SIAM J. APPL. MATH.
Vol. 78, No. 3, pp. 1350–1377

GENERALIZING DIFFUSE INTERFACE METHODS ON GRAPHS: NONSMOOTH POTENTIALS AND HYPERGRAPHS*

JESSICA BOSCH†, STEFFEN KLAMT‡, AND MARTIN STOLL§

Abstract. Diffuse interface methods have recently been introduced for the task of semisupervised learning. The underlying model is well known in materials science but was extended to graphs using a Ginzburg–Landau functional and the graph Laplacian. We here generalize the previously proposed model by a nonsmooth potential function. Additionally, we show that the diffuse interface method can be used for the segmentation of data coming from hypergraphs. For this we show that the graph Laplacian in almost all cases is derived from hypergraph information. Additionally, we show that the formerly introduced hypergraph Laplacian coming from a relaxed optimization problem is well suited to be used within the diffuse interface method. We present computational experiments for graph and hypergraph Laplacians.

Key words. diffuse interface methods, semismooth Newton method, iterative algorithms, semisupervised learning, hypergraphs

AMS subject classifications. 68T05, 65K05, 35R02

DOI. 10.1137/17M1117835

1. Introduction.

The classification of high-dimensional data on graphs is a challenging problem in many application areas [56, 48] and several techniques have been developed to efficiently tackle this problem. Recently, Bertozzi and Flenner [6] have established a method for the interface of graph-based methods and partial differential equations (PDEs). Their method, which has already been extended to other cases (see [31, 43]), utilizes the information of the underlying graph via its graph Laplacian and then uses diffuse interface techniques for the separation of the given data into two classes. Diffuse interface techniques are a classical tool within the materials science community [50, 1]. The new technique of Bertozzi and Flenner uses an approach taken from image inpainting based on phase-field methods [5] for a semisupervised learning problem. The use of phase-field models in image processing has seen many contributions (cf. [24, 16]).

To further use the inpainting analogy in the semisupervised learning problem, the known or sampled data, which are used to train the method, can be considered the intact part of the image and we aim to restore the damaged or unknown part of the image. This works for both segmentation into two classes for binary images or into multiple classes for gray-valued or color images. PDE-based inpainting has been very successful [15] and the technique introduced in [6] showed very promising results when compared to other methods such as the 1-Laplacian inverse power method of Hein and Bühler [33].

*Received by the editors February 21, 2017; accepted for publication (in revised form) January 17, 2018; published electronically May 8, 2018.
http://www.siam.org/journals/siap/78-3/M111783.html

†Department of Computer Science, The University of British Columbia, 201-2366 Main Mall, Vancouver, BC V6T 1Z4, Canada (jbosch@cs.ubc.ca).

‡Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstr. 1, 39106 Magdeburg, Germany (klamt@mpi-magdeburg.mpg.de).

§Technische Universität Chemnitz, Department of Mathematics, Reichenhainer Straße 41, 09126 Chemnitz, Germany and Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstr. 1, 39106 Magdeburg, Germany (martin.stoll@mathematik.tu-chemnitz.de, stollm@mpi-magdeburg.mpg.de).

Our goal in this paper is the extension of the diffuse interface technique from using smooth potentials to the case of nonsmooth potentials as well as the introduction of the diffuse interface approach when applied to hypergraph-based segmentation. Nonsmooth potentials are now widely used in many materials science applications [23, 8] and our previous work [10, 11] in image processing illustrated their importance also for image inpainting. The incorporation of these potentials into the graph-based approach requires the use of an additional nonlinear solver for which we propose a semismooth Newton method [37]. Furthermore, we show that the segmentation is not limited to two classes but extend this to the multiclass segmentation problem as considered in [31]. Additionally, we aim at showing that the approach from [6] is so general that the underlying structural information does not necessarily have to come from the graph Laplacian but that the often very natural hypergraph formulation is well suited for the combination with phase-field approaches both with smooth and nonsmooth potentials.

We start our discussion by introducing the graph Laplacian and the computation of some of its smallest eigenvalues. We then introduce the diffuse interface technique introduced in [6] and extend it to the case when a nonsmooth potential is used. This is done both for the two-classes segmentation problem as well the multiclass segmentation. We further extend the existing approaches by illustrating the applicability of diffuse interface methods on hypergraphs. Numerical results illustrate that the proposed methods work well on many test problems.

2. The graph Laplacian and fundamentals.

We consider here an undirected graph $G = (V, E)$ consisting of a vertex set $V = \{x_i\}_{i=1}^n$ and the edge set E [18]. Each edge $e \in E$ is a pair of nodes (x_i, x_j) with $x_i \neq x_j$ and $x_i, x_j \in V$. For a weighted graph we also have a weight function $w : V \times V \to \mathbb{R}$ with $w(x_i, x_j) = w(x_j, x_i)$ for all i, j. We assume further that the function is positive for existing edges and zero otherwise. The degree of the vertex $x_i \in V$ is defined as

$$d(x_i) = \sum_{x_j \in V} w(x_i, x_j).$$

The diagonal degree matrix $D \in \mathbb{R}^{n,n}$ is defined as $D_{i,i} = d(x_i)$. Now the crucial tool for further investigations is the graph Laplacian L which is defined via

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

It is clear that we can write $L = D - W$ with the entries of the weight matrix W_{ij} given by $w(x_i, x_j)$. The Laplacian in this form is rarely used as typically its normalized form [52] is employed for segmentation purposes. The normalized Laplacian is defined by

$$L_s = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2},$$

which is a symmetric matrix. In contrast another normalized Laplacian of nonsymmetric form is given by

$$L_w = D^{-1} L = I - D^{-1} W.$$

We will use the eigenvalues of the symmetric and normalized graph Laplacian for numerical purposes later. We now discuss possibilities to compute the eigenvalues and eigenvectors of the matrix L_s.

2.1. Computing eigenvalues of the Laplacian. In practice the computation of several small eigenvalues of a matrix is a very challenging task. For small to moderate sizes the QR algorithm [32] is the method of choice for the computation of all eigenvalues of a matrix. For the computation of a subset of the eigenvalues the Lanczos algorithm for symmetric matrices and the Arnoldi algorithm for nonsymmetric matrices are typically chosen if the matrix is large and sparse [39]. For large and sparse graphs the Laplacian will also be large and sparse and the authors in [6, 31] suggest a Rayleigh–Chebyshev procedure [2]. Since the matrix is semidefinite a straightforward inverse iteration cannot be employed. One could consider projection techniques [9] and employing suitable preconditioners is possible [29]. Our goal (cf. [44]) is to compute the k smallest eigenvalues of $L_s = D^{-1/2}LD^{-1/2} = I - D^{-1/2}WD^{-1/2}$. For this it is clear that one could also focus on the largest eigenvalues λ_j of the matrix $D^{-1/2}WD^{-1/2}$ as the eigenvalues of L_s are given via $1-\lambda_j$. One could use the Lanczos method for the computation of the largest eigenvalues as this method only requires the multiplication by the matrices $D^{-1/2}$ and W. Here the dominating cost is given by the application of W, which potentially could be a dense matrix depending on the structure of the graph. To avoid such an expensive step a more advanced method was proposed by Bertozzi and Flenner in [6]. They use the well-known Nyström extension [27, 28], which can work with submatrices of L_s that are of much smaller dimension. The method operates by approximating the eigenpairs of $D^{-1/2}WD^{-1/2}$ using a quadrature rule with randomly chosen interpolation points. For simplicity, we will rely on the Lanczos process for $D^{-1/2}WD^{-1/2}$ via the MATLAB `eigs` function but recommend the use of randomized methods, such as the Nyström extension, for large-scale graph segmentation.

2.2. Weight function. The choice of weight functions $w(x_i, x_j)$ is a crucial ingredient in the construction of the graph Laplacian. This choice will influence the performance of the segmentation process and the speed of the algorithm. This means that different choices of w result in different segmentation results. The graph Laplacian will be crucially influenced by the weight matrix W. For example a sparse matrix W will allow a much easier computation of the eigenvalues of L_s. This means for complete graphs that the weight matrix needs to neglect certain relations between nodes whereas sparse graphs automatically result in sparse weight matrices.

Typical choices for $w(x_i, x_j)$ are the Gaussian function

$$w(x_i, x_j) = \exp\left(-\frac{\operatorname{dist}(x_i, x_j)^2}{\sigma}\right) \tag{1}$$

for some scaling parameter σ and different choices for the metric $\operatorname{dist}(x_i, x_j)$ result in different methods. Another popular choice was introduced by Zelnik-Manor and Perona [54] as

$$w(x_i, x_j) = \exp\left(-\frac{\operatorname{dist}(x_i, x_j)^2}{\sqrt{\tau(x_i)\tau(x_j)}}\right), \tag{2}$$

where $\tau(x_i) = \operatorname{dist}(x_i, x_k)$ and $\tau(x_j) = \operatorname{dist}(x_j, x_k)$ are local scalings of the weight to the Rth nearest neighbor.[1] It is clear that for the application in image processing the distance $\operatorname{dist}(x, y)$ is the difference between intensities of the pixels y and x. For

[1]For the MATLAB codes computing the graph Laplacian we refer to http://www.vision.caltech.edu/lihi/Demos/SelfTuningClustering.html.

color images this will be the sum of distances within the different channels. For other applications, e.g., machine learning, $\mathrm{dist}(x_i, x_j)$ could measure the Euclidean distance between the corresponding feature vectors (cf. [6]) of x_i and x_j. We now have the ingredients to compute the graph Laplacian as well as approximating k of its smallest eigenvalues and introduce the diffuse interface techniques next.

3. Diffuse interface methods on graphs with nonsmooth potentials.

3.1. Diffuse interface methods.

Diffuse interface methods are a classical and versatile tool in the simulation of materials science problems such as solidification processes [1, 14]. They are an indispensable tool for the simulation of phase separation processes but have over time spread to various other application areas ranging from biomembrane simulation [53] to image inpainting [5, 10].

As these methods describe the separation of a mixed medium into two or more phases this methodology was recently extended by Bertozzi and Flenner [6]. In particular, the techniques typically formulated in an infinite-dimensional setting with differential operators, are now used within a graph-based formulation utilizing the graph Laplacian. The derivation of classical models such as the Allen–Cahn [1] or Cahn–Hilliard equations [14] is typically obtained from a gradient flow of the Ginzburg–Landau energy

$$E(u) = \int \frac{\varepsilon}{2} \left|\nabla u\right|^2 dx + \int \frac{1}{\varepsilon}\psi(u)dx, \tag{3}$$

where u is the phase field and ε the interface parameter, which is typically assumed to be small. The function $\psi(u)$ is a potential that forces the phase field u to take values at either $u \approx -1$ or $u \approx 1$ (the pure phases). We come back to the discussion of the choice of potential as this is one of the contributions of this paper. The minimization of the energy $E(u)$ follows a gradient flow, i.e.,

$$\partial_t u = -\mathrm{grad}(E(u)).$$

Different choices for the gradient lead to different evolution equations for the phase u. We here point to the well-known Allen–Cahn equation written as

$$\partial_t u = \varepsilon \Delta u - \varepsilon^{-1}\psi'(u) \tag{4}$$

with given initial condition u_0 and zero Neumann boundary conditions. Here, $\psi'(u)$ is the derivative of a smooth potential

$$\psi(u) = \frac{1}{4}\left(u^2 - 1\right)^2 . \tag{5}$$

As can be seen, the potential function has two distinct minima, one for each of the two pure phases. Hence, its minimization penalizes values away from the pure phases. In the case of a nonsmooth potential we obtain a variational inequality; see section 3.3. The Allen–Cahn equation has also been used very successfully in image inpainting [22, 40]. For this purpose (4) is modified,

$$\partial_t u = \varepsilon \Delta u - \varepsilon^{-1}\psi'(u) + \omega(x)(f - u), \tag{6}$$

where $\omega(x)$ is a parameter that is zero in the damaged image domain D and typically a large constant ω_0 in the intact parts $\Omega \backslash D$. Here f is the given image that we do not want to change in the undamaged part.

