Active Contour Methods on Arbitrary Graphs Based on Partial Differential Equations

Christos Sakaridis, Nikos Kolotouros, Kimon Drakopoulos, and Petros Maragos

Abstract

This chapter formulates and compares two approaches that have been developed to discretize over arbitrary graphs the partial differential equations (PDEs) of classical active contour models with level sets, which have been widely used in image analysis and computer vision. The first approach takes a finite difference path and proposes geometric approximations of the fundamental continuous differential operators that are involved in active contour PDEs, namely gradient and curvature, on graphs with arbitrary vertex and edge configuration. The second approach leverages finite elements to approximate the solution of these PDEs on graphs with arbitrary vertex configuration that constitute a triangulation. We present numerical algorithms and compare both approaches while using them to successfully apply the popular models of geodesic active contours (GACs) and active contours without edges (ACWE) to arbitrary 2D graphs for graph cluster detection and image segmentation.

Index Terms

Active contours on graphs, graph segmentation, discretized partial differential equations.

I. Introduction

EVOLUTION of curves via active contour models has been applied extensively in computer vision for image segmentation. In the standard image setting which involves a regular grid of pixels, the discretization of partial differential equations (PDEs) governing the motion of active contours is well-established [1] and ensures proper convergence of the contour to object boundaries. Recently, active contour models based on level set PDE formulations have been extended [2]–[4] to handle more general inputs in the form of graphs whose vertices are arbitrarily distributed in a two-dimensional Euclidean space. In particular, the input in this case consists of:

1) an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \left\{\mathbf{v}_i \in \mathbb{R}^2 : i = 1, \ldots, n\right\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, and
2) a real-valued function $I : \mathcal{V} \to \mathbb{R}$ defined on the vertices of the graph, which resembles the image function in the standard grid setting. For the rest of the paper, we will use the term image function to refer to I.

Applications of segmentation of such graphs span not only image processing, but also geographical information systems and generally any field where data can assume the form of a set of pointwise samples of a real-valued function. The arbitrary spatial structure of these graphs poses a significant challenge to the discretization of active contour PDEs.

This chapter presents two fundamentally distinct approaches that have been developed to accomplish the aforementioned discretization. The first approach, which is introduced in [2] and [3], takes a *finite difference* path and proposes geometric approximations of the continuous differential operators that are involved in active contour PDEs, namely *gradient* and *curvature*, on graphs with arbitrary vertex and edge configuration. The second approach, which is proposed in [4], leverages *finite elements* to approximate the solution of these PDEs on graphs with arbitrary vertex configuration that constitute a triangulation. Both approaches have been used in [3] and [4] to successfully apply the widely used models of geodesic active contours (GACs) [5] and active contours without edges (ACWE) [6] to arbitrary 2D graphs for graph and image segmentation.

The chapter is structured as follows. Section II reviews related work on active contours, graph-based morphology and segmentation, and PDE-based methods on graphs, and provides the necessary background on the employed active contour models. Section III outlines the finite difference approach of [3] for active contours on graphs. It is composed of Section III-A on the geometric gradient approximation on arbitrary graphs and its asymptotic consistency and accuracy in the limit of infinite vertices for the class of random geometric graphs [7], Section III-B on the geometric curvature approximation and its convergence in probability, and Section III-C on improved Gaussian smoothing on graphs for initialization of the GAC model through normalization. Section IV outlines the finite element approach of [4]. More specifically, Section IV-A discusses the problem formulation and analyzes the key aspects of the finite element approximation for active contour models. Section IV-B presents an extension of the previous framework for locally constrained active contour models that can additionally be used for speeding up contour evolution. Last, Section V presents experimental results of the two approaches on segmentation from the translation of GACs [5] and ACWE [6] to arbitrary graphs created from regular images or containing geographical data and Section VI recapitulates the main components of the chapter and provides a discussion on future research directions.

C. Sakaridis is with the Department of Information Technology and Electrical Engineering, ETH Zürich, Switzerland. N. Kolotouros is with the Department of Computer and Information Science, University of Pennsylvania, USA. K. Drakopoulos is with Marshall School of Business, University of Southern California, USA. P. Maragos is with the School of Electrical and Computer Engineering, National Technical University of Athens, Greece.

II. Background and Related Work

Active contour models for curve evolution towards image edges originate from "snakes" [8]. These early approaches could not in general handle topological changes of the contour, for instance splitting into two disjoint parts to detect the boundaries of two distinct objects. The level set method [1] and its related discretized PDEs solved this problem and enabled improved alternative approaches such as the geometric active contour model [9] which was initially introduced and subsequently complemented to establish the GAC model [5]. The former model involves two forces that govern curve motion: a balloon force that expands or shrinks it, and a curvature-dependent force that maintains its smoothness. The latter model adds an extra spring force that attracts the contour towards salient image edges. Both methods embed the active contour as a *level set* of a function $u(x, y, t)$, which is termed the embedding function and is the unknown in the PDE that models curve evolution, allowing the use of numerical schemes of the type proposed in [1]. The formulation of the PDE of the GAC model is the following:

$$\frac{\partial u}{\partial t} = \underbrace{\kappa \left\| \nabla u \right\| g(I)}_{\text{curvature}} - \underbrace{c \left\| \nabla u \right\| g(I)}_{\text{balloon}} + \underbrace{\nabla u \cdot \nabla g(I)}_{\text{spring}}, \tag{1}$$

where κ is the curvature of the level sets of u at time t, $g(I)$ is an image-dependent function that stops the contour at edges, commonly referred to as the edge stopping function, and c is a constant.

The curvature-dependent force is also used for regularization in ACWE [6], which abandon the aforementioned edge-driven paradigm and leverage the Mumford-Shah functional [10] to formulate a piecewise constant segmentation model, with automatic detection of interior contours and reduced sensitivity to initialization. The formulation of the ACWE model is the following:

$$\begin{gathered}\frac{\partial u}{\partial t} = \delta_\epsilon(u) \left(\mu\kappa - \nu - \lambda_1 (I - c_1(u))^2 + \lambda_2 (I - c_2(u))^2 \right), \\ c_1(u) = \text{average}(I) \text{ in } \{u \geq 0\},\ c_2(u) = \text{average}(I) \text{ in } \{u < 0\},\end{gathered} \tag{2}$$

where κ is the curvature of the level sets of u at time t, δ_ϵ is a regularized version of the Dirac δ function, and μ, ν, λ_1 and λ_2 are positive constants. This model is further studied in [11], where the authors prove that the global minimizers of the original nonconvex problem for given values of c_1 and c_2 can be recovered via solving a convex reformulation of it.

Graphs have long been connected to image processing, in part through their study in terms of mathematical morphology. The application of morphological transforms on neighborhood graphs was established in [12], while a wide variety of graph structures, algorithms for their construction and early applications in computer vision were surveyed in [13]. The notion of structuring element in classical morphology was extended to graphs in [14], where the proposed structuring graph enables a generalization of neighborhood functions on a graph beyond the one induced by its set of edges. Morphological operators on graphs have been studied further in [15], where the lattice of the subgraphs of a graph is considered in order to define filters that treat the graph as a whole.

Recently, several works, including [2], [3], [16]–[27], have focused on the construction of PDE-based rather than algebraically defined morphological operators on graphs, which are then used as one of the basic ingredients to define active contour models as well as other PDE-based optimization schemes on graphs. All these works are based on the definition of a gradient operator on graphs. However, [16]–[27] work on weighted graphs and they all start from defining a *discrete* gradient at a vertex as a vector whose dimensionality is the same as the cardinality of the vertex's neighborhood. This type of gradient is a difference operator, which *replaces* the original continuous gradient operator and enables the adaptation of continuous PDE schemes on graphs by defining analogous partial difference equations. Earlier works that used partial difference equations on graphs that mimic the variational formulation of boundary value problems include [28], [29]. This class of approaches invariably sacrifices consistency of the resulting scheme with the original, continuous PDE formulation, as it removes the link between the discrete graph domain and the underlying continuum; even the initial continuous definitions of the operators in [17] are eventually substituted by discrete counterparts in order to apply the proposed optimization framework to graphs. The foundation of these approaches can be based on the theory of *discrete calculus* [30], [31], which provides a framework for reformulating continuous energies on graphs such that the corresponding solutions share similar properties [32]–[37]. On the other hand, [2], [3] consider unweighted graphs and *approximate* the original, continuous gradient at each vertex. Their approach effectively constitutes a generalized numerical scheme for solving the original, continuous active contour PDE on graphs that have arbitrary spatial vertex configuration and hence greater complexity than the standard 2D image grid, as was described in Section I. Thus, their approach contributes towards a *discretized calculus on arbitrary graphs*, in the spirit of [38] who discretized classical PDEs of mathematical physics on regular grid graphs. They establish approximations of the gradient and curvature operators on graphs which are *asymptotically consistent* in the limit of large graphs, in order to ensure a stable *discretization* of level set-based active contour models on arbitrary 2D graphs, as has been previously achieved for the 2D image grid. The asymptotic upper bound that is established in [3] for the error of their geometric gradient approximation on random geometric graphs is the first of its kind.

A different class of approaches to graph segmentation which has gained a lot of interest in the image processing community is based on graph cuts. These approaches, in contrast to [3], [4], usually operate on a regular image grid and define weighted edges between image pixels based on certain cues like spatial or appearance proximity, in order to find a cut of minimal cost for the resulting weighted graph. The cost of a cut is normalized in [39] so that balanced partitions are preferred. Approximate solutions to multi-label problems are proposed in [40], guaranteeing constant-factor optimality. A link between geodesic active contours and graph cuts is established in [41], where the graph is constructed so that the cost of the cut corresponds to the contour's length under the induced anisotropic metric, and this link is extended to the arbitrary graph setting in [2]. The random walk algorithm of [42] assigns unlabeled pixels to user-defined seeds interactively. Efficient algorithms for watershed-like segmentation that are formulated as graph cuts are introduced in [43], [44]. The power watershed framework of [45] unites and generalizes several graph-based optimization methods for image segmentation by expressing their energies in a common, parametric form.

The analysis of consistency of algorithms operating on point clouds or graphs which are constructed from point clouds has received a lot of interest, especially for machine learning tasks such as clustering. The focus of this line of research is on proving that optimizers of functionals which are defined on the discrete input converge to optimizers of the limiting functional which is defined on the underlying continuum as the number of points or graph vertices goes to infinity. The consistency of k-means is studied in [46]. More recently, [47]–[49] examine spectral clustering and graph Laplacians in terms of consistency by analyzing the convergence of the respective eigenvalues and eigenvectors. The works of [50]–[52] on the consistency of various graph-cut methods are more closely related to the analysis in [3]: they define a spatial scale parameter, which controls the connectivity of the graph through the edge weights and depends on the number of vertices, and derive certain conditions on this parameter in order for the respective graph-cut algorithm to be consistent. This analysis is performed in the setting of pointwise convergence in [50], [51], whereas [52] obtain results on Γ-convergence. In comparison, the analysis of consistency of the approximations in [3] also involves a spatial scale parameter, the radius of the graph, which is presented in Definition 1, with similar function as in the above works. In a slightly different setting, the work in [53] proves Γ-convergence of the graph-based Ginzburg-Landau functional that was developed for binary clustering in [18] to the discrete anisotropic total variation that models the min-cut cost; in this case, the spatial scale parameter pertains to the scale of the diffuse interface that is induced by the Ginzburg-Landau functional and it also depends on the number of vertices. An important distinction between these works and [3] is that the latter proves consistency at the level of operators that are used in active contour models on graphs, not at the level of active contour algorithms themselves. Given that the aforementioned approaches do not cover the case of input which is presented in Section I and includes an image function I defined on the vertices of the graph, the study of the consistency of level set-based active contour algorithms on graphs constitutes an interesting topic for future research; theoretical results in [3] are a promising first step in this direction.

Finally, the approach in [3] bears some resemblance to unsupervised clustering of data that are represented as graphs embedded in Euclidean domains, where techniques based on non-negative matrix factorization [54]–[56] have been applied successfully. Nonetheless, the framework of [3] is not directly applicable to this setting, since the input is limited to the set of vertex locations $\mathcal{V}$ and it does not include an image function I defined on the vertices (cf. Section I), which is essential for formulating active contour models. In the same sense, [3] cannot be extended to stochastic block models [57] for graph segmentation, given that these models do not associate each vertex with a scalar value (which would model the image function) either.

III. Active Contours on Graphs via Geometric Approximations of Gradient and Curvature

A. Geometric Gradient Approximation on Graphs

The first term of level set-based active contour evolution models to be approximated is the gradient of the bivariate embedding function. A general method is thus developed in Sakaridis et al. [3] for calculating the gradient of a real-valued, bivariate function that is implicitly defined on a continuous domain, although its values are known only at a finite set of 2D points, which constitute the vertices of the graph. The context is the same as in the preceding work of Drakopoulos and Maragos [2].

1) Formulation: Compared to the gradient approximations proposed in [2], the geometric gradient approximation of [3] leverages the local spatial configuration of vertices in the formulation of the approximation. More specifically, it introduces the concept of the *neighbor angle*, i.e. the angle around a vertex which is "occupied" by each of its neighbors. The motivation for this approach comes from the following lemma in bivariate calculus.

