# AN MBO SCHEME ON GRAPHS FOR SEGMENTATION AND IMAGE PROCESSING

EKATERINA MERKURJEV, TIJANA KOSTIĆ, ANDREA L. BERTOZZI*

**Abstract.** In this paper we present a computationally efficient algorithm utilizing a fully or semi nonlocal graph Laplacian for solving a wide range of learning problems in data clustering and image processing. Combining ideas from $L^1$ compressive sensing, image processing and graph methods, the diffuse interface model based on the Ginzburg-Landau functional was recently introduced to the graph community for solving problems in data classification. Here, we propose an adaptation of the classic numerical Merriman-Bence-Osher (MBO) scheme for graph-based methods and also make use of fast numerical solvers for finding eigenvalues and eigenvectors of the graph Laplacian. We present various computational examples to demonstrate the performance of our model, which is successful on images with texture and repetitive structure due to its nonlocal nature.

**Key words.** Image processing, Nyström extension, Ginzburg-Landau functional, MBO scheme

## 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 [12, 37], and the graph Laplacian, which is discussed in more detail in section 1.2, is introduced as one of the fundamental concepts. In imaging, spectral methods are often used in image segmentation applications as shown in [43, 29, 13].

We are particulary 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, 26], but were successfully applied to many other image processing problems such as inpainting and reconstruction in [27, 51, 39], image deblurring in [32] and manifold processing in [16].

As an alternative to $L^1$ compressed sensing methods, Bertozzi and Flenner introduce a graph-based model based on the Ginzburg-Landau functional in their work [8]. To define the functional on a graph, the spatial gradient is replaced by a more general graph gradient operator. Analogous to the continuous case, the first variation of the model yields a gradient descent equation with the graph Laplacian, which is then solved by a numerical scheme with convex splitting. To reduce the dimension of the graph Laplacian and make the computation more efficient, the authors propose the Nyström extension method [23] to approximate eigenvalues and the corresponding eigenvectors of the graph Laplacian.

Many applications suggest that the MBO scheme of Merriman, Bence and Osher [35] for approximating the motion by mean curvature performs very well in minimizing functionals built around the Ginzburg-Landau functional. For example, the authors of [20] propose an adaptation of the scheme to solve the piecewise constant Mumford-Shah functional. This inspired us to adapt the MBO scheme [35] for solv-

*Mathematics Department, UCLA, Box 951555, Los Angeles 90095-1555, USA ing graph based equations to create an algorithm that achieves faster convergence through a small number of computationally inexpensive iterations. In this paper, we apply our algorithm to solve various problems in data clustering, segmentation, object recognition and inpainting.

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 segmentation and inpainting in sections 2 and 3, respectively, show results and include comparisons to some of the recent methods. The advantage of this new method is its speed and its ability to recover texture and repetitive structure in an image.

## 1.1. The Motivation for the New Algorithm.

### 1.1.1. Ginzburg-Landau Functional.

Many papers, such as [26], show that the total variation (TV) semi-norm

$$||u||_{TV} = \int_\Omega |\nabla u| dx \tag{1.1}$$

has been used successfully in many image processing applications. It has also been applied to numerical analysis of differential equations [30].

A proof in [31] shows that the TV semi-norm is the limit in the sense of $\Gamma$-convergence of the following Ginzburg-Landau functional

$$GL(u) = \frac{\epsilon}{2}\int |\nabla u|^2 dx + \frac{1}{\epsilon}\int W(u) dx \tag{1.2}$$

where $W(u)$ 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.

Therefore, one can write

$$GL(u) \to_\Gamma C|u|_{TV}. \tag{1.3}$$

This convergence allows the two functionals to be interchanged in some cases. One might prefer to use the GL functional instead of the TV semi-norm since its highest order term is purely quadratic which allows for efficient minimization procedures. In contrast, minimization of the TV semi-norm leads to a nonlinear curvature term, making it less trivial to solve numerically. However, recent advances, such as the split Bregman method described in [28], have made progress in such problems.

Due to its connection to the TV semi-norm, the Ginzburg-Landau functional has also often been used in image processing and in various image processing applications, such as inpainting [14, 7] and segmentation [17, 20]. In practice, one would minimize

$$E(u) = GL(u) + F(u, u_0) \tag{1.4}$$

where $F$ is the fidelity term. 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 $-1$, 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".

#### 1.1.2. Bertozzi/Flenner Algorithm.

In their work [8], Bertozzi and Flenner propose a segmentation algorithm by minimizing the Ginzburg-Landau functional with a fidelity term

$$E(u) = \frac{\epsilon}{2}\int |\nabla u|^2 dx + \frac{1}{\epsilon}\int W(u)dx + F(u, u_0). \tag{1.5}$$

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 1.2 and 1.3, so that

$$E(u) = \epsilon u \cdot L_s u + \frac{1}{\epsilon}\int W(u)dx + \int F(u, u_0). \tag{1.6}$$

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

$$\frac{\partial u}{\partial t} = -\epsilon L_s u - \frac{1}{\epsilon}W'(u) - \frac{\partial F}{\partial u}. \tag{1.7}$$

Note that this is just the Allen-Cahn equation with fidelity term with $\Delta u$ replaced by a graph operator term $-L_s$, to be explained in sections 1.2 and 1.3. Taking $F$ to be $\frac{1}{2}C\lambda(x)(u - u_0)^2$ for some constant $C$, one obtains

$$\frac{\partial u}{\partial t} = -\epsilon L_s u - \frac{1}{\epsilon}W'(u) - C\lambda(x)(u - u_0). \tag{1.8}$$

The authors then describe a numerical scheme involving convex splitting to evolve equation (1.8) to steady state.

The main purpose of this paper is to develop a fast and simple method for minimizing (1.6) in the small $\epsilon$ limit. The algorithm is discussed in section 2, after the sections 1.2 - 1.4 on the relevant background.

### 1.2. Background on graphs.

In this paper, to create a nonlocal method, we generalize to the theory of graphs, described in [12]. 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

$$d(i) = \sum_{j \in V} w(i, j) \tag{1.9}$$

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

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

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

$$Lu(i) = \sum_j w(i,j)(u(i) - u(j)) \tag{1.10}$$

$$u \cdot Lu = \frac{1}{2} \sum_{i,j} w(i,j)(u(i) - u(j))^2 \tag{1.11}$$

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

$$L_s = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \tag{1.12}$$

that satisfies

$$u \cdot L_s u = \frac{1}{2} \sum_{i,j} \frac{w(i,j)(u(i) - u(j))^2}{\sqrt{d(i)d(j)}} \qquad \forall u \in \mathbb{R}^n. \tag{1.13}$$

It is also a symmetric matrix.

Another version that is commonly used is the random walk Laplacian

$$L_w = D^{-1} L = I - D^{-1} W \tag{1.14}$$

which is related to Markov processes. More detail about normalized Laplacians is given in [12] and [47].

**1.2.1. Graph clustering and the graph Laplacian.** 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 so that

$$\mathit{cut}(A, \bar{A}) = \sum_{x \in A, y \in \bar{A}} w(x,y) \tag{1.15}$$

is minimized. This mincut problem is solved using an efficient algorithm in [44].

