i^-\right)\\ &\quad + \frac{\partial p_i}{\partial y_i}(y,z)\\ &= 2\delta^i\left(y_i - \sum_{j=1}^{i-1} a_{ij}y_j - c_i\right) + \frac{\partial p_i}{\partial y_i}(y,z).\end{aligned}$$

Hence if $y_i > \sum_{j=1}^{i-1} a_{ij}y_j - c_i - \frac{1}{2\delta^i}\frac{\partial p_i}{\partial y_i}(y,z)$, then $\frac{\partial p}{\partial y_i}(y,z) > 0$, and by the KKT conditions we must have $y_i = 0$. If, on the other hand, $y_i < \sum_{j=1}^{i-1} a_{ij}y_j - c_i - \frac{1}{2\delta^i}\frac{\partial p_i}{\partial y_i}(y,z)$, then $\frac{\partial p}{\partial y_i}(y,z) < 0$, and by the KKT conditions we must have $y_i = 1$. Thus, in all cases we have $y_i = \mathsf{T}(\sum_{j=1}^{i-1} a_{ij}y_j - c_i - \frac{1}{2\delta^i}\frac{\partial p_i}{\partial y_i}(y,z))$. □

3.3 Proof of the Backpropagation Lemma

In this section we prove the Backpropagation Lemma. We begin by recalling some notation, as well as introducing some new notation. We let $f : \mathbb{R}^2 \to \mathbb{R}$ denote the function represented by the circuit C. For any $\pi = (\pi_i)_{i \in [n] \setminus \{1,2\}} \in \mathbb{R}^{n-2}$, we let C^π denote the circuit C perturbed by π, namely, for each $i \in [n] \setminus \{1,2\}$ the ith gate $x_i := \mathsf{T}(\sum_{j=1}^{i-1} a_{ij}x_j + c_i)$ is replaced by $x_i := \mathsf{T}(\sum_{j=1}^{i-1} a_{ij}x_j + c_i + \pi_i)$. We let $f^\pi : \mathbb{R}^2 \to \mathbb{R}$ denote the function represented by the perturbed circuit C^π. For any sign vector $s = (s_i)_{i \in [n] \setminus \{1,2\}} \in \{+1,-1\}^{n-2}$, we let $s \cdot \pi \in \mathbb{R}^{n-2}$ denote the coordinate-wise product of vector s with vector π, i.e., $[s \cdot \pi]_i = s_i \pi_i$ for all $i \in [n] \setminus \{1,2\}$. Below we also use λ_{-i} to denote $1 - \lambda_i$.

The Backpropagation Lemma is a consequence of the following technical lemma.

Lemma 3.6. *Let (y, z) be a KKT point of QP (3), for some $\delta \in (0, 1/16K^2)$. Then there exists a perturbation vector $\pi = (\pi_i)_{i \in [n] \setminus \{1,2\}} \in \mathbb{R}^{n-2}$ satisfying*

- *$|\pi_i| \le 8K^2\delta$ for all $i \in [n] \setminus \{1, 2\}$,*
- *for all $s \in \{+1, -1\}^{n-2}$, $f^{s \cdot \pi}$ is differentiable in a small neighborhood around $(x_1, x_2) = (y_1, y_2)$.*

In addition, there exists $\lambda = (\lambda_i)_{i \in [n] \setminus \{1,2\}} \in [0, 1]^{n-2}$ such that for $k = 1, 2$

$$\frac{\partial p}{\partial y_k}(y, z) = \delta^{n+1} \sum_{s \in \{+1,-1\}^{n-2}} \left(\prod_{j=3}^{n} \lambda_{s_j \cdot j} \right) \frac{\partial f^{s \cdot \pi}}{\partial x_k}(y_1, y_2).$$

Before moving to the proof of the technical lemma, let us see why it implies the Backpropagation Lemma. From the two bullets we obtain by definition of the generalized circuit gradient that

$$\nabla f^{s \cdot \pi}(y_1, y_2) = \left(\frac{\partial f^{s \cdot \pi}}{\partial x_1}(y_1, y_2), \frac{\partial f^{s \cdot \pi}}{\partial x_2}(y_1, y_2) \right) \in \widetilde{\partial}_{\delta'} C(y_1, y_2)$$

for all $s \in \{+1, -1\}^{n-2}$, and where we let $\delta' := 8K^2\delta$. As a result of the last part of the technical lemma we can write

$$\frac{1}{\delta^{n+1}} \cdot \left(\frac{\partial p}{\partial y_1}(y, z), \frac{\partial p}{\partial y_2}(y, z) \right) = \sum_{s \in \{+1,-1\}^{n-2}} \left(\prod_{j=3}^{n} \lambda_{s_j \cdot j} \right) \nabla f^{s \cdot \pi}(y_1, y_2) \in \widetilde{\partial}_{\delta'} C(y_1, y_2)$$

since this is a convex combination of elements in $\widetilde{\partial}_{\delta'} C(y_1, y_2)$, and this set is convex by definition. This is exactly the statement of the Backpropagation Lemma.

