Title
An MBO Scheme on Graphs for Classification and Image Processing

Journal
SIAM Journal on Imaging Sciences, 6(4)

ISSN
1936-4954

Authors
Merkurjev, Ekaterina
Kostić, Tijana
Bertozzi, Andrea L

Publication Date
2013

DOI
10.1137/120886935

Peer reviewed

SIAM J. Imaging Sciences
Vol. 6, No. 4, pp. 1903–1930

An MBO Scheme on Graphs for Classification and Image Processing*

Ekaterina Merkurjev†, Tijana Kostić†, and Andrea L. Bertozzi†

Abstract. In this paper we present a computationally efficient algorithm utilizing a fully or seminonlocal graph Laplacian for solving a wide range of learning problems in binary data classification and image processing. In their recent work [*Multiscale Model. Simul.*, 10 (2012), pp. 1090–1118], Bertozzi and Flenner introduced a graph-based diffuse interface model utilizing the Ginzburg–Landau functional for solving problems in data classification. Here, we propose an adaptation of the classic numerical Merriman–Bence–Osher (MBO) scheme for minimizing graph-based diffuse interface functionals, like those originally proposed by Bertozzi and Flenner. We also make use of fast numerical solvers for finding eigenvalues and eigenvectors of the graph Laplacian. Various computational examples are presented to demonstrate the performance of our algorithm, which is successful on images with texture and repetitive structure due to its nonlocal nature. The results show that our method is multiple times more efficient than other well-known nonlocal models.

Key words. image processing, Nyström extension, Ginzburg–Landau functional, MBO scheme

AMS subject classifications. 68U10, 49M25, 62H30, 91C20, 49-04

DOI. 10.1137/120886935

1. Introduction.

This work develops a fast algorithm for a recent variational method in a graph setting. The method is inspired by diffuse interface models that have been used in a variety of problems, such as those in fluid dynamics and materials science. We consider data represented as nodes in a weighted graph, and each edge is assigned a numerical value describing the similarity between the nodes. In spectral graph theory, this approach is successfully used to perform various learning tasks in imaging and data clustering. The standard techniques of the theory are thoroughly described in [13, 45], and the graph Laplacian, which is discussed in more detail in section 2.2, is introduced as one of the fundamental concepts. In imaging, spectral methods are often used in image segmentation applications, as shown in [54, 33, 14].

We are particularly interested in nonlocal total variation methods, as they are a link between spectral graph theory and diffuse interface models and thus can be used as a motivation for our algorithm. These methods are used in numerous image processing applications. They were initially developed as methods for image denoising [9, 29] but were successfully applied to many other image processing problems such as inpainting and reconstruction in [30, 63, 49], image deblurring in [40], and manifold processing in [18].

Bertozzi and Flenner introduce a graph-based model based on the Ginzburg–Landau func-

*Received by the editors August 3, 2012; accepted for publication (in revised form) May 30, 2013; published electronically October 10, 2013. This work was supported by ONR grants N000141210040 and N00014120838, AFOSR MURI grant FA9550-10-1-0569, and by the W. M. Keck Foundation.
http://www.siam.org/journals/siims/6-4/88693.html

†Mathematics Department, UCLA, Los Angeles, CA 90095-1555 (kmerkurev@math.ucla.edu, kostict@math.ucla.edu, bertozzi@math.ucla.edu). The first author was also supported by an NSF graduate fellowship.

tional in their work [8]. To define the functional on a graph, the spatial gradient is replaced by a more general graph gradient operator. Bertozzi and Flenner propose a classification algorithm by minimizing the Ginzburg–Landau functional with a fidelity term,

$$E(u) = \frac{\epsilon}{2}\int |\nabla u|^2 dx + \frac{1}{\epsilon}\int W(u)dx + F(u, u_0), \tag{1.1}$$

where u_0 is the initial state of the system. They replace the $\frac{\epsilon}{2}\int |\bigtriangledown u|^2 dx$ term with a more general graph operator term, $\epsilon u \cdot L_s u$, to be discussed in detail in sections 2.2 and 2.3, so that

$$E(u) = \epsilon u \cdot L_s u + \frac{1}{\epsilon}\int W(u)dx + \int F(u, u_0). \tag{1.2}$$

The functional is minimized using the method of gradient descent, resulting in the following expression:

$$\frac{\partial u}{\partial t} = -\epsilon L_s u - \frac{1}{\epsilon}W'(u) - \frac{\partial F}{\partial u}. \tag{1.3}$$

Note that this is just the Allen–Cahn equation with a fidelity term, where Δu is replaced by a graph operator term $-L_s$. Properties of the operator L_s will be explained in more detail in sections 2.2 and 2.3. Taking F to be $\frac{1}{2}C\lambda(x)(u - u_0)^2$ for some constant C, one obtains

$$\frac{\partial u}{\partial t} = -\epsilon L_s u - \frac{1}{\epsilon}W'(u) - C\lambda(x)(u - u_0). \tag{1.4}$$

The main purpose of this paper is to develop a fast and simple method for solving (1.4) in the small ϵ limit. To achieve our goal, we created a graph-based MBO (Merriman–Bence–Osher) scheme. The famous MBO scheme uses simple threshold dynamics to approximate motion by mean curvature. Since the Allen–Cahn equation is closely related to motion by mean curvature, the MBO scheme has been proven to be a very successful tool in solving different variants of the Allen–Cahn equation. For example, the authors of [22] propose an adaptation of the MBO scheme to minimize the piecewise constant Mumford–Shah functional. Inspired by the efficiency and the robustness of the MBO scheme, we decide to adapt it solve (1.4). However, the implementation of the proposed scheme poses many computational challenges. The quadratic size of the graph Laplacian could make the iterative process of our algorithm very computationally expensive. To reduce the dimension of the graph Laplacian and make the computation more efficient, the authors of [8] propose the Nyström extension method [25] for approximating eigenvalues and the corresponding eigenvectors of the graph Laplacian. To maximize the performance of our algorithm, for each data set we use either the Nyström extension or the Raleigh–Chebyshev algorithm proposed in [1]. The details of our algorithm are discussed in section 3, after sections 2.2–2.4 on the relevant background.

There are several reasons for the efficiency of our algorithm. Our method is a geometric approach to the minimization problem, as opposed to the more traditional L1 minimization approach. In addition, our method is more simple and less computationally complex than that described in the paper of Bertozzi and Flenner [8]. Moreover, we take advantage of fast numerical solvers for eigenvalue/eigenvector decomposition as well as using only a small number of principal components in our numerical iterations.

In this paper, we go beyond the applications presented in [8] and devise a nonlocal inpainting algorithm. To show the efficiency of our algorithm, we compare the computational times of our algorithm against those obtained by the state-of-the-art nonlocal inpainting algorithm form [30]. For the data classification problem presented in Figure 2, the comparison of our results to those obtained in the new work [8] demonstrate that our algorithm produces a significant speedup, while producing comparable results in terms of quality. In section 4, the inpainting results show that our method is about five times more efficient than other well-known nonlocal models, while being comparable in quality.

This paper is organized as follows. In section 1, we review the motivation for our method as well as some relevant background such as diffuse interfaces, the Ginzburg–Landau functional, graphs, nonlocal operators, and the MBO scheme. We then introduce our algorithm, which is applied to classification and inpainting in sections 2 and 3, respectively; show results; and include comparisons to some of the recent methods. The performance of our data classification algorithm is compared against the algorithm from [8]. The algorithm from [8] is more accurate than other standard classification algorithms, such as a naive Bayes decision trees from [50] and p-Laplacian spectral clustering from [56]. We compared our inpainting results to those obtained using the split Bregman method from [62], which is a well-known efficient algorithm for nonlocal TV minimization. The advantage of this new method is its speed and its ability to recover texture and repetitive structure in an image.

2. Background.

In this section, we present some useful background information. In section 2.1, we review the Ginzburg–Landau functional, which is the core of the model of Bertozzi and Flenner in [8]. Our algorithm is graph-based, so a background on graphs is given in section 2.2. To understand the motivation behind the definition of the graph Laplacian, one may turn to the theory of nonlocal operators, outlined in section 2.3. The MBO scheme, which serves as the core element in our model, is described in section 2.4. Finally, two methods to compute eigenvalues and associated eigenvectors are described in section 2.6.

2.1. Ginzburg–Landau functional.

Numerous image segmentation energy functionals use a binary segmentation function that takes a certain value inside the segmented region and a different one outside of the segmented region. In their pioneering work [46], Mumford and Shah propose an energy functional that uses the perimeter of the segmentation function as a regularizer. Many papers, such as [12], successfully use the total variation (TV) semi norm

$$||u||_{TV} = \int_{\Omega} |\nabla u| dx \tag{2.1}$$

to approximate the perimeter of the front between the two values of the segmentation function. As an alternative to this approach, some researchers, such as Esedoḡlu and Tsai in their work [22], use the Ginzburg–Landau functional

$$GL(u) = \frac{\epsilon}{2} \int |\nabla u|^2 dx + \frac{1}{\epsilon} \int W(u) dx \tag{2.2}$$

to approximate the perimeter of the front. $W(u)$ is a double well potential. In this work, $W(u) = (u^2 - 1)^2$ is used. Note that, due to the nature of the potential, the functional is used for binary data.

A proof in [39] shows that the perimeter is the limit in the sense of Γ-convergence of the Ginzburg–Landau functional. Therefore, one can write

$$GL(u) \to_\Gamma C|u|_{TV}. \tag{2.3}$$

This convergence allows the two functionals to be interchanged in some cases. One might prefer to use the Ginzburg–Landau functional instead of the TV seminorm since its highest order term is purely quadratic, which allows for efficient minimization procedures. In contrast, minimization of the TV seminorm leads to a nonlinear curvature term, making it less trivial to solve numerically. However, recent advances, such as the split Bregman method described in [31], have made progress in such problems.

Due to its connection to the TV seminorm, the Ginzburg–Landau functional has also often been used in image processing and in various image processing applications, such as inpainting [16, 7] and segmentation [19, 22]. In practice, one would minimize

$$E(u) = GL(u) + F(u, u_0), \tag{2.4}$$

where F is the fidelity term and u_0 is the initial state of the system. In the case of inpainting, the fidelity term is $C\int(u - u_0)^2$, where one integrates over the known region only. For denoising, the term is an L^2 fit, $C\int(u-u_0)^2$. In the case of deblurring, it is $C\int(K*u-u_0)^2$, where K is some kernel. Of course, a different norm, such as the L^1 norm, can be used.