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 [36]. 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

$$vol(A) = \sum_{x \in A} d(x). \tag{1.16}$$

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

$$Ncut(A, \bar{A}) = \frac{cut(A, \bar{A})}{vol(A)} + \frac{cut(A, \bar{A})}{vol(\bar{A})} \tag{1.17}$$

is minimized. This is a NP hard discrete problem [48]. 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:

$$min_{A \subset Y} \langle u, L_s u \rangle, \qquad u \perp D^{\frac{1}{2}} \mathbf{1}, \qquad ||u||^2 = vol(Y). \tag{1.18}$$

The fact that the above problem obtains a real-valued solution instead of a discrete-valued solution, like problem (1.17), is emphasized.

The relaxed problem (1.18) has been applied to many segmentation problems; for example, appealing results are shown in [43]. 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 $L_s$ [47].

The theory shown above justifies the use of the (thresholded) second eigenvector of $L_s$ as an initialization when applying our segmentation algorithm to the two-moons data set, which will be described in section 2.3.1.

## 1.3. Nonlocal Operators.

In general, image processing methods that are local fail to produce satisfactory results on images with repetitive structures and textures because they only operate 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 [55, 52, 54, 53] formulated a theory of nonlocal operators that is related to the discrete graph Laplacian described in section 1.2. Buades, Coll and Morel applied this nonlocal theory to denoising algorithms in their work [9]. Osher and Gilboa proposed using nonlocal operators to define functionals involving the TV semi-norm for various image processing applications in their work [26].

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

$$\frac{\partial u}{\partial y}(x) = \frac{u(y) - u(x)}{d(x,y)}, \qquad x, y \in \Omega \tag{1.19}$$

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

$$w(x,y) = \frac{1}{d(x,y)^2}, \tag{1.20}$$

the nonlocal derivative can be written as

$$\frac{\partial u}{\partial y}(x) = (u(y) - u(x))\sqrt{w(x,y)}. \tag{1.21}$$

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

$$(\vec{v}_1 \cdot \vec{v}_2)(x) = \int_\Omega v_1(x,y)v_2(x,y)dy \tag{1.22}$$

$$\langle \vec{v}_1, \vec{v}_2 \rangle = \langle \vec{v}_1 \cdot \vec{v}_2, 1 \rangle = \int_{\Omega\times\Omega} v_1(x,y)v_2(x,y)dxdy \tag{1.23}$$

The magnitude of a vector can be defined as

$$|v|(x) = \sqrt{\vec{v}\cdot\vec{v}} = \sqrt{\int_\Omega v(x,y)^2 dy}. \tag{1.24}$$

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

$$(\nabla_w u)(x,y) = (u(y) - u(x))\sqrt{w(x,y)}, \qquad x, y \in \Omega. \tag{1.25}$$

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

$$(div_w \vec{v})(x) = \int_\Omega (v(x,y) - v(y,x))\sqrt{w(x,y)}dy. \tag{1.26}$$

The Laplacian is now defined as

$$\Delta_w u(x) = \frac{1}{2} div_w(\nabla_w u(x)) = \int_\Omega (u(y) - u(x))w(x,y)dy. \tag{1.27}$$

Since the graph Laplacian was defined in section 1.2 as

$$Lu(x) = \sum_y w(x,y)(u(x) - u(y)) \tag{1.28}$$

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,

$$\int_\Omega |\nabla u|^2 dx = \int_{\Omega\times\Omega} (u(y) - u(x))^2 w(x,y)dxdy. \tag{1.29}$$

Since

$$u \cdot Lu = \frac{1}{2}\sum_{x,y} w(x,y)(u(x) - u(y))^2 \tag{1.30}$$

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 $\frac{\epsilon}{2}\int|\nabla u|^2dx$ term of the Ginzburg-Landau function by $\epsilon u \cdot Lu(x)$. However, normalization of the Laplacian is necessary, so instead they use

$$\epsilon u \cdot L_s u = \frac{\epsilon}{2}\sum_{x,y}\frac{wx,y)(u(x)-u(y))^2}{\sqrt{d(x)d(y)}}. \tag{1.31}$$

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

$$u \cdot L_s u = C + 4\sum_{x\in A, y\in \bar{A}}\frac{w(x,y)}{\sqrt{d(x)d(y)}} - 2\left(\sum_{x\in A,y\in A}\frac{w(x,y)}{\sqrt{d(x)d(y)}} + \sum_{x\in\bar{A},y\in\bar{A}}\frac{w(x,y)}{\sqrt{d(x)d(y)}}\right) \tag{1.32}$$

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 $\frac{\epsilon}{2}\int|\bigtriangledown u|^2dx$ term in the Ginzburg-Landau functional with $\epsilon u \cdot L_s u$, thus creating a graph based version of the functional, and then minimizing the resulting equation, one achieves the desired segmentation.

The $\Gamma$-convergence of the graph based Ginzburg-Landau functional is investigated in [46]. The authors prove that as $\epsilon \to 0$, the limit is related to the total variation semi-norm and cut from (1.15).

**1.4. Review of MBO scheme for differential operators.** The idea to approximate mean curvature flow using threshold dynamics is introduced in the paper [35] 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 $C$ using diffusion of the characteristic function $\chi$ of the set $\Sigma$, where $\partial\Sigma = C$. If one imagines an interface, such as $\chi$, and then applies the heat equation $\chi_t = \Delta\chi$, then the diffusion blunts the sharp points on the boundary, 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 [35] 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**. Let $v(x) = S(\delta t)u_n(x)$ where $S(\delta t)$ is the propagator (by time $\delta t$) of

$$\frac{\partial v}{\partial t} = \Delta v \tag{1.33}$$

- **Step 2**. Set

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

We are interested in motion by mean curvature because it is closely related to the Allen-Cahn equation

$$\frac{\partial u}{\partial t} = 2\epsilon \Delta u - \frac{1}{\epsilon} W'(u) \tag{1.34}$$

obtained by applying the method of gradient descent to the Ginzburg-Landau functional. Here $W$ is the double well potential $W(u) = (u^2 - 1)^2$. It is proven in [40] 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 [21] present rigorous proofs that the MBO algorithm approximates motion by mean curvature. This implies that for the small values of $\epsilon$, the MBO thresholding scheme can be used to numerically solve the Allen-Cahn equation. Note that if one uses a time splitting scheme to solve the equation, the second step is propagation using

$$\frac{\partial u}{\partial t} = -\frac{1}{\epsilon} W'(u) \tag{1.35}$$

which turns into thresholding (second step of the MBO scheme) as $\epsilon \to \infty$.

Multiple extensions, adaptations and applications of the MBO scheme are present in literature. In their work [20], Esedoglu and Tsai propose a thresholding scheme for minimizing the piecewise constant Mumford-Shah functional of image segmentation. The authors also propose a generalization of their binary segmentation method that successfully solves a multi-phase segmentation problem. Some other extensions of the MBO scheme appeared in [18, 19, 34]. An efficient algorithm for motion by mean curvature using adaptive grids was proposed in [41].

