# An Efficient Pruning Algorithm for Robust Isotonic Regression

**Cong Han Lim** [*]
School of Industrial Systems and Engineering
Georgia Tech
Altanta, GA 30332
clim31@gatech.edu

## Abstract

We study a generalization of the classic isotonic regression problem where we allow separable nonconvex objective functions, focusing on the case where the functions are estimators used in robust regression. One can solve this problem to within $\epsilon$-accuracy (of the global minimum) in $O(n/\epsilon)$ using a simple dynamic program, and the complexity of this approach is independent of the underlying functions. We introduce an algorithm that combines techniques from the convex case with branch-and-bound ideas that naturally exploits the shape of the functions. Our algorithm achieves the best known bounds for both the convex case ($O(n \log(1/\epsilon))$) and the general nonconvex case. Our experiments show that this algorithm performs much faster than the dynamic programming approach on robust estimators, especially as the desired accuracy increases.

## 1 Introduction

In this paper we study the following optimization problem with monotonicity constraints:

$$\min_{x\in[0,1]^n} \sum\nolimits_{i\in[n]} f_i(x_i) \;\text{ where }\; x_i \le x_{i+1} \text{ for } i \in [n-1] \tag{1}$$

where the functions $f_1, f_2, \ldots, f_n : [0,1] \to \mathbb{R}$ *may be nonconvex* and the notation $[n]$ denotes the set $\{1, 2, \ldots, n\}$. Our goal is to develop an algorithm that achieves an objective $\epsilon$-close to the *global* optimal value for any $\epsilon > 0$ with a complexity that scales along with the properties of $f$. In particular, we present an algorithm that simultaneously achieves the best known bounds when $f_i$ are convex and also for general $f_i$, while scaling much better in practice than the straightforward approach for $f$ used in robust estimation such as the Huber Loss, Tukey's biweight function, and SCAD.

Problem (1) is a generalization of the classic *isotonic regression* problem (Brunk, 1955; Ayer et al., 1955). The goal there to find the best isotonic fit in terms of *Euclidean distance* to a given set of points $y_1, y_2, \ldots, y_n$. This corresponds to setting each $f_i(x)$ to $\|x_i - y_i\|_2^2$. Besides having applications in domains where such a monotonicity assumption is reasonable, isotonic regression also appears as a key step in other statistical and optimization problems such as learning generalized linear and single index models Kalai and Sastry (2009), submodular optimization (Bach, 2013), sparse recovery (Bogdan et al., 2013; Zeng and Figueiredo, 2014), and ranking problems (Gunasekar et al., 2016).

There are several reasons to go beyond Euclidean distance and to consider more general $f_i$ functions. For example, using the appropriate Bregman divergence can lead to better regret bounds for certain online learning problems over the convex hull of all rankings (Yasutake et al., 2011; Suehiro et al., 2012), and allowing general $f_i$ functions has applications in computer vision (Hochbaum, 2001;

[*]Work done while affiliated with Wisconsin Institute for Discovery, University of Wisconsin-Madison.

Kolmogorov et al., 2016). In this paper we will focus on the use of quasiconvex distance functions, the use of which is much more robust to outliers (Bach, 2017)[2]. Figure 1 describes this in more detail.

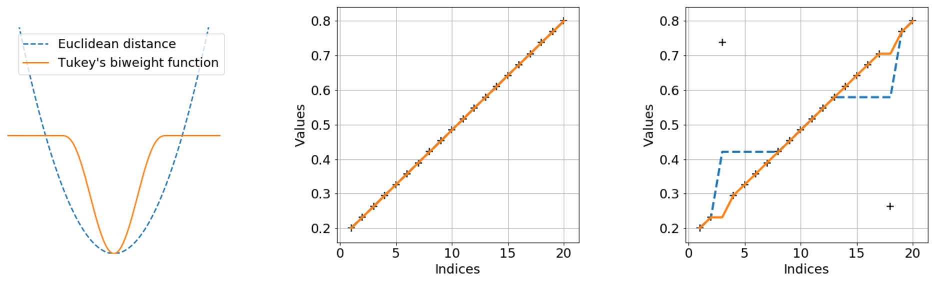

Figure 1: Isotonic regression in the presence of outliers. The left image shows the value of the Euclidean distance and Tukey's biweight function (a canonical function for robust estimation) from $x = -1$ to 1, the middle image demonstrates isotonic regression on a simple linear and noiseless example, and the right image shows how outliers can adversely affect isotonic regression under Euclidean distance.

For general $f_i$ functions we cannot solve Problem (1) exactly (without some strong additional assumptions), and instead we focus on the problem

$$\min_{x \in \mathcal{G}_k^n} \sum\nolimits_{i \in [n]} f_i(x_i) \ \text{ where } \ x_i \leq x_{i+1} \text{ for } i \in [n-1] \tag{2}$$

where instead of allowing the $x_i$ values to lie anywhere in the interval $[0, 1]$, we restrict them to $\mathcal{G}_k := \{0, 1/k, 2/k, \ldots, 1\}$, a equally-spaced grid of $k + 1$ points. This discretized version of the problem will give a feasible solution to the original problem close to optimal. The relation between the granularity of the grid and the approximation quality for any optimization problem over the $[0, 1]^n$ box can be described in terms of the Lipschitz constants of the objective function, and for this particular problem has been described in Bach (2015, 2017) — if functions $f_i$ are Lipschitz continuous with constant $L$, then to obtain a precision of $\epsilon$ in terms of the objective value, it suffices to choose $k \geq 2nL/\epsilon$. One can achieve better bounds using higher-order Lipschitz constants. The main approach for solving Problem (2) for general nonconvex functions is to use dynamic programming (see for example Felzenszwalb and Huttenlocher (2012)) that runs in $O(nk)$. When all the $f_i$ are convex, one can instead use the faster $O(n \log k)$) scaling algorithm by (Ahuja and Orlin, 2001).