3.2. Diffuse interface methods on graphs. In a very similar way, Bertozzi and Flenner formulated the semisupervised learning problem. Here f represents the learned data that have to be maintained throughout the evolution process, and their model separates the domain Ω into two parts, i.e., two phases. The formulation for the separation of a graph is inherently different from the continuous case given before as we want to separate a set of points connected via a graph into two categories. For this infinite-dimensional problem (4) is now defined using the description of the underlying graph Laplacian L_s to give

$$E_s(u) = \frac{\varepsilon}{2} u^T L_s u + \sum_{x \in V} \frac{1}{\varepsilon} \psi(u(x)) + F(f, u) \tag{7}$$

with the energy contribution $F(f, u)$ describing the fidelity term that would lead to $\omega(x)(f - u)$ in the continuous Allen–Cahn equation. The graph Laplacian is a classical tool used in image segmentation, typically based on the eigenvector of its first nonzero eigenvalue [52]. The algorithm proposed here performs a segmentation of the graph under the assumption that a small set of the nodes is already properly classified, i.e., a semisupervised learning method. Here, $u^T L_s u$ is defined via

$$\frac{\varepsilon}{2} u^T L_s u = \frac{\varepsilon}{2} \sum_{x_i, x_j} w(x_i, x_j) \left(\frac{u(x_i)}{\sqrt{d(x_i)}} - \frac{u(x_j)}{\sqrt{d(x_j)}} \right)^2 .$$

Note that for the normalized graph cut described by a vector u the generalized Rayleigh quotient $\frac{u^T L u}{u^T D u}$ is equivalent to the normalized cut [21, 30]. The Rayleigh quotient is equivalent to minimizing

$$\frac{\tilde{u}^T D^{-1/2} L D^{-1/2} \tilde{u}}{\tilde{u}^T \tilde{u}} = \frac{\tilde{u}^T L_s \tilde{u}}{\tilde{u}^T \tilde{u}}.$$

For the minimization of $u^T L u$ it can be seen that if $u = 1$ in the set A and $u = -1$ in $\bar{A}$ one obtains

$$\frac{\varepsilon}{2} u^T L u = \frac{\varepsilon}{2} \sum_{\substack{x_i \in A, x_j \in \bar{A} \\ \text{or } x_i \in \bar{A}, x_j \in A}} w(x_i, x_j)(u(x_i) - u(x_j))^2 = \frac{4\varepsilon}{2} \sum_{\substack{x_i \in A, x_j \in \bar{A} \\ \text{or } x_i \in \bar{A}, x_j \in A}} w(x_i, x_j).$$

This clearly indicates that $u^T L u$ is minimal if the weights across the interface, i.e., in-between values from A and $\bar{A}$, are minimized. For a more detailed discussion of the comparison of diffuse interface methods to other segmentation methods such as graph cuts and nonlocal means we refer to [6].

We are now ready to write down the corresponding modified Allen–Cahn equation for the graph Laplacian as

$$u_t = -\varepsilon L_s u - \varepsilon^{-1} \psi'(u) + \omega(x)(f - u) \tag{8}$$

(see [51, 42] for details). Instead of the infinite-dimensional Laplacian this equation is driven by the graph Laplacian, a matrix, and the potential is still assumed to be smooth. Before discussing the details of the discretization we introduce a convexity splitting scheme that has been used very effectively for Cahn–Hilliard and Allen–Cahn equations with fidelity terms (see [5, 47, 25, 10]). For this the energy is split as

$$E(u) = E_1(u) - E_2(u)$$

with

$$E_1(u) = \int \frac{\varepsilon}{2} |\nabla u|^2 \, dx + \int \frac{c}{2} |u|^2 \, dx$$

and

$$E_2(u) = -\int \frac{1}{\varepsilon}\psi(u) dx + \int \frac{c}{2} |u|^2 \, dx - \int \frac{\omega(x)}{2}(f-u)^2 dx.$$

In order to guarantee the convexity of the energy terms, we require $c \geq \omega_0 + \frac{1}{\epsilon}$; see, e.g., the Cahn–Hilliard case in [5, p. 287] or [13, p. 1156]. Using an implicit Euler scheme for E_1 and explicit treatment for E_2 for the temporal evolution results in

$$\frac{u(x) - \bar{u}(x)}{\tau} - \varepsilon \Delta u(x) + cu(x) = -\frac{1}{\varepsilon}\psi'(\bar{u}(x)) + c\bar{u}(x) + \omega(x)(f - \bar{u}(x)).$$

Note we did not introduce an index for the temporal discretization but rather assume that all values $u(x)$ are evaluated at the new time point whereas $\bar{u}$ indicates the previous time point. These equations are a model based on the infinite-dimensional formulation but our goal is to use the graph-Laplacian-based formulation as introduced in [6]. We obtain the same equations when our formulation is based on the graph Ginzburg–Landau energy, i.e.,

$$\frac{u(x) - \bar{u}(x)}{\tau} + \varepsilon L_s u(x) + cu(x) = -\frac{1}{\varepsilon}\psi'(\bar{u}(x)) + c\bar{u}(x) + \omega(x)(f - \bar{u}(x)),$$

where the dimensionality of u is adjusted to the size of the graph Laplacian. Assuming that (λ_j, ϕ_j) are the eigenpairs of L_s we can write $u(x) = \sum_{k=1}^m \mathbf{u}_k \phi_k$ and from this we get

$$\frac{\mathbf{u}_k - \bar{\mathbf{u}}_k}{\tau} + \varepsilon \lambda_k \mathbf{u}_k + c\mathbf{u}_k = -\frac{1}{\varepsilon}\bar{\mathbf{b}}_k + c\bar{\mathbf{u}}_k + \bar{\mathbf{d}}_k, \tag{9}$$

where $\bar{\mathbf{b}} = \psi'\left(\sum_{k=1}^m \bar{\mathbf{u}}_k \phi_k\right)$ and $\bar{\mathbf{d}} = \omega\left(f - \sum_{k=1}^m \bar{\mathbf{u}}_k \phi_k\right)$. We further rewrite this to obtain

$$\left(1 + \varepsilon\tau\lambda_k + c\tau\right)\mathbf{u}_k = \frac{\tau}{\varepsilon}\bar{\mathbf{b}}_k + (1 + c\tau)\bar{\mathbf{u}}_k + \tau\bar{\mathbf{d}}_k. \tag{10}$$

With the choice of $\psi(u) = \frac{1}{4}(u^2 - 1)^2$ we obtain the scheme introduced in [6].

3.3. Nonsmooth potentials. In classical phase-field simulations the choice of potential function typically plays a crucial role. This goes back to the phase separation process. Such a separation occurs if a high-temperature mixture, existing in a state of isothermal equilibrium, is rapidly quenched to a uniform temperature θ below a critical temperature θ_c. Depending on the temperature reduction, various types of the potential function have been introduced. Originally, Cahn and Hilliard [14] suggested a logarithmic potential (which is out of the scope of the current paper). When the quench $\theta < \theta_c$ is additionally shallow, i.e., $\theta \approx \theta_c$, the logarithmic potential function is usually approximated by a quartic polynomial like (5). Note that the polynomial potential allows violations of the condition $u \in [-1, 1]$. For the deep quench limit $\theta \to 0$, i.e., a very rapid cooling of the mixture resulting in temperatures $\theta \ll \theta_c$, Oono and Puri [45] introduced the nonsmooth double-obstacle potential

$$\psi_{ns}(u) := \begin{cases} \frac{1}{2}(1 - u^2), & -1 \leq u \leq 1, \\ \infty, & \text{otherwise}. \end{cases} \tag{11}$$

As for the smooth polynomial potential, the minima of ψ_{ns} are attained at ± 1. However, the nonsmooth potential admits a sharper, steeper interface and does not allow violations of the condition $u \in [-1,1]$. Such a steep interface will reduce the width of the region where entries are farther away from 1 and -1. This explains why the performance is expected to be improved when using the obstacle potential as fewer entries will lie between -1 and 1. In the nonsmooth setting, we obtain the following modified Allen–Cahn equation

$$\partial_t u = \varepsilon \Delta u - \frac{1}{\varepsilon}(\psi_0'(u) + \mu) + \omega(x)(f-u), \tag{12}$$

$$\mu \in \partial \beta_{[-1,1]}(u), \tag{13}$$

$$-1 \leq u \leq 1, \tag{14}$$

$$\frac{\partial u}{\partial n} = \frac{\partial \Delta u}{\partial n} = 0 \quad \text{on } \partial\Omega. \tag{15}$$

Here we have written ψ_{ns} in (11) via the indicator function[2] as

$$\psi_{ns}(u) = \psi_0(u) + I_{[-1,1]}(u)$$

and $\psi_0(u) := \frac{1}{2}(1-u^2)$. Our problem is a variational inequality, which can be obtained from (12)–(15) by applying the weak formulation. Variational inequalities may be reformulated by introducing Lagrange multipliers associated with the constraints in (14) as done, e.g., in [7]. However, they do not allow a pointwise interpretation which complicates the numerical treatment. Motivated by Hintermüller, M. Hinze, and Tber [36], we apply the Moreau–Yosida regularization technique which can circumvent the treatment of the variational inequality as well as the box constraints in (14). More precisely, we regularize the energy (3) with the Moreau–Yosida penalty term [11, 36] and obtain

$$E(u_\nu) = \int_\Omega \frac{\varepsilon}{2}|\nabla u_\nu|^2 + \frac{1}{\varepsilon}\psi_0(u_\nu) + \frac{1}{2\nu}|\max(0, u_\nu - 1)|^2 + \frac{1}{2\nu}|\min(0, u_\nu + 1)|^2 dx$$

with $0 < \nu \ll 1$ being the regularization or penalty parameter. The regularization terms $\frac{1}{2\nu}|\max(0, u_\nu - 1)|^2$ and $\frac{1}{2\nu}|\min(0, u_\nu + 1)|^2$ are introduced to be able to handle the nonsmooth potential. The smaller ν is, the larger is the penalization for the violation of the condition $|u_\nu| \leq 1$. Hence, the limit $\nu \to 0$ represents the original nonsmooth problem.

Again, we consider the convexity splitting for the energy functional above and obtain

$$E_1(u_\nu) = \int \frac{\varepsilon}{2}|\nabla u_\nu|^2 dx + \int \frac{c}{2}|u_\nu|^2 dx + \int \frac{1}{2\nu}|\max(0, u_\nu - 1)|^2 + \frac{1}{2\nu}|\min(0, u_\nu)|^2 dx$$

and

$$E_2(u_\nu) = -\int \frac{1}{\varepsilon}\psi_0(u_\nu) dx + \int \frac{c}{2}|u_\nu|^2 dx - \int \frac{\omega(x)}{2}(f - u_\nu)^2 dx.$$

In order to guarantee the convexity of the energy terms, we require $c \geq \omega_0$; see, e.g., the Cahn–Hilliard case in [10, p. 73]. This leads to the following evolution equation

$$\frac{u_\nu - \bar{u}}{\tau} - \varepsilon \Delta u_\nu + c u_\nu + \theta_\nu(u_\nu) = -\frac{1}{\varepsilon}\psi_0'(\bar{u}) + c\bar{u} + \omega(f - \bar{u}), \tag{16}$$

[2]Here the indicator function is zero in the indicated interval and otherwise infinity.

where

$$\theta_\nu(u_\nu) := \frac{1}{\nu}\max(0, u_\nu - 1) + \frac{1}{\nu}\min(0, u_\nu + 1).$$

As in the previous section, we did not introduce an index for the temporal discretization but rather assume that all values u_ν are evaluated at the new time point whereas $\bar{u}$ indicates the solution at the previous time point (and does not explicitly depend on ν). In the previous setup the nonlinearity coming from the potential term was shifted towards the right-hand side as it was treated explicitly. In the nonsmooth setting we obtain a nonlinear relation due to the nonsmooth relation given by $\theta_\nu(u_\nu)$, which we treat with the well-known semismooth Newton method [37]. For (16) written as

$$F(u_\nu) = \left(c + \frac{1}{\tau}\right) u_\nu - \varepsilon\Delta u_\nu + \theta_\nu(u_\nu) + \frac{1}{\varepsilon}\psi_0'(\bar{u}) - \left(c + \frac{1}{\tau}\right)\bar{u} - \omega(f - \bar{u}) = 0,$$

the Newton system is given via

$$u_\nu^{(l+1)} = u_\nu^{(l)} - G(u_\nu^{(l)})^{-1}F(u_\nu^{(l)}),$$

where l denotes the Newton step.

