Black-Box Methods for Restoring Monotonicity

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

[1]University of Wisconsin-Madison

[2]Microsoft Research

ICML, July 2020

Overview - Motivation

- *Example 1: the deadline is approaching!*

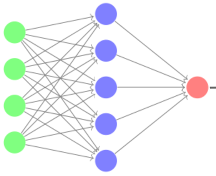
ICML Page limit: 10

Delete sentence

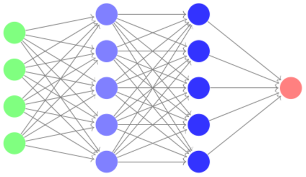
11 pages

12 pages

- *Example 2: hyperparameter tuning*



Add layers

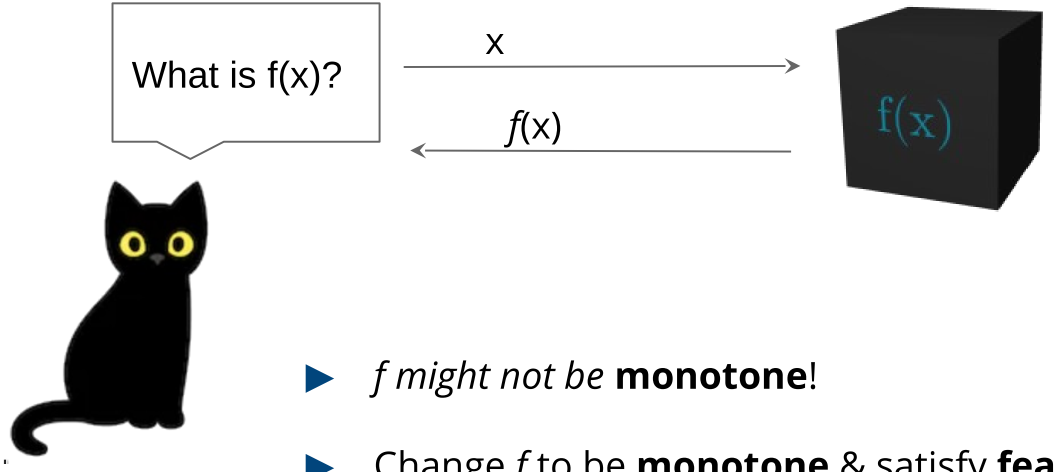


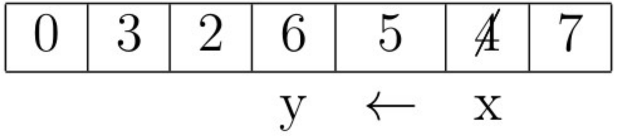
error k

error 3k

How do we fix monotonicity,
in a black box way?

Our Model



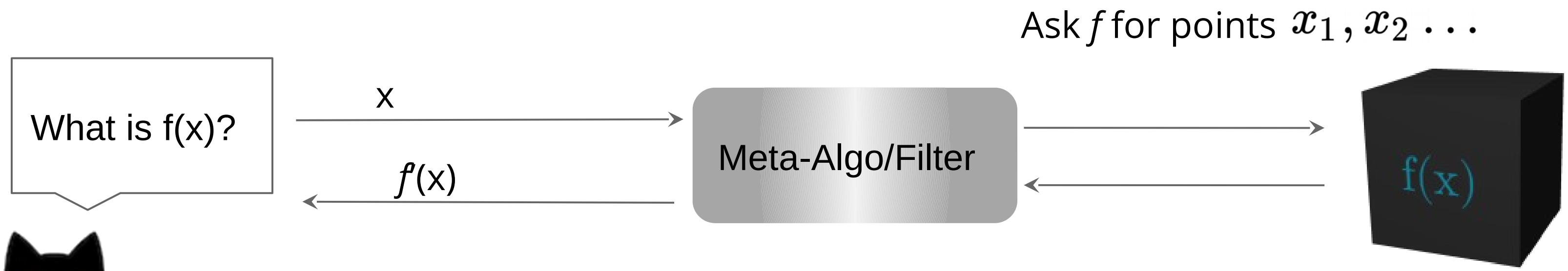


Example for 1 dimension

f′(x) = 1 or 7 may not be feasible but
f′(x) = 6 is feasible

- *f might not be* **monotone**!
- Change *f* to be **monotone** & satisfy **feasibility**
- **Feasibility**: Unclear what solutions/values are feasible
  - *Solutions for smaller inputs feasible for larger ones*
  - Output any f(y) from $y \leq x$

