PARAMETRIC MAXIMUM FLOW ALGORITHMS FOR FAST TOTAL VARIATION MINIMIZATION

DONALD GOLDFARB* AND WOTAO YIN†

Abstract. This report studies the global minimization of anisotropically discretized total variation (TV) energies with an \( L^p \) (in particular, \( L^1 \) and \( L^2 \)) fidelity term using parametric maximum flow algorithms to minimize \( s-t \) cut representations of these energies. The TV/\( L^2 \) model, also known as the Rudin-Osher-Fatemi (ROF) model, is suitable for restoring images contaminated by Gaussian noise, while the TV/\( L^1 \) model is able to remove impulsive noise from grey-scale images and perform multi-scale decompositions of them. Preliminary numerical results on large-scale two-dimensional CT and three-dimensional Brain MR images are presented to illustrate the effectiveness of these approaches.

keywords. maximum flow, minimum cut, graph cut, parametric maximum flow, total variation, image denoising, MRI.

1. Introduction. Let a grey-scale 2-dimensional image be represented by a function \( f \) on a domain \( \Omega \) in \( \mathbb{R}^2 \). In this paper, to simplify our presentation, we restrict our discussion to rectangular domains. The ROF (TV/\( L^2 \)) [41] and TV/\( L^1 \) models [2, 34, 12, 51] obtain a decomposition of \( f \), \( f = u_\lambda^* + v_\lambda^* \), by solving the following models, respectively,

\[
\begin{align*}
\text{ROF:} & \quad \inf_{u \in BV} TV(u) + \lambda \| f - u \|^2_{L^2}, \\
\text{TV/}\!L^1: & \quad \inf_{u \in BV} TV(u) + \lambda \| f - u \|_{L^1},
\end{align*}
\]

for their minimizers \( u_\lambda^* \), where \( BV \) is the space of functions of bounded variation, \( TV(u) \) is the total variation [53] of \( u \), and \( f \in L^1(\Omega) \). The latter assumption is needed for technical reasons. The ROF model (1.1) was proposed in [41] for obtaining a restored image \( u^* \) from an input image \( f \) corrupted by Gaussian noise. Early models for doing this were based on least squares and had the unfortunate property of either smoothing edges or creating spurious oscillations near edges, i.e., the well-known ringing phenomenon. Minimizing \( TV(u) \) allows \( u \) to have discontinuities; hence edges and important features in the original image are preserved by the ROF approach. The TV/\( L^1 \) model differs from the ROF model by using an \( L^1 \) instead of an \( L^2 \) data fidelity term. Previous work on the TV/\( L^1 \) model for image/signal processing include Alliney’s pioneering study [2, 3, 4] of the discrete version of (1.2), Nikolova’s [34, 35, 36] discovery of its usefulness for removing impulsive noise, Chan and Esedoglu’s [12] further analysis of it with comparisons to the ROF model, Yin, Goldfarb, and Osher’s [51] study of its multi-scale decomposition properties based on [12], and applications of it in computer vision by Chen et al. [16, 14, 15] and in biomedical imaging by Yin et al. [50].

There are various numerical methods for solving the ROF and TV/\( L^1 \) models. The most basic one is the time-marching PDE method based on the Euler-Lagrange equations for (1.1) and (1.2), which takes small gradient-descent steps with respect to the objective functions of (1.1) and (1.2). Other methods include Chambolle’s dual algorithm [10], Chan, Zhou, and Chan’s primal-dual interior-point method [13], Vogel’s multigrid method [43], Vogel and Oman’s iterative method [44, 45], Ito and Kunish’s active set method [31], Li and Santosa’s affine scaling method [33], Wohlberg and Rodriguez’s iteratively reweighted norm method

*Department of Industrial Engineering and Operations Research, Columbia University, goldfarb@columbia.edu. Research supported in part by NSF Grant DMS 06-06712, ONR Grants N00014-03-0514 and N00014-08-1-1118 and DOE Grants DE-FG01-92ER-25126 and DE-FG02-08ER25856.

†Department of Computational and Applied Mathematics, Rice University, wotao.yin@rice.edu. Research supported by NSF CAREER Award DMS-0748839 and ONR Grant N00014-08-1-1101.
[47], and Goldfarb and Yin’s second-order cone programming method [27]. Some of these methods can also be applied to deconvolution problems in which \( f - u \) is replaced by \( f - Au \) in the ROF and TV/L^1 model for certain convolution operators \( A \). Recently, fast methods for these problems have been proposed by Fu, Ng, Nikolova, and Barlow [24] and Wang, Yang, Yin, and Zhang [46, 48, 49]. The former uses an anisotropic discretization of TV combined with a linear programming interior-point method while the latter uses variable splitting and parameter continuation.

Our parametric maximum flow based methods were developed by combining our recent work [51] with the graph-cut algorithms in [32]. However, we soon became aware of the work of Darbon and Sigelle [22], which is also based on the graph-cut approach. Later, an anonymous referee brought to our attention the work of Chambolle [11] on a Markov Random Field based method. The authors of these two papers independently proposed essentially identical algorithms. Our algorithms are based on the same graph construction but use a different algorithm for finding the minimum s-t cut in the graphs by solving a sequence of max-flow/min-cut problems in one pass. This max-flow/min-cut algorithm is due to Gallo, Grigoriadis, and Tarjan [25] and has the best complexity bound. While writing this paper we learned that Hochbaum [30] had previously proposed this approach and given its complexity. Our contributions include two different recipes for constructing graphs, an existing graphical approach and a more formal one based on optimization, several modifications to the original parametric max-flow/min-cut algorithm in [25] to improve its efficiency, as well as a mixed algorithm for solving the ROF and TV/L^1 models more efficiently by combining the Gallo-Grigoriadis-Tarjan algorithm with the divide-and-conquer approach proposed in [22].

Applying network flow algorithms for finding minimizers of energies/functions is not new. Some of the earliest contributions were made by Balinski [5], Picard and Ratliff [38], and Picard and Queyranne [37]. Subsequent work on imaging include [28], [39], [30], [52], and [40], as well as a large number of papers in computer vision.

The rest of the paper is organized as follows. In Section 2, we provide discrete formulations for problems (1.1) and (1.2). In particular, we present various anisotropic discretizations of the total variation functional that allow us to decompose problems (1.1) and (1.2) into a series of decoupled problems in binary variables. In Section 3 we show how to construct capacitated networks whose s-t cuts correspond one-to-one to the binary variables for these decoupled problems. In Section 4 we give a formal justification of the term-by-term network construction in Section 3 based on the linear programming formulation of the min s-t cut problem, and point out that a min-cut can be computed by finding the corresponding s-t max-flow in the network. In Section 5, we present the preflow-flow algorithm of Goldberg and Tarjan [26] for solving a single max-flow/min-cut problem, and its extension [25] for solving a sequence of related max-flow/min-cut problems. In this section we also describe modifications to these algorithms that improve their efficiency in solving the max-flow/min-cut problems associated with the ROF and TV/L^1 models. To illustrate the effectiveness of the proposed algorithms, various numerical results on two-dimensional and three dimensional images are presented in Section 6. We also discuss the dependence of the running times of our algorithms on various input parameters and the memory requirements of the algorithms. Finally, we present some conclusions in Section 7.

2. Level Set Problem Formulations. The two-dimensional images in this paper are defined as matrices in \( \mathbb{Z}_+^{m \times n} \), where \( \mathbb{Z}_+ \) denotes the set of nonnegative integers that represent the grey-scale levels of images. The \((i, j)\)-th element of these matrices represents the grey-scale level or brightness of the \((i, j)\)-th image pixel. Similarly, three-dimensional images are represented by three-dimensional arrays in \( \mathbb{Z}_+^{m \times n \times p} \), in which the \((i, j, k)\)-th element represents the grey-scale level of the \((i, j, k)\)-th voxel. For ease of notation,
where we primarily focus on two-dimensional images. Our results for two-dimensional images can be extended to three-dimensional images in an obvious way. Numerical experiments on on both two and three dimensional MRI images are presented in Section 6.

Let \( f \in \mathbb{Z}_+^{n \times n} \) and \( u \in \mathbb{Z}_+^{n \times n} \) denote, respectively, the input and output images, and \( v = f - u \) their difference. Hence, the elements of these matrices satisfy

\[
f_{i,j} = u_{i,j} + v_{i,j} \quad \text{for } i = 1, \ldots, m, \; j = 1, \ldots, n.
\]

Moreover, we assume that all images \( u \) satisfy Neumann conditions on the boundary of the domain \( \Omega \), and hence that

\[
u_{0,j} = u_{1,j} \quad \text{and} \quad u_{m,j} = u_{m+1,j} \quad \text{for} \quad j = 1, \ldots, n,
\]

\[
u_{i,0} = u_{i,1} \quad \text{and} \quad u_{i,n} = u_{i,n+1} \quad \text{for} \quad i = 1, \ldots, m.
\]

Two different discretized representations of TV, isotropic and anisotropic, are used in image processing. The standard isotropic discretization of TV is

\[
TV^{iso}(u) = \sum_{i,j} \sqrt{|u_{i+1,j} - u_{i,j}|^2 + |u_{i,j+1} - u_{i,j}|^2}.
\]

However, unless the elements of \( u \) are binary, \( TV^{iso}(u) \) cannot be expressed in terms of the super-level indicators of \( u \): \( 1_{\{u \geq 1\}} \), for all \( l \in \mathbb{Z}_+ \). \( 1_{\{u \geq 1\}} \) is a matrix with binary elements \( 1_{\{u \geq 1\}(i,j)} \).

Therefore, in this paper, we only consider anisotropic discretizations of TV. These depend on which neighboring pixels are used to represent \( TV(u) \): the most commonly used neighborhoods, involving 4, 8 and 16 neighbors, respectively, are

\[
N^4(i,j) = \{(k,l) : |k - i| + |l - j| = 1\},
\]

\[
N^8(i,j) = N^4(i,j) \cup \{(k,l) : |k - i| = 1, |l - j| = 1\},
\]

\[
N^{16}(i,j) = N^8(i,j) \cup \{(k,l) : |k - i| + |l - j| = 3, |k - i| \# |l - j| \neq 0\}.
\]

and are depicted in Figure 2.1. For example, these neighborhoods of pixel \((3,3)\) are:

\[
N^4(3,3) = \{(2,3), (3,2), (3,4), (4,3)\},
\]

\[
N^8(3,3) = N^4(3,3) \cup \{(2,2), (2,4), (4,2), (4,4)\},
\]

\[
N^{16}(3,3) = N^8(3,3) \cup \{(1,2), (2,1), (1,4), (4,1), (2,5), (5,2), (4,5), (5,4)\}.
\]

Using the definition

\[
n^p_{i,j}(u) = \sum_{(l,k) \in N^p(i,j)} (u_{l,k} - u_{i,j})^+,\]

where \( x^+ = \max\{x,0\} \), we define the following anisotropic discretizations of \( TV(u) \) corresponding to \( N^4 \), \( N^8 \) and \( N^{16} \), respectively:

\[
TV^4(u) := \sum_{i,j} w^{4.4}|u_{i+1,j} - u_{i,j}| + w^{4.4}|u_{i,j+1} - u_{i,j}| = \sum_{i,j} w^{4.4}n^4_{i,j}(u),
\]
Fig. 2.1. A graph representing the 4, 8, and 16-neighborhoods \( \mathcal{N}_4(i,j) \), \( \mathcal{N}_8(i,j) \), and \( \mathcal{N}_{16}(i,j) \) of \((i,j)\). These neighborhoods consist of those nodes connected to \((i,j)\) by the red (thick), blue (intermediate thick), and all directed arcs, respectively.