## 2. Segmentation Algorithm.

We construct a new segmentation algorithm by proposing a different approach to minimize (1.6) than the one in [8] to obtain a more simple and efficient method that eliminates the diffuse interface parameter $\epsilon$. Our scheme is based on a variation of the MBO scheme.

As was shown in section 1.4, for small $\epsilon$, 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 (1.6) is found by splitting equation (1.8), 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 equation (1.8) is split so that the thresholding step resembles the one in the original MBO scheme, as is done in [20].

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) Let $y(x)$= S$(\delta t)u_n(x)$ where S$(\delta t)$ is the propagator (by time $\delta t$) of

$$\frac{\partial z}{\partial t} = -L_s z - C_1 \lambda(x)(z - z_0) \tag{2.1}$$

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 (2.1) above in the following manner:

$$\frac{u^{n+1} - u^n}{dt} = -L_s u^{n+1} - C_1 \lambda(x)(u^n - u_0). \tag{2.2}$$

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 all the scales of the eigenvalues to be resolved numerically.

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 equation (2.2), we obtain

$$a_k^{n+1} = \frac{a_k^n - dt d_k^n}{1 + dt \lambda_k} \tag{2.3}$$

where $\lambda_k$ are the eigenvalues of the symmetric graph Laplacian.

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 only necessary 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)
        $$u^{n+1}(x) = \begin{cases} 1, & \text{if } y > 0 \\ -1, & otherwise \end{cases}$$
    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 $\delta t$ is chosen using trial and error. The stopping criteria we use in our work is $\frac{||u_{new} - u_{old}||_2^2}{||u_{new}||_2^2} < \alpha = 0.0000001$.

**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 so that two vertices that are heavily weighted by $w$, *i.e.* $w(i,j)$ is large, are also closely related in the data. Although, several options for $w$ are discussed in [47], the choice depends on the problem, so no general theory can be formulated.

One popular choice for the similarity function is the Gaussian function

$$w(i,j) = e^{-\frac{d(i,j)^2}{\sigma^2}} \tag{2.4}$$

where $D(i,j)$ is some distance measure between the two vertices $i$ and $j$, and $\sigma$ is a parameter to be chosen. Von Luxburg in [47] explains that $\sigma$ 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 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 [49]:

$$w(i,j) = e^{-\frac{d(i,j)^2}{\sqrt{\tau(i)\tau(j)}}} \tag{2.5}$$

where the local parameter $\sqrt{\tau(i)} = d(i,k)$ and $k$ is the $M^{th}$ closest vertex to vertex $i$. As noted in [8], one should use this similarity function for segmentation when there exist multiple scales to be segmented. In [49], $M$ is chosen to be 7, while in [45], it is 10. Depending on the data set, we use either (2.4) or (2.5).

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 a $n$-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 3.1 and 3.2.

## 2.2. Computation of Eigenvectors.

Our method involves the computation of eigenvalues and associated eigenvectors of the symmetric graph Laplacian. In practice, one computes only a fraction of the eigenvalues and eigenvectors, 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 $5000 \times 5000$ 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 segmentation, the Nyström extension method, to be described in the next section, is used.

### 2.2.1. Nyström extension for fully connected graphs.

Nyström extension [8, 24, 23, 4] is a matrix completion method often used in many image processing applications, such as kernel principle component analysis [15] and spectral clustering [38]. 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 $\lambda$ 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, $\lambda$ be an eigenvalue of $W$, and $\phi$ its associated eigenvector. The Nyström method approximates the eigenvalue equation

$$\int_\Omega w(y,x)\phi(x)dx = \lambda\phi(x) \tag{2.6}$$

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

$$\sum_{j=1}^{L} c_j(y)\phi(x_j) = \int_\Omega w(y,x)\phi(x)dx + E(y) \tag{2.7}$$

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 $Y$. Partioning $Z$ into $Z = X \cup Y$ and letting $\phi_k(x)$ be the the $k^{th}$ eigenvector of $W$ and $\lambda_k$ its associated eigenvalue, we obtain the system of equations

$$\sum_{x_j \in X} w(y_i, x_j)\phi_k(x_j) = \lambda_k\phi_k(y_i) \qquad \forall y_i \in Y, \quad \forall k \in 1, ..., L. \tag{2.8}$$

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

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

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 eigenvalues 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

$$\begin{aligned} d_X &= W_{XX}\mathbf{1}_L + W_{XY}\mathbf{1}_{N-L} \\ d_Y &= W_{YX}\mathbf{1}_L + (W_{YX}W_{XX}^{-1}W_{XY})\mathbf{1}_{N-L} \end{aligned} \tag{2.9}$$

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

$$\begin{aligned} \hat{W}_{XX} &= W_{XX}./(s_X s_X^T) \\ \hat{W}_{XY} &= W_{XY}./(s_X s_X^Y) \end{aligned} \tag{2.10}$$

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 $\Gamma$ are matrices such that

$$A^T \Gamma A = \hat{W}_{XX} + \hat{W}_{XX}^{-\frac{1}{2}} \hat{W}_{XY} \hat{W}_{YX} \hat{W}_{XX}^{-\frac{1}{2}} \tag{2.11}$$

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.

**2.3. Results for Segmentation.** We applied our segmentation 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] is displayed in tables 2.1 and 2.2. The tables show that our method significantly reduces the number of iterations and the minimization time.

**2.3.1. Two Moons.** This data set was used by Bühler *et al.* in [10] in relation to spectral clusering 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.d.d. Gaussian noise with standard deviation 0.02 is added to each coordinate. The goal is to segment those two half circles.

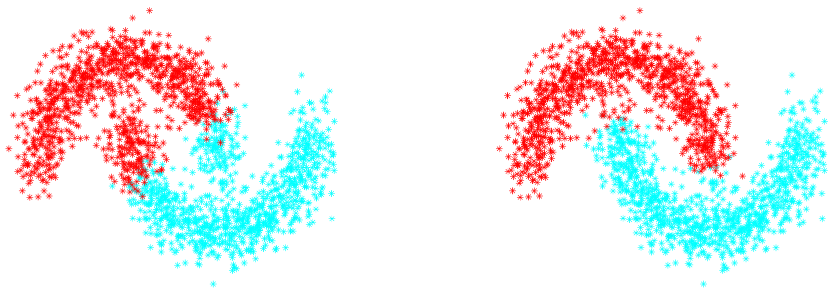
An affinity matrix $W$ is created using the weight function $w(i,j) = e^{-\frac{d(i,j)^2}{\sqrt{(\tau(i)\tau(j))}}}$, a weight function introduced by Zelnik-Manor and Perona in [49], where $\tau(i)$ is the Euclidean distance between point $i$ and the $M$th closest point to it, and $d(i,j)$ is the Euclidean 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].

In step IV of the algorithm, there is no fidelity term so $\lambda(x) = 0$ for all $x$. Thus, $d_k^n = 0$ for all $k$ and $n$. In addition, there is a zero mass constaint $\int u(x)dx = 0$ due to the nature of the problem. Therefore, in the algorithm, 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 constraint has been applied to it. The use of such initialization was justified in section 1.2.1.

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. 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.

We also compared our results to a spectral clustering method of thresholding the second eigenvector of $L_s$. The results are displayed in Figure 2.1. Clearly, clustering using the second eigenvector does not result in an accurate segmentation.



