A FINITE HYPERPLANE TRAVERSAL ALGORITHM FOR 1-DIMENSIONAL L^1pTV MINIMIZATION, FOR $0 < p \leq 1$

HEATHER A. MOON AND THOMAS J. ASAKI

Abstract. In this paper, we consider a discrete formulation of the one-dimensional L^1pTV functional and introduce a finite algorithm that finds exact minimizers of this functional for $0 < p \leq 1$. Our algorithm for the special case for L^1TV returns globally optimal solutions for all $\lambda \geq 0$ at the same computational cost of determining a single optimal solution associated with a particular value of lambda. This finite set of minimizers contains the scale signature of the known initial data. A variation on this algorithm returns locally optimal solutions for all $\lambda \geq 0$ for the case when $0 < p < 1$. The algorithm utilizes the geometric structure of the set of hyperplanes defined by the nonsmooth points of the L^1pTV functional. We discuss efficient implementations of the algorithm for both general and binary data.

1. Introduction

In this paper, we introduce an efficient finite hyperplane traversal (ht) algorithm for solving the 1D discrete L^1pTV problem, for all parameters $\lambda > 0$

$$\min_{u\in\mathbb{R}^{m+1}} \left\{ G_p(u) \equiv \sum_{i=0}^{m-1} |u_{i+1} - u_i|^p + \lambda \sum_{i=0}^{m} |f_i - u_i| \right\}, \tag{1}$$

where $0 < p \leq 1$, $f \in \mathbb{R}^{m+1}$ is some given data, with either fixed ($u_0 = f_0$ and $u_m = f_m$) or free boundary conditions. The ht algorithm requires only finitely many iterations to obtain a complete set of exact minimizers for (1) for all $\lambda > 0$. For $p = 1$, these minimizers are global minimizers, while for $p < 1$, the minimizers are local. Computationally effcient implementations of the ht algorithm are presented for both general and binary data.

The ht algorithm uses the geometric structure of the discrete L^1pTV function, G_p, in (1). Notice that G_p is nonsmooth on hyperplanes of the form $\{u|u_i = u_j, \text{ where } |i-j| = 1, 0 \leq i, j \leq m\}$ and $\{u|u_i = f_i, 0 \leq i \leq m\}$ and smooth everywhere else in $\mathbb{R}^{m+1}$. We show that minimizers of G_p are located at intersections of these hyperplanes. While the ht algorithm is an iterative improved-point algorithm, this geometric structure allows us to avoid computing descent directions and step sizes in the typical optimization sense. The algorithm iteratively reduces the parameter λ (in the spirit of parametric programming) determining at each step an optimal descent direction and performing a "line search" on a finite subset of points. Furthermore, it is never necessary to leave hyperplanes of the form $\{u|u_i = u_j, \text{ where } |i - j| = 1, 0 \leq i, j \leq m\}$ and $\{u|u_i = f_i, 0 \leq i \leq m\}$, so that the dimension of the problem is reduced after each iteration.

A complete set of minimizers for (1) can also be obtained by reformulation to a linear program and solved using a parametric simplex method (for example, see [8]). However, the ht algorithm is distinctly advantageous for several reasons. Significantly, a reformulation is only possible for the $p = 1$ case, whereas the $p < 1$ solutions are of importance as described later. A reformulation changes an unconstrained problem in m variables to a linear program with $5m$ variables and $2(m-1)$ constraints and $5(m-1)$ additional sign restrictions. Thus, it cannot take advantage of any dimension reduction strategies and requires a large amount of data storage and computation due to the large number of constraints. For the $p = 1$ case, the ht algorithm produces a sequence of minimizers that correspond to basic feasible solutions (or minimizing

Date: September 5, 2014.

linear combinations) of the reformulated problem. However, the ht algorithm does not rely on the high-dimensional geometry of the feasible set of the reformulated problem, working in a continually reduced dimensional space.

Graph-cut methods have also been successful in efficiently solving (1) for both one and two dimensional data [10, 9]. They can only address the $p = 1$ case. However, the λ-parametric problem can be solved by this method. The advantage of the hyperplane traversal algorithm is in simplicity of implementation and minimal data footprint.

Problem (1) is a discretization of

$$\min_{u\in\mathcal{B}_n(\Omega)} \int_\Omega |\nabla u|^p \, dx + \lambda \int_\Omega |f-u| \, dx, \quad \text{for } 0 < p \leq 1, \tag{2}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a given function, Ω is a bounded domain, and the minimization is taken over all functions of bounded variation, $u : \mathbb{R}^n \to \mathbb{R}$. Problems such as (2) are motivated by applications in signal and image analysis tasks such as denoising and finding scales in data. Variational and PDE based methods of denoising have been used for more than two decades ([14],[12],[2],and [3]). The most notable variational techniques solve the problem

$$u^* \in \arg\min_{u\in L^2(\Omega)} \mathcal{F}_{p,q}(u, \nabla u) \equiv \int_\Omega |\nabla u|^p + \lambda |f-u|^q dx. \tag{3}$$

Now, if λ is large, the term containing $\lambda|f-u|^q$ must be small, therefore making our minimizer u^* close to f in the L^q sense. But, if λ is small, $\|\nabla u\|_p$ will be small, that is, u will have smaller variation in the L^p sense. In the applications of denoising and finding scales, we note that u^* is less noisy than f (or has less of the smaller scales of f) and more flat when λ is small and u^* is more like f (more of the noise remains) when λ is large. We now briefly discuss a few of the results for particular values of p and q.

$\mathcal{F}_{2,2}(u, \nabla u)$ was first introduced by Tikhonov [14]. This functional is strictly convex. Using results from the calculus of variations (see [5, 6, 7]), we can say that there exists a unique minimizer. In image denoising, we find the minimizer of $\mathcal{F}_{2,2}(u, \nabla u)$, has smoothed edges around objects ($\mathcal{F}_{2,2}(u, \nabla u)$ is larger for functions with jump discontinuities than for those that will increase steadily and thus losing edge location). Pixel intensity is also lost. This problem then is not sufficient for images with regions of high contrast or well defined object edges.

Rudin, Osher, and Fatemi proposed [12] minimizing $\mathcal{F}_{1,2}(u, \nabla u)$, also called the ROF functional, to allow jump discontinuities in u^* which makes sense in many real images. $\mathcal{F}_{1,2}(u, \nabla u)$ is also strictly convex. In image denoising, we see that minimizers preserve the location of object edges, but still lose contrast (even when f is a noiseless image) and features with high curvature are lost [13]. That is, corners get rounded.

In [2], Chan and Esedoḡlu show that minimizing the L^1TV functional, $\mathcal{F}_{1,1}$, for imaging tasks will preserve pixel intensity. However, features of high curvatures are still lost. This functional is again convex, but this time it is not strictly convex. Therefore it should be noted that we cannot guarantee a unique minimizer for L^1TV. For a discussion about the discretized L^1TV see also [1] and [11].

In 2007, Chartrand [3] proposed to minimize $\mathcal{F}_{p,2}$ for $0 < p < 1$. It is worth noting that the functionals, $\mathcal{F}_{p,2}$, are not convex and therefore standard methods do not guarantee that we find a global minimizer. Despite this lack of guarantee, Chartrand found success in obtaining what appear to be local minimizers. For a cartoon image or an image with piecewise constant intensities, these solutions preserve object edges, pixel intensity, and areas of high curvature where sharp corners occur.

In Section 2 we present a variety of properties of (1) when $p = 1$ and provide helpful definitions and notation used throughout the paper. In Section 3 we provide a formal description of a preliminary ht algorithm including motivating concepts. In Section 4 we show that this algorithm provides a global minimizer (at fixed λ) for the case $p = 1$. In Section 5 we show that the iterative solution of the $\lambda = 0$ ($p = 1$) problem provides a complete set of global minimizers for all $\lambda \geq 0$, and formally describe an efficient ht algorithm. In Section 6 we show results of time trials. In Section 7 we consider an example of using ht for extracting

scale information from daily sunspot number data. In Section 8 we discuss the generalization of ht to the $p < 1$ case. Finally, in Section 9 we provide concluding remarks.

2. Properties of the discrete L^1TV function

In this section we present a variety of properties of (1) for $p = 1$:

$$\min_{u\in\mathbb{R}^{m+1}} \left\{ G_1(u) \equiv \sum_{i=0}^{m-1} |u_{i+1} - u_i| + \lambda \sum_{i=0}^{m} |f_i - u_i| \right\}. \tag{4}$$

We also provide helpful definitions and notation used throughout the paper. In particular, we consider the geometric properties of G_1 and consequences for finding solutions to (4).

First, we define the sets on which G is nonsmooth.

$$\mathcal{S}_u := \{u = (u_1, \ldots, u_m) | u_i = u_{i+1}, \text{ for some } i = 0, ..., m-1\} \tag{5}$$

and

$$\mathcal{S}_f := \{u = (u_1, u_2, \ldots, u_m) | u_i = f_i, \text{ for some } i = 0, ..., m\}. \tag{6}$$

Now we collect some properties that describe the minimizers of G_1.

Lemma 2.1. *G_1 has a minimizer.*

Proof. G_1 is bounded below, convex, and coercive in that as $|u| \to \infty$ $G_1 \to \infty$. Thus, by the Weierstrass optimality condition, G_1 has a minimizer. □

Definition 2.1. We define the set of global minimizers, $\mathcal{M}_\lambda$ for G_1, noting that this set depends on the value $\lambda > 0$,

$$\mathcal{M}_\lambda \equiv \arg\min_u G_1.$$

Lemma 2.2. *$\mathcal{M}_\lambda$ is bounded and convex.*

Proof. The boundedness and convexity of $\mathcal{M}_\lambda$ follow directly from the boundedness and convexity of G_1. □

Lemma 2.3. *If u is a local minimizer of G_1, then $u \in \mathcal{M}_\lambda$. That is, if u is a local minimizer, then it is also a global minimizer.*

Proof. This lemma follows directly from the convexity of G_1. □

Definition 2.2. Let G_1 be defined as in (4). Let Y be the set of points of intersections of at least $m + 1$ hyperplanes of the form $\{u_i = f_i\}$ and/or $\{u_i = u_{i+1}\}$.

Lemma 2.4. $|Y| \leq \frac{(2m+1)!}{(m+1)!m!} < \infty$.

Proof. First note that the number of hyperplanes of the form $\{u_i = u_{i+1}\}$ is m. The number of hyperplanes of the form $\{u_i = f_i\}$ is $m + 1$. Using the definition of Y, we see that the number of all the possible intersections of $m + 1$ such hyperplanes is $\begin{pmatrix} 2m+1 \\ m+1 \end{pmatrix}$. That is $|Y| \leq \begin{pmatrix} 2m+1 \\ m+1 \end{pmatrix} = \frac{(2m+1)!}{(m+1)!(m)!}$. (Here, the first is an inequality because we may have hyperplanes that are everywhere the same.) □

Lemma 2.5. *There is a minimizer, u^*, of G_1 in Y.*

Proof. Let $\hat{u}$ be a minimizer of G_1 and $\hat{u} \notin Y$. Suppose first that $\nabla G_1(\hat{u})$ exists. Then $\nabla G_1(\hat{u}) = 0$. But then because G_1 is affine at points where it is differentiable, G_1 is constant in the whole region containing $\hat{u}$ up to and including the bounding hyperplanes where G_1 is nonsmooth. By the coercivity condition, this region must be bounded. Therefore, there is a point, u^*, on the boundary of this region that is in Y such that $G_1(u^*) = G_1(\hat{u})$ where G_1 is given in (4).

Second, suppose that $\hat{u} \in \mathcal{H}$ where $\mathcal{H}$ is the intersection of $\ell < m$ hyperplanes where G_1 is nonsmooth, then we consider the function $\tilde{G}_1$ which is G_1 restricted to $\mathcal{H}$. Then $\nabla\tilde{G}_1(\hat{u})$ exists and is zero. We can then use the same argument above to get that the set of minimizers includes a point in Y. □

3. Preliminary ht Algorithm for discrete L^1TV

We now propose a preliminary ht algorithm for solving (4) for a single fixed value of λ. The convergence properties of this algorithm are examined in Section 4. In Section 5 we show that a finite λ-parametric iterative implementation can be used to discover globally optimal solutions for all $\lambda \geq 0$.

Consider a pedagogical example. Let $\lambda = 1, f = (0, 0.9, 0.4, 1)$ with fixed boundary conditions, $u_0 = f_0, u_3 = f_3$. The geometry of G_1 is illustrated in Figure 1. The level lines of G_1 appear as simple polygons with blue indicating lower value. There are seven hyperplanes (lines) where G_1 is nonsmooth and $|Y| = 11$. The preliminary algorithm is an improved iterate algorithm in which all iterates lie in $\mathcal{S}_u \bigcup \mathcal{S}_f$, and an optimal point lies in Y.