To understand the connection between the three ingredients time, regularization, and the semismooth Newton method, let us quickly explain their functioning in the solution method. We have three loops: First, we have the loop over time. Therein is the second loop, which runs over the regularization parameter ν. Note that the regularization technique results in an iterative way for solving the time-discrete modified Allen–Cahn equation: For a sequence $\{\nu_p\}_{p\in\mathbb{N}}$ with $\nu_p \to 0$, we need to solve (16). As explained above, in order to solve (16), we apply a semismooth Newton method. Hence, inside the second loop is the third loop which is the semismooth Newton iteration, which runs over the Newton step l. Note that each semismooth Newton iteration is initialized by the approximate solution of the previous one. Hence we always work with a good initial guess and expect fast convergence of the semismooth Newton method. We define the sets

$$\begin{aligned}\mathcal{A}(u_\nu) &:= \{x \in \Omega : u_\nu > 1 \text{ or } u_\nu < -1\},\\ \mathcal{A}_+(u_\nu) &:= \{x \in \Omega : u_\nu > 1\},\\ \mathcal{A}_-(u_\nu) &:= \{x \in \Omega : u_\nu < -1\},\end{aligned}$$

and write down the Newton system as

$$G(u_\nu^{(l)})u_\nu^{(l+1)} = G(u_\nu^{(l)})u_\nu^{(l)} - F(u_\nu^{(l)}) \tag{17}$$

$$= -\nu^{-1}\left(\chi_{\mathcal{A}_-(u_\nu^{(l)})} - \chi_{\mathcal{A}_+(u_\nu^{(l)})}\right)\mathbf{1} + \left(\frac{1}{\varepsilon} + c + \frac{1}{\tau}\right)\bar{u} + \omega(f - \bar{u}), \tag{18}$$

where $G(u_\nu^{(l)}) := (c + \frac{1}{\tau})I - \varepsilon\Delta + \frac{1}{\nu}\chi_{\mathcal{A}(u_\nu^{(l)})}$ with I the identity operator. Again, we have first introduced the classical problem which was previously studied in [10]. The novelty is its extension to the graph domain. The equivalent formulation using the graph Laplacian is given via

$$\frac{u_\nu - \bar{u}}{\tau} + \varepsilon L_s u_\nu + c u_\nu + \theta_\nu(u_\nu) = -\frac{1}{\varepsilon}\psi_0'(\bar{u}) + c\bar{u} + \omega(f - \bar{u}), \tag{19}$$

and we obtain the Newton system

$$G(u_\nu^{(l)})u_\nu^{(l+1)} \;=\; -\nu^{-1}\left(\chi_{\mathcal{A}_-(u_\nu^{(l)})} - \chi_{\mathcal{A}_+(u_\nu^{(l)})}\right)\mathbf{1} \,+\, \left(\frac{1}{\varepsilon} + c + \frac{1}{\tau}\right)\bar{u} \,+\, \omega(f \,-\, \bar{u}),$$

where $G(u_\nu^{(l)}) := (c+\frac{1}{\tau})I+\varepsilon L_s+\frac{1}{\nu}\chi_{\mathcal{A}(u_\nu^{(l)})}$ with I the identity matrix. In the following we drop the index ν. This Newton system is the equivalent to the infinite-dimensional Newton system and in a Galerkin fashion we assume $u^{(l)} = \sum_{k=1}^m \mathbf{u}_{k,l}\phi_k = \Phi\mathbf{u}_{(l)}$ with a small number k of terms chosen as the projection basis. This results in the projected system

$$\begin{aligned}(20)\quad \Phi^T G(u^{(l)})\Phi\mathbf{u}_{(l+1)} = -\frac{1}{\nu}\Phi^T\left(\chi_{\mathcal{A}_-(u^{(l)})} - \chi_{\mathcal{A}_+(u^{(l)})}\right)\mathbf{1}\\ + \left(\frac{1}{\varepsilon} + c + \frac{1}{\tau}\right)\Phi^T\Phi\bar{\mathbf{u}} + \Phi^T\omega(f - \Phi\bar{\mathbf{u}}).\end{aligned}$$

Here the crucial operator becomes

$$\Phi^T G(u^{(l)})\Phi = \left(c + \frac{1}{\tau}\right)I + \varepsilon\Lambda + \frac{1}{\nu}\Phi^T\chi_{\mathcal{A}(u^{(l)})}\Phi,$$

where Λ is the diagonal matrix containing the k eigenvalues used in the approximation. It is clear that (20) requires the solution of a small $k \times k$ linear system for which we use the CG method [35] or use a direct solver based on a factorization of the matrix.

4. Diffuse interface methods on graphs—the vector-valued case.

4.1. Vector-valued smooth diffuse interface methods.

Section 3 was devoted to scalar diffuse interface models on graphs. In this section, we present their generalization to the vector-valued case. This can then be used for the multiclass segmentation problem.

In practice, often more than two components occur (see, e.g., the biomembrane simulation [53], image inpainting of gray value images [11]) and, thus, diffuse interface models have been extended to deal with multicomponent systems. The Ginzburg–Landau energy for two components in (3) generalizes to

$$(21)\qquad E(\mathbf{u}) = \int \frac{\varepsilon}{2}\sum_{i=1}^K |\nabla u_i|^2 dx + \int \frac{1}{\varepsilon}\psi(\mathbf{u})dx$$

for $K > 2$ components. Here, $\mathbf{u} = (u_1, \ldots, u_K)^T$ is now the vector-valued phase field, and the potential function $\psi(\mathbf{u})$ has K distinct minima instead of two. This section deals with smooth potentials, and the smooth potential in the scalar case generalizes to the vector-valued case as $\psi(\mathbf{u}) = \frac{1}{4}\sum_{i=1}^K u_i^2(1-u_i)^2$. We come back to the discussion of nonsmooth potentials in section 4.2.

Recently, Garcia-Cardona et al. [31] as well as Merkurjev et al. [43] have extended these continuous models to the graph domain. In the following, we summarize their approach. As before, n is the number of data points. We introduce the matrix $U = (\mathbf{u}_1, \ldots, \mathbf{u}_n)^T \in \mathbb{R}^{n,K}$. Here, the kth component of $\mathbf{u}_i \in \mathbb{R}^K$ is the strength for data point i to belong to class k. For each node i, the vector $\mathbf{u}_i$ has to be an element of the Gibbs simplex Σ^K,

$$\Sigma^K := \left\{(x_1, \ldots, x_K)^T \in [0,1]^K \,\middle|\, \sum_{k=1}^K x_k = 1\right\}.$$

The Ginzburg–Landau energy functional on graphs in (7) generalizes to the multiclass case as

$$E(U) = \frac{\varepsilon}{2}\langle U, L_s U\rangle + \frac{1}{\epsilon}\psi(U) + F(\hat{U}, U). \tag{22}$$

Here,

$$\langle U, L_s U\rangle = \operatorname{trace}\left(U^T L_s U\right)$$

measures variations in the vector field, the potential term

$$\psi(U) = \frac{1}{2}\sum_{i\in V}\left(\prod_{k=1}^{K}\frac{1}{4}\|\mathbf{u}_i - \mathbf{e}_k\|_{L_1}^2\right) \tag{23}$$

drives the system closer to the pure phases, and the fidelity term

$$F(\hat{U}, U) = \sum_{i\in V}\frac{\omega_i}{2}\|\hat{\mathbf{u}}_i - \mathbf{u}_i\|_{L_2}^2$$

enables the encoding of a priori information with $\hat{U} = (\hat{\mathbf{u}}_1, \dots, \hat{\mathbf{u}}_n)^T$ representing the learned data. In the potential term, $\mathbf{e}_k \in \mathbb{R}^K$ is the vector whose kth component equals one and all other components vanish. The vectors $\mathbf{e}_1, \dots, \mathbf{e}_K$ correspond to the pure phases. Note that the authors [31, 43] use an L_1-norm for the potential term as it prevents an undesirable local minimum from occurring at the center of the simplex, as would be the case with an L_2-norm for large K. As in the scalar case, ω_i is a parameter that takes the value of a positive constant ω_0 if i is a fidelity node and zero otherwise.

As in section 3, the authors use a convexity splitting scheme to minimize the Ginzburg–Landau functional in the phase-field approach. For this, the energy (22) is split as

$$E(U) = E_1(U) - E_2(U)$$

with

$$E_1(U) = \frac{\varepsilon}{2}\langle U, L_s U\rangle + \frac{c}{2}\langle U, U\rangle$$

and

$$E_2(U) = -\frac{1}{\epsilon}\psi(U) - F(\hat{U}, U) + \frac{c}{2}\langle U, U\rangle.$$

In order to guarantee the convexity of the energy terms, we require $c \geq \omega_0 + \frac{1}{\epsilon}$; see [31, p. 6]. The convexity splitting scheme results in

$$\frac{U - \bar{U}}{\tau} + \varepsilon L_s U + cU = -\frac{1}{2\varepsilon}T(\bar{U}) + c\bar{U} + \Pi(\hat{U} - \bar{U}), \tag{24}$$

where the elements T_{ik} of the matrix $T(\bar{U})$ are given as

$$T_{ik} = \sum_{l=1}^{K}\frac{1}{2}\left(1 - 2\delta_{kl}\right)\|\bar{\mathbf{u}}_i - \mathbf{e}_l\|_{L_1}\prod_{m=1, m\neq l}^{K}\frac{1}{4}\|\bar{\mathbf{u}}_i - \mathbf{e}_m\|_{L_1}^2$$

and Π is a diagonal matrix with elements ω_i. Again, we assume that all values U are evaluated at the new time point whereas $\bar{U}$ indicates the previous time point.

Multiplying (24) by Φ^T from the left and using the eigendecomposition $L_s = \Phi\Lambda\Phi^T$, we obtain

$$\mathcal{U} = B^{-1}\left[(1+c\tau)\bar{\mathcal{U}} - \frac{\tau}{2\epsilon}\Phi^T T(\bar{U}) + \tau\Phi^T\Pi(\hat{U}-\bar{U})\right], \tag{25}$$

where all calligraphic fonts have the meaning $\mathcal{U} = \Phi^T U$. Since $B = (1+c\tau)I + \epsilon\tau\Lambda$ is a diagonal matrix with positive entries, its inverse is easy to apply.

After the update, we have to project the solution back to the Gibbs simplex Σ^K. In order to do this, we make use of the projection procedure in [17]. For the initialization of the segmentation problem, we first assign random values from the standard uniform distribution on $(0,1)$ to the nodes. Then, we project the result to the Gibbs simplex Σ^K and set the values in the fidelity points to the pure phases. Here, we finish the presentation of the model proposed in [31, 43]. Next, we extend this approach to the use of nonsmooth potentials.

4.2. Vector-valued nonsmooth diffuse interface methods. In this section, we extend the approach above to the use of nonsmooth potentials. We start with the continuous setting. The potential function in (21) is now given as

$$\psi(\mathbf{u}) = \begin{cases} \psi_0(\mathbf{u}) & \mathbf{u}\in\Sigma^K, \\ \infty, & \text{otherwise,} \end{cases} \tag{26}$$

where the smooth part is given as $\psi_0(\mathbf{u}) = -\frac{1}{2}\mathbf{u}\cdot T^{ns}\mathbf{u}$. Here, $T^{ns}\in\mathbb{R}^{K,K}$ is a symmetric matrix, which contains constant interaction parameters T^{ns}_{ij}. From physical considerations, T^{ns} must have at least one positive eigenvalue, namely, $\psi(\mathbf{u})$ should have more than one local minimum. The assumption was made in the existing literature; see, e.g., [4, 3, 12], and goes back to the analysis in [20]. A typical choice is $T^{ns} = I - \mathbf{1}\mathbf{1}^T$ with $\mathbf{1} = (1,\ldots,1)^T\in\mathbb{R}^K$ and the identity matrix $I\in\mathbb{R}^{K,K}$, which means that the interaction between all different components is equal and no self-interaction occurs. In the numerical examples, we work with this choice of T^{ns}.

As before in the scalar case, we propose to regularize the energy with a Moreau–Yosida penalty term and obtain

$$E(\mathbf{u}_\nu) = \int \frac{\varepsilon}{2}\sum_{i=1}^K |\nabla u_{\nu,i}|^2 + \frac{1}{\varepsilon}\psi_0(\mathbf{u}_\nu) + \frac{1}{2\nu}\sum_{i=1}^K |\min(0,u_{\nu,i})|^2 dx. \tag{27}$$

Here, ν is again the penalty parameter. Applying the convexity splitting scheme to (27) in the same way as in the nonsmooth scalar case, we obtain the following time-discrete scheme

$$\frac{u_{\nu,i}-\bar{u}_i}{\tau} - \varepsilon\Delta u_{\nu,i} + cu_{\nu,i} + \theta_\nu(u_{\nu,i}) = \frac{1}{\varepsilon}(T^{ns}\bar{\mathbf{u}})_i + c\bar{u}_i + \omega(x)(\hat{u}_i - \bar{u}_i) \tag{28}$$

for $i = 1,\ldots,K$, where

$$\theta_\nu(u_{\nu,i}) := \frac{1}{\nu}\min(0,u_{\nu,i}).$$

In order to guarantee the convexity of the energy terms, we require $c\geq\omega_0$, similarly to the previous cases.