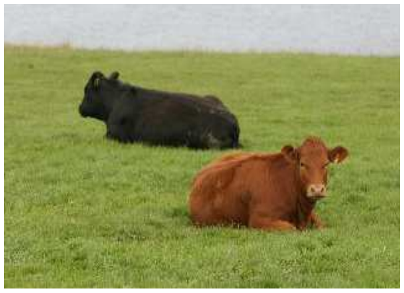
(a) second eigenvector segmentation- 83.75%

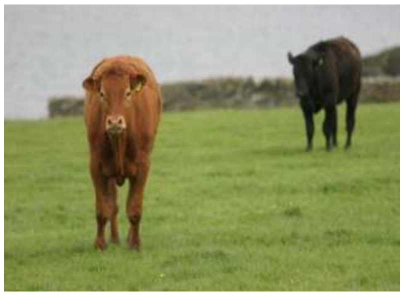
(b) our method's segmentation- 97.7%

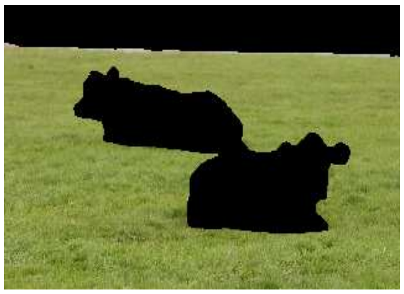
Fig. 2.1: Segmentation by thresholding the second eigenvector and our method, respectively. The four parameters $s$ (in step IV of our algorithm), number of eigenvectors, $dt$, and $M$ (parameter in the Zelnik-Manor and Perona weight function) are set to 3, 25, 0.725 and 13, respectively.

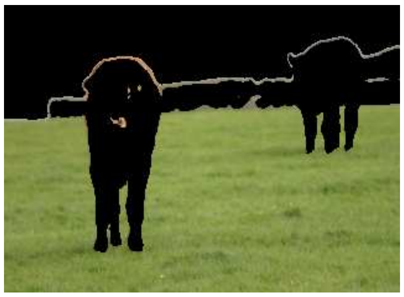
**2.3.2. Image Segmentation.** We also applied our algorithm to segment objects in images of cows from the Microsoft image database. The goal was semi-supervised image segmentation, where two images are inputted into the algorithm, one of which has been hand segmented into the two classes. The algorithm segments the second image based on the segmentation of the first.

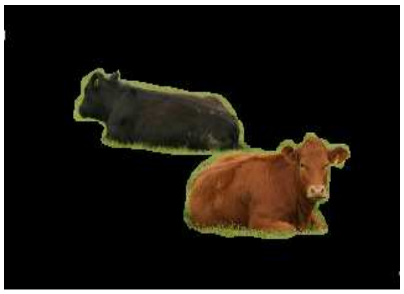
A fully connected graph is constructed in this case, and the entries in the affinity matrix are calculated using feature vectors. Every pixel in the image is assigned a feature vector consisting of intensity values of pixels in its neighborhood, which was of size $7 \times 7$ in our segmentation tests. 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 and texture features. Note that the weight function can be modified according to the image. For

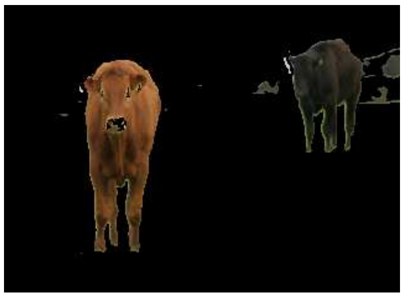

(a) Original Labeled Image

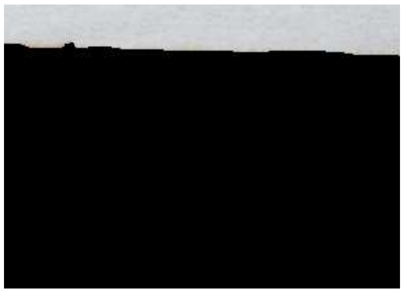

(b) Unlabeled Image


(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

Fig. 2.2: The grass, cow and sky labels were transferred to another image using our algorithm. The number of eigenvectors, $C_1$ and $\sigma$ 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.

| | Minimization time in method in [8] | Minimization time in our method |
|---|---|---|
| grass label | 8 s | 3.5 s |
| cow label | 18 s | 3.5 s |
| sky label | 6 s | 1.8 s |

Table 2.1: Comparison of minimization time of the two methods

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 2.2, where it is shown that our algorithm is robust to mislabeling in the hand labeled image. To transfer the label for the grass, cows and sky, our method needed about 29, 29, 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 Tables 2.1 and 2.2. The calculations show that our method significantly reduces the minimization time and the number of iterations.

**2.3.3. House voting records from 1984.** We applied our algorithm to the US House of Representatives voting records data set, 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 Democrat party using the 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.

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 $M^{th}$ 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, a SVD solver is used. In step IV 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 person in the list. The Democrats were chosen to be the third and fourth person in the list. In the fidelity term, $\lambda(x)$ was set to 1 for each of five known individuals and 0 for the rest.

The parameters $C_1$ (fidelity term parameter), $s$ (in step IV of our algorithm), number of eigenvectors, $dt$, $\sigma$ 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].

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.

| | # of iterations in method in [8] | # of iterations in our method |
|---|---|---|
| two moons | 300 | 40 |
| grass label | 130 | 22 |
| cow label | 274 | 29 |
| sky label | 84 | 11 |
| voting data set | 400 | 5 |

Table 2.2: Comparison of # of iterations of the two methods

**3. 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 solve these problems in the current literature. Local TV methods became state-of-the-art techniques for image impainting. However, since they do not perform well on images with high texture, methods that decompose images into cartoon and texture and simultaneously inpaint both are developed [5, 42]. The problem is also solved with nonlocal inpainting methods. We are particularly interested in the nonlocal inpainting algorithm from [27] as we develop a computationally efficient nonlocal method. Some very successful nonlocal methods for inpainting and sparse reconstruction are given in [2] and [22]. Recently, the class of methods that use dictionaries of small patches that commonly appear in natural images became increasingly popular. Those methods, besides inpainting, are also successful in denoising as shown in [33]. 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 are also successfully applied to solve inpainting problems [14, 11].

We modify our segmentation algorithm slightly for the purpose of binary and grayscale image inpainting. The algorithm consists of the same 4 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 only necessary 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 some important differences to be discussed in sections 3.1 and 3.2.

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

**3.1. Binary Image Inpainting.** Although the key steps of the segmentation algorithm remain the same when it is modified for image inpainting, there are key differences to be noted. For example, it is clear that if a damaged image is used to construct the adjacency matrix $W$, the results might not be accurate, so we first apply a fast and simple $H^1$ inpainting algorithm on the image and then use the result to create $W$. 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 [25], something that is achieved by our algorithm.

The matrix $W$ is built by using a window of a certain size around each 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 2.3.2,$i.e,$. 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 IV 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.