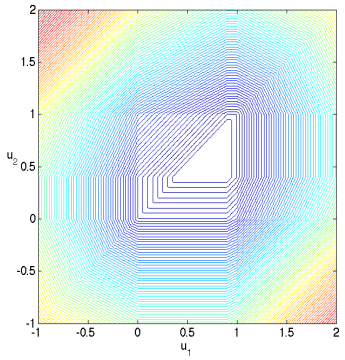

Figure 1. Level lines for the function $G_1(u_1, u_2) = |u_2 - u_1| + |u_1| + |1 - u_2|$, showing the affine nature of the discrete formulation for L^1TV. Here blue represents low values and red represents high values.

The preliminary algorithm works as follows. Start at the point $u = f$. Check positive and negative coordinate directions for descent in G_1. Sum all coordinate descent directions to get an α-descent direction (formal definition below). Perform a finite line search on the set of points where the search line intersects the hyperplane set $\mathcal{S}_u \bigcup \mathcal{S}_f$. If the algorithm steps to a point in $\mathcal{S}_f$, compute a new α-descent direction and repeat. Otherwise, (if the algorithm steps to a point in $\mathcal{S}_u$), project the current point onto $\mathbb{R}^{\ell_k}$, the space that is isomorphic to the intersections of the hyperplanes of the form $\{u_i^{(k)} = u_j^{(k)}\}$. Repeat until no coordinate descent exists.

The formal algorithm is given in Tables 1 and 2 and makes use of the following definitions. Let the coordinate directions, e_i be defined as usual, that is

$$e_i = (0, \ldots, 0, 1, 0, \ldots, 0), \text{ where the 1 is located in the } i^{\text{th}} \text{ position.}$$

Definition 3.1. We define an α-direction to be a vector v so that

$$v = \sum_i \alpha_i e_i,$$

where $\alpha_i \in \{-1, 0, 1\}$. We say that v is an α-descent direction for G_1 at the point u if v is an α-direction and $G_1(u + \tilde{\gamma}v) < G_1(u)$ for all $0 < \tilde{\gamma} < \gamma$, for some $\gamma > 0$.

4. Global Minimizers for a fixed λ

Using Lemmas 2.1, 2.4, and 2.5, we know that G_1 has a minimizer in the finite set Y. Below we show that Algorithm 3.1 finds a minimizer after finitely many iterations. We begin with the statement of this theorem.

Theorem 4.1. *Algorithm 3.1 converges to a minimum and is finite.*

Algorithm 3.1. (L^1TV)
Given $f = (f_1, \ldots, f_m)$;
Set $u^{(0)} = (u_1^{(0)}, \ldots, u_m^{(0)}) = (f_1, \ldots f_m)$;
Set $k \leftarrow 0$;
do
Compute $d^{(k)}$ using Algorithm 3.2
$\alpha_k \leftarrow \arg\min_\alpha \left\{ G_1(u^{(k)} + \alpha d^{(k)}) \middle| u^{(k)} + \alpha d^{(k)} \in \mathcal{S}_u \cup \mathcal{S}_f \right\}$;
$u^{(k+1)} \leftarrow u^{(k)} + \alpha_k d^{(k)}$;
$k \leftarrow k + 1$;
until $d^{(k)} = 0$

TABLE 1. Preliminary L^1TV algorithm

Algorithm 3.2. (Descent at iteration k)
Given $u_1, \ldots, u_m$;
$i = 1$;
Evaluate $G0 = G_1(u_1, \ldots, u_m)$;
Set $d \leftarrow 0$;
while $i \leq m$
$l_{\max} = \arg\max \{l \,|u_h = u_i,\ \forall i \leq h \leq i + l\}$;
$v = \sum_{l=i}^{i+l_{\max}} e_l$;
$G1 = G_1\,(u + (u_{i-1} - u_i)v)$;
$G2 = G_1\,(u + (u_{i+l_{\max}+1} - u_{i+l_{\max}})v)$;
if $G1 < G0$
$d \leftarrow d + \text{sign}(u_{i-1} - u_i)v$;
elseif $G2 < G0$
$d \leftarrow d + \text{sign}(u_{i+l_{\max}+1} - u_{i+l_{\max}})v$;
end
$i \leftarrow i + l_{\max} + 1$;
end

TABLE 2. α-descent algorithm

The proof of Theorem 4.1 is a consequence of the next four lemmas which we will prove in subsequent subsections.

Lemma 4.1. *Wherever descent exits, α-descent also exists.*

Lemma 4.2. *Whenever α-descent exists, the α-direction of Algorithm 3.1 also exists.*

Lemma 4.3. *α-directions found in Algorithm 3.1 give strict descent.*

Lemma 4.4. *Only finitely many steps of the algorithm are needed to get from one point in Y to another.*

4.1. **Proof of Lemma 4.1.** In this section we will show that if at u, G_1 has a descent direction, then G_1 also has an α-descent direction at u. We begin by borrowing from [4], the definition of the generalized gradient and generalized derivative which we use to discuss descent directions for G_1.

Definition 4.1. We define the generalized gradient of a locally Lipschitz function g at a point u to be

$$\partial g(x) = \text{co}\left\{ \lim_{i\to\infty} \nabla g(x_i) \middle| x_i \to x, \nabla g(x_i) \text{ exists} \right\}. \tag{7}$$

We define the generalized derivative, $g^\circ(u; v)$, of a function g at a point u in the direction v to be

$$g^\circ(u; v) \equiv \limsup_{y \to u, t \searrow 0} \frac{g(y + tv) - g(y)}{t}, \tag{8}$$

where $y \in \mathbb{R}^m$ and $t > 0$.

We also take from [4] the following proposition:

Proposition 4.1. *Let $g : X \to \mathbb{R}$ be Lipschitz near u. Then for every $v \in X$, we have*

$$g^\circ(x; v) = \max\{\langle \zeta, v \rangle | \zeta \in \partial g(x)\}. \tag{9}$$

Because in a neighborhood of each point $u \in \mathbb{R}^m$ there are only finitely many $\nabla g(y)$, Equation 9 reduces to

$$\partial g(u) = \operatorname{co}\{\nabla g(u_1), \ldots, \nabla g(u_\ell)\} = \left\{ \alpha_1 \nabla g(u_1) + \ldots + \alpha_\ell \nabla g(u_\ell) \,\middle|\, \alpha_i \geq 0, i = 1...\ell, \sum_i^\ell \alpha_i = 1 \right\}. \tag{10}$$

and

$$g^\circ(u; v) = \max_{(\alpha_1, \ldots, \alpha_\ell)} \left\{ \alpha_1 \nabla g(u_1) \cdot v + \ldots + \alpha_\ell \nabla g(u_\ell) \cdot v \,\middle|\, \alpha_i \geq 0, i = 1...\ell, \sum_i^\ell \alpha_i = 1 \right\}.$$

Let $K(u)$ be the cone of descent directions for G_1 at u. We now prove a more general statement about functions that are continuous and piecewise affine.

Lemma 4.5. *Let $g : \mathbb{R}^n \to \mathbb{R}$ be a continuous, piecewise affine (with finitely many pieces) function that is smooth on convex domains. If $g^\circ(u; v) < 0$ then v is a descent direction.*

Proof. Let u be a point so that $\nabla g(u)$ does not exist. This means that u is on a section of the boundary of ℓ domains where g is smooth. Because the domains where g is smooth are convex, we can choose points, $u_1, \ldots, u_\ell$, one in each of these domains, so that g is linear along the line segments connecting u and u_i and then using Definition 4.1 we have that if $g^\circ(u; v) < 0$ then $\nabla g(u_i) \cdot v < 0$ for $i = 1, ..., \ell$. Otherwise if for some j, $\nabla g(u_j) \cdot v \geq 0$ then we could choose $\alpha_i = 0$ for $i \neq j$ and $\alpha_j = 1$ and $g^\circ(u; v) \geq 0$. Let $t_0 > 0$ be sufficiently small so that g is linear along the line $u + tv$ for $0 < t \leq t_0$. Then, we know that

$$g(u_i) - g(u) = g(u_i + tv) - g(u + tv).$$

Thus,

$$g(u + tv) - g(u) = g(u_i + tv) - g(u_i) = t \nabla g(u_i) \cdot v < 0.$$

Thus, v is a descent direction. □

We recall that G_1 divides $\mathbb{R}^m$ into domains where G_1 is linear in the interior of these domains and G_1 is nonsmooth on the boundary of these domains. Notice if R is one such domain, then ∂R is contained in the union of hyperplanes of the form $\{u_i = u_j\}$ and/or $\{u_i = f_i\}$ for some i, j.

Using Lemma 4.5, we prove in the next few lemmas that if at a point u on the boundary of one of these regions there is a descent direction for G_1, then there is also an α-descent direction in the lower dimensional space to which we have stepped.

Lemma 4.6. *As above, let $K(u)$ be the cone of descent directions for G_1 at u.*

a. $v \in K(u)$ if and only if $G_1^\circ(u; v) \leq 0$.
b. If $v \in \partial K(u)$ then $G_1(u + tv) = G_1(u)$ for all $t > 0$ sufficiently small.

Proof.

a. First note that using Lemma 4.5, and since G_1 is piecewise affine with finitely many pieces, we get

$$G_1^\circ(u;v) < 0 \Rightarrow v \in K(u).$$

We need only show that

$$v \in K(u) \Rightarrow G_1^\circ(u;v) < 0.$$

Let $v \in K(u)$. Then $G_1(u+tv) - G_1(u) < 0$ for all $t > 0$ sufficiently small. We also know that since G_1 is convex we have that $G_1(u) - G_1(u - tv) < 0$ for any $t > 0$. Suppose that at u, $u_i \neq u_j$ for some i, j and $u_k \neq f_k$ for some k, then we choose $\varepsilon > 0$ sufficiently small so that for all $\tilde{u} \in B_\varepsilon(u)$, $\tilde{u}_i \neq \tilde{u}_j$ and $\tilde{u}_k \neq f_k$. Let $y \in B_\varepsilon(u)$. Let R be a region as described above. We now break this argument into cases:

Case 1: Suppose thats $u, u + tv, y, y + tv \in \partial R$. We know then that G_1 is continuous and affine in $B_\varepsilon(u) \cap \partial R$. Thus, $G_1(y + tv) - G_1(y) < 0$.

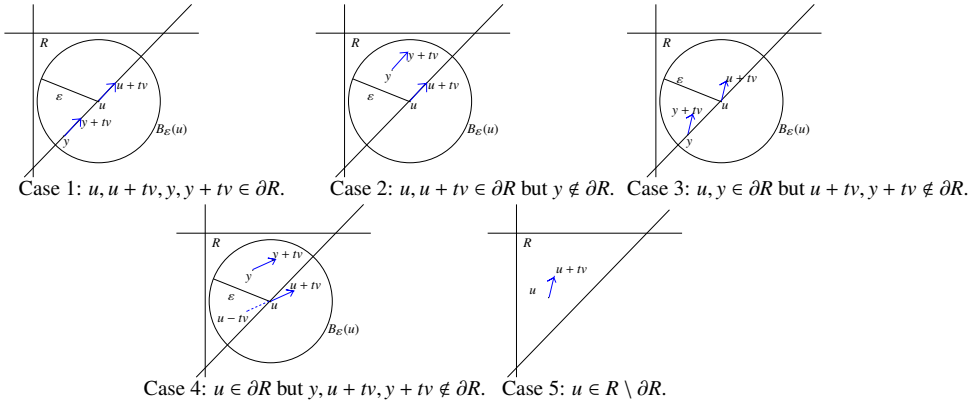

FIGURE 2. 1D examples for the cases of Lemma 4.6.

Case 2: Suppose that $u, u + tv \in \partial R$, but that $y \notin \partial R$. Since G_1 is affine in $B_\varepsilon(u) \cap \overline{R}$, $G_1(y + tv) - G_1(y) < 0$.

Case 3: Suppose that $u, y \in \partial R$, but $u + tv, y + tv \notin \partial R$, then using that G_1 is affine in R, we see that $G_1(y + tv) - G_1(y) < 0$.
In each of the above cases, we see then that $G_1^\circ(u;v) < 0$.

Case 4: Suppose that $u \in \partial R$, but $u + tv, y \notin \partial R$. Then by convexity of G_1 and that G_1 is affine on $B_\varepsilon(u) \cap \overline{R}$, using case 3, we know that

$$G_1(y + tv) - G_1(y) = G_1(u) - G_1(u - tv) < 0. \tag{11}$$

Case 5: Finally, suppose that $u \in R \setminus \partial R$, then G_1 is smooth at u. Thus $G_1^\circ(u;v) = \nabla G_1(u) \cdot v < 0$.

b. Let $v \in \partial K(u)$ then we can construct a sequence $\{v_k\} \subseteq K(u)$ so that $v_k \to v$. Then there is a k_0 sufficiently large so that for all $k \geq k_0$, $\|v - v_k\| < \epsilon$ for some $\epsilon > 0$. But $G_1(u + tv_k) - G_1(u) < 0$ for all k since $v_k \in K(u)$. G_1 continuous gives us that $G_1(u + tv) - G_1(u) < \epsilon$ for all $\epsilon > 0$. Thus $G_1(u + tv) - G_1(u) \leq 0$. But if $G_1(u + tv) - G_1(u) < 0$ then, by continuity, $v \in K(u)$. Thus, $G_1(u + tv) - G_1(u) = 0$.