Lemma 1. *The gradient of a differentiable function* $u : \mathbb{R}^2 \to \mathbb{R}$ *at point* $\mathbf{x}$ *is*

$$\nabla u(\mathbf{x}) = \frac{\displaystyle\int_0^{2\pi} D_\phi u(\mathbf{x})\,\mathbf{e}_\phi\,d\phi}{\pi}, \tag{3}$$

where $\mathbf{e}_\phi$ *is the unit vector in direction* ϕ *and* $D_\phi u(\mathbf{x})$ *is the directional derivative of* u *at* $\mathbf{x}$ *in this direction, defined by*

$$D_\phi u(\mathbf{x}) = \lim_{h\to 0} \frac{u(\mathbf{x}+h\mathbf{e}_\phi)-u(\mathbf{x})}{h}.$$

Based on Lemma 1, the goal is to approximate the gradient at a vertex of the graph by substituting the integral

$$\mathcal{I} = \int_0^{2\pi} D_\phi u(\mathbf{x})\,\mathbf{e}_\phi\,d\phi \tag{4}$$

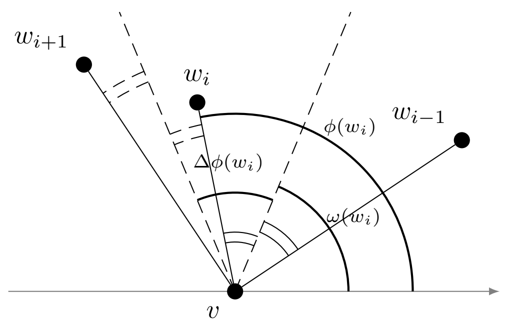


Fig. 1. Angles $\phi(w_i)$, $\Delta\phi(w_i)$ and $\omega(w_i)$. Figure source: [3].

with a sum over all the neighbors of the vertex. To this end, let us first introduce several key concepts.

The Euclidean distance between vertices v and w of a graph $\mathcal{G}$ is denoted by $d(v, w)$ and the unit vector in the direction of the edge vw starting at v is denoted by $\mathbf{e}_{vw}$. We define $\phi(w) \in [0, 2\pi)$ as the angle between the vector $\mathbf{e}_{vw}$ and the horizontal axis, as in Fig. 1. A vertex v will be alternatively denoted by $\mathbf{v}$ to declare its position vector. Moreover, we denote by $\mathcal{N}(v)$ the set of neighbors of v in $\mathcal{G}$, with cardinality $N(v)$. For the sake of brevity in notation, this cardinality will be written simply as N. We write $\mathcal{N}(v) = \{w_1, w_2, ..., w_N\}$ so that the angles $\phi(w_i)$ are in ascending order. Based on this ordering, we define the angle around v "occupied" by w_i, which is called *neighbor angle*, as

$$\Delta\phi(w_i) = \begin{cases} \dfrac{\phi(w_{i+1}) - (\phi(w_N) - 2\pi)}{2} & \text{if } i = 1, \\ \dfrac{\phi(w_1) + 2\pi - \phi(w_{i-1})}{2} & \text{if } i = N, \\ \dfrac{\phi(w_{i+1}) - \phi(w_{i-1})}{2} & \text{otherwise.} \end{cases} \tag{5}$$

In a similar fashion, we define the angle corresponding to the bisector between two consecutive neighbors as

$$\omega(w_i) = \begin{cases} \dfrac{\phi(w_i) + \phi(w_N) - 2\pi}{2} & \text{if } i = 1, \\ \dfrac{\phi(w_i) + \phi(w_{i-1})}{2} & \text{otherwise.} \end{cases} \tag{6}$$

A visual representation of the neighbor angle is provided in Fig. 1. Using the above notation, the geometric gradient approximation at v is given by the following formula:

$$\nabla u(v) \approx \frac{\displaystyle\sum_{i=1}^{N} \frac{u(w_i) - u(v)}{d(v, w_i)} \mathbf{e}_{vw_i} \Delta\phi(w_i)}{\pi}. \tag{7}$$

The directional derivative term in (3) is approximated by the difference quotient of the function along each edge. On the other hand, the angle differential is handled through the neighbor angles, which effectively constitute a Voronoi tessellation of the circle around v, created from its neighbors. The reasoning behind this approach is to use information about the change of u along each particular direction that comes from the neighbor which is *closest* to this direction.

If the neighbor angles were not taken into account, we would place equal importance on all neighbors of v and return to an approximation similar to the weighted sum that was introduced in [2]:

$$\nabla u(v) \approx \frac{\displaystyle\sum_{i=1}^{N} \frac{u(w_i) - u(v)}{d(v, w_i)} \mathbf{e}_{vw_i}}{N}. \tag{8}$$

2) Convergence for Random Geometric Graphs: In the remaining theoretical analysis of this section, we mainly focus on a certain type of graphs, namely *random geometric graphs* [7].

Definition 1. *A random geometric graph (RGG) $\mathcal{G}(n, \rho(n))$ is comprised of a set $\mathcal{V}$ of vertices and a set $\mathcal{E}$ of edges. The set $\mathcal{V}$ consists of n points distributed uniformly at random and independently in a bounded region $D \subset \mathbb{R}^2$. The set $\mathcal{E}$ of edges is defined through the radius $\rho(n)$ of the graph: an edge connects two vertices v and w if and only if their distance is at most $\rho(n)$, i.e. $d(v, w) \leq \rho(n)$.*

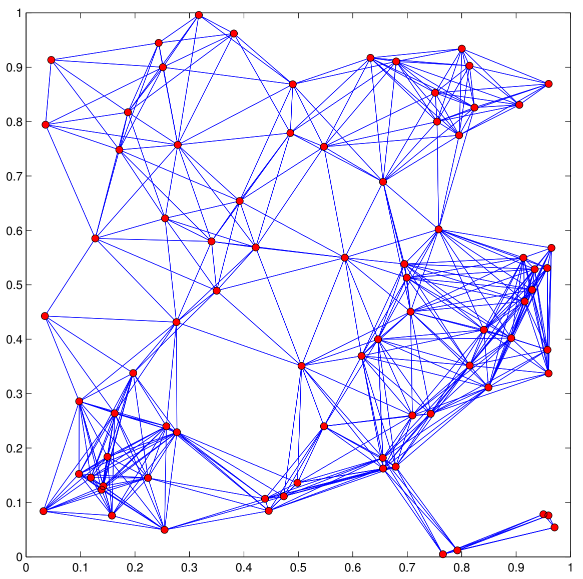


Fig. 2. A random geometric graph embedded in $D = [0,1]^2$, with $n = 80$ vertices and radius $\rho = 0.25$. Figure source: [3].

An instance of an RGG is given in Fig. 2. For RGGs, the approximation of (7) converges in probability to the true value of the gradient as the number of vertices increases, under some conditions on the radius which constrain the density of the graph. Before stating the related theorem, we remind the reader of some definitions for the asymptotic notations which are used in the analysis.

Definition 2. *Let f and g be two non-negative functions. Then,*

$$\begin{aligned} f(n) \in O(g(n)) &\Leftrightarrow \exists k > 0\ \exists n_0\ \forall n \geq n_0 :\ f(n) \leq kg(n), \\ f(n) \in \Theta(g(n)) &\Leftrightarrow f(n) \in O(g(n)) \wedge g(n) \in O(f(n)), \\ f(n) \in o(g(n)) &\Leftrightarrow \forall k > 0\ \exists n_0\ \forall n \geq n_0 :\ f(n) < kg(n), \\ f(n) \in \omega(g(n)) &\Leftrightarrow \forall k > 0\ \exists n_0\ \forall n \geq n_0 :\ f(n) > kg(n). \end{aligned}$$

Theorem 1. *Let $u : \mathbb{R}^2 \to \mathbb{R}$ be a differentiable function and $\mathcal{G}(n, \rho(n))$ an RGG embedded in $D = [0,1]^2$, with $\rho(n) \in \omega\left(n^{-1/2}\right) \cap o(1)$. For every vertex v of $\mathcal{G}$, the gradient approximation of (7) converges in probability to $\nabla u(v)$.*

The proof of Theorem 1 is provided in [3].

3) Asymptotic Analysis of Approximation Error: Note that [2] also proved that two of their gradient approximations converge in probability to the true value of the gradient. Going one step further, [3] obtain an asymptotic bound on the *rate* of convergence to the true gradient as the size of the RGG grows large. Since the framework in RGGs is stochastic, their result involves the *expectation* of the approximation error.

Let us denote the error in approximating $\mathcal{I}$ with

$$\mathcal{S} = \sum_{i=1}^{N} \frac{u(w_i) - u(v)}{d(v, w_i)} \mathbf{e}_{vw_i} \Delta\phi(w_i) \tag{9}$$

by $\mathcal{E} = \mathcal{S} - \mathcal{I}$. Comparing the two expressions, we deduce that the approximation with $\mathcal{S}$ is threefold:

1) Directional derivatives along edges are approximated with difference quotients.
2) The approximate value for the directional derivative along each edge is used as a constant estimate for all the directions "falling into" the respective neighbor angle.
3) The unit vector in the direction of each edge is also used for all the directions corresponding to the respective neighbor angle.

The calculation of the error that is performed in the proof of the following theorem involves construction of intermediate expressions between $\mathcal{S}$ and $\mathcal{I}$, bounding the magnitudes of the resulting differences and combining these individual bounds using the triangle inequality.

Theorem 2. *Let $u : \mathbb{R}^2 \to \mathbb{R}$ be a differentiable function and $\mathcal{G}(n, \rho(n))$ an RGG embedded in $D = [0,1]^2$, with $\rho(n) \in \omega\left(n^{-1/2}\right) \cap o(1)$. For the gradient approximation error at every vertex v of $\mathcal{G}$, it holds that*

$$\mathrm{E}[\|\mathcal{E}\|] \in O\left(\rho(n) + \frac{1}{n\,\rho^2(n)}\right). \tag{10}$$

The full proof of Theorem 2 is given in [3]. The radius of the graph is effectively the factor that determines the strictness of the asymptotic bound. To provide better intuition, we study the case when $\rho(n) \in \Theta\left(n^{-a}\right),\ a \in (0, 1/2)$. Substituting in (10), we obtain

$$\mathrm{E}[\|\mathcal{E}\|] \in O\left(n^b\right),\ b = \begin{cases} -a & \text{if } a \in \left(0, \frac{1}{3}\right], \\ -1 + 2a & \text{if } a \in \left(\frac{1}{3}, \frac{1}{2}\right). \end{cases} \tag{11}$$

The strictest upper bound is $O\left(n^{-1/3}\right)$, it is achieved for $a = 1/3$ and it constitutes a trade-off between minimizing the first error term, which calls for small radii, and the other two terms, which require more neighbors and consequently larger radii. On the other hand, when $a \notin (0, 1/2)$, the conditions of Theorems 1 and 2 for $\rho(n)$ are not met and convergence to the true value of the gradient is not guaranteed in general. For instance, for $a = 0$, we get $\rho(n) \in \Theta(1)$, which means that the radius does not approach 0 in the limit. In turn, this implies that the difference quotients are not guaranteed to converge to the respective directional derivatives, since the distance $d(v, w) \leq \rho(n)$ does not go to 0 in the limit. In addition, for $a = 1/2$, it holds that $\rho(n) \in \Theta(n^{-1/2})$ and hence $\rho^2(n) \in \Theta(1/n)$. As a result, the expected number of neighbors is $\mathrm{E}[N] = (n-1)\pi\rho^2(n) \in \Theta(1)$. In other words, the sum $\mathcal{S}$ is finite in the limit, which forbids convergence to the integral $\mathcal{I}$.

In Theorems 1 and 2, the domain D of the RGG is assumed to be the unit square. It is, however, straightforward to generalize the results of both theorems to arbitrary rectangular regions, since their proofs in [3] only use this assumption for calculating the area of D. This generalization involves applying a uniform scaling to both coordinates by $1/\sqrt{|D|}$, so that the transformed region has unitary area. The values of u are also scaled by the same factor. These steps ensure that all terms in (7) remain unaffected by the transformation. The two theorems are then applicable to the transformed input. In order to transfer the results back to the original input, one just needs to scale the radius ρ by the constant factor $\sqrt{|D|}$, which leaves the asymptotic bounds in both theorems unaffected.

4) Practical Application: Despite convergence of the geometric gradient approximation to the true value of the gradient in the case of RGGs, in practice there is a non-negligible error for graphs with finite number of vertices. This error is propagated to the embedding function of the active contour after each update and may be accumulated after several iterations. To mitigate this, Sakaridis et al. [3] apply smoothing filtering on the approximate gradient values at a local, neighborhood level as an empirical means to eliminate potential outliers by taking into account the values at neighboring vertices. This smoothing is also applicable to curvature, as we discuss in Section III-B. The smoothing filter can be either an average or a median filter, receiving as input the set of function values at the vertex itself and all its neighbors. In the case of curvature this is straightforward, while for gradient, each of the two vector components are filtered separately. Application of smoothing filtering is statistically motivated for the case of smooth (differentiable) functions by showing that using the neighbors of a vertex to form an ensemble of estimators of the approximated quantity at that vertex reduces the variance of the estimation compared to the basic approximation while not changing the bias.

