Lipschitz Unimodal and Isotonic Regression on Paths and Trees*

Pankaj K. Agarwal Jeff Phillips Bardia Sadri

Department of Computer Science
Duke University
{pankaj, jeffp, sadri}@cs.duke.edu

Abstract

Given a sequence $\boldsymbol{t} = (t_1, ..., t_n)$ of n values, the solution to the Lipschtiz isotonic regression (LIR) problem finds another sequence $\boldsymbol{s} = (s_1, ..., s_n)$ that satisfies $s_i \leq s_{i+1} \leq s_i + \gamma$, for some Lipschitz parameter γ, while minimizing $\sum_{i=1}^{n}(t_i - s_i)^2$. This is a generalization of the so-called isotonic regression (IR) problem, where the second (Lipschitz) constraint is not given (i.e. $\gamma = \infty$). We solve the LIR problem in $O(n \log n)$ time. We also extend this framework to and provide algorithms for data that is given as a rooted tree on which every leaf to root path is expected to be a γ-Lipschitz isotonic sequence. We also study the Lipschitz unimodal regression (LUR) problem in which choosing the optimal root for the tree is part of the problem. We provide an $O(n \operatorname{polylog}(n))$ algorithm for solving Lipschitz unimodal regression (LUR) problem on paths and an $O(n^{3/2} \operatorname{polylog}(n))$ algorithm for trees.

*Research supported by NSF under grants CNS-05-40347, CFF-06-35000, and DEB-04-25465, by ARO grants W911NF-04-1-0278 and W911NF-07-1-0376, by an NIH grant 1P50-GM-08183-01, by a DOE grant OEG-P200A070505, and by a grant from the U.S.–Israel Binational Science Foundation.

1 Introduction

Given a sequence of obervations $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, it is a common practice in statistical analysis to relate a dependent (response) variable y into one or more independent (explanatory) variables $\boldsymbol{x}$ by fitting a function $y = f(\boldsymbol{x})$ through the observations, especially when there is some prior knowledge of the class of functions according to which $\boldsymbol{x}$ and y are related. If no function of the given class can be entirely fitted to the input, a fitting error is measured and the goal of the fitting procedure, commonly called *regression* in statistical jargon, is to minimize this error. *Linear regression* is a prominent example of this procedure in which a linear function is fitted to the existing data and an error measure such as least mean squares is applied.

The family of functions used in a regression problem are often characterized algebraically; for example one can take the family to be the set of polynomials of a constant degree d. However, this characterization can also be of a topological nature, for example, one can consider functions with a bounded number of minima, maxima, or saddles. *Isotonic regression (IR)* is an example of such a case where a *non-decreasing* function is fitted to the input data. In fact, one can specify certain topological relationships among the points at which observations are made. For example, they can be the vertices of a directed acyclic graph and the family of functions of interest can be those that increase along a directed path on the graph.

Problem statement. Consider a partial order $\prec$ on the set $\{1, \ldots, n\}$. It can represented as a *directed acyclic graph (DAG)* $\mathcal{D}$ with vertex set $\{1, \ldots, n\}$ in which there is an edge from j to i if and only if $j \prec i$. A *value assignment* to vertices of $\mathcal{D}$ is simply a sequence $\boldsymbol{s} = (s_1, \ldots, s_n) \in \mathbb{R}^n$ where s_i is a said to be *assigned* to the vertex i. With respect to $\mathcal{D}$ and a for a fixed parameter γ, a value assignment $\boldsymbol{s}$ is said to be

(I) *isotonic*, if $j \prec i$ implies $s_j \leq s_i$, and

(L) γ-*Lipschitz*, if $j \prec i$ implies $s_j + \gamma \geq s_i$.

Given a value assignment $\boldsymbol{t} = (t_1, \ldots, t_n)$ to vertices of $\mathcal{D}$, the γ-*Lipschitz isotonic regression (γ-LIR)* of $\boldsymbol{t}$ with respect to $\mathcal{D}$ is an assignment $\boldsymbol{s}$ that is γ-Lipschitz and isotonic and minimizes $\|\boldsymbol{s} - \boldsymbol{t}\|^2 = \sum_{i=1}^{n}(s_i - t_i)^2$. Given $\mathcal{D}$, $\boldsymbol{t}$, and γ, the LIR problem is to compute the γ-LIR of $\boldsymbol{t}$. *Isotonic regression (IR)* is the special case of the problem in which $\gamma = \infty$, and therefore only the isotonicity conditions (I) is enforced.

The γ-LIR problem can be regarded as that of finding the point closest to $\boldsymbol{t}$ in the convex set that is the intersection of the halfspaces determined by the given isotonicity and Lipschitz constraints. In other words, one grows an n-dimensional ball centered at $\boldsymbol{t}$ until the ball touches the convex region determined by the given constraints at the point $\boldsymbol{s}$. This immediately implies that the problem always has a unique solution.

An important special case of a DAG that we consider in this paper is a rooted tree, i.e. a tree with some node marked as root on which $i \prec j$ implies that j is the *parent* of i meaning that it is closer to the root than i. Essentially, on a rooted tree, we must solve the path LIR problem simultaneously on all the leaf to root paths. Given an undirected tree, the choice of a vertex as the root immediately results in a rooted tree by letting $i \prec j$ whenever i and j are adjacent and j is closer to the root than i. A related problem to IR on rooted trees is the problem of *unimodal regression (UR)* on undirected trees where given a value assignment $\boldsymbol{t}$ to the vertices of the tree, the goal is to choose a root for which the IR regression of $\boldsymbol{t}$, with respect to the resulting rooted tree, minimizes $\|\boldsymbol{s} - \boldsymbol{t}\|$ among all the choices of the root. The *Lipschitz unimodal regression (LUR)* is defined by adding the Lipschitz condition. As a special case, one can define the UR and LUR problems on an undirected path. Note that if we have an algorithm for solving LIR problem on a tree, we can use it n times, once with each node chosen as the root, and pick the best solution as the solution to LUR. Thus computing the LUR problem this way costs a linear factor more than LIR and the challenge in solving LUR is to do better than this.

In this paper, we study the LIR problem on two special types of DAGs, namely paths and rooted trees. A motivating practical problem for studying of these regressions, particularly on paths and trees, is the problem of optimally *carving* a terrain with the purpose of eliminating unimportant (low persistence) critical points, which are caused by measurement error, interpolation artifacts, and man-made features (e.g. bridges). In many applications involving polyhedral terrains, most notably in flow routing, one wishes to eliminate the effect of "shallow pits" (which lead to flow networks being fragmented) by "carving" narrow valleys on the terrain that connect shallow pits to deeper ones, thus allowing them to "drain". As a result, the computed flow network is less influenced by insignificant terrain noise (see e.g. [10]). It turns out that the carvings happen on a number of trees embedded on the terrain where the heights of the vertices of each tree are to be changed to vary monotonically towards a chosen "root" for for that tree. In other words, to perform the carving, we need to solve the IR problem on each tree. The downside of doing so is that the optimal IR solution happens to be a step function along each path toward the root of the tree with potentially large jumps. Enforcing the Lipschitz condition prevents sharp jumps in function value and thus provides a more natural solution to the problem.