□

Definition 4.2. Let $\mathcal{P} \subset \mathbb{R}^m$. We define

$$\pi : \mathcal{P} \to \mathbb{R}^{\tilde{m}} \quad \text{by} \quad \pi(u) = \tilde{u},$$

to be the projection map that removes redundancy in u, where $\tilde{m} \leq m$. That is, if $\{u_i = u_j\}$ is active at u, then the ith or jth (whichever is larger) component is removed from u to get $\tilde{u}$ and if $\{u_i = f_i\}$ is active at u, then the ith component of u is removed to get $\tilde{u}$.

Note, this projection is invertible.

Example 4.1. For example, let $\mathcal{P} \subset \mathbb{R}^4$ be the 2-dimensional subset given by

$$\mathcal{P} = \{(u_1, u_2, u_3, u_4) | u_2 = u_1 \text{ and } u_4 = f_4\}. \tag{12}$$

We define $\pi : \mathcal{P} \to \mathbb{R}^2$ by

$$\pi(u_1, u_1, u_3, f_4) = (u_1, u_3). \tag{13}$$

If we pick any point in $\mathbb{R}^2$, we can find its inverse projection in $\mathcal{P}$ by

$$\pi^{-1}(u_1, u_2) = (u_1, u_1, u_2, f_4). \tag{14}$$

Lemma 4.7. *Let $u \in \partial R$ where R is one of the regions described above. Let $K(u) \neq \emptyset$. Then $\mathcal{N}(u) \cap K(u) \cap \partial R$ has an α-descent direction, where $\mathcal{N}(u)$ is some neighborhood of u.*

Proof. We know that ∂R is contained in the union and intersection of some hyperplanes of the form $\{u_i = u_j\}$ and/or $\{u_i = f_i\}$ for some i, j. By looking in $\mathcal{N}(u) \cap K(u) \cap \partial R$ we can restrict G_1 to points in the lower dimensional space, $\mathcal{P}$, defined by the active hyperplanes at u. Let

$$\tilde{G_1} : \mathbb{R}^{\tilde{m}} \to \mathbb{R}$$

be defined by $\tilde{G_1}(\tilde{u}) = G_1(u)$, where $\pi^{-1}(\tilde{u}) = u$.

Then we see that $\nabla \tilde{G_1}(\tilde{u})$ exists. Since G_1 is affine in $\overline{R}$ we have that $K(\tilde{u}) = \{v | \langle v, \nabla G_1(u) \rangle < 0\}$ which is a half space and therefore contains an α-direction, $\tilde{v}$. We then have that $v = \pi^{-1}(\tilde{v})$ is an α-descent direction in $\mathcal{P}$. □

Using the proof of Lemma 4.7, the following result holds immediately.

Remark 4.1. v is an α-descent direction at $u \Leftrightarrow \pi(v)$ is an α-descent direction at $\pi(u)$.

Remark 4.2. Lemma 4.7 gives us that if at $u^{(k)}$ there is a descent direction for G_1 then there is also an α-descent direction for G_1 at $u^{(k)}$.

4.2. **Proof of Lemma 4.2.**

Lemma 4.8. *If at $\tilde{u} \in \mathbb{R}^\ell$ an α-descent direction, $\tilde{v}$, exists then there exists an α-descent direction of the form*

$$\hat{\tilde{v}} = \sum \alpha_i \tilde{e}_i, \tag{15}$$

where $\alpha_i \in \{-1, 0, 1\}$, $\tilde{e}_i$ are coordinate directions, and $\alpha_i \tilde{e}_i$ are descent directions whenever $\alpha_i \neq 0$.

Proof. We can let $\tilde{v} = \sum_{i=1}^{\ell} \alpha_i \tilde{e}_i$ where $\alpha_i \in \{-1, 0, 1\}$ and $\tilde{e}_i$ are coordinate directions. Since $\tilde{G_1}$ is linear on $\mathbb{R}^\ell$

$$\begin{aligned} G_1(\tilde{u} + t\tilde{v}) &= G_1\left(\tilde{u} + t \sum_{i=1}^{\ell} \alpha_i \tilde{e}_i\right) = G_1\left(\sum_{i=1}^{\ell} (\eta_i \tilde{u} + t\alpha_i \tilde{e}_i)\right) \\ &= G_1\left(\sum_{i=1}^{\ell} \eta_i \left(\tilde{u} + \frac{t}{\eta_i} \alpha_i \tilde{e}_i\right)\right) = \sum_{i=1}^{\ell} \eta_i G_1\left(\tilde{u} + \frac{t}{\eta_i} \alpha_i \tilde{e}_i\right), \end{aligned} \tag{16}$$

where $\sum_{i=1}^{\ell} \eta_i = 1$. Now, since $\tilde{v}$ is a descent direction, some of the terms, $G_1\left(\tilde{u} + \frac{t}{\eta_i} \alpha_i \tilde{e}_i\right) < G_1(\tilde{u})$. Otherwise, if $G_1\left(\tilde{u} + \frac{t}{\eta_i} \alpha_i \tilde{e}_i\right) \geq G_1(\tilde{u})$, we would have

$$G_1(\tilde{u} + t\tilde{v}) = \sum_{i=1}^{\ell} \eta_i G_1\left(\tilde{u} + \frac{t}{\eta_i} \alpha_i \tilde{e}_i\right) \geq \sum_{i=1}^{\ell} \eta_i G_1(\tilde{u}) = G_1(\tilde{u}). \tag{17}$$

Let $\mathcal{I} = \{i | G_1\left(\tilde{u} + \frac{t}{\eta_i}\alpha_i\tilde{e}_i\right) < G_1(\tilde{u})\}$. Now, we know that $G_1(\tilde{u} + t\tilde{v}) < G_1(\tilde{u})$ for all $t > 0$ sufficiently small. Thus, $G_1(\tilde{u} + \tilde{t}\alpha_i\tilde{e}_i) < G_1(\tilde{u})$ for $\tilde{t} > 0$ sufficiently small and $i \in \mathcal{I}$. Thus, $\alpha_i\tilde{e}_i$ is a descent direction whenever $i \in \mathcal{I}$. And we can create $\hat{\tilde{v}}$, by choosing $\hat{\alpha}$ in the following way

$$\hat{\alpha}_i = \begin{cases} \alpha_i & \text{whenever } i \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases} . \tag{18}$$

Then

$$\hat{\tilde{v}} = \sum_{j=1}^{\ell} \hat{\alpha}_i\tilde{e}_i. \tag{19}$$

Then

$$G_1(\tilde{u}) > G_1(\tilde{u} + t\hat{\tilde{v}}) = G_1(\tilde{u}) + t\sum_{j=1}^{s} G_1(\alpha_{i_j}\tilde{e}_i). \tag{20}$$

□

4.3. **Proof of Lemma 4.3.** Recall that in Algorithm 3.1, we step to hyperplanes where G_1 is nonsmooth. Then we work within the lower dimensional space defined by the hyperplanes to which we have stepped. We define clusters below and use these clusters to algorithmically define the lower dimensional space.

Definition 4.3. Let $\mathcal{C}^{(k)} = \left\{c_1 = 1 < c_2 < \ldots < c_{q_k} \leq m \middle| u^{(k)}_{c_i-1} \neq u^{(k)}_{c_i}\right\}$ be the set of indices so that $u^{(k)}_j = u^{(k)}_{c_i}$ for all $c_i \leq j \leq c_{i+1} - 1$. Then we define a cluster, $C^{(k)}_i$, to be the set of indices j so that $u^{(k)}_j$ has the same value as $u^{(k)}_{c_i}$, that is,

$$C^{(k)}_i \equiv \{j | \text{ for all } c_i \leq j \leq c_{i+1} - 1\}.$$

Notice, that a cluster will have size $|C^{(k)}_i| = c_{i+1} - c_i$ (the last cluster will have size $m - c_{q_k} + 1$) and $\sum_{i=1}^{q} |C^{(k)}_i| = m$. We will say that $u^{(k)}_j$ is in a cluster C_i and mean $j \in C_i$.

Definition 4.4. If $u^{(k)} = \left(u^{(k)}_1, \ldots u^{(k)}_m\right) \in \mathbb{R}^m$ is obtained by k iterations of Algorithm 3.1, we let

$$\alpha^{(k)}_i = \begin{cases} -1 & \text{if } -\sum_{j=c_i}^{c_{j+1}-1} e_j \text{ is a descent direction for } G_1 \text{ at } u^{(k)} \\ 1 & \text{if } \sum_{j=c_i}^{c_{j+1}-1} e_j \text{ is a descent direction for } G_1 \text{ at } u^{(k)} \\ 0 & \text{otherwise} \end{cases}$$

for $1 \leq i \leq m$. For fixed boundary conditions, we set $\alpha^{(k)}_1, \alpha^{(k)}_m = 0$.

The next two lemmas show that our clusters only get larger in Algorithm 3.1 and that we find descent when no point, $u^{(k)}_j$ in cluster C_i will move independently of the cluster. This means that we never go back to the higher dimensional space, that is, the algorithm continues to step to lower and lower dimensional spaces. Actually, these two lemmas are for a more general algorithm, that is, for an algorithm that finds minimizers of L^1pTV for $0 < p \leq 1$. The first of these two lemmas gives us this result for iteration 1 of Algorithm 3.1 and the second lemma gives the result for all other iterations. Both lemmas are proved by looking at the various neighborhood cases that are possible at each point u_i to show that if u_i is in a cluster, C_i, it won't break away from the cluster in next steps. And then we determine which λ values give us descent when moving a cluster C_i.

4.3.1. *Clusters need not break up for descent (iteration 1).*

Lemma 4.9. *Let $0 < p \leq 1$. Let $\eta_\ell \equiv |u_{c_i-1} - u_{c_i}|$ and $\eta_r \equiv |u_{c_{i+1}} - u_{c_{i+1}-1}|$. Then the following statements hold.*

(1) If there exists a cluster C_i with $\{u_{c_i-1} > u_{c_i}$ and $u_{c_{i+1}-1} > u_{c_{i+1}}\}$ and

$$0 < \lambda < \frac{\eta_\ell^p - (\eta_\ell - \alpha\eta)^p + \eta_r^p - (\eta_r + \alpha\eta)^p}{\eta|C_i|} \tag{21}$$

then a descent direction for G_p, at the point $u = f$, is

$$\sum_{j=c_i}^{c_{i+1}-1} e_j \text{ when } \eta_r < \eta_\ell \quad \text{and} \quad -\sum_{j=c_i}^{c_{i+1}-1} e_j \text{ when } \eta_r > \eta_\ell. \tag{22}$$

(2) If there exists a cluster C_i with $\{u_{c_i-1} < u_{c_i}$ and $u_{c_{i+1}-1} < u_{c_{i+1}}\}$ and

$$0 < \lambda < \frac{\eta_\ell^p - (\eta_\ell + \alpha\eta)^p + \eta_r^p - (\eta_r - \alpha\eta)^p}{\eta|C_i|} \tag{23}$$

then a descent direction for G_p, at the point $u = f$, is

$$\sum_{j=c_i}^{c_{i+1}-1} e_j \text{ when } \eta_r > \eta_\ell \quad \text{and} \quad -\sum_{j=c_i}^{c_{i+1}-1} e_j \text{ when } \eta_r < \eta_\ell. \tag{24}$$

(3) If there exists a cluster C_i with $\{u_{c_i-1} < u_{c_i}$ and $u_{c_{i+1}-1} < u_{c_{i+1}}\}$ and

$$0 < \lambda < \frac{\eta_\ell^p - (\eta_\ell - \alpha\eta)^p + \eta_r^p - (\eta_r - \alpha\eta)^p}{\eta|C_i|} \tag{25}$$

then a descent direction for G_p, at the point $u = f$, is

$$\sum_{j=c_i}^{c_{i+1}-1} e_j. \tag{26}$$

(4) If there exists a cluster C_i with $\{u_{c_i-1} < u_{c_i}$ and $u_{c_{i+1}-1} < u_{c_{i+1}}\}$ and

$$0 < \lambda < \frac{\eta_\ell^p - (\eta_\ell + \alpha\eta)^p + \eta_r^p - (\eta_r + \alpha\eta)^p}{\eta|C_i|} \tag{27}$$

then a descent direction for G_p, at the point $u = f$, is

$$-\sum_{j=c_i}^{c_{i+1}-1} e_j. \tag{28}$$

Notice, for $p = 1$, in cases 1 and 2, the condition for λ is $0 < \lambda < 0$. Since there is no such λ, we see that our L^1TV algorithm will not find descent in these cases. The condition for λ for $p = 1$ in cases 3 and 4 is

$$0 < \lambda < \frac{2}{|C_i|}. \tag{29}$$

Proof. We begin this proof by showing that if we start with $u = f$, then to get descent we need not break up clusters. We break this into 2 cases. We assume for these cases that u_{c_i} is not a point on the boundary of Ω ($1 < c_i < m$). We prove this by considering whether or not

$$\begin{aligned} G_p(u + \eta\alpha e_i) - G_p(u) = |u_i + \eta\alpha - u_{i-1}|^p + |u_i + \eta\alpha - u_{i+1}|^p + \lambda|u_i + \eta\alpha - f_i| \\ -|u_i - u_{i-1}|^p + |u_i - u_{i+1}|^p + \lambda|u_i - f_i| < 0. \end{aligned} \tag{30}$$

Case 1: Suppose $u_{c_i-1} = u_{c_i} = u_{c_i+1}$. We assume that $\eta > 0$ is small and compute

$$G_p(u + \eta\alpha e_i) - G_p(u) = 2\eta^p + \lambda\eta > 0, \ \forall \eta > 0. \tag{31}$$

Thus, in this case, no descent exists. That is, a data point in the middle of a cluster will not move in the first iteration. We can also see that for each point we move from the inside of a cluster causes an increase in the

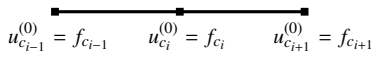

FIGURE 3. Case1: $u^{(0)}_{c_i}$ is a point in the middle of a cluster.

fidelity term and we also see that the variation term will not decrease. Thus, we see that no descent is found by moving any points from inside the cluster in a direction different than the rest of cluster.