\[
TV_8(u) := \sum_{i,j} (w_{4,8} n_{4,i,j}(u) + w_{8,8} n_{8,i,j}(u)),
\]

\[
TV_{16}(u) := \sum_{i,j} (w_{4,16} n_{4,i,j}(u) + w_{8,16} n_{8,i,j}(u) + w_{16,16} n_{16,i,j}(u)),
\]

where the scalars \( w^{p,q} \) are weights chosen according to cut metrics described in [7]. Specifically, for edge weights in a planar graph, [7] presents the formula

\[
w_k = \delta^2 \cdot \Delta \phi_k / 2 |e_k|,
\]

for all edges of type \( k \), where \( \delta \) is the grid size (which is unity in this paper), \( \Delta \phi_k \) is the angle that a type-\( k \) edge covers, and \( |e_k| \) is the length of a type-\( k \) edge. For each node, the sum of \( \Delta \phi_k \) for all of its outgoing edges equals \( 2\pi \). For example, in a 4-neighbor graph in which each node \((i,j)\) is connected to \( \mathcal{N}_4(i,j) \), all edges are of the same type, \( \Delta \phi_k = \pi/2 \) and \( |e_k| = 1 \). Hence in (2.1),

\[w_{4,4} = 1 \cdot \pi/2 \cdot 1 = \pi/4.\]

Similarly, in a 8-neighbor graph there are two types of edges out of each pixel \((i,j)\), those to \( \mathcal{N}_4(i,j) \) and those to \( \mathcal{N}_8(i,j) \setminus \mathcal{N}_4(i,j) \). Their weights are given, respectively, by

\[w_{4,8} = \frac{1 \cdot \pi/4}{2 \cdot 1} = \frac{\pi}{8} \quad \text{and} \quad w_{8,8} = \frac{1 \cdot \pi/4}{2 \cdot \sqrt{2}} = \frac{\sqrt{2}\pi}{16}.\]

It is easy to show that \( w_{4,16} = \frac{1}{2} \tan^{-1}(\frac{1}{2}) \), \( w_{8,16} = \frac{\sqrt{2}}{4}(\frac{\pi}{4} - \tan^{-1}(\frac{1}{2})) \), and \( w_{16,16} = \frac{\sqrt{5}}{80} \pi \). Formula (2.4) was derived based upon the Cauchy-Crofton formula, which implies that the length \( L \) of a regular, plane curve equals half of the measure \( m/2 \) (counting multiplicity) of all lines in the plane that intersect the curve. The weights given in (2.4) make the total weights of the graph edges cut by the curve a good approximate to the measure \( m/2 \), and thus to the length \( L \) of the curve.

For three-dimensional images, we define the following neighborhood and corresponding anisotropic dis-
cretization of $TV(u)$:

\begin{equation}
\mathcal{N}^{3D,6}(i, j, k) := \{(i', j', k') : |i' - i| + |j' - j| + |k' - k| = 1\},
\end{equation}

and

\begin{equation}
TV^{3D,6}(u) := w^{3D,6} \sum_{i,j,k} |u_{i+1,j,k} - u_{i,j,k}| + |u_{i,j+1,k} - u_{i,j,k}| + |u_{i,j,k+1} - u_{i,j,k}|
\end{equation}

\begin{equation}
= \sum_{i,j,k} \left( \sum_{\alpha,\beta,\gamma \in \mathcal{N}^{3D,6}(i,j,k)} w^{3D,6}(u_{\alpha,\beta,\gamma} - u_{i,j,k}) \right),
\end{equation}

where the weight $w^{3D,6}$ equals $\frac{\pi}{4}$ according to cut metrics [7].

For each pair of neighboring pixels $(i, j)$ and $(l, k)$, we can express $(u_{i,j} - u_{l,k})^+$ in terms of the elements of $1_{\{u \geq \mu\}}$ over all grey-scale levels $\mu = 0, 1, \ldots, l_{\max}$ as follows:

\begin{equation}
(u_{i,j} - u_{l,k})^+ = \sum_{\mu=0}^{l_{\max}} (1_{\{u_{i,j} \geq \mu\}} - 1_{\{u_{l,k} \geq \mu\}})^+,
\end{equation}

where $l_{\max} = \max_{i,j} \{u_{i,j}\} \leq 2^\rho - 1$ for $\rho$-bit grey-scale images ($\rho$ is typically 8, 16, or 32). Using (2.9) and defining

\begin{equation}
n_{i,j}^{\rho,\mu}(u) = \sum_{(l,k) \in \mathcal{N}(i,j)} (1_{\{u_{l,k} \geq \mu\}} - 1_{\{u_{i,j} \geq \mu\}})^+,
\end{equation}

we have for $p = 4, 8, 16$,

\begin{equation}
TV^p(u) = \sum_{\mu=0}^{l_{\max}} TV^p(1_{\{u \geq \mu\}}),
\end{equation}

where

\begin{equation}
TV^4(1_{\{u \geq \mu\}}) := \sum_{i,j} w^{4,4,4,4,4}(u),
\end{equation}

\begin{equation}
TV^8(1_{\{u \geq \mu\}}) := \sum_{i,j} (w^{4,8,4,8,4,8}(u) + w^{8,8,4,8,8,8}(u)), \text{ and}
\end{equation}

\begin{equation}
TV^{16}(1_{\{u \geq \mu\}}) := \sum_{i,j} (w^{4,16,4,16,4,16}(u) + w^{8,16,8,16,8,16}(u) + w^{16,16,16,16,16,16}(u)).
\end{equation}

Because the neighborhood relation is reflective, for each neighborhood pair of pixels $(i, j)$ and $(k, l)$, the terms involving both these pixels in (2.12)-(2.14) have the form:

\begin{equation}
w \left( (1_{\{u_{i,j} \geq \mu\}} - 1_{\{u_{k,l} \geq \mu\}})^+ + (1_{\{u_{k,l} \geq \mu\}} - 1_{\{u_{i,j} \geq \mu\}})^+ \right) = w |1_{\{u_{k,l} \geq \mu\}} - 1_{\{u_{i,j} \geq \mu\}}|,
\end{equation}

for some weight $w > 0$.

Next we express the $L^1$ and $L^2$ fidelity terms in terms of $1_{\{u \geq \mu\}}$ and $1_{\{f \geq \mu\}}$. Using the facts that for $a, b \in \mathbb{Z}_+$, $1_{\{b < \mu\}} = 1 - 1_{\{b \geq \mu\}}$ and $|a - b| = \sum_{\mu=0}^{\max(a,b)} (1_{\{b < \mu\}}1_{\{a \geq \mu\}} + 1_{\{b \geq \mu\}}1_{\{a < \mu\}})$, we obtain for the
$L^1$ fidelity term,

$$\sum_{i,j} |u_{i,j} - f_{i,j}| = \sum_{\mu=0}^{l_{\max}} \sum_{i,j} ((1 - 1_{\{f_{i,j} \geq \mu\}}) 1_{\{u_{i,j} \geq \mu\}} + 1_{\{f_{i,j} \geq \mu\}} 1_{\{u_{i,j} < \mu\}}).$$

The advantage of having both $1_{\{u_{i,j} \geq \mu\}}$ and $1_{\{u_{i,j} < \mu\}}$ in the above formula will become clear in next section.

It is easy to verify that, for $b \in \mathbb{Z}_+$, $|a - b|^2 = (a + 1)^2 + \sum_{\mu=0}^{a}((a - \mu)^2 - (a - (\mu - 1))^2) = (a + 1)^2 + 2\sum_{\mu=0}^{a}[\mu - \frac{1}{2} - a]$; hence,

$$|u_{i,j} - f_{i,j}|^2 = (f_{i,j} + 1)^2 + \sum_{\mu=0}^{l_{\max}} \sum_{i,j} ((f_{i,j} - \mu)^2 - (f_{i,j} - (\mu - 1))^2) 1_{\{u_{i,j} \geq \mu\}}$$

$$= (f_{i,j} + 1)^2 + 2\sum_{\mu=0}^{l_{\max}} \left(\mu - \frac{1}{2} - f_{i,j}\right) 1_{\{u_{i,j} \geq \mu\}}.$$ 

Therefore, we obtain for the $L^2$ fidelity term [11]

$$\sum_{i,j} |u_{i,j} - f_{i,j}|^2 = \sum_{i,j} (f_{i,j} + 1)^2 + \sum_{\mu=0}^{l_{\max}} 2 \sum_{i,j} \left(\mu - \frac{1}{2} - f_{i,j}\right) 1_{\{u_{i,j} \geq \mu\}}$$

$$= C + \sum_{\mu=0}^{l_{\max}} 2 \sum_{i,j} \left(\mu - \frac{1}{2}\right) 1_{\{u_{i,j} \geq \mu\}} + f_{i,j} 1_{\{u_{i,j} < \mu\}};$$

where $C$ is a constant that does not depend on $u$.

The above discretizations of $TV(u)$ and the data fidelity terms can be generalized to arbitrary discrete intensity levels $l_0 < l_1 < \ldots < l_K$ instead of 0, 1, \ldots, $l_{\max}$. Finally, combining $TV^p(u)$, for a particular choice of $p$ and the data fidelity terms gives the discretizations:

$$E^1(u; \lambda, f) = TV^p(u) + \lambda \sum_{i,j} |u_{i,j} - f_{i,j}|$$

$$= \sum_{\mu=0}^{l_{\max}} \left(TV^{p,\mu}(1_{\{u_{i,j} \geq \mu\}}) + \lambda \sum_{i,j} \left(1 - 1_{\{f_{i,j} \geq \mu\}}\right) 1_{\{u_{i,j} \geq \mu\}} + 1_{\{f_{i,j} \geq \mu\}} 1_{\{u_{i,j} < \mu\}}\right).$$

$$E^2(u; \lambda, f) = TV^p(u) + \lambda \sum_{i,j} |u_{i,j} - f_{i,j}|^2$$

$$= \sum_{\mu=0}^{l_{\max}} \left(TV^{p,\mu}(1_{\{u_{i,j} \geq \mu\}}) + 2\lambda \sum_{i,j} \left(\mu - \frac{1}{2}\right) 1_{\{u_{i,j} \geq \mu\}} + f_{i,j} 1_{\{u_{i,j} < \mu\}}\right) + \text{const.}$$

For simplicity, let $U_{i,j} = 1_{\{u_{i,j} \geq \mu\}}$ and correspondingly $(1 - U_{i,j}) = 1_{\{u_{i,j} < \mu\}}$, and fix a level $\mu \in \mathbb{Z}_+$. Consider the following two discrete geometry energy minimization problems that correspond, respectively, to the problems of minimizing the discrete $TV/L^1$ energy $E^1(u; \lambda, f)$ and ROF energy $E^2(u; \lambda, f)$ above for a fixed level $\mu$, again depending on the choice of $p$:

$$\min_{U \in \{0,1\}^{n \times n}} E_G(U; \lambda, \mu, f) = TV^p(U) + \lambda \sum_{i,j} [1 - 1_{\{f_{i,j} \geq \mu\}}] U_{i,j} + 1_{\{f_{i,j} \geq \mu\}} (1 - U_{i,j})$$

(2.16)
there exists a network $G$ approach similar to the one described in [22, 11] to construct flow networks for which there is a one-to-one

$$\min_{U \in [0,1]^{m \times n}} E_2^G(U; \lambda, \mu, f) = TV^p(U) + 2\lambda \sum_{i,j} \left( \left( \mu - \frac{1}{2} \right) U_{i,j} + f_{i,j}(1 - U_{i,j}) \right),$$

where $TV^p(U)$ is given by (2.10) - (2.14) with $1_{\{u_{i,j} \geq \mu\}}$ replaced by $U_{i,j}$. For given input $\lambda$ and $f$, we will use $U^*_\mu$, for $\mu \in \{0, 1, \ldots, l_{\text{max}}\}$ to denote the minimizers of (2.16) and (2.17).