**3.2. Grayscale Image Inpainting.** To generalize to graycale inpaining, we split the signal bit-wise into channels, as in [14]:

$$u(x) = \sum_{m=0}^{K-1} u_m(x) 2^{-m} \tag{3.1}$$

where $u_m$ denotes the $m^{th}$ combonent or digit in the binary representation of the signal, and $u_m \in \{0, 1\}$ for $\forall x$.

A fully connected graph is created in the same way as in section 2.3.2. Again, we first apply the $H^1$ inpainting algorithm on the image, and use the result to build the matrix $W$.

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

In step IV of the algorithm, $\lambda(x)$ in the fidelity term is set to 0 on the inpainting region (which is either black or white) 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 area where the bit is 0. The same stopping criterion is used, except $\alpha = 0.0001$. For some images, step IV 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 damaged image 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”.

### 3.3. Binary Image Inpainting Results.

We applied our algorithm on an image of Barbara and one of stripes. The results and their PSNR are displayed in Figure 3.1. 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.

### 3.4. 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 3.2-3.7. Figure 3.7 is a reconstruction of the original image 3.3a. 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 results are displayed in Table 3.1. We also show our method and nonlocal TV inpainting at certain iterations in Figure 3.8. To implement the nonlocal TV inpainting algorithm, we used the Bregmanized version detailed in [50] 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.

| | Total time for nonlocal TV | Total time for our method |
|---|---|---|
| chessboard-like pattern | 266 s | 48 s |
| text inpainting | 410 s | 67 s |
| small rectangle inpainting | 1882 s | 443 s |
| large rectangle inpainting | 3397 s | 832 s |
| 50% inpainting | 1402 s | 333 s |

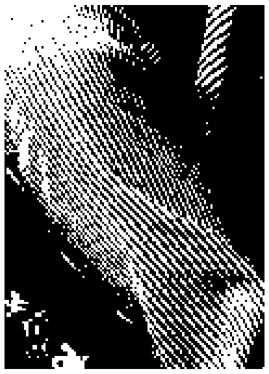
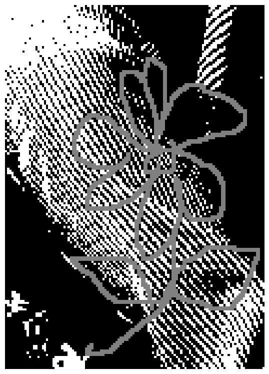
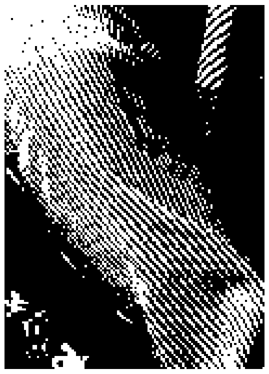
Table 3.1: Timing Comparison

## 4. Conclusion.

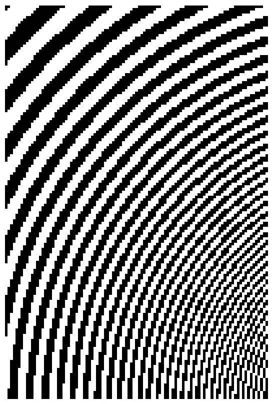
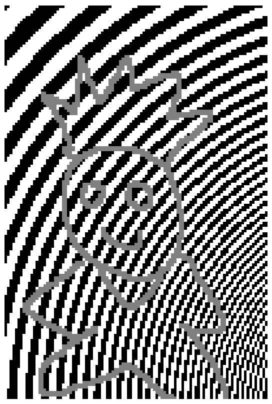
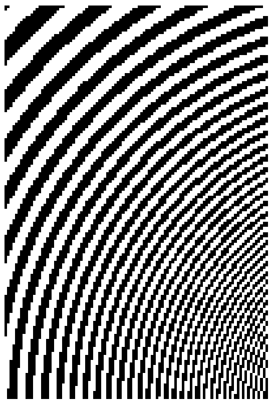
This work presents an algorithm, derived from graph methods and the MBO scheme [35], that links together ideas of $L^1$ compressed sensing, graphs and image processing. The results show that using threshold dynamics in combination with an efficient eigenvalue solver, such as Nyström extension or the Raleigh-Chebyshev procedure of [1], develops an efficient method that can be applied to clustering or image processing. In addition, the nonlocal nature of our method allows it to be successful on images with high texture and repetitive structure.

### 4.1. Acknowledgements.

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, and Blake Hunter for useful discussions regarding this work. This work was supported by ONR grants N000141210040, N000141010472, N00014120838 and AFOSR MURI grant FA9550-10-1-0569. Ekaterina Merkurjev is also supported by the NSF graduate fellowship.

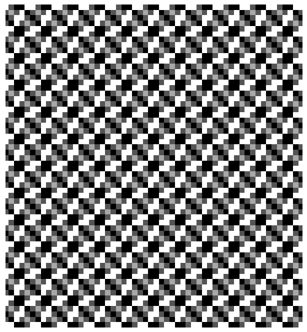




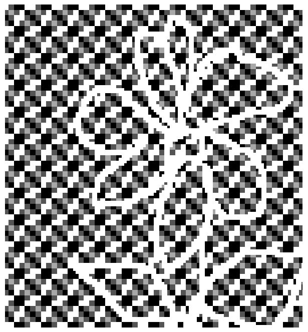
(a) original image- Barbara(b) damaged image- Barbara(c) our method's result- PSNR 20.6896

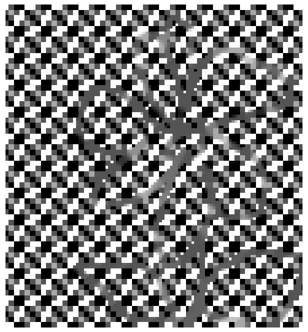




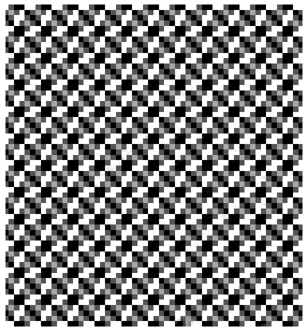
(d) original image- stripes(e) damaged image- stripes(f) our method's result- PSNR 25.0687

Fig. 3.1: Binary Inpainting. For the Barbara image, the simulation took 113 seconds, and there were 6 iterations in the two-step scheme. We used $C_1 = 700$, $dt = 0.003$, $\sigma = 45$, $31 \times 31$ neighborhood for feature vector calculation, $21 \times 21$ window and calculated 400 eigenvectors. For the image of stripes, the simulation took 66 seconds, and there were 4 iterations in the two-step scheme. We used $C_1 = 700$, $dt = 0.002$, $\sigma = 45$, $17 \times 17$ neighborhood for feature vector calculation, $21 \times 21$ 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