When one minimizes the Ginzburg–Landau functional, the function u approaches either one of the two minimizers, 1 and -1, of the double well potential. However, the presence of the gradient term will force u to be somewhat smooth, i.e., without any sharp transitions between 1 and -1. Therefore, the function that minimizes the functional will have regions where it is close to -1, regions close to 1, and a thin region of scale $O(\epsilon)$ where it is somewhere in between. Since the minimizer appears to have two phases with an interface between them, models involving the Ginzburg–Landau functional are typically referred to as "diffuse interface models."

2.2. Background on graphs. In this paper, to create a nonlocal method, we use the theory of graphs, described in [13]. Consider an undirected graph $G = (V, E)$, where V and E are the sets of vertices and edges, respectively. In the tests done in this paper, the vertices are, for example, points in $\mathbb{R}^n$ or pixels in an image. Let w be the weight function, where $w(i, j)$ represents the weight (often measured between 0 and 1) between vertices i and j and $w(i, i)$ is set to zero. The weight represents a measure of similarity between the vertices; thus, two vertices having a weight close to 1 are very similar to each other, and two vertices having a weight close to 0 are dissimilar.

Now let the degree of a vertex $i \in V$ be defined as

$$d(i) = \sum_{j\in V} w(i, j). \tag{2.5}$$

Using the above, one defines the graph Laplacian to be the matrix L such that

$$L(i, j) = \begin{cases} d(i) & \text{if } i = j, \\ -w(i, j) & \text{otherwise.} \end{cases}$$

If we define the degree matrix D to be the $N \times N$ diagonal matrix with diagonal elements $d(i)$, then the graph Laplacian can be written in matrix form as $L = D - W$, where W is the matrix $w(i,j)$. The matrix W is sometimes referred to as the "affinity matrix."

Note that the graph Laplacian satisfies the equations

$$Lu(i) = \sum_j w(i,j)(u(i) - u(j)), \tag{2.6}$$

$$u \cdot Lu = \frac{1}{2} \sum_{i,j} w(i,j)(u(i) - u(j))^2 \tag{2.7}$$

for all $u \in \mathbb{R}^n$ and has nonnegative, real-valued eigenvalues, including 0.

When working with the graph Laplacian, one must consider the behavior that arises as the sample size grows larger. Increasing sample size leads to decreasing grid size; thus, the operator must be scaled to converge to the differential Laplacian as $N \to \infty$, where N is the number of vertices. Although several versions that have been shown to have the correct scaling in the limit exist, the one used in this paper is the symmetric Laplacian

$$L_s = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \tag{2.8}$$

that satisfies

$$u \cdot L_s u = \frac{1}{2} \sum_{i,j} \frac{w(i,j)(u(i) - u(j))^2}{\sqrt{d(i)d(j)}} \quad \forall u \in \mathbb{R}^n. \tag{2.9}$$

We use this version since the symmetric property of the matrix allows for more efficient algorithms for calculating eigenvectors, which is necessary for our algorithm.

Another version that is commonly used is the random walk Laplacian,

$$L_w = D^{-1} L = I - D^{-1} W, \tag{2.10}$$

which is related to Markov processes. More detail about normalized Laplacians is given in [13] and [59].

2.2.1. Choice of similarity function. As mentioned in previous sections, the weight function $w(i,j)$ is a function that measures the degree of similarity between vertices i and j. Therefore, it is necessary to choose the function in such a way that two vertices that are heavily weighted by w, i.e., $w(i,j)$ large, are also closely related in the data. Although several options for w are discussed in [59], the choice depends on the problem, so no general theory can be formulated.

One popular choice for the similarity function is the Gaussian function

$$w(i,j) = e^{-\frac{d(i,j)^2}{\sigma^2}}, \tag{2.11}$$

where $D(i,j)$ is some distance measure between the two vertices i and j, and σ is a parameter to be chosen. Von Luxburg in [59] explains that σ can be chosen to be on the order of

$\log(n)+1$, where n is the number of vertices. This similarity function is an appropriate choice when vertices are, for example, points in $\mathbb{R}^n$, since two points that are close together are more likely to belong to the same cluster than are two points that are far apart.

Another choice for the similarity function used in this work is the Zelnik-Manor and Perona weight function for sparse matrices described in [61]:

$$w(i,j) = e^{-\frac{d(i,j)^2}{\sqrt{\tau(i)\tau(j)}}}, \tag{2.12}$$

where the local parameter $\sqrt{\tau(i)} = d(i,k)$ and k is the Mth closest vertex to vertex i. As noted in [8], one should use this similarity function for classification when there exist multiple scales to be classified. In [61], M is chosen to be 7, while in [56], it is 10. Depending on the data set, we use either (2.11) or (2.12).

The choice of $d(i,j)$ varies with the data set. If one wants to cluster points in $\mathbb{R}^n$, a reasonable choice for $d(i,j)$ is the Euclidean distance between points i and j. In the case of image processing, where the vertices are the pixels in the image, to construct $d(i,j)$, we use the concept of feature vectors, as in [8]. Each vertex i is assigned an n-dimensional feature vector, and $d(i,j)$ is then the weighted 2-norm (where each coordinate of the vector is assigned a weight) of the difference of the feature vectors of pixels i and j. More details on $d(i,j)$ in this case are given in sections 4.1 and 4.2.

2.2.2. Graph clustering and the graph Laplacian. The theory shown below justifies the use of the (thresholded) second eigenvector of L_s as an initialization when applying our classification algorithm to the two-moons data set, which will be described in section 3.1.1.

The goal of graph clustering is to partition the graph so that the weights between vertices of different groups are small and the weights between vertices within the same group are large. In this section, we deal with a binary problem only. A mincut approach to the above problem is to partition a set of vertices V into sets A and $\bar{A}$ in such a way that

$$cut(A,\bar{A}) = \sum_{x\in A, y\in \bar{A}} w(x,y) \tag{2.13}$$

is minimized. This mincut problem can be solved using an efficient algorithm like the ones in [55, 37, 38].

However, this problem leads to poor classification in many cases since the resulting "bad" partition often isolates one vertex from the rest of the set [44]. One way to overcome this problem is to use correct normalization, i.e., to force the sets A and $\bar{A}$ to be "large." Let

$$vol(A) = \sum_{x\in A} d(x). \tag{2.14}$$

Then the modified problem is to find a subset A of V such that

$$Ncut(A,\bar{A}) = \frac{cut(A,\bar{A})}{vol(A)} + \frac{cut(A,\bar{A})}{vol(\bar{A})} \tag{2.15}$$

is minimized. This is a NP-hard discrete problem [60]. One way to simplify it would be to allow the solution to take arbitrary values in $\mathbb{R}$. This leads to the following relaxed Ncut

problem:

$$\min_{A\subset Y}\langle u, L_s u\rangle, \quad u \perp D^{\frac{1}{2}}\mathbf{1}, \quad ||u||^2 = vol(Y). \tag{2.16}$$

The fact that the above problem obtains a real-valued solution instead of a discrete-valued solution, like problem (2.15), is emphasized.

The relaxed problem (2.16) has been applied to many segmentation problems; for example, appealing results are shown in [54]. To solve the above problem, one can apply the Raleigh–Ritz theorem, and the solution is given by the second eigenvector of the symmetric graph Laplacian L_s [59].

2.3. Nonlocal operators. In general, image processing methods that are local fail to produce satisfactory results on images with repetitive structures and textures because they operate only on small neighborhoods, without using any information about the whole domain. The advantage of nonlocal operators is that they contain data about the whole vertex set and are thus more successful with those types of images.

Zhou and Schölkopf in their papers [67, 64, 66, 65] formulated a theory of nonlocal operators that is related to the discrete graph Laplacian described in section 2.2. Buades, Coll, and Morel applied this nonlocal theory to denoising algorithms in their work [9]. Gilboa and Osher proposed using nonlocal operators to define functionals involving the TV seminorm for various image processing applications in their work [29].

We review nonlocal calculus below, where all definitions are continuous. Let $\Omega \in \mathbb{R}^n$, $u(x)$ be a function $u : \Omega \to \mathbb{R}$, and the nonlocal derivative be defined as

$$\frac{\partial u}{\partial y}(x) = \frac{u(y) - u(x)}{d(x,y)}, \quad x, y \in \Omega, \tag{2.17}$$

where d is some positive distance defined on the space and $0 < d(x,y) \leq \infty$ for all x, y. If the (symmetric) weight function is defined as

$$w(x,y) = \frac{1}{d(x,y)^2}, \tag{2.18}$$

the nonlocal derivative can be written as

$$\frac{\partial u}{\partial y}(x) = (u(y) - u(x))\sqrt{w(x,y)}. \tag{2.19}$$

We now consider vectors and denote them by $\vec{v} = v(x,y) \in \Omega \times \Omega$. Let $\vec{v}_1$ and $\vec{v}_2$ be two such vectors. We define the dot product and the inner product as

$$(\vec{v}_1 \cdot \vec{v}_2)(x) = \int_\Omega v_1(x,y)v_2(x,y)dy, \tag{2.20}$$

$$\langle \vec{v}_1, \vec{v}_2\rangle = \langle \vec{v}_1 \cdot \vec{v}_2, 1\rangle = \int_{\Omega\times\Omega} v_1(x,y)v_2(x,y)dxdy. \tag{2.21}$$

The magnitude of a vector can be defined as

$$|v|(x) = \sqrt{\vec{v} \cdot \vec{v}} = \sqrt{\int_\Omega v(x,y)^2 dy}, \tag{2.22}$$

while the nonlocal gradient $\bigtriangledown_w u(x) : \Omega \to \Omega \times \Omega$ is the vector of all partial derivatives:

$$(\nabla_w u)(x,y) = (u(y) - u(x))\sqrt{w(x,y)}, \qquad x, y \in \Omega. \tag{2.23}$$

With the above definitions, the nonlocal divergence $\mathrm{div}_w \vec{v}(x) : \Omega \times \Omega \to \Omega$ is defined as the adjoint of the nonlocal gradient:

$$\left(\operatorname*{div}_w \vec{v}\right)(x) = \int_\Omega (v(x,y) - v(y,x))\sqrt{w(x,y)}dy. \tag{2.24}$$

The Laplacian is now defined as

$$\Delta_w u(x) = \frac{1}{2}\operatorname*{div}_w(\nabla_w u(x)) = \int_\Omega (u(y) - u(x))w(x,y)dy. \tag{2.25}$$

Since the graph Laplacian was defined in section 2.2 as

$$Lu(x) = \sum_y w(x,y)(u(x) - u(y)), \tag{2.26}$$

one can interpret $-Lu(x)$ as a discrete approximation of $\Delta_w u$. Note that a constant of $\frac{1}{2}$ was needed here to relate the two Laplacians.