Our Model



▶ *Idea 1: $f' = 0$ for all inputs*
- f' monotone ✓
- f' better than f ✗
- Filter is query efficient ✓

0	3	2	6	5	4	7

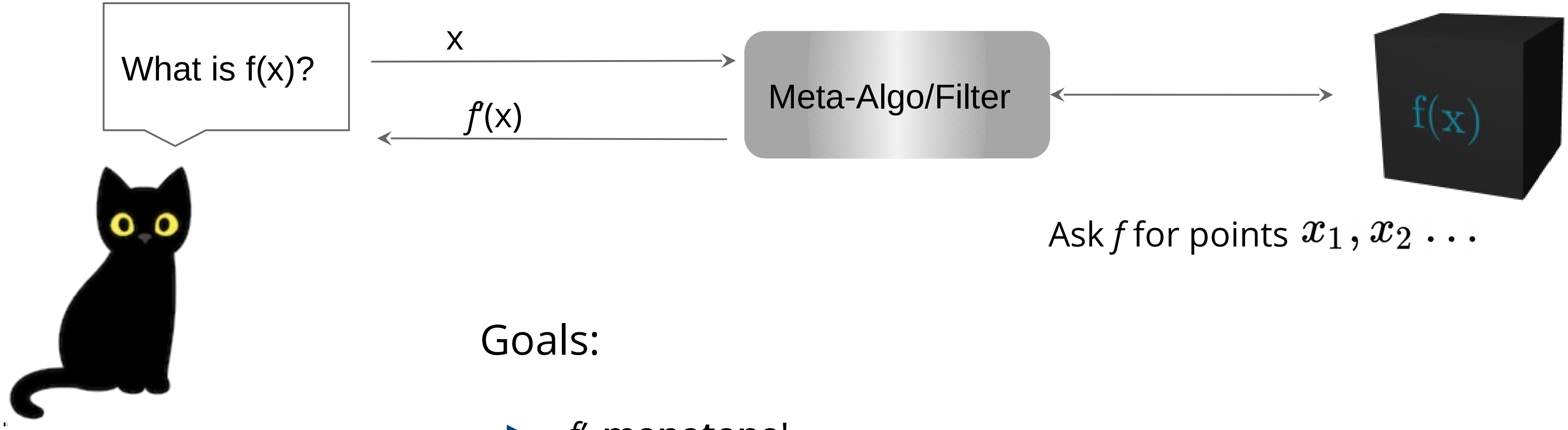
0	**0**	**0**	**0**	**0**	**0**	**0**

▶ *Idea 2:* $f' = \max_{y \le x} f(y)$
- f' monotone ✓
- f' better than f ✓
- Filter is query efficient ✗