Experimental validation of the gradient and curvature approximations as well as the aforementioned smoothing filtering is performed by using closed-form functions defined on RGGs. In the experiments that follow in the rest of Sections III and V, the radius of an RGG is chosen as $\rho(n) = 0.6n^{-1/3}$ unless otherwise mentioned, so as to achieve the strictest asymptotic bound for gradient approximation error according to the results of Section III-A3. Besides, in Sections III-A4 and III-B2, all RGGs are embedded in $[0, 1]^2$. For each graph, the function's gradient and the curvature of its level sets are approximated at each vertex and afterwards the results are filtered with an average or median filter. The analytical expressions of the function's gradient and curvature are then compared to the estimates. Performance is measured using a global, graph-level error metric which is called *relative error* and denoted by e_r. The error at each individual vertex of the graph is defined as the difference between the approximate value and the true analytical value, and the relative error is simply the ratio of the energy of the error signal to the energy of the true signal on the entire graph:

$$e_r = \frac{E_{\text{error}}}{E_{\text{analytical}}}. \tag{12}$$

The relative error of the geometric gradient approximation is evaluated for an isotropic Gaussian on RGGs whose size ranges from 1000 to 10000 vertices. The analytical form of the Gaussian is

$$\exp\left\{-\left[(x - x_0)^2 + (y - y_0)^2\right] / 2\sigma^2\right\}, \tag{13}$$

with $\sigma = 0.25$ and $x_0 = y_0 = 0.5$. Figure 3(a) shows average values of e_r over 10 different graphs for each size to reduce variance in the reported performance. Using either an average or a median filter reduces the relative error substantially irrespective of size. Based on this result, smoothing filtering is generally applied for gradient in practice when performing active contour evolution.

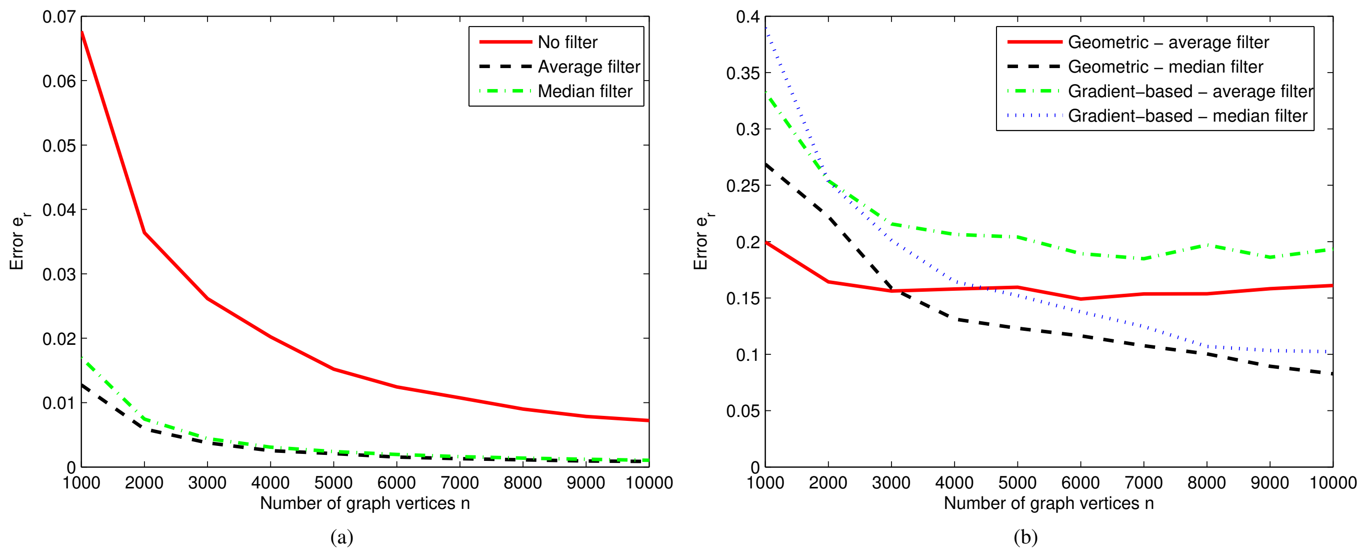


Fig. 3. (a) Relative error of gradient approximations for a Gaussian function defined on RGGs of increasing size. The geometric approximation with no filtering is compared to its filtered versions with an average or median filter. (b) Relative error of curvature approximations for a conic function defined on RGGs of increasing size. All four combinations of type of approximation (geometric or gradient-based) and smoothing filter (average or median) are compared. Figure source: [3].

Another practical consideration about the geometric gradient approximation (7) is that its absolute error increases as the magnitude of the true gradient grows large, i.e. when the function exhibits abrupt variations. This does not pose a problem for the calculation of gradient direction (which is relevant as input for approximating the curvature), since the latter does not depend on the range of the function's variation around the examined vertex. Utilizing all incident edges in the weighted sum of (7) ensures that all available information in the neighborhood of the vertex is used to estimate which direction the gradient points to, as emphasized in [2]. However, the estimated gradient *magnitude* with the geometric gradient approximation (7) is prone to greater error, as it depends on the range of the function's variation. The use of difference quotients in (7) accentuates this effect for dense graphs, where distances between neighboring vertices that appear in the denominator of the quotients approach zero. Thus, the preferable approximation for gradient magnitude in practice is the maximum absolute difference of values of the function along edges that are incident on v, introduced in [2]:

$$\|\nabla u(v)\| \approx \max_{w \in \mathcal{N}(v)} \{|u(w) - u(v)|\}. \tag{14}$$

B. Geometric Curvature Approximation on Graphs

After having devised an approximation scheme for the gradient of an embedding function, the next step is to build upon this scheme in order to estimate the curvature of the level sets of this function. The difference from the gradient case is that the input gradient values for curvature approximation are already approximate themselves, i.e. a *cascaded* approximation is attempted. Therefore, the error in curvature approximation on a graph is expected to accumulate compared to gradient approximation error on the same graph, since the estimated curvature at a vertex inherits the error of the estimated gradients at its neighboring vertices.

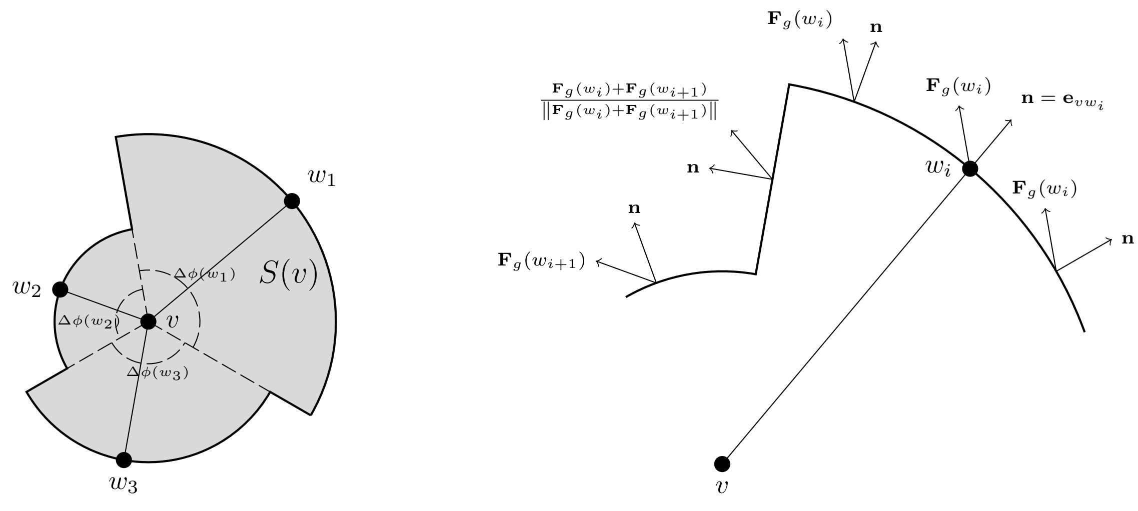
1) Formulation: The geometric curvature approximation that is proposed in [3] is based on the expression of curvature as the divergence of the unit gradient field $\mathbf{F} = \nabla u / \|\nabla u\|$ of the embedding function u:

$$\kappa(v) = \operatorname{div} \mathbf{F}(v), \ \nabla u(v) \neq \mathbf{0}. \tag{15}$$

The integral definition of divergence as

$$\operatorname{div} \mathbf{F}(v) = \lim_{S \to \{\mathbf{v}\}} \frac{\oint\limits_{\Gamma(S)} \mathbf{F} \cdot \mathbf{n} \, d\ell}{|S|} \tag{16}$$

can then be used as a basis for approximating curvature, where S is a region with area $|S|$ and boundary $\Gamma(S)$ and $\mathbf{n}$ is the outward unit normal to this boundary. In [2], the integral in (16) is approximated using a polygonal region to form a finite sum over the neighbors of the vertex (as shown in [2, p. 8]). However, certain arrangements of the neighbors of the examined vertex can lead to regions with ill-defined area, boundary and normals when using the approach of [2], which prevents the presentation of theoretical guarantees for the convergence of this approach.



(a) (b)

Fig. 4. (a) Definition of region $S(v)$ through the neighbor angles. (b) The profile of values of $\mathbf{F}_g$ at a part of the boundary of $S(v)$ which corresponds to neighbor w_i of v. Figure source: [3].

To tackle these problems, the geometric curvature approximation of [3] employs the *neighbor angles* that were introduced in Section III-A, in order to define the region S in (16) in a more compact and principled fashion. Let $S(v, \rho, \theta_0, \theta)$ be a circular sector centered at v, with radius ρ, occupying an angle $\theta > 0$ and whose rightmost radius is in the direction θ_0. For vertex v, $S(v)$ is formed as a union of circular sectors, each of them corresponding to a neighbor of v, as we show in Fig. 4(a). More formally, for each neigbor w of v, the respective circular sector is $S(v, d(v, w), \omega(w), \Delta\phi(w))$. The area of $S(v)$ can then be expressed as

$$|S(v)| = \sum_{i=1}^{N} \frac{\Delta\phi(w_i)}{2}\, d^2(v, w_i). \tag{17}$$

The challenge imposed by this construction of $S(v)$ is the choice of suitable values for $\mathbf{F}$ along the boundary of this region, given only its values at the locations of neighbors of v. The resulting boundary consists of arcs, each of which contains a neighbor of v, and line segments which connect these arcs. The proposed approximation fixes the value of $\mathbf{F}$ along each arc at the geometric gradient approximation computed for the corresponding neighbor w using (7), $\mathbf{F}_g(w)$. Moreover, for every line segment, the normalized mean of the approximate values of $\mathbf{F}$ along the two neighboring arcs is used. The idea is again to use information from the closest vertex, which should be more reliable. We visualize the described configuration in Fig. 4(b).

Using the above approximations, the line integral in (16) is substituted with a sum of simple line integrals over single arcs and line segments, which have closed analytical forms. If we denote the integral over the arc $C_a(w_i)$ containing neighbor w_i by $I_a(w_i)$ and the integral over the line segment $C_l(w_i)$ that connects the arcs $C_a(w_i)$ and $C_a(w_{i+1})$ by $I_l(w_i)$, we obtain

$$I_a(w_i) = d(v, w_i)\, \mathbf{F}_g(w_i) \cdot (\sin(\omega(w_{i+1})) - \sin(\omega(w_i)), \cos(\omega(w_i)) - \cos(\omega(w_{i+1}))) \tag{18}$$

and

$$I_l(w_i) = (d(v, w_{i+1}) - d(v, w_i))\, \frac{\mathbf{F}_g(w_i) + \mathbf{F}_g(w_{i+1})}{\|\mathbf{F}_g(w_i) + \mathbf{F}_g(w_{i+1})\|} \cdot (\sin(\omega(w_{i+1})), -\cos(\omega(w_{i+1}))). \tag{19}$$

The geometric approximation of curvature proposed in [3] is given by

$$\kappa(v) \approx \frac{\sum_{i=1}^{N} I_a(w_i) + I_l(w_i)}{|S(v)|}. \tag{20}$$

While there was no proof of convergence for the curvature approximation of [2], the geometric curvature approximation of [3] is proved to converge in probability for RGGs, as in the geometric gradient approximation case, although the conditions are stronger for curvature. The full proof of the following Theorem 3 is given in the supplementary material of [3].

Theorem 3. *Let $\mathcal{G}(n, \rho(n))$ be an RGG embedded in $D = [0, 1]^2$, with $\rho(n) \in \omega\left(n^{-1/2}\right) \cap o(1)$ and v a vertex of $\mathcal{G}$. If $u : \mathbb{R}^2 \to \mathbb{R}$ is continuously differentiable and $\nabla u(v) \neq \mathbf{0}$, then the curvature approximation of* (20) *converges in probability to $\kappa(v)$.*

An alternative, gradient-based curvature approximation is additionally presented in [3] by leveraging the differential definition of divergence instead of the integral definition of (16). This gradient-based curvature approximation is also proved to converge in probability for RGGs, under slightly stricter conditions than the geometric curvature approximation.