The discrete TV/$L^1$ and ROF energies $E^1(u; \lambda, f)$ and $E^2(u; \lambda, f)$ above are decomposed into leveled energies of the matrices $U_\mu$ over the grey-scale levels $\mu$. However, these matrices are not independent across levels as they all depend on $u$. Therefore, minimizing the discrete TV/$L^1$ and ROF energies for $u$ may not be equivalent to minimizing the level-$\mu$ energies (2.16) and (2.17) for $U_\mu$ at every value of $\mu$ independently because $U^*_\mu$ may not equal $1_{\{u^* \geq \mu\}}$ for the same $\lambda$ and $f$. To establish this equivalence, one needs to show that given a set of solutions $\{U^*_\mu : \mu = 0, \ldots, l_{\text{max}}\}$, where $U^*_\mu$ minimizes the level-$\mu$ energy of (2.16) and (2.17), one can construct a minimizer $u^*$ of the total TV/$L^1$ and ROF energy from $\{U^*_\mu : \mu = 0, \ldots, l_{\text{max}}\}$.

In this result, which was proven for the case with discrete levels in [21, 22] and for the one with continuous levels in [51, 21, 1], is based on the fact that, for $E^1(u; \lambda, f)$ and $E^2(u; \lambda, f)$, there exist minimizers $U^*_\mu$ for $\mu = 1, \ldots, l_{\text{max}}$, such that $U^*_{\mu_1} \supset U^*_{\mu_2}$ whenever $\mu_1 < \mu_2$ and

$$u^*_{i,j} = \max\{\mu : U^*_\mu(i,j) = 1\},$$

minimizes $E^1(u; \lambda, f)$ and $E^2(u; \lambda, f)$, respectively.

In next two sections, we show that for given $f$ and $\lambda$, and each value of $\mu = 1, \ldots, l_{\text{max}}$, there exist capacitated flow networks corresponding to $E^1_G(U; \lambda, \mu, f)$ and $E^2_G(U; \lambda, \mu, f)$ for which every $s$-$t$ cut defines a matrix $U$ with a capacity equal to $E^1_G(U; \lambda, \mu, f)$ and $E^2_G(U; \lambda, \mu, f)$, respectively, up to a constant.

3. Representation of TV-Energies in Capacitated Flow Networks. We first present an approach similar to the one described in [22, 11] to construct flow networks for which there is a one-to-one correspondence between each $s$-$t$ cut and $U$, and thus between the capacity of the $s$-$t$ cut and $E^1_G(U; \lambda, \mu, f)$ and $E^2_G(U; \lambda, \mu, f)$. Henceforth, we will omit the dependence of the energies $E^1(u; \lambda, f)$ and $E^2(u; \lambda, f)$ and the geometry energies $E^1_G(U; \lambda, \mu, f)$ and $E^2_G(U; \lambda, \mu, f)$ on $f$ and $\lambda$, as these are fixed in typical applications; i.e., these energies will be denoted by $E^1(u)$, $E^2(u)$, $E^1_G(U; \mu)$ and $E^2_G(U; \mu)$, respectively. This construction enables us to find minimizers $U^*_\mu$ of $E^1_U(U; \mu)$ and $E^2_G(U; \mu)$ by finding the minimum $s$-$t$ cuts, and corresponding maximum flows, in the constructed flow networks. In Section 4 we present a more formal construction that yields the same flow networks by identifying the minimization of $E^1_G(U; \mu)$ and $E^2_G(U; \mu)$ as min-cut linear programs. The graphical/energy function approach in this section is intended to help the reader visualize the flow networks, while the max-flow/min-cut formulation approach in Section 4 rigorously justifies the network construction.

Let $G = (V, E, c, s, t)$ denote a graph (network), where $V$ and $E$ are the sets of nodes (vertices) and directed arcs (edges) $(v, w)$ from $v \in V$ to $w \in V$ with $v \neq w$, respectively, where $c$ is a nonnegative function of edge capacities, i.e., $c(v, w)$ is the capacity of edge $(v, w) \in E$, and the source $s$ and the sink $t$ are two special terminal nodes in $V$.

An $s$-$t$ cut of $G$ is a 2-partition $(S, \overline{S})$ of $V$ (i.e., $S \cup \overline{S} = V$ and $S \cap \overline{S} = \emptyset$) satisfying $s \in S$ and $t \in \overline{S}$. The capacity (value) of $(S, \overline{S})$ is defined as $c(S, \overline{S}) = \sum_{v \in S, w \in \overline{S}} c(v, w)$, i.e., it is the sum of the capacities of the arcs from $S$ to $\overline{S}$ across the cut. For example, $(S, \overline{S}) = \{(s, v_1, v_2), \{t\})$ is a cut of the network depicted in Figure 3.1 with capacity 8. The edges from $S$ to $\overline{S}$ across this cut are highlighted.

Definition 3.1 ([32]). A function $E(x_1, \ldots, x_n)$ of $n$ binary variables is called network representable if there exists a network $G = (V, E, c)$ with source node $s$ and sink node $t$ and a set of nodes $V_0 = \{v_1, \ldots, v_n\} \subseteq$
The value of the cut \((S, \bar{S}) = \{(s,1,2),(t)\}\) equals the sum of the capacities of the two highlighted arcs, which is 3 + 5 = 8.

Fig. 3.2. A network representation of \(\omega_{vw}E(U_v, U_w) + \omega_{vw}\), where \(E(U_v, U_w)\) is defined as \(E(1,1) = 0, E(0,0) = 0, E(1,0) = 1,\) and \(E(0,1) = 1\). Introduce the relationship \(U_v = 1 \Leftrightarrow v \in S\) and \(U_w = 1 \Leftrightarrow w \in S\). Then, s-t cuts corresponding to the 4 configurations \((U_v, U_w) = (1,1), (0,0), (1,0),\) and \((0,1)\) are s-t cuts \((S, \bar{S}) = \{(s,v,w),(\{s\}), (\{v,w\},\{t\}), (\{s,w\},\{v,t\}), (\{v\},\{w,t\})\},\) respectively, which have capacities \(\omega_{vw}, \omega_{vw}, 2\omega_{vw},\) and \(2\omega_{vw}\). It is easy to check that for each configuration of \((U_v, U_w)\), \(\omega_{vw}E(U_v, U_w) + \omega_{vw}\) is equal to the capacity of the corresponding s-t cut.

V - \(\{s,t\}\) such that, for any (fixed) configuration \((x_1, \ldots, x_n) \in \{0,1\}^n\), \(E(x_1, \ldots, x_n)\) is equal to a constant plus the value of the minimum s-t cut among all s-t cuts \((S, \bar{S})\) in \(G\), where \(v_i \in S \Leftrightarrow x_i = 1\) and \(v_i \in \bar{S} \Leftrightarrow x_i = 0\), for \(i = 1, \ldots, n\). **Remark.** The above definition allows the use of normal nodes representing binary variables, as well as extra nodes not associated with any binary variables. If the network contains extra nodes, the one-to-one correspondence between a configuration \((x_1, \ldots, x_n)\) and a \((S, \bar{S})\)-partition of \(\{v_1, \ldots, v_n\}\) is established by taking the min-cut with respect to the extra nodes in \(V \setminus V_0\) in the network \(G\) obtained from \(G\) by fixing the partition of \(\{v_1, \ldots, v_n\}\) according to \(v_i \in S \Leftrightarrow x_i = 1\) and \(v_i \in \bar{S} \Leftrightarrow x_i = 0\). Moreover, one can switch \(x_i = 1\) with \(x_i = 0\) for any subset or all of the nodes.

Using the additivity theorem below, we can construct networks for \(E^1_G(U; \mu)\) and \(E^2_G(U; \mu)\) by combining the sub-networks constructed for all irreducible (i.e., simplest) terms in \(E^1_G(U; \mu)\) and \(E^2_G(U; \mu)\).

**Theorem 3.2 ([32]).** The sum of finite network-representable functions, each represented by a network \(G^k = (V^k, E^k, c^k)\), is network-representable by \(G = (E, V, c)\) where \(V = \cup_k V^k\) and \(E = \cup_k E^k\) with capacities \(c(v, w) = \sum_k c^k(v, w)\), for all \((v, w) \in E\).

Theorem 3.2 is proved in [32]. It also follows from the relationship between the linear programming formulation of the minimum s-t cut problem and the minimization of network-representable functions discussed in Section 4 below.

We now show that, for every term in \(TV^r(U)\), there exists an equivalent network. It follows from (2.15) and the definition \(U_v = 1_{\{w \geq \mu\}}\) that every term in \(TV^r(U)\) has the form \(\omega_{vw}E(U_v, U_w) = \omega_{vw}(U_v - U_w)^+ + \omega_{vw}(U_w - U_v)^+\) for some \(\omega_{vw} \geq 0\). \(\omega_{vw}E(U_v, U_w)\), or more precisely \(\omega_{vw}E(U_v, U_w) + \omega_{vw}\), can be represented by the network depicted in Figure 3.2. The one-to-one correspondence between each of the four configurations of \((U_v, U_w)\) and an s-t cut is listed below Figure 3.2. The network representation is not unique. Figure 3.3 depicts another network that also represents \(\omega_{vw}E(U_v, U_w)\). We implemented the latter

\[\begin{aligned}
&\omega_{vw}E(U_v, U_w) \\
&\omega_{vw}E(U_v, U_w) + \omega_{vw}\end{aligned}\]
Fig. 3.3. This network represents the same energy $\omega_{vw}E(U_v, U_w)$ as the network depicted in Figure 3.2. There is no edge out of $s$ or into $t$; equivalently, these edges have 0 capacity. Under the relationship $U_v = 1 \Leftrightarrow v \in S$ and $U_w = 1 \Leftrightarrow w \in S$, $\omega_{vw}E(U_v, U_w) + \omega_{vw}$, for each configuration of $U_v$ and $U_w$ is equal to the capacity of the corresponding $s$-$t$ cut.

Fig. 3.4. The left and right networks represent $E^1(U_v) := \lambda(1-y)U_v + \lambda y(1-U_v)$ and $E^2(U_v) = 2\lambda((\mu - \frac{1}{2})U_v + z(1-U_v))$, respectively. In both networks, $c(s,v)$ and $c(v,t)$ are equal to the coefficients of the terms $(1-U_v)$ and $U_v$, respectively. Here, $y = 1_{\{f_{i,j} \geq \mu\}}$ and $z = f_{i,j}$, for each pixel $(i,j)$, in the TV/L1 and ROF models, respectively.

representation with pairs of oppositely directed edges in our code. In the complete network for $TV^p(U)$ obtained by combining the networks for all pixels $(i,j)$, each node $v_{i,j}$ is connected to its neighbors defined by $N^p$ by such pairs of directed edges with capacities equal to their corresponding coefficients in $TV^p(u)$.

The data fidelity terms in the TV/L1 and ROF models are composed of terms of the forms

$$E^1(U_v) := \lambda(1-y)U_v + \lambda y(1-U_v)$$

and

$$E^2(U_v) = 2\lambda \left((\mu - \frac{1}{2})U_v + z(1-U_v)\right),$$

for every pixel $v = (i,j)$, respectively. For example, for pixel $(i,j)$, $y = 1_{\{f_{i,j} \geq \mu\}}$, $z = f_{i,j}$ and $U_v = U_{i,j}$. $E^1(U_v)$ and $E^2(U_v)$ can be represented by the networks depicted in Figure 3.4. The reason for expressing energies in terms of both $U_v$ and $(1-U_v)$ now becomes clear: the capacities $c(s,v)$ and $c(v,t)$ are set equal to the coefficients of the terms $(1-U_v)$ and $U_v$, respectively.

Finally, in Figures 3.5 and 3.6, we present the complete networks constructed for $2 \times 3$ instances of $E_1^p(U; \mu)$ and $E_2^p(U; \mu)$ with $\mu = 2$ and

$$f = \begin{bmatrix} 2 & 4 & 0 \\ 1 & 3 & 0 \end{bmatrix},$$

and hence $1_{\{f \geq \mu\}} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$.