Our main contribution is an algorithm that also achieves $O(nk)$ in the general case and $O(n \log k)$ in the convex case by exploiting the following key fact — *the dynamic programming method runs in time linear in the sum of possible $x_i$ values over all $x_i$*. Thus, by limiting the range of values, we can significantly improve the running time. We achieve this by combining ideas from *branch-and-bound* and the scaling algorithm by Ahuja and Orlin (2001) with the dynamic programming approach. When restricted to convex functions, our algorithm is a variant of the scaling algorithm.

Our algorithm works by solving the problem over a finer and finer set of points, choosing not to add points that we know will not get us closer to the global optimum. We use two ways to exclude points, the first of which only requires that we know the Lipschitz constants of each $f_i$, and the second requires us to be able to compute a linear underestimate of $f_i$ over an interval efficiently. This information is readily available for a variety of quasiconvex distance functions, and we provide an example of how to compute this for Tukey's biweight function. In practice, this leads to an algorithm that requires far less function evaluations to achieve the same accuracy as dynamic programming, which translates into a faster running time even considering the additional work needed in each iteration. In synthetic experiments the two methods perform similarly at $k = 128$, with the performance gap scaling to hundreds of times better for our algorithm at $k = 2^{16} = 65536$.

The paper is organized as follows. In the rest of the introduction, we will survey other methods for isotonic regression for specific classes of sets of $f_i$ functions and also mention related problems. Section 2 describes the standard dynamic programming approach. In Section 3, we describe our main pruning algorithm and the key pruning rules for removing $x_i$ values that we need to consider. Section 5 demonstrates the performance of the algorithm on a series of experiments.

[2] Our focus in this paper is on developing algorithms for global optimization. For more on robust estimators, we refer the reader to the classic textbooks by Huber (2004); Hampel et al. (2011).

**Existing methods for isotonic regression.** We will first discuss the main methods for *exactly* solving Problem (1) and the classes of functions the methods can handle. For convex $f_i$ functions, the *pool-adjacent-violators* (PAV) algorithm (Ayer et al., 1955; Brunk, 1955) has been the de facto method for solving the problem. The algorithm was originally developed for the Euclidean distance case and in fact works for any set of convex $f_i$, provided that one can exactly solve intermediate subproblems of the form $\arg\min_{z\in[0,1]} \sum_{i\in S} f_i(z)$ (Best et al., 2000) over some subset $S$. PAV requires solving up to $n$ such subproblems, and the total cost of solving can be just $O(n)$ for a wide range of functions, including for many Bregman divergences (Lim and Wright, 2016).

There are algorithms designed to handle nonconvex functions that are piecewise convex. Let $q$ denote the total number of pieces over all the $f_i$ functions. In the convex case, piecewise-linear and -quadratic functions can be handled in $O(q \log\log n)$ and $O(q \log n)$ time respectively (Kolmogorov et al., 2016; Hochbaum and Lu, 2017), , while in the general nonconvex case it is $O(nq)$.

In some cases, we cannot solve the problem exactly and instead deal with the discretized problem (2). For example, this is the case when our knowledge to the functions $f_i$ can only be obtained through function evaluation queries (i.e. $x_i \to f_i(x_i)$). In the convex case, PAV can be used to obtain an algorithm with $O(n^2 \log k)$ time to solve the problem over a grid of $k$ points, but a clever recursive algorithm by Ahuja and Orlin (2001) takes only $O(n \log k)$. A general approach that works for arbitrary functions is dynamic programming, which has a complexity of $O(nk)$.

Bach (2015) recently proposed a framework for optimizing continuous submodular functions that can be applied to solving such functions over monotonicity constraints. This includes separable nonconvex functions as a special case. Although the method is a lot more versatile, when specialized to our setting it results in an algorithm with a significantly worse complexity of $O(n^2k^2)$. This and dynamic programming are the main known methods for general nonconvex functions.

**Related Problems.** There have been many extensions and variants of the classic isotonic regression problem, and we will briefly describe two of them. One common extension is to use a partial ordering instead of a full ordering. This significantly increases the difficulty of the problem, and in the most general case can be solved by recursively solving network flow problems. For a detailed survey of this area, we refer the reader to Stout (2014). One can also replace the ordering constraints with the pairwise terms $\sum_{i\in[n-1]} g_i(x_{i+1} - x_i)$ where $g_i : \mathbb{R} \to \mathbb{R} \cup \{\infty\}$. By choosing $g_i$ appropriately, we recover many known variants of isotonic regression, including nearly-isotonic regression (Tibshirani et al., 2011), smoothed isotonic regression (Sysoev and Burdakov, 2016; Burdakov and Sysoev, 2017), and a variety of problems from computer vision. The most general recent work (involving piecewise linear functions) is by Hochbaum and Lu (2017). We note that the works by Bach (2015, 2017) also applies in many of these settings.