0	3	2	6	5	4	7

0	3	**3**	6	**6**	**6**	7

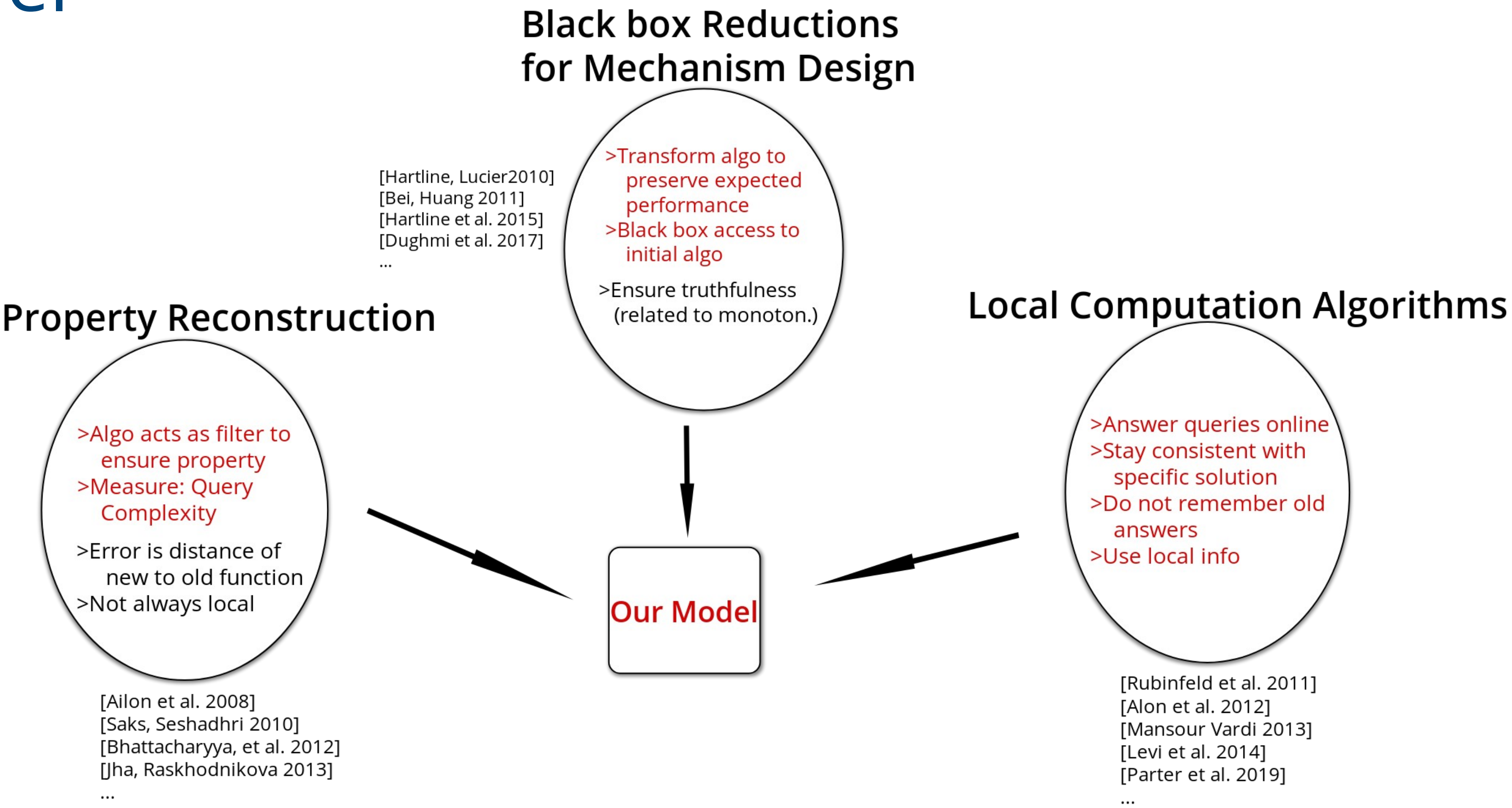
Our Model



Goals:

- f' **monotone**!
- Feasible
- $E[f']$ better than $E[f]$
- Query efficient

Model


Black box Reductions
for Mechanism Design
[Hartline, Lucier2010]
[Bei, Huang 2011]
[Hartline et al. 2015]
[Dughmi et al. 2017]
...
>Transform algo to preserve expected performance
>Black box access to initial algo
>Ensure truthfulness (related to monoton.)
Property Reconstruction
>Algo acts as filter to ensure property
>Measure: Query Complexity
>Error is distance of new to old function
>Not always local
[Ailon et al. 2008]
[Saks, Seshadhri 2010]
[Bhattacharyya, et al. 2012]
[Jha, Raskhodnikova 2013]
...
Local Computation Algorithms
>Answer queries online
>Stay consistent with specific solution
>Do not remember old answers
>Use local info
[Rubinfeld et al. 2011]
[Alon et al. 2012]
[Mansour Vardi 2013]
[Levi et al. 2014]
[Parter et al. 2019]
...
Our Model

Our Results

Theorem 1: *There is a meta-algorithm that "monotonizes" any univariate function and*

1. *Is feasible*
2. *Has expected performance loss at most ε*
3. *Uses at most* $O(\log \frac{1}{\varepsilon})$ *queries.*

The algorithm extends to ***d****-variate functions and uses at most* $O(\log \frac{d}{\varepsilon})^d$ *queries*

Theorem 2: *Any meta-algorithm that "monotonizes" functions with non-trivial performance guarantees must make exponential in* ***d*** *queries.*

Our Results

We can escape this exponential hardness by considering a weaker notion of monotonicity: monotonicity of marginals

Theorem 3: *There is a meta-algorithm that corrects k marginal monotonicity of any f and*

1. *Is feasible*
2. *Has expected loss at most ε*
3. *Uses at most* $(d/\varepsilon)^{O(k)}$ *queries.*

