Lower Bounds for Adaptive Relaxation-Based Algorithms for Single-Source Shortest Paths

Sunny Atalig
University of California, Riverside, CA, USA

Alexander Hickerson
University of California, Riverside, CA, USA

Arrdya Srivastav
University of California, Riverside, CA, USA

Tingting Zheng
Guangdong University of Technology, Guangzhou, China

Marek Chrobak
University of California, Riverside, CA, USA

Abstract

We consider the classical single-source shortest path problem in directed weighted graphs. D. Eppstein proved recently an $\Omega(n^3)$ lower bound for oblivious algorithms that use relaxation operations to update the tentative distances from the source vertex. We generalize this result by extending this $\Omega(n^3)$ lower bound to *adaptive* algorithms that, in addition to relaxations, can perform queries involving some simple types of linear inequalities between edge weights and tentative distances. Our model captures as a special case the operations on tentative distances used by Dijkstra's algorithm.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases single-source shortest paths, lower bounds, decision trees

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.8

Funding Research partially supported by National Science Foundation grant CCF-2153723.

1 Introduction

We consider the classical single-source shortest path problem in directed weighted graphs. In the case when all edge weights are non-negative, Dijkstra's algorithm [8], if implemented using Fibonacci heaps, computes the shortest paths in time $O(m + n \log n)$, where n is the number of vertices and m is the number of edges. In the general case, when negative weights are allowed (but not negative cycles), the Bellman-Ford algorithm [20, 2, 19, 12] solves this problem in time $O(nm)$.

Both algorithms work by repeatedly executing operations of *relaxations*. (This type of algorithms are also sometimes called label-setting algorithms [7].) Let ℓ_{uv} denote the weight of an edge (u, v). For each vertex v, these algorithms maintain a value $D[v]$ (that we will refer to as the *D-value* at v) that represents the current upper bound on the distance from the source vertex s to v. A relaxation operation for an edge (u, v) replaces $D[v]$ by $\min\{D[v], D[u] + \ell_{uv}\}$. That is, $D[v]$ is replaced by $D[u] + \ell_{uv}$ if visiting v via u turns out to give a shorter distance to v, based on the current distance estimates. When the algorithm completes, each value $D[v]$ is equal to the correct distance from s to v. Dijkstra's algorithm executes only one relaxation for each edge, while in the Bellman-Ford algorithm each edge can be relaxed $\Theta(n)$ times.

We focus on the case of complete directed graphs, in which case $m = n(n-1)$. For complete graphs, the number of relaxations in Dijkstra's algorithm is $\Theta(n^2)$. In contrast, the Bellman-Ford algorithm executes $\Theta(n^3)$ relaxations. This raises the following natural question: *is it possible to solve the shortest-path problem by using asymptotically fewer than $O(n^3)$ relaxations, even if negative weights are allowed?*

To make this question meaningful, some restrictions need to be imposed on allowed algorithms. Otherwise, an algorithm can "cheat": it can compute the shortest paths without any explicit use of relaxations, and then execute $n-1$ relaxations on the edges in the shortest-path tree, in order of their hop-distance from s, thus making only $n-1$ relaxations.

Eppstein [9] circumvented this issue by assuming a model where the sequence of relaxations is independent of the weight assignment. Then the question is whether there is a short "universal" sequence of relaxations, namely one that works for an arbitrary weight assignment. The Bellman-Ford algorithm is essentially such a universal sequence of length $O(n^3)$. Eppstein [9] proved that this is asymptotically best possible; that is, each universal relaxation sequence must have $\Omega(n^3)$ relaxations. This lower bound applies even in the randomized case, when the relaxation sequence is generated randomly and the objective is to minimize the expected number of relaxations.

The question left open in [9] is whether the $\Omega(n^3)$ lower bound applies to relaxation-based *adaptive* algorithms, that generate relaxations based on information collected during the computation. (This problem is also mentioned by Hu and Kozma [15], who remark that lower bounds for adaptive algorithms have been "elusive".) We answer this question in the affirmative for some natural types of adaptive algorithms.

In our computation model, an algorithm is allowed to perform two types of operations: (i) *queries*, which are simple linear inequalities involving edge weights and D-values, and (ii) *relaxation updates*, that modify D-values. The action at each step depends on the outcomes of the earlier executed queries. Such algorithms can be represented as decision trees, with queries and updates in their nodes, and with each query node having two children, one corresponding to the "yes" outcome and the other to the "no" outcome.

Specifically, we study *query/relaxation-based algorithms* that can make queries of three types:

D-comparison query: "$D[u] < D[v]$?", for two vertices u, v,
Weight-comparison query: "$\ell_{uv} < \ell_{xy}$?", for two edges (u, v), (x, y),
Edge query: "$D[u] + \ell_{uv} < D[v]$?", for an edge (u, v),
and can update D-values as follows:
Relaxation update: "$D[v] \leftarrow \min\{D[v], D[u] + \ell_{uv}\}$", for an edge (u, v).

Throughout the paper, for brevity, we will write "D-query" instead of "D-comparison query" and "weight query" instead of "weight-comparison query".

We assume that initially $D[s] = 0$ and $D[v] = \ell_{sv}$ for all vertices $v \neq s$. This initialization and the form of relaxation updates ensure that at all times each value $D[v]$ represents the length of some simple path from s to v. Thus D-queries and edge queries amount to comparing the lengths of two paths from s. Further, the D-values induce a tentative approximation of the shortest-path tree, where a node u is the parent of a node v if the last decrease of $D[v]$ resulted from a relaxation of edge (u, v). So the algorithm's decision at each step depends on this tentative shortest-path tree.

Our contributions. We start by considering algorithms that use only edge queries. For such algorithms we prove the following $\Omega(n^3)$ lower bound:

▶ **Theorem 1.** *(a) Let $\mathcal{A}$ be a deterministic query/relaxation-based algorithm for the single-source shortest path problem that uses only edge queries. Then the running time of $\mathcal{A}$ is $\Omega(n^3)$, even if the weights are non-negative and symmetric (that is, the graph is undirected). (b) If $\mathcal{A}$ is a randomized algorithm then the same $\Omega(n^3)$ lower bound holds for $\mathcal{A}$'s expected running time.*

We first give the proof of Theorem 1(a), the lower bound for deterministic algorithms. In this proof, (in Section 3), we view the computation of $\mathcal{A}$ as a game against an adversary who gradually constructs a weight assignment, consistent with the queries, on which most of the edge queries performed by $\mathcal{A}$ will have negative outcomes, thus revealing little information to $\mathcal{A}$ about the structure of the shortest-path tree.

Then, in Section 4, we show how to extend this lower bound to all three types of queries if negative weights are allowed, proving Theorem 2(a) below.

▶ **Theorem 2.** *(a) Let $\mathcal{A}$ be a deterministic query/relaxation algorithm for the single-source shortest path problem that uses the three types of queries: D-queries, weight-queries, and edge queries, as well as relaxation updates. Then the running time of $\mathcal{A}$ is $\Omega(n^3)$. (b) If $\mathcal{A}$ is a randomized algorithm then the same $\Omega(n^3)$ lower bound holds for $\mathcal{A}$'s expected running time.*

Our query/relaxation model captures as a special case the operations on tentative distances used by Dijkstra's algorithm, because D-queries are sufficient to maintain the ordering of vertices according to their D-values. More broadly, Theorem 2 may be helpful in guiding future research on speeding up shortest-path algorithms for the general case, when negative weights are allowed, by showing limitations of naïve approaches based on extending Dijkstra's algorithm.

