Black-box Methods for Restoring Monotonicity

Evangelia Gergatsouli [1] **Brendan Lucier** [2] **Christos Tzamos** [1]

Abstract

In many practical applications, heuristic or approximation algorithms are used to efficiently solve the task at hand. However their solutions frequently do not satisfy natural monotonicity properties of optimal solutions. In this work we develop algorithms that are able to restore monotonicity in the parameters of interest. Specifically, given oracle access to a (possibly non-monotone) multi-dimensional real-valued function f, we provide an algorithm that restores monotonicity while degrading the expected value of the function by at most ε. The number of queries required is at most logarithmic in $1/\varepsilon$ and exponential in the number of parameters. We also give a lower bound showing that this exponential dependence is necessary. Finally, we obtain improved query complexity bounds for restoring the weaker property of k-marginal monotonicity. Under this property, every k-dimensional projection of the function f is required to be monotone. The query complexity we obtain only scales exponentially with k.

1. Introduction

"You are preparing a paper for an upcoming deadline and try to fit the content within the page limit. You identified a redundant sentence and remove it but to your surprise, the page count of your paper increases!"

This is an example where a natural monotonicity property expected from the output fails to hold, resulting in unintuitive behavior. As another example, consider the knapsack problem where given a collection of items with values and weights the goal is to identify the most valuable subset of items with total weight less than W. Now if the capacity of the knapsack increases, the new set of items about to be selected are expected to be at least as valuable as before. While such a monotonicity property holds under the optimal solution, when heuristic or approximate algorithms are used, monotonicity can often fail.

There are also numerous cases in ML applications that such a behaviour is desired. For example in hyperparameter tuning it is expected that the error will reduce when the complexity of the model increases, which is not always the case. Additionally, monotone classification and isotonic regression require the dataset to be monotone, and since this is not always possible, they usually handle by changing the dataset. The method we propose could act as a filter that provides the learning algorithm with monotone points from the dataset in a black box way.

In this work we develop tools to restore monotonicity in a black-box way. Our goal is to create a meta-algorithm that is guaranteed to be monotone, while querying a provided (possibly non-monotone) algorithm behind the scenes. For example, such a meta-algorithm might query the provided oracle at many different inputs in an attempt to smooth out non-monotonicities.

More precisely, we can describe the output of a (possibly non-monotone) algorithm using a function $f : \mathbb{R}^d \to [0, 1]$ that measures the quality of the solution at any given point $x \in \mathbb{R}^d$. We assume that inputs are drawn from a known product distribution, and that there is an (unknown) feasibility condition being satisfied by the function f. Since f may not initially be monotone, we want to correct it through our meta-algorithm while additionally maintaining the following three properties: it needs to be query efficient, feasible and comparable to the original algorithm in terms of expected solution quality. To ensure feasibility, at any given point x we allow outputting any solution that corresponds to some smaller input $y \preceq x$ (coordinate-wise), thus achieving quality $f(y)$ at input x. The exact constraints of the initial algorithm are unknown, and we have to infer them using f, in a black-box way. We want our meta-algorithm to do this in a way that the resulting quality function $\tilde{f} : \mathbb{R}^d \to [0, 1]$ is monotone but also have expected quality $\mathbb{E}[\tilde{f}(x)]$ that is comparable to the expected quality of the original algorithm, $\mathbb{E}[f(x)]$ with respect to the input distribution. We finally require that our meta-algorithm is query efficient, meaning

[*]Equal contribution [1]University of Wisconsin - Madison, WI, USA [2]Microsoft Research, Cambridge, MA, USA. Correspondence to: Evangelia Gergatsouli <evagerg@cs.wisc.edu>, Brendan Lucier <brlucier@microsoft.com>, Christos Tzamos <tzamos@wisc.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

that it does not require querying too many points of the original function in order to return an answer for a given point.

One natural way of solving this problem is to pick for every x the best answer that corresponds to any input $y \preceq x$, resulting in $\tilde{f}(x) = \max_{y \preceq x} f(y)$. While this ensures that solutions are monotone and that the expected quality is always better than before, it requires a large number of queries to identify the best possible answer among inputs $y \preceq x$. A more query efficient strategy would be to always output a constant solution that corresponds to the smallest possible solution. While this is monotone, feasible and uses few queries, it has very low expected quality.

1.1. Results

Our main result is stated informally below:

Theorem 1. *There is a meta-algorithm $\mathcal{M}_f$ that corrects the monotonicity of any given function f through queries. The meta-algorithm is feasible and has expected quality loss ε under a given product distribution of inputs. The total (expected) number of queries required for every input is at most $O(\log(d/\varepsilon))^d$.*

Our meta-algorithm starts by discretizing the space to a fine grid of d/ε points. We show that this step incurs at most an ε penalty in expected function value. Given this discretization it is straightforward to correct monotonicity by querying all the points in the grid at a cost of $(d/\varepsilon)^d$ and for any x returning the best solution for any smaller input.

We remark that our meta-algorithm is significantly more efficient than this naive approach, achieving number of queries that is at most poly-logarithmic in $1/\varepsilon$. It is able to obtain this speedup by searching over a hierarchical partition of the space to efficiently determine which value to assign to the given input at query time, rather than preprocessing all the answers. Additionally, our algorithm is *local* and computes answers on the fly without requiring any precomputation and without the need to remember any prior queries to answer the future ones.

We also note that our algorithm depends exponentially on the number of input parameters d. This means that while the algorithm is extremely efficient for small d, the savings become less significant when d grows. Such an exponential dependence in the number of parameters, however, is unavoidable for correcting monotonicity. As we show, for any black-box scheme that aims to correct monotonicity, there are always some problems where either it would fail to be feasible or monotone, or it would require exponentially many queries to calculate the appropriate answers.

Theorem 2. *Let $\mathcal{M}$ be any feasible meta-algorithm for monotonicity with query complexity $q = 2^{o(d)}$. Then, there exists an input function $f : \{0,1\}^d \to \{0,1\}$ with $\mathbb{E}[f(x)] \geq 1 - 2^{-\Omega(d)}$ such that $\mathbb{E}[\mathcal{M}_f(x)] \leq 2^{-\Omega(d)}$.*

Theorem 2 shows that ensuring monotonicity when d is large can be quite costly, either in queries or the quality of the solution. To better understand this tradeoff, we consider a weakening of monotonicity, namely k-marginal monotonicity, where we only require monotonicity of the k-dimensional projections of the function to be monotone. That is, when $k = 1$, we want that for any coordinate $i \in [d]$, the function $\mathbb{E}[\mathcal{M}_f(x)|x_i]$ to be monotone with respect to x_i. For larger $k > 1$, we want that for any subset $\mathcal{I} \subseteq [d]$ of size $|\mathcal{I}| \leq k$, the function $\mathbb{E}[\mathcal{M}_f(x)|x_{\mathcal{I}}]$ is monotone with respect to x_S. For this setting, we show that:

Theorem 4. *There is a meta-algorithm $\mathcal{M}_f$ that corrects the k-marginal monotonicity of any given function f through queries. The meta-algorithm is feasible and has expected quality loss ε under a given product distribution of inputs. The total (expected) number of queries required for every input is at most $\left(\frac{d}{\varepsilon}\right)^{O(k)}$.*

Note that when requiring marginal monotonicity, the dependence on d is improved from exponential to polynomial. Instead, the query complexity is only exponential in k. It is an interesting open question whether this can be improved.

