Math. Stat. Learn. 9 (2026), 145–198
DOI 10.4171/MSL/56

The query complexity of local search and Brouwer in rounds

Simina Brânzei and Jiawei Li

Abstract. We consider the query complexity of finding a local minimum of a function defined on a graph. This abstract problem is fundamental to many optimization tasks, such as finding a local minimum of the loss function when training deep neural networks. In such applications, each query is an expensive loss evaluation, making it crucial to parallelize computations. This motivates our study of local search where at most k rounds of interaction (aka adaptivity) with the oracle are allowed.

We focus on the d-dimensional grid $\{1, 2, \ldots, n\}^d$, where the dimension $d \geq 2$ is a constant. Our main contribution is to give algorithms and lower bounds that characterize the query complexity of finding a local minimum in k rounds, when k is constant and polynomial in n, respectively. Our proof technique for lower bounding the query complexity in rounds may be of independent interest as an alternative to the classical relational adversary method of Aaronson from the fully adaptive setting. The local search analysis also enables us to characterize the query complexity of computing a Brouwer fixed point in rounds.

1. Introduction

Local search is a powerful heuristic embedded in many natural processes, which is often used to solve hard optimization problems. Algorithms based on local search include gradient methods, the Lin–Kernighan algorithm for the traveling salesman problem, and the Metropolis–Hastings algorithm for sampling. The complexity of local search has been extensively studied in both the white box model (see, e.g., [36]) and the black box (aka query) model (see, e.g., [3]).

In the black box model, there is a graph $G = (V, E)$ and an unknown function $f: V \to \mathbb{R}$. An algorithm must query a vertex v to learn $f(v)$. The goal is to find a vertex v that is a local minimum: $f(v) \leq f(u)$ for all $(u, v) \in E$. The query complexity is the number of oracle queries needed to find a local minimum in the worst case.

The query complexity of local search was first considered by Aldous [3], who showed that steepest descent with a warm start is a good randomized algorithm: first query t vertices $x_1, \ldots, x_t$ selected uniformly at random and pick the vertex x^* that

Mathematics Subject Classification 2020: 68Q25.
Keywords: local search, query complexity, rounds, parallelism, optimization, fixed points.

minimizes the function among these.[1] Then run steepest descent from x^* and stop when no further improvement can be made, returning the final vertex reached. When $t = \sqrt{\Delta N}$, where N is the number of vertices and Δ the maximum degree of the graph G, the algorithm issues $O(\sqrt{\Delta N})$ queries in expectation and has roughly as many rounds of interaction with the oracle.

Local search in rounds. Aldous' algorithm described above is highly sequential, i.e., requires many rounds of interaction with the oracle, even though its total query complexity is essentially optimal for graphs such as the Boolean hypercube and the d-dimensional grid [3, 48, 57]. Multiple rounds of interaction can be expensive in applications. For instance, when algorithms such as gradient descent are run on data stored in the cloud, there can be delays due to back and forth messaging across the network. A remedy for such delays is designing protocols with fewer rounds of communication.

In this paper, we analyze the query complexity of local search in rounds, focusing on the case where the graph G is the d-dimensional grid of side length n. When an algorithm has k rounds of interaction with the oracle, it asks a batch of queries in each round i, receives the answers, and then issues the batch of queries for round $i + 1$ (so, within a round, queries are not adaptive). The algorithm stops and outputs an answer by the end of round k.

Examples. To illustrate our model, we describe an analogy to the classical optimization problem of linear regression with mean squared error (MSE) loss. Given a data set of m labeled examples $\{(\vec{a}_i, b_i)\}_{i=1}^m \subseteq \mathbb{R}^d \times \mathbb{R}$, linear regression aims to find a vector $\vec{x} \in \mathbb{R}^d$ of regression coefficients that minimize the loss function:

$$L(\vec{x}) = \frac{1}{m} \sum_{i=1}^{m} \big(\langle \vec{a}_i, \vec{x} \rangle - b_i\big)^2.$$

While the ideal coefficients exist in a continuous space ($\mathbb{R}^d$), any practical optimization algorithm operates by taking discrete steps within a bounded region, iteratively updating an estimate $\vec{x}_{\text{old}}$ to a new one, $\vec{x}_{\text{new}}$. To formally study the complexity of such a search, we can abstract this process by modeling the search space as a discrete grid. We consider a bounded region of potential coefficients, such as $[-B, B]^d$ for some $B > 0$, and discretize it into the grid $[n]^d$. Each vertex $\vec{x} \in [n]^d$ corresponds to a candidate vector of regression coefficients. Querying a vertex $\vec{x}$ returns its MSE loss, $f(\vec{x}) = L(\vec{x})$. The neighborhood of a vertex $\vec{x}$ consists of all adjacent grid points, representing the smallest possible adjustments to a single regression coefficient.

[1] That is, the vertex x^* is defined as: $x^* = x_j$, where $j = \arg\min_{i=1}^{t} f(x_i)$.

A local minimum in this model is a set of coefficients whose loss cannot be improved by changing one coefficient by a discrete step. Performing local search in k rounds corresponds to evaluating multiple candidate vectors simultaneously in k batches. In each round, an algorithm queries a batch of points simultaneously, uses the results to select the next batch, and repeats this process k times.

While linear regression with MSE loss is convex, many practical variations are not. For instance, adding a non-convex regularizer like the minimax concave penalty (MCP) [56] creates a loss function with multiple local minima:

$$L_{\mathrm{MCP}}(\vec{x}) = \frac{1}{m}\sum_{i=1}^{m}\left(\langle \vec{a}_i, \vec{x}\rangle - b_i\right)^2 + \sum_{j=1}^{d}\mathrm{MCP}_{\lambda,\gamma}(x_j),$$

where $\mathrm{MCP}_{\lambda,\gamma}(x_j)$ represents the MCP penalty function for the j-th regression coefficient x_j; λ is a regularization parameter that controls the overall strength of the penalty; and γ is a tuning parameter that controls the concavity of the MCP function.

Since finding the global minimum of non-convex functions is computationally infeasible in general, the practical goal shifts to efficiently finding a local minimum.

Roadmap to the paper. The remainder of the paper is organized as follows. We formalize the model in Section 2 and present our main theorems in Section 3. Related work is discussed in Section 4. The core technical sections follow, with Section 5 providing proof overviews for our local search results, while Section 6 details the application to the Brouwer fixed-point problem, which has a connection to local search via game theory. Full technical proofs are deferred to the appendix.

2. Model

We introduce the model for local search and Brouwer, then define the query complexity.

Local search. Let $G = (V, E)$ be an undirected graph and $f : V \to \mathbb{R}$ a function that assigns a value to every vertex. The function f is unknown but can be queried; a query is a vertex $\mathbf{x} \in V$ and the answer to the query is the value $f(\mathbf{x})$ of the function at $\mathbf{x}$. A local minimum is a vertex $\mathbf{x}$ with the property that $f(\mathbf{x}) \leq f(\mathbf{y})$ for all neighbors $\mathbf{y}$ of $\mathbf{x}$. The goal is to find a local minimum using as few queries as possible.

We focus on the setting where the graph is a d-dimensional grid of side length n. Thus $V = [n]^d$, where $[n] = \{1, 2, \ldots, n\}$ and $(\mathbf{x}, \mathbf{y}) \in E$ if $\|\mathbf{x} - \mathbf{y}\|_1 = 1$. The dimension d for both local search and Brouwer fixed-point is a constant. Unless otherwise specified, we have $d \geq 2$.

The local search results imply bounds for the problem of finding a Brouwer fixed point, which is defined next.

Brouwer. In the Brouwer setting, we are given an L-Lipschitz function

$$f : [0,1]^d \to [0,1]^d,$$

where $L > 1$ is a constant[2] such that

$$\|f(\mathbf{x}) - f(\mathbf{y})\|_\infty \leq L\|\mathbf{x} - \mathbf{y}\|_\infty \quad \forall \mathbf{x}, \mathbf{y} \in [0,1]^d.$$

The computational problem is: given a tuple (ε, L, d, f), find a point $\mathbf{x}^* \in [0,1]^d$ such that $\|f(\mathbf{x}^*) - \mathbf{x}^*\|_\infty \leq \varepsilon$. An exact fixed point exists by Brouwer's fixed point theorem.

Query complexity and rounds. We have query (aka oracle) access to the function f and at most k rounds of interaction with the oracle. An algorithm running in k rounds will submit in each round j a batch of queries, then wait for the answers, and then submit the batch of queries for round $j+1$. The choice of queries submitted in round j can only depend on the results of queries from earlier rounds. At the end of the k-th round, the algorithm must stop and output a solution.

The deterministic query complexity is the total number of queries necessary and sufficient to find a solution. The randomized query complexity is the expected number of queries required to find a solution with probability at least $2/3$ for any input, where the expectation is taken over the coin tosses of the protocol.[3]

3. Our results

Local search. To set the stage, the query complexity of local search on the d-dimensional grid $[n]^d$ is well understood in the fully adaptive setting. The classical result by Llewellyn, Tovey, and Trick [39] showed that the query complexity for deterministic fully adaptive algorithms is $\Theta(n^{d-1})$, and this upper bound is achieved by a divide-and-conquer algorithm using only $O(\log n)$ rounds; in the randomized setting, Aldous' algorithm [3] takes $O(n^{d/2})$ queries and rounds, and it was proven to be optimal later [48, 57]. More related works are discussed in Section 4.

We show the following bounds for local search on the d-dimensional grid $[n]^d$ in k rounds, which quantify the trade-offs between the number of rounds of adaptivity and the total number of queries.

[2] When $L < 1$ we obtain the Banach fixed-point theorem, where the unique fixed point can be approximated quickly.

[3] Any other constant greater than $1/2$ will suffice.

Theorem 3.1 (Local search, constant rounds). *Let $d, k \in \mathbb{N}$ be constant, where $d \geq 2$ and $k \geq 1$. The query complexity of local search in k rounds on the d-dimensional grid $[n]^d$ is $\Theta(n^{(d^{k+1}-d^k)/(d^k-1)})$ for both deterministic and randomized algorithms.*

For example, when $k = 2$, the query complexity is $\Theta(n^{d^2/(d+1)})$ for both deterministic and randomized algorithms. When $k \to \infty$, the bound of Theorem 3.1 is close to $\Theta(n^{d-1})$, with gap smaller than any polynomial.

Recall that the $O(\log n)$ rounds divide-and-conquer algorithm take $O(n^{d-1})$ queries [39]. Thus, our result fills the gap between one round algorithms and logarithmic rounds algorithms except for a small margin. For constant number of rounds, Theorem 3.1 shows that deterministic algorithms are optimal, so there is no difference between the deterministic and randomized query complexity.

Theorem 3.2 (Local search, polynomial rounds). *Consider the d-dimensional grid with side length $n \in \mathbb{N}$, where $d \geq 2$. Let $k = n^\alpha \in \mathbb{N}$, where $\alpha \in (0, d/2)$ is a constant. The randomized query complexity of local search in k rounds on the d-dimensional grid $[n]^d$ is:*

- $\Theta(n^{(d-1)-((d-2)/d)\alpha})$ *when* $d \geq 5$;
- $O(n^{3-\alpha/2})$ *and* $\widetilde{\Omega}(n^{3-\alpha/2})$ *when* $d = 4$;
- $O(n^{2-\alpha/3})$ *and* $\Omega(\max(n^{2-2\alpha/3}, n^{3/2}))$ *when* $d = 3$.

When $\alpha \to 0$, the bound approaches $\Theta(n^{d-1})$, i.e., the bound of constant and logarithmic rounds algorithm. When $\alpha \to (d/2)$, the upper bound is close to $\Theta(n^{d/2})$, i.e., the randomized fully adaptive algorithm [3, 57]. Thus, our result fills the gaps between constant (or logarithmic) rounds algorithms and fully adaptive algorithms, except for a small gap when $d \in \{3, 4\}$.

The bound for $d = 2$ in polynomial number of rounds was known, since the divide-and-conquer approach by Llewellyn, Tovey, and Trick [39] gives an upper bound of $O(n)$ using only $O(\log n)$ rounds, while Sun and Yao [48, Theorem 1] give a lower bound of $\Omega(n^{1-\delta})$ for all fixed $\delta > 0$ for fully adaptive randomized algorithms.

When $d = 1$, the query complexity of computing a local minimum on the 1-dimensional grid $[n]$ in k rounds is $\Theta(n^{1/k})$, for both deterministic and randomized algorithms (Theorem F.1).

A summary of our results for local search on the d-dimensional grid can be found in Table 1, together with the bounds known in the existing literature.

At a high level, when the number of rounds k is constant, the optimal algorithm is closer to the deterministic divide-and-conquer algorithm [39]. When the number of rounds is polynomial (i.e. $k = n^\alpha$, for $0 < \alpha < d/2$), the algorithm is closer to the randomized algorithm from the fully adaptive setting, which does steepest descent with a warm start [3].

Local search on the grid $[n]^d$	Deterministic	Randomized
Constant rounds: $k = O(1)$	$\Theta\left(n^{(d^{k+1}-d^k)/(d^k-1)}\right)$ (*)	$\Theta\left(n^{(d^{k+1}-d^k)/(d^k-1)}\right)$ (*)
Polynomial rounds: $k = n^\alpha, \alpha \in (0, d/2)$	$\Theta(n^{d-1})$ [5,38,39]	$d \geq 5$: $\Theta\left(n^{(d-1)-((d-2)/d)\alpha}\right)$ (*) $d = 4$: $O(n^{3-\alpha/2})$ and $\widetilde{\Omega}(n^{3-\alpha/2})$ (*) $d = 3$: $O(n^{2-\alpha/3})$ and $\Omega\left(\max\left(n^{2-2\alpha/3}, n^{3/2}\right)\right)$ (*) $d = 2$: $\widetilde{\Theta}(n)$ [39,48]
Fully adaptive: $k = \infty$	$\Theta(n^{d-1})$ [5,38,39]	$d \geq 3$: $\widetilde{\Theta}(n^{d/2})$ [3,57] $d = 2$: $\widetilde{\Theta}(n)$ [39,48]

Table 1. Query complexity of local search in k rounds on d-dimensional grid of side length n, for $d \geq 2$. Our results are marked with (*). The deterministic divide-and-conquer algorithm [39] takes $O(\log n)$ rounds, while the randomized warm-start algorithm [3] needs $O(n^{d/2})$ rounds.

Brouwer. Our local search results above also imply a characterization for the query complexity of finding an approximate Brouwer fixed point in constant number of rounds on the d-dimensional cube. The connection between local search and Brouwer stems from game theory. Finding a Nash equilibrium in a general game is equivalent in complexity to solving the Brouwer problem. However, for specific classes of games, such as potential games, pure Nash equilibria are guaranteed to exist and can be found via a sequence of best-response moves—a process equivalent to local search [7,40,45].

For Brouwer, we consider only constant rounds, since Brouwer can be solved optimally in $O(\log(1/\varepsilon))$ rounds [18].

Theorem 3.3 (Brouwer, constant rounds). *Let $d, k \in \mathbb{N}$ be constant. For any $\varepsilon > 0$, the query complexity of the ε-approximate Brouwer fixed-point problem in the d-dimensional unit cube $[0,1]^d$ with k rounds is $\Theta((1/\varepsilon)^{(d^{k+1}-d^k)/(d^k-1)})$ for both deterministic and randomized algorithms.*

We also show that when $d = 1$, the query complexity of finding an ε-approximate Brouwer fixed point in k rounds is $\Theta((1/\varepsilon)^{1/k})$, for both deterministic and randomized algorithms.

4. Related work

Query complexity of local search. The query complexity of local search was studied first experimentally by Tovey [49], while Aldous [3] provided the first theoretical analysis. For the Boolean hypercube $\{0,1\}^n$, Aldous [3] gave an upper bound of $O(n \cdot 2^{n/2})$ using steepest descent with a warm start and a lower bound of $\Omega(2^{n/2-o(n)})$ for randomized algorithms. The lower bound construction is as follows. Consider an initial vertex v_0 uniformly at random. The function value at v_0 is $f(v_0) = 0$. From this vertex, start an unbiased random walk $v_0, v_1, \ldots$. For each vertex v in the graph, set $f(v)$ equal to the first hitting time of the walk at v; that is, $f(v) = \min\{t \mid v_t = v\}$. The function f defined this way has a unique local minimum at v_0. By analyzing this distribution, Aldous [3] showed a lower bound of $2^{n/2-o(n)}$ on the Boolean hypercube.

The steepest descent with a warm start algorithm [3] is essentially optimal for graphs such as the hypercube and the d-dimensional grid [3, 48, 57]. At the same time, the algorithm is highly sequential. Llewellyn, Tovey, and Trick [39] analyzed the deterministic query complexity of local search and devised a divide-and-conquer approach, which has higher total query complexity but uses fewer rounds. Their algorithm is deterministic and identifies in the first step a vertex separator S of the input graph G.[4] Afterward, it queries all the vertices in S to find the minimum v among these. If v is a local minimum of G, then return it. Otherwise, there is a neighbor w of v with $f(w) < f(v)$. Repeat the whole procedure on the new graph G', defined as the connected component of $G \setminus S$ containing w. Correctness holds since the steepest descent from w cannot escape G'. On the d-dimensional grid, the vertex separator S can be defined as the $(d-1)$-dimensional wall that divides the current connected component evenly; thus a local optimum can be found with $O(n^{d-1})$ queries in $O(\log n)$ rounds.

We observe the contrast between the steepest descent with a warm start algorithm given by Aldous [3], which is randomized and almost sequential—running in $O(n^{d/2})$ rounds—and the deterministic divide-and-conquer algorithm from [39], which can be implemented in $O(\log n)$ rounds. Even though the randomized algorithm in [3] is (essentially) optimal in terms of number of queries, it takes many rounds. Thus it is natural to ask whether steepest descent with a warm start can be parallelized and what is the tradeoff between the total query complexity and the number of rounds.

Althöfer and Koschnick [5], and Llewellyn and Tovey [38] applied the adversarial argument to show that $\Omega(n^{d-1})$ queries are necessary for any deterministic algorithm

[4] A vertex separator is a set of vertices $S \subseteq V$ with the property: there exist vertices $u, v \in V$, where V is the set of vertices of G, such that any path between u and v passes through S.

on the d-dimensional grid of side length n. Llewellyn, Tovey, and Trick [39] also studied arbitrary graphs, showing that $O(\sqrt{N} + \Delta \log N)$ queries are sufficient on graphs with N vertices when the maximum degree of the graph is Δ and the graph has constant genus.

Aaronson [1] improved the lower bounds for randomized algorithms by designing a novel technique called the *relational adversarial method* inspired by the adversarial method in quantum computing. This method avoids analyzing the posterior distribution during the execution directly and gave improved lower bounds for both the hypercube and the grid. Follow-up work by Zhang [57] and Sun and Yao [48] obtained even tighter lower bounds for the grid using this method with better choices on the random process; their lower bound is $\widetilde{\Omega}(n^{d/2})$, which is nearly optimal.

Santha and Szegedy [46] gave a quantum lower bound of $\Omega(\sqrt[8]{s/\Delta} / \log(N))$, where s is the separation number of the graph and Δ the maximum degree. This implies the same lower bound in a randomized context, using the spectral method. Meanwhile, the best known upper bound is $O((s + \Delta) \cdot \log N)$ due to [46], which was obtained via a refinement of the divide-and-conquer procedure [39].