2) Practical Application: The neighborhood-based smoothing filtering that has been introduced in Section III-A4 is also applicable to the output of curvature approximations. In this case, the same type of filter is applied both to the gradient estimates before using them for computing curvature and to the final curvature estimates. In Fig. 3(b), we present the results of an experiment similar to that in Fig. 3(a), this time focusing on the curvature of a function that corresponds to an elliptical cone and comparing average versus median filtering as well as the geometric versus the gradient-based curvature approximation. The form of this function is

$$\sqrt{\frac{(x-x_0)^2}{\alpha^2}+\frac{(y-y_0)^2}{\beta^2}}, \tag{21}$$

where $\alpha = 0.4$, $\beta = 0.3$, $x_0 = -0.25$ and $y_0 = 0.5$. Median filtering is superior: the median-filtered approximations exhibit lower error than the corresponding average-filtered ones over almost the entire range of graph sizes (except for the smallest sizes). More importantly, the relative error of median-filtered curvature is strictly decreasing for increasing graph size both with the geometric and the gradient-based approximation, in contrast to the average-filtered cases, where the error stops decreasing around 4000 vertices. Due to these facts, median filtering is preferred for smoothing gradient and curvature. In addition, the geometric approximation induces a consistently smaller error than the gradient-based approximation.

C. Gaussian Smoothing on Graphs

Apart from the approximations of gradient and curvature, an additional ingredient which is required in particular for the GAC model (1) to operate on graphs is the edge-dependent stopping function g which helps attract the active contour to salient boundaries. The function $g(\|\nabla I_\sigma\|) : \mathbb{R}_+ \to [0,\, 1]$ is typically a decreasing function of the gradient magnitude of a *smoothed* version I_σ of the original image function I, where smoothing acts as a regularization to limit the effect of small local variations on the evolution of the contour. For this smoothing, [2] propose a simple graph-based isotropic Gaussian filter, which is improved in [3] with two alternative formulations that both include normalization to account for nonuniform spatial vertex configurations in the arbitrary graph setting.

The isotropic 2D Gaussian filter with standard deviation σ is defined as

$$G_\sigma(\mathbf{x}) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{\|\mathbf{x}\|^2}{2\sigma^2}\right). \tag{22}$$

Smoothing is performed in [2] via a simple graph-based convolution of (22) with the image function:

$$I_\sigma(\mathbf{v}) = \sum_{\mathbf{w}\in\mathcal{V}} I(\mathbf{w})G_\sigma(\mathbf{v}-\mathbf{w}). \tag{23}$$

The stopping function g is then calculated through the following formula:

$$g(\|\nabla I_\sigma\|) = \frac{1}{1+\frac{\|\nabla I_\sigma\|^2}{\lambda^2}}. \tag{24}$$

However, the arbitrary graph setting introduces nonuniformities: in some parts of the graph, the vertices might be distributed more densely than in other parts. This implies that (23) will operate counter-intuitively, introducing variations to the smoothed image in regions of the graph where the original image function is constant. To demonstrate this behavior, we use a simple binary image of a disk, shown in Fig. 5(a). The result of applying (23) on this image is shown in Fig. 5(b). Not only has the range of image values changed, but the interior of the original disk also exhibits significant variations in image values. This shortcoming is propagated to $\|\nabla I_\sigma\|$ and g, as shown in Fig. 5(e). There is a deviation of g from the ideal value of 1 in the interior of the disk and a variation in its values as well, which means that the gradient of g is not equal to $\mathbf{0}$ in the interior of the disk as it should.

This issue is tackled in [3] by including a normalization term in (23) to address nonuniformities:

$$I_\sigma(\mathbf{v}) = \frac{\sum\limits_{\mathbf{w}\in\mathcal{V}} I(\mathbf{w})G_\sigma(\mathbf{v}-\mathbf{w})}{\sum\limits_{\mathbf{w}\in\mathcal{V}} G_\sigma(\mathbf{v}-\mathbf{w})}. \tag{25}$$

This method is termed normalized Gaussian filtering and its result for the examined disk image is shown in Fig. 5(c). The smoothed image is now similar to the respective output of simple Gaussian filtering in the usual image processing setting with a regular grid. As a result, the profile of the corresponding g function (shown in Fig. 5(f)) meets expectations.

An alternative formulation of Gaussian smoothing proposed in [3] is based on the fact that the input of the stopping function is the gradient of the smoothed image rather than the smoothed image itself. Since the derivatives of the Gaussian filter have

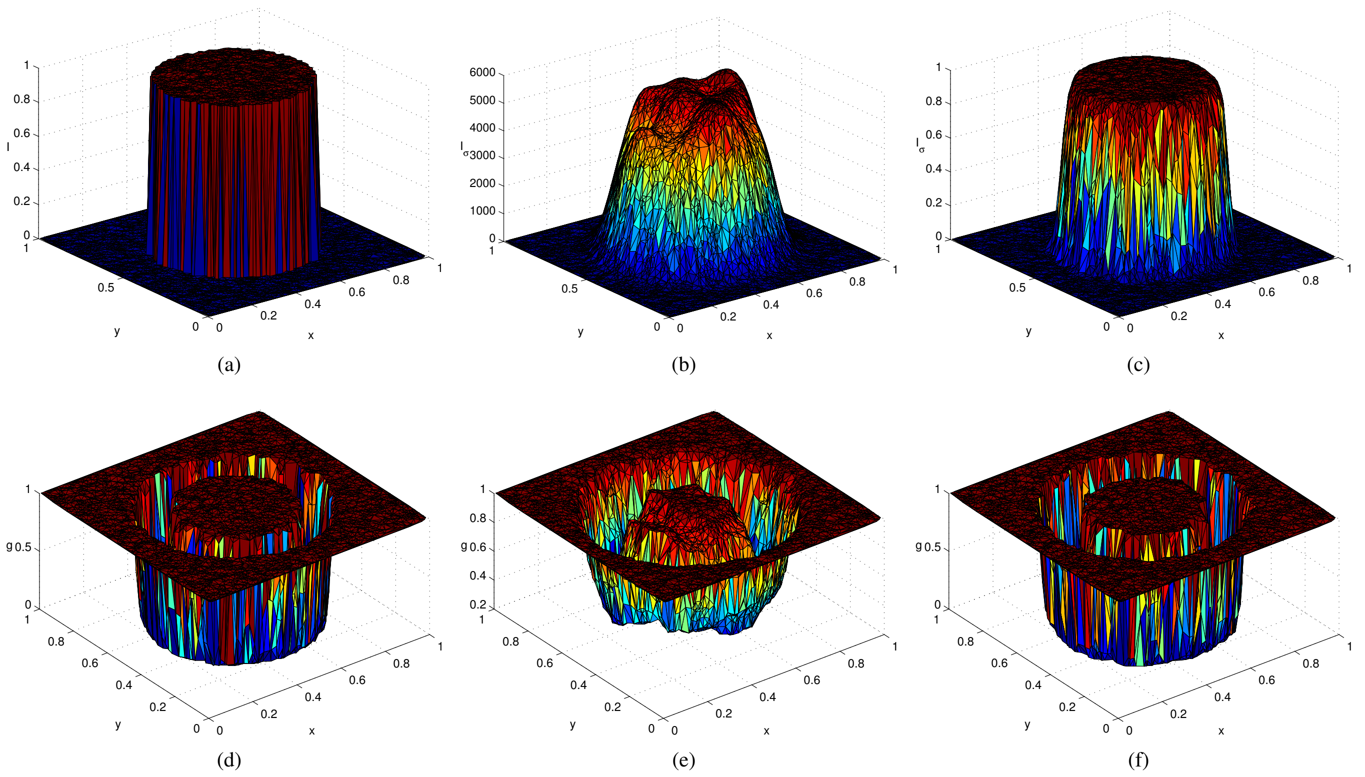


Fig. 5. Comparison of methods for Gaussian smoothing and computation of stopping function g. The original image on the graph (a disk) is shown in (a). The rest of the figure is organized as follows: the two rightmost plots of the top row contain smoothed versions I_σ of the original image and the bottom row contains results for g using the three compared methods. The results in the left column pertain to Gaussian derivative filtering with separate normalization of [3] using $\sigma = 0.02$ and $\lambda = 0.05$, those in the middle column pertain to simple Gaussian filtering of [2] using $\sigma = 0.05$ and $\lambda = 1000$ and those in the right column correspond to normalized Gaussian filtering of [3] with $\sigma = 0.02$ and $\lambda = 0.05$. The approximation of (14) is used to compute the gradient magnitude for simple Gaussian filtering and normalized Gaussian filtering. For all three methods, $\|\nabla I_\sigma\|$ is filtered with a median filter before feeding it to (24) for computing g. Figure source: [3].

closed analytical forms, it is possible to exchange the convolution with the gradient operator and convolve the image directly with Gaussian derivatives in order to obtain the gradient of I_σ without performing numerical approximations. In this case, normalization is not straightforward as in normalized Gaussian filtering: Gaussian derivatives assume both positive and negative values. This difficulty is circumvented by splitting the vertices into two sets, according to the sign of the Gaussian derivative with respect to the processed vertex, and performing separate normalization for each of these sets. This separation can be easily expressed in terms of the vertices' coordinates. If we denote $\mathbf{v} = (v_1,\, v_2)$, then Gaussian derivative filtering with separate normalization is defined as

$$\begin{aligned}&\nabla I_\sigma(\mathbf{v})\\ &= \left(\frac{\displaystyle\sum_{\substack{\mathbf{w}\in\mathcal{V}:\\ w_1\geq v_1}} I(\mathbf{w})\frac{\partial G_\sigma(\mathbf{v}-\mathbf{w})}{\partial x}}{\displaystyle\sum_{\substack{\mathbf{w}\in\mathcal{V}:\\ w_1\geq v_1}} \frac{\partial G_\sigma(\mathbf{v}-\mathbf{w})}{\partial x}} + \frac{\displaystyle\sum_{\substack{\mathbf{w}\in\mathcal{V}:\\ w_1< v_1}} I(\mathbf{w})\frac{\partial G_\sigma(\mathbf{v}-\mathbf{w})}{\partial x}}{\displaystyle -\sum_{\substack{\mathbf{w}\in\mathcal{V}:\\ w_1< v_1}} \frac{\partial G_\sigma(\mathbf{v}-\mathbf{w})}{\partial x}},\; \frac{\displaystyle\sum_{\substack{\mathbf{w}\in\mathcal{V}:\\ w_2\geq v_2}} I(\mathbf{w})\frac{\partial G_\sigma(\mathbf{v}-\mathbf{w})}{\partial y}}{\displaystyle\sum_{\substack{\mathbf{w}\in\mathcal{V}:\\ w_2\geq v_2}} \frac{\partial G_\sigma(\mathbf{v}-\mathbf{w})}{\partial y}} + \frac{\displaystyle\sum_{\substack{\mathbf{w}\in\mathcal{V}:\\ w_2< v_2}} I(\mathbf{w})\frac{\partial G_\sigma(\mathbf{v}-\mathbf{w})}{\partial y}}{\displaystyle -\sum_{\substack{\mathbf{w}\in\mathcal{V}:\\ w_2< v_2}} \frac{\partial G_\sigma(\mathbf{v}-\mathbf{w})}{\partial y}} \right). \qquad (26)\end{aligned}$$

The result of Gaussian derivative filtering with separate normalization on the examined disk image in Fig. 5(d) is at least as satisfactory as in the normalized Gaussian filtering case of Fig. 5(f) and clearly superior to simple Gaussian filtering of [2].

IV. Active Contours on Graphs Using a Finite Element Framework

A. Problem Formulation and Numerical Approximation

In this section we will discuss a method for solving active contour evolution equations on graphs using the Finite Element method as presented by Kolotouros and Maragos [4]. Finite Element Analysis is a powerful framework that enables us to solve complex PDEs in a simple and elegant manner. The main difference between finite difference and finite element methods is that the former try to approximate the differential operators by discretizing them whereas the latter approximate the solution with functions belonging to finite dimensional function spaces. This can be useful, especially in the case of small graphs where the accurate approximation of differential operators is a challenging problem. Additionally, compared to finite differences, the finite element method is more suitable for functions defined on irregular domains or for modeling discontinuities. There is a number of works that attempt to apply the finite element method for solving level set equations. The work in [58] presented a finite element solution to the level set equation by enforcing that the level set function remains a signed distance function throughout the evolution. A solution to the problem that exploits the partition-of-unity property of the finite element shape functions was proposed in [59]. A generic finite element algorithm approach for solving the Hamilton-Jacobi and front propagation equation was developed in [60].

In this section we will consider general active contour models that can be modeled with level set equations of the form

$$\begin{gathered} F(u)\frac{\partial u}{\partial t} = \operatorname{div}\left(G(u)\nabla u\right) + H(u) \\ \nabla u \cdot \mathbf{n} = 0 \text{ on the boundary } \partial\Omega \\ u(x,y,0) = \operatorname{dist}^*(x,y), \end{gathered} \tag{27}$$

where F, G and H are functionals of u and $\operatorname{dist}^*(x,y)$ is the signed distance function from the initial curve. Popular active contour models that can be expressed in the above form are

- *Erosion/Dilation*:
$$F(u) = \frac{1}{\|\nabla u\|}, \quad G(u) = 0, \quad H(u) = c, \tag{28}$$
- *Geometric Active Contours*:
$$F(u) = \frac{1}{g\|\nabla u\|}, \quad G(u) = \frac{1}{\|\nabla u\|}, \quad H(u) = 0, \tag{29}$$
- *Geodesic Active Contours*:
$$F(u) = \frac{1}{\|\nabla u\|}, \quad G(u) = \frac{g}{\|\nabla u\|}, \quad H(u) = \beta, \tag{30}$$

where $g(x,y)$ is the edge stopping function defined in (24).