1.2. Related work

A very related line of work to our paper considers the problem of online property-preserving data reconstruction. In this framework introduced by Ailon et al. (Ailon et al., 2008) there is a set of unreliable data that should satisfy a certain structural property, like monotonicity or convexity. The reconstruction algorithm acts as a filter for this data, such that whatever query the user makes on them is answered in a way consistent with the property they should satisfy. Ailon at al. (Ailon et al., 2008) proposed the first algorithm for monotonicity reconstruction, and in the follow-up work, Saks and Seshadhri (Saks & Seshadhri, 2010) designed a more efficient and local algorithm for the same problem. The main focus of this work is to compute a function that is not different than the original function in a large number of inputs. In comparison, we allow our algorithm to output arbitrary solutions at any point subject to a feasibility criterion and consider the expected quality of the solution as a measure of performance.

In addition to these works on upper bounds on monotonicity reconstruction, Bhattacharyya et al. in (Bhattacharyya et al., 2012) proved a lower bound for local monotonicity reconstruction of Saks and Seshadhri using transitive-closure spanners. Several reconstruction algorithms have also been proposed for reconstructing Lipschitzness (Jha & Raskhodnikova, 2013), convexity (Chazelle & Seshadhri, 2011), connectivity of directed or undirected graphs (Campagna et al., 2013), or a given hypergraph property (Austin & Tao,

2010), while (Chakraborty et al., 2014) focuses on lower bounds.

Our work can also be viewed as a special case of the *Local Computation Algorithms* framework introduced by (Rubinfeld et al., 2011) and (Alon et al., 2012). In this model the algorithm is required to answer queries that arrive online such that the answers given are always consistent with a specific solution. The algorithms presented also do not need to remember old answers to remain consistent, exactly as our algorithms do. Such local algorithms have been designed for many problems like maximal independent set maximal matching, approximate maximum matching, set cover, vertex coloring, hypergraph coloring (Alon et al., 2012; Mansour & Vardi, 2013; Levi et al., 2014; Even et al., 2014; Levi et al., 2016; 2017; Parter et al., 2019; Ghaffari & Uitto, 2019; Grunau et al., 2020).

The framework of *sampling correctors*, introduced by Canonne et al in (Canonne et al., 2016) is also relevant to our work. One important difference is that even if we set the required property in their framework to be monotonicity (to match our problem) they only have access to a non-monotone distribution while we have query access to a function.

Finally, our work is also closely related to the black-box reductions literature in algorithmic mechanism design, where we are given access to an algorithmic solution - an oracle- and the goal is to implement an incentive compatible mechanism with similar performance. Ensuring incentive compatibility amounts to preserving a similar monotonicity property, like cyclic monotonicity. This line of work was initiated by Hartline and Lucier (Hartline & Lucier, 2015), and later several reductions were given under different solution concepts, approximate or exact Bayesian Incentive Compatibility (Bei & Huang, 2011; Hartline et al., 2015; Dughmi et al., 2017; Gergatsouli et al., 2019) along with a lower bound for Dominant Strategy Incentive Compatibility (Chawla et al., 2012).

2. Preliminaries

We are given oracle access to a function $f : \mathbb{R}^d \to [0, 1]$, and a stream of input points $x \in \mathbb{R}^d$ for which we need to evaluate f. Our goal is to give an answer $\mathcal{M}_f$ for every point such that it satisfies some notion of monotonicity and it also has the following properties

1. Feasibility: $\mathcal{M}_f(x) \leq \max_{y \preceq x} f(y)$
2. Close in expectation to the initial function: $\mathbb{E}[\mathcal{M}_f(x))] \geq \mathbb{E}[f(x)] - \varepsilon$

Evaluation We evaluate our algorithms on their query complexity. Query complexity is defined as the maximum number of times we invoke the oracle f in order to give us an answer and the goal is to invoke the oracle as few times as possible.

Distributional assumptions We assume that each coordinate x_i of every point x in the input sequence is drawn according to a distribution $\mathcal{D}_i$. We denote the product of these $\mathcal{D}_i$ by $\mathcal{D} \in \Delta(\mathbb{R}^d)$. We want $\mathbb{E}_{x\sim\mathcal{D}}[f(x)]$ to be close to the expectation of the transformed f which we denote by $\mathcal{M}_f$. From now on, we will omit the $x \sim \mathcal{D}$ when it is clear from the context. It is worth noting that our results are robust to some noise in the distribution.

Monotonicity We proceed by defining the various notions of monotonicity we will be using throughout this work. For $x, y \in \mathbb{R}^d$ we say that $x \preceq y$ when $x_i \leq y_i$ for all $i \in [d]$. We say that the function f is monotone if $x \preceq y$ implies $f(x) \leq f(y)$ for every $x, y \in \mathbb{R}^d$.

A relaxation of the monotonicity requirement is that of *marginal* monotonicity. We say that f is *marginally* monotone if $f_i(x_i) \triangleq \mathbb{E}_{x_{-i}\sim\mathcal{D}}[f(x_i, x_{-i})]$ is monotone, where by x_{-i} we denote the vector of all coordinates except the i'th. Note that this is weaker than monotonicity since even if all marginals are monotone, this does not imply that f is also monotone.

A further relaxation of the marginal monotonicity is the k-marginal monotonicity. Similarly to k-wise independent variables, we say that a function is k-marginally monotone when for every $\mathcal{I} \subseteq [d]$ such that $|\mathcal{I}| \leq k$ the function

$$f_{\mathcal{I}}(x_{\mathcal{I}}) \triangleq \mathbb{E}_{x_{-\mathcal{I}}}[f_{\mathcal{I}}(x_{\mathcal{I}}, x_{-\mathcal{I}})]$$

is monotone, where we denote by $x_{\mathcal{I}}$ (or $x_{-\mathcal{I}}$) the size k-vector of all the coordinates i that also belong (or do not belong) to the set $\mathcal{I}$, and by $f_{\mathcal{I}}(x_{\mathcal{I}})$ the marginal of the coordinates $i \in \mathcal{I}$ that gets as input a vector of size $|\mathcal{I}|$ with the coordinates $x_{\mathcal{I}}$.