Dinh and Russell [27] studied Cayley and vertex transitive graphs and gave lower bounds for local search as a function of the number of vertices and the diameter of the graph. Verhoeven [53] obtained upper bounds as a function of the genus of the graph. Brânzei, Choo, and Recker [12] gave lower bounds for arbitrary graphs, as a function of the vertex congestion and separation number of the graph. These imply a lower bound of $\Omega(\sqrt{N}/\log N)$ for bounded-degree expanders, nearly matching the upper bound of $O(\sqrt{N})$ given by the randomized algorithm from Aldous [3] for such graphs.

Communication complexity of local search. In [7], Babichenko, Dobzinski, and Nisan analyzed the communication complexity of local search, which captures the hardness of finding a local optimum in distributed environments, where data may be stored on different computers, from the point of view of the communication cost.

White box complexity of local search and related problems. The computational complexity of local search in the white-box setting is captured by the class PLS, which was defined by Johnson, Papadimitriou, and Yannakakis [36] to model the difficulty of finding locally optimal solutions to optimization problems.

A class related to PLS is PPAD, which was introduced by Papadimitriou [43] to study the computational complexity of finding a Brouwer fixed-point. PPAD contains many natural problems that are computationally equivalent to finding a Brouwer fixed point [19], such as finding an approximate Nash equilibrium in a multi-player or two-player game [20, 24], an Arrow–Debreu equilibrium in a market [21, 52], and a local min-max point [26]. The query complexity of computing an ε-approximate Brouwer fixed point was studied in a series of papers for fully adaptive algorithms [18, 22, 33].

The classes PLS and PPAD are both a subset of TFNP, which contains total function problems that can be solved in non-deterministic polynomial time. Fearnley et al. [29] showed that the class CLS, introduced by Daskalakis and Papadimitriou [25] to capture continuous local search, is equal to PPAD ∩ PLS. The query complexity of continuous local search has also been studied (see, e.g., [34]).

The d-dimensional grid with constant dimension is important in the study of TFNP in complexity theory. For example, the complexity class CLS was first defined as any problem that could be reduced to the 3D-CONTINUOUS-LOCALOPT problem [25]. Göös et al. [32] showed that EOPL = PPAD ∩ PLS using the 2-dimensional grid as a unified model to define problems from several complexity classes (PPAD, PPADS, PLS, EOPL, and SOPL).

Our results also imply bounds for the computational problem BROUWER, where the goal is to compute an approximate fixed point of a Lipschitz function defined on the d-dimensional cube, the existence of which is guaranteed by Brouwer's theorem. BROUWER is computationally equivalent to problems such as finding a Nash equilibrium in an n-player game [20, 24, 41, 43] and a local min-max equilibrium in a two player game with nonconvex-nonconcave utilities [26], the latter of which has applications to training generative adversarial networks [6, 26, 31].

Parallel complexity. Parallel complexity is a fundamental concept, which was studied extensively for problems such as sorting, selection, finding the maximum element of a vector, and the sorted top-k elements [4, 11, 13, 14, 23, 30, 44, 50, 54]. An overview on parallel sorting algorithms is given in the book of Akl [2].

The parallel complexity of optimization was first considered by Nemirovski [42]. This was analyzed in a series of follow-up works for submodular functions (see, e.g., [8–10, 28]). Algorithms with few rounds of interaction with the function oracle (aka low depth) for the problem of computing stationary points were considered by Bubeck and Mikulincer [17]. Bubeck et al. [16] studied parallel convex optimization where the oracle can answer poly(d) queries in each round, where d is the dimension.

Another setting of interest where rounds are important is active learning, where there is an "active" learner who can submit queries—in the form of unlabeled instances—to be annotated by an oracle (e.g., a human) [47]. However, each round of interaction with the human annotator has a cost, which can be captured through a budget on the number of rounds.

5. Local search

In this section, we state our results for local search and give an overview of the proofs.

5.1. Local search in constant rounds

When the number of rounds k is a constant, we obtain the following bounds.

Theorem 3.1 (Local search, constant rounds, restated). *Let $d, k \in \mathbb{N}$ be constant, where $d \geq 2$ and $k \geq 1$. The query complexity of local search in k rounds on the d-dimensional grid $[n]^d$ is $\Theta(n^{(d^k+1-d^k)/(d^k-1)})$ for both deterministic and randomized algorithms.*

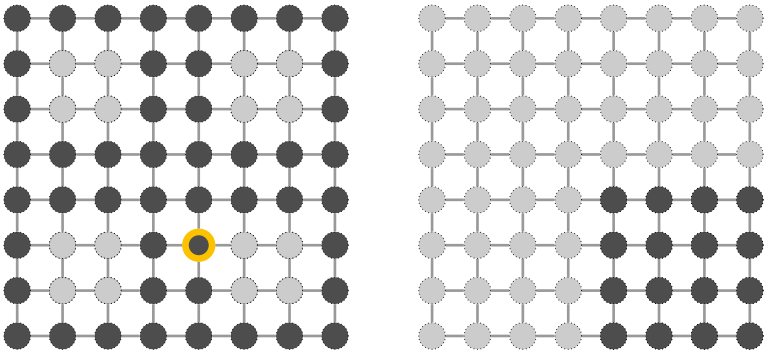
For example, when $k = 2$, the query complexity is $\Theta(n^{d^2/(d+1)})$.

5.1.1. Upper bound overview for local search in constant rounds. The algorithm for constant rounds works as follows. Initialize a cube C_0 to the whole grid $[n]^d$.

In each round $i = 1, \ldots, k-1$, the algorithm divides the current cube C_{i-1} into a set of mutually exclusive sub-cubes $C_i^1, \ldots, C_i^{n_i}$ of side length ℓ_i (for a carefully set value of ℓ_i) that cover C_{i-1}. Then it queries all the points on the boundary of the sub-cubes $C_i^1, \ldots, C_i^{n_i}$ and select the point $\mathbf{x}_i^*$ with minimal value among them. Set $C_i = C_i^j$, where C_i^j is the sub-cube that $\mathbf{x}_i^*$ belongs to and repeat. Finally, in round k, query all the points in the current cube C_{k-1} and find the solution point.

Figure 1 shows an illustration for the case $k = 2$ and $d = 2$.

The main observation that allows the proof to work is that when the space is divided into sub-cubes and $\mathbf{x}_i^*$ is the point with smallest value on the boundaries of all the sub-cubes, steepest descent started at $\mathbf{x}_i^*$ cannot escape the sub-cube in which it

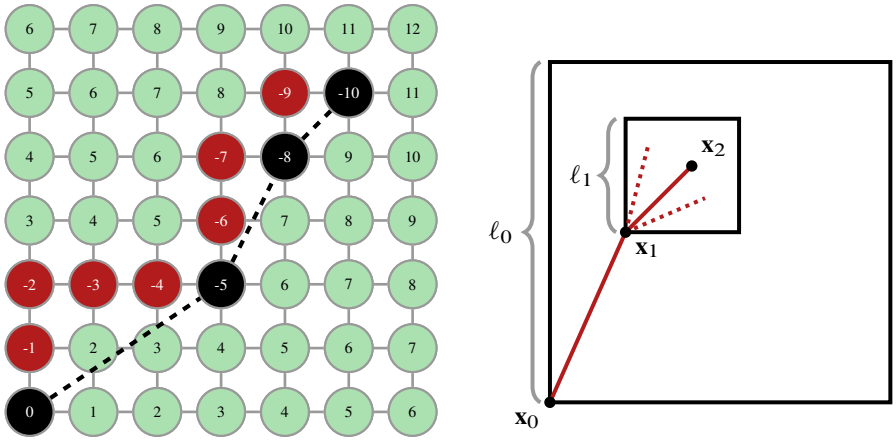


(a) Round 1 **(b)** Round 2

Figure 1. 2-dimensional grid of size 8×8. Suppose there are two rounds. In round 1, illustrated on the left, the algorithm queries all the black points and selects the minimum among all these points (shown in yellow). In round 2, illustrated on the right, it queries the entire sub-square in which the minimum from the first round was found. When $d = 2$ and $k = 2$, the query complexity is $\Theta(n^{4/3})$ by setting $\ell_1 := n^{2/3}$.

is contained. By choosing the side lengths of the sub-cubes appropriately, we get the required upper bound. The detailed proof is included in Appendix A, together with the other omitted proofs.

5.1.2. Lower bound overview for local search in constant rounds. To show randomized lower bounds, we apply Yao's minimax theorem [55]: first we provide a hard distribution of inputs, then show that no *deterministic* algorithm could achieve accuracy larger than some constant on this distribution. The hard distribution will be given by a *staircase* construction [33, 51]. A staircase will be a random path with the property that the unique local optimum is hidden at the end of the path. We present a sketch here, while the complete calculations can be found in Appendix C.



(a) Staircase in two dimensions for 3 rounds **(b)** Stylized staircase in two dimensions

Figure 2. Figure (a) shows a 2-dimensional staircase for $k = 3$ rounds. The number of black points is equal to $k + 1$. The black points are connected by "folded-segments" shown in red. The idea is that in each round, the algorithm will learn at most one more folded-segment. Figure (b) shows in a stylized way the process of selecting the next part of the staircase given that the first folded-segment was determined. The folded-segments are displayed here as straight lines for simplicity.

Figure 2 shows an example of a staircase, which consists of the black and red vertices. The bottom left black vertex is the starting point of the staircase and the value of the function there is set to zero. Then the value decreases by one with each step along the staircase, like going down the stairs. The value of the function at any point outside the staircase is equal to the distance to the entrance of the staircase.

Intuitively, the algorithm cannot find much useful structural information of the input and thus has no advantage over a path following algorithm. The staircase construction can be embedded in two different tasks: finding a local minimum of a function [1, 3, 34, 48, 57] and computing a Brouwer fixed-point [22, 33].

The most challenging part is rigorously proving that such intuition is correct. The lower bound proof from [3] for the hypercube $\{0,1\}^n$ has complex probability analysis on the random walk and relies on the structural property of hypercube $\{0,1\}^n$.

All other works on the randomized lower bound of local search are based on Aaronson's relational adversarial method [1]. However, we found it is inherently difficult to consider multiple queries in one round in the framework of relational adversarial method. Therefore, our main technical innovation is a new technique to incorporate the round limit into the randomized lower bounds. This could also serve as a simpler alternative method of the classical relational adversarial method [1] in the fully adaptive setting.

Staircase definition. We define a staircase as an array of connecting grid points $(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t)$, for $0 \leq t \leq k$. A uniquely determined path called folded-segment is used to link every two consecutive points $(\mathbf{x}_i, \mathbf{x}_{i+1})$. The start point $\mathbf{x}_0$ is fixed at corner $\mathbf{1}$, and the remaining connecting points are chosen randomly in a smaller and smaller cube region with previous connecting points as corner. For k rounds algorithms, we choose a distribution of staircases of "length" k, where the length is defined as the number of connecting points in the staircase minus 1.

Good staircases. We say that a length t staircase is "good" with respect to a deterministic algorithm $\mathcal{A}$ if for each $1 \leq i < t$, any point in the suffix of the staircase (i.e., after the connecting point $\mathbf{x}_i$)[5] is not queried in rounds $1, \ldots, i$, when $\mathcal{A}$ runs on the input generated by this staircase.

The input functions generated by good staircases are like adversarial inputs: $\mathcal{A}$ could only (roughly) learn the location of the next connecting point $\mathbf{x}_i$ in each round i, and still know little about the staircase from $\mathbf{x}_i$ onwards.

We show that if $9/10$ of all possible length k staircases are good, then the algorithm will make a mistake with probability at least $7/40$. We ensure that each possible staircase is chosen with the same probability; their total number is easy to estimate.

Thus the main technical part of our proof is counting the number of good staircases.

Counting good staircases. We show the following properties about the *prefix* of good staircases:

(P1) If s is a good staircase, then any "prefix" s' of s is also a good staircase.

(P2) Let s_1, s_2 be two good staircases with respect to algorithm $\mathcal{A}$. If the first i connecting points of the staircases are same, then $\mathcal{A}$ will submit the same queries in rounds $1, \ldots, i$ when given as input the functions generated by s_1 and s_2, respectively.

[5] That is, the point $\mathbf{x}_i$ is not included.

We first fix a good staircase $s^{(i-1)}$ of length $i-1$ and consider two good staircases s_1, s_2 of length i that have $s^{(i-1)}$ as prefix. By property (P2), the algorithm $\mathcal{A}$ will make the *same* queries in rounds $1, \ldots, i$ when running on the inputs generated by s_1 and s_2, respectively. This enables estimating the total number of good staircases of length $i+1$ with $s^{(i-1)}$ as prefix. Then by summing over all good staircases of length $i-1$, we get a recursive equation between the number of good staircases of length $i-1$, i, and $i+1$. This will be used to show that most staircases of length k are good.

5.2. Local search in polynomial rounds

When the number of rounds k is polynomial in n, that is $k = n^\alpha$ for some constant $\alpha > 0$, the algorithm that yields the upper bound in Theorem 3.1 is no longer efficient.

We design a different algorithm for this regime and also show an almost matching lower bound. With polynomial rounds we can focus on $d \geq 3$. Recall that Sun and Yao [48] proved a lower bound of $\widetilde{\Omega}(n)$ for fully adaptive randomized algorithms in two dimensions and the divide-and-conquer algorithm by Llewellyn, Tovey, and Trick [39] achieves this bound with only $O(\log(n))$ rounds.

Theorem 3.2 (Local search, polynomial rounds, restated). *Consider the d-dimensional grid with side length $n \in \mathbb{N}$, where $d \geq 2$. Let $k = n^\alpha \in \mathbb{N}$, where $\alpha \in (0, d/2)$ is a constant. The randomized query complexity of local search in k rounds on the d-dimensional grid $[n]^d$ is:*

- $\Theta(n^{(d-1)-(d-2)/d\alpha})$ *when* $d \geq 5$;
- $O(n^{3-\alpha/2})$ *and* $\widetilde{\Omega}(n^{3-\alpha/2})$ *when* $d = 4$;
- $O(n^{2-\alpha/3})$ *and* $\Omega(\max(n^{2-2\alpha/3}, n^{3/2}))$ *when* $d = 3$.

5.2.1. Upper bound overview for local search in polynomial rounds. Since the constant rounds algorithm is not optimal in this regime, we design an algorithm that randomly samples many points in round 1 and then starts searching for the solution from the smallest valued point from round 1. This is similar to the algorithm in [3], except the steepest descent part of Aldous' algorithm is highly sequential.

To get better parallelism, we design a recursive procedure ("fractal-like steepest descent") which parallelizes the steepest descent steps at the cost of more queries. We present the main ideas next.

Sequential procedure. Let $C(\mathbf{x}, s) := \{\mathbf{y} \in [n]^d : \|\mathbf{y} - \mathbf{x}\|_\infty \leq s\}$ be the set of grid points in the d-dimensional cube of side length $2 \cdot s$, centered at point $\mathbf{x}$. Let $\mathrm{rank}(\mathbf{x})$ be the number of points with smaller function value than point $\mathbf{x}$.

Assume that we already have a procedure P and a number $s < n$ such that $P(\mathbf{x})$ will either return a point $\mathbf{y} \in C(\mathbf{x}, s)$ with $\mathrm{rank}(\mathbf{y}) \leq \mathrm{rank}(\mathbf{x}) - s$, or output a correct

solution and halt. Suppose in both cases $P(\mathbf{x})$ takes at most r rounds and Q queries in total for any $\mathbf{x}$. For a concrete example, the base case of the procedure P is s-steps of steepest descent. In this case, $P(\mathbf{x})$ takes s rounds and $O(s)$ queries.

If we want to find a point $\mathbf{x}^*$ with $\text{rank}(\mathbf{x}^*) \leq \text{rank}(\mathbf{x}_0) - t \cdot s$ for any given $\mathbf{x}_0$ or output a correct solution, the naive approach is to run P *sequentially* t times, taking

$$\mathbf{y}_1 = P(\mathbf{x}_0), \mathbf{y}_2 = P(\mathbf{y}_1), \ldots, \mathbf{x}^* = \mathbf{y}_t = P(\mathbf{y}_{t-1}).$$

Since each call of P must wait for the result from the previous call, the naive approach will take $t \cdot r$ rounds and $t \cdot Q$ queries.

Parallel procedure. We can parallelize the previous procedure P using auxiliary variables that are more expensive in queries, but cheaper in rounds. For $i \in [t]$, let $\mathbf{x}_i$ be the point with minimum function value on the boundary of cube $C(\mathbf{x}_{i-1}, s)$, which can be found in only *one* round with $O(s^{d-1})$ queries after getting $\mathbf{x}_{i-1}$, i.e., we get the location of $\mathbf{x}_{i-1}$ at the start of round i. Next, we take $\mathbf{y}_i$ to be $P(\mathbf{x}_{i-1})$ instead of $P(\mathbf{y}_{i-1})$; thus the location of $\mathbf{y}_i$ will be available at round $i + r$. To ensure correctness, we will compare the value of point $\mathbf{y}_i$ with the value of point $\mathbf{x}_i$. If $f(\mathbf{x}_i) \leq f(\mathbf{y}_i)$ then

$$\text{rank}(\mathbf{x}_i) \leq \text{rank}(\mathbf{y}_i) \leq \text{rank}(\mathbf{x}_{i-1}) - s. \tag{5.1}$$

Otherwise, since $\mathbf{y}_i \in C(\mathbf{x}_{i-1}, s)$ has smaller value than any point on the boundary of $C(\mathbf{x}_{i-1}, s)$, we could use a slightly modified version of the divide-and-conquer algorithm of [39] to find the solution within the sub-cube $C(\mathbf{x}, s)$ in $\log n$ rounds and $O(s^{d-1})$ queries, and then halt all running procedures. If $f(\mathbf{x}_i) \leq f(\mathbf{y}_i)$ holds for any $i \in [t]$, applying inequality 5.1 for t times we will get $\text{rank}(\mathbf{x}) \leq \text{rank}(\mathbf{x}_t) - t \cdot s$, so we could return $\mathbf{x}^* = \mathbf{x}_t$ in this case. This parallel approach will take only $t + r$ rounds and $O(t \cdot (Q + s^{d-1}))$ queries.

The base case of procedure P is the steepest descent algorithm. Then, multiple layers of the recursive process as described above are implemented to ensure the round limit is met. The parameters of the algorithm, such as s and t above, are described in Appendix B.

5.2.2. Lower bound overview for local search in polynomial rounds. For polynomial rounds, we still use a staircase construction and hide the solution at the end of the staircase. Recall the bottom left vertex will be the starting point of the staircase and the value of the function there is set to zero. Then the value decreases by one with each step along the path. The value of the function at any point outside the staircase is equal to the distance to the entrance of the staircase.

However, the case of polynomial number of rounds is both conceptually and technically more challenging. We explain the main ideas next; the full proof is in Appendix D.

Choice of random walk. Let $\mathcal{Q}_k$ denote the total number of queries allowed for an algorithm that runs in k rounds. Let $\mathcal{Q} = \mathcal{Q}_k/k$ be the average number of queries in each round. The minimum point among $\mathcal{Q}_k/2$ uniformly random queries will be at most $100 \cdot n^d/\mathcal{Q}_k$ steps away from the solution with high probability.

We set the number of points in the staircase to $\Theta(n^d/\mathcal{Q}_k)$. This strikes a balance between two extremes. If the staircase is too long, then an algorithm like steepest descent with warm-start [3], which starts by querying many random points in round 1, is likely to hit the staircase in a region that is $O(n^d/\mathcal{Q}_k)$ close to the endpoint. If the staircase is too short, then an algorithm such as steepest descent will find the end of the staircase in a few rounds.