3.3.1 Proof of the Technical Lemma. Let (y, z) be a KKT point of (3). For $i \in [n] \setminus \{1\}$, let $\varepsilon_i := (2K\delta)^{n-i}$.

Construction of π. Let $i \in [n] \setminus \{1, 2\}$. We construct π_i as follows

- If $|\sum_{j=1}^{i-1} a_{ij} y_j + c_i - 1| \le 2K\varepsilon_{i-1}$, then we set $\pi_i := -4K\varepsilon_{i-1}$.
- If $|\sum_{j=1}^{i-1} a_{ij} y_j + c_i - 0| \le 2K\varepsilon_{i-1}$, then we set $\pi_i := 4K\varepsilon_{i-1}$.
- In all other cases we set $\pi_i := 0$.

Note that the two first cases cannot both occur, since $2K\varepsilon_{i-1} \le 2K\varepsilon_{n-1} = 4K^2\delta < 1/4$, since $\delta < 1/16K^2$. Furthermore, since $2K\varepsilon_{i-1} < 1/4$, we also have that $\sum_{j=1}^{i-1} a_{ij} y_j + c_i + \pi_i \notin (-2K\varepsilon_{i-1}, 2K\varepsilon_{i-1}) \cup (1 - 2K\varepsilon_{i-1}, 1 + 2K\varepsilon_{i-1})$, and similarly $\sum_{j=1}^{i-1} a_{ij} y_j + c_i - \pi_i \notin (-2K\varepsilon_{i-1}, 2K\varepsilon_{i-1}) \cup (1 - 2K\varepsilon_{i-1}, 1 + 2K\varepsilon_{i-1})$.

Construction of λ. Let $i \in [n] \setminus \{1, 2\}$. We construct $\lambda_i \in [0, 1]$ as follows

- If $\sum_{j=1}^{i-1} a_{ij} y_j + c_i < -2K\varepsilon_{i-1}$, then set $\lambda_i := 0$.
- If $\sum_{j=1}^{i-1} a_{ij} y_j + c_i > 1 + 2K\varepsilon_{i-1}$, then set $\lambda_i := 0$.
- If $\sum_{j=1}^{i-1} a_{ij} y_j + c_i \in (2K\varepsilon_{i-1}, 1 - 2K\varepsilon_{i-1})$, then set $\lambda_i := 1$.
- Otherwise, pick $\lambda_i \in [0, 1]$ as a solution of the equation

$$\frac{1}{2\delta^i} \cdot \frac{\partial p_i}{\partial y_i}(y, z) \cdot \lambda_i = \mathsf{T}\left(\sum_{j=1}^{i-1} a_{ij} y_j + c_i \right) - \mathsf{T}\left(\sum_{j=1}^{i-1} a_{ij} y_j + c_i - \frac{1}{2\delta^i} \cdot \frac{\partial p_i}{\partial y_i}(y, z) \right). \tag{6}$$

Note that such $\lambda_i \in [0, 1]$ always exists.

In fact, it is not hard to see that λ_i satisfies (6) in all four cases. This follows from the fact that by the proof of Lemma 3.5

$$\left| \frac{1}{2\delta^i} \cdot \frac{\partial p_i}{\partial y_i}(y, z) \right| \le (2K\delta)^{n+1-i} = \varepsilon_{i-1} \le K\varepsilon_{i-1}.$$

Furthermore, note that by Lemma 3.5 we can rewrite the equation satisfied by λ_i as

$$\frac{1}{2\delta^i} \cdot \frac{\partial p_i}{\partial y_i}(y, z) \cdot \lambda_i = \mathsf{T}\left(\sum_{j=1}^{i-1} a_{ij} y_j + c_i \right) - y_i. \tag{7}$$

Before stating an important claim satisfied by π and λ, we introduce some additional notation. For $i \in [n] \setminus \{1, 2\}$ and any $s_i \in \{+1, -1\}$, we define the function $\phi_i^{s_i \cdot \pi_i} : \mathbb{R}^{i-1} \to \mathbb{R}$ by $\phi_i^{s_i \cdot \pi_i}(x_1, \ldots, x_{i-1}) = \mathsf{T}(\sum_{j=1}^{i-1} a_{ij} x_j + c_i + s_i \cdot \pi_i)$. Recall that we use λ_{-i} to denote $1 - \lambda_i$. The following claim is proved in the full version.