Discretization of the Domain In all the algorithms described in the following sections, we assume the domain is discrete. To do that we use the discretization process described below. Intuitively, we split the domain into smaller intervals with equal probability and then "shift" every coordinate's distribution downward by sampling a point from the lower interval. This is made more formal here, we first describe the process for the single dimensional case ($d = 1$), and then for general d.

We convert the oracle f to an oracle for a piecewise constant function $\widetilde{f}$ with $1/\varepsilon$ pieces. The function $\widetilde{f}$ is such that $\mathbb{E}[\widetilde{f}(x)] \geq \mathbb{E}[f(x)] - \varepsilon$, and $\widetilde{f}(x) \leq \max_{y\preceq x} f(y)$.

For this purpose, we split the support of the distribution into $m = \frac{1}{\varepsilon}$ intervals $I_1, ..., I_m$ such that each has probability ε, i.e. for every $i \in [m]$, $\int_{x\in I_i} \mathcal{D}(x) = \varepsilon$. Then, for each

interval I_i we draw a random x_i according to the conditional distribution $\mathcal{D}|I_i$. For any $x \in I_i$, we set

$$\widetilde{f}(x) = \begin{cases} 0 & i = 1 \\ f(x_{i-1}) & i > 0 \end{cases}.$$

It is easy to see that $\widetilde{f}(x) \leq \max_{y \preceq x} f(y)$ since $\widetilde{f}(x)$ is either 0 or equal to $f(x_{i-1})$ where $x_i \leq x$. To bound the expectation, we note that

$$\begin{aligned}
\mathbb{E}_{x\sim\mathcal{D}}[\widetilde{f}(x)] &= \sum_{i=1}^{m-1} \varepsilon \mathbb{E}_{x_i \sim \mathcal{D}|I_i}[f(x_i)] \\
&= \sum_{i=1}^{m} \varepsilon \mathbb{E}_{x_i \sim \mathcal{D}|I_i}[f(x_i)] - \varepsilon \mathbb{E}_{x_m \sim \mathcal{D}|I_m}[f(x_m)] \\
&= \mathbb{E}_{x\sim\mathcal{D}}[f(x)] - \varepsilon \mathbb{E}_{x_m \sim \mathcal{D}|I_m}[f(x_m)] \\
&\geq \mathbb{E}_{x\sim\mathcal{D}}[f(x)] - \varepsilon
\end{aligned}$$

For $d > 1$ dimensions the above process is essentially the same for every coordinate with the difference that now we choose $m = d/\varepsilon$ points and every interval I_{ik} for $k \in [m]$ of coordinate i has probability ε/d. Let the input vector be $\boldsymbol{y} = (y_1, y_2, \ldots, y_d)$. For every coordinate $y_i \in I_{ik}$, we draw a random z_i from the conditional $\mathcal{D}|I_{i(k-1)}$ and finally evaluate f at these points $\boldsymbol{z} = (z_1, z_2, \ldots, z_d)$. Note that this is feasible, similarly to before, as each input returns an outcome generated by f on a pointwise smaller input. This perturbation effectively shifts each coordinate's distribution downward, removing the range I_{im}, therefore removing from the expectation any input that had at least one coordinate in the last interval. This occurs with probability at most $1 - \left(1 - \frac{\varepsilon}{d}\right)^d \leq \varepsilon$, and therefore the new expectation is not ε far from the old.

Assuming a discrete domain, where every coordinate $x_i \in \{w_{i1}, w_{i2}, \ldots, w_{im}\}$, we define the low 1-neighborhood of a point as the set of points that are smaller in exactly 1 coordinate. Formally, the low 1-neighborhood of a point y is $\mathcal{N}(y) = \{x : \exists! k \in [m]$ such that $x_k = w_{ik}$ and $y_k = w_{k(i+1)}$ for some $i \in [m-1]\}$.

3. Monotonicity

In this section we show how to design an algorithm to correct a non monotone function while preserving its expectation, using $O\left(\log \frac{d}{\varepsilon}\right)^d$ queries.

Theorem 1. *There exists a meta-algotithm $\mathcal{M}_f$ that corrects the monotonicity for any function $f : \mathbb{R}^d \to [0,1]$. $\mathcal{M}_f$ is* feasible*, has query complexity $O(\log \frac{d}{\varepsilon})^d$, and has $\mathbb{E}[\mathcal{M}_f(x))] \geq \mathbb{E}[f(x)] - \varepsilon$, where the first expectation is taken over the input distribution and the randomness in the meta-algorithm.*

To establish Theorem 1 we first show how to solve the problem for the single-dimensional case, $d = 1$. Observe initially that the trivial meta-algorithm that simply queries the function f at every point in the (sufficiently discretized) domain has complexity $O(1/\epsilon)$. To get the improvement to $O(\log(1/\epsilon))$ we first consider the following "greedy" meta-algorithm: query the (discretized) domain in a uniformly random order, and for each point x sequentially, define the transformed output $\mathcal{M}_f(x)$ to be the value closest to $f(x)$ that maintains monotonicity with the values of $\mathcal{M}_f$ constructed so far. We prove that this random meta-algorithm preserves the function value in expectation over the random query order and then that this transformation can be implemented locally, with only logarithmically many queries per input. The details of these steps are presented in subsection 3.1 below.

3.1. Single-dimensional case ($d = 1$)

Using the discretization process described in section 2, we may assume that f is supported on m points and the underlying distribution is uniform. We now provide an algorithm to obtain a monotone function f' that has the same expectation as f and performs $O(\log \frac{m}{\delta})$ queries to f with probability $1 - \delta$. We later choose $\delta = \varepsilon$ and cap the queries to $O(\log \frac{m}{\varepsilon})$ and show that it is possible to do this while maintaining feasibility, monotonicity and incurring error at most ε.