Since we choose the staircase via a random walk, there are two key factors affect the difficulty of finding the solution: the mixing time and what we call the "local predictability" of the walk. We now explain how the algorithm could exploit a slow mixing time or a high local predictability with two random walks in Figure 3.

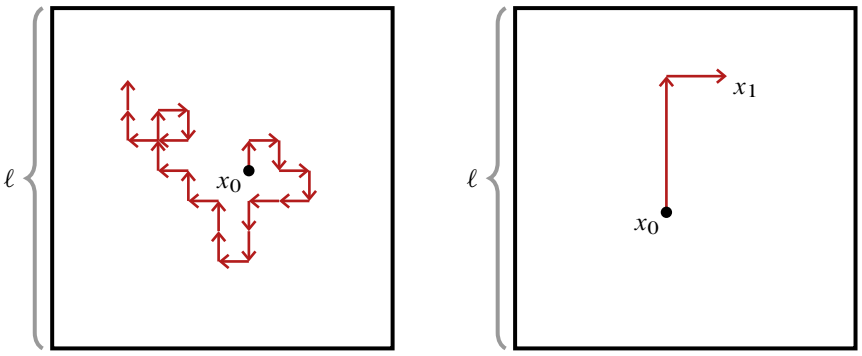


Figure 3. A 2-dimensional illustration for two types of random walk for the lower bound in polynomial rounds. The random walk on the left is more convoluted locally, but also mixes slower; the walk on the right has simpler structure, but mixes faster.

The first random walk (Figure 3, left) randomly moves to one of its neighbor in each step. This random walk has very low local predictability and may be difficult for fully adaptive algorithms to learn. However, it mixes rather slowly, which could be exploited by the following simple algorithm: In each round, it queries all the points on the boundary of the cube of side length s, with the current best point as the center. This random walk will keep dwelling within this cube before reaching the boundary of it for $\Theta(s^2)$ steps in expectation. Therefore, the best point on the boundary gets $\Theta(s^2)$ steps closer to the end point than the previous best point in expectation. However, we only allow this algorithm to get $O(s)$ steps closer to the end if we want a tight bound.

The second random walk (Figure 3, right) moves from x_0 to a uniform random point x_1 selected from a cube of side length ℓ centered at x_0 and uses d straight

segments to connect these two points. This random walk is more locally predictable, since each straight segment of length $O(\ell)$ could be learned with $O(\log \ell)$ queries via binary search. Thus, this walk could be tracked more efficiently than the naive steepest descent in the fully adaptive setting. On the other hand, this random walk mixes faster than the one in the left figure: it takes only $O(\ell)$ steps to mix in the cube region of side length ℓ. Thus, the second random walk might be better when there are not enough rounds to find every straight segment via binary search.

By controlling the parameter ℓ, we get a trade-off between the mixing time and the local predictability. We choose $\ell = \Theta(\mathcal{Q}^{1/(d-1)}) = n^{1-2\alpha/d}$ for $k = n^{\alpha}$ in our proof, which corresponds to the max possible side length of the cube if using $\mathcal{Q}$ queries to cover its boundary.

Measuring the progress. Good staircases are a central concept in the proof for constant rounds. Roughly, the algorithm can learn the location of exactly one more connecting point in each round. However, such a requirement is too strong with polynomial rounds. Instead, we allow the algorithm to learn more than one connecting points in some rounds, while showing that it learns no more than two connecting points in each round in expectation.

Using amortized analysis, we quantify the maximum possible *progress* of an algorithm in each round by a constant Γ, which only depends on the random walk, not the algorithm. Constant Γ could be viewed as the difficulty of the random walk, which takes both the mixing time and the local predictability into account.

6. Brouwer

The problem of finding an ε-approximate fixed point of a continuous function was defined in Section 2. To quantify the query complexity of this problem, it is useful to consider a discrete version, obtained by discretizing the unit cube $[0, 1]^d$. The discrete version of Brouwer is equivalent to the approximate fixed point problem in the continuous setting [18].

The algorithm for Brouwer is reminiscent of the constant rounds algorithm for local search. We divide the space in sub-cubes then find the one guaranteed to have a solution by checking a boundary condition given in [18]. Then a parity argument will show there is always a sub-cube satisfying the boundary condition. In the last round, the algorithm queries all the points in the remaining sub-cube and returns the solution.

The lower bound for Brouwer is obtained by reducing local search instances generated by staircases to discrete fixed-point instances. We can naturally let the staircase within the local search instance to be a long path in discrete fixed-point problem. See Appendix E for details.

7. Discussion

A natural direction for future research is analyzing general graphs. In the fully adaptive setting, prior work has successfully established bounds for local search on general graphs by relating query complexity to structural properties such as expansion, separation number (which is equivalent to the treewidth), and vertex congestion. Determining the relationship between these graph parameters and query complexity in rounds remains an interesting open question.

A. Algorithm for local search in constant rounds

In this section we give an algorithm for local search on the d-dimensional grid in constant rounds, which will imply the upper bound of Theorem 3.1.

Notation. A d-dimensional *cube* is a Cartesian product of d connected (integer) intervals. We use *cube* to indicate d-dimensional *cube* for brevity, unless otherwise specified. The boundary of cube C is defined as all the points $\mathbf{x} \in C$ with fewer than $2d$ neighbors in C.

Proof of upper bound of Theorem 3.1. Given the d-dimensional grid $[n]^d$, consider a sequence of cubes contained in each other: $C_0 \supset C_1 \supset C_2 \supset \cdots \supset C_{k-1}$, where $C_0 = [n]^d$ is the whole grid.

For each $0 \leq i < k$, set $\ell_i = n^{(d^k - d^i)/(d^k - 1)}$ as the side length of cube C_i. The values of ℓ_i are chosen for balancing the number of queries in each round, which will be proved later. Note ℓ_i is an integer divisor of n. Consider Algorithm 1, which is illustrated in Figure 4.

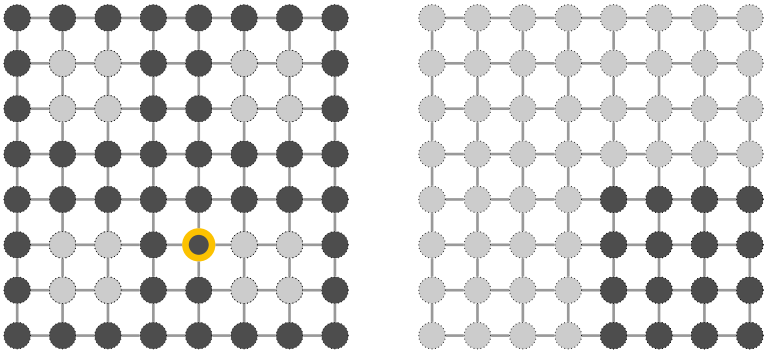
Algorithm 1 (**Local search in constant rounds**).

(1) Initialize the current cube to C_0.

(2) In each round $i \in \{1, \ldots, k-1\}$:

- Divide the current cube C_{i-1} into a set of mutually exclusive sub-cubes $C_i^1, \ldots, C_i^{n_i}$ of side length ℓ_i that cover C_{i-1}. Note that $n_i = (\ell_{i-1}/\ell_i)^d$ by definition.
- Query all the points on the boundary of sub-cubes $C_i^1, \ldots, C_i^{n_i}$. Let $\mathbf{x}_i^*$ be the point with minimal value among them.
- Set $C_i = C_i^j$, where C_i^j is the sub-cube that $\mathbf{x}_i^*$ belongs to.

(3) In round k, query all the points in the current cube C_{k-1} and find the solution point.



(a) Round 1 **(b)** Round 2

Figure 4. 2-dimensional grid of size 8×8. Suppose there are two rounds. In round 1 the algorithm queries all the black points and selects the minimum among all these points (illustrated with a yellow boundary). In round 2, it queries the entire sub-square in which the minimum from the first round was found.

To argue that the algorithm finds a local minimum, note that in each round $i \leq k-1$, the steepest descent starting from $\mathbf{x}_i^*$ will never leave the sub-cube C_i, since if it did it would have to exit C_i through a point of even smaller value than $\mathbf{x}_i^*$, which contradicts the definition of $\mathbf{x}_i^*$. Thus there must exist a local optimum within C_i.

Now we calculate the number of queries in each round. In round $i \in \{1, \ldots, k-1\}$, the number of points on the boundary of all sub-cubes $S_1, \ldots, S_{n_i}$ is

$$n_i \cdot (2d \cdot {\ell_i}^{d-1}) = \left(\frac{\ell_{i-1}}{\ell_i}\right)^d \cdot 2d \cdot {\ell_i}^{d-1} = 2d \cdot \ell_{i-1}^d \cdot \left(\frac{1}{\ell_i}\right)$$
$$= 2d \cdot n^{(d\cdot(d^k-d^{i-1})-(d^k-d^i))/(d^k-1)} = 2d \cdot n^{(d^{k+1}-d^k)/(d^k-1)}.$$

The number of queries in round k is

$$(\ell_{k-1})^d = n^{(d^{k+1}-d^k)/(d^k-1)}.$$

Since k and d are constants, the algorithm makes $O(n^{(d^{k+1}-d^k)/(d^k-1)})$ queries in total as required. ∎

B. Algorithm for local search in polynomial rounds

In this section we prove the upper bound of Theorem 3.2. Thus we show the following upper bounds on the query complexity of local search in polynomial rounds:

- $O(n^{(d-1)-(d-2)/d\alpha})$ when $d \geq 5$,
- $O(n^{3-\alpha/2})$ when $d = 4$, and
- $O(n^{2-\alpha/3})$ when $d = 3$.

The algorithm for polynomial number of rounds was described in Section 5.2. Here we give the pseudocode with precise definitions and the proof of correctness.

Notation. Let the number of rounds be $k = n^\alpha$, where $\alpha \in (0, d/2)$ is a constant. Given a point $\mathbf{x}$ on the grid, let $\text{rank}(\mathbf{x})$ be the number of grid points with smaller value than point $\mathbf{x}$. Let $C(\mathbf{x}, s) := \{\mathbf{y} \in [n]^d : \|\mathbf{y} - \mathbf{x}\|_\infty \leq s\}$ be the set of grid points in the d-dimensional cube of side length $2 \cdot s$, centered at point $\mathbf{x}$. We say a point $\mathbf{x}$ is the minimum in a set S if $\mathbf{x} \in S$ and $f(\mathbf{x}) \leq f(\mathbf{y})$ for all $\mathbf{y} \in S$.

There are multiple query requests from different procedures in one round. These queries will first be collected together and then submitted at once at the end of each round.

Algorithm 2 (**Local search in polynomial number of rounds**).

Input. Size of the instance n, dimension d, round limit $k = n^\alpha$, value function f. These are global parameters accessible from any subroutine.

Output. Local minimum $\mathbf{x}^*$ in $[n]^d$.

(1) Set $\beta \leftarrow (d-1) - \alpha((d-2)/d)$; $h \leftarrow \lfloor 1/\alpha + (d-2)/d \rfloor + 1$; $\widetilde{k} \leftarrow k/h$.

(2) Query $Q_1 = 100 \cdot h \cdot n^\beta$ points chosen u.a.r. in round 1 and set $\mathbf{x}_1$ to the minimum of these.

(3) Set $s \leftarrow (1/h) \cdot n^{1+\alpha(d-2)/d}$.

(4) Return $\texttt{FLSD}(s, h, \mathbf{x}_1, 2)$.

Procedure (**Fractal-Like Steepest Descent (FLSD)**).

Input. Size s, depth D, grid point $\mathbf{x}$, round r.

Output. Point $\mathbf{x}^* \in C(\mathbf{x}, s)$ with $\text{rank}(\mathbf{x}^*) \leq \text{rank}(\mathbf{x}) - s$; if in the process of searching for such a point it finds a local minimum, then it outputs it and halts everything.

(1) Set $\mathbf{x}_0 \leftarrow \mathbf{x}$; step $\leftarrow s/\widetilde{k}$. // *executed in round r*

(2) If $D = 1$, then: // *make s steps of steepest descent, since s is small enough when* $D = 1$

  (a) For $i = 0$ to $s - 1$: // *executed in rounds* $r + i$ *to* $r + i + 1$

    (i) Query all the neighbors of $\mathbf{x}_i$; let $\mathbf{x}_{i+1}$ be the minimum among them.

(ii) If $\mathbf{x}_i = \mathbf{x}_{i+1}$, then: // *thus* $\mathbf{x}_i$ *is a local min*
Output $\mathbf{x}^* \leftarrow \mathbf{x}_i$ and halt all running FLSD and DACS calls.

(b) Return $\mathbf{x}^* \leftarrow \mathbf{x}_s$. // *executed in round* $r + s$

(3) For $i = 0$ to $\widetilde{k} - 1$: // *divide the whole task into* $\widetilde{k}$ *pieces; executed in rounds* $r + i$ *to* $r + i + 1$

(a) $\mathbf{y}_{i+1} \leftarrow \texttt{FLSD}(\text{step}, D - 1, \mathbf{x}_i, r + i)$. // *execute call in parallel with current procedure*

(b) Query the *boundary* of $C(\mathbf{x}_i, \text{step})$ to find the point $\mathbf{x}_{i+1}$ with minimum value on it. // *making a "giant step" of size step by cheap substitute* $\mathbf{x}_i$

(4) For $i = 0$ to $\widetilde{k} - 1$: // *check if each giant step does make giant progress by using the feedback from sub-procedures; executed in round* $r + i + (D - 1) \cdot \widetilde{k}$, *after* y_{i+1} *was received in Step* (3a)

(a) If $f(\mathbf{y}_{i+1}) < f(\mathbf{x}_{i+1})$ then: // *a solution exists in* $C(\mathbf{x}_i, \text{step})$, *call* DACS *to find it*

(i) Set $\mathbf{x}^* \leftarrow \texttt{DACS}(C(\mathbf{x}_i, \text{step}))$. // *stop and wait for the result of* DACS

(ii) Output $\mathbf{x}^*$ and halt all running FLSD and DACS calls.

(5) Return $\mathbf{x}^* \leftarrow \mathbf{x}_{\widetilde{k}}$. // *executed in round* $r + D \cdot \widetilde{k}$

Procedure (**Procedure Divide-and-Conquer Search (DACS)**).

Input. Cube C_0.

Output. Local minimum $\mathbf{x}^*$ in C_0.

(1) Set $\mathbf{x}^* \leftarrow$ *Null*.

(2) For $i = 1$ to $\lceil \log_2 n \rceil$:

(a) If C_{i-1} contains only one point $\widetilde{\mathbf{x}}$, then:

(i) Set $\mathbf{x}^* = \widetilde{\mathbf{x}}$; break.

(b) Partition C_{i-1} into 2^d disjoint sub-cubes $C_i^1, \ldots, C_i^{2^d}$, each with side length half that of C_{i-1}.

(c) Query all the points on the boundary of each sub-cube $C_i^1, \ldots, C_i^{2^d}$.

(d) Let $\mathbf{x}_i^*$ be the point with minimum value among *all points queried* in C_{i-1}, including queries made by Algorithm 2 and all FLSD calls. // *break ties lexicographically*

(e) Let $C_i \in \{C_i^1, \ldots, C_i^{2^d}\}$ be the unique sub-cube with $\mathbf{x}_i^* \in C_i$.

(3) Return $\mathbf{x}^*$.

Analysis. We first establish that Algorithm 2 is correct.

Lemma B.1. *If the procedure* $\texttt{FLSD}(s, D, \mathbf{x}, r)$ *does return at Step* (2b) *or Step* (5)*, it will return within* $\widetilde{k} \cdot D$ *rounds after the start round* r*; otherwise, the procedure* $\texttt{FLSD}(s, D, \mathbf{x}, r)$ *will halt within at most* $\widetilde{k} \cdot D + \lceil \log n \rceil$ *rounds after the start round* r.[6]

Proof. We proceed by induction on the depth D. The base case is when $D = 1$. By the definition of $\widetilde{k}$ and h, we have

$$\left(\frac{1}{\widetilde{k}^h}\right) \cdot n^{1+((d-2)/d)\alpha} < 1.$$

Also notice that the parameter s will be divided by $\widetilde{k}$ when D decreases by one, so when $D = 1$, the current size s will be at most $\widetilde{k}$, i.e., the steepest descent will return in $\widetilde{k}$ rounds. Assume it holds for $1, \ldots, D-1$.

For any $D > 1$, all the queries made by the procedure itself need $\widetilde{k}$ rounds and each sub-procedure will take at most $\widetilde{k} \cdot (D-1)$ rounds by the induction hypothesis. Since all the procedures are independent of each other and could be executed in parallel, the total number of rounds needed for this procedure is $\widetilde{k} \cdot D$. The first part of the lemma thus follows by induction.

Finally, recall that the divide-and-conquer procedure $\texttt{DACS}$ takes $\lceil \log n \rceil$ rounds, so the procedure will halt within $\widetilde{k} \cdot D + \lceil \log n \rceil$ rounds. ∎

Lemma B.2. *If the procedure* $\texttt{FLSD}(s, D, \mathbf{x}, r)$ *does return a point* $\mathbf{x}^*$ *at Step* (2b) *or Step* (5)*, then* $\mathbf{x}^*$ *is in the cube* $C(\mathbf{x}, s)$ *and satisfy the inequality*

$$\text{rank}(\mathbf{x}^*) \leq \text{rank}(\mathbf{x}) - s.$$

Proof. We proceed by induction on the depth D. The base case is when $D = 1$. Then we know that $s \leq \widetilde{k}$ by the same argument in Lemma B.1, thus s steps of steepest descent will ensure that $\mathbf{x}^* \in C(\mathbf{x}, s)$ and $\text{rank}(\mathbf{x}^*) \leq \text{rank}(\mathbf{x}) - s$. Assume it holds for $1, \ldots, D-1$ and show for D.

For any $D > 1$, by Step (4a) we have

$$\text{rank}(\mathbf{x}_i) \leq \text{rank}(\mathbf{y}_i)$$

for any $1 \leq i \leq \widetilde{k}$; by the induction hypothesis, we have

$$\text{rank}(\mathbf{y}_i) \leq \text{rank}(\mathbf{x}_{i-1}) - \text{step}$$

[6] All of the procedures $\texttt{FLSD}$ considered in the following analysis are initiated during the execution of Algorithm 2. Thus Lemma B.1, Lemma B.2 and Lemma B.5 may not work for $\texttt{FLSD}$ with arbitrary parameters.

for any $1 \leq i \leq \widetilde{k}$. Combining them, we get

$$\text{rank}(\mathbf{x}_i) \leq \text{rank}(\mathbf{x}_{i-1}) - \text{step}$$

for any $1 \leq i \leq \widetilde{k}$. Thus

$$\text{rank}(\mathbf{x}^*) = \text{rank}(\mathbf{x}_{\widetilde{k}}) \leq \text{rank}(\mathbf{x}_0) - \widetilde{k} \cdot \text{step} = \text{rank}(\mathbf{x}) - s.$$

Also notice that the L_∞ distance from $\mathbf{x}^*$ to $\mathbf{x}$ is at most $\widetilde{k} \cdot \text{step} = s$, i.e., $\mathbf{x}^* \in C(\mathbf{x}, s)$. This concludes the proof of the lemma for any depth D. ∎

Lemma B.3. *The point* $\mathbf{x}^*$ *returned by* `FLSD` *at Step* (i) *is a local minimum.*

Proof. We use notation $\mathcal{D}.\mathbf{x}$ to denote the variable $\mathbf{x}$ in the procedure `DACS` and $\mathcal{F}.\mathbf{x}$ to denote the variable $\mathbf{x}$ in the procedure `FLSD`, which calls the procedure `DACS`.