## 2 Dynamic Programming

Let $\mathcal{G}_k$ denote the grid $\{0, 1/k, 2/k, \ldots, 1\}$. We now provide a DP reformulation of Problem (2). Let $A_n^k(x_n) = f_n(x_n)$. For any $i \in [n-1]$, we can define the following functions:

$$A_i^k(x_i) \coloneqq f_i(x_i) + C_i^k(x_i), \qquad \text{(aggregate)}$$

$$C_i^k(x_i) \coloneqq \min_{x_{i+1}\in\mathcal{G}_k} A_{i+1}^k(x_{i+1}) \ \text{ where } \ x_i \le x_{i+1}. \qquad \text{(min-convolution)}$$

The $A_i^k$ functions *aggregate* the accumulated information from the indices $i+1, i+2, \ldots, n$ with the information at the current index $i$, where the $C_i^k$ functions represent the *minimum-convolution* of the $A_{i+1}^k$ function with the indcator function $g$ where $g(z) = 0$ if $z \le 0$, and $g = \infty$ otherwise. With this notation, the problem $\min_{x_1\in\mathcal{G}_k} A_1^k(x_1)$ has the same objective and $x_1$ values as Problem (2).

We can use the dynamic programming form to solve the problem efficiently, and we formally describe this in Algorithm 1. The dynamic programming algorithm can be viewed an application of the Viterbi algorithm to this setting. The algorithm does a backward pass, building up all the $A_i^k, C_i^k$ values from $i = n$ to $i = 1$. Once $A_1^k$ has been computed, we know the minimizer $x_1$. We then work our way fowards, each time picking an $x_i$ that minimizes $A_i^k$ subject to the condition that $x_i \ge x_{i-1}$. It is easy to see that the total running time of this algorithm is $O(nk)$.

**Algorithm 1** Dynamic Program for fixed grid $\mathcal{G}_k$

```
input: Functions $f_1, f_2, \ldots, f_n$, Parameter $k$
$A_n^k(z) \leftarrow f_n(z)$ for $z \in \mathcal{G}_k$
for $i = n-1, \ldots, 1$ do                                        ▷ Backwards Pass
    $C_i^k(1) \leftarrow A_{i+1}^k(1)$
    $A_i^k(1) \leftarrow f_i(1) + C_i^k(1)$
    for $z = {}^{k-1}/_{k}, {}^{k-2}/_{k}, \ldots, 0$ do
        $C_i^k(z) \leftarrow \min(A_{i+1}^k(z), C_i^k(z + {}^{1}/_{k}))$
        $A_i^k(z) \leftarrow f_i(z) + C_i^k(z)$
$x_0 \leftarrow 0$
for $i = 1, 2 \ldots, n$ do                                          ▷ Forward Pass
    $x_i \leftarrow \arg\min_{z \in \mathcal{G}_k, z \geq x_{i-1}} A_i^k(z)$
return $(x_1, x_2, \ldots, x_n)$
```

The main drawback of the dynamic programming approach is that it requires us to pick the desired accuracy a priori via choosing an appropriate $k$ value and the overall running time is then $O(nk)$, *no matter the properties of the $f_i$ functions.*

## 3 A Pruning Algorithm for Robust Isotonic Regression

Instead of solving the full discretized problem (2) directly, we can work over a much smaller set of points. Let $x^*$ denote an optimal solution to the problem, and for each $i \in [n]$ let $\mathcal{S}_i \subseteq \mathcal{G}_k$ denote a set of points such that $x_i^* \in \mathcal{S}_i$. Then

$$\min\nolimits_{x \in \mathcal{S}_1 \times \ldots \mathcal{S}_n} \sum\nolimits_{i \in [n]} f_i(x_i) \ \text{ where } \ x_i \leq x_{i+1} \text{ for } i \in [n-1],$$

has the same solution $x^*$ and it is easy to modify the DP algorithm to work for this problem. All that is needed is to perform the following replacements:

- $z = \ldots$ and $z \in \ldots$ with the appropriate series of points in $\mathcal{S}_i$,
- $C_i^k(z_{\max}) \leftarrow \min_{z \geq z_{\max}} A_{i+1}^k(z)$ for $z_{\max} = \arg\max(\mathcal{S}_i)$, and
- $C_i^k(z) \leftarrow \min(A_{i+1}^k(z'), C_i^k(z''))$ where $z', z'' \geq z$.

The values of the $C_i^k, A_i^k$ functions are the same for both problem formulations on the points in $\mathcal{S}_i$.

The modified operations can be performed efficiently by maintaining the appropriate minimum values, and this results in an algorithm with a complexity of just $O(|\mathcal{S}_1| + \ldots |\mathcal{S}_n|)$. Our goal is thus to restrict the size of $\mathcal{S}_i$ sets, and we do this by recursively refining the sets. For each $i \in [n]$ we will maintain a *set of intervals* $\mathcal{I}_i$ that we can explore, in addition to maintaining $\mathcal{S}_i$. From here on, we will assume that $k$ is a power of 2 for convenience. Algorithm 2 describes the basic framework.