The core of the Finite Element method is that it converts partial differential equations to integral equations that can then be solved using function approximations. It can be verified that using some simple manipulations, (27) can be converted to an equivalent integral equation

$$\iint\limits_{\Omega} F(u)\frac{\partial u}{\partial t}\phi\, dxdy = -\iint\limits_{\Omega} G(u)\nabla u \cdot \nabla\phi\, dxdy + \iint\limits_{\Omega} H(u)\phi\, dxdy \tag{31}$$

The above form is known as the weak form and it must hold for all functions $\phi \in H^1(\Omega)$. $H^1(\Omega)$ is the Sobolev space consisting of all functions defined in Ω whose first order derivatives –in the distributional sense– belong to $L^2(\Omega)$.

The previous steps did not involve any numerical approximation. The original problem was converted to an integral form that is expected to be equivalent to the original problem. Next, the solution will be approximated using the Galerkin method. Another main idea of the Finite Element analysis is that it does not discretize continuous differential operators using finite differences but instead approximates the solution of the equations with functions belonging to finite dimensional subspaces of $H^1(\Omega)$. Let V_n be a n-dimensional subspace of $H^1(\Omega)$ and $\{\phi_i\}_{i=1}^n$ a basis of V_n. The approximate solution $\bar{u}$ can be written as a linear combination of the basis functions and since the problem is time-dependent, we allow the coefficients of the linear combination to be functions of time. Thus we have

$$\bar{u}(x,y,t) = \sum_{i=1}^{n} c_i(t)\phi_i(x,y). \tag{32}$$

We demand that (31) holds at least for all the functions that belong to the subspace V_n. Since (31) is a linear functional in ϕ and $\{\phi_i\}_{i=1}^n$ form a basis of V_n, this is equivalent to demanding that it holds for each of the basis functions ϕ_i. If we use the fact that

$$\frac{\partial \bar{u}}{\partial t}(x,y,t) = \sum_{i=1}^{n} \dot{c}_i(t)\phi_i(x,y), \tag{33}$$

and substitute in (31) we get

$$\sum_{i=1}^{n} \dot{c}_i \iint\limits_{\Omega} F(\bar{u})\phi_i\phi_j\, dxdy = -\sum_{i=1}^{n} \iint\limits_{\Omega} G(\bar{u})\nabla\phi_i \cdot \nabla\phi_j\, dxdy + \iint\limits_{\Omega} H(\bar{u})\phi_j,\ j = 1,\dots n. \tag{34}$$

This is a non-linear system of ordinary differential equations (ODEs) that can be written in the form

$$\mathbf{A}(\mathbf{c})\dot{\mathbf{c}} = \mathbf{b}(\mathbf{c}), \tag{35}$$

where $\mathbf{A} = \{\mathrm{A}_{ij}\}$ is a $n \times n$ matrix that depends on $\mathbf{c}$ and $\mathbf{b} = \{b_i\}_1^n$, $\mathbf{c} = \{c_i\}_1^n$ are n-dimensional vectors with

$$\mathrm{A}_{ij}(\mathbf{c}) = \mathrm{A}_{ji}(\mathbf{c}) = \iint_\Omega F\left(\sum_{k=1}^n c_k\phi_k\right)\phi_i\phi_j\,dxdy, \tag{36}$$

and

$$b_i(\mathbf{c}) = -\sum_{j=1}^n \iint\limits_\Omega G\left(\sum_{k=1}^n c_k\phi_k\right)\nabla\phi_i\cdot\nabla\phi_j\,dxdy + \iint\limits_\Omega H\left(\sum_{k=1}^n c_k\phi_k\right)\phi_i\,dxdy. \tag{37}$$

The initial condition for the above system of differential equations can be obtained from the projection $\bar{u}(x,y,0)$ of $u(x,y,0)$ onto the subspace V_n, i.e.

$$\bar{u}(x,y,0) = \sum_{i=1}^n \langle u(x,y,0), \phi_i\rangle\phi_i, \tag{38}$$

and thus

$$c_i(0) = \langle u(x,y,0), \phi_i\rangle, \tag{39}$$

where $\langle\cdot,\cdot\rangle$ is the usual inner product defined on $H^1(\Omega)$ with

$$\langle f, g\rangle = \iint\limits_\Omega fg\,dxdy + \iint\limits_\Omega \nabla f\cdot\nabla g\,dxdy. \tag{40}$$

Consider a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$. We assume that the graph has a total of n vertices and each vertex v_i is a point in Ω and can be described by its respective planar coordinates (x_i, y_i). Note that every finite set of points in the plane can be mapped into the unit square by applying a translation followed by a scaling. Thus if we apply an appropriate geometrical transformation we can map the set of vertices to a subset of Ω. For each vertex $v_i \in \mathcal{V}$ we choose a function ϕ_i that belongs to $H^1(\Omega)$ with the following properties:

1) $\{\phi_i\}_{i=1}^n$ are linearly independent, and
2) for any non-adjacent vertices v_i, v_j, $\mathrm{supp}(\phi_i)\cap\mathrm{supp}(\phi_j) = \emptyset$.

We can see that the functions $\{\phi_i\}_{i=1}^n$ form a basis of some subspace S of $H^1(\Omega)$. Also, for reasonably smooth functions ϕ_i, $\mathrm{supp}(\nabla\phi_i)\backslash\mathrm{supp}(\phi_i)$ is a null set and thus $\mathrm{supp}(\nabla\phi_i)\cap\mathrm{supp}(\nabla\phi_j)$ is also a null set. Thus we can conclude that if v_i and v_j are two non-adjacent vertices of $\mathcal{G}$, then $\langle\phi_i,\phi_v\rangle = 0$ and thus ϕ_i, ϕ_j are orthogonal.

Proposition 1. *For any graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ with $v_i = (x_i, y_i) \in \Omega$ there is at least one set of functions $\{\phi_i\}_{i=1}^n$ with the properties 1) and 2).*

For a proof of this fact we refer the reader to [4].

Next we will describe how we can obtain the solution of (35) on a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ equipped with the functions $\{\phi_i\}_{i=1}^n$. First, from (36) we can see that $\mathrm{A}_{ij}(\mathbf{c}) = 0$ if v_i is not a neighbor of v_j. If the number of edges of each vertex is small compared to the total number of vertices then $\mathbf{A}(\mathbf{c})$ is sparse. Additionally the summation in (37) reduces to a summation in the neighborhood of v_i and thus

$$b_i(\mathbf{c}) = -\sum_{v_j\in N_i\cup\{v_i\}} \iint\limits_\Omega G\left(\sum_{k=1}^n c_k\phi_k\right)\nabla\phi_i\cdot\nabla\phi_j\,dxdy + \iint\limits_\Omega H\left(\sum_{k=1}^n c_k\phi_k\right)\phi_i\,dxdy, \tag{41}$$

where N_i is the neighborhood of vertex v_i.

For the rest of the analysis the class of graphs will be restricted to the family of Delaunay Graphs and more specifically the case of Delaunay graphs that are constructed from the Delaunay triangulation of a finite set of points in the unit square. Delaunay triangulation is a method of dividing the convex hull of a set points into triangles that tends to avoid creating sharp triangles, a property that ensures good convergence results for the solution of PDEs. For more details about the Delaunay triangulation and the algorithms used to produce it, we refer the reader to [61]. Images can be thought of as a special case of Delaunay graphs with the vertices being the pixels of the image. In the case of graphs, the image function I is defined as in Section I, i.e. only on the vertices of the graph. For the case of the image gradient ∇I, however, since we will be using numerical integration, it is more convenient to define it as a 2-D function defined on Ω. This function is piecewise constant in each triangle of the triangulation and its value is the gradient of the plane that is formed by the values of I in the three vertices of the triangle. Similarly, $g = g(\|\nabla I\|)$ will also be constant in each triangle of the triangulation.

Let $\mathcal{G}(\mathcal{V},\mathcal{E})$ be a Delaunay graph and $\{T_i\}_{i=1}^m$ the triangles of the Delaunay triangulation. Kolotouros and Maragos [4] approximate the solution with a function belonging to the space of continuous functions on Ω which are linear in each triangle of the triangulation. We will denote this space as S_{lin}. More formally

$$S_{lin} = \{f \in C(\Omega) : f|_{T_i} = a_i x + b_i y + c_i,\ \text{for } i = 1,\ldots,m\}. \tag{42}$$

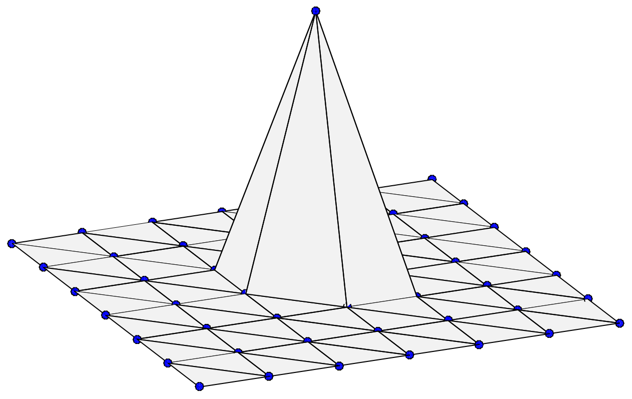
This choice of subspace provides good convergence properties and also simplifies calculations as we shall see later on. After the selection of the subspace we have to choose a basis of S_{lin}. A possible choice satisfying the aforementioned properties are the pyramid functions. For each vertex v_i of the graph we define the function ϕ_i as

- $\phi_i(v_i) = 1$.
- $\phi_i(v_j) = 0$, for $j \neq i$.
- ϕ_i is linear in each triangle.

It is easy to verify that $\text{supp}(\phi_i) \cap \text{supp}(\phi_j) = \emptyset$ if v_i is not a neighbor of v_j and $\phi_i \in S_{lin}$. It can also be proven that $\{\phi_i\}_{i=1}^n$ are linearly independent and thus form a basis of S_{lin}. In Fig. 6a we can see the form of the ϕ_i functions for the case of a rectangular grid. With this choice we can see that ϕ_i overlaps only with the 6 other ϕ_j that correspond to the vertices v_j that are adjacent to v_i. Thus each row of $\mathbf{A}$ contains at most 7 non zero elements and at the same time only 7 different coefficients c_i appear in these expressions. We can think of ϕ_i as functions that interpolate discrete data in a rectangular grid. Assume for example that we have samples of a 2D function f at n points (x_i, y_i) on the plane. By creating the Delaunay graph that corresponds to the above set of points we can construct a continuous function $\bar{f}$ that approximates f as

$$\bar{f}(x, y) = \sum_{i=1}^{N} f(x_i, y_i)\phi_i(x, y). \tag{43}$$

From the above construction and the properties of ϕ_i, it is easy to verify that $\bar{f}(x_i, y_i) = f(x_i, y_i)$ and in each triangle of the triangulation it essentially performs a form of linear interpolation. In Fig. 6b we can see an example of an interpolation of discrete data in a rectangular grid.


(a)

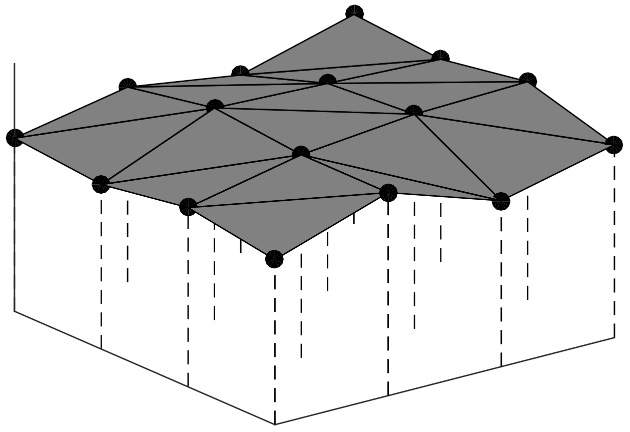

(b)

Fig. 6. (a) Pyramid function ϕ_i centered in node v_i for a rectangular grid. (b) Interpolation of discrete data in a 4×4 rectangular grid using the ϕ_i functions. With solid circles we represent the original discrete data

The final remaining step is the discretization in time. For a system of ODEs we have several options for the approximation of derivatives, each one with its advantages and disadvantages and the choice of a particular method depends on the specific properties of each problem. We want to calculate the solution in a subset of the time interval $[0, +\infty)$ starting from the initial condition until we reach convergence. Let t_k, $k \geq 0$ be the sequence of time points at which we calculate the solution with $t_0 = 0$ and $t_k = k\Delta t$. Here we assume that we use a fixed time step Δt. Also with $\mathbf{c}_k$ we will denote the approximation of $\mathbf{c}(t_k)$. Kolotouros and Maragos [4] used the explicit Euler method which approximates the time derivative with the forward difference

$$\dot{\mathbf{c}}(t_k) = \frac{\mathbf{c}_{k+1} - \mathbf{c}_k}{\Delta t}. \tag{44}$$

If we substitute this into (35) we obtain

$$\mathbf{A}(\mathbf{c}_k)\mathbf{c}_{k+1} = \mathbf{A}(\mathbf{c}_k)\mathbf{c}_k + \Delta t \cdot (\mathbf{c}_k). \tag{45}$$

This reduces to solving a linear system for each time step. This method is easy to implement but puts limits on the choice of time step Δt. It is necessary to choose a very small time step—often in the order of $1/n^2$—to ensure that the curve evolution is stable.