Recall $\mathcal{D}.\mathbf{x}_i^*$ is the point with minimum value among *all points queried* in the sub-cube C_{i-1}, including queries made by the entire algorithm (Algorithm 2) and all `FLSD` calls. By Step (4a), we have

$$f(\mathcal{F}.\mathbf{y}_{i+1}) < f(\mathcal{F}.\mathbf{x}_{i+1}).$$

Then for each $\mathcal{D}.\mathbf{x}_i^*$ and index i, we have the following inequality by definition of $\mathcal{D}.x_i^*$:

$$f(\mathcal{D}.\mathbf{x}_i^*) \leq f(\mathcal{F}.\mathbf{y}_{i+1}) < f(\mathcal{F}.\mathbf{x}_{i+1}).$$

Therefore, the steepest descent from $\mathcal{D}.\mathbf{x}_i^*$ will never leave the cube $\mathcal{D}.C_i$, especially the cube $\mathcal{D}.C_0$.

As noted above, $\mathcal{D}.\mathbf{x}_i^*$ is taken as the minimum of *all* queried points in $\mathcal{D}.C_{i-1}$, not just for points on the boundary. This makes sure that the point $\mathcal{F}.\mathbf{y}_{i+1}$ is taken into account, thus any point with smaller or equal value than $\mathcal{F}.\mathbf{y}_{i+1}$ could not escape from $\mathcal{D}.C_0$ by steepest descent; otherwise, the point $\mathcal{F}.\mathbf{x}_{i+1}$ itself may be the minimum during all queries on the boundaries, but it could possibly escape the cube $\mathcal{D}.C_0$ by steepest descent, and cause the procedure `DACS` return a non-local optimum.

Finally, let $\widetilde{C}$ be the cube that consists only of the point $\mathcal{D}.\tilde{\mathbf{x}}$ in the `DACS` procedure. By previous argument, the steepest descent from $\mathcal{D}.\mathbf{x}^* = \mathcal{D}.\tilde{\mathbf{x}}$ does not leave the cube $\widetilde{C}$, which means that $\mathcal{D}.\mathbf{x}^*$ is a local optimum. ∎

Lemma B.4. *Algorithm 2 outputs the correct answer with probability at least* $9/10$.

Proof. The point $\mathbf{x}^*$ output at Step (ii) is always a local optimum. By Lemma B.3, the point $\mathbf{x}^*$ output at Step (ii) is also a local optimum. Thus, we only need to argue that Algorithm 2 will output the solution and halt with probability at least $9/10$.

Recall that Algorithm 2 queries $Q_1 = 100h \cdot n^\beta$ random points in the first round. We have

$$\frac{n^d}{Q_1} = \frac{1}{100h} n^{1+((d-2)/d)\alpha}.$$

Then, by definition,

$$\begin{aligned}\Pr\left[\operatorname{rank}(\mathbf{x}_1) > \frac{1}{2h} \cdot n^{1+((d-2)/d)\alpha}\right] &= \left(1 - \frac{1/2h \cdot n^{1+((d-2)/d)\alpha}}{n^d}\right)^{Q_1} \\ &= \left(1 - \frac{50}{Q_1}\right)^{Q_1} \le \frac{1}{10}.\end{aligned}$$

In other words, with probability at least $9/10$, we have

$$\operatorname{rank}(\mathbf{x}_1) \le \frac{1}{2h} \cdot n^{1+((d-2)/d)\alpha}. \tag{B.1}$$

If (B.1) holds, then the procedure $\texttt{FLSD}(1/h \cdot n^{1+((d-2)/d)\alpha}, h, \mathbf{x}_1, 2)$ should halt within a number of rounds at most

$$T = \widetilde{k} \cdot \left((h-1) + \frac{2}{3}\right) + \lceil \log n \rceil < \widetilde{k} \cdot h = k.$$

Otherwise, let $t = \lfloor 2\widetilde{k}/3 \rfloor$. By Lemma B.1, we know that y_t is already available from Step (3a) by round T. Then by Lemma B.2, we have

$$\operatorname{rank}(\mathbf{y}_t) < \left(\frac{1}{2} - \frac{2}{3}\right) \cdot \left(\frac{1}{h} n^{1+((d-2)/d)\alpha}\right) < 0,$$

which is impossible. Thus, the call $\texttt{FLSD}(1/h \cdot n^{1+((d-2)/d)\alpha}, h, \mathbf{x}_1, 2)$ must halt within T rounds in this case, which completes the argument. ∎

Now consider the total number of queries made by Algorithm 2.

Lemma B.5. *A call of procedure* $\texttt{FLSD}(s, D, \mathbf{x}, r)$ *will make* $O(\lceil s/\widetilde{k} \rceil^{d-1} \cdot \widetilde{k})$ *queries, including the queries made by its sub-procedure.*

Proof. We proceed by induction on the depth D. The base case is when $D = 1$. In this case, the procedure FLSD performs s steps of steepest descent, where $s \le \widetilde{k}$. Thus it will make $s \cdot 2d = O(\widetilde{k})$ queries.

For any depth $D > 1$,

- the number of queries made by Step (3b) is at most

$$2d \cdot \left(2\left\lceil \frac{s}{\widetilde{k}} \right\rceil + 1\right)^{d-1} \cdot \widetilde{k} = O\left(\left\lceil \frac{s}{\widetilde{k}} \right\rceil^{d-1} \cdot \widetilde{k}\right);$$

- the number of queries made by all sub-procedures is bounded as follows by the induction hypothesis:

$$\widetilde{k} \cdot O\Big(\Big\lceil \frac{s}{\widetilde{k}^2} \Big\rceil^{d-1} \cdot \widetilde{k}\Big) = O\Big(\Big\lceil \frac{s}{\widetilde{k}} \Big\rceil^{d-1} \cdot \widetilde{k}\Big);$$

- the number of queries made by DACS is at most

$$\Big(2\Big\lceil \frac{s}{\widetilde{k}} \Big\rceil + 1\Big)^{d-1} \cdot 2d \cdot 2 = O\Big(\Big\lceil \frac{s}{\widetilde{k}} \Big\rceil^{d-1}\Big).$$

Thus the total number of queries is $O(\lceil s/\widetilde{k} \rceil^{d-1} \cdot \widetilde{k})$, which concludes the proof for all D. ∎

Proof of upper bound of Theorem 3.2. The correctness of Algorithm 2 is established by Lemma B.4. By Lemma B.5, the total number of queries issued by Algorithm 2 is

$$100h \cdot n^{(d-1)-((d-2)/d)\alpha} + O\Big(\big(n^{1-2\alpha/d}\big)^{d-1} \cdot \Big(\frac{n^\alpha}{h}\Big)\Big) = O\big(n^{(d-1)-((d-2)/d)\alpha}\big).$$

This concludes the proof of the upper bound part in Theorem 3.2. ∎

C. Randomized lower bound for local search in constant rounds

In this section, we show the lower bound for local search in constant number of rounds. We start with a few definitions.

Notation and definitions. Recall that $\ell_i = n^{(d^k-d^i)/(d^k-1)}$, $0 \le i < k$. Let

$$m := \sum_{0 \le i < k} \ell_i \le 2n.$$

We now consider the grid of side length m in this subsection for technical convenience.

For a point $\mathbf{x} = (x_1, \ldots, x_d)$, let $W_i(\mathbf{x})$ be the grid points that are in the cube region of size $(\ell_i)^d$ with corner point $\mathbf{x}$:

$$W_i(\mathbf{x}) = \big\{\mathbf{y} = (y_1, \ldots, y_d) \in [m]^d : x_j \le y_j < x_j + \ell_i, \forall\, j \in [d]\big\}.$$

Next we define a basic structure called *folded-segment*. Intuitively, the folded-segment connecting two points $\mathbf{x}, \mathbf{y}$ is the following path of points: Starting at point $\mathbf{x}$, we initially change the first coordinate of $\mathbf{x}$ towards the first coordinate of $\mathbf{y}$. Then we change the second coordinate, the third coordinate, and so on, until finally reaching the point $\mathbf{y}$. The formal definition is as follows.

Definition C.1 (Folded-segment). For any two points $\mathbf{x}, \mathbf{y} \in [m]^d$, the *folded-segment* $\mathrm{FS}(\mathbf{x}, \mathbf{y})$ is a set of points connecting $\mathbf{x}$ and $\mathbf{y}$, defined as follows.

Let $\mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{y} = (y_1, \ldots, y_d)$. For any $1 \le i \le d$, define point set

$$E_i(\mathbf{x}, \mathbf{y}) := \big\{\mathbf{z} = (y_1, \ldots, y_{i-1}, z_i, x_{i+1}, \ldots, x_d) \in [m]^d : \min(x_i, y_i) \le z_i \le \max(x_i, y_i)\big\}.$$

Then define

$$\mathrm{FS}(\mathbf{x}, \mathbf{y}) = \left(\bigcup_{i \in [d]} E_i(\mathbf{x}, \mathbf{y}) \right) \setminus \{\mathbf{x}\}.$$

Staircase structure. A staircase $s^{(i)}(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_i)$ of length i, $0 \le i \le k$, consists of a path of points defined by an array of $i+1$ "connecting" points $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_i \in [m]^d$. We call $\mathbf{x}_0$ as the start point and $\mathbf{x}_i$ as the end point. For each $0 \le j < i$, the pair of consecutive connecting points $(\mathbf{x}_j, \mathbf{x}_{j+1})$ is connected by a folded-segment $\mathrm{FS}(\mathbf{x}_j, \mathbf{x}_{j+1})$. We denote the j-th $(0 \le j \le i)$ connecting point of $s^{(i)}$ by $\mathbf{x}_j$, and the j-th $(1 \le j \le i)$ folded-segment of $s^{(i)}$ as $\mathrm{FS}(\mathbf{x}_{j-1}, \mathbf{x}_j)$.

The probability distribution for staircases of length i, $0 \le i \le k$, is as follows. The start point $\mathbf{x}_0$ is always set to be the corner point $\mathbf{1}$; points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i$ are picked in turn: the point $\mathbf{x}_j$ is chosen uniformly at random from the sub-cube $W_{j-1}(\mathbf{x}_{j-1})$.[7] Two staircases are different if they have different set of connecting points, even if the paths defined by the two sets of connecting points are the same. Thus, $L^{(i)} := \Pi_{j=0}^{i-1} {\ell_j}^d$ is the number of all possible length i staircases, and each staircase will be selected with the same probability $1/L^{(i)}$.

A staircase s of length i *grows* from a staircase s' of length $(i-1)$ if the first $i-1$ folded-segment of s and s' are same. A prefix of staircase s is any staircase formed by a prefix of the connecting points sequence of s. To simplify the analysis, we assume the algorithm is given the location of $\mathbf{x}_i$ after round i, except for the round k; this only strengthens the lower bound.

Value function. After fixing a staircase s, we can define the value function f^s corresponding to s. Within the staircase, the value of the point is *minus* the distance to $\mathbf{x}_0$ by following the staircase backward, except the end point $\mathbf{x}_i$ of the staircase. The value of any point outside is the L_1 distance to $\mathbf{x}_0$.[8] The value of the end point $\mathbf{x}_i$

[7] Recall that we take the side length of the grid as m rather than n. Thus the staircase will not be clipped out by the boundary of grid.

[8] Though the distance to $\mathbf{x}_0$ by following the staircase backward is same as the L_1 distance, since all staircases constructed above only *grow* non-decreasingly at each coordinate. But the staircases we constructed in the next subsection for the lower bound of polynomial rounds algorithm does not have such properties, and thus the distance by following the staircase backward is not same as the L_1 distance in that case.

is set as minus the distance to the point $\mathbf{x}_0$ by following the staircase backward with probability $1/2$, and the L_1 distance to $\mathbf{x}_0$ with probability $1/2$. Thus the end point of the staircase must be queried, otherwise the algorithm will incorrectly guess the *location* of the unique solution point with probability at least $1/2$.

This value function makes sure that for any two different staircases s_1 and s_2, the functions f^{s_1} and f^{s_2} have the same value on the common prefix *and* on any point outside of both s_1 and s_2. Also notice that f^s is deterministic on every point except the end point $\mathbf{x}_i$ of staircase s.

Good staircases. Next we introduce the concept of *good staircases*, on which the algorithm could only *learn* one more connecting point in each round.

Definition C.2 (Good staircase). A staircase $s^{(i)}(\mathbf{x}_0, \ldots, \mathbf{x}_i)$ of length i is *good* with respect to a deterministic algorithm $\mathcal{A}$ that runs in k-rounds, if the following condition is met:

- When $\mathcal{A}$ is running on the value function $f^{s^{(i)}}$, for any $0 < j < i$, any point $\mathbf{x} \in FS(\mathbf{x}_j, \mathbf{x}_{j+1})$ is *not* queried by algorithm $\mathcal{A}$ in rounds $1, \ldots, j$.

We use good staircase to indicate the good staircase *with respect to a fixed deterministic algorithm* $\mathcal{A}$ for brevity. Good staircases have the following properties.

Lemma C.3. (1) *If s is a good staircase, then any prefix s' of s is also a good staircase.*

(2) *Let s_1, s_2 be any two good staircases. If the first $i+1$ connecting points of the staircases are the same, then $\mathcal{A}$ will receive the same replies in rounds $1, \ldots, i$ and issue the same queries in rounds $1, \ldots, i+1$ while running on both f^{s_1} and f^{s_2}.*

Proof. We first prove part (2). By the definition of good staircase, in round $1, \ldots, i$, algorithm $\mathcal{A}$ will not query any point on s_1 or s_2 that is after the first $i+1$ connecting points, where f^{s_1} and f^{s_2} may have different value. Since $\mathcal{A}$ is deterministic, by induction from round 1, we get that $\mathcal{A}$ will issue the same queries and receive the same replies in rounds $1, \ldots, i$. The queries in round $i+1$ are also same because they only depend on the replies in round $1, \ldots, i$. The proof of part (1) is similar by taking $s_1 = s$ and $s_2 = s'$. ∎

If most of the possible staircases are good staircases, then $\mathcal{A}$ will fail on a constant fraction of the inputs generated by length k staircases.

Lemma C.4. *If the algorithm $\mathcal{A}$ issues at most $\mathcal{Q}_k = 1/10 \cdot n^{(d^{k+1}-d^k)/(d^k-1)}$ queries in each round, and $9/10$ of all possible length k staircases are good , then $\mathcal{A}$ will fail to get the correct solution with probability at least $7/40$.*

Proof. If $s^{(k)}(\mathbf{x}_0, \ldots, \mathbf{x}_{k-1}, \mathbf{x}_k)$ is good, then $\mathcal{A}$ could not distinguish it from another good staircase $s'(\mathbf{x}_0, \ldots, \mathbf{x}_{k-1}, \mathbf{x}'_k)$ with same first k connecting points before the last round by Lemma C.3.

Recall that before the last round, the algorithm is given the value of $\mathbf{x}_{k-1}$. Let $z_{s^{(k-1)}}$ be the fraction of length k good staircases among all length k staircases growing from the length $(k-1)$ staircase $s^{(k-1)}$. Then define the random variable Z such that $Z = z_{s^{(k-1)}}$ if the length $(k-1)$ staircase $s^{(k-1)}$ is chosen. Since each length $(k-1)$ staircase is selected with the same probability, the expectation of Z is $9/10$.

Thus by Markov inequality on $(1-Z)$, a half of all the length $(k-1)$ staircases $s^{(k-1)}$ satisfy $z_{s^{(k-1)}} \geq 8/10$. We say a length $(k-1)$ staircase $s^{(k-1)}$ is *nice* if $z_{s^{(k-1)}} \geq 8/10$.

Recall there are $(\ell_{k-1})^d = n^{(d^{k+1}-d^k)/(d^k-1)}$ staircases of length k, growing from a staircase of length $(k-1)$.

However, the algorithm can only query $1/10 \cdot n^{(d^{k+1}-d^k)/(d^k-1)}$ points in the last round. Thus if $s^{(k)}$ is a good staircase growing from a length $k-1$ nice staircase $s^{(k-1)}$, then $\mathcal{A}$ will not query the endpoint $\mathbf{x}_k$ of $s^{(k)}$ with probability at least $7/8$. We define the following events:

$$\begin{aligned}
\text{Fail} &= \{\mathcal{A} \text{ makes mistake}\},\\
\text{Hit} &= \{\mathcal{A} \text{ never queries the end point } \mathbf{x}_k \text{ during the execution}\},\\
\text{Nice} &= \{\text{the length } k-1 \text{ prefix } s^{(k-1)} \text{ of } s^{(k)} \text{ is nice}\},\\
\text{Good} &= \{\text{the whole length } k \text{ staircase } s^{(k)} \text{ is good}\}.
\end{aligned}$$

Thus $\mathcal{A}$ will make a mistake with probability at least

$$\begin{aligned}
\Pr[\text{Fail}] &\geq \frac{1}{2} \cdot \Pr[\text{Hit}] \geq \frac{1}{2} \cdot \Pr[\text{Hit} \mid \text{Nice} \wedge \text{Good}] \cdot \Pr[\text{Nice} \wedge \text{Good}]\\
&\geq \frac{1}{2} \cdot \frac{7}{8} \cdot \Pr[\text{Good} \mid \text{Nice}] \cdot \Pr[\text{Nice}] \geq \frac{1}{2} \cdot \frac{7}{8} \cdot \frac{8}{10} \cdot \frac{1}{2} = \frac{7}{40},
\end{aligned}$$

where the probability is taken on the staircase $s^{(k)}$ and the value of $\mathbf{x}_k$. ∎

Counting the number of good staircases. Counting the number of good staircases is the major technical challenge in the proof. The concept of *probability score function* is useful.

Definition C.5 (Probability score function). For a fixed deterministic algorithm $\mathcal{A}$, let $Q_{\mathcal{A}}$ be the set of points queried by $\mathcal{A}$ during its execution. Given a point $\mathbf{x} \in [m]^d$, for any $0 \leq i < k$, define the set of points

$$W_i(\mathbf{x}, Q_{\mathcal{A}}) := \{\mathbf{y} \in W_i(\mathbf{x}) : FS(\mathbf{x}, \mathbf{y}) \cap Q_{\mathcal{A}} = \emptyset\}.$$

The *probability score function* for the point $\mathbf{x}$ is $\mathrm{SC}_i(\mathbf{x}, Q_{\mathcal{A}}) := |W_i(\mathbf{x}, Q_{\mathcal{A}})|/{\ell_i}^d$.

The probability score function for a good staircases $s^{(i)}(\mathbf{x}_0, \ldots, \mathbf{x}_i)$ of length i, for $0 \le i < k$, is defined as $\mathrm{SC}(s^{(i)}) := \mathrm{SC}_i(\mathbf{x}_i, Q_{\mathcal{A}})$, where $Q_{\mathcal{A}}$ is the set of points that have been queried by $\mathcal{A}$ after the round i, if $\mathcal{A}$ is executed on value function $f^{s^{(i)}}$.[9]

Informally, if $Q_{\mathcal{A}}$ is the set of points queried by $\mathcal{A}$ in the first i rounds, the probability score function $\mathrm{SC}(s^{(i)})$ can be interpreted as the probability that the length $i+1$ staircase $s^{(i+1)}$ being good if it grows from the good prefix $s^{(i)}$.