The proof of Theorem 2(a) is essentially via a reduction, showing that the model with all three types of queries can be reduced to the one with only edge queries, and then applying the lower bound from Theorem 1(a). This reduction modifies the weight assignment, making it asymmetric and introducing negative weights. As a side result, we also observe in Theorem 4 that this reduction works even for arbitrary (not necessarily complete) graphs, giving a lower bound that generalizes the one in [9], as it applies to adaptive algorithms in our query/relaxation model.

Finally, in Section 5 we extend both lower bounds to randomized algorithms. The proofs are based on Yao's principle [22]; that is, we give a probability distribution on weight assignments on which any deterministic algorithm performs poorly.

Our lower bound results are valid even if all weights are integers of polynomial size. In the proof of Theorem 1 all weights are non-negative integers with maximum value $\ell_{\max} = O(n)$. The proof of Theorem 2 uses Golomb rulers [21, 10, 4] (also known as Sidon sets) to construct weight assignments with maximum value $\ell_{\max} = O(n^4)$. In the randomized case, these bounds increase by a factor of $O(n)$.

As explained near the end of Section 5, the lower bounds for expectation in Theorems 1(b) and 2(b) can be quite easily extended to high-probability bounds.

Related work. As earlier mentioned, the Bellman-Ford algorithm can be thought of as a universal relaxation sequence. It consists of $n-1$ iterations with each iteration relaxing all edges in some pre-determined order, so the length of this sequence is $(1+o(1))n^3$. The leading constant 1 in this bound was reduced to $\frac{1}{2}$ by Yen [23], who designed a universal sequence with $(\frac{1}{2}+o(1))n^3$ relaxations. Eppstein's lower bound in [9] shows in fact a lower bound of $\frac{1}{6}$ on the leading constant, and just recently Hu and Kozma [15] proved that constant $\frac{1}{2}$ is in fact optimal.

Bannister and Eppstein [1] showed that the leading constant can be reduced to $\frac{1}{3}$ with randomization, namely that there is a probability distribution on relaxation sequences for which a sequence, drawn from this distribution, will compute correct distances in expected time $(\frac{1}{3} + o(1))n^3$ (or even with high probability). Eppstein's lower bound proof [9] for randomized sequences shows that this constant is at least $\frac{1}{12}$.

Some of the above-mentioned papers extend the results to graphs that are not necessarily complete. In particular, Eppstein [9] proved that for n-vertex graphs with m edges, $\Omega(mn/\log n)$ relaxations are necessary.

The average-case complexity of the Bellman-Ford and Dijkstra's algorithms has also been studied. For example, Meyer et al. [18] show that the Bellman-Ford algorithm requires $\Omega(n^2)$ steps on average, if the weights are uniformly distributed random numbers from interval $[0, 1]$.

Some work has been done on improving lower and upper bounds in models beyond our query/relaxation setting. Of those, the recent breakthrough paper by Fineman [11] is particularly relevant. It gives a randomized $\tilde{O}(mn^{8/9})$-expected-time algorithm for computing single-source shortest paths with arbitrary weights. Fineman's computation model is not far from ours in the sense that the weights are arbitrary real numbers and the only arithmetic operations on weights are additions and subtractions, but it also needs branch instructions that cannot be expressed using our queries.

The special case when weights are integers is natural and has been extensively investigated (see [6, 13, 3], for example). In the integer domain one can extract information about the weight distribution, and thus about the structure of the shortest-path tree, using operations other than linear inequalities involving weights. The state-of-the-art in this model is the (randomized) algorithm by Bernstein et al. [3] that achieves running time $O(m \log^8 n \log W)$ with high probability for weight assignments where the smallest weight is at least $-W$ (and $W \geq 2$).

Some lower bounds have also been reported for related problems, for example for shortest paths with restrictions on the number of hops [5, 14, 17] or k-walks [16].

2 Preliminaries

The input is a weighted complete directed graph G. The set of all vertices of G is denoted by V, and $s \in V$ is designated as the *start vertex*. The set of all edges of G is denoted by E and a *weight assignment* is a function $\ell\colon E \to \mathbb{Z}$. (While real-valued weights are common in the literature, in our constructions we only need integers.) We will use notations $\ell(u, v)$ and ℓ_{uv} for the weight of an edge (u, v). By $\ell_{\max}$ we denote the maximum absolute value of an edge weight, that is $\ell_{\max} = \max_{(u,v)\in E} |\ell_{uv}|$.

Whenever we write "path" we mean a "simple path", that is a path where each vertex is visited at most once. The *distance from x to y* is defined as the length of the shortest path from x to y. We will assume that the input graph does not have negative cycles. Note that this assumption gives an algorithm additional information that can potentially be used to reduce the running time.

The edges in the shortest paths from s to all other vertices form a tree that is called the shortest-path tree. The root of this tree is s. (There is a minor subtlety here related to ties. A more precise statement is that *there is a way to break ties*, so that the shortest paths form a tree.)

Formalizing query/relaxation models. We now formally define our computation model. We assume that each vertex v has an associated value $D[v]$, called the D-value at v. Initially $D[s] = 0$ and $D[v] = \ell_{sv}$ for $v \neq s$. A *query* is a boolean function whose arguments are edge

weights and D-values. A *query model* $\mathbb{Q}$ is simply a set of allowed queries. For example, the model that has only edge queries is $\mathbb{Q} = \{1_{D[v]<D[u]+\ell_{uv}} \mid (u,v) \in E\}$, where 1_ξ is the indicator function for a predicate ξ. The query/relaxation model in Theorem 1 has query model $\mathbb{Q}$ consisting of all D-queries, weight-queries, and edge queries. (The reduction in Section 4 is actually for an even more general query model.)

An algorithm $\mathcal{A}$ using a query/relaxation model $\mathbb{Q}$ is then a decision tree, where each internal node corresponds to either a query from $\mathbb{Q}$ (with one "yes" and one "no" branch) or a relaxation operation (which has one branch), and each leaf is a relaxation operation. (These leaves have no special meaning.) With this definition, at any step of the computation, $D[v]$ represents the length of a path from s to v. This decision tree must correctly compute all distances from s; that is, for each weight assignment ℓ, when the computation of $\mathcal{A}$ reaches a leaf then for each vertex v the value of $D[v]$ must be equal to the distance from s to v.

The running time of $\mathcal{A}$ for a weight assignment ℓ is defined as the number of steps performed by $\mathcal{A}$ until each value $D[v]$ is equal to the correct distance from s to v. Notice that this is not the same as the depth of the decision tree, which could be greater. (This definition matches the concept of "reduced cost" used in [9] for non-adaptive algorithms. For deterministic algorithms we could as well define the running time as the maximum tree depth, but this definition wouldn't work in the randomized case.)

Edge weights using potential functions. We define a *potential function* as a function $\phi : V \to \mathbb{Z}$ with $\phi(s) = 0$. (These functions are also sometimes called *price functions* in the literature.) To reduce clutter, we will sometimes write the potential value on v as ϕ_v instead of $\phi(v)$.

A potential function induces a weight assignment $\Delta\phi$ defined by $\Delta\phi(u,v) = \phi_v - \phi_u$, for each $(u,v) \in E$. Such potential-induced weights satisfy the following *path independence property:* For any two vertices u, v, all paths from u to v have the same length, namely $\Delta\phi(u,v)$. Note that $\Delta\phi$ will have some negative weights, unless ϕ is identically 0, but it does not form negative cycles. Also, every spanning tree rooted at s is a shortest-path tree for $\Delta\phi$.