These networks can be obtained by combining the subnetworks for all of the individual terms in $E_1^p(U; \mu)$ and $E_2^p(U; \mu)$. In these networks, the black and colored edges represent the TV and fidelity terms, respectively.
Fig. 3.5. A network representation of $E_1^G(U; \lambda, \mu, f)$ with $1_{\{f \geq \mu\}} = [1 1 0; 0 1 0]$ and Neumann boundary conditions.

Fig. 3.6. A network representation of $E_2^G(U; \lambda, \mu, f)$ with $f = [2 4 0; 1 3 0]$, $\mu = 2$, and Neumann boundary conditions.

For better visualization, we place the nodes $(i, j)$ with the black edges in a plane and put $s$ and $t$ above and beneath the plane, respectively.

4. Max-Flow/Min-Cut LP Formulations. In this section, we formulate the problems of minimizing $E_1^G(U; \mu)$ and $E_2^G(U; \mu)$ as minimum $s$-$t$ cut linear programs in a capacitated flow network. First, we introduce the primal-dual pair of linear programs for the maximum $s$-$t$ flow and minimum $s$-$t$ cut problems.

A flow $f$, is a nonnegative function defined on the edges $(v, w)$ of a network $G = (V, E, c, s, t)$ that satisfies

(4.1) Flow bounds: $0 \leq f(v, w) \leq c(v, w), \quad \forall (v, w) \in E$

(4.2) Flow conservation: $\sum_{(w,v) \in E} f(w, v) - \sum_{(v,w) \in E} f(v, w) = 0, \forall v \in V \setminus \{s, t\}$. 
To interpret (4.2), we let \( f_m(v) \equiv \sum_{(w,v) \in E} f(w, v) \) denote the total amount of flow along the directed edges \((w, v)\) into \(v\), and \( f_o(u) \equiv \sum_{(v,u) \in E} f(v, u) \) denote the total amount of flow along the directed edges \((v, u)\) out of \(v\). Therefore, the flow conservation condition (4.2) specifies that for any node \(v\) other than \(s\) and \(t\), \( f_m(v) = f_o(v) \).

Given a set of nonnegative edge capacities \(c(v, w) \geq 0\), the maximum flow problem seeks to find a flow with the maximum in-flow into the sink \(t\), or equivalently, maximum out-flow from \(s\). This problem can be formulated as the following linear program:

\[
\begin{align*}
(4.3) \quad & \max f_m(v), \quad \text{subject to (4.1) and (4.2)}.
\end{align*}
\]

The dual of the linear program (4.3) is the minimum-cut problem:

\[
\begin{align*}
(4.4) \quad & \min_{\gamma, \delta} \sum_{(v, w) \in E} c(v, w) \delta(v, w) \\
& \text{subject to } \gamma(w) - \gamma(v) + \delta(v, w) \geq 0, \quad \forall (v, w) \in E \\
& \gamma(s) = 1, \quad \gamma(t) = 0, \\
& \delta(v, w) \geq 0, \quad \forall (v, w) \in E.
\end{align*}
\]

Due to the unimodularity property of node-arc incidence matrices of directed networks and the special right-hand side structure of (4.4), there exists a binary solution \((\gamma^*, \delta^*)\) of the minimum-cut linear program, i.e., a solution in which every element of \(\gamma^*\) and \(\delta^*\) is either 0 or 1. This binary solution identifies the minimum \(s-t\) cut \((S, \bar{S})\) as follows: \(S := \{v \in V : \gamma^*(v) = 1\}\) and \(\bar{S} := V \setminus S\). Note that \(s \in S\) and \(t \in \bar{S}\). By strong duality for the primal-dual pair of linear programs (4.3) and (4.4), we have, for the maximum flow \(f^*\) of (4.3) and the minimum cut defined by \(\gamma^*\) of (4.4),

\[
(4.5) \quad f_m^*(t) = \sum_{(v,t) \in E} f^*(v,t) = \sum_{(v,w) \in E} c(v,w) \delta^*(v,w)
\]

\[
(4.6) \quad \geq \sum_{(v,w) \in E} c(v,w)(\gamma^*(v) - \gamma^*(w))
\]

\[
(4.7) \quad = \sum_{(v,w) \in S, w \in \bar{S}} c(v,w).
\]

Specifically, (4.5) follows from the fact that by strong duality the optimal objective function values of (4.3) and (4.4) are equal, (4.6) follows from \(c(v, w) \geq 0\) and the first constraint in (4.4), and the last equality (4.7) follows from the definitions of \(S\) and \(\bar{S}\). But clearly for any \(s-t\) cut, and hence for \((S, \bar{S})\),

\[
f_m^* \leq \sum_{(v,w) \in S, w \in \bar{S}} c(v,w).
\]

(\text{This is weak duality.) Thus, we have that the maximum amount of flow that can be pushed from } s \text{ to } t \text{ is equal to the capacity of a minimum } s-t \text{ cut in the network.}

Next, we represent each term in \(E_1^U(U; \lambda)\) and \(E_2^U(U; \lambda)\) as the solution to a linear program in terms of an independent variable \(\delta\), while treating \(\gamma = U\), which it shares with the linear programs corresponding to the other terms, as parameters. Because these linear programs do not share any of the independent variables, we can combine them into one linear program that has an optimal objective value equal to \(E_1^U(U; \lambda)\) or \(E_2^U(U; \lambda)\). To minimize \(E_1^U(U; \lambda)\) or \(E_2^U(U; \lambda)\), we then treat \(\gamma = U\) as unknown variables and solve the resulting linear program for \(\delta\) and \(\gamma\). This last linear program has precisely the same form as the minimum-cut linear program (4.4).

\textbf{Regularization terms.} Each term in \(TV^p(U)\) has the form \(w_{uv} E(U_v, U_w) = w_{uv}(U_v - U_w)^+ + w_{uv}(U_w - U_v)^+\), where \(v\) and \(w\) stand for two different pixels/nodes. Using the fact that \((a-b)^+ = \min\{c : c \geq a - b\}\),
\[ b - a + c \geq 0, \ c \geq 0 \], we can evaluate \( w_{vw}E(U_v, U_w) \) using the formulation below, where \( c(v, w) = c(w, v) = w_{vw} \), and \( \gamma(v) = U_v \), and \( \gamma(w) = U_w \) take on arbitrary binary values:

\[
\begin{align*}
&\min \quad c(v, w)\delta(v, w) + c(w, v)\delta(w, v) \\
&\text{subject to} \quad \gamma(v) - \gamma(w) + \delta(w, v) \geq 0 \\
&\quad \gamma(w) - \gamma(v) + \delta(v, w) \geq 0, \\
&\quad \delta(v, w) \geq 0, \ \delta(w, v) \geq 0.
\end{align*}
\]

(4.8)

We note that, the minimum objective value of (4.8) is equal to \( w_{vw}E(U_v, U_w) \) for any assignment of \( \gamma(v) = U_v \) and \( \gamma(w) = U_w \). Note that for the moment, we are treating \( \gamma(v) \) and \( \gamma(w) \) as given and \( \delta(v, w) \) and \( \delta(w, v) \) as the unknown variables.

**Fidelity terms.** Each term in the sums that comprise the fidelity terms in \( E_1(U; \mu) \) and \( E_2(U; \mu) \) has the form \( aU_v + b(1 - U_v) \), for given \( a \) and \( b \) nonnegative. For example, \( a = \lambda(1 - 1_{\{f_{i,j} \geq 0\}}) \) is the coefficient of \( U_{i,j} \) and \( b = \lambda 1_{\{f_{i,j} \geq 0\}} \) is the coefficient of \( (1 - U_{i,j}) \) in the fidelity term in \( E_1(U; \mu) \). Since \( ax = \min_y\{ay : 0 - x + y \geq 0, y \geq 0\} \) for \( a \geq 0 \) and \( b(1-x) = \min_y\{by : x - 1 + y \geq 0, y \geq 0\} \) for \( b \geq 0 \), the value of \( aU_v + b(1 - U_v) \) is equivalent to the optimal objective value of

\[
\begin{align*}
&\min_y \quad c(s, v)\delta(s, v) + c(v, t)\delta(v, t) \\
&\text{subject to} \quad \gamma(v) - \gamma(s) + \delta(s, v) \geq 0, \\
&\quad \gamma(t) - \gamma(v) + \delta(v, t) \geq 0, \\
&\quad \gamma(s) = 1, \ \gamma(t) = 0, \\
&\quad \delta(s, v) \geq 0, \ \delta(v, t) \geq 0,
\end{align*}
\]

(4.9)

for \( \gamma(v) = U_v \), \( c(s, v) = b \), and \( c(v, t) = a \), given as parameters in (4.9). For both \( U_v = 0 \) and \( U_v = 1 \), the above min-cut linear program over the unknowns \( \delta(s, v) \) and \( \delta(v, t) \) has a minimum objective value equal to \( aU_v + b(1 - U_v) \).

Using (4.8), we can represent every term in \( TV^P(U) \), with two independent variables: \( \delta(v, w) \) and \( \delta(w, v) \) and we can represent the data fidelity terms in \( E_1(U; \mu) \) and \( E_2(U; \mu) \) using \( \delta(s, v) \) and \( \delta(v, t) \) in (4.9). Therefore, we can represent \( E_1(U; \mu) \) and \( E_2(U; \mu) \) for any given \( U \) by combining the linear programs (4.8) and (4.9) for all terms of \( E_1(U; \mu) \) and \( E_2(U; \mu) \). Specifically, this gives a larger linear program in the unknown variables \( \delta \) with an objective function equal to the sum of the objective functions in (4.8) and (4.9), as well as a set of constraints consisting of all of the constraints in (4.8) and (4.9), again, for all terms in \( E_1(U; \mu) \) and \( E_2(U; \mu) \). In fact, combining these linear programs having the form of (4.8) and (4.9) is equivalent to connecting the subnetworks defined by (4.8) and (4.9) for each \( (v, w) \) and \( v \) into a large flow network. In this sense, we have constructed the same flow network and min-cut problem as those in Section 3. The problem of minimizing \( E_1(U; \mu) \) or \( E_2(U; \mu) \) for an optimal \( U^*_\mu \), therefore, is equivalent to the solving for both \( \gamma \) and \( \delta \) in the combined linear program.

Comparing the objective functions and constraints of (4.8) and (4.9) to those of (4.4), it is easy to see that (4.8) and (4.9) each resembles parts of (4.4), except that \( \gamma \) is not an unknown in (4.8) and (4.9); hence, the combined linear program is also in the form of (4.4). Clearly, minimizing a linear program in the form of (4.4) for both \( \gamma \) and \( \delta \) is a min-cut problem. According to our previous analysis, every min-cut linear program has a binary minimizer \( \gamma^* \), which defines the optimal \( U^*_\mu \) for \( E_1(U; \mu) \) or \( E_2(U; \mu) \). Therefore, minimizing \( E_1(U; \mu) \) or \( E_2(U; \mu) \) is simply a min-cut linear program. In the following section, we present efficient algorithms for finding \( \gamma^* \).
5. Finding Maximum Flows and Minimum Cuts. In this section, we describe how to solve a single min-cut linear program using Goldberg-Tarjan’s Preflow-Push algorithm [26].

5.1. The Goldberg-Tarjan Preflow Algorithm. A preflow \( f \) on \( G \) is similar to a flow except that the conservation of flow requirement (4.2) is relaxed to:

\[
e^f(v) := f_{\text{in}}(v) - f_{\text{out}}(v) \geq 0 \quad \text{for } v \in V - \{s\}.
\]

Here \( e^f(v) \) denotes the excess of the preflow \( f \) at node \( v \). The capacity and nonnegativity bounds on the \( f \) must still be satisfied. The value of a preflow \( f \) is defined as \( e^f(t) \), the excess of preflow at the sink \( t \). For every preflow \( f \), there exists a flow \( \tilde{f} \) satisfying \( \tilde{f}(v,w) \leq f(v,w) \) for all \((v,w) \in E \) and \( e^f(t) = e^f(t) \).