Case 2: Suppose $u_{c_i-1} = u_{c_i} < u_{c_i+1}$ (see Figure 4)

We assume that $0 < \eta$ is at most $\eta_r \equiv |u_{c_i+1} - u_{c_i}|$ and compute

$$\begin{aligned} G_p(u + \eta\alpha e_i) - G_p(u) = &\quad \eta^p + (\eta_r - \alpha\eta)^p - \eta_r^p + \lambda\eta \\ = &\quad \eta^p(1 - a^p + (a - \alpha)^p) + \lambda\eta \quad \text{where } a = \frac{\eta_r}{\eta} \geq 1 \end{aligned} \tag{32}$$

Notice that if $\alpha = -1$, we have $G_p(u + \eta\alpha e_i) - G_p(u) = 1 - a^p + (a + 1)^p + \lambda\eta > 0$. Now if $\alpha = 1$, we have $G_p(u + \eta\alpha e_i) - G_p(u) = 1 - a^p + (a - 1)^p + \lambda\eta$. This is also positive since $a^p - 1 = (a - 1)^p$ when $a = 1$ and the left-hand side is increasing faster than the right-hand side for $a > 1$. Thus, in this case, no descent

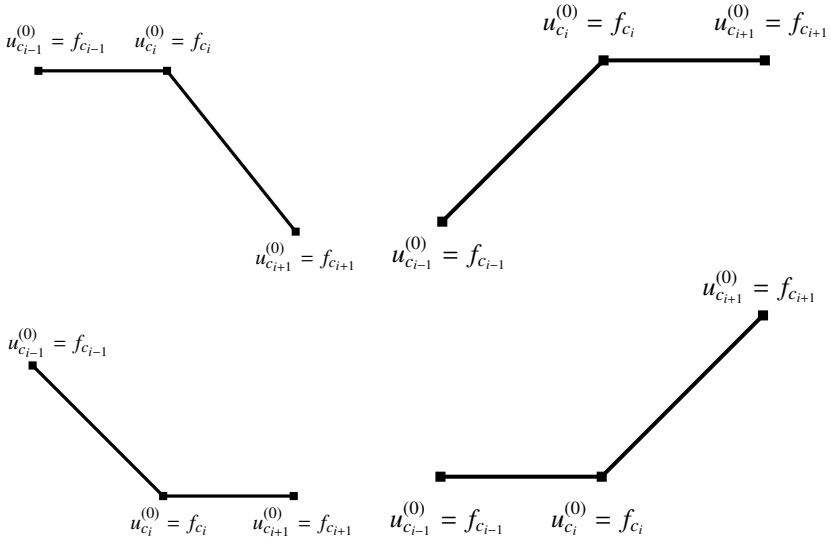

FIGURE 4. Case2: $u^{(0)}_{c_i}$ is a point on the end of a cluster (four cases).

exists. That is, a data point on the end of a cluster will not move in the first iteration. Notice the other cases for $u_{c_i-1} = u_{c_i} \neq u_{c_i+1}$ or $u_{c_i+1} = u_{c_i} \neq u_{c_i-1}$ have the same result and are proved similarly.

Notice that if we moved several points at the end together away from the rest of the cluster, we will see the same result in the variation term, but we will multiply the fidelity term by the number of points we move. Therefore, clusters will not break apart in the first iteration of our algorithm. Next we show that when clusters move together in the first iteration we find descent when λ satisfies the conditions stated in the lemma. We show this by considering whether or not

$$\begin{aligned} G_p\left(u + \eta\alpha \sum_{j=c_i}^{c_{i+1}-1} e_j\right) - G_p(u) &= |u_{c_i} + \eta\alpha - u_{c_i-1}|^p + |u_{c_{i+1}-1} + \eta\alpha - u_{c_{i+1}}|^p + \lambda \sum_{j=c_i}^{c_{i+1}-1} |u_j + \eta\alpha - f_j| \\ &- |u_{c_i} - u_{c_i-1}|^p + |u_{c_{i+1}-1} - u_{c_{i+1}}|^p + \lambda \sum_{j=c_i}^{c_{i+1}-1} |u_j - f_j| < 0. \end{aligned} \tag{33}$$

We break this step into four cases. Let $\eta_r \equiv |u_{c_{i+1}} - u_{c_{i+1}-1}|$ and $\eta_\ell \equiv |u_{c_i} - u_{c_i-1}|$. We also assume $\eta \leq \min\{\eta_r, \eta_\ell\}$.

Case 1: Suppose $u_{c_i-1} > u_{c_i} = \ldots = u_{c_{i+1}-1} > u_{c_{i+1}}$. We compute

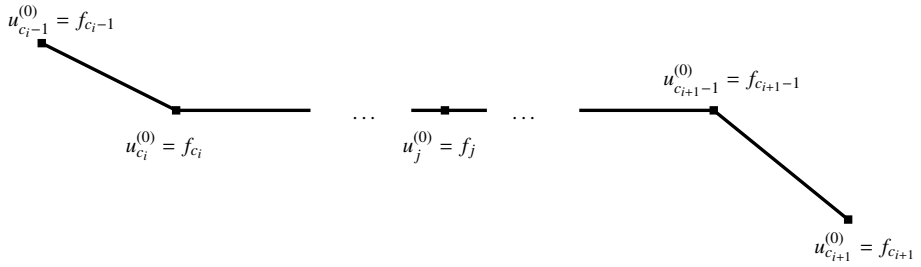

FIGURE 5. Case 1: $C_i(u_i^{(0)})$ has left neighbor above and right neighbor below.

$$G_p\left(u + \alpha\eta \sum_{j=c_i}^{c_{i+1}-1} e_j\right) - G_p(u) = \quad (\eta_\ell - \alpha\eta)^p - \eta_\ell^p + (\eta_r + \alpha\eta)^p - \eta_r^p + \lambda|C_i|\eta. \tag{34}$$

We see that for this case, we find descent when

$$0 < \lambda < \frac{\eta_\ell^p - (\eta_\ell - \alpha\eta)^p + \eta_r^p - (\eta_r + \alpha\eta)^p}{\eta|C_i|}. \tag{35}$$

That is, descent is found with this λ by moving the cluster up when $\eta_r < \eta_\ell$ and down when $\eta_r > \eta_\ell$. (For $p = 1$, this condition is $0 < \lambda < 0$, thus descent does not exist.)

Case 2: Suppose $u_{c_i-1} < u_{c_i} = \ldots = u_{c_{i+1}-1} < u_{c_{i+1}}$. If we use a similar argument to that of Case 1, we see that for this case, we find descent when

$$\lambda < \frac{\eta_\ell^p - (\eta_\ell + \alpha\eta)^p + \eta_r^p - (\eta_r - \alpha\eta)^p}{\eta|C_i|}. \tag{36}$$

That is, descent is found, with this λ, by moving the cluster up when $\eta_r > \eta_\ell$ and down when $\eta_r < \eta_\ell$. (For $p = 1$, this condition is $0 < \lambda < 0$, thus descent does not exist.)

Case 3: Suppose $u_{c_i} = \ldots = u_{c_{i+1}-1} < u_{c_i-1}, u_{c_{i+1}}$. We compute

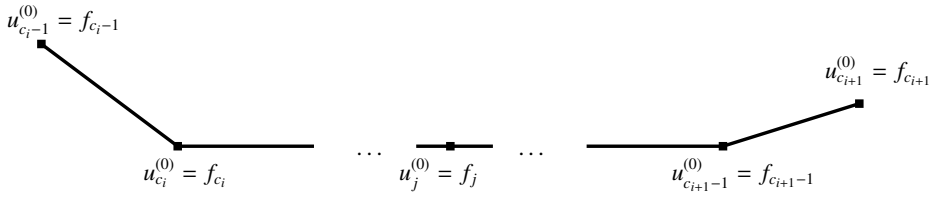

FIGURE 6. Case 3: $C_i(u_i^{(0)})$ has both neighbors above.

$$G_p\left(u + \alpha\eta \sum_{j=c_i}^{c_{i+1}-1} e_j\right) - G_p(u) = \quad (\eta_\ell - \alpha\eta)^p - \eta_\ell^p + (\eta_r - \alpha\eta)^p - \eta_r^p + \lambda|C_i|\eta. \tag{37}$$

We see that for this case, we find descent by moving the cluster up when

$$\lambda < \frac{\eta_\ell^p - (\eta_\ell - \eta)^p + \eta_r^p - (\eta_r - \eta)^p}{\eta|C_i|}. \tag{38}$$

(For $p = 1$, this condition is $0 < \lambda < 2/|C_i|$.)

Case 4: Suppose $u_{c_i} = \ldots = u_{c_{i+1}-1} > u_{c_i-1}, u_{c_{i+1}}$. Again, this case is similar to Case 3. A similar argument gives us that moving the cluster down gives descent when

$$\lambda < \frac{\eta_\ell^p - (\eta_\ell - \eta)^p + \eta_r^p - (\eta_r - \eta)^p}{\eta |C_i|}. \tag{39}$$

(For $p = 1$, this condition is $0 < \lambda < 2/|C_i|$.)

Finally, in the case that we choose the free boundary option (letting boundaries move), we show that if u_{c_i} is on the boundary, then we find descent when λ satisfies the conditions stated in the lemma. We prove this for the left endpoint of the data since the argument for the right endpoint is similar. We break this into two cases.

Case 1: $u_1 = u_2$. In this case, we assume $\eta > 0$ is small and we compute

$$G_p(u + \eta \alpha e_i) - G_p(u) = \eta^p + \lambda \eta > 0. \tag{40}$$

Thus we will not find descent by moving this endpoint without its neighbors.

Case 2: $u_1 = \ldots = u_{c_2-1} < u_{c_2}$. In this case, we assume $0 < \eta \leq \eta_r = (u_{c_2} - u_{c_2-1})$ and we compute

$$G_p(u + \eta \alpha e_i) - G_p(u) = |C_1| \lambda \eta + (\eta_r - \alpha \eta)^p - \eta_r^p. \tag{41}$$

Thus, we find descent by moving the endpoint up whenever $\lambda < \frac{\eta_r^p - (\eta_r - \eta)^p}{\eta |C_1|} = \frac{\eta_r^p}{\eta |C_1|}$. (For $p = 1$, this condition is $0 < \lambda < 1/|C_1|$.) □

4.3.2. *Clusters need not break up for descent (iteration k).* For this lemma, we need to define some notation.