**Algorithm 2** Algorithmic Framework for Faster Robust Isotonic Regression

```
input: Functions $f_1, f_2, \ldots, f_n$, Parameter $k$
$k' \leftarrow 1$
$\mathcal{S}_i \leftarrow \{0, 1\}$ for $i \in [n]$
$\mathcal{I}_i \leftarrow \{[0, 1]\}$ for $i \in [n]$
while $k' < k$ do
    $k' \leftarrow 2k'$
    Refine $\mathcal{S}_1, \ldots, \mathcal{S}_n$ and $\mathcal{I}_1, \ldots, \mathcal{I}_n$
$x \leftarrow$ run dynamic program on $\mathcal{S}_1, \ldots, \mathcal{S}_n$
return $x$
```

At the end of each round of the loop, we want the minimizer $x^*$ for the subproblem $\mathcal{G}_{k'}$ be contained in $\mathcal{S}_1 \times \ldots \times \mathcal{S}_n$. This will ensure that we find the optimal point in the final grid $\mathcal{G}_k$. We also want (i)) $\mathcal{S}_1, \ldots, \mathcal{S}_n \subseteq \mathcal{G}_{k'}$, and (ii) $\mathcal{I}_i$ to consist only of intervals of width ${}^{1}/_{k}$ with endpoints contained in $\mathcal{S}_i$. These two conditions ensures that the overall number of points considered over all the $\mathcal{S}_i$ over all iterations is at most $O(nk)$, and by bounding the sizes of the $\mathcal{S}_i$ in each iteration we can achieve a significantly better complexity bound. In particular, the scaling algorithm for convex functions by Ahuja and Orlin (2001) can be seen as a particular realization of this framework where the refinement process keeps the size of each $\mathcal{S}_i$ to exactly two.

In the rest of this section, we will describe two efficient rules for refining the sets $\mathcal{S}_i$ and $\mathcal{I}_i$ and analyze the complexity of the overall algorithm. The first rule requires knowledge of the Lipschitz constant $L$ of the $f_i$ functions, while the second requires one to be able to efficiently construct good linear underestimators of the $f_i$ functions within intervals.

## 3.1 Pruning via lower/upper bounds

Recall from the discussion in the introduction that we can bound the error of the objective value by utilizing the Lipschitz constant $L$ of the $f_i$ functions. In particular this pruning rule will use the fact that the objective value for Problem (2) is within $0.5nL/k'$ of Problem (1) when using the grid $\mathcal{G}_{k'}$.

**Lemma 3.1.** *Suppose there exists points $a, b \in \mathcal{G}'_k$ such that $A_1^{k'}(a) + 0.5nL/k' < A_1^{k'}(b)$. Then, for any optimal solution $x^*$ to Problem* (1) *we have $|x_1^* - b| > 0.5/k'$.*

*Proof.* Let $f(x)$ denote the objective $\sum_{i\in[n]} f_i(x_i)$. Consider a point $y$ where $|y_1 - b| \leq 0.5/k'$. Let $y^b$ denote the vector obtained from rounding each point in $y$ to the nearest point in $\mathcal{G}_{k'}$. This new point still satisfies the monotonicity constraints, and since $y_i^b = b$, we have $A_1^{k'}(b) \leq f(y^b)$, and so from the Lipschitz constant we have $f(y^b) \leq f(y) + 0.5nL/k'$ and

$$A_1^{k'}(a) < A_1^{k'}(b) - 0.5nL/k' \leq f(y^b) - 0.5nL/k' \leq f(y),$$

which means $y$ is not optimal. □

Lemma 3.1 allows us to prune or remove grid points that are not close to a global optimal solution, and also remove intervals that we know will not contain a global optimal $x_i$. We can also generalize the results to apply to all indices $i$ and not just for $x_1$. Algorithm 3 does this recursively by first pruning the points for $\mathcal{S}_i$, and then using the trimmed $\mathcal{S}_i$ to decide which points to prune for $\mathcal{S}_{i+1}$.

**Algorithm 3** Pruning $\mathcal{S}, \mathcal{I}$ via Lower/Upper Bounds

```
input: Functions $A_1^{k'}, A_2^{k'}, \ldots, A_n^{k'}$, Parameter $k'$, Sets $\mathcal{S}_1, \ldots, \mathcal{S}_n, \mathcal{I}_1, \ldots, \mathcal{I}_n$
$z' \leftarrow \arg\min_{z\in S_1} A_1^{k'}(z)$
$\mathcal{S}_1 \leftarrow \{z \in S_1 \mid A_1^{k'}(z') + 0.5nL/k \geq A_1^{k'}(z)\}$
$\mathcal{I}_1 \leftarrow \{[a,b] \in \mathcal{I}_1 \mid [a,b]$ has at least one endpoint in $\mathcal{S}_1\}$
for $i = 2, \ldots, n$ do
    $\mathcal{S}' \leftarrow \emptyset$
    for $z_j \in \mathcal{S}_{i-1}$ (in ascending order) do
        $\mathcal{T} \leftarrow \{z \in S_i \mid z_j \leq z < z_{j+1}\}$
        $z' \leftarrow \arg\min_{z\in\mathcal{T}} A_i^{k'}(z)$
        $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{z \in \mathcal{T} \mid A_i^{k'}(z') + 0.5(n-i+1)L/k \geq A_i^{k'}(z)\}$
    $\mathcal{S}_i \leftarrow \mathcal{S}'$
    $\mathcal{I}_i \leftarrow \{[a,b] \in \mathcal{I}_i \mid [a,b]$ has at least one endpoint in $\mathcal{S}_i\}$
return $\mathcal{S}_1, \ldots, \mathcal{S}_n$
```