According to the nonlocal calculus described above,

$$\int_\Omega |\nabla u|^2 dx = \int_{\Omega\times\Omega} (u(y) - u(x))^2 w(x,y)dxdy. \tag{2.27}$$

Since

$$u \cdot Lu = \frac{1}{2}\sum_{x,y} w(x,y)(u(x) - u(y))^2, \tag{2.28}$$

one can consider $2u \cdot Lu$ to be the discrete graph version of $\int |\nabla u|^2 dx$.

In their paper [8], Bertozzi and Flenner replace the $\frac{\epsilon}{2}\int |\nabla u|^2 dx$ term of (2.2) by $\epsilon u \cdot Lu(x)$. However, normalization of the Laplacian is necessary (refer to the beginning of section 2.2 or to [8]), so instead they use

$$\epsilon u \cdot L_s u = \frac{\epsilon}{2}\sum_{x,y} \frac{w(x,y)(u(x) - u(y))^2}{\sqrt{d(x)d(y)}}. \tag{2.29}$$

When the variational solution u takes the values -1 or 1,

$$u \cdot L_s u = C + 4\sum_{x\in A, y\in \bar{A}} \frac{w(x,y)}{\sqrt{d(x)d(y)}} - 2\left(\sum_{x\in A, y\in A} \frac{w(x,y)}{\sqrt{d(x)d(y)}} + \sum_{x\in \bar{A}, y\in \bar{A}} \frac{w(x,y)}{\sqrt{d(x)d(y)}}\right). \tag{2.30}$$

In this case, C is a constant that varies with the graph but not with the partition. The representation shows that the above is minimized when the normalized weights between vertices of different groups are small but the normalized weights between vertices within a group are large. This is precisely the goal of graph clustering. Therefore, by replacing the $\frac{\epsilon}{2}\int |\bigtriangledown u|^2 dx$ term of (2.2) with $\epsilon u \cdot L_s u$, thus creating a graph-based version of (2.2), and then minimizing the resulting equation, one achieves the desired segmentation.

The Γ-convergence of the graph-based Ginzburg–Landau functional is investigated in [57]. The authors prove that as $\epsilon \to 0$, the limit is related to the TV seminorm and cut from (2.13).

Another important operator that arises from the need to define variational methods on graphs is the mean curvature on graphs. This nonlocal operator was introduced by Osher and Shen in [48], who defined it via graph-based p-Laplacian operators. p-Laplacian operators are a family of quasi-linear elliptic partial differential operators defined for $1 \leq p < \infty$:

$$L^p(f) = \nabla \cdot (|\nabla f|^{p-2} \nabla f). \tag{2.31}$$

In the special case $p = 2$, the p-Laplacian is just a regular Laplacian. For $p = 1$, the p-Laplacian represents curvature.

The discrete graph version of p-Laplace operators is defined in [18] as

$$L^p(u(x)) = \frac{1}{p} \sum_{(x,y)\in E} w(x,y)(\|\nabla u(x)\|^{p-2} + \|\nabla u(y)\|^{p-2})(u(x) - u(y)). \tag{2.32}$$

Note that the graph 2-Laplacian is just the graph Laplacian, which is consistent with the continuous case.

Let us now define the mean curvature on graphs—the discrete analogue of the mean curvature of the level curve of a function defined on a continuous domain of $\mathbb{R}^N$:

$$\kappa_w = \frac{1}{2} \sum_{(x,y)\in E} w(x,y) \left(\frac{1}{\|\nabla u(x)\|} + \frac{1}{\|\nabla u(y)\|} \right) (u(x) - u(y)). \tag{2.33}$$

Note that in the case of an unweighted mesh graph, κ_w becomes a numerical discretization of the mean curvature. This curvature, κ_w, is also used in [15] as a regularizer in a graph adaptation of the Chan–Vese method. In their work-in-progress [58], van Gennip et al. propose a different definition of mean curvature on graphs and prove convergence of the MBO scheme on graphs.

2.4. Review of the MBO scheme. The idea of approximating mean curvature flow using threshold dynamics was introduced in [43] by Merriman, Bence, and Osher. To explain the intuition behind the numerical scheme they propose, the authors analyze the mean curvature flow of the curve $C = \partial\Sigma$ using diffusion of the characteristic function χ of the set Σ. If one imagines an interface, such as χ, and then applies the heat equation $\chi_t = \Delta\chi$, then the diffusion blunts the sharp points on the front but has little impact on the flatter parts, thus leaving the $\chi = \frac{1}{2}$ level set invariant to diffusion. By changing the coordinates to polar form, the authors of [43] show that the $\frac{1}{2}$-level set also moves according to some curvature dependent motion. Therefore, if one diffuses the characteristic function of a set with boundary C for a short time and then identifies the boundary of the "new set" with the $\frac{1}{2}$-level set, the curve C

moves with a normal velocity that is at any given point equal to the mean curvature at that point. The above analysis is local, so the timestep needs to be short enough so that it is valid but long enough so that the curve is moving.

From the previous discussion follows the MBO numerical scheme for approximation of the motion of u by mean curvature at discrete times:

- *Step* 1 (diffusion). Let $v(x) = S(\delta t)u_n(x)$, where $S(\delta t)$ is the propagator (by time δt) of

$$\frac{\partial v}{\partial t} = \Delta v. \tag{2.34}$$

- *Step* 2 (thresholding). Set

$$u^{n+1}(x) = \begin{cases} 1 & \text{if } v(x) \geq \frac{1}{2}, \\ 0 & \text{if } v(x) < \frac{1}{2}. \end{cases}$$

2.5. Applications of the MBO scheme. We are interested in motion by mean curvature because it is closely related to the Ginzburg–Landau functional, which we use here as a regularizer. The gradient descent of the Ginzburg–Landau functional yields the Allen–Cahn equation:

$$\frac{\partial u}{\partial t} = 2\epsilon\Delta u - \frac{1}{\epsilon}W'(u). \tag{2.35}$$

Here W is the double well potential $W(u) = (u^2 - 1)^2$. It is proven in [51] that as $\epsilon \to 0^+$, the rescaled solutions $u_\epsilon(x, t/\epsilon)$ of the above equation move according to mean curvature of the interface between the -1 and 1 phases of the solutions. In addition, [3] and [23] present rigorous proofs that the MBO algorithm approximates motion by mean curvature. This implies that for the small values of ϵ, the MBO thresholding scheme can be used to numerically solve the Allen–Cahn equation.

Multiple extensions, adaptations, and applications of the MBO scheme are present in literature. We find the modification of the MBO scheme for solving the inhomogeneous Allen–Cahn equation proposed in [22] particularly interesting. To create a fast image segmentation algorithm, Esedoḡlu and Tsai propose a thresholding scheme for minimizing a diffuse interface version of the piecewise constant Mumford–Shah functional,

$$MS_\epsilon(u, c_1, c_2) = \int_D \epsilon|\nabla u|^2 + \frac{1}{\epsilon}W(u) + \lambda\{u^2(c_1 - f)^2 + (1 - u)^2(c_2 - f)^2\}dx, \tag{2.36}$$

where f is the image. The first variation of the model (2.36) yields the following gradient descent equation:

$$u_t = 2\epsilon\Delta u - \frac{1}{\epsilon}W'(u) + 2\lambda\{u(c_1 - f)^2 + (1 - u)(c_2 - f)^2\}, \tag{2.37}$$

and the adaptation of the MBO scheme is used to solve it. Esedoḡlu and Tsai propose the following scheme (similar to the MBO scheme, where the propagation step based on the heat equation is combined with thresholding):

- *Step* 1. Let $v(x) = S(\delta t)u_n(x)$, where $S(\delta t)$ is a propagator by time δt of the equation
$$w_t = \Delta w - 2\tilde{\lambda}\left(w(c_1 - f)^2 + (1 - w)(c_2 - f)^2\right)$$
with appropriate boundary conditions.
- *Step* 2. Set
$$u_{n+1}(x) = \begin{cases} 0 & \text{if } v(x) \in (-\infty, \frac{1}{2}], \\ 1 & \text{if } v(x) \in (\frac{1}{2}, \infty). \end{cases}$$

We also use an adaptation of the MBO scheme to solve our equation (1.4), but the most important difference between our method and the above approach, besides a different energy to minimize, is that we generalize to graphs.

Some other extensions of the MBO scheme appeared in [20, 21, 42]. An efficient algorithm for motion by mean curvature using adaptive grids was proposed in [52].

2.6. Computation of eigenvectors. Our method involves the computation of eigenvalues and associated eigenvectors of the symmetric graph Laplacian. In practice, one needs to compute only a fraction of the eigenvalues and eigenvectors (since eigenvectors with very small eigenvalues are not very significant computationally), and different methods of doing so are used depending on the size of the domain.

When the graph is sparse and is of moderate size, around 5000×5000 or less, we use a Rayleigh–Chebyshev procedure outlined in [1]. It is a modification of an inverse subspace iteration method that uses adaptively determined Chebyshev polynomials. The procedure is also a robust method that converges rapidly and that can handle cases when there are eigenvalues of multiplicity greater than one.

When the graph is very large, such as in the case of image classification, the Nyström extension method, to be described in the next section, is used.

2.6.1. Nyström extension for fully connected graphs. Nyström extension [8, 26, 25, 4] is a matrix completion method often used in image processing applications, such as kernel principle component analysis [17] and spectral clustering [47]. This procedure performs much faster than many alternate techniques because it uses approximations based on calculations on small submatrices of the original large matrix. When the size of the matrix becomes very large, this method is especially valuable.

Note that if λ is an eigenvalue of $\hat{W} = D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$, then $1 - \lambda$ is an eigenvalue of L_s, and the two matrices have the same eigenvectors. We formulate a method to calculate the eigenvectors and eigenvalues of $\hat{W}$ and thus of L_s.

Let w be the similarity function, λ be an eigenvalue of W, and ϕ its associated eigenvector. The Nyström method approximates the eigenvalue equation

$$\int_\Omega w(y, x)\phi(x)dx = \lambda\phi(x) \tag{2.38}$$

using a quadrature rule, a technique to find weights $c_j(y)$, and a set of L interpolation points $X = \{x_j\}$ such that

$$\sum_{j=1}^{L} c_j(y)\phi(x_j) = \int_\Omega w(y, x)\phi(x)dx + E(y), \tag{2.39}$$

where $E(y)$ represents the error in the approximation.