Claim 1. *For $i \in [n] \setminus \{1, 2\}$ and for any $s_i \in \{+1, -1\}$, the function $\phi_i^{s_i \cdot \pi_i}$ is differentiable (with respect to its inputs $x_1, \ldots, x_{i-1}$) over $\prod_{j \in [i-1]} [y_j - \varepsilon_{i-1}, y_j + \varepsilon_{i-1}]$ and we have*

$$\frac{\partial \phi_i^{s_i \cdot \pi_i}}{\partial x_k}(v_1, \ldots, v_{i-1}) = \frac{\partial \phi_i^{s_i \cdot \pi_i}}{\partial x_k}(y_1, \ldots, y_{i-1})$$

for all $k \in [i-1]$ and all $v \in \prod_{j \in [i-1]} [y_j - \varepsilon_{i-1}, y_j + \varepsilon_{i-1}]$. Furthermore, we also have

$$\lambda_i \cdot \frac{\partial \phi_i^{\pi_i}}{\partial x_k}(y_1, \ldots, y_{i-1}) + \lambda_{-i} \cdot \frac{\partial \phi_i^{-\pi_i}}{\partial x_k}(y_1, \ldots, y_{i-1}) = \lambda_i \cdot a_{ik}$$

for all $k \in [i-1]$.

We are now ready to prove the technical lemma. We will prove a slightly stronger statement by induction. For this, we need some additional notation. For $i \in [n] \setminus \{1\}$, we let C_i^π denote the circuit C^π but where we have only kept the gates $i + 1, \ldots, n$. We think of C_i^π as having input variables $x_1, x_2, \ldots, x_i$ (even though some of those variables might be unused and thus not have any impact on the output of the circuit). We let $f_i^\pi : \mathbb{R}^i \to \mathbb{R}$ denote the function represented by C_i^π. Note that $f_2^\pi = f^\pi$ and $f_n^\pi(x_1, \ldots, x_n) = x_n$.

Claim 2. *For any $i \in [n] \setminus \{1\}$ we have*

1. *For any $s \in \{+1, -1\}^{n-2}$, the function $f_i^{s \cdot \pi}$ is differentiable (with respect to its inputs $x_1, x_2, \ldots, x_i$) over $\prod_{j \in [i]} [y_j - \varepsilon_i, y_j + \varepsilon_i]$ and we have*
$$\frac{\partial f_i^{s \cdot \pi}}{\partial x_k}(v_1, \ldots, v_i) = \frac{\partial f_i^{s \cdot \pi}}{\partial x_k}(y_1, \ldots, y_i)$$
for all $k \in [i]$ and all $v \in \prod_{j \in [i]} [y_j - \varepsilon_i, y_j + \varepsilon_i]$.
2. *For any $k \in [i]$ we have*
$$\frac{\partial p_i}{\partial y_k}(y, z) = \delta^{n+1} \sum_{s : s_j = 1\ \forall j \le i} \left(\prod_{j=i+1}^{n} \lambda_{s_j \cdot j} \right) \frac{\partial f_i^{s \cdot \pi}}{\partial x_k}(y_1, \ldots, y_i).$$

The proof of the claim is omitted due to space constraints and can be found in the full version. The technical lemma (Lemma 3.6) simply follows from the claim by noting that for $i = 2$ we have $f_2^{s \cdot \pi} = f^{s \cdot \pi}$ and $p_2 = p$. Furthermore, note that for all $i \in [n] \setminus \{1, 2\}$ we have $|\pi_i| \le 4K\varepsilon_{i-1} \le 4K\varepsilon_{n-1} = 8K^2\delta$, as desired.

4 CONCLUSIONS, OPEN PROBLEMS

Open problems include the question of whether our result continues to hold for restrictions of QPs such as the bilinear case, where there are no squared terms. [4] show that it is CLS-complete to find a KKT-point of a *multilinear* degree-five polynomial. They use this multilinearity to show CLS-completeness also for the problem of finding a mixed equilibrium of degree-five polytensor games, on the way to showing CLS-completeness for finding a mixed equilibrium of a congestion game. One of the main open problems arising from their work is whether it is CLS-hard to find a mixed equilibrium of a "network coordination" game, i.e., a degree-2 polytensor game. This would follow if our result continued to hold without squared terms. Other special cases of interest include: no linear terms (the NP-hardness results of [2] apply to QPs that have no linear terms, i.e., quadratic forms). Our result also exploits exponential ratios between coefficients, leaving open the question of whether it should hold if coefficients are presented in unary. In connection with their discussion of KKT solutions, [27] also ask about the special case where there is a single local minimum (hence the global minimum). This latter problem would have to be treated as a promise problem.