This talk: focus on **Theorems 1+2**

Formal Model

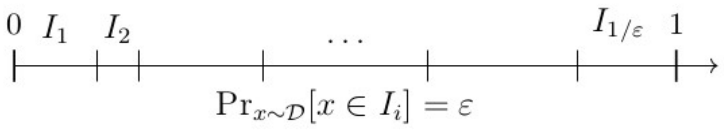
- Oracle access to $f : \mathbb{R}^d \to [0, 1]$
- Input $\mathbf{x} = (x_1 \ldots x_d)$
  - Product distribution $\mathcal{D}$
- Monotonicity: *f(x) ≤ f(y) whenever x ≤ y* coordinate-wise
  - Marginal monotonicity: $f_i(x_i) = E_{x_{-i} \sim \mathcal{D}_{-i}}[f(x_i, x_{-i})]$ is monotone
  - k-marginal monotonicity: for any subset $\mathcal{I}$ of coordinates, $f_{\mathcal{I}}(x_{\mathcal{I}}) = E_{x_{-\mathcal{I}}}[f_{\mathcal{I}}(x_{\mathcal{I}}, x_{-\mathcal{I}})]$ is monotone

Discretization

- Output $\tilde{f}(x)$ a piece-wise constant function
  - Split into 1/ε pieces of equal probability
  - Define

$$\tilde{f}(x) = \begin{cases} 0 & x \in I_1 \\ f(x_{i-1}) & x \in I_i \end{cases}$$

for a random x_{i-1} drawn from I_{i-1}



- $\tilde{f}(x)$:
  - is feasible
  - loses at most ε in expectation

- We only need to monotonize $\tilde{f}(x)$ which is discrete

Monotonicity in 1 dimension

- Input x
- Randomly permute points $\{1, \ldots, 1/\varepsilon\}$
- Keep track of Lower & Upper bounds $(L_i, U_i) = (-\infty, \infty)$
- Keep track of *relevant interval* $[\ell, h] = [0, 4]$

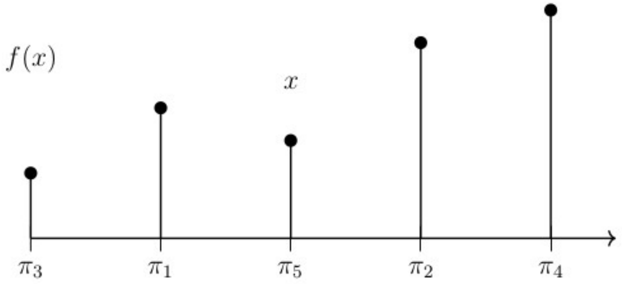


Figure 1: Function before fixing monotonicity

Monotonicity in 1 dimension

Lower & Upper bounds $(-\infty, \infty)$
Relevant interval $[\ell, h] = [0, 4]$

- Pick next point π_i in permutation
  - If $x > \pi_i$ then
    - new Bounds: $(f(\pi_1), \infty)$
    - Relevant interval is $[\pi_i + 1, h] = [2, 4]$
  - If $x < \pi_i$
    - ...

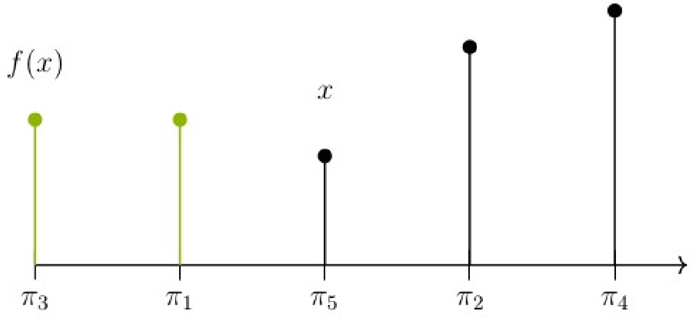


Figure 2: Function after processing π_1

Monotonicity in 1 dimension

Lower & Upper bounds $(f(\pi_1), \infty)$
Relevant interval $[\ell, h] = [2, 4]$

- Pick next point π_i in permutation
  - ...
  - If $x < \pi_i$ then
    - new Bounds: $(f(\pi_1), f(\pi_2))$
    - Relevant interval is $[\ell, \pi_i - 1] = [2, 2]$