Any weight assignment ℓ can be combined with a potential function ϕ to obtain a new weight assignment $\ell' = \ell + \Delta\phi$. Such ℓ' satisfies the following *distance preservation property:* For any two vertices u, v and any path P from u to v, we have $\ell'(P) = \ell(P) + \phi_v - \phi_u$.

Due to the above properties, potential functions have played a key role in the most recent single-source shortest path algorithms [3, 11], in particular being used to transform a negative weight assignment into a non-negative one so that Dijkstra's algorithm can be applied. However, in this paper we will use them for an entirely different purpose, which is to construct difficult weight assignments in Section 4. Roughly, a potential-induced assignment $\Delta\phi$ can act as a "mask" on top of existing weights that renders D-queries and weight queries useless.

3 Lower Bound for Deterministic Algorithms with Edge Queries

This section gives the proof of Theorem 1(a). That is, we prove that every deterministic algorithm that uses relaxations and edge queries needs to make $\Omega(n^3)$ operations to compute correct distances. This lower bound applies even if all weights are non-negative and the weight assignment is symmetric. (One can think of it as an undirected graph, although we emphasize that in the proof below we use directed edges.)

For the proof, fix an algorithm $\mathcal{A}$. We will show how to construct a weight assignment such that only after $\Omega(n^3)$ operations the D-values computed by $\mathcal{A}$ represent the correct distances from the source vertex.

Each weight assignment considered in our construction is symmetric and is uniquely specified by a permutation of the vertices. The weight assignment corresponding to a permutation $\pi = x_0, x_1, ..., x_{n-1}$, where $x_0 = s$, is defined as follows: for any $0 \leq i < j < n$,

$$\ell_\pi(x_i, x_j) = \begin{cases} 2 & \text{if } j = i+1 \\ L - 5i/2 & \text{if } j \geq i+2 \text{ and } i \text{ is even} \\ L & \text{if } j \geq i+2 \text{ and } i \text{ is odd} \end{cases}$$

where L is some sufficiently large integer, say $L = 5n$. Then the shortest path tree is just a Hamiltonian path $x_0, x_1, ..., x_{n-1}$. Note that the distance between any two vertices is less than $2n$, while each edge not on this path has length larger than $2n$.

The proof is by showing an adversary strategy that gradually constructs a permutation of the vertices in response to $\mathcal{A}$'s operations. The strategy consists of $(n-1)/2$ phases. (For simplicity, assume that n is odd.) When a phase k starts, for $k = 1, ..., (n-1)/2$, the adversary will have already revealed a prefix $X_{k-1} = x_0, x_1, ..., x_{2k-2}$ of the final permutation. The goal of this phase is to extend X_{k-1} by two more vertices, responding to $\mathcal{A}$'s queries and updates so as to force $\mathcal{A}$ to make as many operations as possible within the phase, without revealing anything about the rest of the permutation.

To streamline the proof, we think about the initial state as following the non-existent $0'$th phase, and we assume that the D-values for all vertices other than s are initialized to $L+1$, instead of L.

We now describe the adversary strategy in phase k, by specifying how the adversary responds to each operation of $\mathcal{A}$ executed in this phase. Let $Y_{k-1} = V \setminus X_{k-1}$, let A be set of the edges from x_{2k-2} to Y_{k-1} and B be the set of edges inside Y_{k-1}. The adversary will maintain marks on the edges in $A \cup B$, starting with all edges unmarked. We will say that $\mathcal{A}$ *accesses* an edge (u, v) if it executes either an edge query or a relaxation for (u, v).

The idea is this: because of the choice of edge weights and the invariants on the D-values (to be presented soon), each edge query for an edge $(x_{2k-2}, y) \in A$ not yet relaxed in this phase will have a positive outcome. This way, these responses will not reveal what the next vertex x_{2k-1} on the path is. The adversary waits until $\mathcal{A}$ relaxes all these edges, and keeps track of these relaxations by marking the relaxed edges. At the same time, $\mathcal{A}$ may be accessing edges in B. The adversary waits until the last access of $\mathcal{A}$ to an edge $(u, v) \in B$ for which edge (x_{2k-2}, u) is already marked. Until this point, all queries to edges in B have negative outcomes. Only this last edge will have a positive outcome to an edge query, if it's made by $\mathcal{A}$, and the adversary will further make sure that this edge gets relaxed, before ending the phase.

To formalize this, let (u, v) be the edge accessed by $\mathcal{A}$ in the current operation. We describe the adversary's response by distinguishing several cases:

(s1) $(u, v) = (x_{2k-2}, v) \in A$. If this is a relaxation, mark (u, v). If this is an edge query do this: if (u, v) is unmarked, respond "yes", else respond "no".

(s2) $(u, v) \in B$. We have two sub-cases depending on the type of access.

- **Relaxation:** If (x_{2k-2}, u) is marked, mark (u, v). If all edges in $A \cup B$ are marked, end phase k.
- **Edge query:** If (x_{2k-2}, u) is not marked, respond "no". So suppose that (x_{2k-2}, u) is marked. In that case, if (u, v) is not the last unmarked edge in $A \cup B$, mark it and respond "no". If (u, v) is the last unmarked edge, respond "yes" (without marking).

(s3) $(u, v) \notin A \cup B$. If this is an edge query, respond "no". If this is a relaxation for (u, v), do nothing.

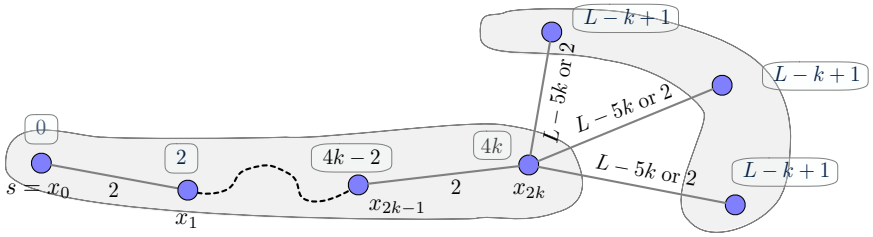

Figure 1 The state of the game right after phase k ends. Framed numbers next to vertices represent their D-values.

Let (u^*, v^*) be the edge marked last in this phase. This is the edge $(u, v) \in B$ from rule (s2) that becomes marked when it gets relaxed with all other edges in B already marked, ending the phase. At this point the adversary lets $x_{2k-1} = u^*$ and $x_{2k} = v^*$, and (if $k < (n-1)/2$) starts phase $k+1$.

For any permutation π of V starting with s, through the rest of the proof we denote by D_π the variable D-values produced by $\mathcal{A}$ when processing weight assignment ℓ_π. For each $k = 0, 1, ..., (n-1)/2$, the $(2k+1)$-permutation $x_0, x_1, ..., x_{2k}$ chosen by the adversary following the strategy above will be called the *kth cruel prefix*.

Invariant (I). We claim that the following invariant holds for each $k = 0, 1, ..., (n-1)/2$. Let $x_0, x_1, ..., x_{2k}$ be the adversary's kth cruel prefix. Then for each permutation π starting with $x_0, x_1, ..., x_{2k}$, the following properties hold when phase k of the adversary strategy ends (see Figure 1 for illustration):

(I0) All adversary's answers to $\mathcal{A}$'s queries in phases $1, 2, ..., k$ are correct for weight assignment ℓ_π.

(I1) $D_\pi[x_j] = 2j$ for $j = 0, 1, ..., 2k$.

(I2) If $w \in V \setminus \{x_0, x_1, ..., x_{2k}\}$ then $D_\pi[w] = L - k + 1$.