Let $\mathbf{x} \in [m]^d$, $\mathbf{y} \in Q_{\mathcal{A}}$. Let $B_i(\mathbf{x}, \mathbf{y}) := \{\mathbf{z} \in W_i(\mathbf{x}) : \mathbf{y} \in FS(\mathbf{x}, \mathbf{z})\}$. Thus $|B_i(\mathbf{x}, \mathbf{y})|$ is the number of *folded-segments* $FS(\mathbf{x}, \mathbf{z}), \mathbf{z} \in W_i(\mathbf{x})$ that are intersected by a point $\mathbf{y} \in Q_{\mathcal{A}}$. We then define the *cost* incurred by a point $\mathbf{y} \in Q_{\mathcal{A}}$ on the probability score function of a point $\mathbf{x}$ for any $0 \le i < k$ as

$$\mathrm{cost}_i(\mathbf{x}, \mathbf{y}) := \frac{|B_i(\mathbf{x}, \mathbf{y})|}{{\ell_i}^d}.$$

The merit of our random staircase structure is that any single queried point will not hit too many of staircases. This property is quantitatively characterized by the following lemma via the *cost*.

Lemma C.6. *The* total cost *incurred by one point* $\mathbf{y} \in Q_{\mathcal{A}}$ *for all* $\mathbf{x} \in [m]^d$*, i.e.,* $\sum_{\mathbf{x}} |B_i(\mathbf{x}, \mathbf{y})|/({\ell_i}^d)$ *is at most* $d \cdot \ell_i$.

Proof. If $\mathbf{y} \notin W_i(\mathbf{x})$, we simply have $\mathrm{cost}_i(\mathbf{x}, \mathbf{y}) = 0$. Therefore, we only need to estimate $\mathrm{cost}_i(\mathbf{x}, \mathbf{y})$ or equivalently, $|B_i(\mathbf{x}, \mathbf{y})|$ for pairs $\mathbf{x}, \mathbf{y}$ such that $\mathbf{y} \in W_i(\mathbf{x})$.

For two points $\mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{y} = (y_1, \ldots, y_d)$, let $t(\mathbf{x}, \mathbf{y}) \in [d]$ denote the smallest index such that for any j satisfying $t(\mathbf{x}, \mathbf{y}) < j \le d$, there is $x_j = y_j$. Then, by the definition of *folded-segments*, we have

$$B_i(\mathbf{x}, \mathbf{y}) = \left\{(y_1, \ldots, y_{t(\mathbf{x},\mathbf{y})-1}, z_{t(\mathbf{x},\mathbf{y})}, z_{t(\mathbf{x},\mathbf{y})+1}, \ldots, z_d) \in W_i(\mathbf{x}) : z_{t(\mathbf{x},\mathbf{y})} \ge y_{t(\mathbf{x},\mathbf{y})}\right\}.$$

Therefore, we have $|B_i(\mathbf{x}, \mathbf{y})| \le {\ell_i}^{d-t(\mathbf{x},\mathbf{y})+1}$.

Finally, for a point $\mathbf{y}$, the number of points $\mathbf{x}$ such that $\mathbf{y} \in W_i(\mathbf{x})$ and $t(x, y) = j$ is at most ${\ell_i}^j$. Therefore, we have

$$\begin{aligned}\sum_{\mathbf{x}\in[m]^d} \frac{|B_i(\mathbf{x}, \mathbf{y})|}{{\ell_i}^d} &= \sum_{j\in[d]} \sum_{\mathbf{x}\in[m]^d} \frac{|B_i(\mathbf{x}, \mathbf{y})|}{{\ell_i}^d} \cdot \mathbf{1}[t(\mathbf{x}, \mathbf{y}) = j] \\ &\le \sum_{j\in[d]} \frac{{\ell_i}^j \cdot {\ell_i}^{d-j+1}}{{\ell_i}^d} = d \cdot \ell_i.\end{aligned}$$

■

[9] By the definition of good staircase, $\mathcal{A}$ will not query the end point of staircase $s^{(i)}$ in rounds $1, \ldots, i-1$. Since $\mathcal{A}$ is a deterministic algorithm and $f^{s^{(i)}}$ is deterministic except at the end point of $s^{(i)}$, the set $Q_{\mathcal{A}}$ here is uniquely defined.

We can now count the number of length k good staircases with the *two-stage analysis*, which is formally summarized in below.

Lemma C.7. *If the number of queries issued by algorithm $\mathcal{A}$ is at most*

$$\mathcal{Q}_k = \frac{1}{10d \cdot k} \cdot n^{(d^{k+1}-d^k)/(d^k-1)},$$

then $9/10$ *of all possible length k staircases are good with respect to $\mathcal{A}$.*

Proof. For $0 \le i \le k$, let $G^{(i)}$ denote the set of all good staircases of length i. Then $|G^{(i)}|$ is the number of all length i good staircases, and $R^{(i)} := |G^{(i)}|/L^{(i)}$ is the fraction of length i good staircases. In particular, $L^{(0)} = |G^{(0)}| = R^{(0)} = 1$.

Let us first fix a length i, $0 \le i < k-1$, good staircase $s^{(i)} \in G^{(i)}$, and denote the set of all good staircases growing from $s^{(i)}$ by $G(s^{(i)})$. Now consider the sum of the probability score function for every staircase in $G(s^{(i)})$. By Lemma C.3, algorithm $\mathcal{A}$ will make the *same* queries in rounds $1, \ldots, i+1$ when running on any instance $f^{s^{(i+1)}}$ generated by staircase $s^{(i+1)}$ in $G(s^{(i)})$. Thus, we can use Lemma C.6 and the union bound to upper bound the total cost to the sum of their probability score function:

$$\begin{aligned}\sum_{s^{(i+1)} \in G(s^{(i)})} \mathrm{SC}\big(s^{(i+1)}\big) &\ge {\ell_i}^d \cdot \mathrm{SC}\big(s^{(i)}\big) - \mathcal{Q}_k \cdot d\,\ell_{i+1} \\ &\ge {\ell_i}^d \cdot \Big(\mathrm{SC}\big(s^{(i)}\big) - \frac{1}{10k}\Big) \qquad \text{(C.1)}\end{aligned}$$

The second inequality comes from the fact that $\mathcal{Q}_k \cdot \ell_{i+1} = \ell_i^d/d \cdot (10k)$.

Next, let us consider the number of all length $i+2$ good staircases:

$$\begin{aligned}|G^{(i+2)}| &= \sum_{s^{(i+1)} \in G^{(i+1)}} \mathrm{SC}\big(s^{(i+1)}\big) \cdot {\ell_{i+1}}^d && \text{(C.2)} \\ &= \sum_{s^{(i)} \in G^{(i)}} \Bigg(\sum_{s^{(i+1)} \in G(s^{(i)})} \mathrm{SC}\big(s^{(i+1)}\big) \cdot {\ell_{i+1}}^d \Bigg) && \text{(C.3)} \\ &\ge \big({\ell_{i+1}}^d \cdot {\ell_i}^d\big) \cdot \sum_{s^{(i)} \in G^{(i)}} \Big(\mathrm{SC}\big(s^{(i)}\big) - \frac{1}{10k}\Big) && \text{(C.4)} \\ &= {\ell_{i+1}}^d \cdot \Big(|G^{(i+1)}| - {\ell_i}^d \cdot \frac{|G^{(i)}|}{10k}\Big) && \text{(C.5)}\end{aligned}$$

Equality (C.2) and equality (C.5) are from the definition of probability score function. Inequality (C.4) is from inequality (C.1). Dividing each side of (C.5) by $L^{(i+2)}$, we get

$$R^{(i+2)} \ge R^{(i+1)} - \frac{R^{(i)}}{10k} \ge R^{(i+1)} - \frac{1}{10k}.$$

We know that $R^{(1)} = 1$ by definition, so $R^{(k)} \ge 9/10$ as required. ∎

The correctness of the lower bound statement in Theorem 3.1 follows from Lemma C.4 and Lemma C.7.

D. Randomized lower bound for local search in polynomial rounds

In this section, we show the lower bound for local search in polynomial number of rounds.

D.1. Notation and definitions

Recall the number of rounds is $k = n^{\alpha}$, where $\alpha \in (0, d/2)$. Define $\ell = n^{1-(2/d)\cdot\alpha}$ and $m = n/\ell$, where the definition of m from Section C is overridden here.

Define $\mathcal{Q}_k$ to be the number of queries allowed for the k-round algorithm stated in Theorem 3.2:

$$\mathcal{Q}_k = \begin{cases} c_d \cdot n^{(d-1)-((d-2)/d)\alpha} & \text{if } d > 4, \\ (c_4/\log n) \cdot n^{3-(1/2)\alpha} & \text{if } d = 4, \\ c_3 \cdot n^{(d-1)-(2/3)\alpha} & \text{if } d = 3. \end{cases} \tag{D.1}$$

The value c_d above is a constant depending on dimension d.

For any point $x \in [n]$ in the 1-dimensional grid, define $W^1(x)$ as the set of points within ℓ steps of going right from x, and assuming 1 is the next point of n by wrapping around. Formally,

$$W^1(x) = \{y \in [n] : x < y \le x + \ell \text{ or } y \le x + \ell - n\}.$$

Similarly, for a point $\mathbf{x} = (x_1, \ldots, x_d)$ in the d-dimensional grid $[n]^d$, let $W(\mathbf{x})$ be the set of grid points that are in the cube region of side length ℓ with corner point $\mathbf{x}$, and wrapping around the boundary if exceeding. That is,

$$W(\mathbf{x}) = \{\mathbf{y} = (y_1, \ldots, y_d) \in [n]^d : y_j \in W^1(x_j), \forall j \in [d]\}.$$

For any index $i \in [d]$, define

$$\begin{aligned} W_i^{-1}(\mathbf{x}) &= \{\mathbf{y} \in [n]^d : \forall j \le i, x_j \in W^1(y_j); \forall j > i, y_j = x_j\}, \\ W_i^{b}(\mathbf{x}) &= \{\mathbf{y} \in [n]^d : \forall j < i, x_j \in W^1(y_j); \\ &\qquad \max\{n - \ell, x_i\} < y_i \le n; \forall j > i, y_j = x_j\}. \end{aligned}$$

Let

$$W^{-1}(\mathbf{x}) = \bigcup_{i \in [d]} W_i^{-1}(\mathbf{x}), \quad W^{b}(\mathbf{x}) = \bigcup_{i \in [d]} W_i^{b}(\mathbf{x}), \quad W^{r}(\mathbf{x}) = W^{-1}(\mathbf{x}) \cup W^{b}(\mathbf{x}).$$

When calculating the coordinate of points, we always keep wrapping around it into the range $[n]^d$ for convenience.

Staircase structure. Let $\mathbf{x}_0 = \mathbf{1}$ be the corner point. The remaining connecting points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i$ are selected in turn, where $\mathbf{x}_j$ is chosen uniformly from

(1) the set $W(\mathbf{x}_{j-1})$ if $(j \mod m) \neq 0$;

(2) the set $[n]^d$ otherwise.

We still use the folded-segment as in Definition C.1 to link every two consecutive connecting points. The length of a staircase is the number of folded-segments in it. Note that the folded-segments in a staircase can now intersect; the value function on those intersecting points has to be handled with caution.

Let $S^{(i)}$ be the set of all possible staircases of length i. Let $S^{(i)}(s_j)$ be the subset of $S^{(i)}$ in which every staircase has staircase s_j as prefix. Every possible staircase of length i occurs with the same probability and their total number is $L^{(i)} = |S^{(i)}|$. Each staircase s_i of length i has the same number of length j staircases growing from it, namely, $|S^{(i)}(s_j)| = L^{(i)}/L^{(j)}$.

Let $p(\mathbf{x}, \mathbf{y}, i, t)$ to be the probability that point $\mathbf{y}$ is on the $(i + t - 1)$-th folded-segment, conditioned on point $\mathbf{x}$ being the i-th connecting point.[10] Similarly, define $q(\mathbf{x}, \mathbf{y}, i, t)$ as the probability that point $\mathbf{y}$ is the $(i + t)$-th connecting point, conditioned on point $\mathbf{x}$ being the i-th connecting point.

Value function. The value function f^s for staircase s is determined by the same rule as the constant rounds case in Section C.

A point is called a *self-intersection point* if it lies on multiple folded-segments[11] and we deem it to belong to the folded-segment closest to the solution. Thus, the distance from it to the start point is defined by tracing through all the previous points on the staircase.

Recall an important property of these value functions that if two staircases s_1, s_2 share a common length i prefix, then f^{s_1} and f^{s_2} have same value except for points that are after the i-th connecting points[12] of s_1 or s_2.

In the rest of this section, we will show that the distribution of value functions generated by all length $K = 2k$ staircases is hard for any k-rounds deterministic algorithm $\mathcal{A}$ with at most $\mathcal{Q}_k$ queries.

[10] Ignore the restriction that the 0-th connecting point $\mathbf{x}_0$ has to be $\mathbf{1}$ here, otherwise it may be impossible for an arbitrary point $\mathbf{x} \in [n]^d$ being the i-th connecting point.

[11] By Definition C.1 of folded-segment, the i-th connecting point is on the i-th folded-segment, but not the $(i + 1)$-th folded-segment.

[12] In particular, for a self-intersection point, it is after the i-th connecting points if any one of the folded-segments it intersects with are after the i-th connecting points.

D.2. Useful assumptions

We make several assumptions to simply the proof in this section.

Assumption D.1. Once the algorithm $\mathcal{A}$ has queried a point on the i-th folded-segment $\mathrm{FS}(\mathbf{x}_{i-1}, \mathbf{x}_i)$, the location of point $\mathbf{x}_i$ and all previous connecting points are provided to the algorithm $\mathcal{A}$.

With this assumption, we can quantify the *progress* of algorithm $\mathcal{A}$ at a certain round by the number of connecting points it *knows*, where we say algorithm $\mathcal{A}$ *knows* or *learns* the connecting point $\mathbf{x}_i$ if $\mathcal{A}$ is already given the location of $\mathbf{x}_i$.

Assumption D.2. Algorithm $\mathcal{A}$ learns at least one more connecting point in each round, and it *succeeds* on a specific input if it knows the location of the end point $\mathbf{x}_K$ of the staircase by the end of round k.

Notice that Assumptions D.1 and D.2 only make the bound stronger.

If $\mathcal{A}$ learns t more connecting points in one round, we say $\mathcal{A}$ *saves* $t-1$ rounds, since we expect $\mathcal{A}$ learns exactly one more connecting points in each round by default.

Assumption D.3. Algorithm $\mathcal{A}$ issues the same number of queries in each round, i.e., $\mathcal{Q} := \mathcal{Q}_k/k$.

Note that every k rounds of algorithm $\mathcal{A}$, with $\mathcal{Q}_k$ queries in total, can be converted to an algorithm $\mathcal{A}'$ in $2k$ rounds with $\mathcal{Q}_k/k$ queries in each round: if $\mathcal{A}$ makes t queries in a round, $\mathcal{A}'$ can simulate it by submitting those queries in $\lceil t/(\mathcal{Q}_k/k)\rceil$ rounds. This simulation can only increase the total number of rounds by a factor of 2, which is fine in the polynomial-rounds setting.

Assumption D.4. Algorithm $\mathcal{A}$ will keep running until it learns the end point of staircase, regardless of the round limit k.

A fixed round limit is more technically challenging to deal with. With Assumption D.4, we could instead argue that any such algorithm $\mathcal{A}$ will run more than $1.1k$ rounds in expectation. Then, by applying the Markov inequality, we will show that $\mathcal{A}$ fails to learn the end point of the staircase with probability at least 0.1 if the round limit k is imposed.

Let $\mathrm{SV}(s)$ be the total number of rounds $\mathcal{A}$ saved when $\mathcal{A}$ is running on the input generated by a fixed staircase s. By definition, the total number of rounds needed for learning the end point of s is $K - \mathrm{SV}(s)$.

With the assumptions and arguments above, the next two subsections are devoted to proving the following lemma, which establishes $\mathcal{Q}_k$ as the lower bound.

Lemma D.5. *The following inequality holds:*

$$\frac{L^{(K)} \cdot K - \sum_{s \in S^{(K)}} SV(s)}{L^{(K)}} \geq 1.1k,$$

where the left-hand side is the expected number of rounds needed to learn the end point considering the input distribution generated by length-K staircases.

D.3. Estimate the savings

Let r_i^s be the number of the round in which the i-th connecting point on staircase s is first known to $\mathcal{A}$ when running on the input f^s.

Definition D.6 (Critical point). The i-th connecting point is a *critical point* of a length t staircase s if $r_i^s \neq r_{i+1}^s$ or $i = t$.

By Assumption D.1, an equivalent definition is that when the i-th connecting point is first learned at round r_i^s, $\mathcal{A}$ has not queried any point that is after the i-th connecting point; that is, the i-th connecting point is learned by querying a point on the i-th folded-segment.

Intuitively, each critical point takes one round to learn, while each non-critical points are given for free by Assumption D.1.

By amortizing the rounds saved for each non-critical points to the next closest critical point, we define $\mathrm{SV}(s, i)$ as the number of rounds *saved* by the i-th connecting point of staircase s. More formally,

(1) $\mathrm{SV}(s, i) = i - \max\{j : r_j^s = r_i^s - 1\} - 1$ if i-th connecting point is a *critical point*;

(2) $\mathrm{SV}(s, i) = 0$ otherwise.

By definition, $\mathrm{SV}(s) = \sum_{i=1}^{k} \mathrm{SV}(s, i)$.

Let $r_{i,j}^s$, $j \geq i$ be the number of the round in which the i-th connecting point on staircase s is first learned by $\mathcal{A}$, when running on the input generated by the length j prefix of s.

Lemma D.7. *The following inequality holds:* $r_{i,j}^s \geq r_i^s$.

Proof. Denote the instance generated by the staircase s by f^s and the instance generated by the length j prefix of s by f_j^s.

Assume towards a contradiction that $r_{i,j}^s < r_i^s$. Then we prove by induction that the queries made by $\mathcal{A}$ from round 1 to round $r_{i,j}^s$ are the same on inputs f^s and f_j^s.

Base case. $\mathcal{A}$ makes the same queries in round 1, since $\mathcal{A}$ is a deterministic algorithm.

Induction step. Consider round $t \leq r^s_{i,j}$. By the induction hypothesis, all the queries in the previous $t-1$ rounds are the same. Notice that f^s is equal to f^s_j except for points on s which are after j-th connecting point. Moreover, $\mathcal{A}$ never queried any such point in the first $t-1$ rounds; otherwise we would have $r^s_i \leq t-1$ by Assumption D.1, which is impossible. Thus $\mathcal{A}$ also gets the same feedback from queries in the first $t-1$ rounds. This directly implies that $\mathcal{A}$ will make the same queries in round t since the algorithm is deterministic.

When running on the instance f^s_j, $\mathcal{A}$ queried a point $\mathbf{x}$ on the length j prefix of s at round $r^s_{i,j}$, which reveals the location of i-th connecting point by Assumption D.1. Obviously, $\mathbf{x}$ is also on the whole staircase s, and r^s_i should be equal to $r^s_{i,j}$, which contradicts the assumption that $r^s_{i,j} < r^s_i$. Thus the assumption was false and the inequality stated in the lemma holds. ∎

Lemma D.8. *If the j-th connecting point is a critical point of s, the following hold:*

(1) *the queries issued by $\mathcal{A}$ from round 1 to round $r^s_j + 1$ are the same when running on either f^s or f^s_j;*

(2) *$r^s_{i,j} = r^s_i$ for all $i \leq j$.*

Proof. Recall the equivalent definition of critical point: when running on input f^s, $\mathcal{A}$ will not query any point on s that is after the j-th connecting point in the first r^s_j rounds.