Next, if we write (28) in the form $F_i(u_{\nu,i}) = 0$ for $i=1,\ldots,K$, the semismooth Newton system

$$u_{\nu,i}^{(l+1)} = u_{\nu,i}^{(l)} - G_i(u_{\nu,i}^{(l)})^{-1}F_i(u_{\nu,i}^{(l)})$$

is given as

$$G_i(u_{\nu,i}^{(l)})u_{\nu,i}^{(l+1)} = \left(c + \frac{1}{\tau}\right)\bar{u}_i + \frac{1}{\varepsilon}(T^{ns}\bar{\mathbf{u}})_i + \omega(\hat{u}_i - \bar{u}_i),$$

where $G_i(u_{\nu,i}^{(l)}) := (c + \frac{1}{\tau})I - \varepsilon\Delta + \frac{1}{\nu}\chi_{\mathcal{A}(u_{\nu,i}^{(l)})}$ with

$$\mathcal{A}(u_{\nu,i}^{(l)}) := \{x \in \Omega \colon u_{\nu,i}^{(l)}(x) < 0\}.$$

This is the classical problem formulation. In the graph domain, (28) reads

$$\frac{\mathbf{u}_{\nu,i} - \bar{\mathbf{u}}_i}{\tau} + \varepsilon L_s \mathbf{u}_{\nu,i} + c\mathbf{u}_{\nu,i} + \theta_\nu(\mathbf{u}_{\nu,i}) \;=\; \frac{1}{\varepsilon}\left(T^{ns}\bar{U}^T\right)_i + c\bar{\mathbf{u}}_i + \Pi(\hat{\mathbf{u}}_i - \bar{\mathbf{u}}_i) \tag{29}$$

for $i = 1, \ldots, K$, where $\bar{U} = (\bar{\mathbf{u}}_1, \ldots, \bar{\mathbf{u}}_K) \in \mathbb{R}^{n,K}$ similarly to the previous section,

$$\left(T^{ns}\bar{U}^T\right)_i = -\sum_{j=1, j\neq i}^{K} \bar{\mathbf{u}}_j,$$

and Π is a diagonal matrix with elements ω_i. The resulting Newton system is given as

$$G_i(\mathbf{u}_{\nu,i}^{(l)})\mathbf{u}_{\nu,i}^{(l+1)} = \left(c + \frac{1}{\tau}\right)\bar{\mathbf{u}}_i + \frac{1}{\varepsilon}\left(T^{ns}\bar{U}^T\right)_i + \Pi(\hat{\mathbf{u}}_i - \bar{\mathbf{u}}_i), \tag{30}$$

where $G_i(\mathbf{u}_{\nu,i}^{(l)}) := (c + \frac{1}{\tau})I + \varepsilon L_s + \frac{1}{\nu}\chi_{\mathcal{A}(\mathbf{u}_{\nu,i}^{(l)})}$. Multiplying (30) by Φ^T from the left and using the eigendecomposition $L_s = \Phi\Lambda\Phi^T$, we obtain

$$G_i(\mathbf{u}_{\nu,i}^{(l)})\mathcal{U}_{\nu,i}^{(l+1)} = \left(c + \frac{1}{\tau}\right)\bar{\mathcal{U}}_i + \frac{1}{\varepsilon}\Phi^T(T^{ns}\bar{U}^T)_i + \Phi^T\Pi(\hat{\mathbf{u}}_i - \bar{\mathbf{u}}_i),$$

where $G_i(\mathbf{u}_{\nu,i}^{(l)}) := (c + \frac{1}{\tau})I + \varepsilon\Lambda + \frac{1}{\nu}\Phi^T\chi_{\mathcal{A}(\mathbf{u}_{\nu,i}^{(l)})}\Phi$ and all calligraphic fonts have the meaning $\mathcal{U} = \Phi^T\mathbf{u}$. Since this requires the solution of a small $k \times k$ linear system, we make use of the MATLAB backslash command.

Finally, after each time step, we project the solution back to the Gibbs simplex Σ^K using the procedure in [17].

As highlighted for the scalar case in section 3.3, the performance is expected to be improved when using the obstacle potential. In the vector-valued graph domain case, however, the different effects between smooth and nonsmooth potentials are blurred due to two modifications: First, note that we replace the L_2-norm in the classical smooth potential by the L_1-norm (see (23)) as suggested in [31, 43]. Second, in both models, smooth and nonsmooth, we project the solutions back to the Gibbs simplex, which further blurs the different effects of both potential functions. Note that there is a projection in the scalar case as well: If u denotes the approximate solution in the scalar case, the final segmentation is given by $\text{sign}(u)$. So we do one projection at the very end. However, we believe this projection is less significant than the ones in the vector-valued case, since here we need to do a projection in each time step.

Here, we finish the discussion about diffuse interface methods on graphs. Next, we introduce the diffuse interface approach when applied to hypergraph-based segmentation.

5. Hypergraphs and Laplacians. In this section, we want to show how to generalize the beforementioned methodology to the case of hypergraphs.

A hypergraph is considered as $G = (V, E)$ with $V = \cup\{x_i\}$ a family of objects and E a family of subsets e of V such that $\cup_{e\in E}\, e = V$. We call V the vertices and E the hyperedge set of G. If a weight $w(e)$ is associated with each hyperedge then the hypergraph is called weighted. We can also define the degree $d(x_j)$ as $d(x_j) = \sum_{\{e\in E: x_j\in e\}} w(e)$. Also the edge in a hypergraph has a degree which is simply $\delta(e) = |e|$. The matrix $H \in \mathbb{R}^{|V|,|E|}$ is the incidence matrix of the hypergraph where the rows correspond to the vertices and the columns to the hyperedges. In most applications, the entry $H_{i,j}$ is equal to one if the vertex x_i is contained in the set that defines the hyperedge j, otherwise the entries are set to zero. In all our applications, the set of hyperedges is generated based on the different attributes that describe the problem. For the `zoo` dataset we create several hyperedges where all animals with the same number of legs are in the same hyperedge. The matrices D_V and D_E are diagonal matrices containing the degrees of the vertices and hyperedges, respectively. And the diagonal matrix W_H is the weight matrix containing the weights of the hyperedges. One can then define the adjacency matrix $HW_HH^T - D_V$, where we later use W_H as the identity matrix.

One might now wonder why the introduction of a hypergraph is a useful concept in the segmentation of data. Previous work explicitly using hypergraphs is given in [55, 34]. We here want to point out that in fact most real-world examples are initially represented via hypergraphs be it the image segmentation mentioned earlier where each vertex, i.e., pixel, has an associated vector of RGB values or the congress voting records used in [6] where for each congressman the voting record is stored in a feature vector. Since the incidence matrix of the hypergraph is typically not square, in order to use the graph Laplacian the structure has to be transformed into a graph to represent pairwise relationships. One can obtain a classical graph Laplacian if one creates edges between all vertices that are contained in a hyperedge; in this way the square weight matrix W from (1) can be obtained.

This means that in principle the methodology introduced earlier already takes hypergraph information that is then projected onto a simple graph where the information from the hyperedges is projected into the weight matrix W.

We here use the representation not as pairwise relationships but through the hypergraph Laplacian introduced in [55]. This approach is based on a relaxed problem that one considers instead of the NP-hard cut problem. In more detail, one typically considers a relaxed optimization problem (cf. [55])

$$\operatorname{argmin}_{u\in\mathbb{R}^{|V|}} \frac{1}{2}\sum_{e\in E}\sum_{\{x_i,x_j\}\subseteq e}\frac{w(e)}{\delta(e)}\left(\frac{u(x_i)}{\sqrt{d(x_i)}}-\frac{u(x_j)}{\sqrt{d(x_j)}}\right)^2 \tag{31}$$

subject to

$$\sum_{x_j\in V} f(x_j)^2 = 1, \sum_{x_j\in V} u(x_j)\sqrt{d(x_j)} = 0. \tag{32}$$

Defining the matrices $\Theta = D_V^{-1/2}HW_HD_E^{-1}H^TD_V^{-1/2}$ and $L_s = I - \Theta$, it was shown in [55] that

$$\sum_{e\in E}\sum_{\{x_i,x_j\}\subseteq e}\frac{w(e)}{\delta(e)}\left(\frac{u(x_i)}{\sqrt{d(x_i)}}-\frac{u(x_j)}{\sqrt{d(x_j)}}\right)^2 = u^TL_su. \tag{33}$$

It is clear, due to W and D_E being diagonal matrices, that the matrix L_s is symmetric, and the definiteness follows from (33). If $u = 1$ in the set A and $u = -1$ in $\bar{A}$, one obtains for $\{x_i, x_j\} \subseteq e$

$$u^T L_s u = \sum_{e\in E} \left(\sum_{\substack{x_i\in A, x_j\in\bar{A} \\ \text{or} \\ x_i\in\bar{A}, x_j\in A}} \frac{w(e)}{\delta(e)} \left(\frac{1}{\sqrt{d(x_i)}} + \frac{1}{\sqrt{d(x_j)}} \right)^2 \right. \tag{34}$$

$$+ \sum_{\substack{x_i\in A, \\ x_j\in A}} \frac{w(e)}{\delta(e)} \left(\frac{1}{\sqrt{d(x_i)}} - \frac{1}{\sqrt{d(x_j)}} \right)^2 \tag{35}$$

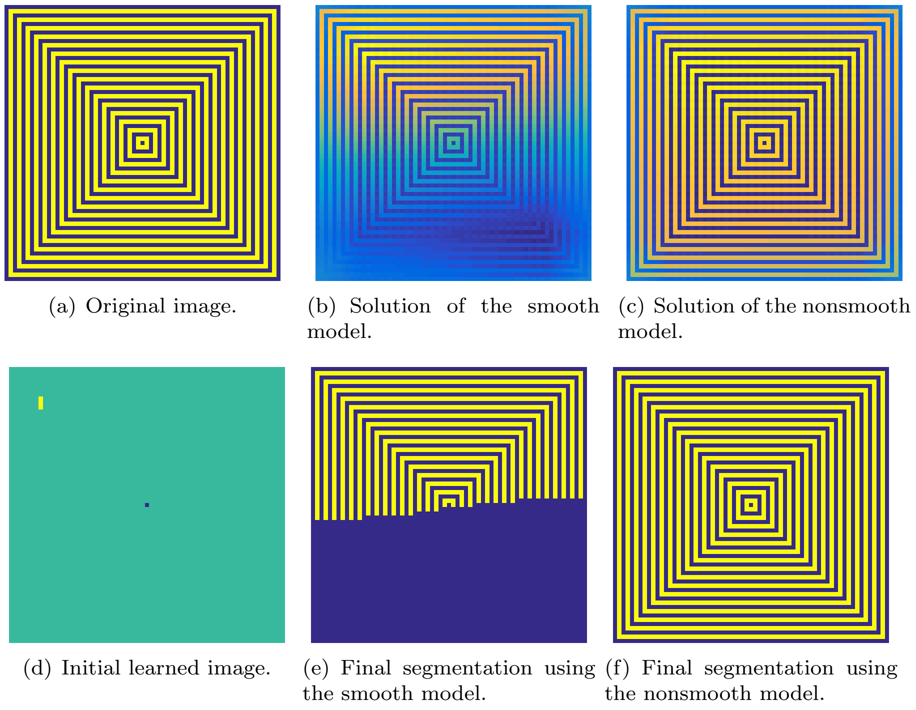
$$\left. + \sum_{\substack{x_i\in\bar{A}, \\ x_j\in\bar{A}}} \frac{w(e)}{\delta(e)} \left(\frac{-1}{\sqrt{d(x_i)}} + \frac{1}{\sqrt{d(x_j)}} \right)^2 \right). \tag{36}$$

The last equation motivates the use of the diffuse interface approach, as assuming that the degrees of the vertices are similar implies that (34) is the dominating term. This means using the hypergraph Laplacian for $u^T L_s u$. Minimizing this quantity is achieved if the weights across the interface, i.e., $w(e)$ in the first summand, are minimal. We hence use the hypergraph Laplacian in the same way as the graph Laplacian for the segmentation of the vertices and run all diffuse interface models with the eigenvectors and eigenvalues of the hypergraph Laplacian instead of the graph Laplacian that could also be derived from hypergraph information.