We postpone the proof of this invariant, and show first that it implies the $\Omega(n^3)$ lower bound. Indeed, from Invariant (I2) we conclude that the D-values will not represent the correct distances until after the last step of phase $(n-1)/2$. Since in each phase k all edges in the set $A \cup B$ for phase k will end up marked, the number of edge accesses in this phase is at least $|A \cup B| = |A| + |A|(|A| - 1) = |A|^2 = (n - 2k + 1)^2$. Thus, adding up the numbers of edge accesses in all phases $k = 1, 2, ..., (n-1)/2$, we obtain that the total number of steps in algorithm $\mathcal{A}$ is at least $(n-1)^2 + (n-3)^2 + ... + 2^2 = \frac{1}{6}n(n^2-1) = \Omega(n^3)$, giving us the desired lower bound.

It remains to prove Invariant (I). The invariant is true for $k = 0$, by the way the D-values are initialized. To argue that the invariant is preserved after each phase $k \geq 1$, we show that within this phase a more general invariant (J) holds, below.

Invariant (J). Let π be a permutation with prefix $x_0, x_1, ..., x_{2k}$, let $X_{k-1} = x_0, x_1, ..., x_{2k-2}$, and $Y_{k-1} = V \setminus X_{k-1}$. We claim that the following properties are satisfied during the phase, including right before and right after the phase.

(J0) All adversary's answers to $\mathcal{A}$'s queries up to the current step are correct for ℓ_π.

(J1) $D_\pi[x_j] = 2j$ for $j = 0, 1, ..., 2k-2$.

(J2.1) If $w \in Y_{k-1} \setminus \{u^*, v^*\}$ then $D_\pi[w] = \begin{cases} L-k+2 & \text{if } (x_{2k-2}, w) \text{ unmarked} \\ L-k+1 & \text{if } (x_{2k-2}, w) \text{ marked} \end{cases}$

(J2.2) If $w = u^*$ then $\quad D_\pi[u^*] = \begin{cases} L-k+2 & \text{if } (x_{2k-2}, u^*) \text{ unmarked} \\ 4k-2 & \text{if } (x_{2k-2}, u^*) \text{ marked} \end{cases}$

(J2.3) If $w = v^*$ then $\quad D_\pi[v^*] = \begin{cases} L-k+2 & \text{if } (x_{2k-2}, v^*) \text{ unmarked} \\ L-k+1 & \text{if } (x_{2k-2}, v^*) \text{ marked and } (u^*, v^*) \text{ unmarked} \\ 4k & \text{if } (u^*, v^*) \text{ marked} \end{cases}$

When phase k starts, these properties are identical to Invariant (I) applied to the ending of phase $k-1$. We now show that these invariants are preserved within a phase k. Assume that the invariants hold up to some step, and consider the next operation when $\mathcal{A}$ accesses an edge (u, v). If $w \neq v$ then $D_\pi[w]$ is not affected, so assume that $w = v$. In the case analysis below, if the current step is a relaxation, we will use notation $D_\pi[v]$ for the D-value at v before this step and $D'_\pi[v]$ for the D-value at v after the step.

Case (s1). $(u, v) = (x_{2k-2}, v) \in A$. We consider separately the cases when this step is a relaxation or an edge query.

Relaxation: Suppose first that $v \neq u^*$. Then we have $D_\pi[x_{2k-2}] + \ell(x_{2k-2}, v) = (4k-4) + (L-5k+5) = L-k+1$. Thus, using (J2.1) and (J2.3), if (x_{2k-2}, v) was already marked then nothing changes, and if (x_{2k-2}, v) wasn't marked then $D'_\pi[v] = L-k+1$, preserving (J2) because (x_{2k-2}, v) gets marked. (Note that in the special case $v = v^*$, edge (u^*, v^*) is not marked yet.)

For $v = u^*$ the argument is similar, except that now we use (J2.2): We have $D_\pi[x_{2k-2}] + \ell(x_{2k-2}, u^*) = (4k-4) + 2 = 4k-2$, so either (x_{2k-2}, u^*) is already marked and nothing changes, or $D'_\pi[u^*] = 4k-2$ and (x_{2k-2}, u^*) gets marked.

Edge query: The reasoning here is analogous to the case of relaxation above. If $v \neq u^*$, then $D_\pi[x_{2k-2}] + \ell(x_{2k-2}, v) = L-k+1$ and the correctness of the adversary's answers follows from (J2.1) and (J2.3), If $v = u^*$, then $D_\pi[x_{2k-2}] + \ell(x_{2k-2}, u^*) = (4k-4) + 2 = 4k-2$, and the correctness of the adversary's answers follows from (J2.2).

Case (s2). $(u, v) \in B$. We consider separately the cases when this step is a relaxation or an edge query.

Relaxation: Suppose first that $u \neq u^*$. Then $D_\pi[u] \geq L-k+1$, by (J2.1) and (J2.3). (This is true for the special case $u = v^*$, because (u^*, v^*) is not yet marked.) Since also $\ell(u, v) \geq 2$, this relaxation will not change the value of $D_\pi[v]$.

Next, consider the case $u = u^*$. If (x_{2k-2}, u^*) is unmarked then (J2.2) also implies (as in the previous sub-case) that the value of $D_\pi[v]$ will not change. So assume now that (x_{2k-2}, u^*) is marked, in which case $D_\pi[u^*] = 4k-2$. If $v \neq v^*$ then the relaxation will not change the value of $D_\pi[v]$ because $\ell(u^*, v) = L$. For $v = v^*$, we have $D'_\pi[v^*] = D_\pi[u^*] + \ell(u^*, v^*) = (4k-2) + 2 = 4k$, preserving (J2.3), because this relaxation will mark (u^*, v^*).

Edge query: If (x_{2k-2}, u) is not marked then, by (J2.1)-(J2.3) we have $D_\pi[u] = L-k+2$, and $\ell(u, v) \geq 2$, so the "no" answer by the adversary is correct.

Next, assume that (x_{2k-2}, u) is marked and $u \neq u^*$. Then $D_\pi[u] = L-k+1$, by conditions (J2.1) and (J2.3) (since (u^*, v^*) is still not marked). So in this case the answer "no" is also correct.

The final case is when $u = u^*$ and (x_{2k-2}, u^*) is marked, so $D_\pi[u^*] = 4k-2$. Now, if $v \neq v^*$ then the adversary responds "no", and since $\ell(u^*, v) = L$, this is correct. For $v = v^*$ we have $D_\pi[u^*] + \ell(u^*, v^*) = (4k-2) + 2 = 4k < D_\pi[v^*]$, where the last inequality is true because (u^*, v^*) is not marked. So the "yes" answer is also correct.

Case (s3). $(u, v) \notin A \cup B$. In this case we claim that $D_\pi[u] + \ell(u, v) \geq D_\pi[v]$, which implies the correctness for both cases, when this operation is a relaxation and edge update. The argument involves a few cases.

The first case is when $u \in Y_{k-1}$ and $v \in X_{k-1}$. Then we have $D_\pi[u] + \ell_{uv} \geq (4k-2)+2 > D_\pi[v]$, applying (J1)-(J2.3).

If $u \in X_{k-1} \setminus \{x_{2k-2}\}$ and $v \in Y_{k-1}$ then there are two sub-cases, and in both we apply (J1) and (J2.1)-(J2.3). If $u = x_{2k-3}$ in which case $\ell(x_{2k-3}, v) = L$, the claim is trivial. If $u = x_j$ for $j \leq 2k-4$, then $D_\pi[x_j] = 2j$ and $\ell(x_j, v) \geq L - 5j/2$, so $D_\pi[u] + \ell(u, v) \geq (2j) + (L - 5j/2) = L - j/2 \geq L - k + 2 \geq D_\pi[v]$.