Definition 4.5. We define q_g, q_l, q_e to be the number of elements, $u_j^{(k)}$, in the cluster that are greater than, less than, and equal to (respectively) the corresponding f_j:

$$q_g = \left| \left\{ u_j^{(k)} \in C\left(u_i^{(k)}\right) \middle| \, u_j^{(k)} > f_j \right\} \right|,$$

$$q_l = \left| \left\{ u_j^{(k)} \in C\left(u_i^{(k)}\right) \middle| \, u_j^{(k)} < f_j \right\} \right|,$$

and

$$q_e = \left| \left\{ u_j^{(k)} \in C\left(u_i^{(k)}\right) \middle| \, u_j^{(k)} = f_j \right\} \right|.$$

Lemma 4.10. *Let $0 < p \leq 1$. Let q_g, q_e, and q_ℓ be defined as above. Let $\eta_\ell \equiv |u_{c_i-1} - u_{c_i}|$ and $\eta_r \equiv |u_{c_{i+1}} - u_{c_{i+1}-1}|$ and let $u^{(k)}$ be a point obtained using a ht algorithm. Then the following statements hold.*

(1) If there exists a cluster C_i with $\{u_{c_i-1} > u_{c_i} = u_{c_{i+1}-1} > u_{c_{i+1}}\}$ and

$$0 < \lambda < \frac{\eta_\ell^p - (\eta_\ell - \alpha\eta)^p + \eta_r^p - (\eta_r + \alpha\eta)^p}{\eta((q_g - q_\ell)\alpha + q_e)} \tag{42}$$

then a descent direction for G_p at the point $u^{(k)}$ is

$$\sum_{j=c_i}^{c_{i+1}-1} e_j \text{ when } \eta_r < \eta_\ell \quad \text{and} \quad -\sum_{j=c_i}^{c_{i+1}-1} e_j \text{ when } \eta_r > \eta_\ell. \tag{43}$$

(2) If there exists a cluster C_i with $\{u_{c_i-1} < u_{c_i} = u_{c_{i+1}-1} < u_{c_{i+1}}\}$ and

$$0 < \lambda < \frac{\eta_\ell^p - (\eta_\ell + \alpha\eta)^p + \eta_r^p - (\eta_r - \alpha\eta)^p}{\eta((q_g - q_\ell)\alpha + q_e)} \tag{44}$$

then a descent direction for G_p at the point $u^{(k)}$ is

$$\sum_{j=c_i}^{c_{i+1}-1} e_j \text{ when } \eta_r > \eta_\ell \quad \text{and} \quad -\sum_{j=c_i}^{c_{i+1}-1} e_j \text{ when } \eta_r < \eta_\ell. \tag{45}$$

(3) If there exists a cluster C_i with $\{u_{c_i-1} > u_{c_i}$ and $u_{c_i} = u_{c_{i+1}-1} < u_{c_{i+1}}\}$ and

$$0 < \lambda < \frac{\eta_\ell^p - (\eta_\ell - \eta)^p + \eta_r^p - (\eta_r - \eta)^p}{\eta(q_g - q_\ell + q_e)} \tag{46}$$

then a descent direction for G_p at the point $u^{(k)}$ is

$$\sum_{j=c_i}^{c_{i+1}-1} e_j. \tag{47}$$

(4) If there exists a cluster C_i with $\{u_{c_i-1} < u_{c_i}$ and $u_{c_i} = u_{c_{i+1}-1} > u_{c_{i+1}}\}$ and

$$0 < \lambda < \frac{\eta_\ell^p - (\eta_\ell - \eta)^p + \eta_r^p - (\eta_r - \eta)^p}{\eta(-q_g + q_\ell + q_e)} \tag{48}$$

then a descent direction for G_p at the point $u^{(k)}$ is

$$-\sum_{j=c_i}^{c_{i+1}-1} e_j. \tag{49}$$

Again, for $p = 1$, in Cases 1 and 2, the condition for λ is $0 < \lambda < 0$. Thus our L^1TV algorithm will not find descent in these cases. The condition for λ for $p = 1$ in case 3 is

$$0 < \lambda < \frac{2}{q_g - q_\ell + q_e}. \tag{50}$$

And for Case 4, with $p = 1$, the condition for λ is

$$0 < \lambda < \frac{2}{-q_g + q_\ell + q_e}. \tag{51}$$

Proof. We begin again by showing that to get descent we need not break up clusters. Again for ease of notation, we write u instead of $u^{(k)}$. We break this into two cases.
Case 1: $u_{c_i-1} = u_{c_i} = u_{c_i+1}$. We assume $\eta > 0$ is small. We compute

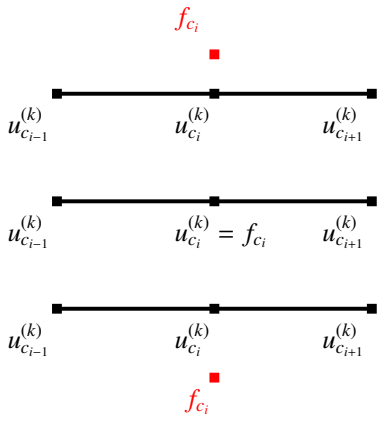

FIGURE 7. Case 1: $u^{(k)}_{c_i}$ is a point in the middle of a cluster (3 possible cases).

$$G_p(u + \eta\alpha e_i) - G_p(u) = 2\eta^p + \lambda\eta((q_g - q_\ell)\alpha + q_e). \tag{52}$$

Here we treat u_{c_i} as a cluster of size 1 by moving it alone. This means that only one of q_g, q_e, q_ℓ is 1, while the others are 0. Notice that if $q_e = 1$, there is no descent. Notice also that $\lambda > 2\eta^{p-1}$ gives descent by moving u_{c_i} toward f_{c_i}, but this would be undoing what we did in a previous step. That is, in the previous

step, we could have moved u_{c_i} to this cluster by itself in which case moving it back by itself is undoing a step that gave us descent and thus it would give us ascent. The other possible case would have been if we moved u_{c_i} with a cluster to this position. In this case, we know from Lemma 4.9 that to move it by itself away would be a step that gives ascent. Consequently, we will not find descent breaking up this cluster.

Case 2: $u_{c_i-1} = u_{c_i} \neq u_{c_i+1}$ or $u_{c_i+1} = u_{c_i} \neq u_{c_i-1}$. We will prove one of these cases, namely $u_{c_i-1} = u_{c_i} < u_{c_i+1}$, because the four cases are similar in argument. We assume that $\eta \leq \min\{\eta_r, |f_{c_i} - u_{c_i}|\}$ and we compute

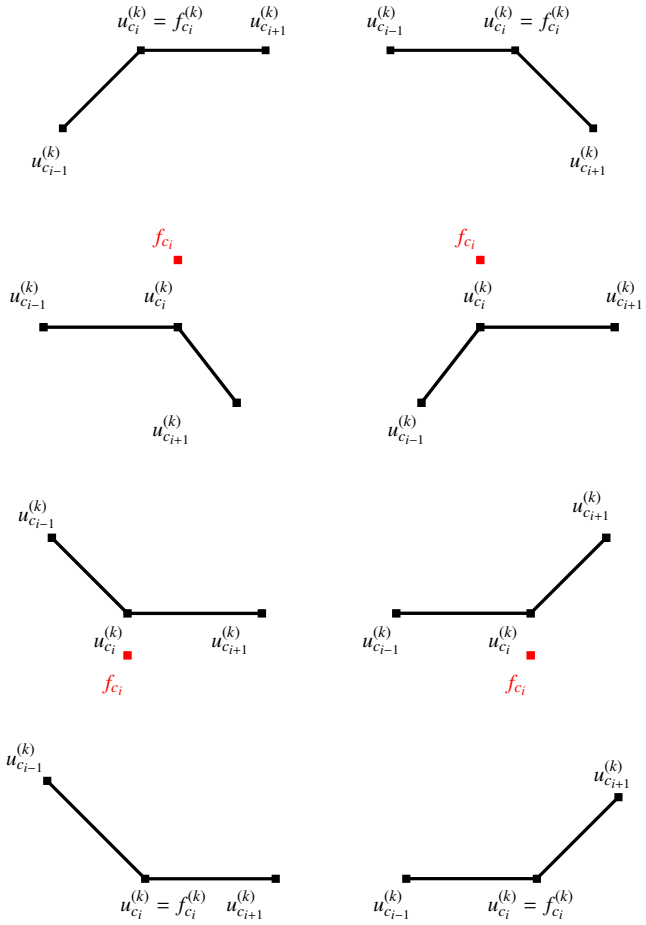

FIGURE 8. Case 2: $u_{c_i}^{(k)}$ is a point on the end of a cluster (8 possible cases).

$$G_p(u + \eta\alpha e_i) - G_p(u) = (\eta_r - \eta\alpha)^p - \eta_r^p + \eta^p + \lambda\eta((q_g - q_\ell)\alpha + q_e). \tag{53}$$

Notice, we get descent if

$$\lambda < \frac{\eta_r^p - \eta^p - (\eta_r - \eta\alpha)^p}{\eta((q_g - q_\ell)\alpha + q_e)} = \eta^{p-1}\frac{a^p - 1 - (a-\alpha)^p}{(q_g - q_\ell)\alpha + q_e} < 0 \tag{54}$$

or

$$\lambda > \frac{-\eta_r^p + \eta^p + (\eta_r - \eta\alpha)^p}{\eta((q_g - q_\ell)\alpha + q_e)} = \eta^{p-1}\frac{a^p - 1 - (a-\alpha)^p}{(q_g - q_\ell)\alpha + q_e} \tag{55}$$

However, notice that this second inequality is taking us back toward f_i which is again, undoing a previous step. The first inequality says $\lambda < 0$. Thus, we do not find descent in this case either. Thus he algorithm will not break up clusters.

Now we consider moving the full cluster together. We will show that we find descent when λ satisfies the conditions stated in the lemma. We break this step into four cases. Let $\eta_r \equiv |u_{c_{i+1}} - u_{c_{i+1}-1}|$ and $\eta_\ell \equiv |u_{c_i} - u_{c_i-1}|$. We also assume $\eta \leq \min\{\eta_r, \eta_\ell\}$.

Case 1: Suppose $u_{c_i-1} > u_{c_i} = \ldots = u_{c_{i+1}-1} > u_{c_{i+1}}$. We compute

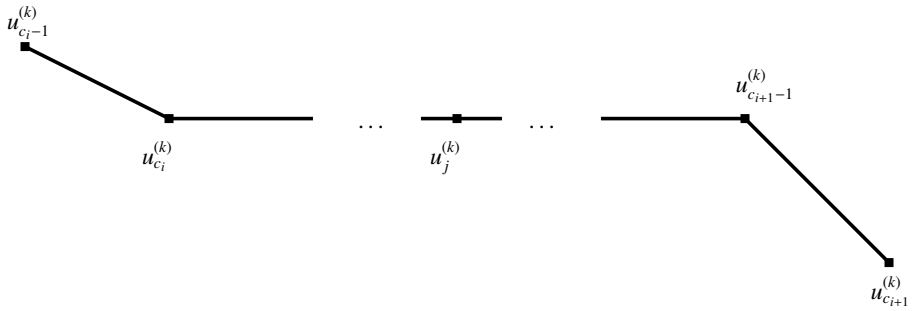

FIGURE 9. Case 1: $C_i(u_i^{(k)})$ has left neighbor above and right neighbor below.

$$G_p\left(u + \alpha\eta \sum_{j=c_i}^{c_{i+1}-1} e_j\right) - G_p(u) = \quad (\eta_\ell - \alpha\eta)^p - \eta_\ell^p + (\eta_r + \alpha\eta)^p - \eta_r^p + \lambda((q_g - q_\ell)\alpha + q_e)\eta. \tag{56}$$

We see that for this case, we find descent when

$$\lambda < \frac{\eta_\ell^p - (\eta_\ell - \alpha\eta)^p + \eta_r^p - (\eta_r + \alpha\eta)^p}{\eta((q_g - q_\ell)\alpha + q_e)}. \tag{57}$$

As in the last lemma, we find descent, with this λ, by moving the cluster up when $\eta_r > \eta_\ell$ and down when $\eta_r < \eta_\ell$. (As we saw in the last lemma, for $p = 1$, this condition is $0 < \lambda < 0$. Thus descent does not exist.)

Case 2: Suppose $u_{c_i-1} < u_{c_i} = \ldots = u_{c_{i+1}-1} < u_{c_{i+1}}$. If we use a similar argument to that of Case 1, we see that for this case, we find descent when

$$\lambda < \frac{\eta_\ell^p - (\eta_\ell + \alpha\eta)^p + \eta_r^p - (\eta_r - \alpha\eta)^p}{\eta((q_g - q_\ell)\alpha + q_e)}. \tag{58}$$

That is, descent is found, with this λ, by moving the cluster up when $\eta_r < \eta_\ell$ and down when $\eta_r > \eta_\ell$. (For $p = 1$ descent does not exist.)

Case 3: Suppose $u_{c_i} = \ldots = u_{c_{i+1}-1} < u_{c_i-1}, u_{c_{i+1}}$. We compute

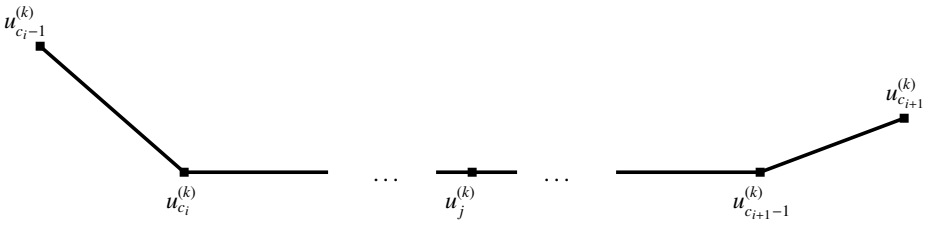

FIGURE 10. Case 3: $C_i(u_i^{(k)})$ has both neighbors above.

$$G_p\left(u + \alpha\eta \sum_{j=c_i}^{c_{i+1}-1} e_j\right) - G_p(u) = \quad (\eta_\ell - \alpha\eta)^p - \eta_\ell^p + (\eta_r - \alpha\eta)^p - \eta_r^p + \lambda((q_g - q_\ell)\alpha + q_e)\eta. \tag{59}$$

We see that for this case, we find descent by moving the cluster up when

$$\lambda < \frac{\eta_\ell^p - (\eta_\ell - \eta)^p + \eta_r^p - (\eta_r - \eta)^p}{\eta(q_g - q_\ell + q_e)}. \tag{60}$$

(For $p = 1$, this condition is $0 < \lambda < 2/(q_g - q_\ell + q_e)$.)