6. Numerical experiments. The aim is to show that the methods introduced in this paper are effective and we chose to compare the smooth and nonsmooth potential version of the diffuse interface method for graph-based and hypergraph-based problems. Our goal is to illustrate the performance of the method for several examples, namely, `mushroom`, `zoo`, and `student performance`, from the UCI database [41] as well as a structural image. We mostly compare our methodology to the popular method introduced by Bertozzi and Flenner. The authors in [6] compare the diffuse interface approach to many other techniques such as the p-Laplacian [49] with a favorable outcome for their approach.

The computation of the eigenvalues is based on `eigs` from MATLAB, which uses the Lanczos process for $D^{-1/2}WD^{-1/2}$ in the graph case and for Θ in the hypergraph Laplacian. The results presented here are snapshots of a high-dimensional space of parameters that can be chosen and we want to illustrate the performance with respect to varying these parameters. Such parameters include the interface parameter ϵ, the number k of eigenvalues for the Laplacian, the convexity splitting parameter c, the (pseudo)-time-step of the Allen–Cahn equation τ, as well as the correct stopping tolerance. One of the crucial questions is also the performance of the algorithms with respect to changes in the number of known or learned data.

Regarding the computational costs for solving the smooth versus the nonsmooth model, it is clear that the first one outperforms the latter. In the smooth case, we need to apply the inverse of one diagonal $k \times k$ matrix per time step. In the nonsmooth case, we have two additional loops within each time step: These are first a loop over the regularization parameter ν. More precisely, we solve for a sequence $\{\nu_p\}_{p\in\mathbb{N}}$ with $\nu_p \to 0$. Inside the regularization loop is the semismooth Newton iteration, which


(a) Original image. (b) Solution of the smooth model. (c) Solution of the nonsmooth model.

(d) Initial learned image. (e) Final segmentation using the smooth model. (f) Final segmentation using the nonsmooth model.

FIG. 1. *Scalar image segmentation.*

runs over the Newton step l. Hence, in the nonsmooth case, we need to apply the inverse of at most $q_{\max} \cdot l_{\max}$ nondiagonal $k \times k$ matrices per time step, where $q_{\max}$ denotes the length of the regularization parameter sequence and $l_{\max}$ the maximum number of Newton steps. While the computations become more expensive due to the nonlinearity that is treated with the semismooth Newton scheme, the examples in many cases below show that the segmentation results are better with the nonsmooth model.

6.1. Graph Laplacian. Graph-based segmentation has been used for both UCI datasets [41] and image-based segmentation. We start with the scalar case for both a point set and an imaging problem. We later extend this to the multiclass segmentation.

Scalar segmentation. The first test we perform is based on the 65×65 image given in Figure 1(a). This image consists of two colors—here given by dark blue and yellow. The learned information of the image used as the initial state for the smooth and nonsmooth models is shown in Figure 1(d). The known image information is given by one pixel in the dark blue part and three pixels in the yellow part. Hence, the known image information constitutes only 0.0947 % of the whole image. The solution u of the smooth model is presented in Figure 1(b), while Figure 1(e) illustrates the final segmentation sign(u). The two corresponding results using the nonsmooth model are given in Figures 1(c) and 1(f). The chosen parameters are given as $\omega_0 = 1$, $\varepsilon = 0.5$, $\tau = 0.01$, $\nu = 10^{-7}$, $c = 2\varepsilon^{-1} + \omega_0$, $R = 21$, and $k = 5$. Here, R is the local scale for the graph Laplacian computation as used in (2), k the number of eigenvalues. The computation of the eigenvalues is based on `svds` from MATLAB. As stopping criterion for the smooth and nonsmooth models, we use

$$\frac{\|u-\bar{u}\|}{\|\bar{u}\|} \leq \epsilon_{tol}, \tag{37}$$

where we set $\epsilon_{tol} = 10^{-6}$, and we fix the maximum number of time steps to $t_{\max} = 500$. Note that for the nonsmooth model, we fix a sequence of penalty parameters $\{\nu_q\}_{q\in\mathbb{N}}$ with $\nu_q \to 0$, and in each time step, we solve the problem $F(u_{\nu_q}) = 0$ for $q = 1,\dots,q_{\max}$ via a semismooth Newton method. In all examples, we use $\nu_1 = 10^{-1} \geq \nu_2 = 10^{-2} \geq \cdots \geq \nu_{q_{\max}} = 10^{-7}$. Each semismooth Newton method is initialized by the approximate solution of the previous one. As stopping criterion for the semismooth Newton method, we use

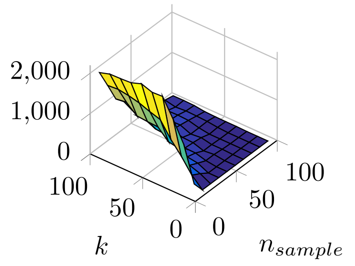
$$\|F(u^{(l+1)})\| \leq \epsilon_{\text{rel}}\|F(u^{(0)})\| + \epsilon_{\text{abs}}, \quad l = 1,\dots,l_{\max}, \tag{38}$$

where we set $l_{\max} = 20$, $\epsilon_{\text{rel}} = 10^{-12}$, and $\epsilon_{\text{abs}} = 10^{-6}$. Finally, we solve the $k \times k$ systems of linear equations arising in each semismooth Newton step with the MATLAB backslash command.

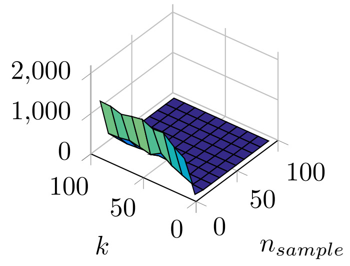
The smooth model stops after $t_{\max} = 500$ time steps with $\frac{\|u-\bar{u}\|}{\|\bar{u}\|} = 7.9 \cdot 10^{-4}$. The CPU time is 0.6 s and the minimum and maximum value of the solution are -1.223427 and 1.335510. The nonsmooth model stops after $t_{\max} = 500$ time steps with $\frac{\|u-\bar{u}\|}{\|\bar{u}\|} = 8.4 \cdot 10^{-3}$. The CPU time is 16.3 s and the minimum and maximum value of the solution are -1.000347 and 1.000110. We observe that the concentrations stay closer within the interval $[-1,1]$ when the nonsmooth potential is used. Moreover, we clearly see from Figures 1(b)–1(f) that the segmentation using the smooth model is either unsuccessful or has not finished after 500 time steps. We will investigate this issue shortly after introducing our measure of quality. In order to evaluate the quality of segmentation, we use misclassification. The number of misclassified pixels is 996 using the smooth model (Figure 1(e)) and zero using the nonsmooth model (Figure 1(f)). Now, we discuss the above-mentioned issue that the smooth model is either unsuccessful or has not finished after 500 time steps. We repeat the same simulation for the smooth model with $t_{\max} = 5000$ and $\epsilon_{tol} = 10^{-14}$: The simulation stops after 1817 time steps and a CPU time of 2.9 s with $\frac{\|u-\bar{u}\|}{\|\bar{u}\|} = 9.8 \cdot 10^{-15}$ and 1024 misclassified pixels. Hence, the segmentation using the smooth model is unsuccessful with the used parameter set.

Next, we show the effect of varying different parameters for the image example given above. Each plot in Figure 2 shows the mean of the number of misclassified pixels which were calculated for 10 runs with randomly chosen samples. In Figures 2(a) and 2(b), we vary the number of given sample points[3] n_{sample} and the number of eigenvalues k for the smooth and nonsmooth model. For both models, the segmentation performance increases with an increasing number of classified samples n_{sample}. We observe in the smooth case that if we reduce n_{sample}, then k should be reduced as well. This effect occurs in the nonsmooth case only for the lower range of n_{sample}. The difference of both results is illustrated in Figures 3(a). Negative values indicate that the nonsmooth potential performed better. Except for the case of small values of k and large values of n_{sample}, the nonsmooth model outperforms the smooth one. In Figures 2(c) and 2(d), we vary the number of eigenvalues k and the distance R which was a local scale for the graph Laplacian computation, for the smooth and

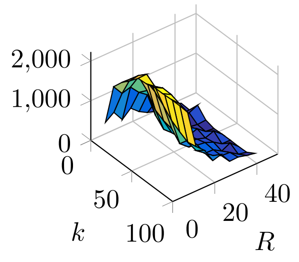
[3]As we know the classification beforehand we randomly sample points with known classification. The sample points here represent the learned or known data.



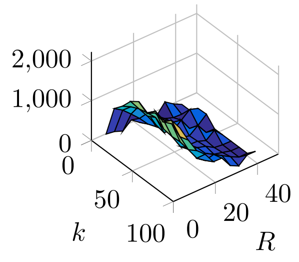
(a) Mean of the misclassification using the smooth model.



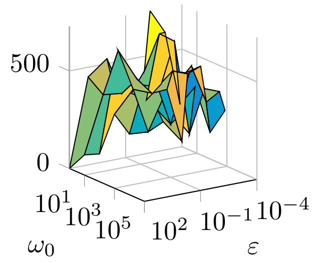
(b) Mean of the misclassification using the nonsmooth model.



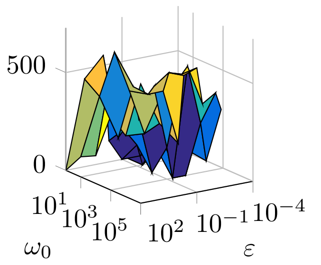
(c) Mean of the misclassification using the smooth model.



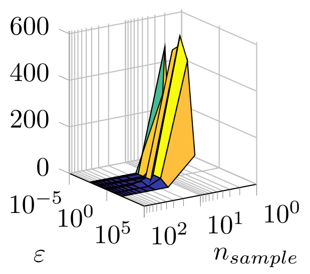
(d) Mean of the misclassification using the nonsmooth model.



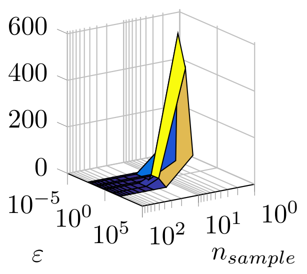
(e) Mean of the misclassification using the smooth model.



(f) Mean of the misclassification using the nonsmooth model.

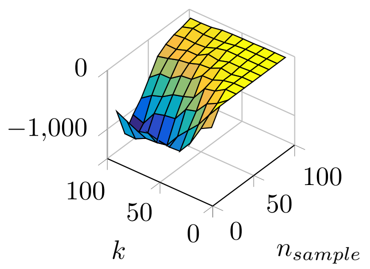


(g) Mean of the misclassification using the smooth model.

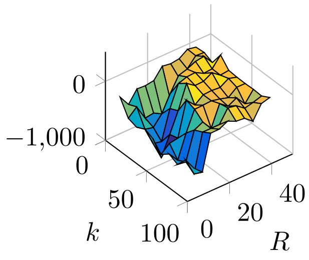


(h) Mean of the misclassification using the nonsmooth model.

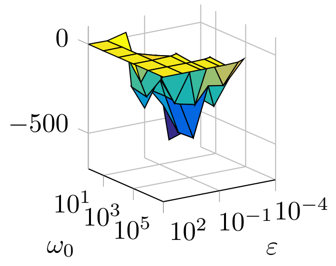
FIG. 2. *Comparison of the smooth and nonsmooth models: The mean of the number of misclassified pixels for the smooth (left column) and nonsmooth (right column) models for varying parameters. For each* (x, y) *pair, we have taken* 10 *runs with randomly chosen samples.*



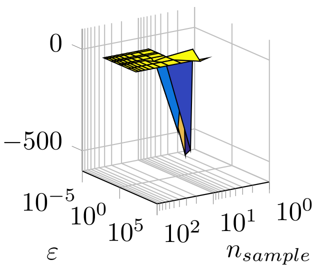
(a) Difference of the results in Figures 2(a) and 2(b).



(b) Difference of the results in Figures 2(c) and 2(d).



(c) Difference of the results in Figures 2(e) and 2(f).