Given a preflow \( f \), we say that an edge \((v,w)\) is saturated if \( f(v,w) = c(v,w) \), leaving no room for any more flow on \((v,w)\), and unsaturated if \( f(v,w) < c(v,w) \). If \( f(v,w) < c(v,w) \) or \( f(w,v) > 0 \) or both, then \((v,w)\) is a residual edge, and its residual capacity is equal to

\[
e^f(v,w) := (c(v,w) - f(v,w)) + f(w,v) > 0,
\]

where the first part of the sum is the maximum flow augmentation over \((v,w)\), and the second part the maximum flow reduction on its reverse edge \((w,v)\). We denote the set of residual edges by \( E^f \). For example, let \((v,w)\) and \((w,v)\) be two directed edges. If \( c(v,w) = 5 \), \( c(w,v) = 2 \), \( f(v,w) = 3 \), and \( f(w,v) = 0 \), then \( e^f(v,w) = 2 \) and \( e^f(w,v) = 5 \).

The preflow-push algorithm [26] maintains a preflow \( f \) and a distance label \( d(v) \) for every node \( v \in V \). A valid labeling \( d \) for a preflow satisfies: \( d(s) = |V| \), \( d(t) = 0 \), and \( d(v) \leq d(w) + 1 \) for all \((v,w) \in E^f \) with \( e^f(v,w) > 0 \). It is easy to see that \( d(v) \) is less than or equal to the least number of edges in any path from \( v \) to \( t \) in the residual network \( G^f = (V, E^f, e^f) \). In the algorithm, \( d \) is used to approximate this “shortest path distance” to direct preflow pushes from nodes with excess along approximately shortest paths to \( t \). Intuitively, since \( t \) is the destination of flow, pushing along a shortest path to \( t \) reduces the total number of pushes required.

Initially, all excess resides at the source node \( s \). A node \( v \) is called active if \( e^f(v) > 0 \); hence \( s \) is the only active node. The preflow-push algorithm pushes the excess at \( s \) to its adjacent nodes \( v \) in \( G \) by saturating the edges \((s,v) \in E \) (i.e., letting \( f(s,v) = c(s,v) \)), so these nodes \( v \) become active and \( s \) becomes inactive. Then, the algorithm iteratively improves the preflow \( f \) by pushing the excess residing at active nodes \( v \) toward \( t \) along admissible edges \((v,w) \in E^f \), defined as those satisfying \( d(v) = d(w) + 1 \), while also updating \( d \) dynamically to better reflect the shortest distances to \( t \). Since \( d(t) = 0 \) and preflow is pushed from nodes with larger labels \( d \) to those with smaller labels, the sink node \( t \) is the ultimate destination of the pushed flow. Without going into the details of the algorithm, it is still easy to see that \( e^f(t) \) gradually increases and finally reaches its maximum. When this happens, there is, generally, still some excess at internal nodes, that are disconnected from \( t \) in the residual network (otherwise, this excess could be pushed to \( t \)). This remaining excess is then pushed back (i.e., returned) to \( s \). These pushes happen to an active node \( v \) when \( d(v) \) is increased and becomes greater than \( d(s) \). After all remaining excess is returned to \( s \), the algorithm terminates with a max-flow \( f \) from \( s \) to \( t \). We note that returning excess to \( s \) is not necessary for finding a min-cut; see Theorem 5.1 below and the discussion following it.

The details of the preflow-push algorithm [26] are given in Algorithms 1 and 2, which include two modifications. First, we require that \( d(v) < |V| \) for a node \( v \) to be active in addition to \( e^f(v) > 0 \). Second, we employ the gap relabeling heuristic [18, 23] to avoid flow pushes that do not help increase \( f_{\text{in}}(t) \): if there
are no nodes that have a distance label equal to \( d \), all nodes \( v \) with \( d(v) > d \) are relabeled to \( d(v) = |V| \). With these two modifications, we have the following theorem:

**Theorem 5.1.** In the preflow algorithm using the gap relabeling heuristic, when a node \( v \) is defined as active if \( d(v) < |V| \) and \( e^f(v) > 0 \), \( S = \{ v : d(v) = |V| \} \) and \( \overline{S} = V \setminus S \) define a min cut \((S, \overline{S})\) at the termination of the algorithm.

**Proof.** We shall show that any edge \((v,w)\), where \( v \in S \) and \( w \in \overline{S} \), is saturated. First, we have \( s \in S \) and \( t \in \overline{S} \) since \( d(s) = |V| \) and \( d(t) = 0 \) always. If \( v = s \), then \((v,w)\) is saturated due to the initial saturating pushes from \( s \) to its neighbors. If \( v \neq s \) and \((v,w)\) is not saturated, then \( d(w) = |V| - 1 \), which is from the result [26]: \( d(w) \geq d(v) - 1 \) for any unsaturated edge \((v,w)\). According the gap relabeling heuristic, there must exist nodes at all distances \( d = 0, 1, \ldots, |V| - 2 \); otherwise, there is a gap, i.e., a distance less than \( |V| - 1 \) without any node, so the heuristic applies to \( w \): this leads to the contradiction \( d(w) = |V| \). However, there are only \(|V|\) nodes including \( s \) at \( d(s) = |V| \) and \( t \) at \( d(t) = 0 \). Therefore, there must exist a gap, so \( d(w) = |V| \), which contradicts \( w \in \overline{S} \). \( \square \)

Theorem 5.1 also shows that any node \( v \) with \( d(v) = |V| \) no longer has a directed path to \( t \) in \( G^f \), so any excess residing at it can only be returned to \( s \). The third modification is to not return this excess to \( s \). Hence, the modified preflow-push algorithm terminates with a maximum preflow instead of a maximum flow.

The discharge\((v)\) procedure in Algorithm 2 sends excess at node \( v \) to its neighbors. For this purpose, each node keeps its neighbors in an ordered list, where the order can be arbitrary but must be fixed, and a pointer current edge pointing to an edge in the list. Initially, current edge points to the first edge in the list. When procedure discharge\((v)\) is called, it starts scanning the edge list of \( v \) from current edge and moves forward in the list until either all excess at \( v \) has been pushed to its neighbors through admissible arcs or current edge reaches the end of the list with excess still left at \( v \). In the former case, current edge points to the last edge in the list in which preflow was pushed, and the procedure terminates. In the latter case, \( v \) is relabeled (i.e., \( d(v) \) is increased) to generate new admissible edges. After relabeling, current edge of \( v \) is reset to the first new admissible edge.

As there are usually several active nodes from which to choose, Algorithm 1 always picks an active node with the highest distance label to discharge, and achieves a worst case running time of \( O(|V|^2 |E|^{1/2}) \) [17]. One can order active nodes for discharge in other ways, for example, in FIFO order [26]. See [19] for a comparison of different orderings. We found that the gap relabeling heuristic is very useful for reducing the total number of pushes and relabels and, thus, total running time. Gap relabeling checks, before each call to relabel (i.e., right after current edge of \( v \) reaches the end of \( v \)'s edge list but \( v \) still has excess) whether or not \( v \) is the only node with a distance label \( d^0 = d(v) \); if true, i.e there will be no node \( v \) with \( d(v) = d^0 \) after relabeling, \( v \) skips the relabeling process by directly setting \( d(v) = |V| \) for all of the nodes \( v \) with \( d(v) \in [d^0, |V| - 1] \). These relabeled nodes can never become active, and hence are put in \( S \).

To keep track of active and inactive nodes with different labels, efficient implementations of the preflow-push algorithm employ an array of distance buckets\(^1\). All active nodes with the same distance label are stored in the same dedicated bucket, as are all inactive nodes with the same distance label. In addition, three variables \( d_{\text{max}}, a_{\text{min}}, \) and \( a_{\text{max}} \) are used to keep track of the current max distance label and min and max distance labels of active nodes. This enables gap relabeling to run at little extra cost.

**5.2. The Parametric Max-Flow Algorithm.** The data fidelity terms in \( E^1_G(U; \mu) \) and \( E^2_G(U; \mu) \) vary with \( \mu \). Therefore, according to the graph constructions in the last section, the capacities of the terminal

\(^1\)a type of data structure (see, for example, [20])
Algorithm 1 Preflow-push algorithm

saturate \((s,v)\) for all neighbors \(v\) of \(s\) \COMMENT{by modifying the residual capacities of \((s,v)\) and excesses \(e(v)\) of these \(v\)’s}  
\[d(s) \leftarrow n\text{ (inactive)}, \quad d(t) \leftarrow 0\text{ (inactive)}\]

for all \(v \in V \setminus \{s,t\}\) do

\[\begin{align*}
  &\text{if } (s,v) \in E \text{ then} \\
  &\quad d(v) \leftarrow 1\text{ (active)} \\
  &\text{else} \\
  &\quad d(v) \leftarrow 1\text{ (inactive)}
\end{align*}\]

end if

dmax,amin,amax \leftarrow 1 \COMMENT{\(d_{\text{max}},a_{\text{min}},a_{\text{max}}\) denote max distance label, maximum and minimum active node labels, respectively}

while there exists an active node in \(V\) \COMMENT{by comparing \(a_{\text{min}}\) and \(a_{\text{max}}\)} do

pick an active node \(v\) with highest \(d(v)\) \COMMENT{Remove \(v\) from active bucket \(a_{\text{max}}\); if this bucket is empty, reduce \(a_{\text{max}}\) by one}

\(\text{discharge}(v)\)

end while

edges (i.e., edges out of \(s\) and into \(t\)) are functions of \(\mu\). In the network representing \(E^1_2(U;\mu)\), the capacities \(\lambda_1(\{f\geq \mu\})\) of \((s,v)\) are nonincreasing in \(\mu\) while the capacities \(\lambda_1(1-\{f\geq \mu\})\) of \((v,t)\) are nondecreasing in \(\mu\). The same holds for the capacities of \((s,v)\) and \((v,t)\) in the network representing \(E^2_2(U;\mu)\). The capacities of all other (non-terminal) edges do not change with \(\mu\). This monotonicity property allows us to use the Gallo-Grigoriadis-Tarjan parametric maximum flow algorithm [25] to compute the max-flows/min-cuts for all values of \(\mu\) instead of calling a single maximum flow algorithm multiple times for all \(2^p\) different values of \(\mu\) for \(p\)-bit grey-scale images. For \(\mu = 0\), the min-cut is trivially \((\mathcal{S},\mathcal{F}) = (\{s\}, V - \{s\})\) since \(f_{i,j} \geq 0\) for all \((i,j)\).

Theorem 5.2. Models (1.1) and (1.2) can be solved for optimal output images with \(p\)-bit gray-scale levels in at most \(O(nm\log(n^2/m) + 2^p m\log(n^2/m))\) time by the parametric max-flow/min-cut algorithm. For Model (1.2), the running time can be reduced to \(O(nm\log(n^2/m))\), independent of the gray-scale depth \(p\). This theorem follows from the time bound for the parametric maximum flow algorithm, which we briefly discuss below.

Assumption B: for \(k = k_1, \ldots, k_K\) and \(k_1 < \ldots < k_K\), let \(G^k = (V,E,c^k,s,t)\) be a sequence of networks with the same node and edge sets \(V\) and \(E\) and terminal nodes \(s\) and \(t\), where as functions of \(k\), \(c^k(s,v)\) for all \((s,v) \in E\) is nondecreasing, \(c^k(v,t)\) for all \((v,t) \in E\) is nonincreasing, and \(c^k(v,w)\) is constant for all other edges \((v,w) \in E\).