Related work. The IR problem has been studied in statistics [2, 3] since the 1950s. It has many applications ranging from statistics [11] to bioinformatics [3] to operations research [7] to differential optimization [6]. Ayer *et. al.* [2] famously solves the IR problem on paths in $O(n)$ time using the Pool Adjacent Violator Algorithm (PAVA). The algorithm works by initially treating each vertex as a level set and merging consecutive level sets that are out of order. It can be shown (see e.g. [9]) that this algorithm is correct regardless of the order of the merges. Brunk [4] and Thompson [12] initiated the study of the IR problem on general DAGs and trees, respectively. Stout [11] solves the UR problem on paths in $O(n)$ time. Pardalos and Xue [8] give an $O(n \log n)$ algorithm for the IR problem on trees. For the special case when the tree is a star they give an $O(n)$ algorithm. Maxwell and Muckstadt [7] give an $O(n^4)$ time algorithm for the IR problem on DAGs. The problems can be solved under the L_1 and L_∞ norms on paths [11] and DAGs [1] as well, with an additional $\log n$ factor for L_1. To our knowledge there is no prior mention of Lipschitz isotonic regressions in the literature.

Our results. We present an $O(n \log n)$ algorithm for computing the Lipschitz isotonic regression on a path of length n (Section 2), and an $O(n \log^2 n)$ algorithm on a tree with n nodes (Section 3). We then present an $O(n \log^2 n)$ algorithm for computing the Lipschitz unimodel regression problem on a path of length n (Section 4). Our algorithm can be extended to solve the LUR problem on an unrooted tree. Currently the running time is $O(n^{3/2} \log^2 n)$, but we believe that it can be improved to $O(n \log^{O(1)} n)$. In addition to proving a few structural properties of LIR, our algorithms take advantage of a special kind of binary search tree, which we call *affine composition tree*. It stores pairs of real numbers and supports a number of operation in addition to classical insertion, deletion, and lookup in $O(\log n)$ time. In particular, one can apply an INTERVALTRANSFORM operation that modifies all the pairs stored in the tree, whose selected (first or second) component lies within a given interval, by applying an affine transformation to them. Moreover, these trees support an operation called THRESHOLDTRANSFORM that apply an affine transformation to all elements for which a selected component exceed a given threshold while applying a second affine transformation to the rest of the pairs. Details on affine composition trees and their supported operations are provided in Section 5.

Our algorithms can be generalized in a number of ways, as listed below, without affect their asymptotic running times. However, for the simplicity of the exposition, we restrain ourselves to the simple version of the problem stated above.

- One can specify different Lipschitz constants for each Lipschitz constraint, thus writing the Lipschitz constraint involving s_i and s_{i+1} as $s_{i+1} \leq s_i + \gamma_i$.

- Isotonicity constraints can be regarded as (backward) Lipschitz constraints for which the Lipschitz constant is zero. One can allow this constant to be chosen arbitrarily and indeed permit different constants for different isotonicity constraints. In other words, the constraint $s_i \leq s_{i+1}$ can be replaced with $s_{i+1} \geq s_i + \delta_i$.
- The ℓ_2 norm $\|\cdot\|$ can be replaced with a "weighted" version $\|\cdot\|_{\boldsymbol{\lambda}}$ for any $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^n$ where
$$\|(x_1, \ldots, x_n)\|_{\boldsymbol{\lambda}}^2 = \sum_{i=1}^{n} \lambda_i x_i.$$

Due to lack of space, proofs of some of the lemmas are included in an appendix.

2 LIR on Paths

In this section we describe an algorithm for solving Lipschitz isotonic regressions on paths, where a path is a total order $1 \prec 2 \prec \cdots \prec n$ on n vertices. A point $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ can be regarded as an ordered sequence $x_1, \ldots, x_n$ of real numbers. In this paper we thus use the terms *point* and *sequence* interchangeably. Thus a sequence $\boldsymbol{x} \in \mathbb{R}^n$ is isotonic if $x_{i+1} \geq x_i$, and γ-Lipschitz for some real constant γ if $x_{i+1} \leq x_i + \gamma$ for each $i = 1, \ldots, n-1$. For the rest of this section we refer to the input sequence as $\boldsymbol{t} = (t_1, \ldots, t_n)$ and the optimal solution is represented by the sequence $\boldsymbol{s} = (s_1, \ldots, s_n)$. With the input sequence $\boldsymbol{t}$ fixed, we define the *energy function* $E : \mathbb{R}^n \to \mathbb{R}$ as

$$E(x_1, \ldots, x_n) = \|\boldsymbol{x} - \boldsymbol{t}\|^2 = \sum_{i=1}^{n} (x_i - t_i)^2.$$

Let $\Gamma \subset \mathbb{R}^n$ be the convex set of feasible solutions to the γ-LIR problem, i.e. the intersection of the $2n-2$ halfspaces: $x_{i-1} \leq x_i, x_i \leq x_{i-1} + \gamma$ for $1 < i \leq n$. By definition, the solution $\boldsymbol{s}$ to the γ-LIR problem minimizes E among all γ-Lipschitz isotonic sequences, i.e. $\boldsymbol{s} = \operatorname{argmin}_{\boldsymbol{x} \in \Gamma} E(\boldsymbol{x})$. As mentioned in Introduction, the function E is convex and therefore has a unique minimizer in the convex set Γ. Consider now the functions $E_i : \mathbb{R}^i \to \mathbb{R}$, $i = 1, \ldots, n$ defined as $E_i(x_1, \ldots, x_i) = \sum_{j=1}^{i} (x_j - t_j)^2$, and define for each $i = 1, \ldots, n$, the function $\tilde{E}_i : \mathbb{R} \to \mathbb{R}$ as

$$\tilde{E}_i(x) = \min_{\boldsymbol{x} = (x_1, \ldots, x_{i-1}, x) \in \Gamma_i} E_i(\boldsymbol{x}),$$

where $\Gamma_i = \{(x_1, \ldots, x_i) \in \mathbb{R}^i \mid x_{j-1} \leq x_j \leq x_{j-1} + \gamma, \forall j \leq i\}$. Intuitively, $\tilde{E}(x)$ is the minimum *energy* of a γ-Lipschitz isotonic point $(x_1, \ldots, x_i) \in \mathbb{R}^i$ that fixes $x_i = x$. By the definition of $\tilde{E}_i$, if we set $\tilde{E}_0 = 0$, then for $i \geq 1$:

$$\tilde{E}_i(x) = \min_{x - \gamma \leq y \leq x} \tilde{E}_{i-1}(y) + (x - t_i)^2. \tag{1}$$

Lemma 2.1 *The function $\tilde{E}_i$ is continuous, piecewise quadratic, and strictly convex for all i.*

Proof: The strict convexity of $\tilde{E}_i$ directly follows from that of E_i; the simple proof of this claim is omitted from this abstract.