Kolotouros and Maragos [4] also provide an analysis of the complexity of the presented algorithm. For the sake of simplicity let us consider the case of the Delaunay graphs that correspond to images, with the graph structure depicted in Fig. 6a. We assume that we have a $h \times w$ image. The resulting graph will have a total of $n = hw$ vertices. Since each row of $\mathbf{A}$ contains at most 7 nonzero elements, $\mathbf{A}(\mathbf{c}_k)$ can be computed in linear time in each time step. Also due to its sparsity, matrix

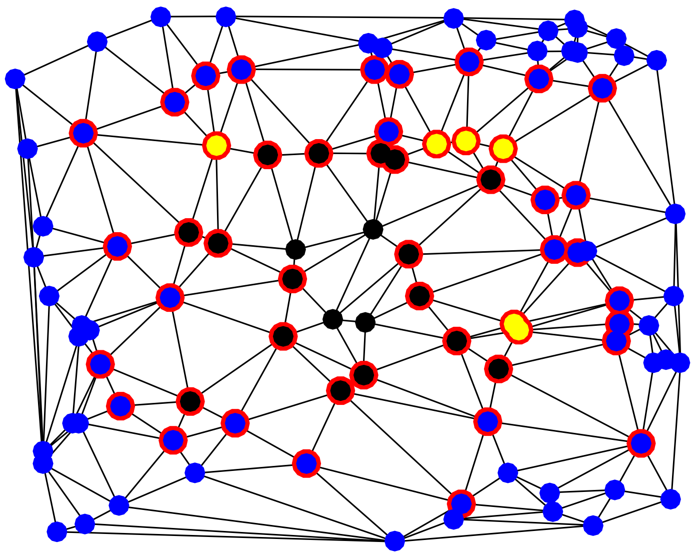


Fig. 7. Illustration of active points for a Delaunay graph with 100 vertices. Blue nodes: Vertices outside the contour. Black nodes: Vertices inside the contour. Yellow nodes: Vertices on the contour. Nodes with red boundary: Active points

multiplication can be performed in linear time. Observing that $\mathbf{b}(\mathbf{c}_k)$ can also be computed in linear time, the right hand side needs $O(n)$ time in each step. With an appropriate node labeling, $\mathbf{A}(\mathbf{c}_k)$ takes the form of a band matrix with a band length of $\min(h,w) = O(\sqrt{n})$. For a $n \times n$ band matrix with band length d the solution of a linear system needs $O(nd^2)$ operations. So $O(n^2)$ operations are required in each time step, which is prohibitive for large images. As far as the time evolution is concerned, the total number of steps until convergence cannot be specified a priori, since it depends on the shape and position of the initial curve. If the initial curve is close to the object boundaries, only a few time steps are needed until convergence.

B. Locally Constrained Contour Evolution

In this part we present an extension of the previous framework to solve curve evolution models of the form

$$\begin{gathered} F(u)\frac{\partial u}{\partial t} = \delta_\epsilon(u)\left(\operatorname{div}\left(G(u)\nabla u\right) + H(u)\right) \\ \nabla u \cdot \mathbf{n} = 0 \text{ on the boundary } \partial\Omega \\ u(x,y,0) = \text{dist}^*(x,y). \end{gathered} \tag{46}$$

where $\delta_\epsilon(x)$ is an approximation of the Dirac δ function, with $\delta_\epsilon \to \delta$, as $\epsilon \to 0$. A typical choice for δ_ϵ is the piecewise constant approximation

$$\delta_\epsilon(x) = \begin{cases} 1/\epsilon, & |x| \leq \epsilon, \\ 0, & |x| > \epsilon \end{cases}. \tag{47}$$

The term δ_ϵ constrains the curve evolution in a small area near the current position of the curve, i.e. the 0-level set. One popular active contour model that can be represented using (46) is the ACWE model of [6].

The first part of the discussion will involve showing how (46) can be approximated on graphs. Define the sets

$$\mathcal{U}_t^+ = \{(x,y) \in \mathcal{V} : u(x,y,t) > 0\}, \tag{48}$$

$$\mathcal{U}_t^- = \{(x,y) \in \mathcal{V} : u(x,y,t) < 0\}, \tag{49}$$

$$\mathcal{U}_t^0 = \{(x,y) \in \mathcal{V} : u(x,y,t) = 0\}. \tag{50}$$

$\mathcal{U}_t^+, \mathcal{U}_t^-$ and $\mathcal{U}_t^0$ are the set of points that are outside, inside and on the curve at time t. Also we define the set of *active points* at time t as

$$\mathcal{U}_t^* = \partial\mathcal{U}_t^+ \cup \partial\mathcal{U}_t^- \cup \mathcal{U}_t^0, \tag{51}$$

where ∂S denotes the boundary of the set $S \subseteq \mathcal{V}$ and is defined by

$$\partial S = \{v \in S : \exists v' \in \mathcal{V} \backslash S \text{ s.t. } v' \sim v\}. \tag{52}$$

The set of active points represent the points that are within "unit" distance from the current position of the active contour. Fig. 7 depicts visually how the active points are computed.

Then—up to a positive scaling factor—$\delta_\epsilon(u)$ can be approximated on $\mathcal{G}$ at time t with

$$\delta_t^{\mathcal{G}}(v) \sim \begin{cases} 1, & \text{if } v \in \mathcal{U}_t^* \\ 0, & \text{elsewhere.} \end{cases} \tag{53}$$

Using the above definition, the evolution equation for (46) can be approximated as

$$\tilde{\mathbf{A}}(\mathbf{c})\dot{\mathbf{c}} = \tilde{\mathbf{b}}(\mathbf{c}), \tag{54}$$

where $\tilde{\mathbf{A}}(\mathbf{c})$ is a $n \times n$ matrix with

$$\tilde{\mathrm{A}}_{ij}(\mathbf{c}) = \begin{cases} \mathrm{A}_{ij}(\mathbf{c}), & \text{if } v_i \in \mathcal{U}_t^* \\ 1, & \text{if } v_i \notin \mathcal{U}_t^* \text{ and } j = i \\ 0, & \text{elsewhere} \end{cases},$$

and $\tilde{\mathbf{b}}(\mathbf{c})$ a n-dimensional vector with

$$\tilde{b}_i(\mathbf{c}) = \begin{cases} b_i(\mathbf{c}), & \text{if } v_i \in \mathcal{U}_t^* \\ 0, & \text{elsewhere} \end{cases}. \tag{55}$$

It is easy to observe that for all non-active vertices v_i, $\dot{c}_i = 0$ and thus the curve evolution is indeed constrained at the subset $\mathcal{U}_t^*$.

The above formulation will help us reduce significantly the computational complexity of each time step. Consider the explicit Euler approximation of (54)

$$\tilde{\mathbf{A}}(\mathbf{c_k})\left(\mathbf{c}_{k+1} - \mathbf{c}_k\right) = \Delta t_k \tilde{\mathbf{b}}(\mathbf{c_k}). \tag{56}$$

For all vertices v_i that are not in the set of active points $(c_i)_{k+1} - (c_i)_k = 0$. So, for all $v_i \in \mathcal{U}_t^*$, $(c_i)_{k+1}$ does not depend on the values $(c_j)_k$, where $v_j \notin \mathcal{U}_{t_k}^*$. Thus the corresponding values $\mathrm{A}_{ij}(\mathbf{c}_k)$ have no influence on the solution of the linear system, so they can be set to 0. Let $\mathbf{A}_k^*$ be a $|\mathcal{U}_{t_k}^*| \times |\mathcal{U}_{t_k}^*|$ submatrix of $\tilde{\mathbf{A}}(\mathbf{c_k})$ that is constructed by keeping only the elements A_{ij} with v_i and $v_j \in \mathcal{U}_{t_k}^*$. Similarly $\mathbf{b}_k^*$ is a $|\mathcal{U}_t^*|$-dimensional vector that is derived from $\mathbf{b}(\mathbf{c}_k)$ by discarding all the elements v_i with $v_i \notin \mathcal{U}_t^*$. If we set $\Delta\mathbf{c}_k = \mathbf{c}_{k+1} - \mathbf{c}_k$ and $\Delta\mathbf{c}_k^* = \mathbf{I}_k^*\Delta\mathbf{c}_k$, where $\mathbf{I}_k$ is the $|\mathcal{U}_{t_k}^*| \times n$ projection matrix with $\mathrm{I}_{ii} = 1$ and $\mathrm{I}_{ij} = 0$ if $i \neq j$ then we get the equivalent linear system

$$\mathbf{A}_k^*\Delta\mathbf{c}_k^* = \Delta t_k \mathbf{b}_k^*. \tag{57}$$

After computing $\Delta\mathbf{c}_k$ from (57) the update rule for the coefficients $\mathbf{c}$ in matrix notation becomes

$$\mathbf{c}_{k+1} = \mathbf{c}_k + \mathbf{I}_k^{\mathrm{T}}\Delta\mathbf{c}_k^*. \tag{58}$$

Most level set evolution models require re-initialization of the embedding function to ensure correct results and there have been several approaches to alleviate this need, as in [62]. In this framework we avoid the need for re-initialization in an indirect way; to ensure the stability of the numerical method and make it possible to use large time steps, we normalize the embedding function after each time step. Specifically, $\bar{u}(x, y, t)$ and consequently the vector of coefficients $\mathbf{c}_k$ is saturated outside the interval $[-r, r]$. A typical choice of r is $r = \max|u_0(x, y)|$ or equivalently $r = \max|\mathbf{c}_0|$. With the use of normalization we can choose a time step $\Delta t_k \sim 1/N$, which is significantly larger than the time step used for the solution of (27). A similar approach cannot be adopted for the solution of the full evolution equation because in each time step the evolution domain covers the whole graph and although the embedding function will be bounded, large time steps will produce noisy artifacts.

As in the previous section, Kolotouros and Maragos [4] provide an estimate for the computational complexity of the proposed algorithm for the case of Delaunay graphs that correspond to images. For a $h \times w$ image with n pixels, each of $\mathcal{U}_{t_k}^+$, $\mathcal{U}_{t_k}^-$ and $\mathcal{U}_{t_k}^0$ is the union of a number of 1D curves. Generally, typical 1D curves in an image grid with n pixels contain $O(\sqrt{n})$ pixels. Thus, in almost all practical cases $\mathcal{U}_{t_k}^*$ will contain $O(\sqrt{n})$ pixels. Since the number of edges in the planar Delaunay graph is bounded by $3n - 6$, a naive calculation of $\mathcal{U}_{t_k}^*$ using the definitions of its components requires $O(n)$ operations. Additionally, it is easy to verify that given the set of active points, the elements of $\mathbf{A}_k^*$ and $\mathbf{b}_k^*$ can be computed in $O(n)$ time. Since $\mathbf{A}(\mathbf{c_k})$ is a sparse band matrix, $\mathbf{A}_\mathbf{k}^*$ will also be a band matrix, but its band length can often be $O(|\mathcal{U}_{t_k}^*|)$. However, using the Reverse Cuthill-McKee algorithm [63] which can be implemented in $O(|\mathcal{U}_{t_k}^*|)$ time we can obtain a permuted matrix with a band length of $O(\sqrt{|\mathcal{U}_{t_k}^*|})$ on average. Thus the solution of the constrained linear system is expected to require $O(n)$ operations. Consequently, we have shown that on average, each time step of the algorithm requires $O(N)$ operations. If we compare this result with the number of operations required for the full curve evolution, we can see that the constrained curve evolution algorithm is faster by an order of magnitude. Its linear complexity with respect to the number of graph vertices makes it feasible to be used in practical applications.

These results point towards a fast implementation of general active contour models that can take the form of (27). The level set-based approach enables us to handle topological changes in the curve in a solid and efficient manner, but introduces an extra computational burden, because we have to evolve a 2D function instead of a curve. The answer to this problem is to try to limit our focus in a small area of interest near the curve—often referred as band—and evolve the embedding function in this subset of the image. Similar methods, called narrow-band methods, were described for the case of images in [64]–[66]. Instead of evaluating the curve evolution on the whole graph, the curve evolution is constrained in a small band near the curve. This approximation is based on the assumption that the embedding function evolves in such a way that points outside or inside the curve will not change status, at least until the moment that the active contour reaches them. The above assumption is valid for the majority of active contour methods, such as simple Erosion/Dilation as well as the Geometric and Geodesic Active Contours.

V. Experimental Results

Using the graph-based approximations of the main differential terms of active contour models with level sets, Sakaridis et al. [3] apply finite difference-based algorithms to solve the respective PDEs that define these models on arbitrary graphs. The input comprises a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and an image function I, as specified in Section I. The two representative active contour models which are used to develop these algorithms are the GAC model [5] presented in (1) and the ACWE model [6] presented in (2). We first present the GAC algorithm and corresponding results and then proceed to ACWE.

The finite difference algorithm of [3] for GACs on graphs includes the following steps:

1) Compute $g(\|\nabla I_\sigma\|)$, using either (25) and (14) or (26) to compute $\|\nabla I_\sigma\|$. In both cases, median filtering is applied to $\|\nabla I_\sigma\|$ before plugging it into (24) for computing g. Then, compute the magnitude of g's gradient using (14) and its direction using (7).
2) Choose a subset X of $\mathcal{V}$ which *contains* the objects to be segmented and initialize the embedding function with the signed distance function from the boundary of X, denoted by u_0. By convention, u_0 is positive inside X.
3) Iterate for $r \in \mathbb{N}$
$$u_r = u_{r-1} + \Delta t((\kappa - c)\,\|\nabla u_{r-1}\|\, g + \nabla g \cdot \nabla u_{r-1}) \tag{59}$$
until convergence. In the difference equation (59), Δt and c are positive constants and κ is the estimated curvature of the level sets of u_{r-1} using (20). The direction of ∇u_{r-1} is computed with (7) and its magnitude with (14). Median filtering is applied both to κ and ∇u_{r-1} before using them in (59).