We use $c_j(y) = w(y, x_j)$ and choose the L interpolation points randomly from the vertex set V. Denote the set of L randomly chosen points by $X = \{x_i\}_{i=1}^{L}$ and its complement by Y. Partitioning Z into $Z = X \cup Y$ and letting $\phi_k(x)$ be the the kth eigenvector of W and λ_k its associated eigenvalue, we obtain the system of equations

$$\sum_{x_j \in X} w(y_i, x_j)\phi_k(x_j) = \lambda_k \phi_k(y_i) \qquad \forall y_i \in Y, \quad \forall k \in 1, \ldots, L. \tag{2.40}$$

This system of equations cannot be solved directly, since the eigenvectors are not known. To overcome this problem, the L eigenvectors of W are approximated using calculations involving submatrices of W.

Let W_{XY} be defined as

$$\begin{bmatrix} w(x_1, y_1) & \ldots & w(x_1, y_{N-L}) \\ \vdots & \ddots & \vdots \\ w(x_L, y_1) & \ldots & w(x_L, y_{N-L}) \end{bmatrix},$$

where W has dimension $N \times N$. The matrices W_{YX}, W_{XX}, and W_{YY} can be defined similarly. Notice that $W_{XY} = {W_{YX}}^T$. Then the matrix W can be written as

$$\begin{bmatrix} W_{XX} & W_{XY} \\ W_{YX} & W_{YY} \end{bmatrix}.$$

To calculate the eigenvalues and eigenvectors of $\hat{W}$, one must correctly normalize the above weight matrix. The correct normalization is achieved by the following calculations, where we denote by $\mathbf{1}_K$ the K-dimensional unit vector.

Let the matrices d_X and d_Y be defined as

$$\begin{aligned} d_X &= W_{XX}\mathbf{1}_L + W_{XY}\mathbf{1}_{N-L}, \\ d_Y &= W_{YX}\mathbf{1}_L + (W_{YX}W_{XX}^{-1}W_{XY})\mathbf{1}_{N-L}. \end{aligned} \tag{2.41}$$

If $A./B$ denotes componentwise division between matrices A and B, and v^T denotes the transpose of vector v, then define the matrices $\hat{W}_{XX}$ and $\hat{W}_{XY}$ as

$$\begin{aligned} \hat{W}_{XX} &= W_{XX}./(s_X s_X^T), \\ \hat{W}_{XY} &= W_{XY}./(s_X s_X^Y), \end{aligned} \tag{2.42}$$

where $s_X = \sqrt{d_X}$ and $s_Y = \sqrt{d_Y}$.

It is shown in [8] that if $\hat{W}_{XX} = B_X D B_X^T$, and if A and Γ are matrices such that

$$A^T \Gamma A = \hat{W}_{XX} + \hat{W}_{XX}^{-\frac{1}{2}} \hat{W}_{XY} \hat{W}_{YX} \hat{W}_{XX}^{-\frac{1}{2}}, \tag{2.43}$$

then the eigenvector matrix V consisting of L eigenvectors of $\hat{W}$ and thus of L_s is given by

$$\begin{bmatrix} B_X D^{\frac{1}{2}} B_X^T A \Gamma^{-\frac{1}{2}} \\ \hat{W}_{YX} B_X D^{-\frac{1}{2}} B_X^T A \Gamma^{-\frac{1}{2}} \end{bmatrix},$$

while $I - \Gamma$ contains the corresponding eigenvalues of L_s in its diagonal entries.

Therefore, the efficiency of the Nyström extension method lies with the fact that when computing the eigenvalues and eigenvectors of an $N \times N$ matrix, where N is large, it approximates them using calculations involving only much smaller matrices, the largest of which has dimension $N \times L$, where L is small.

Although this method is very efficient, there are problems when it is applied to binary image inpainting, especially when the image has a repetitive structure. This occurs because of singular or nearly singular matrices that arise in the calculations of the Nyström extension method. Therefore, in this case, we use the Rayleigh–Chebyshev procedure of [1] to calculate the eigenvalues and associated eigenvectors.

3. Classification algorithm.

We construct a new classification algorithm by proposing a different approach to minimizing (1.2) than that in [8] to obtain a more simple and efficient method that eliminates the diffuse interface parameter ϵ. Our scheme is based on a variation of the MBO scheme.

As was shown in section 2.4, for small ϵ, the MBO thresholding scheme can be used to evolve the Allen–Cahn equation to steady state. The scheme consists of two steps: a heat equation propagation step and a thresholding step.

A candidate for the threshold dynamics of (1.2) is found by splitting (1.4), which is the Allen–Cahn equation plus an extra fidelity term. There are several options, including splitting the equation into three steps, but we choose the possibility in which (1.4) is split so that the thresholding step resembles the one in the original MBO scheme, as is done in [22] and explained in section 2.5.

Therefore, our algorithm consists of alternating between the following two steps to obtain approximate solutions $u_n(x)$ at discrete times:

- *Step* 1 (heat equation with forcing term). Propagate using

$$\frac{\partial y}{\partial t} = -L_s y - C_1 \lambda(x)(y - u_0), \tag{3.1}$$

  starting with u_n. Note that C_1 can be different from the original C.
- *Step* 2 (thresholding). Set

$$u^{n+1}(x) = \begin{cases} 1 & \text{if } y(x) \geq 0, \\ -1 & \text{if } y(x) < 0. \end{cases}$$

Note that we now use 0 as the thresholding value (instead of $\frac{1}{2}$ as in the original MBO scheme), since the values of u are concentrated at -1 and 1, not 0 and 1.

We have decided to discretize (3.1) above in the following manner:

$$\frac{u^{n+1} - u^n}{dt} = -L_s u^{n+1} - C_1 \lambda(x)(u^n - u_0). \tag{3.2}$$

Note that the symmetric Laplacian is calculated implicitly. This is due to the stiffness of the operator, which is caused by a wide range of its eigenvalues. An implicit term is needed, since an explicit scheme requires that all the scales of the eigenvalues be resolved numerically. The

above scheme is used because it is the simplest scheme possible keeping the Laplacian term implicit.

The scheme is solved using the spectral decomposition of the symmetric graph Laplacian. Let $u^n = \sum_k a_k^n \phi_k(x)$ and $C_1\lambda(u^n - u_0) = \sum_k d_k^n \phi_k(x)$, where $\phi(x)$ are the eigenfunctions of the symmetric Laplacian. Using the obtained representations and (3.2), we obtain

$$a_k^{n+1} = \frac{a_k^n - dtd_k^n}{1 + dt\lambda_k}, \tag{3.3}$$

where λ_k are the eigenvalues of the symmetric graph Laplacian.

This spectral decomposition method is chosen because it is very efficient. Without it, the discrete Laplacian term by itself requires $O(N^2)$ calculations (without assuming any sparsity). However, when using spectral decomposition, we obtain the advantage of only having to calculate the first few largest eigenvalues and associated eigenvectors (as the smallest eigenvalues and associated eigenvectors become insignificant in calculations). Therefore, the discrete Laplacian term now requires only $O(NL)$ calculations, where L is the number of eigenvalues/eigenvectors calculated. In our data sets, $L \ll N$. Of course, this method is useful only if there is an efficient way to calculate the eigenvalues and eigenvectors of the symmetric Laplacian. For the results in this paper, we use two methods, described in section 2.6.

Therefore, the new algorithm consists of the following:

- *Step* 1. Create a graph from the data, choose a similarity function, and then calculate the symmetric graph Laplacian.
- *Step* 2. Calculate the eigenvectors and eigenvalues of the symmetric graph Laplacian. It is necessary only to calculate a portion of the eigenvectors.
- *Step* 3. Initialize u.
- *Step* 4. Apply the two-step scheme (to minimize the Ginzburg–Landau functional) described above for a certain number of iterations until a stopping criterion is satisfied. Use the following method:

1. Let $a_k^0 = \sum_x u_0(x)\phi_k(x)$ and $d_k^0(x) = 0$ for all x.
2. Until a stopping criterion is satisfied, do the following:
   a. Repeat for some number s of steps:
      1. $a_k^n \leftarrow \frac{a_k^n - \delta t d_k^n}{1+\delta t\lambda_k}$,
      2. $y(x) = \sum_k a_k^n \phi_k(x)$,
      3. $d_k^n = \sum_x C_1(y - u_0)(x)\phi_k(x)$.
   b. (thresholding part)

$$u^{n+1}(x) = \begin{cases} 1 & \text{if } y > 0, \\ -1 & \text{otherwise.} \end{cases}$$

   c. Let $a_k^{n+1} = \sum_x u_{n+1}(x)\phi_k(x)$ and $d_k^{n+1} = \sum_x C_1(y - u_0)(x)\phi_k(x)$.

The parameter δt is chosen using trial and error. The stopping criteria we use in our work is $\frac{||u_{new} - u_{old}||_2^2}{||u_{new}||_2^2} < \alpha = 0.0000001$.

Table 1
Classification results comparison.

	Min. time in method in [8]	Min. time in our method	# of iter. in method in [8]	# of iter. in our method
Two moons	0.105 s	0.002 s	300	40
Grass label	8 s	3.5 s	130	22
Cow label	18 s	3.5 s	274	29
Sky label	6 s	1.8 s	84	11
Voting data	0.035 s	0.002 s	400	5

3.1. Results for classification. We applied our classification algorithm on three data sets: the two moons data set, an image, and the House of Representatives voting records from 1984. A comparison of the results to those of the method of Bertozzi and Flenner in [8], which includes the same three data sets, is displayed in Table 1. The minimization time is the time needed for step 4 of the algorithm; our model and that in [8] follow similar procedures before this step. The table shows that our method significantly reduces the number of iterations and the minimization time.

The model of Bertozzi and Flenner is a recent work, whose results compare favorably to those some of the best algorithms. Thus, by favorably comparing our results to those of Bertozzi and Flenner we are also favorably comparing them to the best methods. For example, one of the algorithms Bertozzi and Flenner compare their results to is the method of Hein and Buehler in [35].

There are four parameters that were involved in this problem: the number of eigenvectors, C_1, σ, and dt. As long as the number of eigenvectors was not too small compared to the sample size, there was enough information to produce an accurate result. The fidelity term C_1 was also chosen to be relatively big so that the fidelity region is preserved. The weight matrix parameter σ was chosen so that the weights contain a wide range of numbers from 0 to 1; in other words, the situation in which most weights are very close to 0 (or 1) was avoided. The time range dt was the most difficult parameter, and its value differed for each data set. It was mostly chosen by trial and error, but in all cases it was neither too small or too big (in which case there is either no evolution in the iterations or frequent oscillations). The algorithm is relatively robust with the above conditions.