Case 4: Suppose $u_{c_i} = \ldots = u_{c_{i+1}-1} > u_{c_i-1}, u_{c_{i+1}}$. Again, this case is similar to Case 3. A similar argument gives us that moving the cluster down gives descent when

$$\lambda < \frac{\eta_\ell^p - (\eta_\ell - \eta)^p + \eta_r^p - (\eta_r - \eta)^p}{\eta(-q_g + q_\ell + q_e)}. \tag{61}$$

(For $p = 1$, this condition is $0 < \lambda < 2/(-q_g + q_\ell + q_e)$.) □

Using the proof of Lemmas 4.9 and 4.10, we see that for every cluster $C_i^{(k)}$, we get $C_i^{(k)} \subseteq C_i^{(k+1)}$. That is, no point will leave a cluster and at each iteration the algorithm will reduce the problem to minimizing a lower dimensional problem.

4.4. **Proof of Lemma 4.4.** Up to this point, we have shown that Algorithm 3.1 converges to a minimizer. Now we need to show that the algorithm is indeed finite.

Proof. Suppose $u^{(k)} \notin Y$, then $k > 0$ since $u^{(0)} = f \in Y$. Therefore $\nabla G_1(u^{(k)})$ does not exist because the algorithm always steps to a point where G_1 is nonsmooth. Therefore $u^{(k)} \in \mathcal{H}$, where $\mathcal{H}$ is an $\ell < m$ dimensional hyperplane formed from intersections of some of the $\ell - 1$ hyperplanes of the form $\{u_i = u_j | (i, j) \in E\}$ and or $\{u_i = f_i\}$. Note that $\ell > 1$ since the algorithm stops when $\ell = 1$. If $u^{(k)}$ is not a minimizer there exists a descent direction for G_1 at $u^{(k)}$. Then there exists an α-descent direction in $\mathcal{H}$. The algorithm takes the step in this direction to get $u^{(k+1)}$ which lies on a hyperplane whose dimension is smaller than ℓ. We can continue this process at most ℓ times to land at a point $u^{(k')} \in Y$. □

To summarize this section, we have just shown that Algorithm 3.1 converges to a minimum of G_1 by showing that if there is a descent direction at a point u, then the particular α-direction of Algorithm 3.1 exists and gives strict descent. We also showed that Algorithm 3.1 is finite by showing that the algorithm takes only finitely many steps to get from one point in Y to the next. Thus, Theorem 4.1 holds.

5. Global Minimizers for all λ

In this section we will introduce a more efficient algorithm for the case when $p = 1$. In the proof of Lemmas 4.9 and 4.10, we found conditions on λ for which descent occurs. Here we use those conditions to formulate a new algorithm that does not need to compute G_1 values. Recall, we found that descent occurs, in the $p = 1$ case, only when clusters are lower than both neighbors or higher than both neighbors. We restate the conditions here. In the case when C_i is lower than its neighbors, we find descent in moving the cluster up when

$$0 < \lambda < \frac{2}{q_g - q_\ell + q_e}. \tag{62}$$

For such clusters, we call $Q = q_g - q_\ell + q_e$ the effective cluster size. In the case when C_i is higher than its neighbors, we find descent in moving the cluster down when

$$0 < \lambda < \frac{2}{-q_g + q_\ell + q_e}. \tag{63}$$

For these clusters, we call $Q = -q_g + q_\ell + q_e$ the effective cluster size.

Remark 5.1. In the case of free boundary conditions, if a cluster C_i is on the boundary of the data, we use the results from Lemmas 4.9 and 4.10 to say the effective cluster size is twice the effective cluster size of an interior cluster. We see that this makes sense since moving the cluster only affects the variation based on one neighbor instead of two and therefore it takes a smaller λ to move it.

We also recognize that the value of G_1 is constant if a cluster moves to a height somewhere between its highest neighbor and its lowest neighbor. Instead of moving clusters up and down in parallel, we move up (down) all clusters with the appropriate effective cluster size for the given λ first. These clusters will move up (down) to meet another cluster, stopping at $\mathcal{S}_u$ or to meet an f value, stopping at $\mathcal{S}_f$. Since some clusters will join with others, we recompute effective cluster sizes for all clusters that changed and then move down (up) all clusters with the appropriate effective cluster size for the given λ.

For this version of the algorithm, we are still stepping in an α-descent direction to points in $\mathcal{S}_u$ and/or $\mathcal{S}_f$. We are not breaking up clusters as before, but we are now stepping through effective cluster sizes to make the algorithm more efficient. As long as the effective cluster size does not decrease, we know that the convergence given in Section 4 still holds. In fact, we can easily show that effective cluster size does not decrease.

Lemma 5.1. *The effective cluster size (ECS) for any cluster at any iteration given by*

$$Q_{up} = q_g - q_\ell + q_e \quad \text{or} \quad Q_{dwn} = -q_g + q_\ell + q_e, \tag{64}$$

will never decrease, here Q_{up} is the ECS for a cluster intended to move up and Q_{dwn} is the ECS for a cluster intended to move down.

Proof. We will prove this lemma is true for an up cluster C_i. The argument for a down cluster is similar. We recall Definition 4.5.

For this proof, we will say that u_j in C_i contributes to q_g if $u_j > f_j$, to q_ℓ if $u_j < f_j$, and to q_e if $u_j = f_j$. Notice the following:

- If u_j in C_i contributes to q_g and C_i moves up to form the new cluster C_i', then u_j' in C_i' contributes to q_g' since $u_j' > u_j > f_j$.
- If u_j in C_i contributes to q_e and C_i moves up to form the new cluster C_i' then u_j' in C_i' contributes to q_g' since $u_j' > u_j = f_j$.
- If u_j in C_i contributes to q_ℓ and C_i moves up to form the new cluster C_i', then u_j' in C_i' contributes to either q_e' or q_ℓ' since a cluster will stop at the closest of its neighbors or corresponding f values. Thus $f_j \geq u_j' > u_j$.

This all tells us that when C_i moves up, q_ℓ can only change by decreasing, q_g can only change by increasing, and q_e can change by either increasing or decreasing.

Now, we consider the effective cluster size for three cases for the newly formed cluster C_i'.

A $|C_i'| = |C_i|$, the actual cluster size does not change. This happens when C_i moves up to meet an f value,

B C_i' is lower than both of its neighbors, and

C C_i' is a cluster that is higher than both of its neighbors.

We don't consider the case when one of the neighbors of C_i' is below and the other above the cluster, since moving this cluster will not give descent in G_1.

In case A, since both neighboring clusters, C_{i-1} and C_{i+1} are still above C_i, the effective cluster size is given by Q_{up} in (64). Using the argument above we see that the new effective cluster size increases since at least one u_j in C_i that contributes to q_ℓ will move up to u_j' that contributes to q_e thus Q_{up} will increase.

In case B, C_i will move up to join with at least one of its neighboring clusters C_{i-1} and C_{i+1}. Now, let Q_{i-1}, Q_{i+1} denote the effective cluster sizes of C_{i-1}, C_{i+1}, respectively and Q_i' be the contribution from C_i after its move. From the above argument, we know that $Q_i' = q_g' - q_\ell' + q_e' \geq q_g - q_\ell + q_e = Q_i$.

If C_i moves up to join with C_{i-1}, the new effective cluster size is just the sum of the contributions from both clusters, that is, $Q = Q_{i-1} + Q_i'$. If C_i moves up to join with C_{i+1}, the new effective cluster size is $Q = Q_i' + Q_{i+1}$. And if C_i moves up to join with both C_{i-1} and C_{i+1}, the new effective cluster size is $Q = Q_{i-1} + Q_i' + Q_{i+1}$. In these three cases, if $Q_{i-1}, Q_{i+1} > 0$ then the effective cluster size does not decrease.

Notice that case C can only happen if C_i moves up to meet both of its neighbors (for otherwise, at least one will still be above C_i'). Thus, the new effective cluster size is $Q = Q_{i-1} + Q_i' + Q_{i+1}$. Also, for this case,

the new cluster that is formed is a down cluster, that is, Q is computed using Q_{dwn} in (64). Since C_i was below clusters C_{i-1} and C_{i+1} before the move, we know that C_{i-1} and C_{i+1} were down clusters before C_i moved up. Since we are incrementing on the ECS, we know that $Q_{i-1}, Q_{i+1} \geq Q_i$. Since the newly formed cluster, C_i' is a down cluster, the amount that C_i contributes to Q is $Q_i' = -q_g + q_\ell + q_e$. Notice that Q_i' is not an effective cluster size, rather it only contributes to the new effective cluster size therefore it may be negative. If Q_i' is negative, then we will get the smallest value for Q. But the smallest this can be happens when $u_j = f_j$ for all u_j in C_i so that after the move they contributed to q_g, but then $Q_i = |C_j|$ and we get $Q = Q_{i-1} + Q_i' + Q_{i+1} = Q_{i-1} - |C_i| + Q_{i+1} \geq Q_i$. And, we know that, in this case also, the effective cluster size never decreases.

Notice that since the algorithm starts with $u = f$, $Q_i = |C_i| > 0$ for every cluster C_i thus using the above arguments, the minimum effective cluster sizes never decrease. □

Now we give the formal algorithm. Let $C_1^{(k)}, \ldots, C_{q_k}^{(k)}$ be the unique clusters at iteration k. Let $g_i^{(k)}, e_i^{(k)}, \ell_i^{(k)}$ be q_g, q_e, and q_ℓ for cluster i at iteration k.

Algorithm 5.1. (L^1TV)
Given $f = (f_1, \ldots, f_m)$;
Set $u^{(0)} = (u_1^{(0)}, \ldots, u_m^{(0)}) = (f_1, \ldots f_m)$;
Find $C_1^{(0)}, C_2^{(0)}, \ldots, C_{q_0}^{(0)}$;
Set $k \leftarrow 1$;
do
  Compute $g_i^{(k)}, e_i^{(k)}, \ell_i^{(k)}$ for each $i = 1...q_0$;
  $U \leftarrow \{j$: all neighbors of C_j are above $C_j\}$
  $D \leftarrow \{j$: all neighbors of C_j are below $C_j\}$
  $\text{mincs}_k \leftarrow \min_{1 \leq i \leq q_k}\{\min_{i \in U}\{g_i^{(k)} - \ell_i^{(k)} + e_i^{(k)}\}, \min_{i \in D}\{-g_i^{(k)} + \ell_i^{(k)} + e_i^{(k)}\}\}$;
  $\text{mvcl} \leftarrow \{i \in U : g_i^{(k)} - \ell_i^{(k)} + e_i^{(k)} = \text{mincs}\}$;
  if mvcl $\neq \emptyset$
    for $idx = 1 : |\text{mvcl}|$
      Move up, $C_{\text{mvcl}(idx)}$ to closest of f, $C_{I(idx)-1}$, and $C_{I(idx)+1}$
    end
  else
    $\text{mvcl} \leftarrow \{i \in D : -g_i^{(k)} + \ell_i^{(k)} + e_i^{(k)} = \text{mincs}\}$;
    for $idx = 1 : |\text{mvcl}|$
      Move down, $C_{\text{mvcl}(idx)}$ to closest of f, $C_{I(idx)-1}$, and $C_{I(idx)+1}$
    end
  end
  $k \leftarrow k + 1$
  Update list of clusters, $[C_1^{(k)}, \ldots, C_{q_k}^{(k)}]$;
  if $\text{mincs}_k \neq \text{mincs}_{k-1}$
    Append list of solutions with $[C_1, \ldots, C_{q_k}]$;
    Append list of λ with $\frac{2}{mincs+1}$;
  end
until no descent exists.

TABLE 3. Efficient L^1TV algorithm (written with an up preference)

In this algorithm we start at $u^{(0)} = f$ and find the clusters. We then determine which clusters might move up (call them up clusters) and which might move down (call them down clusters) ignoring those that have both a neighbor below and a neighbor above the cluster. In the case of Algorithm 5.1, we see it is written

with a preference to move clusters up first and then down. We find the minimum effective cluster size (ECS) and move up any up cluster, having this ECS, to its nearest neighboring cluster or f value. If no up cluster has this ECS, we move down any down cluster that has the same ECS to its nearest neighboring cluster or f value. We repeat this until the stopping condition is reached. If at any iteration the minimum effective cluster size changed from the previous iteration, we update the list of solutions and the λ value.