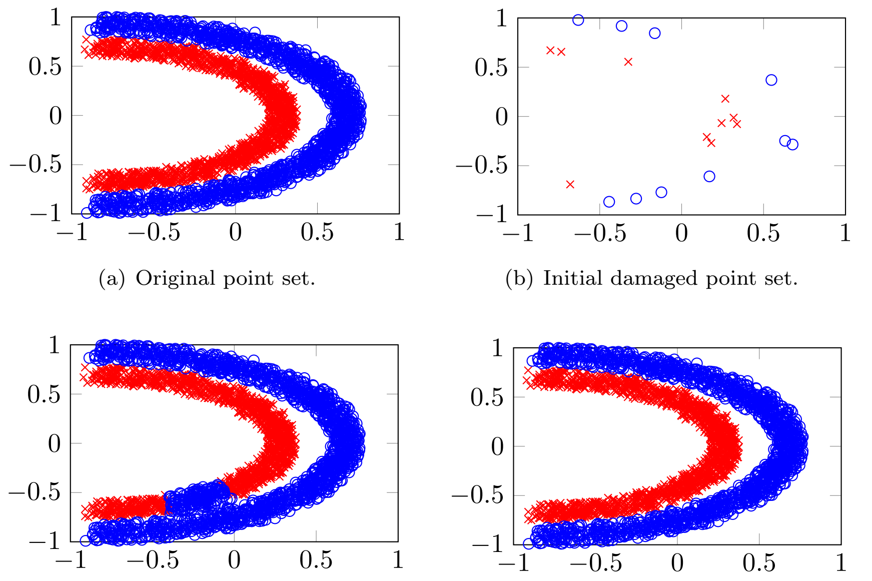


(d) Difference of the results in Figures 2(g) and 2(h).

FIG. 3. *The differences between the mean for the smooth and nonsmooth models with respect to the results in Figure* 2. *Negative values indicate that the nonsmooth potential performed better.*

nonsmooth models. For both models, the segmentation performance increases as R increases. Small values of k give better results with the nonsmooth model. The difference of both results is illustrated in Figures 3(b). In almost every case, the nonsmooth model outperforms the smooth one. In Figures 2(e) and 2(f), we vary the interface parameter ε and the fidelity parameter ω_0 for the smooth and nonsmooth models. For both models, the segmentation performance increases as ε decreases. The difference of both results is illustrated in Figure 3(c). In most cases, the nonsmooth model outperforms the smooth one. In Figures 2(g) and 2(h), we vary the interface parameter ε and the number of given sample points n_{sample} for the smooth and nonsmooth model. For both models, the segmentation performance increases as n_{sample} increases. For a small number of n_{sample}, the smooth model performs not so well, but seems to get better for a decreasing ε. In contrast, the nonsmooth model performs well for a small number of n_{sample} when ε is small. The difference of both results is illustrated in Figure 3(d). In most cases, the nonsmooth model outperforms the smooth one.

Next, we consider a problem with a point set. If not mentioned otherwise, we use the same parameters and stopping criterion as in the previous example. The test is based on the point set given in Figure 4(a), which consists of 3000 data points in total. We have two kinds of points, the red ones and the blue ones, whereby each class contains 1500 points. The damaged data set used as the initial state for the smooth and nonsmooth models is shown in Figure 4(b). The known information is given by 10 data points for each class. Hence, the known data information constitutes only 0.6667 % of the whole data set. The final segmentation using the smooth and nonsmooth models is presented in Figures 4(c) and 4(d), respectively. The chosen parameters are given as $\omega_0 = 1$, $\varepsilon = 0.5$, $\tau = 0.01$, $\nu = 10^{-7}$, $c = 3\varepsilon^{-1} + \omega_0$, $R = 9$,



(a) Original point set.

(b) Initial damaged point set.

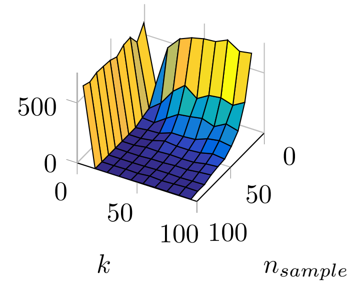
(c) Final segmentation using the smooth model.

(d) Final segmentation using the nonsmooth model.

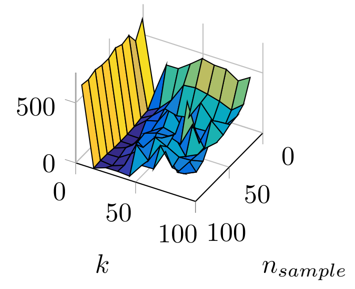
FIG. 4. *Segmentation of a point set into two classes.*

$k = 15$, and $t_{\max} = 400$. The smooth model was not able to correctly classify the area around $(-0.75, -0.6)$. This is exactly the area of a large gap in the initial data, as seen in Figure 4(b).

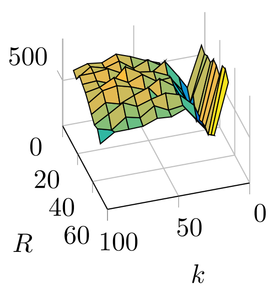
Next, we show the effect of varying different parameters for the 3000 data point set given above. Each plot in Figure 5 shows the mean of the number of misclassified points which were calculated for 10 runs with randomly chosen samples. In Figures 5(a) and 5(b), we vary the number of given sample points n_{sample} and the number of eigenvalues k for the smooth and nonsmooth models. For both models, the segmentation performance increases as n_{sample} increases. For small values of n_{sample}, the nonsmooth model performs better, whereas the smooth model gives better results for larger values of the pair (n_{sample}, k). This can be seen in Figure 6(a), which shows the difference of both results. Negative values indicate that the nonsmooth potential performed better. In Figures 5(c) and 5(d), we vary the number of eigenvalues k and the distance R for the smooth and nonsmooth models. For both models, the segmentation performance is the best for $k = 10$. The difference of both results is illustrated in Figure 6(b). In almost every case, the nonsmooth model outperforms the smooth one. In Figures 5(e) and 5(f), we vary the interface parameter ε and the fidelity parameter ω_0 for the smooth and nonsmooth models. For both models, the segmentation performance increases as ε and ω_0 increase. The difference of both results is illustrated in Figure 6(c). Both models behave mostly similarly. For small values of ε and ω_0, the nonsmooth model performs better. In Figures 5(g) and 5(h), we vary the interface parameter ε and the number of given sample points n_{sample} for the smooth and nonsmooth models. For both models, the segmentation performance increases as ε increases. The difference of both results is illustrated in Figure 6(d). The nonsmooth model tends to have fewer misclassifications compared to the smooth model for small values of ε.



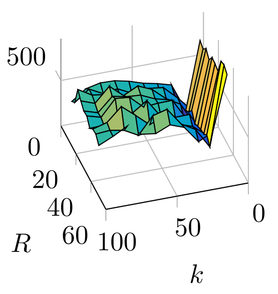
(a) Mean of the misclassification using the smooth model.



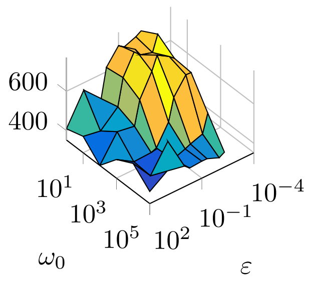
(b) Mean of the misclassification using the nonsmooth model.



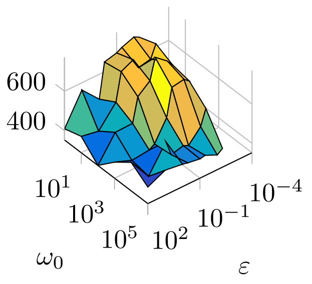
(c) Mean of the misclassification using the smooth model.



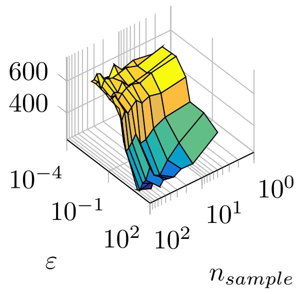
(d) Mean of the misclassification using the nonsmooth model.



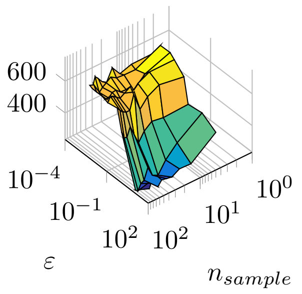
(e) Mean of the misclassification using the smooth model.



(f) Mean of the misclassification using the nonsmooth model.

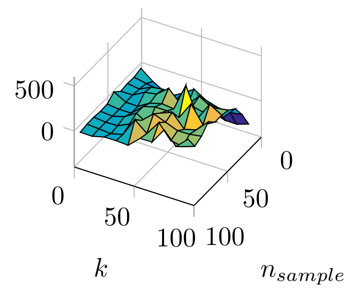


(g) Mean of the misclassification using the smooth model.

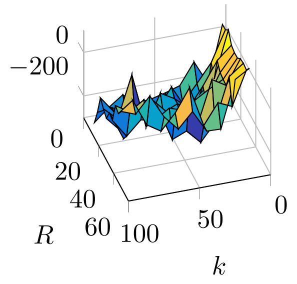


(h) Mean of the misclassification using the nonsmooth model.

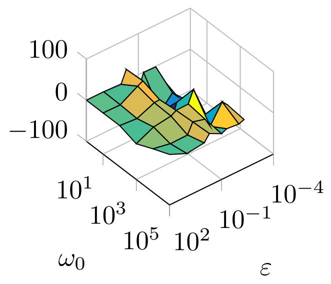
FIG. 5. *Comparison of the smooth and nonsmooth models: The mean of the misclassification for the smooth (left column) and nonsmooth (right column) models for varying parameters. For each (x, y) pair, we have taken* 10 *runs with randomly chosen samples.*



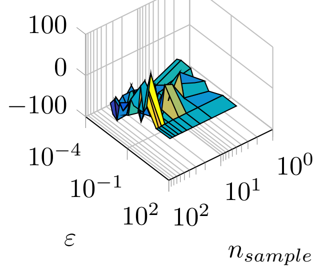
(a) Difference of the results in Figures 5(a) and 5(b).



(b) Difference of the results in Figures 5(c) and 5(d).



(c) Difference of the results in Figures 5(e) and 5(f).



(d) Difference of the results in Figures 5(g) and 5(h).

FIG. 6. *The differences between the mean for the smooth and nonsmooth models with respect to the results in Figure* 5*. Negative values indicate that the nonsmooth potential performed better.*

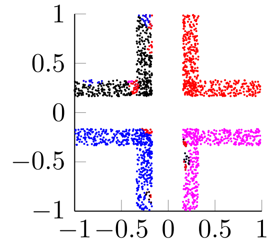
Multiclass segmentation. We show in Figure 7 the results for a segmentation problem into four classes into the four corners.[4] We here vary the number of used eigenvalues of the graph Laplacian as well as the number of samples. We uniformly take the values $n_{sample} = 5, 10, \ldots, 50$ and $k = 5, 10, \ldots, 50$. It can be seen that with an increase in the number of both n_{sample} and k the misclassification is dramatically reduced. Here the one axis shows the variation in n_{sample} and the other the variation in k. For the mean we have taken 10 runs with randomly chosen samples. Figure 7 also shows the difference in the means between the nonsmooth and the smooth potentials. It can be seen that for sufficient information with larger sample and eigenvalues size the difference is negligible but for smaller values of n_{sample} the nonsmooth potential performs better for increasing values of k than the smooth potential. The chosen values are $\omega_0 = 10000$, $\nu = 10^{-7}$, $\varepsilon = 10^1$, $\tau = 0.1$, and $c = (2/\varepsilon) + \omega_0$.

6.2. Hypergraph Laplacian. We now want to present results for our approach regarding hypergraphs where both, the cases of a smooth and nonsmooth potential, are tested. The case for hypergraph-based classification was made by [55] and [34].

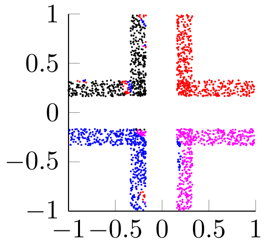
Scalar segmentation. We here focus our attention on two datasets. The first dataset is the so-called `mushroom` dataset[5] as introduced by Schlimmer [46, 41]. The

[4]The data are generated using the MATLAB code http://de.mathworks.com/matlabcentral/fileexchange/41459-6-functions-for-generating-artificial-datasets.