In the case of semisupervised classification, $\lambda(x)$ was set to 1 on the known region and 0 elsewhere, since our fidelity term assumed a least-squares fit on the information supplied. Note that the fidelity term allows for a small amount of misclassification in the known data.

3.1.1. Two moons. This data set was used by Bühler and Hein in [10] in relation to spectral clustering using the p-Laplacian. It is constructed from the following two half circles in $\mathbb{R}^2$ with radius one. The first half circle is centered at the origin and is in the upper half plane. The second half circle is formed by taking the lower half of the circle centered at $(1, 0.5)$. A thousand points are chosen uniformly from each of the two half circles. The two thousand points are then embedded in $\mathbb{R}^{100}$, and i.i.d. (independent and identically distributed) Gaussian noise with standard deviation 0.02 is added to each coordinate. The goal is to segment those two half circles using unsupervised binary classification. This is achieved using the mean zero constraint (applicable since clusters are of equal size).

An affinity matrix W is created using the weight function $w(i,j) = e^{-\frac{d(i,j)^2}{\sqrt{(\tau(i)\tau(j))}}}$, a weight function introduced by Zelnik-Manor and Perona in [61], where $\tau(i)$ is the Euclidean distance between point i and the Mth closest point to it, and $d(i,j)$ is the Euclidean distance between points i and j. The matrix $W(i,j)$ is made sparse by setting $W(i,j)$ equal to zero if point j is not among the Mth closest points to point i. It is then "symmetrized" by setting $W(i,j) = \max(W(i,j), W(j,i))$.

To calculate the eigenvectors, the Rayleigh–Chebyshev procedure [1] is used, since the graph is not large and Nyström extension is inefficient for sparse graphs [8].

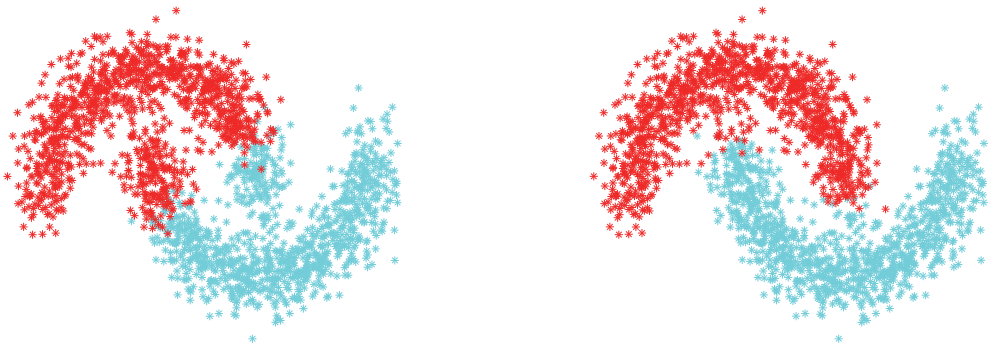
Since the problem is unsupervised binary classification (and thus no prior knowledge of class membership is assumed for any of the points), in Step 4 of the algorithm, there is no fidelity term, so $\lambda(x) = 0$ for all x. Thus, $d_k^n = 0$ for all k and n. However, since the goal is to achieve two clusters of equal size, we can use the mean zero constraint. This results in two classes of the same size. Due to the mean zero constraint, $\int u(x)dx = 0$, before thresholding, one applies the mean constraint to y by subtracting its mean from each element of y. For initialization of u, we use the sign of the second eigenvector of the symmetric Laplacian after the mean zero constraint has been applied to it. The use of such initialization was justified in section 2.2.2.

We compared our results to the method of Bertozzi and Flenner in [8] by running simulations on 35 different randomly generated two moons data sets, each taking around 2 seconds to run. The average accuracy was 96.0520% and 96.0460% for our method and the method in [8], respectively. However, 40 iterations in the minimization procedure were used, compared to 300 needed using the method in [8]. Therefore, our method resulted in a significant decrease in the number of iterations. The minimization time was also decreased from 0.105 seconds to 0.002 seconds. These results are displayed in Table 1.

We also compared our results to a spectral clustering method of thresholding the second eigenvector of L_s. The results are displayed in Figure 1. Clearly, clustering using the second eigenvector does not result in an accurate binary classification.

3.1.2. Semisupervised image labeling. We also applied our algorithm to segmenting objects in images of cows from the Microsoft image database, available from http://research.microsoft.com/en-us/projects/objectclassrecognition/. The goal was semisupervised image labeling, where two images are inputted into the algorithm, one of which has been hand segmented into two classes. The algorithm segments the second image based on the classification of the first.

A fully connected graph is constructed in this case, and the entries in the affinity matrix are calculated using feature vectors. Every pixel in the image is assigned a feature vector consisting of intensity values of pixels in its neighborhood, which was of size 7×7 in our classification tests. We use the formula $w(i,j) = e^{-\frac{d(i,j)^2}{\sigma^2}}$, where $d(i,j)$ is the weighted 2-norm of the difference of the feature vectors of pixels i and j, and we add along the three RGB channels of the image. The weighted 2-norm modifies the components of the entered vector by giving more weight to the pixels close to the original pixel and less weight to those farther away. We use a linearly decreasing kernel, where the weight decreases linearly. This construction can be used to segment different types of objects using, for example, their color



(a) Second eigenvector classification—83.75%. (b) Our method's classification—97.7%.

Figure 1. *Classification by thresholding using the second eigenvector and our method, respectively. The four parameters s (in Step 4 of our algorithm), number of eigenvectors, dt, and M (parameter in the Zelnik-Manor and Perona weight function) are set to* 3, 25, 0.725, *and* 13, *respectively.*

and texture features. Note that the weight function can be modified according to the image. For example, a weight function calculated using the spectral angle may be more effective in the segmentation of hyperspectral images.

To obtain eigenvalues and eigenvectors of L_s, the Nyström extension method is used, since the size of the graph is very large $(70,000 \times 70,000)$.

For the problem in the fidelity term, $\lambda(x)$ was set to 1 on the hand labeled image and 0 on the unlabeled image. On the hand-labeled image, we initialized u to be 1 for one class and -1 for the other class. On the unlabeled image, u_0 was set to zero.

The results are displayed in Figure 2, where it is shown that our algorithm is robust to mislabeling in the hand-labeled image. Although we do not include the results of [8] in this paper, they are very similar to ours. We are also able to capture more of the eyes and the nose of the red cow than does the method in [8]. To transfer the label for the grass, cows, and sky, our method needed about 29, 29, and 27 seconds, respectively.

The number of iterations in the minimization procedure (Step 4 of the algorithm) and minimization time as compared to the method in [8] are displayed in Table 1. The calculations show that our method significantly reduces the minimization time and the number of iterations.

3.1.3. House voting records from 1984. We applied our algorithm to a US House of Representatives voting records data set (http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records), which consists of 16 different votes from each of the 435 individuals. The goal was to assign each individual to either the Republican or the Democratic party using prior knowledge of the party affiliation of only five individuals, two Democrats and three Republicans. The votes were taken in 1984 from the 98th United States Congress, 2nd session.

(a) Original labeled image.

(b) Unlabeled image.

(c) Regions with grass label.

(d) Grass label transferred.

(e) Regions with cow label.

(f) Cow label transferred.

(g) Regions with sky label.

(h) Sky label transferred.

Figure 2. *The grass, cow, and sky labels were transferred to another image using our algorithm. The number of eigenvectors,* C_1*, and* σ *were set to* 200*,* 30*, and* 22*, respectively. The parameter dt was* 0.03*,* 0.003*, and* 0.17 *for the grass, cow, and sky label, respectively.*

An affinity matrix is constructed using calculations involving feature vectors. A 16-dimensional feature vector is assigned to each individual, consisting of his/her 16 votes. A "yes" vote is set to 1, a "no" vote is set to -1, while a "did not vote" recording is set to 0. The weight function used is $w(i,j) = e^{-\frac{d(i,j)^2}{\sigma^2}}$, where $d(i,j)$ is the 2-norm of the difference between the feature vectors of points i and j. The graph is made sparse by setting $W(i,j)$ equal to zero if point j is not among the Mth closest points to point i. The graph is then symmetrized by setting $W(i,j) = \max(W(i,j), W(j,i))$.

To calculate the eigenvectors, an SVD solver is used. In Step 4 of the algorithm, the function u is initialized to 1 for the two Democrats, -1 for the three Republicans, and 0 for the rest of the Representatives. The three Republicans were chosen to be the first, second, and eighth people in the list. The Democrats were chosen to be the third and fourth people in the list. In the fidelity term, $\lambda(x)$ was set to 1 for each of the five known individuals and 0 for the rest.

The parameters C_1 (fidelity term parameter), s (in Step 4 of our algorithm), number of eigenvectors, dt, σ, and M are set to 9.25, 3, 45, 4.675, $\sqrt{5}$, and 10, respectively.

We obtained an accuracy of 94.023%. Only 5 iterations in the minimization procedure were needed, compared to 450 iterations needed by the method in [8]. Each simulation took about 0.7 seconds, and the minimization time was decreased more than 15 times. This information is shown in Table 1.

Some of the votes predicted the party affiliation very well, i.e., above 85%. We investigated the accuracy of our algorithm when these votes were removed. With top two, top six, and top eight most predictive votes removed, our method obtained an accuracy of 90.1149%, 88.34448%, and 81.1494%, respectively. The order of the top eight predictive votes from the most predictive to least predictive is vote 4, 14, 1, 2, 15, 6, 3, and 8.

4. Image inpainting algorithm.

The problem of fitting information in the missing pixels of an image is an important inverse problem in image processing with various applications. Obviously, the goal is to produce a modified image that will look natural to an observer. The problem of inpainting may also be seen as the problem of removing occlusive objects from an image. Sparse reconstruction refers to the problem of recovering randomly distributed missing pixels.