The final case is when $u, v \in X_{k-1}$, say $u = x_i$ and $v = x_j$. Here we use condition (J1). If $i > j$ then $D_\pi[x_i] > D_\pi[x_j]$. If $i < j - 1$ then $\ell(x_i, x_j) \geq 2n$. If $i = j - 1$ then $D_\pi[x_{j-1}] = 2j - 2$, $D_\pi[x_j] = 2j$ and $\ell(x_{j-1}, x_j) = 2$. In each of these sub-cases, the claim holds.

The case analysis above completes the proof of invariants (J0)-(J2.3). By applying these invariants to the end of the phase, when all edges in $A \cup B$ are marked, gives us that invariant (I) holds after the phase, as needed – providing that the phase ends at all.

To complete the analysis of the adversary strategy we need to argue that phase k must actually end, in order for the D-values to represent correct distances from s. This follows directly from invariants (J1)-(J2.3), because they imply that before the very last step of the phase there is at least one vertex in Y_{n-1} with D-value at least $L - k + 1$, which is larger than its distance from s.

It now only remains to remove the assumption that the D-values are initialized to $L + 1$. According to our model, they need to be initialized to edge lengths from s, which are: $\ell(s, x_1) = 2$ and $\ell(s, v) = L$ for $v \neq x_1$ (since $s = x_0$). With this initialization, we only need to modify the first phase by marking all edges of the form (x_0, v) immediately. Invariant (J) then applies without further modification.

4 Lower Bound for Deterministic Algorithms with Three Types of Queries

In this section, we prove Theorem 2(a), an $\Omega(n^3)$ lower bound for deterministic algorithms using all three types of queries. Our argument is essentially a reduction – we show that any algorithm $\mathcal{A}$ that uses D-queries, weight queries, edge queries and relaxation updates can be converted into an algorithm $\mathcal{B}$ that has the same time complexity as $\mathcal{A}$ and uses only edge queries and relaxations. Our lower bound will then follow from Theorem 1(a).

We start with some initial observations that, although not needed for the proof, contain some useful insights. Since edge weights do not change, an algorithm can use weight queries to pre-sort all edges in time $O(n^2 \log n)$, and then it doesn't need to make any more weight queries during the computation. This way, the algorithm's running time is not affected as long as it's at least $\Omega(n^2 \log n)$. Similarly, the algorithm can use D-queries to maintain the total order of the D-values using, say, a binary search tree, paying a small overhead of $O(\log n)$ for each update operation. Then the algorithm's decisions at each step can as well depend on the total ordering of the vertices according to their current D-values. These changes will add at most an $O(\log n)$ factor to the running time.

Potential-oblivious model. Instead of working just with edge queries, we generalize our argument to *potential-oblivious* query models. We say that a query model $\mathbb{Q}$ is potential-oblivious if it satisfies the following property for each weight assignment ℓ and potential ϕ:

for any sequence of relaxations and queries from $\mathbb{Q}$ (with the D-values initialized as described in Section 2), the outcomes of the queries for weight assignments ℓ and $\ell + \Delta\phi$ are the same. By routine induction, any algorithm using a potential-oblivious model will perform the same sequence of queries and relaxations on assignments ℓ and $\ell + \Delta\phi$. Also, it will compute the correct distances on ℓ if and only if it will compute them for $\ell + \Delta\phi$, and in the same number of steps. (To see this, note that for each vertex v the invariant $D'[v] = D[v] + \phi(v)$ is preserved, where we use notations D and D' to distinguish between the D-values in the computations for ℓ and ℓ'. The respective distances $\ell(s, v)$ and $\ell'(s, v)$ satisfy the same equation.)

For example, the edge query only model is potential-oblivious. Due to our initialization and properties of relaxations, each value $D[v]$ always corresponds to the length of some path from s to v. Then the query is equivalent to comparing the length of two paths with the same start and end points, and the query outcome is the same after adding $\Delta\phi$, by the distance preservation property. Using these facts, potential-obliviousness follows from induction on the number of operations performed.

Golomb-ruler potential. For our proof, we need a potential function ϕ for which in the induced weight assignment $\Delta\phi$ all edge weights are different. (This naturally implies that all values of ϕ are also different.) Such an assignment is equivalent to a *Golomb ruler* (also known as a Sidon set), which is a set of non-negative integers with unique pair-wise differences. A simple Golomb ruler can be constructed using fast growing sequences, such as $\left\{2^i - 1\right\}_{i=0}^{n-1}$, but we are interested in sets contained in a small polynomial-in-n range. The asymptotic growth of Golomb rulers is well studied; it is known that there are n-element Golomb rulers that are subsets of $\{1, 2, ..., N\}$, for $N = n^2(1 + o(1))$ [10, 21], and that this bound on N is essentially optimal. Since the Golomb-ruler property is invariant under shifts, we can assume that a Golomb ruler contains number 0. For our purposes, this means that there exists a potential function ϕ that induces distinct edge weights with absolute maximum weight $O(n^2)$. ([4] shows that it is possible to obtain smaller maximum weights for certain classes of non-complete graphs, but this is not relevant to our constructions.) We will call this function a *Golomb-ruler potential*.

▶ **Theorem 3.** *Let $\mathcal{A}$ be a query/relaxation-based algorithm that uses relaxation updates, D-queries, weight queries and any queries from a potential-oblivious model $\mathbb{Q}$, and let $T(n)$ be the running time of $\mathcal{A}$. Then there is an algorithm $\mathcal{B}$ with running time $O(T(n))$ that uses only relaxation updates and queries from $\mathbb{Q}$.*

The idea of the proof is to convert a given weight assignment ℓ into another assignment ℓ' such that, if only queries from $\mathbb{Q}$ (and relaxations) are used, then (i) ℓ' is indistinguishable from ℓ using the queries from $\mathbb{Q}$, and (ii) in ℓ' the ordering of weights and the ordering of all D-values are *independent of* ℓ and, further, the ordering of the D-values is *fixed throughout the computation*, even though the D-values themselves may vary. $\mathcal{B}$ can do this conversion "internally" and simulate $\mathcal{A}$ on ℓ', and then it doesn't need to make any D-queries and weight queries, because their outcomes are predetermined.

Proof. Let $\mathcal{A}$ be a query/relaxation algorithm for that uses D-queries, weight queries, queries from $\mathbb{Q}$, and relaxation updates. We construct $\mathcal{B}$ that uses only queries from $\mathbb{Q}$ and relaxation updates. Let ϕ be the Golomb-ruler potential defined before the theorem. When run on a weight assignment ℓ, $\mathcal{B}$ will internally simulate $\mathcal{A}$ on weight assignment $\ell' = \ell + c\Delta\phi$, for $c = 2\ell_{\max}n + 1$. We use notation D' for the D-values computed by $\mathcal{A}$. The actions of $\mathcal{B}$ depend on the execution of $\mathcal{A}$ on ℓ', as follows:

- When $\mathcal{A}$ executes a weight query "$\ell'_{uv} < \ell'_{xy}$?", $\mathcal{B}$ directly executes the "yes" branch from the query if $\Delta\phi(u,v) < \Delta\phi(x,y)$, or the "no" branch otherwise.
- When $\mathcal{A}$ executes a D-query "$D'[u] < D'[v]$?", then $\mathcal{B}$ executes the "yes" branch if $\phi_u < \phi_v$, else it executes the "no" branch.