Notice that f^s and f^s_j have the same value except for points on s which are after j-th connecting point. Using an induction argument similar to the one in Lemma D.7, we show that the queries made by $\mathcal{A}$ from round 1 to round $r^s_j + 1$ are the same when running on either f^s or f^s_j.

Base case. $\mathcal{A}$ makes the same queries in round 1.

Induction step. Consider round $t \leq r^s_j + 1$. By the induction hypothesis, all the queries in the previous $t-1$ rounds are the same. Notice that f^s is equal to f^s_j except for points on s which are after j-th connecting point. Moreover, $\mathcal{A}$ never queried any such point in the first $t-1$ rounds; otherwise we would have $r^s_{i+1} \leq t-1 \leq r^s_j$, which contradicts to the fact that the j-th connecting point is critical. Thus $\mathcal{A}$ also gets the same feedback from queries in the first $t-1$ rounds. This directly implies that $\mathcal{A}$ will make the same queries in round t.

Since the j-th connecting point is a critical point, $\mathcal{A}$ must query a point that is on the j-th folded-segment in round r^s_j, when running on both input f^s or f^s_j. We then have $r^s_{j,j} = r^s_j$, as the fact that j-th folded-segment is in the length j prefix.

By Assumption D.1, when running on either f^s or f^s_j, for any $i < j$, the i-th connecting point is learned before round $r^s_{j,j} = r^s_j$, and all the queries before round r^s_j are the same. Thus, $r^s_{i,j} = r^s_i$ for any $i \leq j$. ∎

Let $s^{(i)}$ be the length i prefix of s. For all i, define

$$\overline{\mathrm{SV}(s,i)} = \mathrm{SV}\big(s^{(i)},i\big) = i - \max\{j : r^s_{j,i} = r^s_{i,i} - 1\} - 1.$$

The value of $\overline{\mathrm{SV}(s,i)}$ only depends on the length i prefix of s. Similarly, let

$$\overline{\mathrm{SV}(s)} = \sum_{i=1}^{K} \overline{\mathrm{SV}(s,i)}.$$

The next lemma shows that $\overline{SV(s)}$ is an upper bound of $SV(s)$ for any s.

Lemma D.9. *The inequality* $\overline{\mathrm{SV}(s)} \geq \mathrm{SV}(s)$ *holds for any staircase* s.

Proof. It suffices to prove that $\overline{\mathrm{SV}(s,i)} \geq \mathrm{SV}(s,i)$ for any i. The case of the i-th connecting point being non-critical point is easier, since $\overline{\mathrm{SV}(s,i)} \geq 0 = \mathrm{SV}(s,i)$.

If the i-th connecting point is a critical point, we have $r^s_{j,i} = r^s_j$ for any $j \leq i$ by Lemma D.8. In this case, the definitions of $\overline{\mathrm{SV}(s,i)}$ and $\mathrm{SV}(s,i)$ are the same. ∎

Assume algorithm $\mathcal{A}$ has now learned the first i connecting points and denote the i-th connecting point by $\mathbf{x}_i$. Let point $\mathbf{y}$ be a queried point in the next round and consider the number of rounds saved by point $\mathbf{y}$.

If point $\mathbf{y}$ is on the $(i+t-1)$-th folded-segment, t more connecting points are learned and $t-1$ rounds are saved by Assumption D.1. Suppose that algorithm $\mathcal{A}$ *forgets* every query it previously made and the structure of the length i prefix of the staircase except point $\mathbf{x}_i$. Then the probability of $\mathbf{y}$ being on the $(i+t-1)$-th folded-segment is $p(\mathbf{x}_i, \mathbf{y}, i, t)$ by definition.

In this worst-case for the algorithm, the number of rounds saved by $\mathbf{y}$ in expectation is bounded by the term:

$$\Gamma(i) = \max_{\mathbf{x},\mathbf{y}\in[n]^d} \sum_{t=1}^{K-i} (t-1)\cdot p(\mathbf{x},\mathbf{y},i,t). \tag{D.2}$$

The value of $\Gamma(i)$ only depends on the random walk used for generating staircase, regardless the choice of algorithm $\mathcal{A}$.

Define

$$\Gamma = \frac{1}{K}\sum_{i=0}^{K-1} \Gamma(i). \tag{D.3}$$

Even when the algorithm $\mathcal{A}$ does not forget (i.e., could benefit from the queries and knowledge acquired in previous rounds), Γ will be a good overall estimate.

The following key lemma upper bounds the savings of all staircase by the value of Γ.

Lemma D.10. *The following inequality holds:*

$$\sum_{s \in S^{(K)}} \mathrm{SV}(s) \leq L^{(K)} \cdot K \cdot \mathcal{Q} \cdot \Gamma.$$

Proof. We start by applying Lemma D.9 and reformulating the summation in a few steps by definition. We obtain the following:

$$\sum_{s \in S^{(K)}} \mathrm{SV}(s) \leq \sum_{s \in S^{(K)}} \overline{\mathrm{SV}(s)} \tag{D.4}$$

$$= \sum_{s \in S^{(K)}} \sum_{i=1}^{K} \overline{\mathrm{SV}(s,i)} \tag{D.5}$$

$$= \sum_{s \in S^{(K)}} \sum_{i=1}^{K} i - \max\{j : r^s_{j,i} = r^s_{i,i} - 1\} - 1 \tag{D.6}$$

$$= \sum_{s \in S^{(K)}} \sum_{j=0}^{K} \sum_{i=j+1}^{K} \mathbf{1}[r^s_{j,i} = r^s_{j+1,i} - 1 = r^s_{i,i} - 1] \cdot (i - j - 1) \tag{D.7}$$

$$= \sum_{j=0}^{K} \sum_{s_1 \in S^{(j)}} \sum_{i=j+1}^{K} \sum_{s_2 \in S^{(i)}(s_1)} \sum_{s \in S^{(K)}(s_2)} \mathbf{1}[r^s_{j,i} = r^s_{j+1,i} - 1 = r^s_{i,i} - 1] \cdot (i - j - 1) \tag{D.8}$$

$$= \sum_{j=0}^{K} \sum_{s_1 \in S^{(j)}} \sum_{i=j+1}^{K} (i - j - 1) \sum_{s_2 \in S^{(i)}(s_1)} \mathbf{1}[r^{s_2}_{j} = r^{s_2}_{j+1} - 1 = r^{s_2}_{i} - 1] \cdot |S^{(K)}(s_2)| \tag{D.9}$$

$$= L^{(K)} \cdot \sum_{j=0}^{K} \sum_{s_1 \in S^{(j)}} \sum_{i=j+1}^{K} \frac{(i - j - 1)}{L^{(i)}} \sum_{s_2 \in S^{(i)}(s_1)} \mathbf{1}[r^{s_2}_{j} = r^{s_2}_{j+1} - 1 = r^{s_2}_{i} - 1] \tag{D.10}$$

Inequality (D.4) is implied by Lemma D.9. Identities (D.5) and (D.6) are the definition of $\overline{\mathrm{SV}(s)}$. Identities (D.7) and (D.8) switch the order of the summation and enumerate a length K staircase s by first enumerating its length j and length i prefix. For equation (D.9), the value

$$\mathbf{1}[r^s_{j,i} = r^s_{j+1,i} - 1 = r^s_{i,i} - 1]$$

only depends on the length i prefix s_2. Finally, for equation (D.10), recall that for any length i prefix s_2, the value of $|S^{(K)}(s_2)|$ are same, namely $L^{(K)}/L^{(i)}$, which is guaranteed by the way we generate the staircases. An interpretation of equation (D.10) is that we first enumerate j and all length j prefix s_1, and then enumerate i and all length i prefix s_2 growing from s_1, if condition $r_j^{s_2} = r_{j+1}^{s_2} - 1 = r_i^{s_2} - 1$ is true, then there is a contribution of $(i - j - 1) \cdot (L^{(K)}/L^{(i)})$ to the total sum of saved rounds made by s_2.

Define a new condition: $H(s_1, s_2)$ is true if the i-th folded-segment of s_2 is queried in round $r_j^{s_1} + 1$ when $\mathcal{A}$ is running on the instance generated by s_1.

We show that condition $r_j^{s_2} = r_{j+1}^{s_2} - 1 = r_i^{s_2} - 1$ implies $H(s_1, s_2)$:

- If $r_i^{s_2} - 1 = r_j^{s_2}$ holds, then at least one point on i-th folded-segment of s_2 is queried in round $r_j^{s_2} + 1$ when running on the instance generated by s_2.
- If $r_j^{s_2} = r_{j+1}^{s_2} - 1$ holds, then by definition the j-th connecting point is a critical point of s_2. Further applying Lemma D.8, we know $r_j^{s_1} = r_{j,j}^{s_2} = r_j^{s_2}$ and the queries in the first $r_j^{s_2} + 1$ rounds are same when $\mathcal{A}$ is running on instance generated by either s_1 or s_2.

Thus, $H(s_1, s_2)$ holds by combining the two facts above.

Let $Q_{\mathcal{A}}(s, t)$ be the set of queries made by $\mathcal{A}$ in round t when running on instance generated by staircase s. For any point $\mathbf{x} \in Q_{\mathcal{A}}(s_1, r_j^{s_1} + 1)$, define $H(s_1, s_2, \mathbf{x})$ to be the condition that point $\mathbf{x}$ is on the i-th folded-segment of s_2.

We continue the previous calculation by replacing the condition $r_j^{s_2} = r_{j+1}^{s_2} - 1 = r_i^{s_2} - 1$ with $H(s_1, s_2)$ in equation (D.10):

$$\sum_{s \in S^{(K)}} SV(s) \leq L^{(K)} \cdot \sum_{j=0}^{K} \sum_{s_1 \in S^{(j)}} \sum_{i=j+1}^{K} \frac{(i-j-1)}{L^{(i)}} \sum_{s_2 \in S^{(i)}(s_1)} \mathbf{1}[H(s_1, s_2)] \tag{D.11}$$

$$\leq L^{(K)} \cdot \sum_{j=0}^{K} \sum_{s_1 \in S^{(j)}} \sum_{\mathbf{x} \in Q_{\mathcal{A}}(s_1, r_j^{s_1}+1)} \sum_{i=j+1}^{K} \frac{(i-j-1)}{L^{(i)}} \sum_{s_2 \in S^{(i)}(s_1)} \mathbf{1}[H(s_1, s_2, \mathbf{x})] \tag{D.12}$$

$$= L^{(K)} \cdot \sum_{j=0}^{K} \sum_{s_1 \in S^{(j)}} \sum_{\mathbf{x} \in Q_{\mathcal{A}}(s_1, r_j^{s_1}+1)} \sum_{i=j+1}^{K} \frac{(i-j-1)}{L^{(i)}} \cdot |S^{(i)}(s_1)| \cdot \Pr_{s_2 \in S^{(i)}(s_1)}[H(s_1, s_2, \mathbf{x})] \tag{D.13}$$

$$= L^{(K)} \cdot \sum_{j=0}^{K} \frac{1}{L^{(j)}} \sum_{s_1 \in S^{(j)}} \sum_{\mathbf{x} \in Q_{\mathcal{A}}(s_1, r_j^{s_1}+1)}$$

$$\sum_{i=j+1}^{K} (i-j-1) \cdot \Pr_{s_2 \in S^{(i)}(s_1)} [H(s_1, s_2, \mathbf{x})] \tag{D.14}$$

$$\leq L^{(K)} \cdot \sum_{j=0}^{K} \frac{1}{L^{(j)}} \sum_{s_1 \in S^{(j)}} \sum_{\mathbf{x} \in Q_{\mathcal{A}}(s_1, r_j^{s_1}+1)} \Gamma(j) \tag{D.15}$$

$$= L^{(K)} \cdot K \cdot \mathcal{Q} \cdot \Gamma \tag{D.16}$$

In the above, inequality (D.12) is the union bound and inequality (D.15) is implied by the definition of Γ in expression (D.3). ∎

D.4. Estimate Γ

We are left now with proving Lemma D.5 by showing $\Gamma \leq 9/20\mathcal{Q}$.

We first consider several properties of the random walk to simplify the expression for $\Gamma(i)$ given in (D.2).

Observation D.11. For any $\mathbf{x}, \mathbf{y} \in [n]^d, i, t \in [K]$, we have

$$p(\mathbf{x}, \mathbf{y}, i, t) = p(\mathbf{1}, \mathbf{y} - \mathbf{x}, i, t),$$
$$q(\mathbf{x}, \mathbf{y}, i, t) = q(\mathbf{1}, \mathbf{y} - \mathbf{x}, i, t).$$

To simplify notation, let $p(\mathbf{x}, i, t) := p(\mathbf{1}, \mathbf{x}, i, t)$ and $q(\mathbf{x}, i, t) := q(\mathbf{1}, \mathbf{x}, i, t)$. Now we have

$$\Gamma(i) := \max_{\mathbf{x} \in [n]^d} \sum_{t=1}^{K-i} (t-1) \cdot p(\mathbf{x}, i, t).$$

As the value of the function q is easier to estimate, the following lemma upper bounds the value of function p by the function q.

Lemma D.12. *For any $\mathbf{x} \in [n]^d$, the following holds:*

$$\begin{cases} p(\mathbf{x}, i, t) \leq 2\ell \cdot d \cdot \max_{\mathbf{y} \in W^{-1}(\mathbf{x})} q(\mathbf{y}, i, t-1) & \text{if } i + t \bmod m \neq 0, \\ p(\mathbf{x}, i, t) \leq n \cdot d \cdot \max_{\mathbf{y} \in [n]^d} q(\mathbf{y}, i, t-1) & \text{else.} \end{cases}$$

Proof. We will first prove the case of $i + t \bmod m \neq 0$. Let $\mathbf{y}, \mathbf{z}$ be the $(i+t-1)$-th $(i+t)$-th connecting points. Recall Definition C.1, folded-segment $\mathrm{FS}(\mathbf{y}, \mathbf{z})$ consists of d segments traversing in d directions, namely $E_1(\mathbf{y}, \mathbf{z}), \ldots, E_d(\mathbf{y}, \mathbf{z})$.

If $\mathbf{y} \in W_j^{-1}(\mathbf{x}) \setminus W_{j-1}^{-1}(\mathbf{x})$, the segment $E_j(\mathbf{y}, \mathbf{z})$ will visit $\mathbf{x}$ only if the first $j-1$ coordinate of point $\mathbf{z}$ is same to that of $\mathbf{x}$.

Consider a fixed choice of $\mathbf{x}$ and $\mathbf{y}$. As point $\mathbf{z}$ is uniformly drawn from $W(\mathbf{y})$, the probability that $\mathbf{x}$ is on $\mathrm{FS}(\mathbf{y}, \mathbf{z})$ is at most $1/\ell^{j-1}$. Similarly, if $\mathbf{y} \in W_j^b(\mathbf{x})$, the segment $E_j(\mathbf{y}, \mathbf{z})$ will visit point $\mathbf{x}$ in reverse direction only if the first $j-1$ coordinate of $\mathbf{z}$ is same to that of $\mathbf{x}$. The probability that $\mathbf{x}$ is on $\mathrm{FS}(\mathbf{y}, \mathbf{z})$ is also at most $1/\ell^{j-1}$.

Therefore, we obtain

$$p(\mathbf{x}, i, t) = \sum_{\mathbf{y} \in W^r(\mathbf{x})} q(\mathbf{y}, i, t-1) \cdot \Pr_{\mathbf{z} \in W(\mathbf{y})} \left[\mathbf{x} \in FS(\mathbf{y}, \mathbf{z})\right] \tag{D.17}$$

$$\leq \sum_{j \in [d]} \cdot \left(\sum_{\mathbf{y} \in W_j^{-1}(\mathbf{x})} q(\mathbf{y}, i, t-1) \cdot \frac{1}{\ell^{j-1}} + \sum_{\mathbf{y} \in W_j^b(\mathbf{x})} q(\mathbf{y}, i, t-1) \cdot \frac{1}{\ell^{j-1}} \right) \tag{D.18}$$

$$\leq \ell \cdot \sum_{j \in [d]} \frac{1}{\ell^j} \cdot \sum_{\mathbf{y} \in W_j^{-1}(\mathbf{x}) \cup W_j^b(\mathbf{x})} q(\mathbf{y}, i, t-1) \tag{D.19}$$

$$\leq 2\ell \cdot d \cdot \max_{\mathbf{y} \in W^r(\mathbf{x})} q(\mathbf{y}, i, t-1). \tag{D.20}$$

Identity (D.17) follows from $W^{-1}(\mathbf{x}) \cap W^b(\mathbf{x}) = \emptyset$; inequality (D.20) holds by applying the average principle on $q(\mathbf{y}, i, t-1)$, noticing that $|W_j^{-1}(\mathbf{x})|, |W_j^b(\mathbf{x})| \leq \ell^j$.

The case of $i + t \bmod m = 0$ is simpler and follows from same argument. ∎

Observation D.13. For any $\mathbf{x} \in [n]^d$,

$$p(\mathbf{x}, i, t) = p(\mathbf{x}, i + j, t), \quad q(\mathbf{x}, i, t) = q(\mathbf{x}, i + j, t),$$

where $i + j + t < m$.

For any $i < m$, we define the following quantities:

$$p(\mathbf{x}, i) := p(\mathbf{x}, 0, i), \quad q(\mathbf{x}, i) := q(\mathbf{x}, 0, i), \quad p_m(\mathbf{x}, i) := p(\mathbf{x}, m - i, i).$$

By definition, we have the following corollary of Lemma D.12.

Corollary D.14. *For any $\mathbf{x} \in [n]^d, i < m$, we have*

$$p(\mathbf{x}, i) \leq 2\ell \cdot d \cdot \max_{\mathbf{y} \in W^r(x)} q(\mathbf{y}, i-1),\ p_m(\mathbf{x}, i) \leq n \cdot d \cdot \max_{\mathbf{y} \in [n]^d} q(\mathbf{y}, i-1).$$

The value of $\max_{\mathbf{y} \in [n]^d} q(\mathbf{y}, i)$ could be estimated by the Gaussian distribution with mean value $\mathbf{i} \cdot \ell/2$ and co-variance matrix $\ell^2/12 \cdot \mathbf{I}$. The local central limit theorem (e.g., see [37]) guarantees the accuracy of such approximation.

Lemma D.15 ([37, Theorem 2.1.1]). *There is a constant c_L such that for any $i > 0$,*

$$\max_{\mathbf{y} \in [n]^d} q(\mathbf{y}, i) \leq \frac{c_L}{\ell^d \cdot i^{d/2}}.$$

Observation D.16. For any $\mathbf{x} \in [n]^d, i, t \in [K]$, we have

$$p(\mathbf{x}, i, t) = p(\mathbf{x}, i + m, t), q(\mathbf{x}, i, t) = q(\mathbf{x}, i + m, t).$$

By Observation D.16, there is $\Gamma(i) \geq \Gamma(i + m)$ for any $0 \leq i < K$. Therefore, we have

$$\Gamma \leq \frac{1}{m} \cdot \sum_{i=0}^{m-1} \Gamma(i).$$

Observation D.17. For any $\mathbf{x} \in [n]^d, i, t \in [K], i + t \leq K$, if $\lfloor i/m \rfloor < \lfloor (i + t)/m \rfloor$, we have

$$q(\mathbf{x}, i, t) = \frac{1}{n^d}.$$

This observation suggests that the tail part, i.e., the summation term with index $i + t > m$ in $\Gamma(i)$, is easier to estimate. Formally, define the prefix part

$$\Gamma'(i) = \max_{\mathbf{x} \in [n]^d} \sum_{t=1}^{m-i-1} (t-1) \cdot p(\mathbf{x}, t) + (m - i - 1) \cdot p_m(\mathbf{x}, m - i).$$

The following lemma shows that the tail part is small enough.