In practice, after each iteration of step 3 of the algorithm, u_r is smoothed with a median filter before proceeding to the next iteration. The parameters involved in the algorithm are the time step Δt of the difference equation, the balloon force constant c, the scale σ of the Gaussian smoothing filter and parameter λ of the stopping function g (24). Tuning their values depending on the input at hand is pivotal in obtaining satisfactory segmentation results. In the following experiments of Section III, $\Delta t = 0.005$ and $c = 2$ unless otherwise specified.

A simple analysis of the computational complexity of each iteration (59) of the algorithm by examining each term on the right-hand side of (59) and aggregating the individual contributions of all terms shows that the total complexity of each iteration is $\Theta(n + m)$, which is linear in the number of vertices and in the number of edges.

We first discuss and experimentally explore the practical aspect of graph construction when applying such graph-based algorithms on arbitrarily spaced data as well as regular pixel grids. In the case of arbitrary graphs, the raw input often consists only of $\mathcal{V}$ and I, without any information about the edges. This setting leaves us free to choose the type of edge structure of the graph. Sakaridis et al. [3] experiment with two types: RGGs and Delaunay triangulations (DTs) [13]. In the case of regular images on a grid, graph-based active contour methods are still relevant, since the input image can be sampled at a number of arbitrary locations that will serve as the vertices. This brings us to the previous setting where edges can be defined freely. A baseline for such sampling is to place vertices uniformly at random on the image domain. A more sophisticated strategy is to extract vertex locations via watershed transformation. In particular, watershed transformation is applied directly to the gradient of the image and the vertices are placed at the centroids (ultimate erosions) of the resulting superpixels. For the grayscale image with four distinct coins in Fig. 8(a), we compare the results of the GAC algorithm of [3] for all four possible combinations of the aforementioned vertex placement strategies (random, watershed) and choices of edge structure (RGG, DT) in Fig. 8(c)–(f). To ensure a fair comparison, the number of randomly placed vertices is approximately the same as in the watershed case. The most accurate segmentation is achieved with DT and watershed-placed vertices (Fig. 8(f)). Moreover, DT consistently outperforms RGG and watershed-based vertex placement outperforms random placement. Consequently, the DT edge structure is preferable to the random geometric one and usage of watershed transformation to place vertices when a full image is available is preferable to random placement, as it captures image particularities into the spatial structure of the graph. In the rest of the experiments of [3] that are presented in this section, construction of the graphs follows these choices.

Figure 9 presents results of the algorithm of [3] for GACs on graphs on two natural color images coming from the Berkeley Segmentation Dataset BSDS500 [67]. The images were converted to grayscale for applying the algorithm. In general, the algorithm segments the dominant objects in the images successfully, even though background clutter and thin protrusions or concavities of the objects' boundaries sometimes cause minor inaccuracies.

An interesting application of the examined graph segmentation framework is related to geographical data, where the two spatial coordinates are longitude and latitude and the image function can encode information about any type of real-valued signal defined at the vertices of the graph. Such a signal is the signal strength of a cellular network, such as that shown in

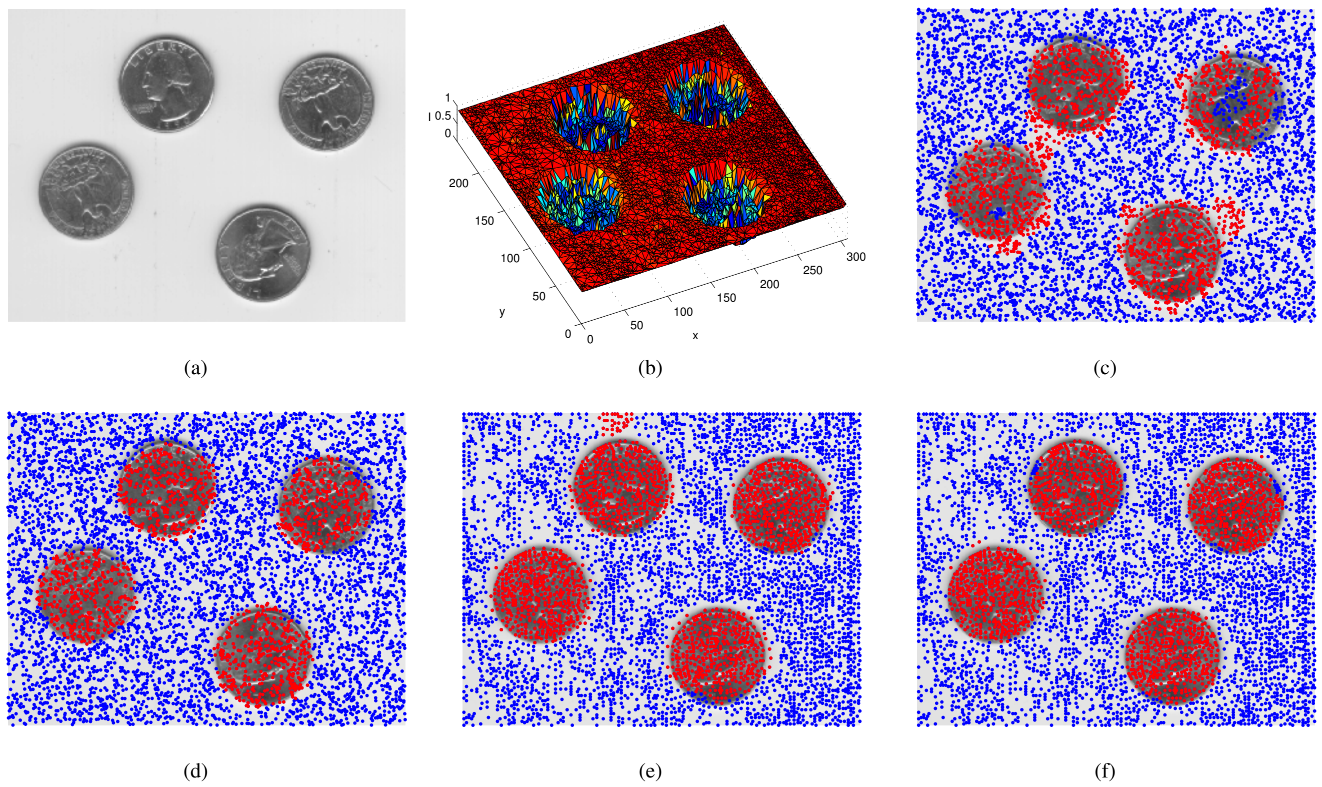


Fig. 8. Segmentation of grayscale objects on graphs with the GAC algorithm of [3]. (a) Full grayscale image with four coins (b) image function on watershed-based DT (c)–(f) final segmentation results overlaid on original image, with segmented objects shown in red and background in blue. We use (c) randomly placed vertices with random geometric structure, $\sigma = 0.02$ and $\lambda = 0.03$, (d) randomly placed vertices with DT structure, $\sigma = 0.02$ and $\lambda = 0.03$, (e) watershed-placed vertices with random geometric structure, $\sigma = 0.008$ and $\lambda = 0.07$, and (f) watershed-placed vertices with DT structure, $\sigma = 0.01$ and $\lambda = 0.07$. The total number of iterations to obtain the final segmentation result is (c) 2200, (d) 4000, (e) 3000 and (f) 4000. Figure source: [3].

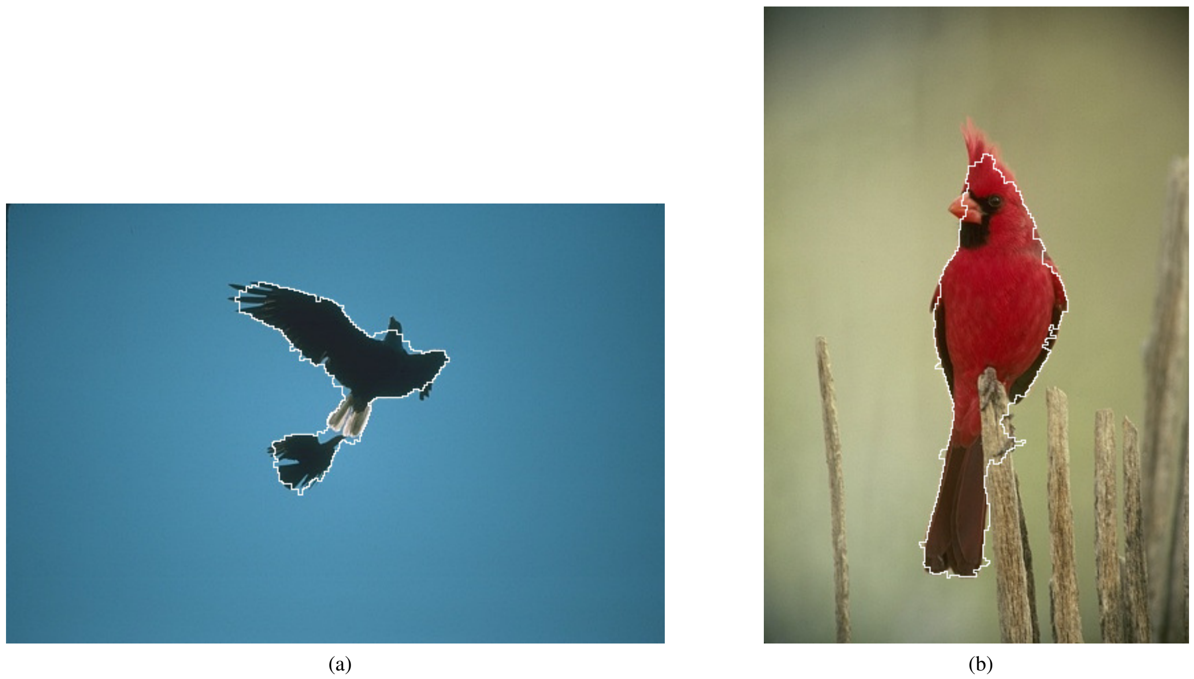


Fig. 9. Segmentation of images from BSDS500 dataset with the GAC algorithm of [3]. The detected boundary, marked white, is defined as the boundary of the union of the watershed superpixels for which the corresponding graph vertex belongs to the final contour's interior. We use $\sigma = 0.005$ and $\lambda = 0.03$ in (a), and $\sigma = 0.01$ and $\lambda = 0.02$ in (b). The total number of iterations to obtain the final segmentation result is (a) 3400 and (b) 5600. Figure source: [3].

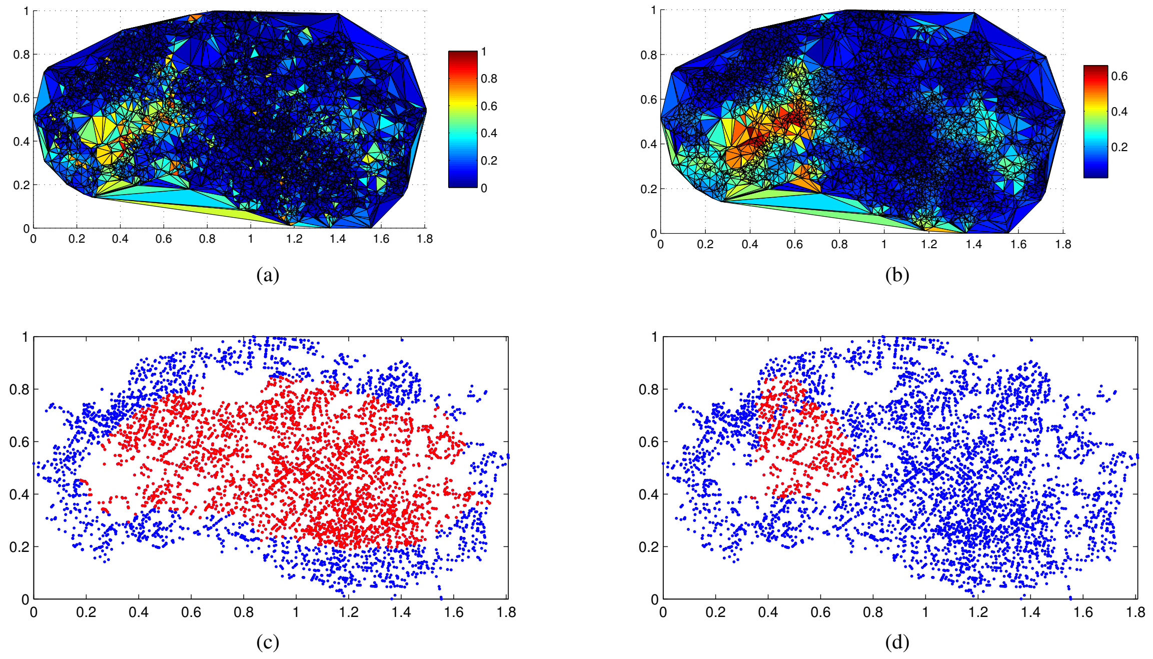


Fig. 10. Segmentation of signal strength data of a cellular network on a graph with the GAC algorithm of [3]. (a) Normalized data on graph (b) Smoothed signal strength (c) Initial contour (d) Final segmentation result after 40000 iterations, with vertices in the contour's interior shown in red and the rest in blue. Figure source: [3].