There are numerous approaches to solving these problems in the current literature. Local TV methods became state-of-the-art techniques for image inpainting. However, since they do not perform well on images with high texture, methods that decompose images into cartoon and texture and simultaneously inpaint both have been developed [5, 53]. The problem is also solved with nonlocal inpainting methods. We are particularly interested in the nonlocal inpainting algorithm from [30] as we develop a computationally efficient nonlocal method. Some very successful nonlocal methods for inpainting and sparse reconstruction are given in [2] and [24]. Recently, the class of methods that use dictionaries of small patches that commonly appear in natural images has become increasingly popular. Those methods, besides inpainting, are also successful in denoising, as shown in [41]. In addition, a method for image inpainting using Navier–Stokes fluid dynamics is proposed in [6]. The authors use Navier–Stokes dynamics to propagate isophotes into the inpainting region, thus simulating the way painting restoration is done. Wavelets and framelets have also been successfully applied to

solve inpainting problems [16, 11].

Our semisupervised image classification algorithm can be modified to inpainting by treating the inpainted region as unclassified and the rest as the fidelity region. However, since there is no information on the inpainted region, we decided to first apply a fast, yet somewhat inaccurate, H^1 inpainting algorithm, and then use the result for weight computation. The H^1 inpainting method consists of minimizing

$$E(u) = \int |\nabla u|^2 dx + C \int \lambda(x)(u - u_0)^2 dx, \tag{4.1}$$

where $\lambda(x)$ is 0 on the inpainting region and 1 elsewhere, and u_0 is the initial state of the image. Although the latter algorithm is very fast, it does not perform well on images with high textures and repetitive structures, nor does it preserve edges [27], something that is achieved by our algorithm.

The algorithm consists of the same four steps:

- Create a graph from the data using pixels as vertices, choose a similarity function, and then create the symmetric graph Laplacian.
- Calculate the eigenvectors and eigenvalues of the symmetric graph Laplacian. It is necessary only to calculate a fraction of the eigenvectors.
- Initialize u.
- Apply the two-step scheme (to minimize the Ginzburg–Landau functional) detailed in section 2 for a certain number of iterations until a stopping criterion is satisfied.

However, there are some important differences to be discussed in sections 4.1 and 4.2.

Our algorithm is an efficient image inpainting algorithm that is able to correct images with repetitive structure or those with high texture content.

4.1. Binary image inpainting. Although the key steps of the classification algorithm remain the same when it is modified for image inpainting, there are key differences to be noted.

Before the weight matrix is calculated, H^1 inpainting is used to preprocess the image. The matrix W is then built by using a window, or a square-shaped neighborhood around a pixel. We set $W(i, j) = 0$ for all pixels j that are not in the window of pixel i. Inside the window, $W(i, j) = w(i, j)$, where the weight function is calculated in the same way as in section 3.1.2, i.e., using feature vectors and the Gaussian weight function. No updating of the matrix W is necessary in the case of binary image inpainting.

The Rayleigh–Chebyshev procedure is used to calculate the eigenvectors and eigenvalues of the graph Laplacian for binary inpainting. As mentioned before, the Nyström extension method encounters some problems when dealing with binary images.

In Step 4 of the algorithm, $\lambda(x)$ in the fidelity term is set to 0 on the inpainting region (which is given the value 0.5 on a 0 to 1 intensity scale) and to 1 on the rest of the image, while u_0 is set to 0 on the inpainting region, 1 on the white area, and -1 on the black area. The same stopping criterion is used.

4.2. Grayscale image inpainting. To generalize to gray scale inpainting, we split the signal bitwise into channels, as in [16],

$$u(x) = \sum_{m=0}^{7} u_m(x) 2^m, \tag{4.2}$$

and inpaint each channel separately. Here u_m denotes the $(m+1)$th component or digit in the binary representation of the signal, and $u_m \in \{0,1\}$ for all x.

A fully connected graph is created in the same way as in section 3.1.2. Again, we first preprocess the image using the H^1 inpainting algorithm, and use the result to build the matrix W. This weight matrix is used in all eight of the inpainting problems.

The Nyström extension method is used to calculate the eigenvalues and corresponding eigenvectors since the size of the graph is very large.

In Step 4 of the algorithm, $\lambda(x)$ in the fidelity term is set to 0 on the inpainting region and to 1 on the rest of the image. The initialization of u varies with the bit. In the inpainting region, u_0 is 0, while in the rest of the image, it is 1 on the area where the bit is 1 and -1 on the area where the bit is 0. The same stopping criterion is used, except $\alpha = 0.0001$. For some images, Step 4 is performed for a certain number of iterations.

Updating the matrix W is often necessary for grayscale inpainting, since the adjacency matrix formed from the preprocessed image using H^1 inpainting is usually not good enough to restore texture and complex patterns, as it contains "bad" regions whose values lie far from the true value. In our tests, every few iterations, the matrix is updated using the result from the last iteration as the "new image."

4.3. Inpainting results. We have tested our inpainting algorithm on both binary and grayscale images. In all cases, we compare our results to the state-of-the-art nonlocal TV inpainting using split Bregman from [62]. Our results are comparable to those achieved using the state-of-the-art method, but the timing of the run is significantly reduced (in most cases by about five times).

4.3.1. Binary image inpainting results. We applied our algorithm on an image of Barbara and one of stripes. The results and their PSNR (peak signal-to-noise ratios) are displayed in Figure 3. In both cases, the algorithm was able to recover the texture and repetitive structure present in the image, something that is unfeasible for simple algorithms such as local TV inpainting.

4.3.2. Grayscale image inpainting results. We applied our algorithm on an image of Barbara and a chessboard-like pattern. The goals ranged from removing occlusive objects, such as a flower, text, or a rectangle, to sparse reconstruction. The results along with their PSNR are displayed in Figures 4–7. Timing and iteration results are displayed in Table 2. In all cases, repetitive structure and texture were recovered.

We compare our results to local and nonlocal TV inpainting. Local TV inpainting fails to recover texture and repetitive structure. While the results of nonlocal TV inpainting are comparable to those of our method, our method is more efficient. Timing and iteration results are displayed in Table 2. We also show our method and nonlocal TV inpainting at certain iterations in Figure 8. To implement the nonlocal TV inpainting algorithm, we used the split Bregman method detailed in [62] and modified it for inpainting. The stopping condition was the same as in our inpainting algorithm, and a quick H^1 inpainting algorithm was run on the image before the weights were calculated.

(a) Original image—Barbara.

(b) Damaged image—Barbara.

(c) Nonlocal TV result—PSNR 20.0129.

(d) Our method's result—PSNR 20.6896.

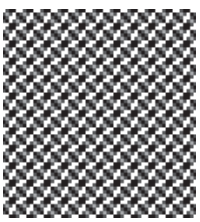
(e) Original image—stripes.

(f) Damaged image—stripes.

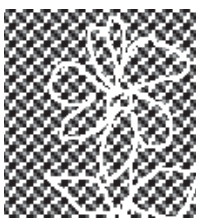
(g) Nonlocal TV result—PSNR 25.02.

(h) Our method's result—PSNR 25.0687.

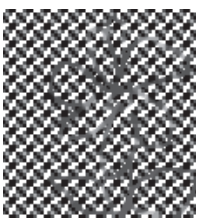
Figure 3. *Binary inpainting. For the Barbara image, we used* $C_1 = 700$, $dt = 0.003$, $\sigma = 45$, 31×31 *neighborhood for feature vector calculation,* 21×21 *window, and calculated* 400 *eigenvectors. For the image of stripes, we used* $C_1 = 700$, $dt = 0.002$, $\sigma = 45$, 17×17 *neighborhood for feature vector calculation,* 21×21 *window, and calculated* 200 *eigenvectors.*



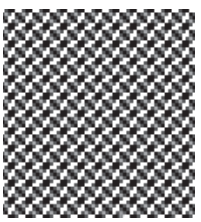
(a) Original image—pattern.



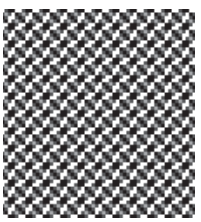
(b) Damaged image—pattern.



(c) Local TV inpainting—PSNR 16.5520.

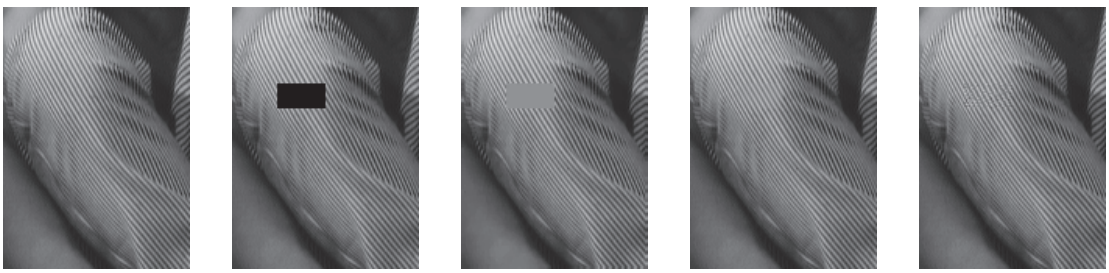


(d) Nonlocal TV inpainting—PSNR 41.3891.

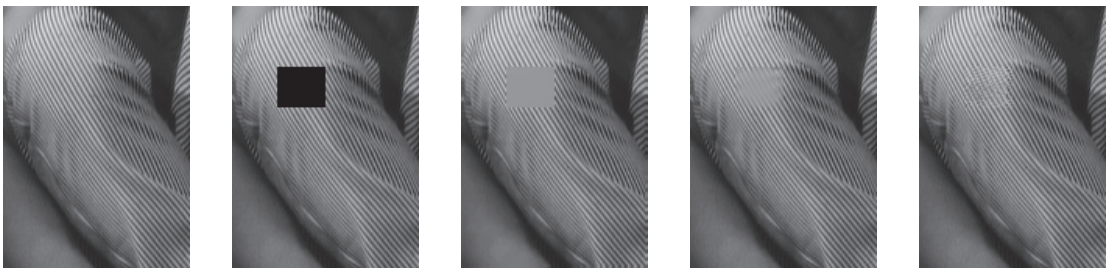


(e) Our method's result—perfect reconstruction.