Since Algorithm 3 only removes points that are not close to the optimal point, we have the following:

**Proposition 3.2.** *Suppose there is an optimal solution $x^*$ to Problem* (2) *such that $\|x^* - x'\|_\infty \leq 0.5/k'$ for some $x' \in \mathcal{S}_1 \times \ldots \mathcal{S}_n$ and $x_i^*$ is contained in some interval in $\mathcal{I}_i$ for all $i \in [n]$. Then, both conditions still hold after applying Algorithm 3.*

## 3.2 Pruning via linear underestimators

The previous pruning rule used information on function values on the points in $\mathcal{S}_i$. We now describe a rule that uses *linear underestimators on intervals in $\mathcal{I}_i$*. In the convex case, one can think of this as using subgradient information [3]. This is what the scaling algorithm of Ahuja and Orlin (2001) uses to obtain a complexity of $O(n \log k)$. We assume that we can construct these estimators eficiently:

**Assumption 3.3.** *Given functions $f_1, \dots, f_n : [0,1] \to \mathbb{R}$, for any given $x \in [0,1]$ and any $i$, we can compute in constant time $m_L, m_R \in \mathbb{R}$ such that $f_i(x) + m_L(z - x) \leq f_i(z)$ for $0 \leq z < x$ and $f_i(x) + m_R(z - x) \leq f_i(z)$ for $x < z \leq 1$.*

As stated, the pruning rule works with any $m$ that satisfies the condition, but the tighter the underestimator, the better our algorithm will perform. In particular, it is ideal to minimize $m_L$ and maimize $m_R$. In the case of convex functions, the best possible $m$ is a subgradient of the function $f_i$ at $x$.

It is straightforward to compute good linear underestimators for many quasiconvex distance functions used in robust statistics. We will discuss how this can be done for the Tukey biweight function, and similar steps can be taken for other popular functions such as the Huber Loss, SCAD, and MCP.

**Example: Tukey's biweight function and how to efficiently compute good m values.** Tukey's biweight function is a classic function used in robust regression. The function is zero at the origin and the derivative is $x(1 - (x/c)^2)^2$ for $|x| < c$ and 0 otherwise for some fixed $c$.

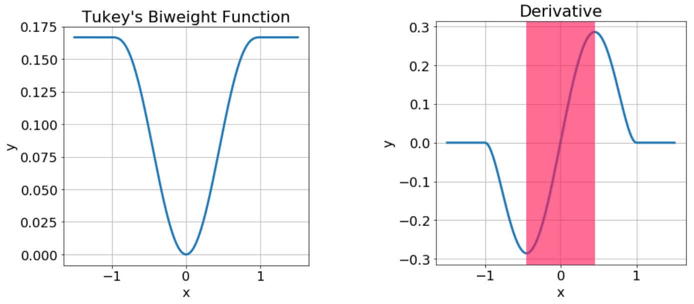

Figure 2: Tukey's biweight function with $c = 1$. In the plot of the derivative, we mark the region in which the function is convex in red ($-1/\sqrt{5} \leq x \leq 1/\sqrt{5}$), while in the other regions at the sides the function is concave.

We will describe how to choose $m_L$ and $m_R$ for $x < 0$, and by symmetry we can use similar methods for $x > 0$. We obtain $m_L$ from connecting $f(x)$ to the largest value of the function. If $x$ is in the convex region, we can simply set $m_R$ to the gradient. We now add a line with slope $-L$ (where $L$ is the largest gradient of the function) to the transition point between the concave and convex regions, and for $x$ in the concave region we obtain $m_R$ by connecting $f(x)$ to this line.

**Using linear underestimators to prune.** Suppose we have the point $z \in \mathcal{S}_i$, and $[z - \delta, z], [z, z + \delta] \in \mathcal{I}_i$ for $i = a, a+1, \dots, b$ and our goal is to decide if we should refine points in the first interval, the second interval, or both. We can perform the procedure in Algorithm 4 to decide which cases to consider. The correctness of this procedure follows from the fact that when it is beneficial to increase $x_i$ (i.e. the interval $[x_i, x_i + \delta]$ has corresponding $m < 0$), we should increase all subsequent terms $x_{i+1}, \dots$ due to the monotonicity constraints. Note that this process partitions the indices $a, a+1, \dots, b$ into three consecutive blocks: (1) where it is only beneficial to consider $[z - \delta, z]$, (2) unclear which is better, (3) only beneficial to consider $[z, z + \delta]$.