Construction of the Oracle The function f' that the algorithm outputs is a function $\mathcal{M}_f^\pi$ for a uniformly random permutation π. Given a permutation π of $[m]$, we define the function $\mathcal{M}_f^\pi : [m] \to [0,1]$ by setting for every $i \in [m]$,

$$\mathcal{M}_f^\pi(\pi_i) = \begin{cases} H_i & f(\pi_i) > H_i \\ L_i & f(\pi_i) < L_i \\ f(\pi_i) & \text{otherwise,} \end{cases}$$

where H_i is the value $\mathcal{M}_f^\pi(\pi_j)$ of the lowest value point $\pi_j > \pi_i$ with $j < i$ or ∞ if such a point does not exist. Similarly, L_i is the value $\mathcal{M}_f^\pi(\pi_j)$ placed at the highest point $\pi_j < \pi_i$ with $j < i$ or $-\infty$ if such a point does not exist. That is, the function $\mathcal{M}_f^\pi$ visits all points according to the permutation π and greedily assigns values consistent with the choices made for the previous points visited so far to preserve monotonicity. Equivalently, one can write that $\mathcal{M}_f^\pi(\pi_i) = \text{median}\{f(\pi_i), L_i, H_i\}$.

We now show that this choice of f' satisfies the properties of Theorem 1 with the following two claims. Their proofs are deferred to section A.1 of the Appendix.

Claim 1. *For any permutation π, the function $\mathcal{M}_f^\pi$ is monotone and satisfies for all $x \in [m]$, $\mathcal{M}_f^\pi(x) \leq \max_{y \leq x} f(y)$.*

Claim 2. $\boldsymbol{E}_{x\sim\mathbb{U}([m])}[f(x)] = \boldsymbol{E}_{x\sim\mathbb{U}([m])}[f'(x)]$.

It remains to show that one can evaluate $\mathcal{M}_f^\pi$ at any point $x \in [m]$ without querying the oracle for f at all points. To do this, we make the following observation: once we have computed $\mathcal{M}_f^\pi(\pi_1)$, in order to calculate the values of $\mathcal{M}_f^\pi(y)$ for any $y > \pi_1$, we don't need to know the values of $\mathcal{M}_f^\pi(z)$ or of $f(z)$ at any $z < \pi_1$. Similarly, to calculate the values of $\mathcal{M}_f^\pi(y)$ for any $y < \pi_1$, we don't need to know the values of $\mathcal{M}_f^\pi(z)$ or of $f(z)$ at any $z > \pi_1$.

Building on this idea, we use the following algorithm to evaluate $\mathcal{M}_f^\pi(x)$.

Description of Oracle for $\mathcal{M}_f^\pi$ At any point in time i, we keep track of a range of relevant points $\{l_i, ..., r_i\}$ starting with $l_1 = 0$ and $r_0 = m$. We also keep track of a lower bound, lb_i, and an upper bound, ub_i, on the value of $\mathcal{M}_f^\pi(x)$ starting with $\text{lb}_0 = -\infty$ and $\text{ub}_0 = +\infty$.

For any $i \in [m]$, if $\pi_i \in \{l_{i-1}, ..., r_{i-1}\}$, then it is relevant and must be evaluated. Its value is then according to the definition $\mathcal{M}_f^\pi(\pi_i) = \text{median}\{f(\pi_i), H_i, L_i\}$, which is equal to $\text{median}\{f(\pi_i), \text{ub}_{i-1}, \text{lb}_{i-1}\}$ for the upper and lower bounds we have so far. Once the value is computed, if $\pi_i > x$, we set $\text{ub}_i = \mathcal{M}_f^\pi(\pi_i)$ while keeping $\text{lb}_i = \text{lb}_{i-1}$ and the relevant interval now becomes $\{l_i, ..., r_i\} = \{l_{i-1}, ..., \pi_i - 1\}$. Similarly, if $\pi_i < x$, we update $\text{lb}_i = \mathcal{M}_f^\pi(\pi_i)$, $\text{ub}_i = \text{ub}_{i-1}$ and set $l_i = \pi_i + 1$ and $r_i = r_{i-1}$. In contrast, if $\pi_i \notin \{l_{i-1}, ..., r_{i-1}\}$, it is irrelevant and is not evaluated. The interval and upper and lower-bounds are then kept the same for the next iteration.

The following claim shows that for a uniformly random permutation π, the above process only queries the oracle f at a few points. The proof is deferred to section A.1 of the appendix.

Claim 3. *With probability $1 - \delta$, the oracle f' can be evaluated at any point $x \in [m]$ using at most $O(\log \frac{m}{\delta})$ queries to oracle f.*

We now argue that it is possible to perform the transformation so that the algorithm always makes at most $O(\log \frac{m}{\varepsilon})$ while maintaining feasibility, monotonicity and incurring error at most ε. Our construction of the oracle maintains for every interval a high and a low value that points in the interval may take. The interval shrinks with good probability by a constant factor at every step which gives the high probability result. To ensure no more than $O(\log \frac{m}{\varepsilon})$ queries, we need to ensure that every interval shrinks at most $O(\log \frac{m}{\varepsilon})$ times. Indeed, we can enforce that after these many rounds, if the interval has not shrunk to a single point every point in the interval is allocated the lowest value. This ensures monotonicity while it incurs a decrease in the expected value. As this decrease happens with probability at most ε and the decrease is bounded by 1 the total error is at most ε.

3.2. Extending to many dimensions

In order to generalize to many dimensions, we apply our construction for this "single-dimensional case" to fix monotonicity in each direction separately starting with the first. The key property we use is that when given oracle access to a function that is monotone in the first $i-1$ coordinates, our construction of the meta-algorithm will fix the monotonicity in the i-th coordinate *while preserving monotonicity in the $i-1$ first coordinates*. This allows us to obtain a chain of oracles $f = f_0, f_1, ..., f_n = f'$ where f_i is monotone in the first i coordinates. Evaluating f_i requires only $O(\log \frac{d}{\varepsilon})$ queries to oracle f_{i-1} and gets error at most ε/d. Thus, to evaluate $f' = f_n$, $O(\log \frac{d}{\varepsilon})^d$ queries to oracle f are sufficient to get error ε. Details are deferred to section A.2 of the appendix.

Observe that our algorithm guarantees consistency when the same point is asked multiple times. As long as the randomization remains fixed, the query for x will always return the same $\mathcal{M}_f(x)$.

4. Lower bound

Having designed the meta-algorithm to "monotonize" a function, in this section we show that the exponential dependence on the dimension our previous algorithm exhibits, as shown in Theorem 1, is actually necessary even when the domain is the boolean hypercube $\{0,1\}^d$ and the distribution $\mathcal{D}$ of values is uniform. The idea for this lower bound is to show that there exists a function such that any monotone and feasible meta-algorithm $\mathcal{M}$ with subexponential query complexity $q = 2^{o(d)}$ will have very low expectation. This is made formal in Theorem 2 below.