This simulation can be more formally described as converting the decision tree of $\mathcal{A}$ into the decision tree of $\mathcal{B}$. The tree of $\mathcal{B}$ is obtained by splicing out each node q representing a D-query or weight query. This splicing consists of connecting the parent of q to either the "yes" or "no" child of q, determined by the appropriate inequality involving ϕ, as explained above.

It remains to prove the correctness of $\mathcal{B}$. We argue first that $\mathcal{B}$ will produce correct distances if run on ℓ' instead of ℓ. For this, we observe that ℓ' satisfies the following properties:

(p1) For any two edges e, f, we have $\ell'_e < \ell'_f$ if and only if $\Delta\phi(e) < \Delta\phi(f)$.

(p2) For any three vertices u, x, y, any u-to-x path P_x and any u-to-y path P_y, we have $\ell'(P_x) < \ell'(P_y)$ if and only if $\phi_x < \phi_y$.

Indeed, both properties follow from the choice of c and straighforward calculation. For (p1), $\ell'_e < \ell'_f$ if and only if $\ell_e - \ell_f < c[\Delta\phi(f) - \Delta\phi(e)]$, and because $|\ell_e - \ell_f| < c$ this inequality is determined by the sign of $\Delta\phi(f) - \Delta\phi(e)$, which is always non-zero, by the Golomb-ruler property. (Note that here we only use that $c > 2\ell_{\max}$.) The justification for (p2) is similar: we have $\ell'(P_x) = \ell(P_x) + c(\phi_x - \phi_u)$ and $\ell'(P_y) = \ell(P_y) + c(\phi_y - \phi_u)$, so $\ell'(P_x) < \ell'(P_y)$ if and only if $\ell(P_x) - \ell(P_y) < c[\phi_y - \phi_x]$, and since $|\ell(P_x) - \ell(P_y| < c$ this inequality is determined by the sign of $\phi_y - \phi_x$.

Properties (p1) and (p2) imply that when we run $\mathcal{A}$ on ℓ', in each weight query we can equivalently use assignment $\Delta\phi$ instead of ℓ', and instead of using a D-query we can compare the corresponding potential values. Therefore $\mathcal{B}$ works correctly for ℓ'. But since now $\mathcal{B}$ uses only relaxations and queries from $\mathbb{Q}$, that are potential-oblivious, and ℓ' is obtained from ℓ by adding a weight assignment induced by potential $c\phi$, $\mathcal{B}$'s computation on ℓ will also be correct. ◀

Theorem 3, together with Theorem 1 implies the $\Omega(n^3)$ lower bound for query/update-based algorithms that use D-queries, weight queries, any set of potential-oblivious queries, and relaxation updates. Since the edge update is potential oblivious, Theorem 2(a) follows.

Further, using the construction from Theorem 2(a), where a weight assignment with maximum weight $O(n)$ was used, the proof of Theorem 3 shows that Theorem 2(a) holds even if all weights are bounded by $O(n^4)$.

A side result for general graphs. The reduction in the proof of Theorem 3 extends naturally to arbitrary graphs. In particular, we can extend a result from Eppstein [9]:

▶ **Theorem 4.** *For any n and m where $n \le m \le n(n-1)$, there exists a graph with n nodes and m edges where any deterministic algorithm $\mathcal{A}$ using D-queries, weight queries, and relaxation updates has worst-case running time $\Omega(nm/\log n)$. If $m = \Omega(n^{1+\varepsilon})$ for some $\varepsilon > 0$, the lower bound can be improved to $\Omega(nm)$.*

Proof sketch. We focus on the case where m is arbitrary. First note that the model using no queries and relaxation updates is equivalent to non-adaptive algorithms (that is, universal relaxation sequences) described in [9]. (It's also obvious that the query model using no queries is potential-oblivious.) Let G be the graph construction described in the proof of [9, Theorem 3]. In particular, G has n nodes and m edges, and for every non-adaptive algorithm on G, there is weight assignment that forces $\Omega(nm/\log n)$ relaxations. If there exists an

algorithm $\mathcal{A}$ for G using D-queries, weight queries, and relaxation updates that runs in $T(n)$ time, then by the same construction as in Theorem 3, there also exists an algorithm $\mathcal{B}$ using only relaxation updates that runs in time $O(T(n))$. Then an $o(nm/\log n)$ running time on G would contradict the lower-bound on non-adaptive algorithms. The proof for the case $m = \Omega(n^{1+\varepsilon})$ is identical. ◀

As explained in Section 5, the reduction also applies to randomized algorithms, and because [9] proves the same lower bounds for expected running time for randomized non-adaptive algorithms, the above bounds also apply to the randomized case.

5 Lower Bounds for Randomized Algorithms

In this section, we extend the proofs in Sections 3 and 4 to obtain $\Omega(n^3)$ lower bounds for randomized algorithms, proving Theorem 1(b) and Theorem 2(b). The proofs are based on Yao's principle [22]: we give a probability distribution on weight assignments for which the expectation of each deterministic algorithm's running time is $\Omega(n^3)$.

We fix the value of n. Let $\ell_{\max}$ be the maximum absolute value of weights used in the proof, whose value will be specified later. Let $\mathbb{L}$ be the family of all weight assignments $\ell : E \to [-\ell_{\max}, \ell_{\max}]$. Denote by $\mathbb{A}$ the (finite) set of all deterministic query/relaxation-based algorithms with running time at most $2n^3$. We only need to consider algorithms in $\mathbb{A}$, because any other algorithm in our model can be modified to run in time at most $2n^3$. To see why, consider this algorithm's decision tree. For any node at depth n^3, replace its subtree by the Bellman-Ford relaxation sequence. The resulting tree remains correct, and its depth is at most $2n^3$.

For a deterministic algorithm $\mathcal{A} \in \mathbb{A}$ and weight assignment $\ell \in \mathbb{L}$, denote by $T(\mathcal{A}, \ell)$ the running time of $\mathcal{A}$ on assignment ℓ. Let $\Pi(\mathbb{A})$ be the set of all probability distributions on $\mathbb{A}$ and $\Pi(\mathbb{L})$ be the set of all probability distributions on $\mathbb{L}$. Any randomized algorithm $\mathcal{R}$ is simply a probability distribution on $\mathbb{A}$, so $\mathcal{R} \in \Pi(\mathbb{A})$. Denote by $\mathrm{Exp}_{x\sim\theta} f(x)$ the expected value of $f(x)$, for a random variable x from distribution θ. The lemma below is a restatement of Yao's principle [22] in our context:

▶ **Lemma 5.** *The following equality holds:*

$$\min_{\mathcal{R}\in\Pi(\mathbb{A})} \max_{\ell\in\mathbb{L}} \mathrm{Exp}_{\mathcal{A}\sim\mathcal{R}} T(\mathcal{A}, \ell) \;=\; \max_{\sigma\in\Pi(\mathbb{L})} \min_{\mathcal{A}\in\mathbb{A}} \mathrm{Exp}_{\ell\sim\sigma} T(\mathcal{A}, \ell).$$

In this lemma, both sides involve the expected running time, with the difference being that on the left-hand side we consider randomized algorithms and their worst-case inputs, while the right-hand side involves the probability distribution on *input permutations* that is worst for *deterministic algorit hms*.