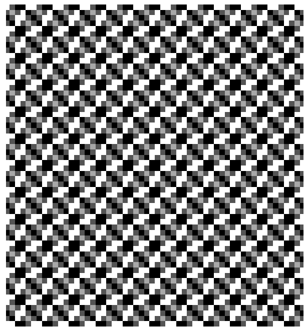
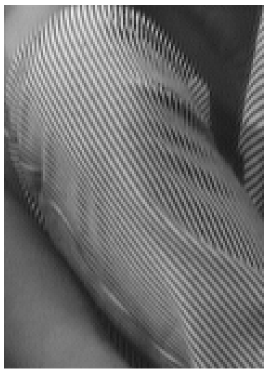
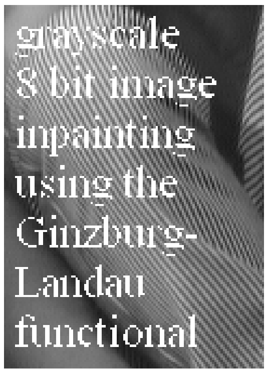
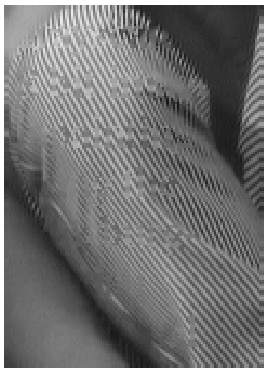

(e) our method's result- perfect reconstruction

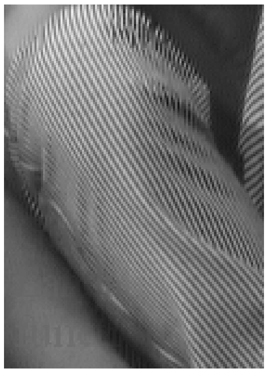
Fig. 3.2: Pattern. The simulation took 48 seconds, and there were 2 iterations in the two-step scheme. We used $C_1 = 700$, $dt = 0.005$, $\sigma = 20$, $41 \times 41$ neighborhood for feature vector calculation, and calculated 600 eigenvectors. No updating of $W$ was necessary. The nonlocal inpainting took 266 seconds.


(a) original image- Barbara



(b) damaged image- Barbara


(c) local TV inpainting- PSNR 29.1508


(d) nonlocal TV inpainting- PSNR 35.6896

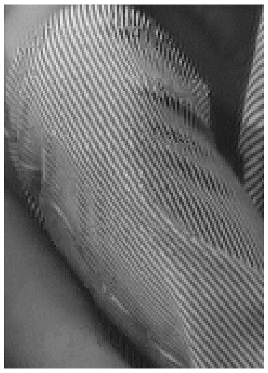
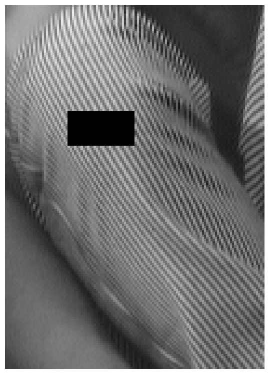
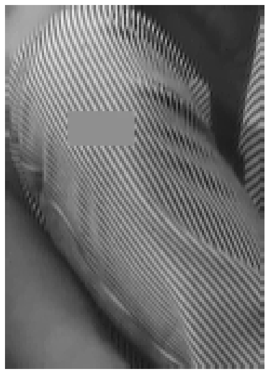
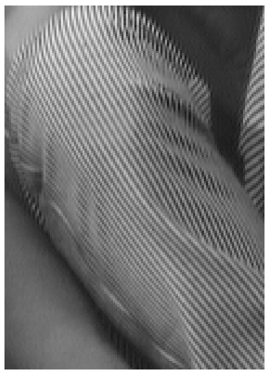

(e) our method's result- PSNR 34.0688

Fig. 3.3: Text Inpainting. The simulation took 67 seconds, and there were 4 iterations in the two-step scheme. We used $C_1 = 700$, $dt = 0.005$, $\sigma = 5$, $21 \times 21$ neighborhood for feature vector calculation, and calculated 500 eigenvectors. We update $W$ every other iteration. The nonlocal inpainting took 410 seconds.

(a) original image- Barbara


(b) damaged image- Barbara


(c) local TV inpainting- PSNR 32.8517


(d) nonlocal TV inpainting- PSNR 44.1469

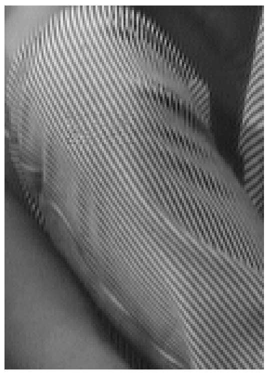
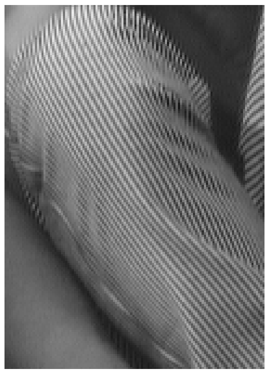
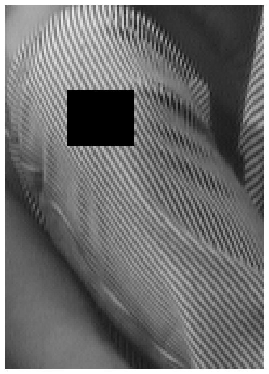
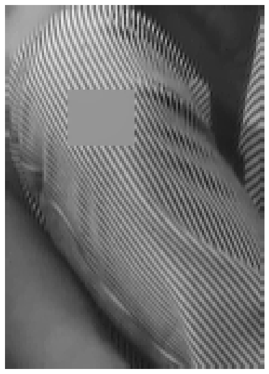
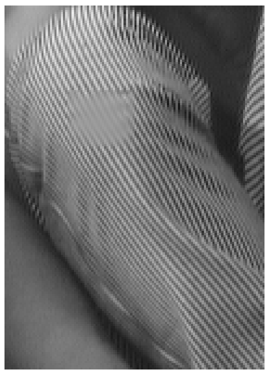

(e) our method's result- PSNR 41.2848

Fig. 3.4: Small Rectangle Inpainting. The simulation took 443 seconds, and there were 13 iterations in the two-step scheme. We used $C_1 = 700$, $dt = 0.01$, $\sigma = 4$, $31 \times 31$ neighborhood for feature vector calculation, and calculated 500 eigenvectors. We update $W$ every iteration. The nonlocal TV inpainting took 1882 seconds.


(a) original image- Barbara


(b) damaged image- Barbara


(c) local TV inpainting- PSNR 31.3673


(d) nonlocal TV inpainting- PSNR 35.0663

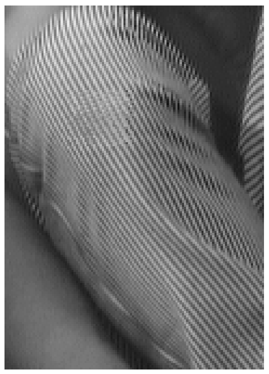
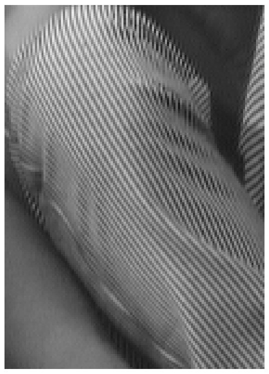
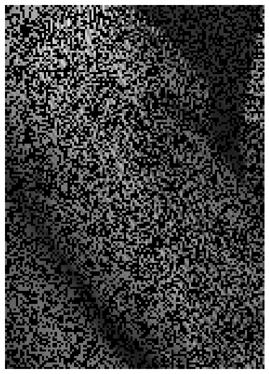
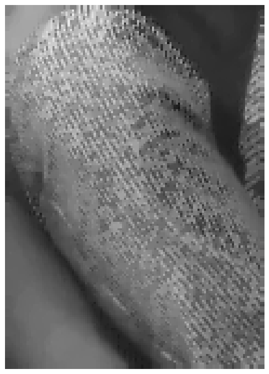
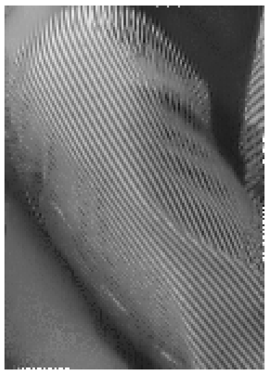