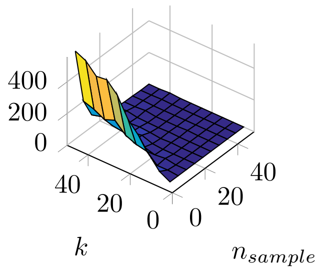
[5]We obtain a MATLAB version of the data from http://people.whitman.edu/~hundledr/courses/M350F14/M350/mushrooms.mat.



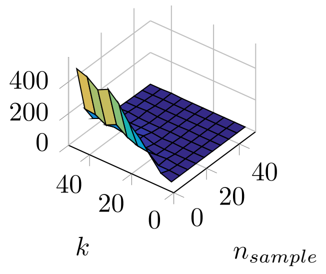
(a) smooth potential



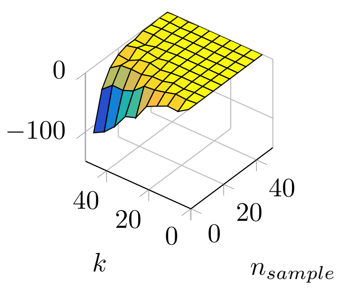
(b) nonsmooth potential



(c) Mean smooth



(d) Mean nonsmooth

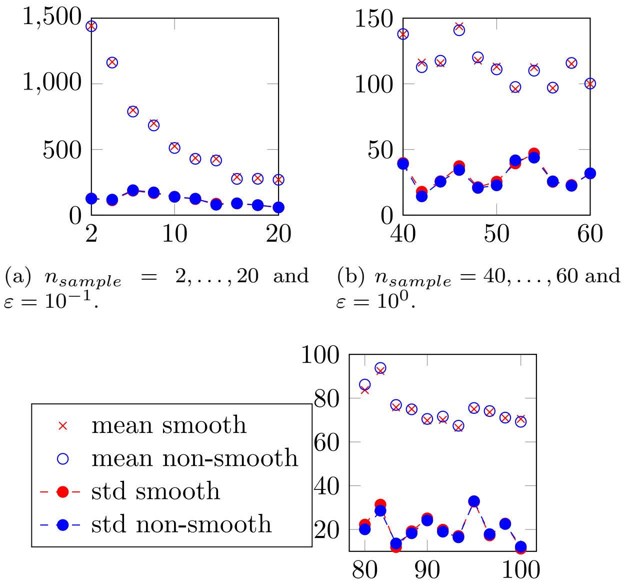


(e) Mean difference

FIG. 7. *Comparison of the smooth* (a) *and nonsmooth* (b) *potentials. In total,* 2000 *points are segmented into* 4 *clusters with* 15 *given sample points for each cluster. With $R = 9$ and the number of used eigenvalues at* 55 *we obtained* 109 *misclassified points in the nonsmooth case and* 137 *in the smooth case. The difference between the mean for nonsmooth* (d) *and smooth* (c) *potentials is shown in* (e). *Negative values indicate that the nonsmooth potential performed better.*

dataset includes descriptions of hypothetical sample species of mushrooms. The goal is to identify each species as edible or nonedible. The latter includes definitely poisonous, unknown edibility, and not recommended. There is no simple or at least safe rule to determine which class a mushroom belongs to. The dataset we used contains 4062 mushroom species with 21 attributes, e.g., one attribute is the cap shape with the attribute values bell, conical, flat, knobbed, and sunken. Similarly to [55] we create a hyperedge whenever one or more species share the same value of a particular attribute. We simply set the entries in the corresponding column in H to 1. Based on this adjacency matrix and a weight vector with constant weight one we obtain the hypergraph Laplacian L_s. For the computation of the hypergraph Laplacian[6] we use the MATLAB functions based on [34]. The results shown in Figure 8 illustrate

[6]The MATLAB code is given under http://www.ml.uni-saarland.de/code/hypergraph/hypergraphcut.zip.



(a) $n_{sample} = 2, \ldots, 20$ and $\varepsilon = 10^{-1}$.

(b) $n_{sample} = 40, \ldots, 60$ and $\varepsilon = 10^{0}$.

(c) $n_{sample} = 80, \ldots, 100$ and $\varepsilon = 10^{0}$.

FIG. 8. *Comparison of the misclassification of the smooth* (a) *and nonsmooth* (b) *potentials for the* mushroom *hypergraph example. We vary the number of sample points and see that for both schemes the results behave similarly and the misclassification reduces. Here the y-axis shows the number of misclassified points while the x-axis shows the number of samples.*

that our approach utilizing the hypergraph Laplacian allows for a solution to the segmentation problem. The performance both for the smooth and the nonsmooth potentials gets better with an increasing number of samples. The difference between both is almost negligible even though the nonsmooth potential gives slightly better results for small sample sizes but at a higher cost due to the nonlinear iteration at its core. The parameters for both methods are chosen as $\omega_0 = 10^5$, $\tau = 0.1$ $c = (3/\varepsilon) + \omega_0$, and $\nu = 10^{-3}$.

Additionally, we employ the semisupervised learning technique used in [34] based on the hypergraph Laplacian [55] where we minimize the function

$$\operatorname{argmin}_u \frac{1}{2} \|u - f\|_2^2 + \frac{\lambda}{2} u^T L_s u, \tag{39}$$

where f is a vector containing the known information just as in the modified Allen–Cahn case. In [34] the authors introduce methods based on proximal mappings that we do not discuss here. We compare here only to the basic approach for solving (39) by solving the system

$$(I + \lambda L_s)\, u = f.$$

In Figure 9, we show the performance of solving (39) for two values of λ as well as the solution of the smooth Allen–Cahn equation for two different numbers of eigenvalues used to approximate L_s. We show the misclassification averaged over 10 runs as well as the standard deviation. The cost of the scheme given in (39) is the solution of a linear system with L_s, which is more expensive than the proximal-maps-based approaches given in [34]. The second example is also taken from the UCI machine learning

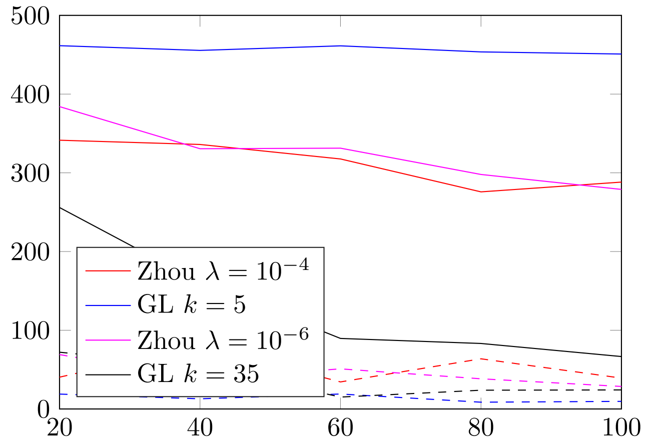


FIG. 9. *The number of misclassified points for the* `mushroom` *dataset. We use two values of λ for the traditional approach (labeled Zhou)*[7] *and vary the number of eigenvectors for the Allen–Cahn equation with smooth potential (labeled GL). Dashed lines indicate the standard deviation of the corresponding experiments. Here the y-axis shows the number of misclassified points while the x-axis shows the number of eigenvectors used.*

repository [41] and is the so-called `student performance` data set as introduced in [19]. The data are given for 395 students with attributes ranging from family size to the job of the parents. All in all 30 attributes are given with three additional columns noting the grades for the first period, the second period, and the final grade. We follow the approach given in [19] where these three performance values can be considered as attributes and hence are embedded into the hypergraph. Again for all 30 attributes one or more pupils share a hyperedge whenever they share an attribute value. Additionally, we include hyperedges for the pupils with the same grades based on the first and/or second period. We always run 5 tests for each scenario and show the mean in Figure 10. The parameters for this example are given via $\omega_0 = 10^8$, $\varepsilon = 10^{-2}$, $\tau = 0.1$, $c = (3/\varepsilon) + \omega_0$, and $\nu = 10^{-6}$. We here classify with respect to the third performance value. The first class is generated for values of the third performance index being less than 10 and the second class for all values greater or equal to 10. We also show the difference in the eigenvalues of the hypergraph Laplacian and the graph Laplacian using a weight matrix W. In order to generate the matrix W we take the feature vector for each of the 395 pupils and use (1). In Figure 11 we show the smallest nonzero eigenvalues of the two Laplacians as well as the misclassification for both the hypergraph and the graph applied to the school example. The parameters are set to $\omega_0 = 10^8$, $\varepsilon = 10^{-2}$, $\tau = 0.1$, $c = (3/\varepsilon) + \omega_0$, and $\nu = 10^{-6}$. It can be seen that the segmentation improves with an increasing number of eigenvectors and we note that we have chosen the same parameters as for the hypergraph Laplacian. It is not clear whether this parameter constellation is the best possible as in this setup the hypergraph Laplacian outperforms the graph Laplacian.

Multiclass segmentation. We again use an example from the UCI machine learning repository. In particular, we focus on the `zoo` dataset introduced in [26]. This dataset contains 101 individuals with 18 attributes such as the number of legs or whether they have hair. The segmentation is made into 7 classes that are already

[7]The Laplacian introduced by Zhou [55] is used in (39).

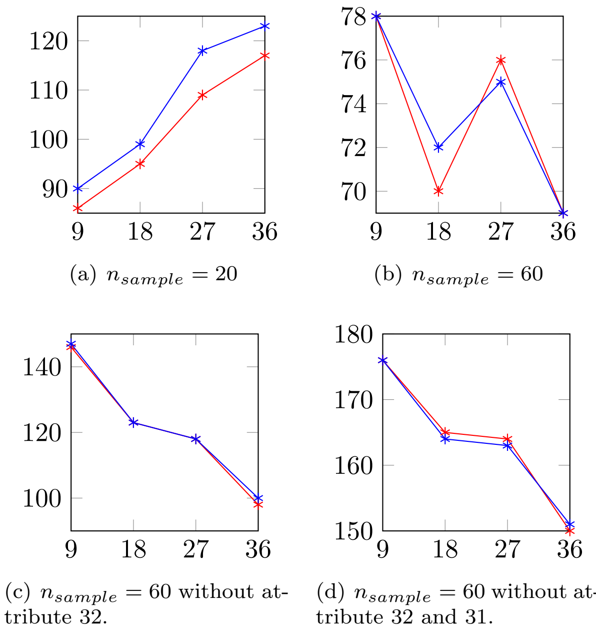


FIG. 10. *Comparison of the misclassification of the smooth (blue) and nonsmooth (red) potentials for the* `student performance` *hypergraph example. We vary the number of sample points and see that for both schemes the results behave similarly and the misclassification is reduced. Here the y-axis shows the number of misclassified points while the x-axis shows the number of eigenvectors.*

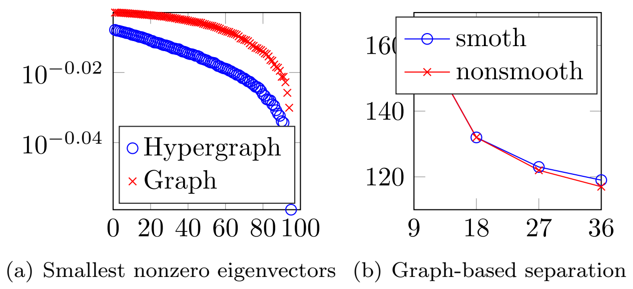
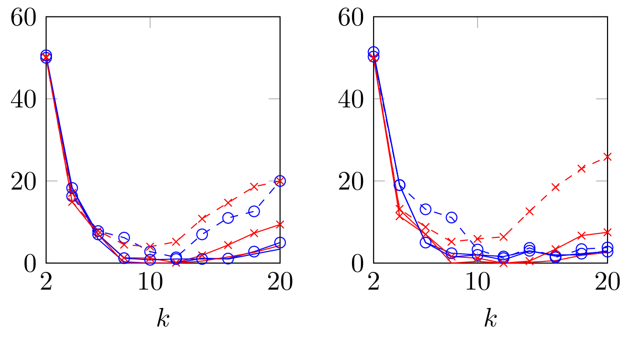


FIG. 11. *Comparison of the magnitude of the eigenvalues of the hypergraph Laplacian and the graph Laplacian applied to the school example. The right picture shows the misclassification (y-axis) for the graph-Laplacian-based segmentation using an increased number of eigenvalues (x-axis) and a sample size* $n_{sample} = 40$.

prespecified. We want our algorithm to segment the data into these 7 classes given only a small number of samples from each class. Figure 12 shows the results for a small number of samples for each class as well as a varying number of eigenvectors of the hypergraph Laplacian. We also test two different values of the interface parameter ε. The results for the nonsmooth potential tend to be slightly better than for the smooth potential, especially when the number of eigenvectors grows. We have set the parameters to $\omega_0 = 100$, $\varepsilon = 10^{-1}$, $\tau = 0.01$, $c = (3/\varepsilon) + \omega_0$, and $\nu = 10^{-4}$ in this example.