Proof of Theorem 1(b) (Sketch).[1] We give a probability distribution σ on weight assignments ℓ for which every deterministic algorithm needs $\Omega(n^3)$ steps in expectation to compute correct distances. This is sufficient, as then Lemma 5 implies that each randomized algorithm $\mathcal{R}$ makes $\Omega(n^3)$ steps in expectation on some weight assignment.

[1] A detailed proof will appear in the full version of this paper.

Recall that in the proof of Theorem 1(a) in Section 3 we used weight assignments associated with permutations of vertices. This is also the case here, although this assignment needs to be modified. For any permutation $\pi = x_0, x_1, ..., x_{n-1}$ of the vertices, the corresponding weight assignment is

$$\ell_\pi(x_i, x_j) = \begin{cases} n & \text{if } j = i+1 \\ L - (n + \frac{1}{2})i & \text{if } j \geq i+2 \text{ and } i \text{ is even} \\ L & \text{if } j \geq i+2 \text{ and } i \text{ is odd} \end{cases}$$

where L is sufficiently large, say $5n^2$. (We explain later why larger weights are necessary.) In our argument here, the adversary chooses the uniform distribution σ on all $(n-1)!$ permutations π starting with s.

In a certain sense, our goal now is simpler than in Section 3, as the adversary's job, which is to choose σ, is already done. We "only" need to lower bound the expected running time of algorithms from $\mathbb{A}$ if the weights are distributed according to σ. The challenge is that this argument needs to work for *an arbitrary algorithm* from $\mathbb{A}$.

So fix any deterministic algorithm $\mathcal{A} \in \mathbb{A}$. We need to prove that $\text{Exp}_{\ell\sim\sigma} T(\mathcal{A}, \ell) = \Omega(n^3)$. A high-level approach in our proof is similar to the proof in Section 3: we partition the computation of $\mathcal{A}$ into $(n-1)/2$ phases, and show that the expected length of each phase $k = 1, 2, ..., (n-1)/2$ is $\Omega((n-2k)^2)$.

For a specific permutation $\pi = x_0, x_1, ..., x_{n-1}$ with $x_0 = s$ and $k = 1, 2, ..., (n-1)/2$, let $t_k(\pi)$ be the first time step such that in steps $1, 2, ..., t_k(\pi)$ the edges $(x_0, x_1), ..., (x_{2k-1}, x_{2k})$ have been accessed by $\mathcal{A}$ (that is, relaxed or queried) in this particular order. We refer to the time interval $(t_{k-1}(\pi), t_k(\pi)]$ as *phase k for permutation π*.

The proof idea is this: In each phase k of the strategy in Section 3 the adversary was able to force the algorithm to relax all edges from x_{2k-2} to Y before revealing edge (x_{2k-1}, x_{2k}), thus ensuring that at all times all D-values differ at most by 1. This is not possible anymore, because now the algorithm can get "lucky" and relax edges (x_{2k-2}, x_{2k-1}) and (x_{2k-1}, x_{2k}) before all edges (x_{2k-2}, u) for $u \in Y$ are relaxed, and then the D-values for such vertices u will reflect the relaxations that occurred in some earlier phases. But we can still bound the differences between D-values. Namely, by our choice of the length function above, any two D-values will differ by at most $n/2$. Thus, since all edge lengths are at least n, the negative answers to all edge queries inside Y are still correct, independently of the suffix $x_{2k-1}, ..., x_{n-1}$ of π.

More specifically, the analysis is based on establishing two invariants, captured by the claim below (formal proof omitted here).

▷ Claim 6. The following invariants are satisfied when each phase k starts:

(R1) The computation of $\mathcal{A}$ up until phase k starts is independent of the suffix $x_{2k-1}, x_{2k}, ..., x_{n-1}$ of π.

(R2) The D-values have the following form: $D[x_j] = jn$ for $j \leq 2k-2$, and $D[x_j] \in [L-k+1, L]$ for $j \geq 2k-1$.

Next, define $\tilde{t}_k$ to be a random variable whose values are $t_k(\pi)$ for permutations π distributed randomly according to σ. We refer to the time interval $(\tilde{t}_{k-1}, \tilde{t}_k]$ as *phase k*, and let $\partial t_k = \tilde{t}_k - \tilde{t}_{k-1}$ be the random variable equal to the length of this phase.

We then prove the following claim:

▷ Claim 7. $\text{Exp}_{\ell\sim\sigma}[\partial t_k] \geq \frac{1}{2}(n-2k+1)(n-2k+2)$.

Let $\bar{z} = z_0, z_1, ...z_{2k-2}$ be some fixed $(2k-1)$-permutation of V with $z_0 = s$. Let H be the event that π starts with $\bar{z}$. It is sufficient to prove the inequality in Claim 7 for the conditional expectation $\mathrm{Exp}_{\ell\sim\sigma}[\partial t_k|H]$.

So assume that event H is true. Let $Y = V \setminus \{z_0, ..., z_{2k-2}\}$. Now the argument is this: The suffix $x_{2k-1}, ..., x_{n-1}$ of π is a random permutation of Y and the edge (x_{2k-1}, x_{2k}) is uniformly distributed among the edges in Y. Algorithm $\mathcal{A}$ is deterministic and all edge queries for edges inside Y, except for edge (x_{2k-1}, x_{2k}) (and only if (x_{2k-2}, x_{2k-1}) has already been relaxed), will have negative answers. Similarly, all queries to edges from x_{2k-2} to Y will have positive answers. So $\mathcal{A}$ will be accessing these edges in some order that is uniquely determined by the state of $\mathcal{A}$ when phase k starts. Since there are $(n-2k+1)(n-2k+2)$ edges in Y, this implies that on average it will take $\frac{1}{2}(n-2k+1)(n-2k+2)$ steps for $\mathcal{A}$ to access (x_{2k-1}, x_{2k}), even if we don't take into account that (x_{2k-2}, x_{2k-1}) needs to be accessed first. This will imply Claim 7.

We now continue the proof of Theorem 1(b). For the algorithm to be correct, if the chosen permutation π is $x_0, x_1, ..., x_{n-1}$, then the algorithm needs to relax the edges on this path in order as they appear on the path. So its running time is at least $t_{(n-1)/2}(\pi)$. Since $\tilde{t}_{(n-1)/2} = \sum_{k=1}^{(n-1)/2} \partial t_k$, using Claim 7 and applying the linearity of expectation we obtain that $\mathrm{Exp}_{\ell\sim\sigma} T(\mathcal{A}, \ell) \geq \mathrm{Exp}_{\ell\sim\sigma}[\tilde{t}_{(n-1)/2}] = \sum_{k=1}^{(n-1)/2} \mathrm{Exp}_{\ell\sim\sigma}[\partial t_k] = \Omega(n^3)$, completing the proof. ◀

Proof of Theorem 2(b). There is not much to prove here, because the reduction described in Section 4 applies with virtually no changes to randomized algorithms. Indeed, just like in the proof of Theorem 2(a) (or more specifically the proof of Theorem 3), suppose that $\mathcal{R}$ is a randomized algorithm that uses all three types of queries: D-queries, weight-queries, the queries from model $\mathbb{Q}$, as well as relaxation updates, and let $T(n)$ be $\mathcal{R}$'s expected running time. We can convert $\mathcal{R}$ into a randomized algorithm $\mathcal{R}'$ with running time $O(T(n))$ that uses only the queries from $\mathbb{Q}$ and relaxation updates. With this, Theorem 2(b) follows from Theorem 1(b). ◀

High-probability bounds. Using standard reasoning (see [9], for example), our lower bound results for expectation imply respective high-probability bounds, namely that there are no randomized algorithms in the models from Theorems 1 and 2 that compute correct distance values in time $o(n^3)$ with probability at least $1 - o(1)$.