Since $\tilde{E}_{i-1}$ is strictly convex, it has a unique minimum, say at $x = s^*_{i-1}$, and is strictly decreasing on $(-\infty, s^*_{i-1})$ and strictly increasing on $(s^*_{i-1}, +\infty)$. Thus depending on whether $s^*_{i-1} < x - \gamma$, $s^*_{i-1} \in$

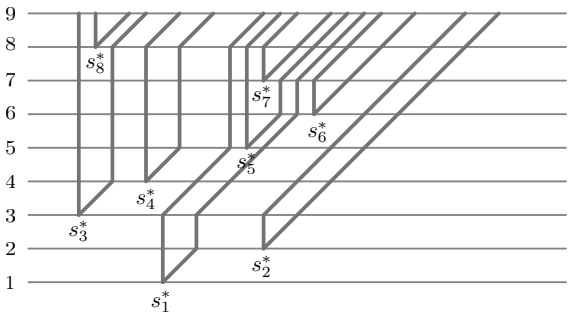

Figure 1: The breakpoints of the function F_i. For each i, s_{i-1}^* and $s_{i-1}^* + \gamma$ are the "new" breakpoints of F_i. All other breakpoints of F_i come from F_{i-1} where those smaller than s_i^* remain unchanged and those larger are increased by γ.

$[x-\gamma, x]$, or $s_{i-1}^* > x$, the value $y \in [x-\gamma, x]$ at which $\tilde{E}_i(y)$ is smallest is x, s_i^*, and $x-\gamma$, respectively, and therefore

$$\tilde{E}_i(x) = (x-t_i)^2 + \begin{cases} \tilde{E}_{i-1}(x-\gamma) & x > s_{i-1}^* + \gamma \\ \tilde{E}_{i-1}(s_{i-1}^*) & x \in [s_{i-1}^*, s_{i-1}^* + \gamma] \\ \tilde{E}_{i-1}(x) & x < s_{i-1}^*. \end{cases} \tag{2}$$

This shows that $\tilde{E}_i$ is continuous and piecewise quadratic. ■

We call a value x that determines the boundary of two neighboring quadratic pieces of $\tilde{E}_i$ a *breakpoint* of $\tilde{E}_i$. Since $\tilde{E}_1$ is a simple (one-piece) quadratic function, it has no break points. For $i > 1$, the breakpoints of the function $\tilde{E}_i$ consist of s_{i-1}^* and $s_{i-1}^* + \gamma$, as determined by recurrence (2), together with breakpoints that come from recursive applications of $\tilde{E}_{i-1}$. Examining equation (2) reveals that all breakpoints of $\tilde{E}_{i-1}$ that are smaller than s_{i-1}^* remain breakpoints in $\tilde{E}_i$ and all those larger than s_{i-1}^* are increased by γ and these form all of the breakpoints of $\tilde{E}_i$ (see Figure 1). Thus $\tilde{E}_i$ has precisely $2i-2$ breakpoints. To compute minimum of $\tilde{E}_i(x)$, i.e. to compute s_i^*, it is enough to scan over these $O(i)$ quadratic pieces and find the unique piece whose minimum lies between its two ending breakpoints.

The relationship between $\tilde{E}_i$ and E_i is as follows: for each $i = 1, \ldots, n$, let $\boldsymbol{s}_i = \operatorname{argmin}_{\boldsymbol{x} \in \Gamma_i} E_i(\boldsymbol{x})$, i.e. $\boldsymbol{s}_i$ is the γ-LIR of $\boldsymbol{t}_i = (t_1, \ldots, t_i)$. From the definitions we get

$$E_i(\boldsymbol{s}_i) = \min_{x \in \mathbb{R}} \tilde{E}_i(x).$$

The uniqueness of the $\boldsymbol{s}_i$ and s_i^* implies that s_i^* is the ith component (last element) of $\boldsymbol{s}_i$. The following lemma shows that determining s_i^*'s is essentially all we need.

Lemma 2.2 *If the sequence $\boldsymbol{s}^* = (s_1^*, \ldots, s_n^*)$ is known, one can compute the sequence $\boldsymbol{s}$ in linear time.*

Lemma 2.2 shows that one can compute the γ-LIR of an input sequence $\boldsymbol{t} = (t_1, \ldots, t_n)$ in linear time if the values $s_1^*, \ldots, s_n^*$ are known. One can compute these values in n iterations by computing in the ith iteration the value s_i^* at which $\tilde{E}_i$ is minimized and use it to compute the function $\tilde{E}_{i+1}$ as given in (2). This results an $O(n^2)$ algorithm for computing the γ-LIR of $\boldsymbol{t}$.

2.1 Computing s_i^* in $O(\log i)$ time

For the sake of simplicity, we assume for the rest of this paper that for each $2 \le i \le n$, s_i^*, i.e. the point minimizing $\tilde{E}_i$ is none of its breakpoints, although it is not hard to relax this assumption algorithmically.

Any sequence for which this assumption fails is called *degenerate*. It is easy to see that the set of degenerate sequences in $\mathbb{R}^n$ have Lesbegue measure zero. Under this assumption, s_i^* belongs to the interior of some interval on which $\tilde{E}_i$ is quadratic. The derivative of this quadratic function is therefore zero at s_i^*. In other words, if we know to which quadratic piece of $\tilde{E}_i$ the point s_i^* belongs, we can determine s_i^* by setting the derivative of that piece to zero.

Lemma 2.3 *The derivative of $\tilde{E}_i$ is a continuous monotonically increasing piecewise linear function.*

Let $F_i(x)$ denote the derivative of $\tilde{E}_i$. Using (2), one the following recurrence can be written for F_i:

$$F_i(x) = 2(x - t_i) + \begin{cases} F_{i-1}(x - \gamma) & x > s_{i-1}^* + \gamma \\ 0 & x \in [s_{i-1}^*, s_{i-1}^* + \gamma] \\ F_{i-1}(x) & x < s_{i-1}^* \end{cases} \tag{3}$$

if we set $F_0 = 0$. As mentioned above, s_i^* is simply the solution of $F_i(x) = 0$ which by Lemma 2.3 always exists and is unique.

In order to find s_i^* efficiently, we use an affine composition tree to represent F_i. Solving $F_i(x) = 0$ then consists of performing a search in the tree. Once s_i^* is computed, the tree representing F_i is updated using a THRESHOLDTRANSFORM operation to reflect F_{i+1}. All this is done in $O(\log i)$ time. As a result, the γ-Lipschitz isotonic regression can be computed in $O(n \log n)$ time.

Let T_i denote the affine composition tree that represents F_i. The set of pairs stored at T_i are $(x, F_i(x))$ where x is a breakpoint of F_i. By Lemma 2.3, the two components of these pairs are ordered consistently and therefore, one can search the tree using either of the two components. Also, by the same lemma, F_i is a piecewise linear continuous function. So, if we know the value of F_i at its the breakpoints, to find $F_i(x)$ for an arbitrary $x \in \mathbb{R}$, we first locate the two breakpoints immediate preceding and succeeding x, i.e. $x^- = \text{PREDECESSOR}_1(x)$ and $x^+ = \text{SUCESSOR}_1(x)$, and then linearly interpolate $F_i(x)$ between the $F_i(x^-)$ and $F_i(x^+)$. However, more care is needed if the query point x happens to be smaller than the smallest breakpoint or greater than the greatest of them. In these cases, it is easy to determine using (3) that the slope of F_i is exactly $2i$ which is enough to do the interpolation using only one endpoint.