Gallo, Grigoriadis, and Tarjan [25] found that the preflow-push algorithm can be modified to compute max-flows/min-cuts in \(G^k\) for all \(k = k_1, \ldots, k_K\) in a worst-case total time of

\[O(|V||E|\log(|V|^2/|E|) + K|E|\log(|V|^2/|E|)).\]

For \(K \leq O(|V|)\), this is the best time bound currently known for a single max-flow computation. Their parametric algorithm works as follows. Suppose for a fixed \(k = k_l\), the preflow-push algorithm terminates with a set of distance labels \(d^k\) and a (pre)flow \(f\). When changing \(k\) from \(k_l\) to \(k_{l+1}\), if \(c^k(s,v)\) strictly increases, then \(f(s,v)\) is replaced by \(c^k(s,v)\); if \(c^k(v,t)\) decreases, then \(f(v,t)\) is replaced by \(\min\{f(v,t), c^k(v,t)\}\). The residual capacities of all affected edges, and the excesses and active statuses of all affected nodes are also updated. The modified (pre)flow \(f\) is still a preflow, and the old distance labels \(d = d^k\) are still valid for
Algorithm 2 discharge($v$)

Require: $v$ is active \{Comment: $d(v) < |V|$ and $e^f(v) > 0$\}

\begin{verbatim}
end-of-edge-list ← false
d ← d(v)
for (v, w) ← current edge of $v$ to the last edge in list do
    \{Comment: It is recommended to have (v, t) as the first arc of $v$\}
    if (v, w) is admissible (i.e., $d(v) = d(w) + 1$ and $e^f(v, w) > 0$) then
        push(v, w) \{Comment: $f(v, w) ← f(v, w) + \min\{e^f(v), e^f(v, w)\}$\}
        if $w ≠ t$ was inactive then
            move $w$ from the inactive bucket $d - 1$ to the active bucket $d - 1$
        \{Comment: Whenever adding nodes to an active bucket, update $a_{\min}, a_{\max}, d_{\max}$\}
        end if
        if $e^f(v) = 0$ then
            Break the for loop
        else if $e^f(v) > 0$ and (v, w) is the last edge in the list then
            end-of-edge-list ← true
            Break the for loop
        end if
    end if
end for
\end{verbatim}

if end-of-edge-list \{Comment: $e^f(v) > 0$\} then

if Both active and inactive buckets $d$ are empty then
    gap relabel(d) \{Comment: $v$ and all $w$'s with $d(w) > d$ are relabeled to $|V|$ (they will never become active); $d_{\max}, a_{\max} ← d - 1$\}
else
    relabel(v) \{Comment: $d(v) ← \min\{\forall (v, w) : e^f(v, w) > 0\} \{d(w)\} + 1$; this task needs to scan all arcs out of $v$, and assign an unsaturated edge $\hat{w}$ with the smallest label as the current arc of $v$; $d_{\max} ← \max\{d_{\max}, d(v) + 1\}$; add $v$ to bucket $d(v)$ (active)\}
end if
else
\{Comment: $e^f(v) = 0$\}
current edge ← (v, w)
add $v$ to the inactive bucket $d$
end if

\begin{algorithm}
\caption{discharge($v$)}
\end{algorithm}

Therefore, if new active nodes are generated by the updates, the preflow-push algorithm can resume and compute a maximum (pre)flow for $G^k$. The reader should refer to [25] for the details and justification of this parametric algorithm.

If our modified algorithm is used, a min-cut ($S^k, S^F$) in $G^k$ can be easily obtained from the min-cut ($S^{k-1}, S^{F-1}$) computed for $G^{k-1}$: $S^k = \{v : d(v) = |V|\}$ is equal to the union of $S^{k-1}$ and the set of nodes relabeled to $d = |V|$ during the max-flow computation in $G^k$.

For solving the TV/$L^1$ and ROF models based on the networks constructed in Sections 3 and 4, we compute the min-cuts of those networks for a decreasing sequence of $\mu$ values. The number of distinct values of $\mu$ is equal to $K$. It is important to note that the min-cuts obtained by this procedure are nested; more specifically, $S^k(\mu) \subseteq S^k(\mu')$ for any $\mu > \mu'$. This allows one to have a nested sequence of $U^*_{\mu}$ that can be used to reconstruct $u^*$ according to (2.18).

For the TV/$L^1$ model, $K$ can be much less than the total number of grey-scales, since the capacities of the arcs from $s$ to node $(i, j)$ or from that node to $t$ change only when the value of $\mu$ becomes equal to $f_{i,j}$. 

(Refer to the left graph of Figure 3.4.) Consequently, \(\mu\) only needs to take on the distinct values taken on by all of the \(f_{i,j}\)'s. For many images, the number of distinct gray-scale values is less than the total number of grey-scale levels, so the parametric max-flow algorithm for the TV/L1 model can be applied with a relatively small number of updates. Moreover, since each \(1_{\{f_{i,j} \geq \mu\}}\) only changes at one value of \(\mu\), the capacities of associated edges change only once. Consequently, no matter how many updates there are, the total work of updating edge capacities is fixed. Therefore, as one can see from the numerical results in Section 6, the parametric max-flow algorithm is well suited for solving the TV/L1 model. However, this is not the case for the ROF model because the capacity \(2\lambda(\mu - \frac{1}{2})\) varies with \(\mu\) directly, so we need to compute min-cuts for every integer value of \(\mu \in [f_{\min}, f_{\max}]\), where \(f_{\min} := \min\{f_{i,j}\}\) and \(f_{\max} := \max\{f_{i,j}\}\).

5.3. A Two-Threaded Parametric Max-Flow Algorithm. With a little change to the constructed network, we can solve the sequence of max-flow/min-cut problems for an increasing sequence of \(\mu\) values. Given a max-flow from \(s\) to \(t\), we can obtain a max-flow from \(s' = t\) to \(t' = s\) in a new network with all directed edges and flows reversed. Since every non-terminal edges \((v, w)\) has a sibling \((w, v)\) with the same capacity, this only affects terminal edges \((s, v)\) and \((v, t)\), giving new edges \((v, t')\) and \((s', v)\); hence, for an increasing sequence of \(\mu\) values, the monotonicity properties of the capacities of the terminal nodes satisfy Assumption B. Therefore, we can solve the set of max-flow/min-cut problems corresponding to a set of \(\mu\)-values using two \(\mu\) sequences at the same time: one sequence with increasing values of \(\mu\) starting from the minimal \(\mu\) value and the other with decreasing values of \(\mu\) starting from the maximal \(\mu\) value, until the two sequences meet at the same \(\mu\) value.

To fully justify this parallel algorithm, we cannot ignore the nesting of the min-cuts generated by the two threads. In general, a network may have multiple min \(s\)-\(t\) cuts, and because of this, a particular sequence of min-cuts of a parametric network may not be nested. Therefore, even though each thread generates a nested min-cut sequence, concatenating the two sequences will not usually result in a full sequence that is nested. However, the preflow algorithm computes a min-cut that is \(S\)-minimum among all min-cuts, i.e., the computed \(S\) is a subset of all \(\hat{S}\) for min-cuts \((\hat{S}, \overline{S})\). Consequently, the preflow output computed by the thread with decreasing \(\mu\)'s are \(S\)-minimum and increasing; the min-cuts computed by the other thread with increasing \(\mu\)'s are \(S'\)-minimum (hence, \(\overline{S}\)-minimum and \(S\)-maximum) and decreasing. When the two threads meet at a value of \(\mu\), the two sequences of min-cuts can be concatenated into one nested sequence of min-cuts, allowing (2.18) to be applied for computing \(u^*\).

There is no synchronization needed for running the two independent threads. Each thread only needs to check, before the \(\mu\) of this thread is changed to the next \(\mu'\), whether the other thread has already started or even finished with \(\mu'\). If it has, then the inquiring thread terminates. Therefore, the parallelization overhead is negligible, yielding an algorithm that is almost twice as fast as the single-threaded one. To fully utilize this property, one should implement this two-threaded algorithm on two CPUs or across two computers that do not share memory/cache pipelines (but signal each other for \(\mu\)) since each thread needs to access memory extensively.

5.4. The Divide-and-Conquer Algorithm. In this section we combine the parametric maximum flow algorithm with aspects of the divide-and-conquer procedure proposed in [30] and later in [22] for image processing. These approaches use ideas of graph contraction and divide-and-conquer first described in [29] for solving parametric max-flow/min-cut problems. Our combined algorithm invalidates the time bound in [29] and [30] but speeds up the calculations for solving the TV/L1 model and especially the ROF model.

We first store all of the values \(k_1, \ldots, k_K\) in a binary tree using the following recursive procedure. We let \(k^-_{K/2^\gamma}\) be the root of the binary tree and let \(k_1, \ldots, k^+_{K/2^\gamma}\) and \(k^-_{K/2^\gamma+1}, \ldots, k_K\) be the values stored
in the left and right binary subtrees, respectively, of this root node. The left and right children of the root
node are the roots of these subtrees (which are determined recursively). A simple example is given in Figure
5.1. If \( K + 1 \) is not a power of 2, then some leaf nodes can be left empty or have the same values as their
parents.

Next, we describe how the network is modified in going from one level to the next deeper level. Let us
consider the case of going from level 1 to level 2. Recall that the parametric algorithm generates min-cuts
\((S^k, \overline{S}^k)\), where \( S^k = \{ v : d^k(v) = |V| \} \) using the labels \( d^k \) obtained at the end of \( k \)-th max-flow subproblem,
that satisfy \( S^{k_1} \subseteq \ldots \subseteq S^{k_K} \) (hence, \( \overline{S}^{k_1} \supseteq \ldots \supseteq \overline{S}^{k_K} \)). In particular, \( \overline{S}^{k_{K/2}} \), obtained by solving the level
1 problem, is a subset of \( \overline{S}^k \) for all \( k = k_1, \ldots, k_{K/2−1} \). Next, we argue that the nodes in \( \overline{S}^{k_{K/2}} \setminus \{ t \} \) do
not play a role in determining the min-cut \((S^k, \overline{S}^k)\) for any \( k < k_{K/2} \). When \( k \) is reduced from \( k_{K/2} \), the
 capacities of the arcs \((v, t)\) are nondecreasing. This may allow some of the excess of flow residing at
the nodes in \( S^{k_{K/2}} \setminus \{ t \} \) to be sent to \( t \). However, this additional excess of flow never passes through any node
\( w \in \overline{S}^{k_{K/2}} \setminus \{ t \} \) because all edges \((v, w)\), where \( v \in S^{k_{K/2}} \), are saturated after solving the level 1 problem;
hence, the set of nodes, \( \overline{S}^{k_{K/2}} \setminus \{ t \} \), cannot help route any excess of flows to \( t \) in the max-flow/min-cut
problems corresponding to any \( k < k_{K/2} \). Consequently, we do not need to include \( \overline{S}^{k_{K/2}} \setminus \{ t \} \), as well
as all edges from or to this set of nodes, in the max-flow calculations for \( k < k_{K/2} \) after solving the level
1 problem for \( k_{K/2} \). Instead of removing any nodes or edges from memory, we disconnect \( S^{k_{K/2}} \) and
\( \overline{S}^{k_{K/2}} \) from each other by simply freezing the residual capacities, as well as the flow, in all edges \((v, w)\) with
\( v \in S^{k_{K/2}} \setminus \{ s \} \) and \( w \in \overline{S}^{k_{K/2}} \setminus \{ t \} \), and only consider the sub-network corresponding to \( S^{k_{K/2}} \cup \{ t \} \) and
its induced edges.

When \( k \) is increased from \( k_{K/2} \), we can derive exactly the opposite result; that is, \( S^{k_{K/2}} \setminus \{ s \} \) does
not play a role for solving the max-flow problem corresponding to any \( k > k_{K/2} \). Therefore, we freeze the
nodes in \( S^{k_{K/2}} \), as well as any edges from or to this set of nodes, and only consider the sub-network of
\( \overline{S}^{k_{K/2}} \cup \{ s \} \) and its induced edges.