The stopping condition for this algorithm depends on the type of data and the boundary conditions. In the general case for data (that is, not necessarily binary data), we stop the algorithm when monotonicity is reached for fixed boundary data or, for free boundary data, when there is only one cluster left (the solution is flat).

For binary data, this algorithm is greatly simplified. There is never a need to check neighbors of a cluster. If the data is binary, then the neighbors have the opposite value of that of the cluster. That is, if the cluster is at a height of 1, then its neighbors are at a height of 0 and it is then a down cluster. For down clusters, C_i, the effective cluster size is dependent on ℓ_i and e_i whereas for an up cluster, the effective cluster size is dependent on g_i and e_i. Another simplification for this algorithm when the data is binary is that we never need to check the distance to f values corresponding to a cluster since these will also be either 0 or 1. Thus, the algorithm will not have moves to heights other than the height of cluster neighbors. Finally, the algorithm will stop when the minimum number of clusters, minq, is reached. The value of minq depends on whether the boundaries are fixed (minq= 2) or free (minq= 1).

The greatest benefit to Algorithm 5.1, for both the general and the binary problems, is that we are able to solve the $\lambda = 0$ problem and in the process get the solutions $\forall \lambda > 0$. That is, the computational task of getting a solution for all $\lambda > 0$, is the same as solving only one problem. We state this result in the next theorem.

Theorem 5.1. *Algorithm 5.1 finds a solution to L^1TV for every $\lambda > 0$.*

Proof. Algorithm 5.1 iterates by increasing the effective clusters size, thus decreasing λ beginning with the largest possible λ so that at least one cluster will move. Because the problem is discrete, the effective cluster sizes are positive integers between 0 and m (the length of the signal). The effective cluster size (and thus, λ) does not change until no cluster of this effective cluster size will move. By the results of Section 4 we know that for each λ, Algorithm 5.1 finds descent whenever descent exists. Therefore at each iteration the algorithm minimize L^1TV for the current λ. And by Lemma 5.1, we know that iterating on the effective cluster size does not skip a particular effective cluster size that might need to be revisited later since the effective cluster size never decreases. Thus, for each $\lambda > 0$, we find a minimizer to the corresponding L^1TV problem. □

5.1. **Extensions of the ht Algorithm.** It is worth noting that the algorithm will not naturally extend to higher dimensional data. The issue lies in the neighborhood structure that occurs at higher dimensions. In higher dimensional problems, such as imaging problems, it becomes beneficial to break up clusters when there is a data point that has more neighbors outside of the cluster than inside. We see this occurring in images with L^1TV when parts of object edges having high curvature are rounded. We conjecture that an adjustment to the algorithm to allow cluster break up only when the number of neighbors outside the cluster is not less than the number inside will give similar results for higher dimensional data.

The ht algorithm does not extend easily to an L^1TV objective function where the data fidelity term involves a linear operator such as in the discretization of $\|K * u - f\|_1$. This transformation causes a rotation of some of the bounding hyperplanes, changing the geometry of the problem. Because the ht algorithm relies strongly on this geometry, we believe it is not obvious , yet worth future investigations, to extend the ht algorithm to such cases.

6. Time Trials for L^1TV ht Algorithm

Finally, we show timing results for both the general and binary cases as well as for fixed and free boundary conditions.

First, we start with fixed boundary conditions. We ran Algorithm 5.1 on 100 random signals of size N. In Table 4 we have recorded the average number of initial clusters, the average number of λ solutions, and the average time in seconds that it takes to perform the main loop of the ht algorithm. The algorithm has a set up that is of order N, but then the main loop depends on the number of initial clusters.

N	Ave. # of initial clusters	Ave. # of λ Solutions	Ave. time in seconds
5	5	2.61	0.0008
10	10	3.65	0.0015
20	20	5.93	0.0031
40	40	9.21	0.0061
80	80	12.62	0.0114
160	160	23.58	0.0229
320	320	39.27	0.0455
640	640	63.76	0.0941
1280	1280	135.61	0.2153
2560	2560	283.39	0.5410
5120	5120	418.02	1.4511

TABLE 4. Time Trials for Algorithm 5.1 for general random signals, of size N, with fixed boundary conditions.

We then ran 100 random binary signals of length N. We recorded the average time to complete the main loop of the ht algorithm, the average number of initial clusters, and average number of λ solutions in Table 5.

N	Ave. # of initial clusters	Ave. # of λ Solutions	Ave. time in seconds
5	2.1	1.39	0.0001
10	3.9	2	0.0002
20	7.4	2.51	0.0004
40	13.8	2.93	0.0007
80	26.9	3.47	0.0012
160	52.2	3.93	0.0020
320	104.4	4.43	0.0035
640	209.1	4.91	0.0067
1280	419.1	5.34	0.0138
2560	836.2	5.91	0.0305
5120	1677.1	6.42	0.0762

TABLE 5. Time Trials for Algorithm 5.1 for random binary signals, of size N, with fixed boundary conditions.

Next, we looked at some time trials, but this time with free boundary conditions. We ran 100 random signals of length N. In Tables 6 (general random signals) and 7 (binary random signals), we recorded the average time to complete the main loop, the average number of initial clusters, and the average number of λ solutions.

We know that for a random binary signal, this algorithm is $O(M)$. From our time trials, it appears that the computational complexity of the main loop of our ht algorithm is $O(M)$, where M is the number of initial clusters in our signal. An initial computation of $O(N)$ is performed on each signal to catalogue the clusters, where N is the length of the signal. Putting these together, we believe that the computational complexity of this algorithm is $O(aN + M)$ signals of length N, where a is small compared to 1. We know that for a binary

N	Ave. # of initial clusters	Ave. # of λ Solutions	Ave. time in seconds
5	5	3.26	0.0010
10	10	4.28	0.0020
20	20	5.93	0.0035
40	40	8.11	0.0063
80	80	11.15	0.0117
160	160	15.15	0.0217
320	320	20.58	0.0417
640	640	30.22	0.0857
1280	1280	41.72	0.1874
2560	2560	58.94	0.4587
5120	5120	84.78	1.2875

TABLE 6. Time Trials for Algorithm 5.1 for general random signals, of size N, with free boundary conditions.

N	Ave. # of initial clusters	Ave. # of λ Solutions	Ave. time in seconds
5	2.3	1.85	0.0002
10	4	2.33	0.0003
20	6.7	2.71	0.0005
40	13.8	3.19	0.0008
80	26.8	3.51	0.0012
160	53.8	4.03	0.0020
320	106.7	4.54	0.0036
640	211.4	4.87	0.0068
1280	420.8	5.36	0.0138
2560	835.8	5.85	0.0305
5120	1678.3	6.40	0.07771

TABLE 7. Time Trials for Algorithm 5.1 for random binary signals, of size N, with free boundary conditions.

signal, the cluster that moves will meet up with both its neighbors. Thus, after each iteration, the number of clusters decreases by 2 for every cluster that moves. This means that there are exactly $\frac{M}{2}$ cluster moves for any binary signal. The worst case scenario, is the binary signal given by

$$u^{(0)} = (0, 1, 0, 1, 0, 1, \ldots).$$

Here the number of initial clusters is N, the signal length. Thus, the number of cluster moves is equal to $\frac{M}{2} = \frac{N}{2}$. In a random binary signal, we expect less than N initial clusters.

7. Example for L^1TV ht Algorithm

As we mention above, L^1TV minimization can be used to find scales in data. In this section we show that our algorithm for L^1TV does indeed give the expected results. In Figure 11(a), we show a plot of daily sunspot numbers obtained from NASA (http://solarscience.msfc.nasa.gov). The sunspot number[1] for any

[1]The sunspot number is commonly refered to as the Wolf Number in honor of Rudolf Wolf who is credited with the concept in 1848.

given day is a standardized measure of the number of sunspots and sunspot groups present on the earth-facing surface of the sun. Sunspots are dynamic and have typical lifetimes of a few days to a few months. However, the dominant feature in the signature is an approximate 11-year period in overall sunspot activity.

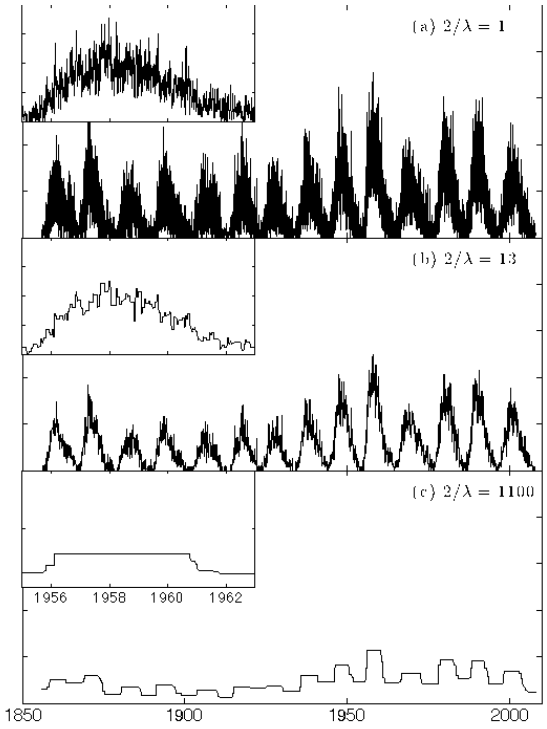

FIGURE 11. Examples of minimizing signals for daily sunspot numbers at significant scales. The raw daily data is represented at scale $2/\lambda = 1$.

We applied the ht algorithm for L^1TV to this sunspot data. The signature shown in Figure 12 is the value of the variation term in (1) ($\sum_{i=0}^{m} |u^*_{i+1} - u^*_i|$) for each distinct value of $2/\lambda$ found by the ht algorithm. The presence of a scale $2/\lambda$ in a signature is revealed by significant changes in variation with respect to $2/\lambda$. Three scale signatures are present, at $2/\lambda \approx 13, 1200, 11000$, in units of days. The largest value indicates the time scale over which sunspot activity rises and falls over decades. The medium scale represents the typical duration of the decadal peak in sunspot activity, about 3-4 years. The 13-day scale correlates well with the typical duration of a sunspot on the face of the sun, limited by half of the the sun's generally accepted synodic rotational period of 26 days. The short scale does appear to be composed of a broad range of scales ranging from one to six weeks. Scales shorter than two weeks may be due to observations of the beginning and end of sunspot life cycles. Scales of several weeks may be due to the duration of sun-wide bursts of activity.

Minimizing signals for these two scales are shown in Figures 11(b) and 11(c). The inset subfigures show the signals over a short range of dates from 1955 through 1963. At these scales, features in the data of width are significant and changing rapidly with respect to λ. Features of effective cluster size (see Section 5) smaller than the given scale are not present.

8. Hyperplane Traversal Algorithm for discrete L^1pTV for $p < 1$

In this section, we consider the discrete formulation for L^1pTV for $0 < p < 1$

$$\min \left\{ G_p \equiv \sum_{i=0}^{m-1} |u_{i+1} - u_i|^p + \lambda \sum_{i=0}^{m} |f_i - u_i| \;\middle|\; u \in \mathbb{R}^{m+1} \right\}. \tag{65}$$

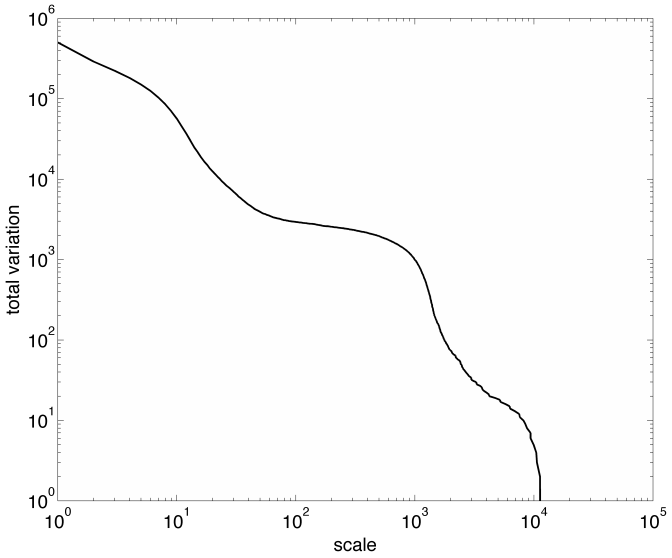

FIGURE 12. Scale Signature for sunspot data

Notice that if u is binary data, the problem is p-independent. Thus, for binary data, this problem reduces to the L^1TV problem.

For this problem, we seek a minimizer in the set Y (Definition 2.2). Because G_p is concave in the regions separated by the hyperplanes of the form $\{u_i = u_{i+1}\}, \{u_i = u_{i-1}\}$, or $\{u_i = f_i\}$ (see Figure 13), we know that minimizers will be in Y. We use a variant of the ht algorithm which stays in Y to find local minimizers. The difference in the algorithm is that the choices made depends on whether neighboring clusters are both above, both below, or one above and one below the cluster being moved.