Solving $F_i(x) = 0$ on T_i corresponds to searching T_i for the elements $(x^-, F_i(x^-))$ and $(x^+, F_i(x^+))$ where $F_i(x^-) = \text{PREDECESSOR}_2(0)$ and $F_i(x^+) = \text{SUCESSOR}_2(0)$. Thus, this can be done in $O(\log i)$ time; recall that number of breakpoints of F_i which is the same as the number of elements in T_i is $2i - 2$. Once these pairs are known, s_i^* is found through linear interpolation between x^- and x^+:

$$s_i^* = x^- + \left(\frac{-F_i(x^-)}{F_i(x^+) - F_i(x^-)} \right) (x^+ - x^-).$$

Once s_i^* is computed, we store it in a separate array for back-solving through Lemma 2.2 and proceed to update T_i to represent F_{i+1}. We call the resulting tree T_{i+1}. This is done by rewriting (3) for a breakpoint x_{i+1} of F_{i+1} that is neither of s_i^* or $s_i^* + \gamma$, using the fact that

$$x_{i+1} = \begin{cases} x_i + \gamma & x_i > s_i^* \\ x_i & x_i < s_i^* \end{cases}, \tag{4}$$

where x_i is the breakpoint of F_i that has become x_{i+1} in F_{i+1}. We get:

$$F_{i+1}(x_{i+1}) = \begin{cases} 2(x_i + \gamma - t_{i+1}) + F_i(x_i) & x_i > s_i^* \\ 2(x_i - t_{i+1}) + F_i(x_i) & x_i < s_i^* \end{cases} \tag{5}$$

We next combine and rewrite equations (4) and (5) as follows:

$$\begin{pmatrix} x_{i+1} \\ F_{i+1}(x_{i+1}) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ F_i(x_i) \end{pmatrix} + \begin{cases} \begin{pmatrix} \gamma \\ 2\gamma - 2t_{i+1} \end{pmatrix} & x_i > s_i^* \\ \begin{pmatrix} 0 \\ -2t_{i+1} \end{pmatrix} & x_i < s_i^* \end{cases}$$

This means that the modification needed on the tree is achieved by a THRESHOLDTRANSFORM$_1$ operation with parameters

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}, \tau = s_i^*, \boldsymbol{b}_1 = \begin{pmatrix} \gamma \\ 2\gamma - 2t_{i+1} \end{pmatrix}, \boldsymbol{b}_2 = \begin{pmatrix} 0 \\ -2t_{i+1} \end{pmatrix},$$

in $O(\log i)$ time. Finally, the new breakpoints, i.e. the pairs $(s_i^*, 2(s_i^* - t_{i+1}))$ and $(s_i^* + \gamma, 2(s_i^* + \gamma - t_{i+1}))$ are also inserted into T_{i+1}, again in $O(\log i)$ time. Now T_{i+1} represents F_{i+1} and can be used to compute s_{i+1}^*. We summarize the result of the above argument in the following theorem.

Theorem 2.4 *Given a sequence $\boldsymbol{t} = (t_1, \ldots, t_n)$ of n real numbers and a Lipschitz constant γ, one can find in $O(n \log n)$ time, the (optimal) γ-Lipschitz isotonic regression of $\boldsymbol{t}$.*

3 LIR on Rooted Trees

We now generalize the algorithm for computing Lipschitz isotonic regressions on paths to trees. Consider a rooted tree $\mathcal{T}$ with vertices numbered $1, \ldots, n$, with n representing the root of $\mathcal{T}$. To help with our notations, we write $j \preceq i$ to indicate that j is a descendent of i, i.e. $j = i$ or $j \prec i_1 \cdots \prec i_k \prec i$ for some $i_1, \ldots, i_k$. To simplify our notation we assume, without loss of generality, that $j \prec i$ implies $j < i$. We also use the notation $\mathcal{T}_i$ for the set of vertices in the subtree of $\mathcal{T}$ rooted at i, i.e. $\mathcal{T}_i = \{j : j \preceq i\}$; thus $\mathcal{T} = \mathcal{T}_n$.

Similar to the case where $\mathcal{T}$ is a simple path, we can define for each $i = 1, \ldots, n$ an energy function $E_i : \mathbb{R}^{|\mathcal{T}_i|} \to \mathbb{R}$ as $E_i(\boldsymbol{x}) = \sum_{j \preceq i} (x_j - t_j)^2$, and define $\tilde{E}_i : \mathbb{R} \to \mathbb{R}$ as

$$\tilde{E}_i(x) = \min_{\boldsymbol{x} \in \Gamma_i, x_i = x} E(\boldsymbol{x}).$$

Here, by some abuse of notation, we write x_i to represent the component of sequence $\boldsymbol{x} \in \mathbb{R}^{|\mathcal{T}_i|}$ that is assigned to vertex i. Also, Γ_i represents the convex subset of $\mathbb{R}^{|\mathcal{T}_i|}$ that satisfies the Lipschitz and isotonicity constraints that involve the vertices of $\mathcal{T}_i$. One can therefore write an equation corresponding to (1) in the case of paths, for a vertex i in the tree with children $i_1, \ldots, i_k$:

$$\tilde{E}_i(x) = (x - t_i)^2 + \sum_{j=1}^{k} \min_{x - \gamma \le y_j \le x} \tilde{E}_{i_j}(y_j). \tag{6}$$

An argument similar to those of Lemmas 2.1 and 2.3 leads to the following statement.

Lemma 3.1 *The function $\tilde{E}_i$ is continuous, strictly convex, and piecewise quadratic and its derivative is continuous, monotonically increasing, and piecewise linear.*

Let F_i denote the derivative of function $\tilde{E}_i$. We can prove that F_i satisfies the following recurrence:

$$F_i(x) = 2(x - t_i) + \sum_{i=1}^{k} F'_{i_j}(x), \tag{7}$$

where for each $j = 1, \ldots, k$,

$$F'_{i_j}(x) = \begin{cases} F_{i_j}(x - \gamma) & x > s^*_{i_j} + \gamma \\ 0 & x \in [s^*_{i_j}, s^*_{i_j} + \gamma] \\ F_{i_j}(x) & x < s^*_{i_j} \end{cases} \tag{8}$$

and here $s^*_{i_j}$ is the minimizer of the function E_{i_j}.

Thus to solve the LIR problem on a tree, we move from the leaves toward the root and when processing a node i, we compute and add up the linear functions F'_{i_j} from its children $i_1, \ldots, i_k$ and use the result in (7) to compute the function F_i. We then solve $F_i(x) = 0$ to find s^*_i. Note that by our assumption that $j \prec i$ implies $j \leq i$, we can process the vertices of $\mathcal{T}$ in the order of their associated ordinal number. As in the case of a path, F_i can be represented by a pair binary search tree where we keep a node $(x, F_i(x))$ for each breakpoint x of F_i. If a node i has only one child $j \prec i$, then finding the solution s^*_j of $F_j(x) = 0$ and the modification of T_j to represent T_i for F_i is precisely the same as the case of paths.