Lemma D.18. *If $c_d \leq 1/40d$, for any $0 \leq i < m$, we have*

$$\Gamma(i) \leq \Gamma'(i) + \frac{1}{10\mathcal{Q}}.$$

Proof. Let function $o_{n,\ell}(i) = n$ if $i \mod m = 0$, and $o_{n,\ell}(i) = \ell$ otherwise. Combining Lemma D.12 and Observation D.17, for any $0 \leq i < m$, we have

$$\begin{aligned}
\Gamma(i) &= \max_{\mathbf{x} \in [n]^d} \sum_{t=1}^{K-i} (t-1) \cdot p(\mathbf{x}, i, t) \\
&= \max_{\mathbf{x} \in [n]^d} \sum_{t=1}^{m-i} (t-1) \cdot p(\mathbf{x}, i, t) + \sum_{t=m-i+1}^{K-i} (t-1) \cdot p(\mathbf{x}, i, t) \\
&\leq \max_{\mathbf{x} \in [n]^d} \sum_{t=1}^{m-i} (t-1) \cdot p(\mathbf{x}, i, t) + \sum_{t=m-i+1}^{K-i} (t-1) \cdot o_{n,\ell}(i + t) \cdot \frac{d}{n^d} \\
&\leq \max_{\mathbf{x} \in [n]^d} \sum_{t=1}^{m-i} (t-1) \cdot p(\mathbf{x}, i, t) + \frac{K^2 \cdot \ell \cdot d}{n^d} \\
&\leq \max_{\mathbf{x} \in [n]^d} \sum_{t=1}^{m-i-1} (t-1) \cdot p(\mathbf{x}, t) + (m - i - 1) \cdot p_m(\mathbf{x}, m - i) + \frac{4d \cdot c_d}{\mathcal{Q}} \\
&\leq \Gamma'(i) + \frac{1}{10\mathcal{Q}}.
\end{aligned}$$

■

Lemma D.19. *The following inequality holds:*

$$\Gamma \le \frac{9}{20\mathcal{Q}}.$$

Proof. Define

$$\Gamma' = \frac{1}{m} \cdot \sum_{i=0}^{m-1} \Gamma'(i).$$

By Lemma D.18, it is equivalent to proving that $\Gamma' \le 7/20\mathcal{Q}$.

Let

$$\Phi_d(t) = \sum_{i=1}^{t} \frac{1}{t^{d/2-1}}.$$

We now estimate Γ' as follows:

$$\begin{aligned}
\Gamma' &\le \frac{1}{m} \cdot \sum_{i=0}^{m-1} \Gamma'(i) \\
&= \frac{1}{m} \cdot \sum_{i=0}^{m-1} \left(\max_{\mathbf{x}\in[n]^d} \sum_{t=1}^{m-i-1} (t-1) \cdot p(\mathbf{x}, t) + (m-i-1) \cdot p_m(\mathbf{x}, m-i) \right) \\
&\le \frac{1}{m} \cdot \sum_{i=0}^{m-1} \left(\sum_{t=1}^{m-i-1} (t-1) \cdot \max_{\mathbf{x}\in[n]^d} p(\mathbf{x}, t) \right. \\
&\qquad \left. + (m-i-1) \cdot \max_{\mathbf{x}\in[n]^d} p_m(\mathbf{x}, m-i) \right) \\
&\le \frac{1}{m} \cdot \sum_{i=0}^{m-1} \left(\sum_{t=1}^{m-i-1} (t-1) \cdot 2\ell \cdot d \cdot \max_{\mathbf{y}\in[n]^d} q(\mathbf{y}, t-1) \right. \\
&\qquad \left. + (m-i-1) \cdot n \cdot d \cdot \max_{\mathbf{y}\in[n]^d} q(\mathbf{y}, m-i-1) \right) \qquad \text{(D.21)} \\
&\le \frac{1}{m} \cdot \sum_{i=0}^{m-1} \left(\sum_{t=1}^{m-i-1} (t-1) \cdot 2\ell \cdot d \cdot \frac{c_L}{\ell^d \cdot (t-1)^{d/2}} \right. \\
&\qquad \left. + (m-i-1) \cdot (m \cdot \ell) \cdot d \cdot \frac{c_L}{\ell^d \cdot (m-i-1)^{d/2}} \right) \qquad \text{(D.22)} \\
&\le \frac{1}{m} \cdot \sum_{i=0}^{m-1} \sum_{t=1}^{m} (t-1) \cdot 2\ell \cdot d \cdot \frac{c_L}{\ell^d \cdot (t-1)^{d/2}} \\
&\qquad + \sum_{i=0}^{m-1} (m-i-1) \cdot \ell \cdot d \cdot \frac{c_L}{\ell^d \cdot (m-i-1)^{d/2}}
\end{aligned}$$

$$\leq \frac{2\Phi_d(m)\cdot d\cdot c_L}{\ell^{d-1}} + \frac{\Phi_d(m)\cdot d\cdot c_L}{\ell^{d-1}}$$
$$= \frac{3\cdot\Phi_d(m)\cdot d\cdot c_L}{\ell^{d-1}}.$$

Inequality (D.21) is implied by Corollary D.14, and inequality (D.22) follows from Lemma D.15. By equation (D.1), we have

$$\frac{1}{\mathcal{Q}} = \frac{k}{\mathcal{Q}_k} = O\Big(\frac{\Phi_d(m)}{\ell^{d-1}}\Big).$$

Therefore, we conclude the proof by picking the constant c_d small enough. ∎

Now we wrap everything up and prove the lower bound of Theorem 3.2.

Lower Bound of Theorem 3.2. Lemma D.5 directly follows from Lemma D.10 and Lemma D.19. Since the query complexity with round limits must be larger than or equal to that of fully adaptive setting, we further improve our bound for $d = 3$ by taking max with $\Omega(n^{3/2})$, which is the bound for the fully adaptive algorithm from [57]. ∎

E. Brouwer in rounds

In this section, we consider the query complexity of finding fixed-points in rounds. For Brouwer we focus on constant rounds, since more than $\log 1/\varepsilon$ rounds do not improve the query complexity for Brouwer [18, 22]. If the number of rounds is a non-constant function smaller than $\log 1/\varepsilon$, this only changes the bound by a sub-polynomial term.

Theorem 3.3 (Brouwer, constant rounds, restated). *Let $k \in \mathbb{N}$ be a constant. For any $\varepsilon > 0$, the query complexity of the ε-approximate Brouwer fixed-point problem in the d-dimensional unit cube $[0,1]^d$ with k rounds is $\Theta((1/\varepsilon)^{(d^{k+1}-d^k)/(d^k-1)})$ for both deterministic and randomized algorithms.*

Similarly to local search in constant rounds, when $k \to \infty$, this bound converges to $\Theta((1/\varepsilon)^{d-1})$ and the gap is smaller than any polynomial. Thus our result fills the gap between one round algorithm and fully adaptive algorithm (logarithmic rounds) except for a small margin.

We first give some background on the Brouwer problem, and then explain how the upper and lower bounds are obtained.

E.1. Background on Brouwer

The existence of fixed points is guaranteed by the Brouwer fixed-point theorem [15].

Theorem E.1 (Brouwer fixed-point theorem). *Let $K \subset \mathbb{R}^n$ be a compact and convex subset of $\mathbb{R}^n$ and $f: K \to K$ a continuous function. Then there exists a point $\mathbf{x}^* \in K$ such that $f(\mathbf{x}^*) = \mathbf{x}^*$.*

We study the computational problem of finding a fixed-point given the function f and the domain K. We focus on the setting where $K = [0, 1]^d$ and sometimes write fixed-point to mean Brouwer fixed-point for brevity.

Since computers cannot solve this problem to arbitrary precision, we consider the approximate version with an error parameter ε and the goal is to find an ε-fixed-point. In this case, the original function f can be approximated by a Lipschitz continuous function $\widetilde{f}$, where the Lipschitz constant of $\widetilde{f}$ can be arbitrarily large and depends on the quality of approximation.

We also consider a discrete analogue that will be useful for understanding the complexity in the continuous setting. The discrete problem was shown to be equivalent to the continuous (approximate) setting in [18].

Discrete Brouwer fixed-point. Given a vector $\mathbf{x} \in [n]^d$, let x_i denote the value at the i-th coordinate of $\mathbf{x}$. Consider a function $f: [n]^d \to \{\mathbf{0}, \pm\mathbf{e}^1, \pm\mathbf{e}^2, \ldots, \pm\mathbf{e}^d\}$, where $\mathbf{e}^i$ is the i-th unit vector and f satisfies the following properties:

- *direction-preserving*: for any $\|\mathbf{x} - \mathbf{y}\|_\infty \leq 1$, we have $\|f(\mathbf{x}) - f(\mathbf{y})\|_\infty \leq 1$;
- *bounded*: for any $\mathbf{x} \in [n]^d$, we have $\mathbf{x} + f(\mathbf{x}) \in [n]^d$.

Then there exists a point $\mathbf{x}^*$ such that $f(\mathbf{x}^*) = \mathbf{0}$ (see [35]).[13]

Figure 5 illustrates a bounded and direction preserving function in two dimensions.

To compare the difficulty of problems under rounds limit on oracle evaluation, we use the *round-preserving* reduction defined as follows.

Definition E.2 (Round-preserving reduction). We call a reduction from the oracle-based problem *P1* to the oracle-based problem *P2 round-preserving* if for any instance of problem *P1* with oracle $\mathcal{O}_1$, the instance of problem *P2* with oracle $\mathcal{O}_2$ given by the reduction satisfies that:

(1) a solution of the *P1* instance can be obtained from any solution of the *P2* instance without any more queries on $\mathcal{O}_1$;

(2) each query to $\mathcal{O}_2$ can be answered by a constant number of queries to $\mathcal{O}_1$ in one round.

[13] The approximate version and the discrete version problems are called AFP and ZP, respectively, in [18]

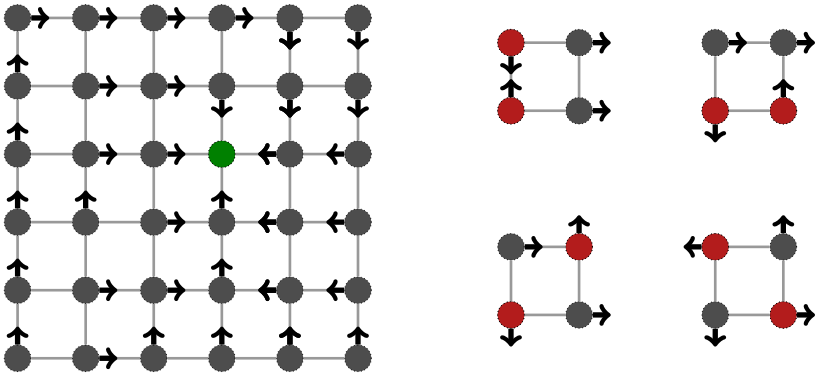


Figure 5. Figure (a) shows a bounded and direction preserving function in two dimensions, on a grid of size 6×6. The fixed-point is shown in green. Several examples of patterns forbidden by the direction preserving property can be seen in Figure (b).

The following lemma established the equivalence between the approximate and the discrete version of fixed-point problem.

Lemma E.3 ([18, Section 5]). *The following statements hold.*

(1) *There is a* round-preserving *reduction such that any instance* (ε, L, d, F) *of approximate fixed-point is reduced to an instance* $(C_1(d) \cdot (L+1)/\varepsilon, d, f)$ *of the discrete fixed-point problem, where* $C_1(d)$ *is a constant that only depends on the dimension* d.

(2) *There is a* round-preserving *reduction with parameter* $L > 1$ *such that any instance* (n, d, f) *of discrete fixed-point problem is reduced to an instance* (ε, L, d, F) *of approximate fixed-point problem, satisfying that*

$$\left(\frac{L-1}{\varepsilon}\right) \geq (C_2(d) \cdot n),$$

where $C_2(d)$ *is a constant that only depends on* d.

Remark E.4. The original reduction from the discrete fixed-point problem to the approximate the version in [18] will take one more round of queries of the function f of discrete fixed-point problem after getting the solution point $\mathbf{x}^*$ of approximate fixed-point problem. However, these extra queries can be avoided if ε is small enough. For example, by taking $\varepsilon = (L-1)/(n \cdot d \cdot 2^d)$, the closest grid point (in L_∞-norm) to the point $\mathbf{x}^*$ will be a zero point of f under the construction in [18].

Since d and L are constants independent of ε, we can study both the upper bound and lower bound of the discrete fixed-point problem first, then replace "n" with "$1/\varepsilon$" to get the bound for the approximate fixed-point computing problem.

E.2. Bounds for Brouwer

We prove the bounds for the discrete version of Brouwer first, and then obtain the bounds for the continuous problem.

Theorem E.5 (Discrete Brouwer fixed-point). *Let k be a constant. The query complexity of computing a discrete Brouwer fixed-point on the d-dimensional grid $[n]^d$ in k rounds is $\Theta(n^{(d^{k+1}-d^k)/(d^k-1)})$, for both deterministic and randomized algorithms.*

Theorem 3.3 will follow by taking $\varepsilon = 1/n$.

E.2.1. Upper bound for Brouwer. Our constant rounds algorithm generalizes the divide-and-conquer algorithm in [18] in the same way as we generalizing the local search algorithm in [39] in Section A. We first present several necessary definitions and lemmas in [18].

Definition E.6 (Bad cube [18, Definition 6]). A 0-dimensional unit cube $C^0 = \{\mathbf{x}\}$ is bad if $f(\mathbf{x}) = \mathbf{e}^1$.

For each $i \geq 1$, an i-dimensional unit cube $C^i \subset [n]^d$ is bad with respect to the function $f : [n]^d \to \{\mathbf{0}, \pm\mathbf{e}^1, \pm\mathbf{e}^2, \ldots, \pm\mathbf{e}^d\}$ if

(1) $\{f(\mathbf{x}) : \mathbf{x} \in C^i\} = \{\mathbf{e}^1, \mathbf{e}^2, \ldots, \mathbf{e}^{i+1}\}$;

(2) the number of bad $(i-1)$-dimensional unit cubes in C^i is odd.

The following theorem on the boundary condition of the existence of the solution is essential for the design of the divide-and-conquer based algorithm.

Theorem E.7 ([18, Theorem 3]). *A $(d-1)$-dimensional unit cube C is on the boundary of a d-dimensional cube C' if every point in C is on the boundary of C'.*

If the number of bad $(d-1)$-dimensional unit cubes on the boundary of the d-dimensional cube C is odd, then the bounded direction-preserving function f has a zero point within C.

The final piece that enables us to use a divide-and-conquer approach is that we can pad the original problem instance on the grid $[n]^d$ to a larger grid: $\{0, 1, \ldots, n+1\}^d$, and make sure that the new instance contains exactly one bad $(d-1)$-dimensional unit cube on its boundary, and no new solution is introduced. Let f and f' be the function for the original and the new instance, respectively. Then $f'(\mathbf{x}) = f(\mathbf{x})$ for any $\mathbf{x} \in [n]^d$; for any other point $\mathbf{x} = (x_1, \ldots, x_d)$, let i be the largest number such

that $x_i \in \{0, n+1\}$, we have $f'(\mathbf{x}) = \mathbf{e}_i$ if $\mathbf{x}_i = 0$, and $f'(\mathbf{x}) = -\mathbf{e}_i$ otherwise. The correctness of this reduction is showed [18, Lemma 5].

Algorithm for Brouwer.

(1) Initialize the cube C_0 as $\{0, 1, \ldots, n+1\}^d$; for each $0 \le i < k$, set $\ell_i = n^{(d^k - d^i)/(d^k - 1)}$.

(2) In each round $i \in \{1, \ldots, k-1\}$:

- Divide the current cube C_{i-1} into sub-cubes $C_i^1, \ldots, C_i^{n_i}$ of side length ℓ_i that cover C_{i-1}. These sub-cubes have mutually exclusive interior, but each $(d-1)$-dimensional unit cube that is not on the boundary of cube C_{i-1} is either (i) on the boundary of two sub-cubes $C_i^{j_1}, C_i^{j_2}$, or (ii) not on the boundary of any sub-cubes.
- Query f' with all the points on the boundary of sub-cubes $C_i^1, \ldots, C_i^{n_i}$.
- Set $C_i = C_i^j$, where C_i^j is the sub-cube that has odd number of bad $(d-1)$-dimensional unit cubes on its boundary. Choose arbitrary one if there are many.

(3) In round k, query all the points in the current cube C_{k-1} and get the solution point.

Upper bound of Theorem E.5. Notice that for any $i \in \{1, \ldots, k-1\}$, the sum of the numbers of bad $(d-1)$-dimensional cubes on the boundary of $C_i^1, \ldots, C_i^{n_i}$ is still odd, since each bad cubes that is not on the boundary of C_{i-1} is counted twice. By the parity argument, C_i always exists. Finally, Lemma E.7 guarantees that there exists a solution in cube C_{k-1}, which concludes the correctness of this algorithm. The calculations in Section A give that this algorithm makes $O(n^{(d^{k+1}-d^k)/(d^k-1)})$ queries in total. ∎

E.2.2. Randomized lower bound for Brouwer. We reduce the local search instances *generated by staircases* to discrete Brouwer fixed-point instances in this section, and thus prove the lower bound part of Theorem E.5.

We use the problem GP defined in [22] as an intermediate problem in the reduction.

Definition E.8 (GP [22, Section 2.2]). A graph directed $G = ([n]^d, E)$ is a grid PPAD graph if

(1) the underlying undirected graph of G is a subgraph of grid graph defined on $[n]^d$;

(2) there is one directed path $\mathbf{1} = \mathbf{x}_0 \to \mathbf{x}_1 \to \cdots \to \mathbf{x}_{T-1} \to \mathbf{x}_T$ with no self-intersection in the graph. Any other point outside of the path is an isolated point.

The structure of G is accessed by the mapping function $N_G(\mathbf{x})$ from $[n]^d$ to $([n]^d \cup \{\text{“no”}\}) \times ([n]^d \cup \{\text{“no”}\})$, such that

(1) $N_G(\mathbf{x}_0) = (\text{“no”}, \mathbf{x}_1)$;

(2) $N_G(\mathbf{x}_T) = (\mathbf{x}_{T-1}, \text{“no”})$;

(3) $N_G(\mathbf{x}_i) = (\mathbf{x}_{i-1}, \mathbf{x}_{i+1})$ for all $i \in \{1, \ldots, T-1\}$;

(4) $N_G(\mathbf{x}) = (\text{“no”}, \text{“no”})$ otherwise.

Then GP is the following search problem: Given n, d and the function $N_G(\mathbf{x})$, find the end of the path $\mathbf{x}_T$.