We can further embed Algorithm 4 into a larger procedure over the entire collection of interval sets $\mathcal{I}_1, \dots, \mathcal{I}_n$. This procedure is detailed in Algorithm 5, and refines the set of intervals by splitting each interval into two and running Algorithm 4 on the pair of adjacent intervals.

**Lemma 3.4.** *Suppose there is an optimal solution $x^*$ to Problem (2) such that $x_i^*$ is contained in some interval in $\mathcal{I}_i$ for all $i \in [n]$. Then, both conditions still hold after applying Algorithm 5.*

**Proposition 3.5.** *Suppose all the $f_i$ are convex, $g_i(x, y)$ returns a subgradient at $x$, and each $\mathcal{I}_i$ contains exactly one interval. Suppose there is an optimal solution $x^*$ to Problem (2) such that $x_i^*$ is contained in the interval in $\mathcal{I}_i$ for all $i \in [n]$. Then, after applying Algorithm 5, the solution is still contained in the intervals and each $\mathcal{I}_i$ remains a singleton set.*

[3]More precisely, we only need subgradients for the linear interpolation of the functions $f_i$ over the target grid $\mathcal{G}_k$, and we can also modify Assumption 3.3 accordingly. To simplify the presentation, we will work with the original functions in the main text.

**Algorithm 4** Pruning Subroutine via Linear Underestimators

```
input: Functions $g_a, g_{a+1}, \dots, g_b$ that compute $m$ in Assumption 3.3, intervals $[z-\delta, z]$ and $[z, z+\delta]$
$L(b) \leftarrow g_b(z, z-\delta)$
$R(b) \leftarrow g_b(z, z+\delta)$
for $i = b-1, \dots, a$ do
    $L(i) \leftarrow g_b(z, z-\delta) + \max(L(i+1), 0)$
    $R(i) \leftarrow g_b(z, z+\delta) + \max(R(i+1), 0)$
mode $\leftarrow$ ‘left only’
for $i = a, a+1, \dots, b$ do
    if $L(i) < 0$ and $R(i) < 0$ then
        mode $\leftarrow$ ‘right only’
    if mode is ‘right only’ then
        $\mathcal{I}_i \leftarrow \{[z, z+\delta]\}$
        continue
    if R(i) < 0 then
        mode $\leftarrow$ ‘both’
    if mode is ‘both’ then
        $\mathcal{I}_i \leftarrow \{[z-\delta, z], [z, z+\delta]\}$
        continue
    $\mathcal{I}_i \leftarrow \{[z-\delta, z]\}$
return $\mathcal{I}_a, \mathcal{I}_{a+1}, \dots, \mathcal{I}_b$
```

**Algorithm 5** Main Algorithm for Refining via Linear Underestimators

```
input: Functions $g_1, g_2, \dots, g_n$ that compute $m$, Interval sets $\mathcal{I}_1, \dots, \mathcal{I}_n$
Let $\mathcal{I}([u, v])$ denote $\{i \mid \mathcal{I}_i$ contains $[u, v]\}$
$\mathcal{I}'_i \leftarrow \emptyset$ for $i \in [n]$
for $[u, v] \in \bigcup_i \mathcal{I}_i$ do
    for each contiguous block of indices $a, a+1, \dots, b$ in $\mathcal{I}([u, v])$ do
        Update $\mathcal{I}'_i$ with intervals from Algorithm 4 on intervals $[u, (u+v)/2]$ and $[(u+v)/2, v]$
$\mathcal{I}_i \leftarrow \mathcal{I}'_i$ for $i \in [n]$
for $i = 1, 2, \dots, n$ do
    $\mathcal{S}_i \leftarrow \{z \mid z$ is an endpoint in some interval in $\mathcal{I}_i\}$
return $\mathcal{S}_1, \dots, \mathcal{S}_n, \mathcal{I}_1, \dots, \mathcal{I}_n$
```

### 3.3 Putting it all together

After stating the pruning and refinement rules for our nonconvex distance functions, we can formally describe in detail the full process in Algorithm 6.

**Algorithm 6** A Pruning Algorithm for Robust Isotonic Regression

```
input: Functions $f_1, f_2, \dots, f_n$, Parameter $k$
$k' \leftarrow 1$
$\mathcal{S}_i \leftarrow \{0, 1\}$ for $i \in [n]$
$\mathcal{I}_i \leftarrow \{[0, 1]\}$ for $i \in [n]$
while $k' < k$ do
    $k' \leftarrow 2k'$
    Run DP on $\mathcal{S}_1, \dots, \mathcal{S}_n$ to obtain $A_i^{k'}$ values
    Use Algorithm 3 with $A_i^{k'}$ to prune $\mathcal{S}_i, \mathcal{I}_i$ for $i \in [n]$
    Use Algorithm 5 to refine and prune $\mathcal{S}_i, \mathcal{I}_i$ for $i \in [n]$
$x \leftarrow$ run DP on $\mathcal{S}_1, \dots, \mathcal{S}_n$
return $x$
```

**Theorem 3.6.** *Algorithm 6 solves Problem* (2) *in $O(nk)$ time in general, and $O(n \log k)$ time for convex functions if we use subgradient information.*

There are two things to note about Algorithm 6. First, it only presents one possible combination of the pruning rules, and they could in fact be combined in different ways. For example, we could choose not to apply the lower/upper bound pruning rule at every iteration. For simplicity, we stick to this particular description in our experiments and theorems. Second, we only require the linear underestimator rule for the $O(n \log k)$ convex bound, since that suffices to ensure that sets $\mathcal{S}_i$ have at most a few points.