The only remaining difficulty is in computing T_i for F_i, given the functions $F_{i_j}, j = 1, \ldots, k$, where $i_1, \ldots, i_k$ are the children of i. For simplicity, we first consider the case where the node i has two children i_1 and i_2. We first convert the trees T_{i_1} and T_{i_2} to respectively represent the functions F'_{i_1} and F'_{i_2}. This is similar to, and indeed simpler than, what we did in the case of paths (Section 2), and can be handled by performing one THRESHOLDTRANSFORM operation on each of T_{i_1} and T_{i_2}. We then compute the tree T_i representing the function $F'_{i_1} + F'_{i_2}$ as follows: assume without loss of generality that F'_{i_2} is the function among F'_{i_1} and F'_{i_2} that has fewer breakpoints. We first compute $F'_{i_1}(x)$ for every breakpoint x of F'_{i_2} through interpolation in T_{i_1}. We then insert, for each breakpoint x of F'_{i_2}, the pair $(x, F'_{i_1}(x))$ into T_{i_1}. At this point, the function T_{i_1} represents is still F'_{i_1} but it includes all the breakpoints of F'_{i_2} as breakpoints as well. Finally, for every consecutive pair of breakpoints x^- and x^+ of F'_{i_2} we find the affine transformation that linearly interpolates between $F'_{i_2}(x^-)$ and $F'_{i_2}(x^+)$, i.e. we find a linear function $\psi(x) = ax + b$ satisfying $\psi(x^-) = F'_{i_2}(x^-)$ and $\psi(x^+) = F'_{i_2}(x^+)$. We then apply an INTERVALTRANSFORM$_1(\psi, x^-, x^+)$ on T_{i_1}. The result of this set of operations maintains the integrity of T_{i_1} as a pair search tree.

Finally, to change the tree T_i which currently represents $F'_{i_1} + F'_{i_2}$ to represent F_i, we apply the affine transformation

$$\phi : \boldsymbol{x} \mapsto \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \boldsymbol{x} + \begin{pmatrix} 0 \\ -2t_i \end{pmatrix}$$

to the entire tree T_i. We can now solve $F_i(x) = 0$ as in Section 2 using T_i.

Using the fact that F_{i_1} has fewer breakpoints than F_{i_2} and the number of affine-composition-tree operations is proprtional to the number of these breakpoints, one can argue that the total number of operations performed on the data structures is $O(n \log n)$. Since each operation takes $O(\log n)$ time, the overall running time is $O(n \log^2 n)$. The algorithm can be extended to trees with higher degree nodes without affecting its asymptotic running time. Putting everything together, we obtain the following:

Theorem 3.2 *Given a tree $\mathcal{T}$ and a value assignment $\boldsymbol{t} = (t_1, \ldots, t_n)$ to the vertices of $\mathcal{T}$ together with a Lipschitz constant γ, one can find in $O(n \log^2 n)$ time, the (optimal) γ-Lipschitz isotonic regression of $\boldsymbol{t}$ on $\mathcal{T}$.*

4 LUR on Paths and Trees

Given a point $\boldsymbol{t} = (t_1, \ldots, t_n) \in \mathbb{R}^n$ and a real value γ, the *Lipschitz unimodal regression* (γ-LUR) of $\boldsymbol{t}$ is the sequence $\boldsymbol{s} = (s_1, \ldots, s_n) \in \mathbb{R}^n$ that minimizes $E(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{t}\|^2$ under the following constraint: there is an $1 \leq i \leq n$ such that

(U1) for all $1 < j \leq i$, $x_{j-1} \leq x_j \leq x_{j-1} + \gamma$, and

(U2) for all $i \leq k < n$, $x_{k+1} \leq x_k \leq x_{k+1} + \gamma$.

For each i, let $\Gamma_i \subseteq \mathbb{R}^n$ be the set of points $\boldsymbol{x} \in \mathbb{R}^n$ that satisfy (U1) and (U2), let $\boldsymbol{\sigma}_i = \min_{\boldsymbol{x} \in \Gamma_i} E(\boldsymbol{x})$, and let $\xi_i = E(\boldsymbol{\sigma}_i)$. We first compute ξ_i for each i, then compute $\kappa = \arg\min_i \xi_i$, and we finally compute the sequence $\boldsymbol{\sigma}_\kappa$. For a fixed i, let $\Gamma_i^- \subseteq \mathbb{R}^i$ be the set of points $(x_1, \ldots, x_i)$ that satisfy (U1), and let $\Gamma_i^+ \subseteq \mathbb{R}^{n-i+1}$ be the set of points $(x_i, \ldots, x_n)$ that satisfy (U2). Define

$$E_i^-(x_1, \ldots, x_i) = \sum_{j=1}^{i} (x_j - t_j)^2, \qquad E_i^+(x_i, \ldots, x_n) = \sum_{k=i}^{n} (x_k - t_k)^2.$$

Then

$$\xi_i = \min_{x_1, \ldots, x_n} E_i^-(x_1, \ldots, x_i) + E_i^+(x_i, \ldots, x_n) - (x_i - t_i)^2 \tag{9}$$

where the minimum is taken over all $x_1, \ldots, x_n \in \mathbb{R}^n$ such that $x_1, \ldots, x_i \in \Gamma_i^-$ and $x_i, \ldots, x_n \in \Gamma_i^+$. The third term in the above equation is included because it exists in both E_i^- and E_i^+. As in Section 2, let

$$\tilde{E}_i^-(x) = \min_{\boldsymbol{y} \in \mathbb{R}^{i-1} : (\boldsymbol{y}, x) \in \Gamma_i^-} E_i^-(\boldsymbol{y}, x), \qquad \tilde{E}_i^+(x) = \min_{\boldsymbol{z} \in \mathbb{R}^{n-i} : (x, \boldsymbol{z}) \in \Gamma_i^+} E_i^+(x, \boldsymbol{z}), \tag{10}$$

$\tilde{E}_i^-$ and $\tilde{E}_i^+$ can be defined recursively as in (2). Set

$$\tilde{E}_i(x) = \tilde{E}_i^-(x) + \tilde{E}_i^+(x) - (x - t_i)^2. \tag{11}$$

If $s_i^* = \arg\min_x \tilde{E}_i(x)$, then $\xi_i = \tilde{E}_i(s_i^*)$. The same argument as in Lemma 2.1 implies that each of $\tilde{E}_i, \tilde{E}_i^-$, and $\tilde{E}_i^+$ is convex and piecewise quadratic. Therefore s_i^* is unique. Let F_i, F_i^-, and F_i^+ be the derivatives of $\tilde{E}_i, \tilde{E}_i^-$, and $\tilde{E}_i^+$, respectively. As in Lemma 2.3, each of the derivatives is a piecewise-linear monotonically increasing function. Furthermore, s_i^* is the value of x that satisfies the equation

$$F_i^-(x) + F_i^+(x) = 2(x - t_i). \tag{12}$$

If we represent F_i^- and F_i^+ using affine composition trees, denoted by T_i^- and T_i^+, then for any x, $F_i(x)$ can be computed in $O(\log n)$ time. Using the fact that F_i, F_i^-, F_i^+ are monotonically increasing, we can solve (12) in $O(\log^2 n)$ time, by doing a binary search on the breakpoints of F_i^- and F_i^+. We spend $O(\log n)$ time at each step to compute F_i at some breakpoint of F_i^- or F_i^+. Initially, T_1^- and T_1^+ can be constructed in $O(1)$ and $O(n \log n)$ time, respectively, and then T_i^- (resp. T_i^+) can be constructed from T_{i-1}^- (resp. T_{i-1}^+) in $O(\log n)$ time; T_i^- can be updated as in Section 2. Updating T_i^+ requires removing the breakpoints corresponding to i, which must be adjacent, and then performing a THRESHOLDTRANSFORM which undoes (is the inverse of) the ThresholdTransform performed when i was added to T_i^+.