(e) our method's result- PSNR 37.0315

Fig. 3.5: Large Rectangle Inpainting. The simulation took 832 seconds, and there were 13 iterations in the two-step scheme. We used $C_1 = 700$, $dt = 0.014$, $\sigma = 4$, $45 \times 45$ neighborhood for feature vector calculation, and calculated 500 eigenvectors. We update $W$ every iteration. The nonlocal inpainting took 3397 seconds.


(a) original image- Barbara


(b) damaged image- Barbara


(c) local TV inpainting- PSNR 23.6049


(d) nonlocal TV inpainting- PSNR 27.8196

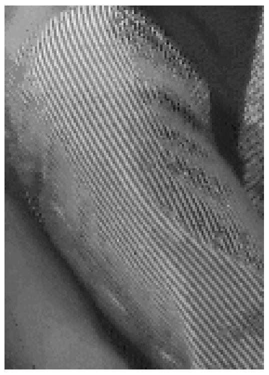
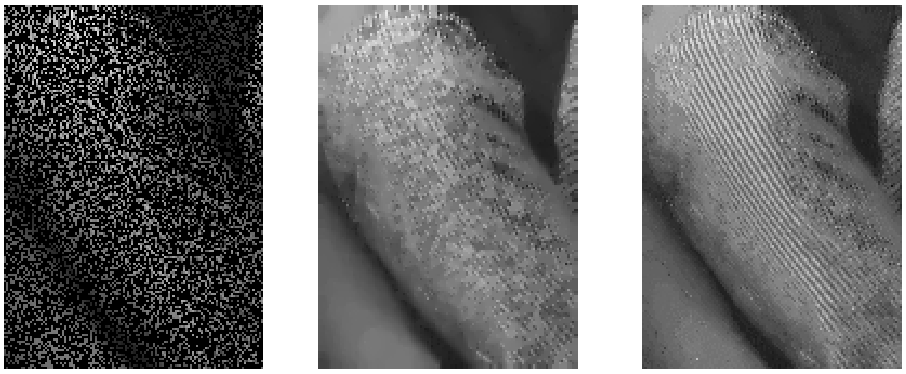
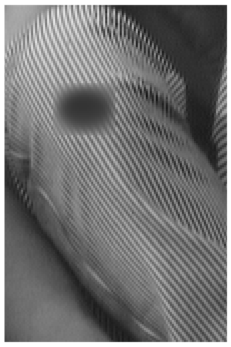

(e) our method's result- PSNR 27.1651

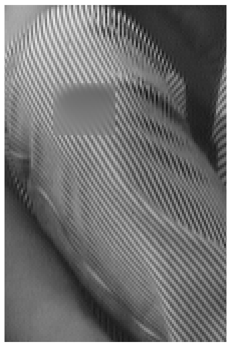
Fig. 3.6: 50% Random Inpainting. The simulation took 333 seconds, and there were 50 iterations in the two-step scheme. We used $C_1 = 700$, $dt = 0.005$, $\sigma = 4$, $7 \times 7$ neighborhood for feature vector calculation, and calculated 400 eigenvectors. We update $W$ every iteration. The nonlocal inpainting took 1402 seconds.

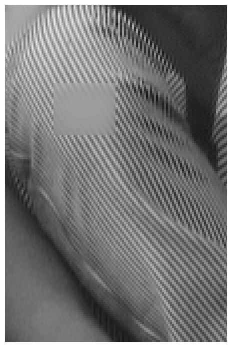


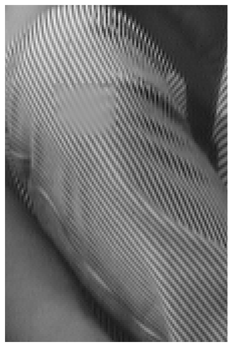
(a) damaged image- 35% of the pixels removed (b) local TV inpainting- PSNR 22.6530 (c) our method's result- PSNR 24.1266

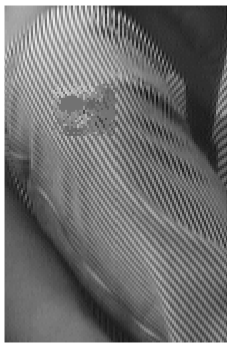
Fig. 3.7: 35% Random Inpainting. The simulation took 1200 seconds, and there were 150 iterations in the two-step scheme. We used $C_1 = 700$, $dt = 0.012$, $\sigma = 4$, $7 \times 7$ neighborhood for feature vector calculation, and calculated 500 eigenvectors. We update $W$ every other iteration.

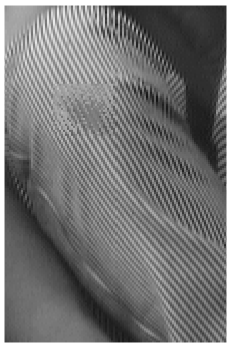

(a) nonlocal TV- after 2 iter.- PSNR 25.7101

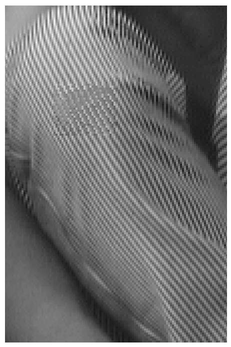

(b) nonlocal TV- after 5 iter.- PSNR 30.7031


(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

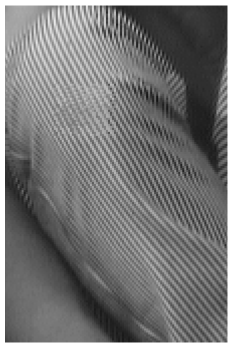

(h) our method- after 13 iter.- PSNR 37.0315

Fig. 3.8: Nonlocal TV inpainting and our method at certain iterations.

## 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*, EMMCVPR 2009, Bonn, Germany, August 2009 Proceedings (2009).

[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*, ECCV 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*, Proceedings of the International Conference Computer Vision and Pattern Recognition, IEEE (2001).

[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*, to appear in Multiscale Model. and Simul. (2012).

[9] A. Buades, B. Coll, and J.-M. Morel, *A non-local algorithm for image denoising*, Proc. IEEE Computer Society on Computer Vision and Pattern Recognition, 2 (2005), pp. 60-65.

[10] T. Bühler and M. Hein, *Spectral clustering based on the graph p-Laplacian*, 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] F. Chung, *Spectral graph theory*, CBMS Reg. Conf. Ser. Math. 92, Providence, RI (1997).