The two sub-networks corresponding to \( k_1 < k_{K/2} \) and \( k_2 > k_{K/2} \) at level 2 have only the frozen
edges in between \( S^{k_{K/2}} \) and \( \overline{S}^{k_{K/2}} \), as well as \( s \) and \( t \), in common. Since \( s \) and \( t \) only have out and in edges,
respectively, they cannot transport flow from one sub-network to another. Therefore, the two sub-networks
can be combined into one by sharing the same \( s \) and \( t \), as well as the frozen edges. The combined network
has exactly the same set of nodes and edges, including the frozen ones in between \( S^{k_{K/2}} \) and \( \overline{S}^{k_{K/2}} \), as
the original network with respect to \( k = k_{K/2} \). Therefore, the only operations needed for moving from
level 1 to level 2 are to update the capacities \( c(v, t) \) (hence, \( c^f (v) \)) and to freeze the set of edges between
\( S^{k_{K/2}} \) and \( \overline{S}^{k_{K/2}} \). It is clear that the above procedure can be recursively applied to the two sub-networks
derived from each of the subnetworks at level 2, and from them, to those at level 3 and so on.
5.5. Combining Divide-and-Conquer and Parametric Max-Flow. In [22], all of the max-flow problems for the combined sub-networks at each level of the binary tree are solved as one max-flow/min-cut problem. We do not do that; however, we also apply our algorithm to the modified version of the original network obtained by freezing certain edges, as described in the previous section, in order to isolate the sub-networks. We traverse the binary tree in right-left preorder, solving the max-flow/min-cut problem associated with each node visited. This order of traversal, which can be defined recursively as visit the root, and then visit (in right-left preorder) the right subtree followed by the left subtree, is illustrated in Figure 5.2 for a 4-level binary tree.

As we have shown in Subsection 5.2, when $k$ is increased, one can reuse the preflow and labels corresponding to the previous $k$; therefore, after solving the problem for some value of $k$, say $k_{current}$, the algorithm can solve the right problem for the new value of $k$, denoted by $k_{right}$, corresponding to the right child of $k_{current}$ without resetting any of the buckets (recall that buckets are used to store nodes according to their labels), preflow, or labels in memory. However, the same does not apply to the left problem for the value of $k$, denoted by $k_{left}$, corresponding to the left child of the previous node; this problem can only be solved after first resetting all distance labels and thus the buckets. Therefore, if both left and right problems are solved together as one max-flow/min-cut problem as was described in the last subsection, then the buckets are shared by both problems at the same time. Although the nodes in the right problem can still reuse their labels (and hence, remain in the buckets) at the end of $k_{current}$, those in the left problem, which must be assigned new distance labels (and hence, put into new buckets), will reduce the efficiency of gap relabeling. Consequently, it is more efficient to solve the left and right problems one by one. To minimize memory operations, we first reset the distance labels and buckets for the nodes in the left problem, solve the right problem using the remaining (unchanged) data, and solve the left problem later. When we apply this strategy to the entire binary tree, solving the max-flow/min-cut problems in right-left preorder fashion allows us to maximally reuse data from previous max-flow/min-cut problems.

In [25] the authors described a strategy based on solving the max-flow/min-cut problem for each $k$ in both the original and reverse networks by running two parametric preflow algorithms in parallel. Once one of the two algorithms terminates, the other can stop after directly constructing the optimal preflow, labels, and buckets from the data of the one that stops first. Recall from Section 5.3 that in the reverse network, when decreasing the value of $k$, the data from a previous $k$ can be reused to solve the new problem. Therefore, for each new $k$ (except for the very first one), in exactly one of the original and reverse networks, the preflow algorithm can resume with the data from the previous value of $k$. Since a max-flow/min-cut solution is obtained once one of the two parallel algorithms stops, it follows that this strategy applied to the entire sequence of max-flow/min-cut problems has a worst-case complexity bound equal to that of solving a single max-flow/min-cut problem by the preflow algorithm. It is not difficult to apply this strategy together with divide-and-conquer, but we believe that the overhead of running two preflow algorithms in parallel, especially on a single CPU, will slow down the computations because our problems have very sparse networks.
5.6. Other Algorithmic Improvements. The first improvement we make over the standard max-flow algorithms is to remove all terminal arcs \((s,v)\) and \((v,t)\) from the data structure, as suggested in [6, 8] and adopted in the code [9].

**Theorem 5.3.** Adding a constant capacity to both \((s,v)\) and \((v,t)\) for any node \(v\) does not change the min-cut of a network.

**Proof.** We use linear programming duality to prove this. Let \(G(V,E,c,s,t)\) denote a network, and \(G'(V,E,c',s,t)\) denote another network that has the same sets of nodes and edges and terminal nodes \(s\) and \(t\) as \(G\) but \(c'(s,v) = c(s,v) + C_v\), \(c'(v,t) = c(v,t) + C_v\) for some \(C_v \geq 0\) for all edges \((s,v)\) and \((v,t)\) in \(E\), and \(c(v,w) = c(v,w)\) for all other edges in \(E\). An \(s\)-\(t\) cut \((S,\overline{S})\) of \(G(V,E,c,s,t)\) is minimum if there exists a (pre)flow \(f\) such that \(c'(t) = c(S,\overline{S})\), i.e., the flow value is equal to the cut capacity. Consider \(f\) in \(G'(V,E,c',s,t)\). Since each \(s\)-\(v\)-\(t\) path has the residual capacity \(C_v\), a flow \(f'\) can be obtained by augmenting \(C_v\) units of flow over all \(s\)-\(v\)-\(t\) paths in \(f\) and hence \(c'(t) = f' + \sum_{e\in(s,v),(v,t)\in E} C_v\). Since for every non-terminal node \(v\) either \((s,v)\) or \((v,t)\) crosses the cut \((S,\overline{S})\), we have \(c'(S,\overline{S}) = c(S,\overline{S}) + \sum_{e\in(s,v),(v,t)\in E} C_v\). Therefore, we have \(c'(t) = c'(S,\overline{S})\) from which it follows that \(f'\) is a max (pre)flow in \(G'(V,E,c',s,t)\), and hence \((S,\overline{S})\) remains a min-cut in \(G'(V,E,c',s,t)\). \(\Box\)

Using Theorem 5.3, any changes that need to be made to \(c^k(s,v)\), as a function of \(k\), can be instead applied to \(c^k(v,t)\) so that \(c^k(s,v)\) remains constant in \(k\). To do this, we let \(c^k(s,v) \equiv c^\text{max}(s,v) = \max_k c^k(s,v)\) and \(c^k(v,t) = c^k(v,t) + [c^k(s,v) - c^k(v,t)]\). When applying the parametric algorithm to the resulting network \(G(V,E,c,s,t)\), we can avoid updating the flow \(f(s,v)\) over all edges \((s,v)\in E\) after each parameter update. Consequently, after initially pushing flow from \(s\) to all of its neighbors \(v\), the flow over all edges \((s,v)\) are no longer changed. This allows us to implement these initial pushes by directly setting \(e(v) = c^k(s,v)\) and not store any data for the edges \((s,v)\in E\) in our algorithm.

We can also avoid storing any data for the edges \((v,t)\in E\). Since there is an edge \((v,t)\) for every \(v\in V\setminus\{s,t\}\), whenever both \(e(v) > 0\) and \(c^k(v,t) > 0\), we can augment the current flow \(f\) by pushing \(\min\{e(v), c^k(v,t)\} > 0\) units of flow from \(v\) to \(t\), and thus reduce \(e(v)\) or \(c^k(v,t)\) or both to zero after the push. Therefore, we combine \(e(v)\) and \(c^k(v,t)\) into a new quantity \(\bar{e}(v)\) that is now allowed to be negative: when \(\bar{e}(v) \geq 0\), \(e(v) = \bar{e}(v)\) and \(c^k(v,t) = 0\); otherwise, \(e(v) = 0\) and \(c^k(v,t) = -\bar{e}(v)\). This change not only saves storage but also implicitly pushes \(\min\{e(v), c^k(v,t)\}\) units of flow to \(t\) whenever it is positive.

6. Numerical results. In this section, we present numerical results that illustrate the effectiveness of our parametric and divide-and-conquer max-flow/min-cut algorithms for solving the TV/\(L^1\) and ROF models on large-scale images with either 8-bit (0 to \(2^8 - 1 = 255\)) or 16-bit (0 to \(2^{16} - 1 = 65,535\)) gray scales.

Our test images include two \(512 \times 512\) 2-dimensional images: Barbara and CT, as well as two \(217 \times 218\) (approximately 7.1 million voxels) 3-dimensional MRI images: MRI-T1 and MRI-T2. The two 2-dimensional original images are depicted in Figure 6.1. The Slices \(z = 50\) and 100 of the two 3-dimensional MRI images are depicted in Figure 6.2.

The values of \(\lambda\) used in our tests were determined experimentally. In the TV/\(L^1\) model, \(\lambda\) acts like a threshold on scale [51] so the choice of \(\lambda\) depends on the application. In the ROF model, \(\lambda\) determines the amount of noise removed (see [42] for example). The issue of how to optimally select \(\lambda\) is important, but outside the scope of this paper. To evaluate the efficiency of our codes, we used two values of \(\lambda\) in both models for each image, one causing slight over-smoothing and the other slight under-smoothing, to demonstrate how running times depend on \(\lambda\).

We developed our codes in C++ based on HIPR, a code in C available from Goldberg’s Network Op-
PARAMETRIC MAX-FLOWS FOR TV MINIMIZATION

6.1. The Parametric Versus Divide-and-Conquer Approaches for Solving the TV/$L^1$ Model.

In Table 6.1, we give the running times for solving the TV/$L^1$ model on Barbara using both the parametric and divide-and-conquer algorithms. For solving the TV/$L^1$ model, the former algorithm is more effective than the latter for all $\lambda$ values and neighborhoods. For each of the four settings, the running time for the latter algorithm is about twice that for the former algorithm on Barbara, which has 228 different gray-scale values. Even though the parametric algorithm had to perform 227 updates to the flow network constructed for the TV/$L^1$ model on Barbara, each of these updates only affected a small number of edges out of $s$ or into $t$. The total number of updates applied to the capacities of the edges $(s,v)$ and $(v,t)$ is limited to two for each $v$ corresponding to the pixel $(i,j)$. Specifically, they were changed when $\mu$ was increased from $f_{i,j}$...
to the next gray-scale level. Therefore, it turns out that the total time required by the parametric algorithm for solving the parametric max-flow problems with 228 levels and thus 227 updates is less than the total time for solving essentially 8 different single max-flow problems plus the overhead of 7 update operations. We also observed similar results in applying the TV/L^1 model to other images; that is, the parametric algorithm was always more efficient than the divide-and-conquer algorithm. Therefore, for solving the TV/L^1 model, we recommend the parametric max-flow algorithm. For this reason, we do not present times for the divide-and-conquer algorithm for solving the TV/L^1 model applied to three-dimensional MRI images in Section 6.5 below.

6.2. The Effect of the Number of Neighbors p and the Value of λ. In Table 6.1, we also observe that the running times depend on both the number of neighbors and the value of λ. Using the network in which each node is connected to 16 neighbors took both algorithms more running time to find the min-cuts than with 4 neighbors, but the multiplicative factors are well less than four. Using a small λ also caused the running times to increase because smaller λ’s reduce the capacities of the terminal arcs out of s and into t are proportional to λ. The reduced capacities of the terminal arcs reduces the total amount flow into the network. This makes the non-terminal arcs less likely to be saturated, and thus allow the max-flow to have more and longer paths, each carrying less flow on average, from s to t. Consequently, the preflow-push algorithm performs more pushes and label updates before finding a maximum preflow. Since the value of λ affects the performance of the preflow-push algorithm, which is the subroutine of both the parametric and divide-and-conquer algorithms, the same λ-effect was observed for both algorithms applied to both the TV/L^1 and ROF models (Ref. Tables 6.2-6.4).