Theorem 2. *Let $\mathcal{M}$ be any feasible meta-algorithm that fixes monotonicity with query complexity $q = 2^{o(d)}$. Then, there exists an input function $f : \{0,1\}^d \to \{0,1\}$ with $\mathbb{E}[f(x)] \geq 1 - 2^{-\Omega(d)}$ such that with $\mathbb{E}[\mathcal{M}_f(x)] \leq 2^{-\Omega(d)}$.*

If the meta-algorithm is infeasible or non-monotone with probability δ, then $\mathbb{E}[\mathcal{M}_f(x)] \leq 2\delta + 2^{-\Omega(d)}$.

To prove the statement, we construct a distribution over functions on $\{0,1\}^d$ and show that any monotone and feasible meta-algorithm must have very low expectation with high probability.

We consider the following family of functions parametrized by (z, S, T) where $z \in \{0,1\}$ and $S, T \subseteq [d]$ so that $S \subseteq T$.

We define for any $X \subseteq [d]$

$$f^z_{S,T}(X) = \begin{cases} 0 & |X| < \frac{4d}{10} \\ \left.\begin{array}{ll} 0 & X \subseteq T \text{ and } |X \setminus S| > \frac{d}{10} \\ z & X \subseteq T \text{ and } |X \setminus S| \leq \frac{d}{10} \\ 1 & X \not\subseteq T \end{array}\right\} & |X| \geq \frac{4d}{10} \end{cases}$$

We also define the function

$$f^1(X) = \mathbb{I}_{\left\{|X| \geq \frac{4d}{10}\right\}}$$

Observe that even though these functions are defined over subsets of $[d]$ it is straightforward to view each of these subsets as a point in $\{0,1\}^d$.

We define a distribution over the family of functions by selecting z uniformly at random. The sets S and T will also be random variables with S including each element with probability $1/2$ and T including each element with probability $3/4$. Since $S \subseteq T$, this means that given S, the set T contains each element outside of S with probability $1/2$. Similarly given T, the set S contains each element of T with probability $2/3$. We also define random variable X that is a uniformly random subset of $[d]$.

What we are trying to achieve with these functions is while f^1 has high expectation, the function $f^z_{S,T}$ does not, but we will not be able to tell them apart. The idea is that in both functions $f^z_{S,T}$ there is the "low" set T where the function outputs 0, but inside it there is a hidden set S where the function is either 1 or 0. The two claims shown below, will prove that first we cannot distinguish between the function that gives S either 0 or 1, and then this function from the "high" function that gives 1 to all large inputs. This is shown in figure 1 below[1].

Claim 4. $\Pr\left[\mathcal{M}_{f^1_{S,T}}(T) \neq \mathcal{M}_{f^0_{S,T}}(T)\right] \leq q2^{-\frac{d}{450}}$

Claim 5. $\Pr\left[\mathcal{M}_{f^1_{S,T}}(S) \neq \mathcal{M}_{f^1}(S)\right] \leq q2^{-\frac{d}{10}}$

The proofs of both the claims are deferred to section B of the appendix. It is now easy to complete the proof of Theorem 2 by setting $f = f^1$.

$$\begin{aligned} \mathbb{E}[\mathcal{M}_{f^1}(x)] &= \mathbb{E}\left[\mathcal{M}_{f^1}(S)\right] \\ &\leq \mathbb{E}\left[\mathcal{M}_{f^1_{S,T}}(S)\right] + q2^{-\frac{d}{10}} \\ &\leq \delta + \mathbb{E}\left[\mathcal{M}_{f^1_{S,T}}(T)\right] + q2^{-\frac{d}{10}} \\ &\leq \delta + \mathbb{E}\left[\mathcal{M}_{f^0_{S,T}}(T)\right] + q2^{-\frac{d}{10}} + q2^{-\frac{d}{450}} \\ &\leq 2\delta + q2^{-\frac{d}{10}} + q2^{-\frac{d}{450}} \\ &= 2\delta + 2^{-\Omega(d)} \end{aligned}$$

where the first line follows since S is chosen uniformly at random, then we use Claim 5 and then we use that f satisfies monotonicity with probability $1-\delta$. Following this, the third line follows from Claim 4 and then we use the fact that $\mathcal{M}_{f^0_{S,T}}(T) \leq \max_{Y \subseteq T} f^0_{S,T}(Y) = 0$ with probability $1-\delta$. Finally we get the result for any $q = 2^{o(d)}$.

In contrast, for the initial function f^1 we get $\mathbb{E}\left[f^1(X)\right] = \Pr\left[|X| \geq \frac{4d}{10}\right] \geq 1 - 2^{-\Omega(d)}$.

5. Marginal Monotonicity

In this section, we switch gears towards the relaxations of monotonicity defined in section 2. We start by considering *marginal monotonicity*. In this case, we want to guarantee that each of the marginals of the function will be monotone, and not loosing much in expectation. As it turns out, we can achieve this in time polynomial in d/ε. The formal statement follows.

Theorem 3. *There exists a feasible meta-algorithm $\mathcal{M}_f$ that fixes marginal monotonicity for any function $f : \mathbb{R}^d \to [0,1]$. $\mathcal{M}_f$ is* feasible*, has query complexity $O\left(poly\left(\frac{d}{\varepsilon}\right)\right)$, and satisfies $\mathbb{E}[\mathcal{M}_f(x)] \geq \mathbb{E}[f] - \varepsilon$, where the first expectation is taken over the input distribution and the randomness of the meta-algorithm.*

The discretization process described in section 2 will also be used here. Observe that we sample the marginals after conditioning, but this can be efficiently implemented by rejection sampling. Therefore we can safely assume that the domain is discretized and supported on m different values which we denote by $w_{i1} < \ldots < w_{im}$.

We start by assuming we are given query access directly to the marginal distribution f_i in every dimension and showing that we can achieve the theorem using $O(dm)$ queries. Then in subsection 5.2, we show how to achieve the same result by only querying the initial function f in order to estimate the marginals.

[1] The figure serves as a simplified example of the structure of the functions, the sizes of the sets are not on scale

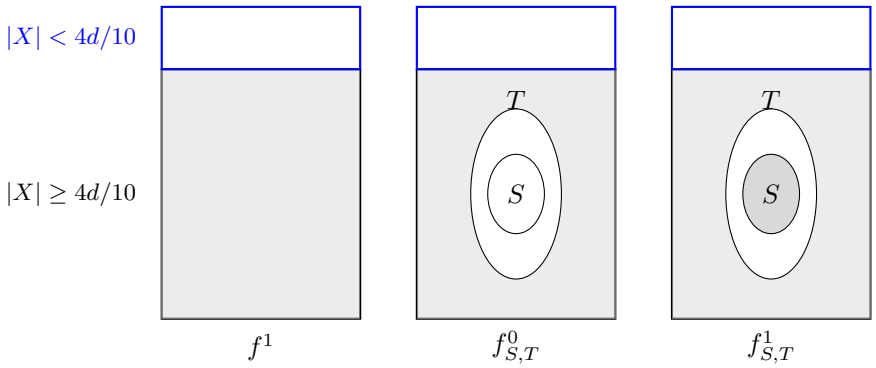

Figure 1. Output for each function. The gray colour means the output is 1, white means 0.