Hence, we can compute all $\xi_1, \ldots, \xi_n$ in a total of $O(n \log^2 n)$ time. After having computed the value of κ, the sequence $\boldsymbol{\sigma}_\kappa$ can be computed in additional $O(n \log n)$ time. Putting everything together, we obtain the following:

Theorem 4.1 *Given a point $\boldsymbol{s} \in \mathbb{R}^n$ and a real value γ, the γ-LUR problem can be solved in $O(n \log^2 n)$ time.*

Our approach can be extended to solve the γ-LUR problem on a tree as well. The algorithm is described in the appendix, but we claim the following result.

Theorem 4.2 *Given an unrooted tree $\mathcal{T}$ with n vertices and a real value γ, the γ-LUR problem on $\mathcal{T}$ can be solved in $O(n^{3/2} \log^2 n)$ time.*

Remark. We believe the running time of the algorithm can be improved to $O(n \operatorname{polylog}(n))$.

5 Data Structure: Affine Composition Tree

Consider a binary tree T in which to each node is a pair $\boldsymbol{v} = (v_1, v_2) \in \mathbb{R}^2$. We say that the tree T is a *pair-search-tree* if T is simultaneously a search tree on first and second component of all nodes, i.e. for each node $\boldsymbol{v}$ with left child $\boldsymbol{v}^l$ and right child $\boldsymbol{v}^r$, $\boldsymbol{v}^l \leq \boldsymbol{v} \leq \boldsymbol{v}^r$, i.e. $v_1^l \leq v_1 \leq v_1^r$ and $v_2^l \leq v_2 \leq v_2^r$. We also write T_1 (resp. T_2) to refer to the ordinary binary search tree that is represented by the first (resp. second) component of every node of T.

An *affine transformation* is a map $\phi : \mathbb{R}^2 \to \mathbb{R}^2$, $\boldsymbol{x} \mapsto A \cdot \boldsymbol{x} + \boldsymbol{b}$ where A is a 2×2 matrix, (a *linear transformation*) and $b \in \mathbb{R}^2$ is a *translation vector* — in our notation we treat $\boldsymbol{x} \in \mathbb{R}^2$ as a column vector. An *affine composition tree* T is a special kind of pair-search tree in which the pair associated to a node is not given explicitly. Instead, each node $\boldsymbol{v}$ stores an affine transformation $\phi_{\boldsymbol{v}} : \boldsymbol{x} \mapsto A_{\boldsymbol{v}} \cdot \boldsymbol{x} + \boldsymbol{b}_{\boldsymbol{v}}$. The real pair implicitly associated to a node $\boldsymbol{v}$ is determined by the composition of the affine transformations associated to the nodes on the path from the root $\boldsymbol{r}$ of T to $\boldsymbol{v}$. Specifically, if $\boldsymbol{r} = \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k = \boldsymbol{v}$ is the path in T, then the pair associated to $\boldsymbol{v}$ is $\Phi_{\boldsymbol{v}}(\mathbf{0})$ where $\mathbf{0} = (0, 0)$ and

$$\Phi_{\boldsymbol{v}}(\boldsymbol{x}) = \Phi_{\boldsymbol{v}_{k-1}}(\phi_{\boldsymbol{v}}(\boldsymbol{x})) = \phi_{\boldsymbol{v}_1}(\phi_{\boldsymbol{v}_2}(\ldots \phi_{\boldsymbol{v}_k}(\boldsymbol{x}) \ldots)),$$

and $\Phi_{\boldsymbol{r}} = \phi_{\boldsymbol{r}}$. Notice that $\Phi_{\boldsymbol{v}}$ is also an affine transformation.

Given an affine composition tree, searching can be performed on either of T_1 or T_2. All we need to perform a search is to be able to compute $\Phi_{\boldsymbol{v}_i}(\mathbf{0})$ at a node $\boldsymbol{v}_i$ along the search path starting at the root of the tree. This is easily done inductively by composing the affine transformation $\Phi_{\boldsymbol{v}_{i-1}}$ computed for the parent $\boldsymbol{v}_{i-1}$ of the node $\boldsymbol{v}_i$ with $\phi_{\boldsymbol{v}_i}$ and evaluating the result at $\mathbf{0}$. Note that after evaluating the pair $\Phi_{\boldsymbol{v}}(\mathbf{0})$ that is implicitly stored at a node $\boldsymbol{v}$, the search can be followed based on either of the two components, or in other words, in either of the two trees T_1 or T_2. In particular, one can define operations $\textsc{Predecessor}_p$ and $\textsc{Sucessor}_p$, where $p \in \{1, 2\}$ that respectively return, given a real argument x, the *predecessor* and the *successor* of x in T_p, in $O(\log n)$ time. Here by the predecessor and successor or x in T_p we respectively mean the greatest element no larger than x and the smallest element no less than x stored in T_p.

We briefly argue here that all rotation-based methods for maintaining ordinary balanced binary search trees can be modified to make an affine composition tree that performs all of the usual operations (INSERT, DELETE, and of course searching) in $O(\log n)$ time when it stores n nodes. The key observations is that a rotation can be performed in $O(\log n)$ time; see Figure 2. one can perform rotations on any parent-child pair in constant time; one only needs to modify the stored affine transformations in a constant number of nodes (see Figure 2) and use the fact that an affine transformation $\phi : \boldsymbol{x} \mapsto A \cdot \boldsymbol{x} + \boldsymbol{b}$ has an inverse transformation $\phi^{-1} : \boldsymbol{x} \mapsto A^{-1} \cdot (x - \boldsymbol{b})$, which is itself an affine transformation, so long as its defining linear transformation A is invertible. Inserting a pair $\boldsymbol{v} \in \mathbb{R}^2$ into the tree is done by first computing the affine transformation $\Phi_{\boldsymbol{u}}$ for the node $\boldsymbol{u}$ that will be the parent of the node $\boldsymbol{v}$ that is to be inserted. To determine $\phi_{\boldsymbol{v}}$ we solve the equation $\Phi_{\boldsymbol{u}}(\phi_{\boldsymbol{v}}(\mathbf{0})) = \boldsymbol{v}$. This is a system of linear equations and by solving it we compute the translation vector $\boldsymbol{b}_{\boldsymbol{v}}$. The linear transformation A_v can be chosen to be an arbitrary invertible linear transformation. For simplicity, at the time of insertion, we always pick the identity transformation for $A_{\boldsymbol{v}}$. Deletion can be handled similarly.