## 4 Empirical Observations

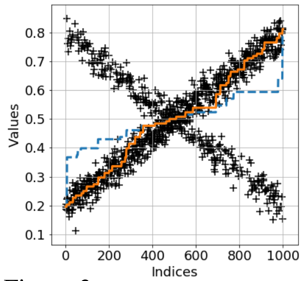

Figure 3: The $y_i$ points (pluses) and results of using Euclidean distance (blue, dashed) vs. Tukey's biweight function (orange, solid).

We evaluate the efficiency of the DP approach and our algorithm on a simple isotonic regression task. We adopt an experiment setup similar to the one used by Bach (2017). We generate a series of $n$ points $y_1, \ldots, y_n$ from 0.2 to 0.8 equally spaced out and added Gaussian random noise with standard deviation of $0.03$. We then randomly flipped $30\%$ of the points around 0.5, and these points act as the outliers. Our goal now is to test the computational effort required to solve the problem

$$\min_{x \in \mathcal{G}_k^n} \sum_{i \in [n]} f(x_i - y_i) \text{ where } x_i \leq x_{i+1} \text{ for } i \in [n-1]$$

where $f$ is the Tukey's biweight function with $c = 0.3$. We set $n$ to 1000 and varied $k$ from $2^7 = 128$ to $2^{16} = 65536$.

We used two metrics to evaluate the computational efficiency. The first measure we use is the total number of points in all $\mathcal{S}_i$ across all iterations, an implementation-independent measure. The second is the wall-clock time taken. The algorithms were implemented in Python 3.6.2, and the experiments were ran on an Intel 4th generation core i5-4200U dual-core processor with 8GB of RAM.

The main empirical observations are summarized in Figure 4, where the results are average over 100 independent trials. In the first figure on the left, we see how the error decreases with an increase in $k$, reflecting the equation that $k \geq O(1/\epsilon)$ is needed to achieve an error of $\epsilon$ in the objective. In the middle figure, we see that the number of points generated throughout the course of the pruning algorithm scales logarithmically with $k$, showing that even though we are using a nonconvex estimator, the computational results reflect what we expect to see in the convex case. We omit the graph comparing the running time with $k$ since the running time scales approximately linearly with the number of points evaluated. Finally, for the graph on the right we see how the difference in running time between the two methods almost linearly increases with an increase with $k$. Note that at $k = 128$, the two methods are almost similar in running time (with the pruning algorithm being approximately 10% faster), reflecting the overhead incurred in the pruning process. Also, it is possible to further speed up the running time of the pruning algorithm by tweaking the frequency of the rules, namely by reducing the frequency of the lower bounding rule.

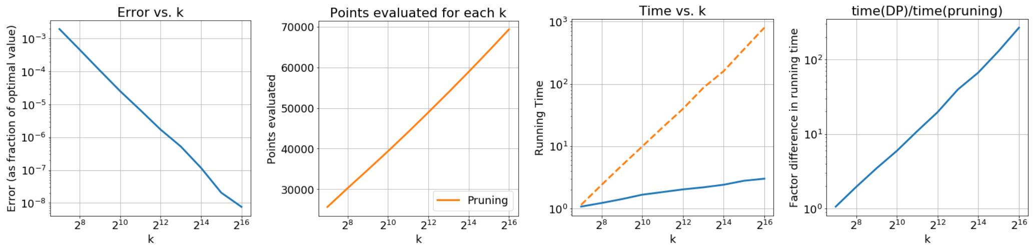

Figure 4: Empirical results. The graph on the left shows how increasing the granularity of the grid decreases the error, the next graph in the center shows how the number of points evaluated increased logarithmically with the number $k$, the third graph shows how running times of the two algorithms changes with $k$, and the graph on the right compares the running time of the two algorithms.

In addition to the above experiments, we also ran experiments varying the value of $n$. As predicted by the theory, the complexity of both methods scale roughly linearly with $n$. Tests on a range of quasiconvex robust estimators shows similar results.

## 5 Conclusions and Future Work

We propose a fast pruning algorithm that builds upon the standard DP algorithm for solving the separable nonconvex isotonic regression problem (1) to any arbitrary accuracy (to the global optimal value). On the theoretical front, we demonstrate that the pruning rules we develop retain the correct points and intervals required to reach the global optimal value, and in the convex case our algorithm becomes a variant of the fast $O(n \log k)$ scaling algorithm. In terms of empirical performance, our initial synthetic experiments show that our algorithm scales significantly better with the desired accuracy level, approximately logarithmically compared to the DP algorithm.

Besides developing more pruning rules that can work on a larger range of nonconvex $f_i$ functions, there are two main directions for extensions to this work, mirroring the line of developments for the classic isotonic regression problem. The first is go beyond monotonicity constraints and instead consider chain functions $g_i(x_i - x_{i+1})$ that link together adjacent indices. A particularly interesting case is the one where $g_i(x_i)$ incorporates a $\ell_2$-penalty in addition to the monotonicity constraints in order to promote smoothness. The second is to go from the full chain ordering we consider here to general partial orders. In that setting, dynamic programming approaches fail, and we would require a significantly different approach. It may be possible to adapt the general submodular-based approach developed by Bach (2017), which works in both the above mentioned extensions.