5.1. Transformation Using Exact Marginals

In this case we assume that we know exactly each one of the marginals $f_i(x_i)$.

Consider meta-algorithms of the following form: in each dimension i there will be a mapping $\phi_i \colon \mathbb{R} \to \mathbb{R}$, with $\phi_i(x_i) \leq x_i$ for all $x_i \in \mathbb{R}$. We will write $\phi(x) = (\phi_1(x_1), \ldots, \phi_d(x_d))$. We will then define $f'(x) = f(\phi(x))$. Observe that any such f' satisfies feasibility, since $f'(x) = f(\phi(x)) \leq \max_{y \preceq x} f(y)$.

We will build the mapping ϕ iteratively, starting with the identity mapping, which we will call ϕ^0. Since the distribution over values is discrete, it suffices to define each mapping ϕ_i on the finitely many values in the support of the distribution (for each one of the d dimensions).

Suppose our current mapping is $\phi^r = (\phi^r_1, \ldots, \phi^r_d)$, for some $r \geq 0$ and let $f^r(x) = f(\phi^r(x))$. If f^r_i, the i'th marginal function is monotone for every i, then we will choose $f' = f^r$.

Otherwise, there is some i and some $j < m$ such that $f^r_i(w_{ij}) > f^r_i(w_{i(j+1)})$. In this case, we will define ϕ^{r+1} as follows: $\phi^{r+1}(w_{i(j+1)}) = \phi^r(w_{ij})$, and $\phi^{r+1} = \phi^r$ on all other inputs. That is, whenever f' is given $w_{i(j+1)}$ as input, we will instead invoke f^r as though we have gotten w_{ij}. Observe that this modification chains: if on some previous iteration we had mapped input w_{ij} to $w_{i(j-1)}$, then after this iteration we will ultimately be passing the input $w_{i(j-1)}$ to the original function f. As argued above, this modified function f^{r+1} will be feasible. Moreover,

$$\begin{aligned}\mathbf{E}_{x\sim\mathcal{D}}[f^{r+1}(x)] &= \mathbf{E}_{x_i}[f^{r+1}_i(x_i)] \\ &\geq \mathbf{E}_{x_i}[f^r_i(x_i)] \\ &= \mathbf{E}_{x\sim\mathcal{D}}[f^r(x)]\end{aligned}$$

so f^{r+1} has only weakly greater expected value than f^r.

We can think of f^{r+1} as acting on a reduced domain, where the possible input $w_{i(j+1)}$ is removed from the support of x_i's distribution and instead its probability mass is added to that of some lower value, $\phi^r(w_{ij})$. Under this interpretation, each iteration reduces the total number of possible input values in the support of $\mathcal{D}$ by 1. This process must therefore stop at or before iteration $r = d(m-1)$, since a marginal over a single input value is always monotone. Thus, after at most $d(m-1)$ iterations, this process will terminate at an function f' that is feasible, monotone, and has $\mathbf{E}_{x\sim\mathcal{D}}[f'(x)] \geq \mathbf{E}_{x\sim\mathcal{D}}[f(x)]$.

5.2. Sampling to Estimate Marginals

The meta-algorithm above assumes direct access to the marginal distributions even after the modifications we make at each step. We will show how to remove these assumptions, at the cost of a loss of ε on the expectation of f'. This ε loss is due to sampling error, and can be made as small as desired with additional sampling.

Prior to viewing the input, our meta-algorithm will estimate each one of the marginals. For each $i \in [d]$, take $O(\log(dm)/\delta^2)$ samples x_{-i} and observe $f(x_i, x_{-i})$. Let $\tilde{f}_i(x_i)$ for the empirical mean of the observed samples. By Hoeffding inequality and a union bound over all coordinates and all possible input values, we will have that $|\tilde{f}_i(x_i) - f_i(x_i)| \leq \delta$ for all i and all x_i, with high probability.

We will then apply our meta-algorithm from above using the marginals $\tilde{f}$ as an oracle. This generates mappings ϕ_i, such that the new "monotonized" marginals $\tilde{f}_i(\phi_i)$ have weakly increased expectation relative to $\tilde{f}_i$. If $|\mathbf{E}[\tilde{f}_i(x_i)] - f_i(x_i)| \leq \delta$, then we also have $|\mathbf{E}[\tilde{f}_i(\phi(x_i))] - f_i(\phi(x_i))| \leq \delta$ for all i and all x_i as well. Monotonicity of $\tilde{f}_i(\phi_i)$ therefore implies (2δ)-approximate monotonicity of $f(\phi)$, and that $\mathbf{E}[f(\phi(x))] \geq \mathbf{E}[f(x)] + 2\delta$.

From Approximate to Exact Monotonicity From the above steps, we can assume access to a function f that is (2δ)-approximately monotone. We will implement a new

	Initial		Replacement
Rule 1	001?1?0?	$\rightarrow$	001?0?0?
Rule 2	0?1?01?1	$\rightarrow$	0?1?01?0
...		...	
Rule ℓ	?01?0?10	$\rightarrow$	?01?0?00

Table 1. Example of replacement rule list. For any input x, the input is sequentially transformed to a different one by applying the rules from top to bottom, the function value at the resulting vector is then returned.

function f' such that, on input x, say with $x_i = w_{ij_{(i)}}$ for each i, returns $\max\{0, f(x) - 2\delta \cdot \sum_i (m - j_{(i)})\}$. Then f' is monotone, as whenever $x > y$ we have that either $f'(y) = 0$ or $f'(x) - f'(y) \geq f(x) - f(y) + 2\delta \geq 0$. Moreover, this modification reduces the expected allocation of f by at most $3md \cdot \delta$. So as long as $\delta \leq \varepsilon/(dm)$, its expected allocation is within ε of f.

6. k-Marginal Monotonicity

Having designed an algorithm for marginal monotonicity, we move on to a generalization that guarantees *k-marginal monotonicity*. In this case, given oracle access to a function $f : \mathbb{R}^d \to [0, 1]$ we want to guarantee that all k-marginals of f will be monotone.