ACKNOWLEDGMENTS

J.F. and R.S. were supported by EPSRC Grant EP/W014750/1. P.W.G. is supported by EPSRC Grant EP/X040461/1. A.H. was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number MB22.00026.

REFERENCES

[1] Amir Ali Ahmadi and Jeffrey Zhang. 2022. Complexity aspects of local minima and related notions. *Advances in Mathematics* 397, Article 108119 (2022). https://doi.org/10.1016/j.aim.2021.108119

[2] Amir Ali Ahmadi and Jeffrey Zhang. 2022. On the complexity of finding a local minimizer of a quadratic function over a polytope. *Mathematical Programming* 195 (2022), 783–792. https://doi.org/10.1007/s10107-021-01714-2

[3] Amar Andjouh and Mohand Ouamer Bibi. 2022. Adaptive Global Algorithm for Solving Box-Constrained Non-convex Quadratic Minimization Problems. *J. Optim. Theory Appl.* 192, 1 (2022), 360–378. https://doi.org/10.1007/s10957-021-01980-2

[4] Yakov Babichenko and Aviad Rubinstein. 2021. Settling the complexity of Nash equilibrium in congestion games. In *Proceedings of the 53rd ACM Symposium on Theory of Computing (STOC)*. 1426–1437. https://doi.org/10.1145/3406325.3451039

[5] Mihir Bellare and Phillip Rogaway. 1995. The complexity of approximating a nonlinear program. *Mathematical Programming* 69 (1995), 429–441. https://doi.org/10.1007/bf01585569

[6] Immanuel M. Bomze, Mirjam Dür, Etienne de Klerk, Cornelis Roos, Arie J. Quist, and Tamás Terlaky. 2000. On Copositive Programming and Standard Quadratic Optimization Problems. *J. Glob. Optim.* 18, 4 (2000), 301–320. https://doi.org/10.1023/A:1026583532263

[7] Sébastien Bubeck. 2014. Convex Optimization: Algorithms and Complexity. *CoRR* abs/1405.4980 (2014). arXiv:1405.4980

[8] Samuel Burer and Adam N. Letchford. 2009. On Nonconvex Quadratic Programming with Box Constraints. *SIAM J. Optim.* 20, 2 (2009), 1073–1089. https://doi.org/10.1137/080729529

[9] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. 2009. Settling the complexity of computing two-player Nash equilibria. *J. ACM* 56, 3 (2009), 14:1–14:57. https://doi.org/10.1145/1516512.1516516

[10] Arka Rai Choudhuri, Pavel Hubáček, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. 2019. Finding a Nash equilibrium is no easier than breaking Fiat-Shamir. In *Proceedings of the 51st ACM Symposium on Theory of Computing (STOC)*. 1103–1114. https://doi.org/10.1145/3313276.3316400

[11] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. 2009. The complexity of computing a Nash equilibrium. *SIAM J. Comput.* 39, 1 (2009), 195–259. https://doi.org/10.1137/070699652

[12] Constantinos Daskalakis and Christos Papadimitriou. 2011. Continuous local search. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 790–804. https://doi.org/10.1137/1.9781611973082.62

[13] Edith Elkind, Abheek Ghosh, and Paul W. Goldberg. 2022. Simultaneous Contests with Equal Sharing Allocation of Prizes: Computational Complexity and Price of Anarchy. In *Proceedings of the 15th International Symposium on Algorithmic Game Theory (SAGT)*. 133–150. https://doi.org/10.1007/978-3-031-15714-1_8

[14] John Fearnley, Paul Goldberg, Alexandros Hollender, and Rahul Savani. 2022. The Complexity of Gradient Descent: CLS = PPAD ∩ PLS. *J. ACM* 70, 1 (2022), 7:1–7:74. https://doi.org/10.1145/3568163

[15] Uriel Feige and Joe Kilian. 1994. Two prover protocols: low error at affordable rates. In *Proceedings of the 26th ACM Symposium on Theory of Computing*. 172–183. https://doi.org/10.1145/195058.195128

[16] Uriel Feige and László Lovász. 1992. Two-Prover One-Round Proof Systems: Their Power and Their Problems (Extended Abstract). In *Proceedings of the 24th ACM Symposium on Theory of Computing*. 733–744. https://doi.org/10.1145/129712.129783