Figure 4. *Pattern. We used* $C_1 = 700$, $dt = 0.005$, $\sigma = 20$, 41×41 *neighborhood for feature vector calculation, and calculated* 600 *eigenvectors. No updating of* W *was necessary.*

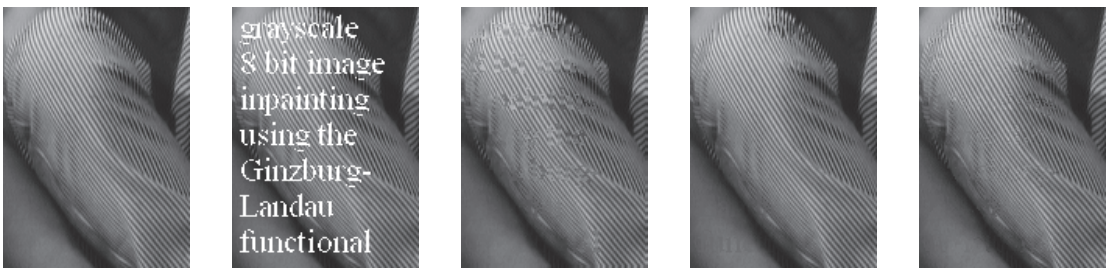


(a) Original image—Barbara. (b) Damaged image—Barbara. (c) Local TV inpainting—PSNR 32.8517. (d) Nonlocal TV inpainting—PSNR 44.1469. (e) Our method's result—PSNR 41.2848.



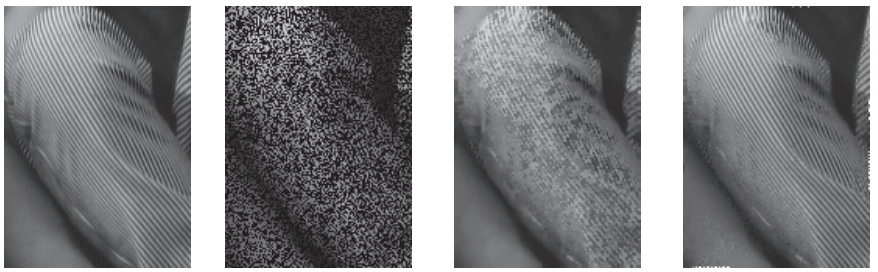
(f) Original image—Barbara. (g) Damaged image—Barbara. (h) Local TV inpainting—PSNR 31.3673. (i) Nonlocal TV inpainting—PSNR 35.0663. (j) Our method's result—PSNR 37.0315.

Figure 5. *Rectangle inpainting. For the small rectangle, we used* $C_1 = 700$, $dt = 0.01$, $\sigma = 4$, 31×31 *neighborhood for feature vector calculation, and calculated* 500 *eigenvectors. For the large rectangle, we used* $C_1 = 700$, $dt = 0.014$, $\sigma = 4$, 45×45 *neighborhood for feature vector calculation, and calculated* 500 *eigenvectors. We update* W *every iteration.*

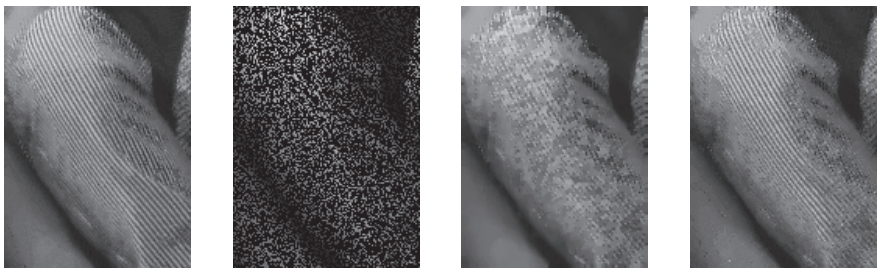


(a) Original image—Barbara. (b) Damaged image—Barbara. (c) Local TV inpainting—PSNR 29.1508. (d) Nonlocal TV inpainting—PSNR 35.6896. (e) Our method's result—PSNR 34.0688.

Figure 6. *Text inpainting. We used* $C_1 = 700$, $dt = 0.005$, $\sigma = 5$, 21×21 *neighborhood for feature vector calculation, and calculated* 500 *eigenvectors. We update* W *every other iteration.*



(a) Original image-Barbara. (b) Damaged image-Barbara. (c) Local TV inpainting-PSNR 23.6049. (d) Nonlocal TV inpainting-PSNR 27.8196.

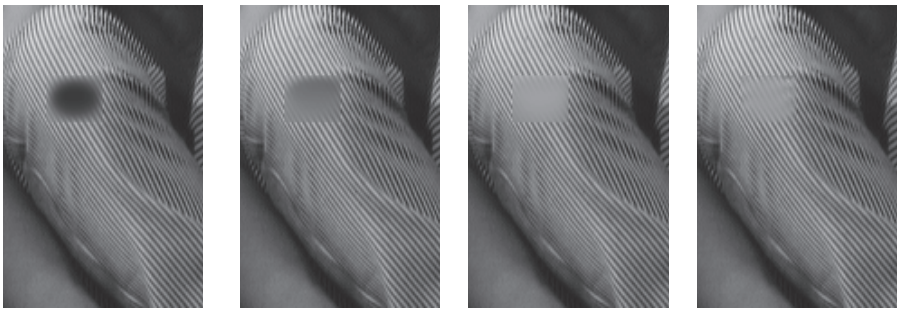


(e) Our method's result-PSNR 27.1651. (f) Damaged image-35% of the pixels removed. (g) Local TV inpainting- PSNR 22.6530. (h) Our method's result-PSNR 24.1266.

Figure 7. *50% and 35% random inpainting. For the top row, we used* $C_1 = 700$, $dt = 0.005$, $\sigma = 4$, 7×7 *neighborhood for feature vector calculation, and calculated* 400 *eigenvectors. We update* W *every iteration. For the bottom row, we used* $C_1 = 700$, $dt = 0.012$, $\sigma = 4$, 7×7 *neighborhood for feature vector calculation, and calculated* 500 *eigenvectors. We update* W *every other iteration.*

Table 2
Inpainting results comparison.

	Total time for nonlocal TV	Total time for our method	# of iterations for our method
Binary Barbara	590 s	113 s	6
Binary stripes	141 s	66 s	4
Chessboard-like pattern	266 s	48 s	2
Text inpainting	410 s	67 s	4
Small rectangle inpainting	1882 s	443 s	13
Large rectangle inpainting	3397 s	832 s	13
50% inpainting	1402 s	333 s	50

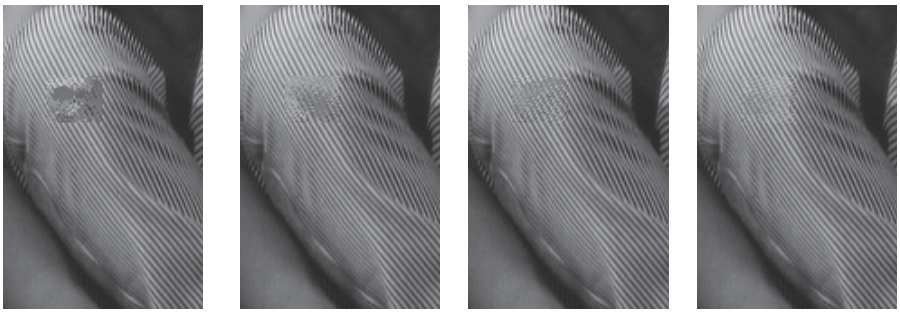


(a) Nonlocal TV, after 2 iter.—PSNR 25.7101.

(b) Nonlocal TV, after 5 iter.—PSNR 30.7031.

(c) Nonlocal TV, after 8 iter.—PSNR 33.2284.

(d) Nonlocal TV, after 13 iter.—PSNR 35.0663.



(e) Our method, after 2 iter.—PSNR 30.4406.

(f) Our method, after 5 iter.—PSNR 31.8993.

(g) Our method, after 8 iter.—PSNR 34.4851.

(h) Our method, after 13 iter.—PSNR 37.0315.

Figure 8. *Nonlocal TV inpainting and our method at certain iterations.*

5. Conclusion. This work presents an algorithm, derived from graph methods and the MBO scheme [43], that links together ideas of graphs and image processing. The results show that using threshold dynamics in combination with an efficient eigenvalue solver, such as the Nyström extension or the Raleigh–Chebyshev procedure of [1], develops an efficient method that can be applied to binary data classification or image processing. In addition, the nonlocal nature of our method allows it to be successful on images with high texture and repetitive structure.

Garcia-Cardona et al. recently extended this paper's binary classification algorithm to a multiclass method using the idea of the n-simplex; the model and the results are described in [28]. Hu et al. also built upon the ideas in this paper in [36] by describing a method based on TV for network modularity optimization using the MBO scheme.

Acknowledgments. The authors would like to thank Yanina Landa for providing a MATLAB version of the code of the algorithm in [8], and Chris Anderson for providing a code

for the Raleigh–Chebyshev procedure of [1]. In addition, we thank Arjuna Flenner, Yves van Gennip, Blake Hunter, and Jerome Darbon for useful discussions regarding this work.