Theorem 4. *There exists a meta-algorithm $\mathcal{M}_f$ to fix k-marginal monotonicity (whp) for any function $f : \mathbb{R}^d \to [0, 1]$. $\mathcal{M}_f$ is* feasible*, has query complexity $\tilde{O}\left(\frac{d^{2k+6}}{\varepsilon^{2k+3}}\right)$ and satisfies $\mathbb{E}[\mathcal{M}_f(x)] \geq \mathbb{E}[f] - \varepsilon$, where the first expectation is taken over the input distribution and the randomness of the meta-algorithm.*

In order to prove the theorem, we start by using the discretization method described in section 2. After having a discrete domain, our meta-algorithm estimates in each step all[2] the d^k marginals and whenever it detects a non-monotonicity in one of them, it fixes it within δ by defining a set of replacement rules. Then for an input x, we sequentially try to apply the rules, bu starting from the first, and trying to match the given point with one of the patterns of the rules. After we applied the possible rules, we reached some other point x' and then we output $f(x')$. An example of the set of replacement rules is shown in Table 1.

We describe the algorithm in three steps; first we describe the replacement process, in order to fix the non-monotonicities, given that we have access to the exact margninals, and then describe the sampling process used to estimate the marginals. Using the first two steps, we are guaranteed only approximate monotonicity, so as the third step we show how to further modify our function to achieve exact monotonicity.

[2]Note that there is no way to avoid fixing all the $\binom{d}{k}$ marginals; even $\binom{d}{k} - 1$ monotone marginals cannot guarantee that the $\binom{d}{k}$'th is also monotone.

6.1. Transformation Using Exact Marginals

Assuming now that we have access to all the k-marginals exactly, we describe an iterative process, in order to correct the non-monotonicities in all the k-marginals. The answer we give is the transformed function f'.

As a direct extension from the previous marginals case, consider transformations of the form: $\phi : \mathbb{R}^d \to \mathbb{R}^d$ such that $\phi(x) \leq x$, for all $x \in \mathbb{R}^d$. We define $f'(x) = f(\phi(x))$.

We denote by $\phi_{\mathcal{I}}(x_{\mathcal{I}}) : \mathbb{R}^k \to \mathbb{R}^d$, for some $\mathcal{I} \subseteq [d]$, with $|\mathcal{I}| \leq k$, the projection of the function ϕ in the k coordinates that are in $\mathcal{I}$, where the variables x_j for $j \notin \mathcal{I}$ are treated as constants and remain the same. Recall that by $x_{\mathcal{I}}$ we denote all the coordinates x_i such that i is in the set $\mathcal{I} \subseteq [d]$ with $|\mathcal{I}| \leq k$.

Intuitively, what the process does is when we detect a non-monotonicity between some input x and its neighbor y, where y is larger in at least one coordinate[3], from then on we always map y to x. This is reflected in the function ϕ that replaces y with x to correct the non-monotonicities. Since this process is done iteratively, when we map y to x and x to some other input z, it means y is ultimately mapped to z.

More formally, we start from the identity function $\phi^0(x) = x$, and iteratively define $\phi^1, \phi^2, \ldots$. In this case we define a non-monotonicity as the case when there exists a set $\mathcal{I} \subseteq [d]$, with $|\mathcal{I}| \leq k$, such that $x_{\mathcal{I}} < y_{\mathcal{I}}$ we have that $f_{\mathcal{I}}(x_{\mathcal{I}}) > f_{\mathcal{I}}(y_{\mathcal{I}}) + \delta$. Observe that in this case, we only ensure that the function is δ monotone. In iteration r, one of the following cases can happen

1. There is no such set $\mathcal{I}$: we terminate the process and return $f'(x) = f(\phi^r(x))$
2. There exists such a set $\mathcal{I}$: we set

$$\phi^{r+1}_{\mathcal{I}}(y_{\mathcal{I}}) = \max_{z \in \mathcal{N}(y)} \phi^r_{\mathcal{I}}(z_{\mathcal{I}})$$

where recall from section 2 that $\mathcal{N}(y)$ is the low 1-neighborhood of y.

Using this process, and exactly how we argued in the previous section, the output function f' is feasible and that $\mathbb{E}_{x \sim \mathcal{D}}[f^{r+1}(x)] \geq \mathbb{E}_{x \sim \mathcal{D}}[f^r(x)]$.

Observe that every time the replacement described above happens, the expectation of the transformed function f' increases by at least δ/m^k, meaning that this process cannot happen more than m^k/δ times in total since $f' \leq 1$.

[3]We can assume without loss of generality that y is higher in *exactly* one coordinate.

When the process halts, the transformed function f' is feasible and approximately δ-monotone, with $\mathbb{E}_{x\sim\mathcal{D}}[f'(x)] \geq \mathbb{E}_{x\sim\mathcal{D}}[f(x)]$.

6.2. Sampling to Estimate Marginals

In the process described above, we assumed that the exact marginals were known. In reality, in every step we estimate the marginals again before checking for non-monotonicities, by sampling points $x \in \mathbb{R}^d$ and observing $f(x)$.

This step differs from the estimation step in the marginals case in that we do not draw samples $x_{\mathcal{I}}$ from each specific marginal $f_{\mathcal{I}}$, but directly from the function f and then we estimate each marginal.

Recall from the discretization process, that now the distribution over the m^k different values f can take is uniform, which means that by drawing samples from f, we need m^k samples in expectation to get a sample from a specific marginal. Using this fact, Hoeffding's inequality and a union bound over all different d^k marginals, m^k values and all m^k/δ rounds this sampling process is happening, we need $\frac{k}{m^k\delta^2}\log(\frac{m^2 d}{\delta})$ samples.

6.3. From Approximate to Exact Monotonicity

This part is exactly the same as the previous section with the only difference that we have access to a function that is 4δ-approximately monotone. This difference is due to the fact that we only guaranteed δ monotonicity when we knew the exact marginals compared to exact monotonicity. Therefore, we now return a new function f' that on input x with $x_i = w_{ij_{(i)}}$ for each i, returns $\max\{0, f(x) - 4\delta\cdot\sum_i(m - j_{(i)})\}$, which as before is guaranteed to be monotone.

This modification reduces the expected allocation of f by at most $4md\cdot\delta$ in this case, so when $\delta \leq \varepsilon/(dm)$, its expected allocation is within ε of f.

Query Complexity In order to calculate the query complexity of the meta-algorithm, recall that there are m^k/δ rounds, where for each round we use $km^k/\delta^2\log(m^2d/\delta)$ queries for the marginal estimation.

Using that $\delta < 1/(dm)$ and that $m = d/\varepsilon$ from the discretization process, we get that the query complexity is $\frac{kd^{2k+6}}{\varepsilon^{2k+3}}\log\left(\frac{d^5}{\varepsilon^3}\right) = \tilde{O}\left(\frac{d^{2k+6}}{\varepsilon^{2k+3}}\right)$.