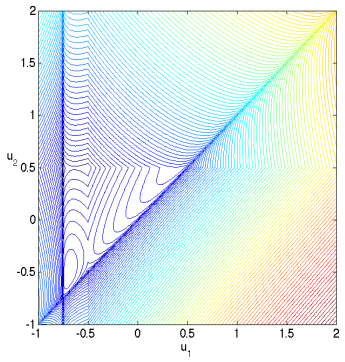

FIGURE 13. Level lines of a simple example of G_p for $p = .5$ and $\lambda = 1$.

8.1. **ht Algorithm for L^1pTV.** Again, we use the proof of Lemmas 4.9 and 4.10 to write an algorithm to find minimizers of (1). We let $C_1^{(k)}, \ldots, C_{q_k}^{(k)}$ be the unique clusters at iteration k. Using the lemmas, we write the conditions on λ for a cluster to move at iteration k. We let q_g, q_e, and q_ℓ be as we defined them in Section 5. To move a cluster that has one neighbor above and one neighbor below, we let η_a be the distance from the cluster to the neighbor above the cluster and η_b be the distance from the cluster to the neighbor below the cluster. And we consider moving the cluster up a distance $\eta = \min\{\eta_a, \{f_i - u_i : \text{ when } f_i > u_i, u_i \in C_i\}\}$.

The algorithm will find descent in this case if

$$0 < \lambda < \frac{\eta_a^p - (\eta_a - \eta)^p + \eta_b^p - (\eta_b + \eta)^p}{\eta(q_g - q_\ell + q_e)}. \tag{66}$$

Notice that since $q_g - q_\ell + q_e > 0$, we need

$$0 < \eta_a^p - (\eta_a - \eta)^p + \eta_b^p - (\eta_b + \eta)^p. \tag{67}$$

This happens only when $\eta_a < \eta_b$. Now we consider moving the cluster up a distance $\eta = \min\{\eta_b, \{u_i - f_i : \text{ when } f_i < u_i, u_i \in C_i\}\}$. The algorithm will find descent in this case if

$$0 < \lambda < \frac{\eta_a^p - (\eta_a + \eta)^p + \eta_b^p - (\eta_b - \eta)^p}{\eta(-q_g + q_\ell + q_e)}. \tag{68}$$

Similar to the previous case, we need that $\eta_b < \eta_a$ for this to make sense. To move a cluster that has both neighbors above, we let η_ℓ be the distance from the cluster to its left neighbor and η_r be the distance from the cluster to its right neighbor. And we consider moving the cluster up a distance $\eta = \min\{\eta_\ell, \eta_r, \{f_i - u_i : \text{ when } f_i > u_i, u_i \in C_i\}\}$. The algorithm will find descent in this case if

$$0 < \lambda < \frac{\eta_r^p - (\eta_r - \eta)^p + \eta_\ell^p - (\eta_\ell - \eta)^p}{\eta(q_g - q_\ell + q_e)}. \tag{69}$$

Finally, To move a cluster that has both neighbors below, we let η_ℓ be the distance from the cluster to its left neighbor and η_r be the distance from the cluster to its right neighbor. And we consider moving the cluster up a distance $\eta = \min\{\eta_\ell, \eta_r, \{u_i - f_i : \text{ when } f_i < u_i, u_i \in C_i\}\}$. The algorithm will find descent in this case if

$$0 < \lambda < \frac{\eta_r^p - (\eta_r + \eta)^p + \eta_\ell^p - (\eta_\ell + \eta)^p}{\eta(-q_g + q_\ell + q_e)}. \tag{70}$$

Using the above information, we see that our algorithm should only check the conditions for moving a cluster up when the closest neighbor is above the cluster. Similarly, we should only check the condition for moving a cluster down when the closest neighbor is below the cluster. As in Algorithm 5.1, we let g_i, e_i, ℓ_i be q_g, q_e, q_ℓ for cluster i. In the algorithm below, we let $Q_{up} = \{g_i - \ell_i + e_i : 1 \le i \le m\}$ and $Q_{dwn} = \{-g_i + \ell_i + e_i : 1 \le i \le m\}$.

In words, this algorithm starts at $u^{(0)} = f$ and finds all of the clusters. The cluster with the maximum λ_{cut} (according to Equations (66), (68), (69), and (70)) is moved in the appropriate direction to the nearest neighbor or nearest f_i for $i \in \{i : u_i \in C_i\}$. We continue this until stopping conditions are reached. As in the L^1TV algorithm, stopping conditions depend upon the type of boundary conditions, that is, whether they are fixed or free boundary conditions.

Notice that since the iterations of Algorithm 8.1 stay in Y (see Definition 2.2), we know that it is also a finite ht algorithm. From Lemmas 4.9 and 4.10 we know that at each iteration, the algorithm finds strict decent. $G_p \ge 0$ and is therefore bounded below. We can see that G_p is coercive. Indeed, we have that $\sum_{i=0}^m |u_{i+1} - u_i|^p \ge 0$ and

$$\lambda \sum_{i=1}^m |f_i - u_i| \le \lambda \sum_{i=1}^m (|u_i| + |f_i|) \to \infty \text{ as } |u| \to \infty. \tag{71}$$

We do not have a convexity condition therefore we are only able to conclude that this algorithm finds local minima.

Like Algorithm 3.1, Algorithm 8.1 steps to the hyperplanes where G_p is nonsmooth and stays in the lower dimensional space. Therefore, this algorithm finds minimizers of lower and lower dimensional problems. The clusters also only get larger, therefore because we operate on clusters rather than the signal, the algorithm increases in efficiency as we progress through the iterations.

Also for Algorithm 8.1, we don't get a local minimizer at the end of each iteration. This happens because the effective cluster size can increase or decrease. The iterations that end with minimizers are those for

Algorithm 8.1. (L^1pTV)
Given $f = (f_1, \ldots, f_m)$;
Set $u^{(0)} = (u_1^{(0)}, \ldots, u_m^{(0)}) = (f_1, \ldots f_m)$;
Find $C_1^{(0)}, C_2^{(0)}, \ldots, C_{q_0}^{(0)}$;
Compute Set $k \leftarrow 1$;
for $i = 1 : m$
    Compute λ_i according to Eqs. (66), (68), (69), and (70));
end
do
    Find $mvcl = \arg\max_i\{\lambda_i\}$;
    Set $\lambda_k^* \leftarrow \lambda_{mvcl}$;
    if $\lambda_k^* < 0$, stop **do** loop
    Set $\eta_r \leftarrow |u_r - u|, \eta_\ell \leftarrow |u_\ell - u|$;
    $\eta \leftarrow \min\left\{\eta_r, \eta_\ell, \min_{f_j, u_j \in C_{mvcl}}\left\{|f_j - u_j| \,\middle|\, \alpha(f_j - u_j) > 0\right\}\right\}$;
    Compute α_{mvcl} according to 8.2 using $u_r, u_\ell, Q_{up_i}, Q_{dwn_i}, \eta_r, \eta_\ell$;
    Move C_{mvcl} the distance $\eta\alpha_{mvcl}$
    $k \leftarrow k + 1$
    Update list of clusters, $[C_1^{(k)}, \ldots, C_{q_k}^{(k)}]$;
    Compute λ_i for clusters that change;
    if $\lambda_k^* < \lambda_{k-1}^*$
        Append list of solutions with $[C_1, \ldots, C_{q_k}]$;
        Append list of λ^* with λ_k^*;
    end
until no descent exists.

TABLE 8. L^1pTV algorithm

which λ decreases in the next iteration. Notice if λ increases in the next iteration, we are still finding descent for the current λ_{cut} value. For example, if $\lambda_{cut}^{(k)} = 10$ we know that at iteration k, we find descent whenever $\lambda < 10$. If the maximum λ that will give descent at iteration $k + 1$ is larger than 10, then we know that we will find descent for $\lambda < 10$ since $\lambda_{cut}^{(k+1)} > 10$. Thus, we get a complete set of local minimizers.

9. SUMMARY

In this paper we presented a novel way of understanding and solving the one-dimensional discrete L^1pTV problem (1). This coercive function is piecewise concave with region boundaries defined by hyperplanes. We show that optimal solutions for any $\lambda \geq 0$ can be found on the finite set of discrete points in $\mathbb{R}^{m+1}$ defined by intersections of $m + 1$ or more distinct hyperplanes. We then provided a hyperplane traversal (ht) algorithm which systematically searches these points. The ht algorithm finds exact optimal solutions for all $\lambda \geq 0$ in finite iterations. For $p = 1$ the solutions are globally optimal, for $0 < p < 1$ solutions are only guaranteed to be locally optimal. Of particular interest is that ht never needs to compute the objective function $G_\lambda(u)$ in order to obtain the complete λ-parametric solution set.

The ht algorithm is a form of parametric programming, but is extremely efficient in terms of computational cost and memory usage. Time trials on randomly-generated test signals indicate that the computational complexity on binary data is $O(M)$, where $M \leq m + 1$ is the initial number of clusters (distinct groupings of either 0 or 1). General non-binary data complexity appears to be $O(am + M)$, where $a << 1$.

One use of the L^1pTV function is the discovery of scale information in data. We illustrated this ability by examining the record of daily sunspot numbers for approximately 152 years of data. Significant scale

Algorithm 8.2. (α)
Given $u, u_r, u_\ell, Q_{up}, Q_{dwn}, \eta_r, \eta_\ell$;
if $u_r, u_\ell > u$
    $\alpha \leftarrow 1$;
elseif $u_r > u$ and $\eta_r < \eta_\ell$
    $\alpha \leftarrow 1$;
elseif $u_\ell > u$ and $\eta_\ell < \eta_r$
    $\alpha \leftarrow 1$;
elseif $\eta_\ell = \eta_r$
    if $u_\ell > u$ or $u_r > r$
        if $0 \le Q_{up} < Q_{dwn}$ or $Q_{dwn} < 0 \le Q_{up}$
            $\alpha \leftarrow 1$;
        elseif $Q_{up} = Q_{dwn}$
            $\alpha \leftarrow$ preferred direction (up=1,down=-1);
        end
    end
else
    $\alpha \leftarrow -1$;
end

TABLE 9. L^1pTV algorithm

signatures are present at approximately 13 days, at 3-4 years and at 30 years. These scales are explained in terms of the known dynamics of the sun and of sunspot activity.

We briefly discussed extensions to higher dimensional data, such as images, and to signals modified by linear operators, such as blurring functions. It is not yet clear which types of extensions can be solved using an ht algorithm because it depends strongly on the geometry of the objective function.

We would like to acknowledge Jamie O'Brien for finding a mistake in our algorithm description. We would also like to thank the reviewers for taking the time to give valuable feedback on our original manuscript.

REFERENCES

[1] S. Alliney. A Property of the Minimum Vectors of a Regularizing Functional Defined by Means of the Absolute Norm. *IEEE Trans. Signal Process.*, 45:913–917, 1997.
[2] T. Chan and Esedoḡlu. Aspects of Total Variation Regularized L^1 Function Approximation. *SIAM J. Appl. Math.*, 65(5):1817–1837, 2005.
[3] Rick Chartrand. Nonconvex regularization for shape preservation. In *IEEE International Conference on Image Processing (ICIP)*, 2007.
[4] Francis H Clarke, Yuri S Ledyaev, Ronald J Stern, and Peter R Wolenski. *Nonsmooth analysis and control theory*, volume 178. Springer, 1997.
[5] B. Dacorogna. *Introduction to the Calculus of Variations*. Imperial College Press, London, 2004.
[6] B. Dacorogna. *Direct Methods in the Calculus of Variations, Second ed.* Springer, 2008.
[7] L. C. Evans. *Partial Differential Equations*. American Mathematical Society, Providence, 2002.
[8] H. Fu, M. K. Ng, M. Nikolova, and J. L. Barlow. Efficient minimization methods of mixed $\ell 2$-$\ell 1$ and $\ell 1$-$\ell 1$ norms for image restoration. *SIAM Journal on Scientific Computing*, 27(6), 2006.
[9] D. Goldfarb and W. Yin. Parametric maximum flow algorithms for fast total variation minimization. *SIAM Journal on Scientific Computing*, 31(5), 2009.
[10] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2), 2004.

[11] M Nikolova. Minimizers of cost-functions involving nonsmooth data-fidelity terms. *SIAM Journal on Numerical Analysis*, 40:965–994, 2003.
[12] L. Rudin, S. Osher, and E. Fatemi. Nonlinear Total Variation Based Noise Removal Algorithms. *Physica D*, 60(1-4):259–268, November 1992.
[13] D. Strong and T. Chan. Edge-Preserving and Scale-Dependent Properties of Total Variation Regularization. *Inverse Problems*, 19, 2003.
[14] A. N. Tikhonov and V. Y. Arsenin. *Solutions of ill-posed problems*. Winston, 1977.

E-mail address: hamoon@smcm.edu

E-mail address: tasaki@wsu.edu

Department of Mathematics, Washington State University, Pullman, WA 99164-3113