[17] Pavel Hubáček and Eylon Yogev. 2020. Hardness of continuous local search: Query complexity and cryptographic lower bounds. *SIAM J. Comput.* 49, 6 (2020), 1128–1172. https://doi.org/10.1137/17M1118014

[18] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. 2021. SNARGs for Bounded Depth Computations and PPAD Hardness from Sub-Exponential LWE. In *Proceedings of the 53rd ACM Symposium on Theory of Computing (STOC)*. 708–721. https://doi.org/10.1145/3406325.3451055

[19] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. 1988. How easy is local search? *J. Comput. System Sci.* 37, 1 (1988), 79–100. https://doi.org/10.1016/0022-0000(88)90046-3

[20] Mark W. Krentel. 1990. On Finding and Verifying Locally Optimal Solutions. *SIAM J. Comput.* 19, 4 (1990), 742–749. https://doi.org/10.1137/0219052

[21] Nimrod Megiddo and Christos H. Papadimitriou. 1991. On total functions, existence theorems and computational complexity. *Theoretical Computer Science* 81, 2 (1991), 317–324. https://doi.org/10.1016/0304-3975(91)90200-L

[22] T.S. Motzkin and E.G. Strauss. 1965. Maxima for graphs and a new proof of a theorem of Turán. *Canadian Journal of Mathematics* 17 (1965), 553–540. https://doi.org/10.4153/CJM-1965-053-6

[23] Katta G. Murty and Santosh N. Kabadi. 1987. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming* 39, 2 (1987), 117–129. https://doi.org/10.1007/BF02592948

[24] Christos H. Papadimitriou. 1994. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. System Sci.* 48, 3 (1994), 498–532. https://doi.org/10.1016/S0022-0000(05)80063-7

[25] Panos M. Pardalos and G. Schnitger. 1988. Checking local optimality in constrained quadratic programming is NP-hard. *Operations Research Letters* 7, 1 (1988), 33–35. https://doi.org/10.1016/0167-6377(88)90049-1

[26] Panos M. Pardalos and Stephen A. Vavasis. 1991. Quadratic programming with one negative eigenvalue is NP-hard. *J. Glob. Optim.* 1, 1 (1991), 15–22. https://doi.org/10.1007/BF00120662

[27] Panos M. Pardalos and Stephen A. Vavasis. 1992. Open questions in complexity theory for numerical optimization. *Mathematical Programming* 57 (1992), 337–339. https://doi.org/10.1007/BF01581088

[28] Panos M. Pardalos, Yinyu Ye, and Chi-Geun Han. 1991. Algorithms for the Solutions of Quadratic Knapsack Problems. *Linear Algebra Appl.* 152 (1991), 69–91. https://doi.org/10.1016/0024-3795(91)90267-Z

[29] Aviad Rubinstein. 2018. Inapproximability of Nash equilibrium. *SIAM J. Comput.* 47, 3 (2018), 917–959. https://doi.org/10.1137/15M1039274

[30] Sartaj Sahni. 1972. Some Related Problems from Network Flows, Game Theory and Integer Programming. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory (SWAT)*. 130–138. https://doi.org/10.1109/SWAT.1972.23

[31] Sartaj Sahni. 1974. Computationally Related Problems. *SIAM J. Comput.* 3, 4 (1974), 262–279. https://doi.org/10.1137/0203021

[32] Emanuel Tewolde, Caspar Oesterheld, Vincent Conitzer, and Paul W. Goldberg. 2023. The Computational Complexity of Single-Player Imperfect-Recall Games. *CoRR* abs/2305.17805 (2023). arXiv:2305.17805

[33] Stephen A. Vavasis. 1990. Quadratic programming is in NP. *Inform. Process. Lett.* 36, 2 (1990), 73–77. https://doi.org/10.1016/0020-0190(90)90100-c

[34] Stephen A Vavasis and Richard Zippel. 1990. *Proving Polynomial-Time for Sphere-Constrained Quadratic Programming*. Technical Report TR90-1182. Cornell University. https://hdl.handle.net/1813/7022

[35] Yinyu Ye. 1992. On affine scaling algorithms for nonconvex quadratic programming. *Mathematical Programming* 56 (1992), 285–300. https://doi.org/10.1007/bf01580903

[36] Yinyu Ye. 1998. On the complexity of approximating a KKT point of quadratic programming. *Math. Program.* 80 (1998), 195–211. https://doi.org/10.1007/BF01581726

[37] Yinyu Ye. 1999. Approximating quadratic programming with bound and quadratic constraints. *Math. Program.* 84, 2 (1999), 219–226. https://doi.org/10.1007/s10107980012a

Received 13-NOV-2023; accepted 2024-02-11