REFERENCES

[1] C. Anderson, *A Raleigh-Chebyshev procedure for finding the smallest eigenvalues and associated eigenvectors of large sparse Hermitian matrices*, J. Comput. Phys., 229 (2010), pp. 7477–7487.
[2] P. Arias, V. Caselles, and G. Sapiro, *A variational framework for nonlocal image inpainting*, in Proceedings of EMMCVPR 2009, Bonn, Germany, 2009, pp. 345–358.
[3] G. Barles and C. Georgelin, *A simple proof of convergence for an approximation scheme for computing motions by mean curvature*, SIAM J. Numer. Anal., 32 (1995), pp. 484–500.
[4] S. Belongie, C. Fowles, F. Chung, and J. Malik, *Partitioning with indefinite kernels using the Nyström extension*, in Proceedings of the European Conference on Computer Vision, Copenhagen, 2002.
[5] M. Bertalmio, L. Vese, G. Sapiro, and S. Osher, *Simultaneous structure and texture inpainting*, IEEE Trans. Image Process., 12 (2003), pp. 882–889.
[6] A. Bertozzi, M. Bertalmio, and G. Sapiro, *Navier-Stokes fluid dynamics and image and video inpainting*, in Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, 2001, pp. 355–362.
[7] A. Bertozzi, S. Esedoḡlu, and A. Gillette, *Inpainting by the Cahn-Hilliard equation*, IEEE Trans. Image Process., 16 (2007), pp. 285–291.
[8] A. Bertozzi and A. Flenner, *Diffuse interface models of graphs for classification of high dimensional data*, Multiscale Model. Simul., 10 (2012), pp. 1090–1118.
[9] A. Buades, B. Coll, and J.-M. Morel, *A non-local algorithm for image denoising*, in Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, 2005, vol. 2, pp. 60-65.
[10] T. Bühler and M. Hein, *Spectral clustering based on the graph p-Laplacian*, in Proceedings of the 26th International Conference on Machine Learning, 2009, pp. 81–88.
[11] J.-F. Cai, R. Chan, and Z. Shen, *A framelet based image inpainting algorithm*, Appl. Comput. Harmon. Anal., 24 (2008), pp. 131–149.
[12] T. Chan and L. Vese *Active contours without edges*, IEEE Trans. Image Process., 10 (2001), pp. 266–277.
[13] F. Chung, *Spectral Graph Theory*, CBMS Reg. Conf. Ser. Math. 92, Providence, RI, 1997.
[14] T. Cour, F. Benezit, and J. Shi, *Spectral segmentation with multiscale graph decomposition*, in Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, 2005, vol. 2, pp. 1124–1131.
[15] X. Desquesnes, A. Elmoataz, and O. Lezoray, *PDEs level sets on weighted graphs*, in Proceedings of the 18th IEEE International Conference on Image Processing (ICIP), 2011, pp. 3377–3380.
[16] J. Dobrosotskaya and A. Bertozzi, *A wavelet-Laplace variational technique for image deconvolution and inpainting*, IEEE Trans. Image Process, 17 (2008), pp. 657–663.
[17] P. Drineas and M.W. Mahoney, *On the Nyström method for approximating a Gram matrix for improved kernel-based learning*, J. Mach. Learn. Res., 6 (2005), pp. 2153–2175.
[18] A. Elmoataz, O. Lezoray, and S. Bougleux, *Nonlocal discrete regularization on weighted graphs: A framework for image and manifold processing*, IEEE Trans. Image Process., 17 (2008), pp. 1047–1060.
[19] S. Esedoḡlu and R. March, *Segmentation with depth but without detecting junctions*, J. Math. Imaging Vision, 18 (2003), pp. 7–15.
[20] S. Esedoḡlu, S.J. Ruuth, and R. Tsai, *Diffusion generated motion using signed distance functions*, J. Comput. Phys., 229 (2010), pp. 1017–1042.
[21] S. Esedoḡlu, S.J. Ruuth, and R. Tsai, *Threshold dynamics for high order geometric motions*, Interfaces Free Bound., 10 (2008), pp. 263–282.
[22] S. Esedoḡlu and Y.R. Tsai, *Threshold dynamics for the piecewise constant Mumford-Shah functional*, J. Comput. Phys., 26 (2004), pp. 367–384.
[23] L.C. Evans, *Convergence of an algorithm for mean curvature motion*, Indiana Univ. Math. J., 42 (1993), pp. 553–557.

[24] G. Facciolo, P. Arias, V. Caselles, and G. Sapiro, *Exemplar-based interpolation of sparsely sampled images*, in Proceedings of the Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition, Bonn, Germany, 2009, pp. 331–344.
[25] C. Fowlkes, S. Belongie, and J. Malik, *Spectral grouping using the Nyström method*, IEEE Trans. Pattern Anal. Mach. Intell., 28 (2006), pp. 469–475.
[26] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, *Efficient spatiotemporal grouping using the Nyström method*, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, HI, 2001, pp. 214–225.
[27] C. Frohn-Schauf, S. Henn, and K. Witsch, *Nonlinear multigrid methods for total variation image denoising*, Comput. Vis. Sci., 7 (2004), pp. 199–206.
[28] C. Garcia-Cardona, E. Merkurjev, A.L. Bertozzi, A. Flenner, and A. Percus, *Fast multiclass segmentation using diffuse interface methods on graphs*, IEEE Trans. Pattern Anal. Mach. Intell., submitted.
[29] G. Gilboa and S. Osher, *Nonlocal operators with applications to image processing*, Multiscale Model. Simul., 7 (2008), pp. 1005–1028.
[30] G. Gilboa and S. Osher, *Nonlocal linear image regularization and supervised segmentation*, Multiscale Model. Simul., 6 (2007), pp. 595–630.
[31] T. Goldstein and S. Osher, *The split Bregman method for L1-regularized problems*, SIAM J. Imaging Sci., 2 (2009), pp. 323–343.
[32] T. Goldstein, X. Bresson, and S. Osher, *Geometric applications of the split Bregman method: Segmentation and surface reconstruction*, J. Sci. Comput., 45 (2010), pp. 272–293.
[33] L. Grady and E.L. Schwartz, *Isoperimetric graph partitioning for image segmentation*, IEEE Trans. Pattern Anal. Mach. Intell., 28 (2006), pp. 469–475.
[34] A. Harten, *High resolution schemes for hyperbolic conservation laws*, J. Comput. Phys., 49 (1983), pp. 357–393.
[35] H. Hein and T. Buehler, *An inverse power method for nonlinear eigenproblems with applications in 1-spectral clustering and sparse PCA*, Adv. Neural Inform. Process. Syst., 23 (2010), pp. 847–855.
[36] H. Hu, T. Laurent, M.A. Porter, and A.L. Bertozzi, *A method based on total variation for network modularity optimization using the MBO scheme*, SIAM J. Appl. Math., (2013), submitted.
[37] D. Karger, and C. Stein, *A new approach to the minimum cut problem*, J. ACM, 43 (1996), pp. 601–640.
[38] D. Karger, *Minimum cuts in near-linear time*, J. ACM, 47 (2000), pp. 46–76.
[39] R.V. Kohn and P. Sternberg, *Local minimizers and singular perturbations*, Proc. Roy. Soc. Edinburgh Sect. A, 11 (1989), pp. 69–84.
[40] Y. Lou, X. Zhang, S. Osher, and A. Bertozzi, *Image recovery via nonlocal operators*, J. Sci. Comput., 42 (2010), pp. 185–197.
[41] J. Mairal, M. Elad, and G. Sapiro, *Sparse representation for color image restoration*, IEEE Trans. Image Process., 17 (2008), pp. 53–69.
[42] B. Merriman and S.J. Ruuth, *Diffusion generated motion of curves on surfaces*, J. Comput. Phys., 225 (2007), pp. 2267–2282.
[43] B. Merriman, J. Bence, and S. Osher, *Diffusion generated motion by mean curvature*, in Proceedings of the Computational Crystal Growers Workshop, Providence, RI, 1992, pp. 73–83.
[44] E. Mezuman and Y. Weiss, *Globally optimizing graph partitioning problems using message passing*, in Proceedings of the 15th International Conference on Artificial Intelligence and Statistics, 22, 2012, pp. 770–778.
[45] B. Mohar, *The Laplacian spectrum of graphs*, Graph Theory Combin. Appl., 2 (1991), pp. 871–898.
[46] D. Mumford and J. Shah *Optimal approximations by piecewise smooth functions and associated variational problems*, Comm. Pure Appl. Math., 42 (1989), pp. 577–685.
[47] M.M. Naeini, G. Dutton, K. Rothley, and G. Mori, *Action recognition of insects using spectral clustering*, in Proceedings of the IAPR Conference on Machine Vision Applications, 2007, pp. 1–4.
[48] S.Osher and J. Shen, *Digitalized PDE method for data restoration*, in Analytical-Computational Methods in Applied Mathematics, E.G.A. Anastassiou, ed., Chapman & Hall/CRC, New York, 2000, pp. 751–771.
[49] G. Peyre, S. Bougleux, and L. Cohen, *Non-local regularization of inverse problems*, in Proceedings of the European Conference on Computer Vision (ECCV 2008), Springer, Berlin, 2008, pp. 57–68.

[50] C.A. Ratanamahatana and D. Gunopulos, *Scaling up the naive Bayesian classifier: Using decision trees for feature selection*, in Proceedings of the IEEE International Conference on Data Mining (ICDM 2002), Maebashi, Japan, 2002, pp. 475–487.
[51] J. Rubinstein, P. Sternberg, and J.B. Keller, *Fast reaction, slow diffusion, and curve shortening*, SIAM J. Appl. Math., 49 (1989), pp. 116–133.
[52] S.J. Ruuth, *Efficient algorithms for diffusion-generated motion by mean curvature*, J. Comput. Phys., 144 (1998), pp. 603–625.
[53] H. Schaeffer and S. Osher, *A low patch-rank interpretation of texture*, SIAM J. Imaging Sci., 6 (2013), pp. 226–262.
[54] J. Shi and J. Malik, *Normalized cuts and image segmentation*, IEEE Trans. Pattern Anal. Mach. Intell., 22 (2000), pp. 888–905.
[55] M. Stoer and F. Wagner, *A simple min-cut algorithm*, J. ACM, 44 (1997), pp. 585–591.
[56] A. Szlam and X. Bresson, *Total variation-based graph clustering algorithm for Cheeger ratio cuts*, in Proceedings of the 27th International Conference on Machine Learning, 2010, pp. 1039–1046.
[57] Y. van Gennip and A.L. Bertozzi, *Γ-convergence of graph Ginzburg–Landau functionals*, Adv. Differential Equations, 17 (2012), pp. 1115–1180.
[58] Y. van Gennip, N. Guillen, B. Osting, and A.L. Bertozzi, *Mean curvature, threshold dynamics, and phase field theory on finite graphs*, (2013) (journal unknown).
[59] U. von Luxburg, *A tutorial on spectral clustering*, Statist. Comput., 17 (2007), pp. 395–416.
[60] D. Wagner and F. Wagner, *Between min cut and graph bisection*, in Mathematical Foundations of Computer Science 1993, Lecture Notes in Comput. Sci. 711, Springer, New York, 1993, pp. 744–750.
[61] L. Zelnik-Manor and P. Perona, *Self-tuning spectral clustering*, Adv. Neutral Inform. Process. Syst., 17 (2004), pp. 1601–1608.
[62] X. Zhang, M. Burger, X. Bresson, and S. Osher, *Bregmanized nonlocal regularization for deconvolution and sparse reconstruction*, SIAM J. Imaging Sci., 3 (2010), pp. 253–276.
[63] X. Zhang and T. Chan, *Wavelet inpainting by nonlocal total variation*, Inverse Problems Imag., 4 (2010), pp. 1–20.
[64] D. Zhou and B. Schölkopf, *Regularization on discrete spaces*, in Pattern Recognition, Springer, Berlin, Germany, 2005, pp. 361–368.
[65] D. Zhou and B. Schölkopf, *Discrete regularization*, in Semi-Supervised Learning, MIT Press, Cambridge, MA, 2006, pp. 221–232.
[66] D. Zhou, J. Huang, and B. Schölkopf, *Learning from labeled and unlabeled data on a directed graph*, in Proceedings of the 22nd International Conference on Machine Learning, 2005, pp. 1041–1048.
[67] D. Zhou, B. Schölkopf, and T. Hofmann, *Semi-supervised learning on directed graphs*, Adv. Neutral Inform. Process. Syst., 17 (2005), pp. 1633–1640.