The following lemma from [22] shows that we can get the lower bound of discrete fixed-point problem from the lower bound of problem GP directly.

Lemma E.9 ([22, Theorem 3.2]). *There is a* round-preserving *reduction such that any instance* (n, d, N_G) *of problem* GP *is reduced to an instance* $(24n + 7, d, f)$ *of discrete fixed-point problem.*

The remaining work is establishing the lower bound of problem GP by reducing the local search instances generated by staircases to GP.

Lemma E.10. *There is a* round-preserving *reduction such that any local search instance* (n, d, f) generated by a possible staircase*, defined in Section* C*, is reduced to an instance* (n, d, N_G) *of problem* GP.

Proof. Naturally, we can let the staircases[14] in the local search instance to be the path in the GP instance. Given a local search problem instance on grid $[n]^d$ with value function f that is constructed by staircase as in Section C, we reduce it to a GP instance on grid $[n]^d$ with function $N_G(\mathbf{x})$.

Each query of $N_G(\mathbf{x})$ is answered by at most $2d + 1$ of queries of value function f for $\mathbf{x}$ and all its neighbors. From these queries of f:

- if $f(\mathbf{x}) > 0$, let $\mathbf{x}$ be an isolated point, i.e., $N_G(\mathbf{x}) = (\text{“no”}, \text{“no”})$;
- otherwise, $\mathbf{x}$ should be on the path. If $f(\mathbf{x}) = 0$, $\mathbf{x}$ is the start of the path; if $\mathbf{x}$ is a local optima, $\mathbf{x}$ is the end of the path. The direction within the paths is from the point with higher value to the point with lower value, and there are at most one neighbor with higher value and at most one neighbor with lower value for any value function f constructed in Section C. Thus we can answer $N_G(\mathbf{x})$ correctly by the values of $\mathbf{x}$ and its neighbor.

[14] Recall that in Section C, the value of the end point $\mathbf{x}$ of the staircases may be a positive value with probability $1/2$. In this case, $\mathbf{x}$ is not considered to be on the path, and the previous point is the end of the path in problem GP.

Since the start point and the end point of a staircase are unique under the construction in Section C, $\mathbf{x}_T$ is the solution of the original local search problem, and the start of the path $\mathbf{x}_0$ satisfies $\mathbf{x}_0 = \mathbf{1}$. This reduction does not change the number of rounds needed, and only increases the number of queries by a constant factor $2d + 1$. Thus the reduction above is a *round-preserving* reduction. ∎

Lower bound of Theorem E.5. Combining Lemmas E.9, E.10 and Theorem 3.1 concludes the proof of the lower bound in Theorem E.5. ∎

F. Local search and Brouwer fixed-point in one dimension

In this section, we study the local search and discrete Brouwer fixed-point on a 1-dimensional grid.

Recall that the white-box version (where the function is given by a polynomial size circuit) of local search is PLS-Complete for any $d \geq 2$. On the other hand, when $d = 1$, a binary search procedure can solve the problem with $O(\log n)$ queries. Therefore, the query complexity in the 1-dimensional case exhibits different properties than that in dimension $d \geq 2$.

Theorem F.1 (1-dimensional case). *Let k be a constant. The query complexity of computing a local minimum on the 1-dimensional grid $[n]$ in k rounds is $\Theta(n^{1/k})$ for both deterministic and randomized algorithms.*

We note that in the fully adaptive case, the query complexity of this problem is $\Theta(\log n)$. We have similar results for Brouwer. Here we are given a function

$$f : [0, 1] \to [0, 1]$$

with Lipschitz constant $L > 1$ and a parameter $\varepsilon > 0$, and the goal is to find an ε-approximate fixed point of f. We first show a bound for the discrete Brouwer problem, which will imply a tight bound for the continuous setting.

Theorem F.2 (Discrete Brouwer in one dimension). *Let k be a constant. Given a bounded and direction-preserving function $f : [n] \to [n]$, the query complexity of computing a discrete Brouwer fixed point in k rounds is $\Theta(n^{1/k})$ for both deterministic and randomized algorithms.*

Corollary F.3 (Brouwer in one dimension). *Let k be a constant. Given a L-Lipschitz continuous function $f : [0, 1] \to [0, 1]$ with $L > 1$ and $\varepsilon > 0$, the query complexity of computing an ε-approximate fixed point of f in k rounds is $\Theta((1/\varepsilon)^{1/k})$ for both deterministic and randomized algorithms.*

We note that in the fully adaptive case, the query complexity of this problem is $\Theta(\log 1/\varepsilon)$.

We conclude the proof of Theorems F.1 and F.2 by a deterministic algorithm in Section F.1 and a matching lower bound for randomized algorithm in Section F.2.

F.1. Algorithm in one dimension

The algorithm is a simple adaptation of previous constant rounds algorithms into one dimension. Each sub-cube is now a sub-interval, and we take $\ell_i = n^{(k-i)/k}$ as the length of the sub-interval at the i-th round to balance the queries in each round.

Algorithm in constant rounds in one dimension.

(1) Initialize the current interval as $[n]$.

(2) In each round $i \in \{1, \ldots, k-1\}$:

- Divide the current interval into $n^{1/k}$ of mutually exclusive sub-intervals, and query all the points on the boundary of these sub-intervals.
- For local search, set the current interval to be the sub-interval with minimum value in it; for Brouwer, set the current interval to be the one has a fixed-point or both boundary points pointing inwards. Break tie arbitrarily.

(3) In round k, query all the points in the current interval and find the solution point.

Upper bound of Theorem F.1 and Theorem F.2. The correctness of the algorithm follows from a similar argument to the $d \geq 2$ case: for local search, the steepest descent from the minimal point will not escape from the current sub-interval; for Brouwer, the direction-preserving function in one dimension cannot change its direction without *stopping* at a fixed-point.

The total number of queries is at most $(k-1) \cdot 2n^{1/k} + n^{1/k} = O(n^{1/k})$. ∎

F.2. Lower bound in one dimension

Lower bound of Theorem F.1 and Theorem F.2. By Yao's minimax theorem, we first construct a distribution of hard instances, and then use the standard decision tree method to show that $O(n^{1/k})$ queries are necessary for any deterministic algorithm.

Hard local search instances. A uniform distribution over n possible local search instances $f_1, \ldots, f_n$, where f_i is defined as follows:

(1) $f_i(i) = 0$;

(2) $f_i(j) = j$ if $j > i$;

(3) $f_i(j) = n - j + 1$ if $j < i$.

The only solution of f_i is the point i. A second observation is that for any set of q query points, there are at most $2q + 1$ possible results considering all f_i, because there are at most $2q + 1$ possible different relative locations between the q queried points.

Now consider the decision tree of any k-rounds deterministic algorithm A. If the number of queries on each node is $o(n^{1/k})$, then every node has at most $o(n^{1/k})$ branches. Since A only has k rounds, the number of leaf nodes at the bottom of the decision tree is only $o(n)$, i.e., A only has at most $o(n)$ different possible outputs. Recall that there are n possible local search instances with different solutions, thus, A must fail with high probability.

Hard discrete Brouwer fixed-point instances. A uniform distribution over n possible instances $f_1, \ldots, f_n$, where f_i is defined as follows:

(1) $f_i(i) = 0$;

(2) $f_i(j) = -1$ if $j > i$;

(3) $f_i(j) = 1$ if $j < i$.

Any function f_i is bounded and direction-preserving, and the only solution is point i. The remaining proof follows from the same argument for local search: There are at most $2q + 1$ different outcomes from any set of q queries. Thus, for any k-rounds deterministic algorithm A, if the number of queries is $o(n^{1/k})$, the number of leaf nodes in the decision tree of A is $o(n)$, which is far from enough to cover all n possible instances.

Therefore, we conclude that $\Omega(n^{1/k})$ queries are necessary for any k rounds algorithm for local search or Brouwer in one dimension. ∎

Acknowledgments. We wish to thank the anonymous COLT and MSL reviewers for very helpful feedback and suggestions. In particular, the question regarding providing bounds for local search in rounds on general graphs and the associated technical challenges was posed by one of the MSL reviewers.

Funding. Simina Brânzei was supported in part by US National Science Foundation CAREER grant CCF-2238372.

References

[1] S. Aaronson, Lower bounds for local search by quantum arguments. *SIAM J. Comput.* **35** (2006), no. 4, 804–824 Zbl 1099.68034 MR 2203568

[2] S. G. Akl, *Parallel sorting algorithms*. Notes Rep. Comput. Sci. Appl.Math. 12, Academic Press, Orlando, FL, 1985 Zbl 0657.68070 MR 0807771

[3] D. Aldous, Minimization algorithms and random walk on the d-cube. *Ann. Probab.* **11** (1983), no. 2, 403–413 Zbl 0513.60068 MR 0690137

[4] N. Alon, Y. Azar, and U. Vishkin, Tight complexity bounds for parallel comparison sorting. In *27th Annual Symposium on Foundations of Computer Science (SFCS 1986)*, pp. 502–510, IEEE, 1986

[5] I. Althöfer and K.-U. Koschnick, On the deterministic complexity of searching local maxima. *Discrete Appl. Math.* **43** (1993), no. 2, 111–113 Zbl 0777.05069 MR 1220960

[6] M. Arjovsky, S. Chintala, and L. Bottou, Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, pp. 214–223, JMLR, 2017

[7] Y. Babichenko, S. Dobzinski, and N. Nisan, The communication complexity of local search. In *STOC'19—Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pp. 650–661, ACM, New York, 2019 Zbl 1433.68143 MR 4003372

[8] E. Balkanski, A. Rubinstein, and Y. Singer, An exponential speedup in parallel running time for submodular maximization without loss in approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 283–302, SIAM, Philadelphia, PA, 2019 Zbl 1431.68144 MR 3909488

[9] E. Balkanski and Y. Singer, The adaptive complexity of maximizing a submodular function. In *STOC'18—Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1138–1151, ACM, New York, 2018 Zbl 1427.68365 MR 3826324

[10] E. Balkanski and Y. Singer, A lower bound for parallel submodular minimization. In *STOC '20—Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 130–139, ACM, New York, 2020 Zbl 1551.68143 MR 4141748

[11] B. Bollobás, Sorting in rounds. *Discrete Math.* **72** (1988), no. 1-3, 21–28 Zbl 0667.68075 MR 0975520

[12] S. Brânzei, D. Choo, and N. Recker, The sharp power law of local search on expanders. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1792–1809, SIAM, Philadelphia, PA, 2024 Zbl 07951144 MR 4699316

[13] M. Braverman, J. Mao, and Y. Peres, Sorted top-k in rounds. In *Proceedings of the Thirty-Second Conference on Learning Theory*, pp. 342–382, Proceedings of Machine Learning Research 99, PMLR, 2019

[14] M. Braverman, J. Mao, and S. M. Weinberg, Parallel algorithms for select and partition with noisy comparisons. In *STOC'16—Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 851–862, ACM, New York, 2016 Zbl 1375.68186 MR 3536619

[15] L. Brouwer, Continuous one-one transformations of surfaces in themselves. In *Huygens Institute - Royal Netherlands Academy of Arts and Sciences (KNAW)*, pp. 286–297, KNAW Proceedings 12, Amsterdam, 1910

[16] S. Bubeck, Q. Jiang, Y. T. Lee, Y. Li, and A. Sidford, Complexity of highly parallel non-smooth convex optimization. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 13923–13932, Curran Associates, Red Hook, NY, 2019

[17] S. Bubeck and D. Mikulincer, How to trap a gradient flow. In *Proceedings of Thirty Third Conference on Learning Theory*, pp. 940–960, Proceedings of Machine Learning Research 125, PMLR, 2020

[18] X. Chen and X. Deng, On algorithms for discrete and approximate Brouwer fixed points. In *STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pp. 323–330, ACM, New York, 2005 Zbl 1192.68351 MR 2181632

[19] X. Chen and X. Deng, On the complexity of 2D discrete fixed point problem. *Theoret. Comput. Sci.* **410** (2009), no. 44, 4448–4456 Zbl 1183.68294 MR 2561571

[20] X. Chen, X. Deng, and S.-H. Teng, Settling the complexity of computing two-player Nash equilibria. *J. ACM* **56** (2009), no. 3, article no. 14 Zbl 1325.68095 MR 2536129

[21] X. Chen, D. Paparas, and M. Yannakakis, The complexity of non-monotone markets. *J. ACM* **64** (2017), no. 3, article no. 20 Zbl 1427.91130 MR 3666863

[22] X. Chen and S.-H. Teng, Paths beyond local search: A tight bound for randomized fixed-point computation. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pp. 124–134, IEEE, 2007

[23] V. Cohen-Addad, F. Mallmann-Trenn, and C. Mathieu, Instance-optimality in the noisy value-and comparison-model: Accept, accept, strong accept: which papers get in? In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, pp. 2124–2143, SIAM, Philadelphia, PA, 2020 Zbl 07304155 MR 4141313

[24] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, The complexity of computing a Nash equilibrium. *SIAM J. Comput.* **39** (2009), no. 1, 195–259 Zbl 1185.91019 MR 2506524

[25] C. Daskalakis and C. Papadimitriou, Continuous local search. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 790–804, SIAM, Philadelphia, PA, 2011 Zbl 1373.68263 MR 2857165

[26] C. Daskalakis, S. Skoulakis, and M. Zampetakis, The complexity of constrained min-max optimization. In *STOC '21—Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1466–1478, ACM, New York, 2021 Zbl 07765262 MR 4398933

[27] H. Dinh and A. Russell, Quantum and randomized lower bounds for local search on vertex-transitive graphs. *Quantum Inf. Comput.* **10** (2010), no. 7-8, 636–652 Zbl 1234.81054 MR 2683409

[28] A. Ene and H. L. Nguyen, Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 274–282, SIAM, Philadelphia, PA, 2019 Zbl 1431.68152 MR 3909487

[29] J. Fearnley, P. Goldberg, A. Hollender, and R. Savani, The complexity of gradient descent: CLS = PPAD ∩ PLS. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 46–59, ACM, New York, NY, 2021

[30] W. Gararch, E. Golub, and C. Kruskal, Constant time parallel sorting: An empirical view. *J. Comput. System Sci.* **67** (2003), no. 1, 63–91 Zbl 1054.68039 MR 1991254

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative adversarial nets. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, pp. 2672–2680, MIT Press, Cambridge, MA, 2014

[32] M. Göös, A. Hollender, S. Jain, G. Maystre, W. Pires, R. Robere, and R. Tao, Further collapses in TFNP. *SIAM J. Comput.* **53** (2024), no. 3, 573–587 Zbl 7852674 MR 4740631

[33] M. D. Hirsch, C. H. Papadimitriou, and S. A. Vavasis, Exponential lower bounds for finding Brouwer fixed points. *J. Complexity* **5** (1989), no. 4, 379–416 Zbl 0696.65045 MR 1028904

[34] P. Hubáček and E. Yogev, Hardness of continuous local search: query complexity and cryptographic lower bounds. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1352–1371, SIAM, Philadelphia, PA, 2017 Zbl 1410.68145 MR 3627817

[35] T. Iimura, A discrete fixed point theorem and its applications. *J. Math. Econom.* **39** (2003), no. 7, 725–742 Zbl 1055.91061 MR 2007101

[36] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, How easy is local search? *J. Comput. System Sci.* **37** (1988), no. 1, 79–100 Zbl 0655.68074 MR 0973658

[37] G. F. Lawler and V. Limic, *Random walk: A modern introduction*. Cambridge Stud. Adv. Math. 123, Cambridge University Press, Cambridge, 2010 Zbl 1210.60002 MR 2677157

[38] D. C. Llewellyn and C. Tovey, Dividing and conquering the square. *Discrete Appl. Math.* **43** (1993), no. 2, 131–153 Zbl 0783.68058 MR 1220962

[39] D. C. Llewellyn, C. Tovey, and M. Trick, Local optimization on graphs. *Discrete Appl. Math.* **23** (1989), no. 2, 157–178 Zbl 0675.90085 MR 0996547

[40] D. Monderer and L. S. Shapley, Potential games. *Games Econom. Behav.* **14** (1996), no. 1, 124–143 Zbl 0862.90137 MR 1393599

[41] J. F. Nash, Jr., Equilibrium points in n-person games. *Proc. Nat. Acad. Sci. U.S.A.* **36** (1950), 48–49 Zbl 0036.01104 MR 0031701

[42] A. Nemirovski, On parallel complexity of nonsmooth convex optimization. *J. Complexity* **10** (1994), no. 4, 451–463 Zbl 0820.68058 MR 1306424

[43] C. H. Papadimitriou, On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. System Sci.* **48** (1994), no. 3, 498–532 Zbl 0806.68048 MR 1279412

[44] N. Pippenger, Sorting and selecting in rounds. *SIAM J. Comput.* **16** (1987), no. 6, 1032–1038 Zbl 0654.68068 MR 0917039

[45] R. W. Rosenthal, A class of games possessing pure-strategy Nash equilibria. *Internat. J. Game Theory* **2** (1973), 65–67 Zbl 0259.90059 MR 0319584

[46] M. Santha and M. Szegedy, Quantum and classical query complexities of local search are polynomially related. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pp. 494–501, ACM, New York, 2004 Zbl 1192.68266 MR 2121635

[47] B. Settles, *Active learning*. Synth. Lect. Artif. Intell. Mach. Learn. 18, Springer, Cham, 2012 Zbl 1270.68006

[48] X. Sun and A. C.-C. Yao, On the quantum query complexity of local search in two and three dimensions. *Algorithmica* **55** (2009), no. 3, 576–600 Zbl 1192.68283 MR 2512035

[49] C. Tovey, Polynomial local improvement algorithms in combinatorial optimization. PhD thesis, Stanford University, 1981

[50] L. G. Valiant, Parallelism in comparison problems. *SIAM J. Comput.* **4** (1975), no. 3, 348–355 Zbl 0311.68033 MR 0378467

[51] S. A. Vavasis, Black-box complexity of local minimization. *SIAM J. Optim.* **3** (1993), no. 1, 60–80 Zbl 0794.90045 MR 1202002

[52] V. V. Vazirani and M. Yannakakis, Market equilibrium under separable, piecewise-linear, concave utilities. *J. ACM* **58** (2011), no. 3, article no. 10 Zbl 1327.91042 MR 2810999

[53] Y. F. Verhoeven, Enhanced algorithms for local search. *Inform. Process. Lett.* **97** (2006), no. 5, 171–176 Zbl 1184.68213 MR 2195213

[54] A. Wigderson and D. Zuckerman, Expanders that beat the eigenvalue bound: explicit construction and applications. *Combinatorica* **19** (1999), no. 1, 125–138 Zbl 0929.05075 MR 1722214

[55] A. C. C. Yao, Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science*, pp. 222–227, IEEE, Long Beach, CA, 1977 MR 0489016

[56] C.-H. Zhang, Nearly unbiased variable selection under minimax concave penalty. *Ann. Statist.* **38** (2010), no. 2, 894–942 Zbl 1183.62120 MR 2604701

[57] S. Zhang, Tight bounds for randomized and quantum local search. *SIAM J. Comput.* **39** (2009), no. 3, 948–977 Zbl 1194.68119 MR 2538845

Received 10 January 2024; revised 13 July 2025.

Simina Brânzei
Department of Computer Science, Purdue University West Lafayette, Purdue Mall, West Lafayette, IN 47907, USA; simina.branzei@gmail.com

Jiawei Li
Department of Computer Science, The University of Texas at Austin, Austin, TX 78712, USA; davidlee@cs.utexas.edu