To justify this, suppose that $\mathcal{R}$ is a randomized algorithm that computes correct distance values in time $T(n) = o(n^3)$ with probability $1 - o(1)$. Consider the algorithm $\mathcal{R}'$ obtained from $\mathcal{R}$ by switching to the Bellman-Ford relaxation sequence right after step $T(n)$. The expected running time of $\mathcal{R}'$ is then at most $T(n) + o(1)n^3 = o(n^3)$, but this would contradict our lower bounds in Theorems 1(b) and 2(b).

6 Final Comments and Open Problems

Our reduction in Section 4 introduces negative weights, raising a natural question: *Is it possible to use $o(n^3)$ relaxations with only weight-queries for instances with non-negative weights?* (This question is of purely theoretical interest, because $O(n^2)$ relaxations can be achieved, using Dijkstra's algorithm, if D-queries are used instead.) Our proof techniques do not work for this variant. The reason is, in the instances we construct the shortest-path tree is a Hamiltonian path, and for such instances this path can be uniquely determined by the weight ordering: start from s, and at each step follow the shortest outgoing edge from the

current vertex to a yet non-visited vertex. So only $n-1$ relaxations are needed. It is unclear what is the "hard" weight ordering in this case. It can be shown that in the two extreme cases: (i) if the weight orderings of outgoing edges from each vertex are agreeable (that is, they are determined by a permutation of the vertices), or (ii) if they are random, then there is a relaxation sequence of length only $O(n^{2.5})$ (and this likely can be improved further).

A natural extension of our query/relaxation model would be to allow *unconditional edge updates* of the form $D[v] \leftarrow D[u] + \ell_{uv}$. A combination of such edge updates and D-queries allows an algorithm to check for properties that are impossible to test if only relaxation updates are used. For example, by applying edge updates repeatedly around cycles, such an algorithm would be able to determine, for any given rational number c, whether one cycle is at least c times longer than some other cycle.

A more open-ended question is to determine if there are simple types of queries, say some linear inequalities involving weights and the D-values (with a constant number of variables), that would be sufficient to yield an adaptive algorithm (possibly randomized) that makes $o(n^3)$ relaxations.

The case of random universal sequences is also not fully resolved. While it is known that the asymptotic bound is $\Theta(n^3)$, there is a factor-of-4 gap for the leading constant, between $\frac{1}{12}$ and $\frac{1}{3}$ [9, 1].

We remark that our proofs are somewhat sensitive to the initialization of the D-values. Recall that in our model we assume that initially $D[v] = \ell_{sv}$ for $v \neq s$. This is natural, and it guarantees that at all times the D-values represent lengths of paths from s. It also has the property of being language- and platform-independent. However, some descriptions of shortest-path algorithms initialize the D-values to infinity, or some very large number. The proof of Theorem 1 in Section 3 can be modified to work if the D-values were initialized to some sufficiently large value M (the adversary can then use $L = M - 1$ in her strategy). However, then the edge lengths are no longer polynomial, and the proof of Theorem 2 in Section 4 does not apply in its current form. Initializing to infinity would also affect the proofs. The reduction in Section 4 can be modified to account for infinite D-values, but we don't know how to adapt the proof in Section 3 to this model. We leave open the problem of finding a more "robust" lower bound proof, that works for an arbitrary valid initialization and uses only polynomial weights.

References

1. Michael J. Bannister and David Eppstein. Randomized speedup of the Bellman-Ford algorithm. In *Proceedings of the 9th Meeting on Analytic Algorithmics and Combinatorics, ANALCO 2012*, pages 41–47. SIAM, 2012. doi:10.1137/1.9781611973020.6.
2. Richard Bellman. On a routing problem. *Quart. Appl. Math.*, 16:87–90, 1958.
3. Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *Proceedings of the 63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 600–611, 2022. doi:10.1109/FOCS54457.2022.00063.
4. Béla Bollobás and Oleg Pikhurko. Integer sets with prescribed pairwise differences being distinct. *European Journal of Combinatorics*, 26(5):607–616, 2005. doi:10.1016/J.EJC.2004.04.008.
5. Gang Cheng and Nirwan Ansari. Finding all hops shortest paths. *IEEE Commun. Lett.*, 8(2):122–124, 2004. doi:10.1109/LCOMM.2004.823365.
6. Michael B. Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\tilde{O}(m^{10/7} \log W)$ time (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 752–771, 2017. doi:10.1137/1.9781611974782.48.

7 Narsingh Deo and Chi-Yin Pang. Shortest-path algorithms: Taxonomy and annotation. *Networks*, 14(2):275–323, 1984. doi:10.1002/NET.3230140208.

8 Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. doi:10.1007/BF01386390.

9 David Eppstein. Lower bounds for non-adaptive shortest path relaxation. In *Proceedings of the 18th International Symposium on Algorithms and Data Structures, WADS 2023*, pages 416–429, 2023. doi:10.1007/978-3-031-38906-1_27.

10 P. Erdös and P. Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, s1-16(4):212–215, 1941.

11 Jeremy T. Fineman. Single-source shortest paths with negative real weights in $\tilde{O}(mn^{8/9})$ time. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 3–14, 2024. doi:10.1145/3618260.3649614.

12 L. R. Ford. *Network Flow Theory*. RAND Corporation, Santa Monica, CA, 1956.

13 Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995. doi:10.1137/S0097539792231179.

14 Roch Guérin and Ariel Orda. Computing shortest paths for any number of hops. *IEEE/ACM Trans. Netw.*, 10(5):613–620, 2002. doi:10.1109/TNET.2002.803917.

15 Jialu Hu and László Kozma. Non-adaptive Bellman-Ford: Yen's improvement is optimal. *CoRR*, abs/2402.10343, 2024. arXiv:2402.10343, doi:10.48550/arXiv.2402.10343.

16 Stasys Jukna and Georg Schnitger. On the optimality of Bellman-Ford-Moore shortest path algorithm. *Theor. Comput. Sci.*, 628:101–109, 2016. doi:10.1016/J.TCS.2016.03.014.

17 Tomasz Kociumaka and Adam Polak. Bellman-Ford is optimal for shortest hop-bounded paths. In *Proceedings of the 31st Annual European Symposium on Algorithms, ESA 2023*, pages 72:1–72:10, 2023. doi:10.4230/LIPICS.ESA.2023.72.

18 Ulrich Meyer, Andrei Negoescu, and Volker Weichert. New bounds for old algorithms: On the average-case behavior of classic single-source shortest-paths approaches. In *Proceedings of the First International ICST Conference on Theory and Practice of Algorithms in (Computer) Systems, TAPAS 2011*, pages 217–228, 2011. doi:10.1007/978-3-642-19754-3_22.

19 E. F. Moore. The shortest path through a maze. In *Proceedings of an International Symposium on the Theory of Switching, Part II*, pages 285–292, 1959.

20 A. Shimbel. Structure in communication nets. In *Proceedings of the Symposium on Information Networks*, pages 199–203. Polytechnic Press of the Polytechnic Institute of Brooklyn, 1955.

21 James Singer. A theorem in finite projective geometry and some applications to number theory. *Trans. Amer. Math. Soc.*, 43(3):377–385, 1938.

22 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977. doi:10.1109/SFCS.1977.24.

23 Y. Yen. *Shortest Path Network Problems*, volume 18 of *Mathematical Systems in Economics*. Verlag Anton Hain, Meisenheim am Glan, 1975.