6.3. The Divide-and-Conquer Approach for Solving the ROF Model. For the ROF model, the comparison between the parametric and divide-and-conquer algorithms yielded the opposite result: the divide-and-conquer algorithm was much more efficient that the parametric algorithm. While the divide-and-
conquer algorithm essentially solves \( \rho \) single max-flow problems and incurs an overhead of \( \rho - 1 \) updates, the parametric algorithm for the ROF model must perform \( f_{\text{max}} - f_{\text{min}} + 1 \) updates, and for each update, the capacities of all arcs into \( t \) need to be changed since their capacities depend on \( \mu \). The total time spent on these inter-level updates (as many as 255 for an 8-bit grey-scale image versus 7 for the divide-and-conquer algorithm) far outweighs the savings accrued by solving a sequence of max-flow problems in a parametric way utilizing the flow and label information from one problem to the next. Therefore, it was not surprising that the running times of the parametric algorithm for solving the ROF models were so excessive that the parametric algorithm was even far slower than PDE-based and second-order cone programming-based algorithms. Therefore, we did not include the parametric algorithm in any experiments for solving ROF models.

Table 6.2 gives the running times of the divide-and-conquer algorithm for solving the ROF models on noisy Barbara with 8-bit and 16-bit gray-scale levels, different lambda values, and different numbers of neighbors. We generated the 16-bit image of noisy Barbara by multiplying the 8-bit image by \( 2^{28} + 1 = 257 \) because this this scales \([0, 2^{28} - 1] \) to \([0, 2^{16} - 1] \). Since the TV term is linear but the data fidelity term is quadratic in the gray-scale depth, we need to multiply the data fidelity term by \( \frac{1}{257} \) to discount the change of gray-scale depth in order to obtain consistent results from the 8-bit and 16-bit inputs. This is equivalent to dividing \( \lambda \) by 257 when calling the same algorithm on 16-bit images. Since for the 16-bit image of noisy Barbara the divide-and-conquer algorithm took 16 iterations, which doubles those for the 8-bit image of noisy Barbara, the running times on the 16-bit image are also about twice as long as those on the 8-bit image under the same settings of neighborhood and \( \lambda \). Moreover, in Table 6.2, we observe that the running times are increased if either \( \lambda \) is decreased or the size of neighborhood is increased.

The running time of the parametric algorithm for solving the TV/L\(^1\) model, however, is not directly affected by the gray-scale depth. No matter what the depth is, the number of sequential max-flows it solves is equal to the number of distinct gray-scale level values in an image, and the total number of capacity updates applied to terminal edges is no more than twice as many as the number of total pixels. Therefore, the running time of the parametric algorithm for solving the TV/L\(^1\) model on a 16-bit image, obtained from a 8-bit image by gray-scale multiplication, would be almost the same to that on the 8-bit image. For this reason, we do not present the redundant timing results for solving the TV/L\(^1\) model on 16-bit images.

6.4. The TV/L\(^1\) Model Applied on the Image CT. The small-scale features in the image CT, depicted in Figure 6.1, make this image a good example for demonstrating the processing by the TV/L\(^1\) model. According to the analysis in [51], the TV/L\(^1\) model decomposes an image into two parts, one with large-scale cartoons and background, the other with small-scale oscillating features. The selection of scales is determined by the value of \( \lambda \). The decomposition results obtained with \( \lambda = 0.5 \) and \( \lambda = 1.0 \) using the

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parametric</th>
<th>Divide-and-conquer</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda )</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>Number of neighbors</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>Graph Generation Time (sec)</td>
<td>0.37</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>0.37</td>
<td>0.08</td>
</tr>
<tr>
<td>Parametric Max-Flow Time (sec)</td>
<td>1.23</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>0.64</td>
<td>0.39</td>
</tr>
<tr>
<td></td>
<td>2.50</td>
<td>1.61</td>
</tr>
<tr>
<td></td>
<td>1.76</td>
<td>1.14</td>
</tr>
</tbody>
</table>

Table 6.3 The TV/L\(^1\) model applied to CT
4-neighborhood are depicted in Figure 6.3. With the smaller $\lambda = 0.5$, most fine features in the image CT are kept in the $v$ part, making the $u$ part of the decomposition easier for many image processing tasks, including tracking, segmentation, registration, etc. However, if $\lambda$ is increased to 1.0, more fine features appear in the $u$ part of the decomposition.

In Table 6.3, we give the running times of this experiment with the image CT. Though the images Barbara and CT are very different in content, brightness, contrast, as well as scales of features, by comparing the running times in Table 6.3 of the tests on CT to those in Table 6.1 of the tests on Barbara, we observe that both algorithms took about the same time to process both images. In addition, the running times in Table 6.3 match our previous analysis on how the size of the neighborhood, the value of $\lambda$, and the choice of parametric or divide-and-conquer approaches affect the running times.

6.5. 3-dimensional MRI Images. We obtained two 3-dimensional MRI images from BrainWeb at http://www.bic.mni.mcgill.ca/brainweb: MRI-T1 and MRI-T2. T1 and T2 stand for two different modalities for MRI imaging which may give different intensities for the same type of tissue. For example, fat appears bright in T1 images but intermediate dark in T2 images; hemangiomas appear dark in T1 images but bright in T2 images. The in-plane pixel size of the testing MRI images is $1\times1\text{mm}$, and the slice thickness is $1\text{mm}$. The image size is $181\times217\times181$, so both images have approximately 7.1 million voxels. We also downloaded from BrainWeb T1 and T2 MRI images each with 5% of Gaussian noise. According to BrainWeb, the “percent noise” number represents the percent ratio of the standard deviation of the Gaussian noise versus the signal for a reference tissue. These images are depicted in the second row of Figure 6.2.

We applied the TV/L1 model on the noise-free T1 and T2 images, and the ROF model on their
The cartoon outputs of the TV/L1 model applied with $\lambda = 0.5$.

The cartoon outputs of the TV/L1 model applied with $\lambda = 1.0$.

The denoising outputs of the ROF model applied with $\lambda = 0.3$.

Table 6.5

<table>
<thead>
<tr>
<th>Image</th>
<th>Barbara</th>
<th>CT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Size</td>
<td>512x512</td>
<td></td>
</tr>
<tr>
<td>Gray-scale</td>
<td>8-bit</td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>TV/L1</td>
<td>ROF</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Parametric</td>
<td>Divide-and-conquer</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>1.0</td>
<td>0.05</td>
</tr>
<tr>
<td>No. of neighbors</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Graph Gen. Time (sec)</td>
<td>0.02</td>
<td>0.09</td>
</tr>
<tr>
<td>Para. Max-Flow Time</td>
<td>0.09</td>
<td>0.47</td>
</tr>
</tbody>
</table>

The TV/L1 and ROF models applied to Barbara and CT that are 4 times smaller and larger than the original. The results show that the run times of both codes are approximately linear in the number of pixels.
<table>
<thead>
<tr>
<th>Image</th>
<th>T1</th>
<th>T2</th>
<th>T1, 5%noise</th>
<th>T2, 5%noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Size</td>
<td>181×217×181</td>
<td>181×217×181</td>
<td>181×217×181</td>
<td>181×217×181</td>
</tr>
<tr>
<td>Gray-scale</td>
<td>TV/L1</td>
<td>5%noise</td>
<td>ROF</td>
<td>5%noise</td>
</tr>
<tr>
<td>Model</td>
<td>TV/L1</td>
<td>5%noise</td>
<td>ROF</td>
<td>5%noise</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Parametric</td>
<td>Divide-and-conquer</td>
<td>Divide-and-conquer</td>
<td></td>
</tr>
<tr>
<td>λ</td>
<td>1.0</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of neighbors</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>No. of pixels</td>
<td>1/8 × org.</td>
<td>1/8 × org.</td>
<td>3/4 × org.</td>
<td>3/4 × org.</td>
</tr>
<tr>
<td>Graph Gen. Time (sec)</td>
<td>0.55</td>
<td>4.44</td>
<td>0.51</td>
<td>4.44</td>
</tr>
<tr>
<td>Para. Max-Flow Time</td>
<td>2.25</td>
<td>18.71</td>
<td>2.22</td>
<td>26.83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.14</td>
<td>27.58</td>
<td>3.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.73</td>
<td>4.73</td>
<td>25.49</td>
</tr>
</tbody>
</table>

**Table 6.6**

The TV/L¹ and ROF models applied to 3D MR images that are 8 times smaller than the original. The results show that the run times of both codes are approximately linear in the number of pixels.

### 6.6. The Effect of the Number of Pixels on Run Times and Memory.

We found that the run times are approximately linear in the total number of pixels (voxels) given that all other parameters are fixed. To demonstrate this, we applied our codes to the 2D images Barbara and CT and the 3D MR images T1 and T2 in new sizes. To the 512×512 images Barbara and CT each, we generated two new images by cropping the center into a smaller 256×256 piece with a quarter of the pixels and replicated the image four times and combined them into a larger 1024×1024 image with four times as many pixels. For the 181×217×181 images T1 and T2 each, we obtained a smaller image from the central 90×106×90 pixels with 1/8 the number of pixels with respect to the original one. The run time results are given in bold in Tables 6.5 and 6.6. Although it seems that the run times grow slightly faster than linearly according to these results, the same experiments on a laptop with an Intel Core Duo CPU gave run times that are almost exactly linear in the number of pixels. This difference suggests that computer architecture factors such as memory size, bus speed, and the amount of cache may have slightly increased the rate of growth.

Memory required by our algorithm grows linearly with the number of pixels. It is dominated by the data of graph nodes and arcs. Each node or arc takes a certain number of bytes to store its associated status, as well as pointers (memory references) to out-going arcs (for nodes), and reverse arcs and head nodes (for arcs). While it is difficult to determine the exact amount of memory required by our code because it is called through MATLAB, which itself takes varying amounts of memory, we observed that the peak memory usages on our 3GB-memory PC reported by the system utility “top” were 6.2%, 6.3%, and 53.0% for 8-bit 512 × 512 images with a 16-neighbor setting, 16-bit 512 × 512 images with a 16-neighbor setting, and 8-bit 181 × 217 × 181 images with a 6-neighbor setting. At idle, MATLAB took 2.9% of memory.

### 7. Conclusion

This paper focuses on super-fast algorithms for solving the TV based L¹ and L² models as parametric max-flow/min-cut problems. To obtain the max-flow/min-cut formulations, we anisotropically discretize total variation using different neighborhood sizes for different accuracies, and decompose each resulting discretized problem into a sequence of decoupled problems in binary variables, each of which we show to be a min-cut problem in a constructed flow network. A solution of the original TV/L¹ or ROF model is constructed from a sequences of nested min s-t cuts, that we obtain using a parametric max-flow/min-cut algorithm modified from Goldberg and Tarjan’s preflow algorithm [26] and Gallo, Grigoriadis, and Tarjan’s parametric preflow algorithm [25]. Our MATLAB/C++ based codes appear to be much faster than any non-graph/network based code and is able to process a full 3-dimensional brain MR image over 7 million voxels on a standard workstation in less than a minute. Our codes can be downloaded from the second author’s homepage.
Finally, we note that even though we do not study solving the more general TV/$L^p$ model, for $1 \leq p < \infty$, a method similar to the one for the ROF model can solve this model. It is also straightforward to extend our method to solving models with a weighted TV and/or weighted $L^p$-fidelity term.

Acknowledgement. We want to thank Jerome Darbon, Paul Rodriguez, William Symes, Yilun Wang, Brendt Wohlberg, and Yin Zhang for their helpful discussions and comments.
REFERENCES