(a) $\varepsilon = 10^0$ (b) $\varepsilon = 10^{-1}$

FIG. 12. *Comparison of the misclassification of the mean of* 10 *runs of the smooth versus nonsmooth potentials. Blue lines represent the nonsmooth and the red ones the smooth potentials. Dashed lines correspond to one given sample point, solid lines with markers to two given sample points, and solid linear markers to three given sample points per class. Shown is the total misclassification against the number of eigenvectors used. The left plot is for* $\varepsilon = 10^0$ *and the right one for* $\varepsilon = 10^{-1}$.

Conclusions and outlook. We have shown that diffuse interface methods while already being very powerful can be further generalized. We illustrated that nonsmooth potentials are a viable option for the separation of data. While the computations become more expensive due to the nonlinearity that is treated with the semi-smooth Newton scheme, the results in many cases show a better performance for the nonsmooth than for the smooth potential. Additionally, we showed that the methods are not limited to the graph Laplacian setup but can successfully be employed for the hypergraph Laplacian. Future work should incorporate more sophisticated eigenvalue methods and our goal is to further investigate different techniques for the segmentation of hypergraphs. Also, the successful MBO scheme [38, 44] has not been tried on hypergraph examples and we aim to do this in future research.

Acknowledgment. The third author would like to thank Cristina Garcia-Cardona for answering questions regarding her work on diffuse interface methods on graphs.

REFERENCES

[1] S. M. Allen and J. W. Cahn, *A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening*, Acta Metall., 27 (1979), pp. 1085–1095.
[2] C. R. Anderson, *A Rayleigh–Chebyshev procedure for finding the smallest eigenvalues and associated eigenvectors of large sparse Hermitian matrices*, J. Comput. Phys., 229 (2010), pp. 7477–7487.
[3] J. W. Barret and J. F. Blowey, *Finite element approximation of a model for phase separation of a multi-component alloy with non-smooth free energy*, Numer. Math., 77 (1997), pp. 1–34.
[4] J. W. Barrett and J. F. Blowey, *An error bound for the finite element approximation of a model for phase separation of a multi-component alloy*, IMA J. Numer. Anal., 16 (1996), pp. 257–287.
[5] A. L. Bertozzi, S. Esedoḡlu, and A. Gillette, *Inpainting of binary images using the Cahn–Hilliard equation*, IEEE Trans. Image Process., 16 (2007), pp. 285–291.
[6] A. L. Bertozzi and A. Flenner, *Diffuse interface models on graphs for classification of high dimensional data*, Multiscale Model. Simul., 10 (2012), pp. 1090–1118.
[7] L. Blank, M. Butz, and H. Garcke, *Solving the Cahn–Hilliard variational inequality with a semi-smooth Newton method*, ESAIM Control Optim. Calc. Var., 17 (2011), pp. 931–954.

[8] L. Blank, M. Butz, H. Garcke, L. Sarbu, and V. Styles, *Allen–Cahn and Cahn–Hilliard variational inequalities solved with optimization techniques*, in Constrained Optimization and Optimal Control for Partial Differential Equations, Springer, Heidelberg, 2012, pp. 21–35.
[9] P. Bochev and R. B. Lehoucq, *On the finite element solution of the pure Neumann problem*, SIAM Rev., 47 (2005), pp. 50–66.
[10] J. Bosch, D. Kay, M. Stoll, and A. J. Wathen, *Fast solvers for Cahn–Hilliard inpainting*, SIAM J. Imaging Sci., 7 (2014), pp. 67–97.
[11] J. Bosch and M. Stoll, *A fractional inpainting model based on the vector-valued Cahn–Hilliard equation*, SIAM J. Imaging Sci., 8 (2015), pp. 2352–2382.
[12] P. Boyanova and M. Neytcheva, *Efficient numerical solution of discrete multi-component Cahn–Hilliard systems*, Comput. Math. Appl., 67 (2014), pp. 106–121.
[13] M. Burger, L. He, and C.-B. Schönlieb, *Cahn–Hilliard inpainting and a generalization for grayvalue images*, SIAM J. Imaging Sci., 2 (2009), pp. 1129–1167.
[14] J. W. Cahn and J. E. Hilliard, *Free energy of a nonuniform system.* I. *Interfacial free energy*, J. Chem. Phys., 28 (1958), pp. 258–267.
[15] T. F. Chan and J. Shen, *Nontexture inpainting by curvature-driven diffusions*, J. Vis. Commun. Image Represent., 12 (2001), pp. 436–449.
[16] T. F. Chan and L. A. Vese, *Active contours without edges*, IEEE Trans. Image Process., 10 (2001), pp. 266–277.
[17] Y. Chen and X. Ye, *Projection Onto a Simplex*, preprint, arXiv:1101.6081, 2011.
[18] F. R. K. Chung, *Spectral Graph Theory*, CBMS Reg. Conf. Ser. Math. 92, AMS, Providence, RI, 1997.
[19] P. Cortez and A. M. G. Silva, *Using data mining to predict secondary school student performance*, in Proceedings of the 5th Future Business Technology Conference, Porto, Portugal, 2008, EUROSIS, Ostend, Belgium, 2008, pp. 5–12.
[20] D. De Fontaine, *An analysis of clustering and ordering in multicomponent solid solutions –* I. *Stability criteria*, J. Phys. Chem. Solids, 33 (1972), pp. 297–310.
[21] I. S. Dhillon, *Co-clustering documents and words using bipartite spectral graph partitioning*, in Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, 2001, pp. 269–274.
[22] J. A. Dobrosotskaya and A. L. Bertozzi, *A wavelet-Laplace variational technique for image deconvolution and inpainting*, IEEE Trans. Image Process., 17 (2008), pp. 657–663.
[23] C. M. Elliott and H. Garcke, *On the Cahn–Hilliard equation with degenerate mobility*, SIAM J. Math. Anal., 27 (1996), pp. 404–423.
[24] S. Esedoḡlu and Y.-H. R. Tsai, *Threshold dynamics for the piecewise constant Mumford–Shah functional*, J. Comput. Phys., 211 (2006), pp. 367–384.
[25] D. J. Eyre, *Unconditionally gradient stable time marching the Cahn–Hilliard equation*, in Computational and Mathematical Models of Microstructural Evolution, MRS Proc. 529, Warrendale, PA, 1998, pp. 39–46.
[26] R. S. Forsyth, *PC/BEAGLE User's Guide*, BUPA Medical Research, Brisbane, 1990.
[27] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, *Spectral grouping using the Nyström method*, IEEE Trans. Pattern Anal. Mach. Intell., 26 (2004), pp. 214–225.
[28] C. Fowlkes, S. Belongie, and J. Malik, *Efficient spatiotemporal grouping using the Nyström method*, in Proceedings of the 2001 IEEE Conference on Computer Vision and Pattern Recognition, Vol. 1, IEEE Computer Society, Los Alamitos, CA, 2001, pp. I–231.
[29] M. A. Freitag and A. Spence, *A tuned preconditioner for inexact inverse iteration applied to Hermitian eigenvalue problems*, IMA J. Numer. Anal., 28 (2008), pp. 522–551.
[30] J. Gallier, *Spectral Theory of Unsigned and Signed Graphs. Applications to Graph Clustering: A Survey*, preprint, arXiv:1601.04692, 2016.
[31] C. Garcia-Cardona, E. Merkurjev, A. L. Bertozzi, A. Flenner, and A. G. Percus, *Multiclass data segmentation using diffuse interface methods on graphs*, IEEE Trans. Pattern Anal. Mach. Intell., 36 (2014), pp. 1600–1613.
[32] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins Stud. Math. Sci., 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[33] M. Hein and T. Bühler, *An inverse power method for nonlinear eigenproblems with applications in* 1*-spectral clustering and sparse PCA*, in Advances in Neural Information Processing Systems, Curran, Red Hook, NY, 2010, pp. 847–855.
[34] M. Hein, S. Setzer, L. Jost, and S. S. Rangapuram, *The total variation on hypergraphs - learning on hypergraphs revisited*, in Advances in Neural Information Processing Systems, Curran, Red Hook, NY, 2013, pp. 2427–2435.

[35] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Stand., 49 (1952), pp. 409–436.

[36] M. Hintermüller, M. Hinze, and M. H. Tber, *An adaptive finite-element Moreau–Yosida-based solver for a non-smooth Cahn–Hilliard problem*, Optim. Methods Softw., 26 (2011), pp. 777–811.

[37] M. Hintermüller, K. Ito, and K. Kunisch, *The primal-dual active set strategy as a semismooth Newton method*, SIAM J. Optim., 13 (2002), pp. 865–888.

[38] 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., 73 (2013), pp. 2224–2246.

[39] R. Lehoucq, D. Sorensen, and C. Yang, *ARPACK Users' Guide*, SIAM, Philadelphia, 1998.

[40] Y. Li, D. Jeong, J.-i. Choi, S. Lee, and J. Kim, *Fast local image inpainting based on the Allen–Cahn model*, Digit. Signal Process., 37 (2015), pp. 65–74.

[41] M. Lichman, *UCI Machine Learning Repository*, https://archive.ics.uci.edu/ml/, (2013).

[42] X. Luo and A. L. Bertozzi, *Convergence Analysis of the Graph Allen–Cahn Scheme*, Technical Report, Department of Mathematics, University of California Los Angeles, Los Angeles, 2016.

[43] E. Merkurjev, C. Garcia-Cardona, A. L. Bertozzi, A. Flenner, and A. G. Percus, *Diffuse interface methods for multiclass segmentation of high-dimensional data*, Appl. Math. Lett., 33 (2014), pp. 29–34.

[44] E. Merkurjev, T. Kostić, and A. L. Bertozzi, *An MBO scheme on graphs for classification and image processing*, SIAM J. Imaging Sci., 6 (2013), pp. 1903–1930.

[45] Y. Oono and S. Puri, *Study of phase-separation dynamics by use of cell dynamical systems. I. Modeling*, Phys. Rev. A (3), 38 (1988), pp. 434–453.

[46] J. C. Schlimmer, *Concept Acquisition through Representational Adjustment*, Ph.D. thesis, Department of Information and Computer Science, University of California, Irvine, CA, 1987.

[47] C.-B. Schönlieb and A. L. Bertozzi, *Unconditionally stable schemes for higher order inpainting*, Commun. Math. Sci, 9 (2011), pp. 413–457.

[48] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, *The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains*, IEEE Signal Process. Mag., 30 (2013), pp. 83–98.

[49] A. Szlam and X. Bresson, *A Total Variation-Based Graph Clustering Algorithm for Cheeger Ratio Cuts*, Technical report, Department of Mathematics, University of California Los Angeles, Los Angeles, 2009.

[50] J. E. Taylor and J. W. Cahn, *Linking anisotropic sharp and diffuse surface motion laws via gradient flows*, J. Stat. Phys., 77 (1994), pp. 183–197.

[51] Y. van Gennip, N. Guillen, B. Osting, and A. L. Bertozzi, *Mean curvature, threshold dynamics, and phase field theory on finite graphs*, Milan J. Math., 82 (2014), pp. 3–65.

[52] U. von Luxburg, *A tutorial on spectral clustering*, Statist. Comput., 17 (2007), pp. 395–416.

[53] X. Wang and Q. Du, *Modelling and simulations of multi-component lipid membranes and open membranes via diffuse interface approaches*, J. Math. Biol., 56 (2008), pp. 347–371.

[54] L. Zelnik-Manor and P. Perona, *Self-tuning spectral clustering*, in Advances in Neural Information Processing Systems 17, MIT Press, Cambridge, MA, 2004, pp. 1601–1608.

[55] D. Zhou, J. Huang, and B. Schölkopf, *Learning with hypergraphs: Clustering, classification, and embedding*, in Advances in Neural Information Processing Systems 19, MIT Press, Cambridge, MA, 2006, pp. 1601–1608.

[56] D. Zhou and B. Schölkopf, *A regularization framework for learning from graph data*, in ICML Workshop on Statistical Relational Learning, ACM, New York, 2004, pp. 132–137.