## Acknowledgements

The author was partially supported by NSF Award CMMI-1634597, NSF Award IIS-1447449 at UW-Madison. Part of the work was completed while visiting the Simons Institute for the Theory of Computing (partially supported by the DIMACS/Simons Collaboration on Bridging Continuous and Discrete Optimization through NSF Award CCF-1740425).

## References

Ahuja, R. K. and Orlin, J. B. (2001). A Fast Scaling Algorithm for Minimizing Separable Convex Functions Subject to Chain Constraints. *Operations Research*, 49(5):784–789.

Ayer, M., Brunk, H. D., Ewing, G. M., Reid, W. T., and Silverman, E. (1955). An Empirical Distribution Function for Sampling with Incomplete Information. *The Annals of Mathematical Statistics*, 26(4):641–647.

Bach, F. (2013). Learning with submodular functions: A convex optimization perspective. *Foundations and Trends® in Machine Learning*, 6(2-3):145–373.

Bach, F. (2015). Submodular Functions: from Discrete to Continous Domains. *arXiv:1511.00394 [cs, math]*. arXiv: 1511.00394.

Bach, F. (2017). Efficient Algorithms for Non-convex Isotonic Regression through Submodular Optimization. *arXiv:1707.09157 [cs, stat]*. arXiv: 1707.09157.

Best, M. J., Chakravarti, N., and Ubhaya, V. A. (2000). Minimizing Separable Convex Functions Subject to Simple Chain Constraints. *SIAM Journal on Optimization*, 10(3):658–672.

Bogdan, M., van den Berg, E., Su, W., and Candes, E. (2013). Statistical estimation and testing via the sorted L1 norm. *arXiv:1310.1969*.

Brunk, H. D. (1955). Maximum Likelihood Estimates of Monotone Parameters. *The Annals of Mathematical Statistics*, 26(4):607–616.

Burdakov, O. and Sysoev, O. (2017). A Dual Active-Set Algorithm for Regularized Monotonic Regression. *Journal of Optimization Theory and Applications*, 172(3):929–949.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2012). Distance Transforms of Sampled Functions. *Theory of Computing*, 8:415–428.

Gunasekar, S., Koyejo, O. O., and Ghosh, J. (2016). Preference Completion from Partial Rankings. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 1370–1378. Curran Associates, Inc.

Hampel, F. R., Ronchetti, E. M., Rousseeuw, P. J., and Stahel, W. A. (2011). *Robust statistics: the approach based on influence functions*, volume 196. John Wiley & Sons.

Hochbaum, D. and Lu, C. (2017). A Faster Algorithm Solving a Generalization of Isotonic Median Regression and a Class of Fused Lasso Problems. *SIAM Journal on Optimization*, 27(4):2563–2596.

Hochbaum, D. S. (2001). An Efficient Algorithm for Image Segmentation, Markov Random Fields and Related Problems. *J. ACM*, 48(4):686–701.

Huber, P. (2004). *Robust Statistics*. Wiley Series in Probability and Statistics - Applied Probability and Statistics Section Series. Wiley.

Kalai, A. and Sastry, R. (2009). The Isotron Algorithm: High-Dimensional Isotonic Regression. In *Conference on Learning Theory*.

Kolmogorov, V., Pock, T., and Rolinek, M. (2016). Total Variation on a Tree. *SIAM Journal on Imaging Sciences*, 9(2):605–636.

Lim, C. H. and Wright, S. J. (2016). Efficient bregman projections onto the permutahedron and related polytopes. In Gretton, A. and Robert, C. C., editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS 2016)*, page 1205–1213.

Stout, Q. F. (2014). Fastest isotonic regression algorithms.

Suehiro, D., Hatano, K., Kijima, S., Takimoto, E., and Nagano, K. (2012). Online prediction under submodular constraints. In Bshouty, N., Stoltz, G., Vayatis, N., and Zeugmann, T., editors, *Algorithmic Learning Theory*, volume 7568 of *Lecture Notes in Computer Science*, pages 260–274. Springer Berlin Heidelberg.

Sysoev, O. and Burdakov, O. (2016). *A Smoothed Monotonic Regression via L2 Regularization*. Linköping University Electronic Press.

Tibshirani, R. J., Hoefling, H., and Tibshirani, R. (2011). Nearly-Isotonic Regression. *Technometrics*, 53(1):54–61.

Yasutake, S., Hatano, K., Kijima, S., Takimoto, E., and Takeda, M. (2011). Online linear optimization over permutations. In Asano, T., Nakano, S.-i., Okamoto, Y., and Watanabe, O., editors, *Algorithms and Computation*, volume 7074 of *Lecture Notes in Computer Science*, pages 534–543. Springer Berlin Heidelberg.

Zeng, X. and Figueiredo, M. A. T. (2014). The Ordered Weighted $\ell_1$ Norm: Atomic Formulation, Projections, and Algorithms. *arXiv:1409.4271*.