What makes affine composition trees interesting to us is their ability to alter the values of all elements in a subtree under an affine transformation by composing that transformation with the affine transformation stored at the root of that subtree. Specifically, we define an operation $\textsc{IntervalTransform}_p(\psi, \tau^-, \tau^+)$ which in $O(\log n)$ time applies an affine transformation ψ to the subset of tree elements whose p-th component (where $p \in \{1, 2\}$) falls within the given range $[\tau^-, \tau^+]$. To do this, we first find the nodes $\boldsymbol{v}^- = \textsc{Predecessor}_p(\tau^-)$ and $\boldsymbol{v}^+ = \textsc{Sucessor}_p(\tau^+)$. We then successively rotate $\boldsymbol{v}^-$ with its parent until it becomes the root of the tree. Next, we do the same with $\boldsymbol{v}^+$ but stop when it becomes the right child of $\boldsymbol{v}^-$. At this stage, the subtree rooted at the left child of $\boldsymbol{v}^+$ contains exactly all the nodes $\boldsymbol{x}$ for

which $x_p \in [\tau^-, \tau^+]$. Thus we compose ψ with the affine transformation at that node and issue the performed rotations in the reverse order to put the tree back in its original (balanced) position. Since $\boldsymbol{v}^-$ and $\boldsymbol{v}^+$ were both within $O(\log n)$ steps from the root of the tree, and since performing each rotation on the tree can only increase the depth of a node by one, $\boldsymbol{v}^-$ is taken to the root in $O(\log n)$ steps and this increases the depth of $\boldsymbol{v}^+$ by at most $O(\log n)$. Thus the hole operation takes $O(\log n)$ time.

It must be noted that performing an INTERVALTRANSFORM with an inappropriate transformation over an interval may jeopardize the integrity of a pair tree since the transformed set of values may simply fail to conform to the search tree structure. A "safe" derivative of INTERVALTRANSFORM is an operation called THRESHOLDTRANSFORM$_p(A, \tau, \boldsymbol{b}_1, \boldsymbol{b}_2)$ which takes as argument a linear transformation A with all non-negative entries, a *threshold* parameter $\tau \in \mathbb{R}$, and two vectors $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ satisfying $\boldsymbol{b}_1 \geq \boldsymbol{b}_2$. As the result of this operation, every element $\boldsymbol{x}$ in the tree for which $x_p \geq \tau$ is transformed by the affine transformation $\psi_1 : \boldsymbol{x} \mapsto A \cdot \boldsymbol{x} + \boldsymbol{b}_1$ and every other element x is transformed by the affine transformation $\psi_2 : \boldsymbol{x} \mapsto A \cdot \boldsymbol{x} + \boldsymbol{b}_2$. It is straightforward to show that THRESHOLDTRANSFORM preserves the order of elements in the tree. This is firstly because the linear transformation A preserves this order since its entries are all non-negative, and secondly because elements $\boldsymbol{x} \geq \boldsymbol{\tau}$ are translated by vector $\boldsymbol{b}_1$ which is no less than $\boldsymbol{b}_2$ under which the rest of the elements are translated. The operation can be implemented similar to INTERVALTRANSFORM: first the element $\boldsymbol{v}^- = \text{PREDECESSOR}_p(\tau)$ is found in and moved to the root of the tree using rotations in $O(\log n)$ steps. At this point the root and the elements in its left subtree should be transformed under ψ_2 and elements in the root's right child must be transformed under ψ_1. We thus apply ψ_1 at the root and $\psi_1^{-1} \circ \psi_2$ to the right child. Finally, the rotations are performed in reverse order to restore the original balance.

6 Conclusion

In this paper we defined the Lipschitz isotonic regression problems on DAGs and provided efficient algorithms for solving it on the special cases where the DAG in question is a path or a rooted tree. We also provided efficient solutions to the Lipschitz unimodal regression problem for undirected paths and trees. Our affine composition tree data structure has proven to be a powerful tool in our approach.

The most important open problem is solving LIR problem on general DAGs. The known algorithm for solving IR on DAGs runs in $O(n^4)$ and does not lend itself to the Lipschitz generalization. The difficulty here is to compute the function $\tilde{E}_i$ (or in fact F_i) when the ith vertex is processed and comes from the fact that the F_{i_j} function, $j = 1, \ldots, k$, where $i_1, \ldots, i_k$ are vertices with incoming edges to i, are not independent. Independently, it is an important question whether IR can be solved more efficiently. In particular, the special case of the problem where the given DAG is planar has important applications in terrain simplification.

References

[1] S. Angelov, B. Harb, S. Kannan, and L.-S. Wang. Weighted isotonic regression under the l_1 norm. In *Proc. 17th Annu. ACM-SIAM Sympos. on Discrete Algs.*, pages 783–791, 2006.

[2] M. Ayer, H. D. Brunk, G. M. Ewing, W. T. Reid, and E. Silverman. An empirical distribution function for sampling with incomplete information. *Annals of Mathematical Statistics*, 26:641–647, 1955.

[3] R. E. Barlow, D. J. Bartholomew, J. M. Bremmer, and H. D. Brunk. *Statistical Inference Under Order Restrictions: The Theory and Application of Isotonic Regression*. Wiley Series in Probability and Mathematical Statistics. John Wiley and Sons, 1972.

[4] H. D. Brunk. Maximum likelihood estimates of monotone parameters. *Annals of Mathematical Statistics*, 26:607–616, 1955.

[5] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in $x + y$ and matrices with sorted columns. *J. Comput. Syst. Sci.*, 24(2):192–208, 1982.

[6] S. J. Grotzinger and C. Witzgall. Projection onto order simplexes. *Applications of Mathematics and Optimization*, 12:247–270, 1984.

[7] W. L. Maxwell and J. A. Muckstadt. Establishing consistent and realistic reorder intervals in production-distribution systems. *Operations Research*, 33:1316–1341, 1985.

[8] P. M. Pardalos and G. Xue. Algorithms for a class of isotonic regression problems. *Algorithmica*, 23:211–222, 1999.

[9] T. Robertson, F. T. Wright, and R. L. Dykstra. *Order Restricted Statistical Inference*. Wiley Series in Probability and Mathematical Statistics. John Wiley and Sons, 1988.

[10] P. Soille, J. Vogt, and R. Colombo. Carving and adaptive drainage enforcement of grid digital elevation models. *Water Resources Research*, 39(12):1366–1375, 2003.

[11] Q. F. Stout. Optimal algorithms for unimodal regression. *Computing Science and Statistics*, 32:348–355, 2000.

[12] W. A. Thompson, Jr. The problem of negative estimates of variance components. *Annals of Mathematical Statistics*, 33:273–289, 1962.

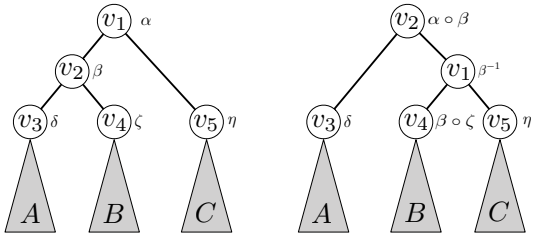

Figure 2: Rotation in affine composition trees.