7. Conclusion

In this paper we presented a meta-algorithm to correct a possibly non monotone function, in a black-box way, so that it always gives answers that are monotone, and maintains the performance guarantees of the initial function up to a small error. After showing that the algorithm presented cannot be improved, we relaxed the monotonicity requirement to that of k-marginal monotonicity. One interesting open question is whether we can improve the marginal monotonicity runtime to logarithmic instead of linear in d/ε.

Another problem that might be interesting is to extend this meta-algorithm beyond product distribution. Our results rely crucially on the independence assumption and we are not aware of a non trivial way to remove this.

References

Ailon, N., Chazelle, B., Comandur, S., and Liu, D. Property-preserving data reconstruction. *Algorithmica*, 51(2):160–182, 2008. doi: 10.1007/s00453-007-9075-9.

Alon, N., Rubinfeld, R., Vardi, S., and Xie, N. Space-efficient local computation algorithms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pp. 1132–1139, 2012.

Austin, T. and Tao, T. Testability and repair of hereditary hypergraph properties. *Random Struct. Algorithms*, 36 (4):373–463, 2010. doi: 10.1002/rsa.20300.

Bei, X. and Huang, Z. Bayesian incentive compatibility via fractional assignments. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pp. 720–733. Society for Industrial and Applied Mathematics, 2011.

Bhattacharyya, A., Grigorescu, E., Jha, M., Jung, K., Raskhodnikova, S., and Woodruff, D. P. Lower bounds for local monotonicity reconstruction from transitive-closure spanners. *SIAM J. Discrete Math.*, 26(2):618–646, 2012. doi: 10.1137/100808186.

Campagna, A., Guo, A., and Rubinfeld, R. Local reconstructors and tolerant testers for connectivity and diameter. In Raghavendra, P., Raskhodnikova, S., Jansen, K., and Rolim, J. D. P. (eds.), *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, volume 8096 of *Lecture Notes in Computer Science*, pp. 411–424. Springer, 2013. doi: 10.1007/978-3-642-40328-6_29.

Canonne, C. L., Gouleakis, T., and Rubinfeld, R. Sampling correctors. In Sudan, M. (ed.), *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pp. 93–102. ACM, 2016. doi: 10.1145/2840728.2840729. URL https://doi.org/10.1145/2840728.2840729.

Chakraborty, S., Fischer, E., and Matsliah, A. Query complexity lower bounds for reconstruction of codes. *Theory of Computing*, 10:515–533, 2014. doi: 10.4086/toc.2014.v010a019.

Chawla, S., Immorlica, N., and Lucier, B. On the limits of black-box reductions in mechanism design. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pp. 435–448, 2012. doi: 10.1145/2213977.2214019.

Chazelle, B. and Seshadhri, C. Online geometric reconstruction. *J. ACM*, 58(4):14:1–14:32, 2011. doi: 10.1145/1989727.1989728.

Dughmi, S., Hartline, J. D., Kleinberg, R., and Niazadeh, R. Bernoulli factories and black-box reductions in mechanism design. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 158–169. ACM, 2017.

Even, G., Medina, M., and Ron, D. Deterministic stateless centralized local algorithms for bounded degree graphs. In Schulz, A. S. and Wagner, D. (eds.), *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pp. 394–405. Springer, 2014. doi: 10.1007/978-3-662-44777-2_33.

Gergatsouli, E., Lucier, B., and Tzamos, C. The complexity of black-box mechanism design with priors. In *Proceedings of the 2019 ACM Conference on Economics and Computation, EC 2019, Phoenix, AZ, USA, June 24-28, 2019*, pp. 869–883, 2019. doi: 10.1145/3328526.3329648.

Ghaffari, M. and Uitto, J. Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In Chan, T. M. (ed.), *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pp. 1636–1653. SIAM, 2019. doi: 10.1137/1.9781611975482.99.

Grunau, C., Mitrovic, S., Rubinfeld, R., and Vakilian, A. Improved local computation algorithm for set cover via sparsification. In Chawla, S. (ed.), *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pp. 2993–3011. SIAM, 2020. doi: 10.1137/1.9781611975994.181.

Hartline, J. D. and Lucier, B. Non-optimal mechanism design. *American Economic Review*, 105(10):3102–24, October 2015. doi: 10.1257/aer.20130712.

Hartline, J. D., Kleinberg, R., and Malekian, A. Bayesian incentive compatibility via matchings. *Games and Economic Behavior*, 92:401 – 429, 2015. ISSN 0899-8256. doi: https://doi.org/10.1016/j.geb.2015.02.002.

Jha, M. and Raskhodnikova, S. Testing and reconstruction of lipschitz functions with applications to data privacy. *SIAM J. Comput.*, 42(2):700–731, 2013. doi: 10.1137/110840741.

Levi, R., Ron, D., and Rubinfeld, R. Local algorithms for sparse spanning graphs. In Jansen, K., Rolim, J. D. P., Devanur, N. R., and Moore, C. (eds.), *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, volume 28 of *LIPIcs*, pp. 826–842. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi: 10.4230/LIPIcs.APPROX-RANDOM.2014.826.

Levi, R., Ron, D., and Rubinfeld, R. A local algorithm for constructing spanners in minor-free graphs. In Jansen, K., Mathieu, C., Rolim, J. D. P., and Umans, C. (eds.), *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, volume 60 of *LIPIcs*, pp. 38:1–38:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi: 10.4230/LIPIcs.APPROX-RANDOM.2016.38.

Levi, R., Rubinfeld, R., and Yodpinyanee, A. Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, 77(4):971–994, 2017. doi: 10.1007/s00453-016-0126-y.

Mansour, Y. and Vardi, S. A local computation approximation scheme to maximum matching. In Raghavendra, P., Raskhodnikova, S., Jansen, K., and Rolim, J. D. P. (eds.), *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, volume 8096 of *Lecture Notes in Computer Science*, pp. 260–273. Springer, 2013. doi: 10.1007/978-3-642-40328-6_19.

Parter, M., Rubinfeld, R., Vakilian, A., and Yodpinyanee, A. Local computation algorithms for spanners. In Blum, A. (ed.), *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pp. 58:1–58:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi: 10.4230/LIPIcs.ITCS.2019.58.

Rubinfeld, R., Tamir, G., Vardi, S., and Xie, N. Fast local computation algorithms. In *Innovations in Computer*

Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings, pp. 223–238, 2011.

Saks, M. E. and Seshadhri, C. Local monotonicity reconstruction. *SIAM J. Comput.*, 39(7):2897–2926, 2010. doi: 10.1137/080728561.