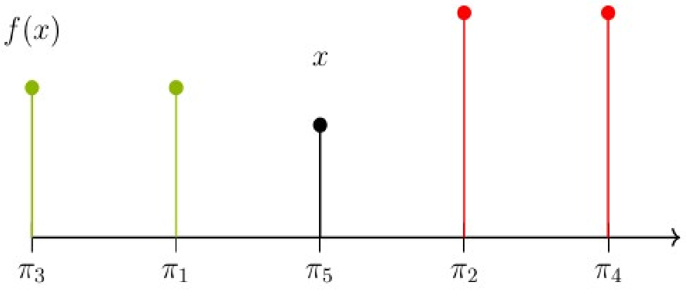


Figure 3: Function after processing π_2

Many Dimensions

- Fix monotonicity in every dimension, how?

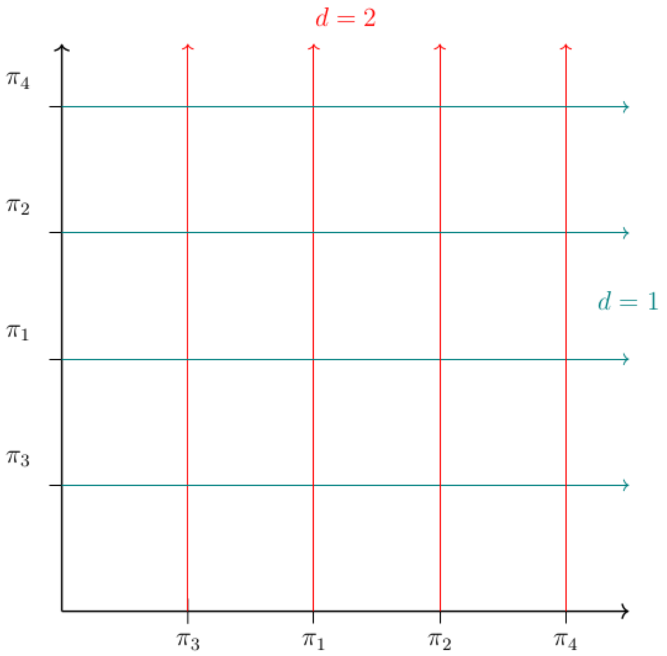
Key Lemma: Given a function monotone in the first i-1 dimensions, we can fix i'th without violating the i-1

- Pick permutations from the beginning
- Starting from f, fix every dimension sequentially
  - f_i Function with the first i dimensions "fixed"

$$f_0 \to f_1 \to \dots f_d \to \text{output}$$

- Query complexity

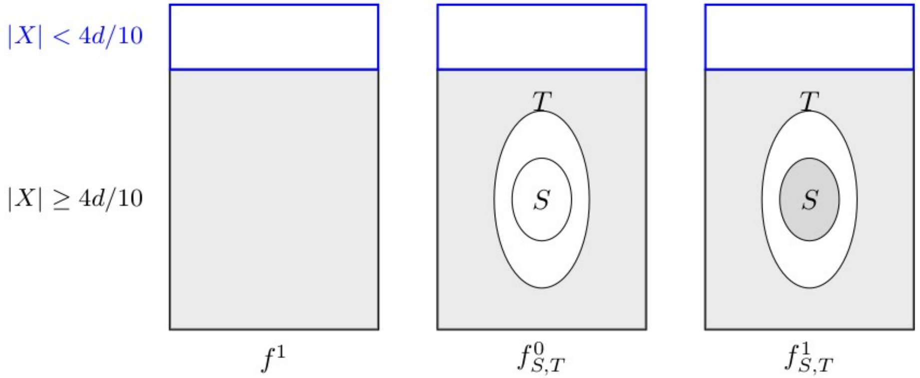
$$O(\log \tfrac{d}{\varepsilon})^d$$



Lower Bound

- For some domain and distribution
- Original function f[1], has high expectation
- Randomized functions f^z_{ST} based on random sets S, T
- f^0_{ST} has low expectation, f^1_{ST} has high
- Cannot distinguish between f^0_{ST} and f^1_{ST}
- Cannot distinguish between f^1_{ST} and f^1

} With fewer than exponential queries



Conclusion & Open problems

- An algorithm to fix non-monotonicity in d dimensions with black-box queries
- Unavoidable exponential dependence in d
- Relaxed versions of monotonicity (k-Marginal Monotonicity) fixable efficiently

Open problems:

- Correlated coordinate distributions?
- Other properties beyond monotonicity?