Appendix

A Some Missing Proofs

Proof of Lemma 2.2. As discussed earlier, $s_n^* = s_n$. We generate the sequence $\boldsymbol{s}$ backwards. Suppose that the numbers $s_{i+1}, \ldots, s_n$ are determined, for some $1 \leq i \leq n-1$, and we wish to determine s_i. Picking $s_i = x$ entails that the energy of the solution will be at least

$$\tilde{E}_i(x) + \sum_{j=i+1}^{n} (s_j - t_j)^2.$$

Thus s_i is to be chosen to minimize $\tilde{E}_i$ on the interval $[s_{i+1} - \gamma, s_{i+1}]$. Observe that if $s_i^* \in [s_{i+1} - \gamma, s_{i+1}]$, then one can pick $s_i = s_i^*$ which is the global minimum for $\tilde{E}_i$. Otherwise, one must pick $s_i = s_{i+1} - \gamma$ or $s_i = s_{i+1}$ depending on whether $s_i^* < s_{i+1} - \gamma$ or $s_i^* > s_{i+1}$, respectively. Thus each s_i can be found in $O(1)$ time and the entire process takes $O(n)$ time. □

Proof of Lemma 2.3. Since by Lemma 2.1, $\tilde{E}_i$ is strictly convex and piecewise quadratic, its derivative is monotonically increasing. To prove the continuity of the derivative, it is enough to verify this at the "new" breakpoints of $\tilde{E}_i$, i.e. at s_i^* and $s_i^* + \gamma$. The continuity of the derivative can be argued inductively by observing that both mappings of the breakpoints from $\tilde{E}_{i-1}$ to $\tilde{E}_i$ (identity or shifting by γ) preserve the continuity of the function and its derivative.

Notice that at s_{i-1}^*, both the left and right derivatives of a quadratic piece of $\tilde{E}_{i-1}$ becomes zero. From the definition of $\tilde{E}_i$ in (2), the left derivative of the function $\tilde{E}_i - (x - t_{i+1})^2$ at s_{i-1}^* agrees with the left derivative of $\tilde{E}_{i-1}$ at the same point and is therefore zero. On the other hand, the right derivative of the function $\tilde{E}_i - (x - t_{i+1})^2$ is zero at $s_{i-1}^* + \gamma$ and agrees with its right derivative at the same point as the function is constant on the interval $[s_{i-1}^*, s_{i-1}^* + \gamma]$. This means that the derivative of $\tilde{E}_i - (x - t_{i+1})^2$ is continuous at s_{i-1}^* and therefore the same holds for $\tilde{E}_i$. A similar argument establishes the continuity of $\tilde{E}_i$ at $s_{i-1}^* + \gamma$. □

B LUR on Trees

For a set of vertices $\boldsymbol{t} = (t_1, \ldots, t_n) \in \mathbb{R}^n$ which are connected to form a tree $\mathcal{T}$, we say the tree is *ordered with respect to a root* r if r is made the root of the tree and edges are directed from a node to its parent. Solving the γ-LIR problem on $\mathcal{T}$ and $\boldsymbol{t}$ places γ-Lipschitz and isotonic constraints on all directed edges. For a tree ordered with respect to vertex r, let $\xi_r = \min_{\boldsymbol{x} \in \Gamma_i} = ||\boldsymbol{t} - \boldsymbol{x}||^2$, where $\boldsymbol{x}$ is the solution to the γ-LIR

problem. The γ-Lipschitz unimodal regression for a point $\boldsymbol{t}$ and a tree $\mathcal{T}$ is the solution to the γ-LIR problem for the root r which minimizes ξ_r.

To solve the γ-LUR problem we first choose an arbitrary vertex r as the root and solve the γ-LIR problem. We then traverse $\mathcal{T}$, alternating which vertex is the root, and for each recomputing the cost of the solution to the γ-LIR problem. After discovering which root r minimizes ξ_r, we compute the solution to the γ-LIR problem for $\mathcal{T}$ rooted at r.

We first need an additional tool. Let $\mathcal{R} = \{T^1, \ldots, T^k\}$ be a *tree set* of affine combination trees. A tree set can maintain a piecewise-linear function F with n breakpoints as does a single tree so $F(x) = \sum_{j=1}^{k} F^j(x)$. We can search for $F(x) = 0$ by performing a binary search on the k trees in parallel using a technique by Frederickson and Johnson [5] in $O(k \log k \log n)$ time. To perform an update by including another vertex from a sequence as in equation (3) we can update a function F_i to F_{i+1} in $O(k \log^2 n)$ time. We choose one tree, T^1, as active and insert the new break points and perform a THRESHOLDTRANSFORM$_1$ operation exactly as in Section 2. On the other trees we want to update the breakpoints, but not add the $2(x - t_i)$ term. The THRESHOLDTRANSFORM$_1$ parameters can be easily modified. Thus this operation takes $O(k \log n)$ time to update all k trees.

To solve the γ-LUR problem, we always maintain a root r and a tree set R_i for each child i of r. We assume that each node in $\mathcal{T}$ has degree at most 3; higher degree nodes can be handled as in Section 3. We need two operations. First, compute $\xi_r = \min_{\boldsymbol{x} \in \Gamma_r}$, where Γ_r is the feasible region corresponding to the constraints imposed by the γ-LIR problem on the $\mathcal{T}$ using r as the root. Second, for a child vertex u of r, make u the new root of the tree and maintain our data structures. This involves for the tree set R_u splitting it so the children u_1, u_2 of u described by this tree each has their own tree set R_{u_1}, R_{u_2}, and for r_1, r_2, the other children of r that are not u, we need to create a single tree set R_r.

We always ensure that no tree in any tree set has more than $O(\sqrt{n})$ breakpoints. Thus no tree set will ever have more than $O(\sqrt{n})$ trees. Computing ξ_r can be done by searching for $F_u(x) + F_{r_1}(x) + F_{r_2}(x) = x - t_r$, where each tree set R_i represents a function F_i. This can be done in $O(\sqrt{n} \log^2 n)$ time, by treating all tree sets from the children as a single tree set.

Transitioning from r as the root to u as the root requires two data structure changes. The first change requires merging 2 tree sets. If there is more than one tree with less than $O(\sqrt{n})$ breakpoints, we merge as in Section 3, otherwise we just put all trees into one big tree set. The second requires splitting a tree set R_u into 2 tree sets R_{u_1} and R_{u_2}. If a tree set from R_u only belongs to a single child, then we just transfer it. There will be at most one tree from R_u remaining to be split. We store the two trees from R_{u_1} and R_{u_2} that combine to describe this tree on the edges from u to u_1 and u_2, respectively.

Thus to move from one vertex to a neighbor takes $O(\sqrt{n} \log n)$ time, and to evaluate s_u^* on the new neighbor takes $O(\sqrt{n} \log^2 n)$ time.

Theorem B.1 *Given a tree $\mathcal{T}$ with n nodes assigned values $\boldsymbol{t}$, we can solve the γ-LIR problem in $O(n^{3/2} \log^2 n)$ time.*