[13] T. Cour, F. Benezit, and J. Shi, *Spectral segmentation with multiscale graph decomposition*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2 (2005), pp. 1124-1131.

[14] J. Dobrosotskaya and A. Bertozzi, *A wavelet-Laplace variational technique for image deconvolution and inpainting*, IEEE Trans. Image Process, 17 (2008), pp. 657-663.

[15] 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.

[16] 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.

[17] S. Esedoḡlu and R. March, *Segmentation with depth but without detecting junctions*, J. Math. Imaging Vision, 18 (2003), pp. 7-15.

[18] S. Esedoḡlu, S.J. Ruuth,and R. Tsai, *Diffusion generated motion using signed distance functions*, J. Comput. Phys., 229 (2010), pp. 1017-1042.

[19] S. Esedoḡlu, S.J. Ruuth, and R. Tsai, *Threshold dynamics for high order geometric motions*, Interfaces Free Bound., 10 (2008), pp. 263-282.

[20] S. Esedoḡlu and Y.R. Tsai, *Threshold dynamics for the piecewise constant Mumford-Shah functional*, J. Comput. Phys., 26 (2004), pp. 367–384.

[21] L.C. Evans, *Convergence of an algorithm for mean curvature motion*, Indiana Univ. Math. J., 42 (1993), pp. 553-557.

[22] G. Facciolo, P. Arias, V. Caselles, and G. Sapiro, *Exemplar-based interpolation of sparsely sampled images*, EMMCVPR, Bonn, Germany, August Proceedings (2009).

[23] C. Fowlkes, S. Belongie,and J. Malik, *Spectral grouping using the Nyström method*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 28 (2006), pp. 469-475.

[24] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, *Efficient spatiotemporal grouping using the Nyström method*, CVPR, Hawaii (2001).

[25] C. Frohn-Schauf, S. Henn, and K. Witsch, *Nonlinear multigrid methods for total variation image denoising*, Comput. Vis. Sci., 7 (2004), pp. 199-206.

[26] G. Gilboa and S. Osher, *Nonlocal operators with applications to image processing*, Multiscale Model. Simul., 7 (2008), pp. 1005-1028.

[27] G. Gilboa amd S. Osher, *Nonlocal linear image regularization and supervised segmentation*, Multiscale Model. Simul., 6 (2007), pp. 595-630.

[28] T. Goldstein and S. Osher, *The split Bregman method for L1-regularized problems*, SIAM J. Imaging Sci., 2 (2009), pp. 323-343.

[29] L. Grady and E. L. Schwartz, *Isoperimetric graph partitioning for image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 28 (2006), pp. 469-475.

[30] A. Harten, *High resolution schemes for hyperbolic conservation laws*, J. Comput. Phys., 49 (1983), pp. 357-393.

[31] R.V. Kohn and P. Sternberg, *Local minimizers and singular perturbations*, Proc. Roy. Soc. Edinburgh Set. A, 11 (1989), pp. 69-84.

[32] Y. Lou, X. Zhang, S. Osher, and A. Bertozzi, *Image recovery via nonlocal operators*, J. Sci. Comput., 42 (2010), pp. 185-197.

[33] J. Mairal, M. Elad, and G. Sapiro, *Sparse representation for color image restoration*, IEEE Trans. Image Process., 17 (2008), pp. 53-69.

[34] B. Merriman and S.J. Ruuth, *Diffusion generated motion of curves on surfaces*, J. Comput. Phys., 225 (2007), pp. 2267-2282.

[35] B. Merriman, J. Bence, and S. Osher, *Diffusion generated motion by mean curvature*, Proceedings of the Computational Crystal Growers Workshop, Providence, Rhode Island (1992), pp. 73-83.

[36] E. Mezuman and Y. Weiss, *Globally optimizing graph partioning problems using message passing*, Proceedings of the 15$^{th}$ International Conference on Artificial Intelligence and Statistics, XX (2012).

[37] B. Mohar, *The Laplacian spectrum of graphs*, Graph Theory, Combinatorics and Applications, 2 (1991), pp. 871-898.

[38] M.M. Naeini, G. Dutton, K. Rothley, and G. Mori, *Action recognition of insects using spectral clustering*, Proceedings of the IAPR Conference on Machine Vision Applications (2007).

[39] G. Peyre, S. Bougleux, and L. Cohen, *Non-local regularization of inverse problems*, European Conference on Computer Vision (ECCV 2008), Springer, Berlin (2008), pp. 57-68.

[40] J. Rubinstein, P. Sternberg, and J. B. Keller, *Fast reaction, slow diffusion, and curve shortening*, SIAM J. Appl. Math., 49 (1989), 116-133.

[41] S.J. Ruuth, *Efficient algorithms for diffusion-generated motion by mean curvature*, J. Comput. Phys., 144 (1998), pp. 603-625.

[42] H. Schaeffer and S. Osher, *A low patch-rank interpretation of texture*, CAM report 11-75 (2011).

[43] J. Shi and J. Malik, *Normalized cuts and image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22 (2000), pp. 888-905.

[44] M. Stoer and F. Wagner, *A simple min-cut algorithm*, J. ACM, 44 (1997), pp. 585-591.

[45] A. Szlam and X. Bresson, *Total variation-based graph clustering algorithm for Cheeger ratio cuts*, Proceedings fo the 27*th* International Conference on Machine Learning (2010), pp. 1039-1046.

[46] Y. van Gennip and A. Bertozzi, *Γ-convergence of graph Ginzburg-Landau functionals*, to appear in Advances in Differential Equations (2012).

[47] U. von Luxburg, *A Tutorial on Spectral Clustering*, Statist Comput., 17 (2007), pp. 395-416.

[48] D. Wagner and F. Wagner, *Between min cut and graph bisection*, Mathematical Foundations of Computer Science: Lecture Notes in Computer Science, 711 (1993), pp. 744-750.

[49] L. Zelnik-Manor and P. Perona, *Self-tuning spectral clustering*, Adv. Neutral Inf. Process. Syst., 17 (2004), pp. 1601-1608.

[50] X. Zhang, M. Burger, X. Bresson and S. Osher, *Bregmanized nonlocal regularization for deconvolution and sparse reconstruction*, SIAM J. on Imaging Sci., 3 (2010), pp. 253-276.

[51] X. Zhang and T. Chan, *Wavelet inpainting by nonlocal total variation*, Inverse Probl. Imaging, 4 (2010), pp. 1-XX.

[52] D. Zhou and B. Schölkopf, *Regularization on discrete spaces*, Springer, Berlin, Germany, pp. 361-368.

[53] D. Zhou and B. Schölkopf, *Discrete regularization*, MIT Press, Cambridge, MA, pp. 221-232.

[54] D. Zhou, J. Huang, and B. Schölkopf, *Learning from labeled and unlabeled data on a directed graph*, Proceedings of the 22$^{nd}$ International Conference on Machine Learning (2005), pp. 1041-1048.

[55] D. Zhou, B. Schölkopf, and T. Hofmann, *Semi-supervised learning on directed graphs*, Adv. Neutral Inf. Process. Syst., 17 (2005), pp. 1633-1640.