Fig. 10(a). The coordinates and the values of signal strength are normalized and a very small time step $\Delta t = 10^{-4}$ is used to guarantee convergence, which requires more iterations until termination than in the previous experiments. Other parameters are set as $\sigma = 0.03$, $\lambda = 0.02$ and $c = 20$. The segmented set of vertices in Fig. 10(d) using the GAC algorithm of [3] comprises two regions, the southern of which is characterized by an increased signal strength compared to the rest of the graph. The graph segmentation framework with active contours is tailored for geographical data with arbitrary spatial configuration such as the above case, which grants greater flexibility when collecting measurements.

At this point, we present a short theoretical and experimental comparison between the finite difference-based GAC algorithm of [3] that was presented previously and the finite element method (FEM) for GACs on graphs that is developed by Kolotouros and Maragos [4] and is detailed in Section IV. Even though the algorithm of [4] inherits the advantages of the well-established framework of finite elements, the finite difference algorithm of Sakaridis et al. [3] features greater generality and lower computational complexity per iteration. On the other hand, the FEM algorithm of [4] achieves slightly better segmentation results in practice.

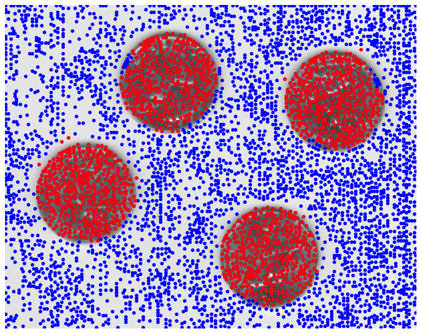
In particular, the finite difference framework of [3] is applicable for any type of connectivity pattern defined through the set of edges $\mathcal{E}$ of the input graph. For instance, when $\mathcal{E}$ is given as part of the input and models connections of vertices that follow an arbitrary, non-local pattern which is known a priori, the framework of [3] is still relevant. On the contrary, the FEM framework of [4] only applies to triangulations; notably, it cannot handle the aforementioned case with arbitrary edge structure, or other commonly used types of graphs such as k nearest neighbor graphs or RGGs. Furthermore, in the case of triangulations, the complexity of each iteration of the finite difference-based GAC algorithm is $O(n)$, i.e. *linear* in the number of vertices n (which follows from the general $\Theta(n + m)$ complexity and planarity of triangulations), whereas the respective complexity of each iteration of the FEM-based GAC algorithm is $O\left(n^2\right)$, i.e. *quadratic* in n.

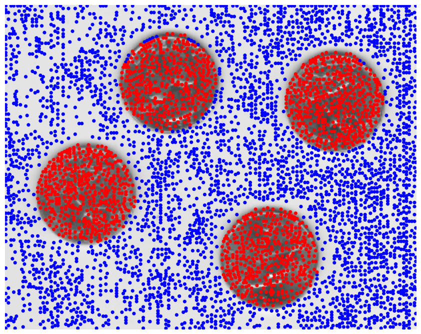
In Fig. 11 we compare the two approaches on the coins image and an image from BSDS500 [67]. The graphs are formed as DTs with watershed-placed vertices. Whenever possible, we use the same or at least similar parameter values for the two approaches, so that a fair comparison is ensured. The results of the finite difference GAC algorithm are presented in Fig. 11(a) and (c) and the respective results of the FEM GAC algorithm are presented in Fig. 11(b) and (d). The two methods demonstrate similar overall performance on both images, with the FEM algorithm producing slightly more accurate boundaries in challenging cases such as the bird's tail.

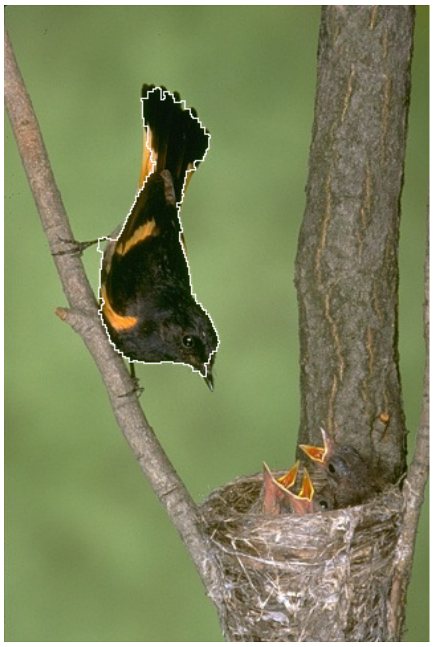
We now proceed to the presentation of the ACWE algorithm of [3] and related results. The finite difference algorithm of [3] for ACWE on graphs includes the following steps:

1) Choose a subset X of $\mathcal{V}$ and initialize the embedding function with the signed distance function from the boundary of X, denoted by u_0. By convention, u_0 is positive inside X.
2) Iterate for $r \in \mathbb{N}$:

$$c_1 = \text{average}\left\{I(v) : u_{r-1}(v) \geq 0\right\},\ c_2 = \text{average}\left\{I(v) : u_{r-1}(v) < 0\right\}, \tag{60}$$


(a)


(b)


(c)

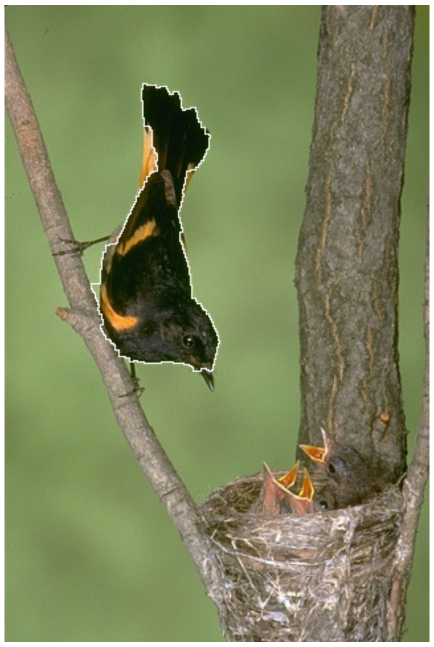

(d)

Fig. 11. Comparison between the finite difference GAC algorithm of [3] and the FEM GAC algorithm of [4]. For the coins image, the foreground is shown in red and the background in blue and we use $\sigma = 0.01$ and $\lambda = 0.07$ for both methods. (a) The result of [3] after 4000 iterations (b) The result of [4] after 301 iterations, using $c = 10$ and $\Delta t = 0.001$. For the bird image, the detected boundary is marked white and we use $\sigma = 0.005$ for both methods. (c) The result of [3] after 4400 iterations, using $\lambda = 0.05$. (d) The result of [4] after 4001 iterations, using $\lambda = 0.07$, $c = 10$ and $\Delta t = 0.001$. Figure source: [3].

$$u_r = u_{r-1} + \Delta t \left(\delta_\epsilon \left(u_{r-1} \right) \left(\mu\kappa - \nu - \lambda_1 (I - c_1)^2 + \lambda_2 (I - c_2)^2 \right) \right) \tag{61}$$

until convergence. In the difference equation (61), Δt, ϵ, μ, ν, λ_1 and λ_2 are positive constants and κ is the estimated curvature of the level sets of u_{r-1} using (20). Moreover, δ_ϵ is the derivative of a regularized version of the Heaviside function which was introduced in [6] as:

$$\delta_\epsilon(x) = H'_\epsilon(x) = \left(\frac{1}{2} \left(1 + \frac{2}{\pi} \arctan\left(\frac{x}{\epsilon} \right) \right) \right)' = \frac{\epsilon}{\pi \left(x^2 + \epsilon^2 \right)}. \tag{62}$$

The standard ACWE model is modified in [3] for color images, so that it uses all three color channels instead of reducing the image to a simpler grayscale version. In order for the averages and Euclidean distances to be meaningful, the original RGB values of the vector-valued image function $\mathbf{I}$ at graph vertices are transformed to CIELAB values, as the latter color space is more perceptually uniform. The update equations are modified to operate on multiple color channels:

$$\mathbf{c}_1 = \text{average} \left\{ \mathbf{I}(v) : \, u_{r-1}(v) \geq 0 \right\}, \; \mathbf{c}_2 = \text{average} \left\{ \mathbf{I}(v) : \, u_{r-1}(v) < 0 \right\} \tag{63}$$

and

$$u_r = u_{r-1} + \Delta t \left(\delta_\epsilon \left(u_{r-1} \right) \left(\mu\kappa - \nu - \lambda_1 \left\| \mathbf{I} - \mathbf{c}_1 \right\|^2 + \lambda_2 \left\| \mathbf{I} - \mathbf{c}_2 \right\|^2 \right) \right). \tag{64}$$

In Fig. 12, we present results that are analogous to Fig. 8(f) and Fig. 9(a) and (b) by replacing the GAC algorithm with the ACWE algorithm and demonstrate the utility of the geometric approximations proposed in [3] for successfully translating various active contour models to graphs. For the color images from BSDS500 in Fig. 12(a) and (c), the modification of ACWE in (63) and (64) is used. The segmentation results are of high quality, e.g. capturing the thin feathers at the eagle's wing very precisely.

Last we present some additional qualitative results when applying the FEM framework for solving more complicated active contour models. Kolotouros and Maragos [4] presented Region-based Geodesic Active Contours (RBGAC), an interactive segmentation algorithm where the users have to specify some initial markers for the foreground and background and then the algorithm iterates between expanding and shrinking contours to compute the segmentation. An example use of this method can be seen in Fig. 13. Also in Fig. 14 we depict a quantitative comparison between RBGAC and several other popular interactive segmentation algorithms.

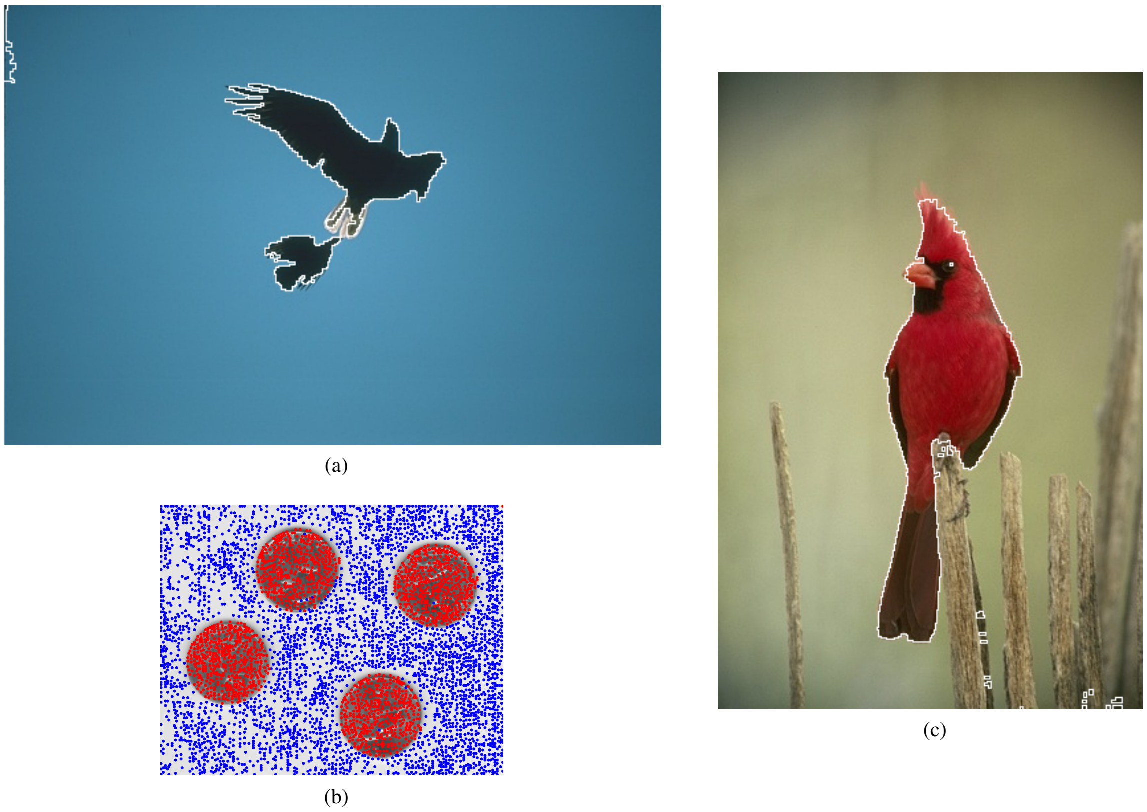


Fig. 12. Segmentation of images with the ACWE algorithm of [3]. (a), (c) Images from BSDS500 with detected boundaries marked white (b) Coins image with detected foreground shown in red and background in blue. We use $\Delta t = 0.05$, $\nu = 0$, $\lambda_1 = \lambda_2 = 1$ and $\epsilon = \text{median}_{vw\in\mathcal{E}}\{d(v,w)\}$ in all cases, $\mu = 0.1$ in (a) and $\mu = 0.5$ in (b) and (c). The total number of iterations to obtain the final segmentation result is (a) 1000, (b) 7000 and (c) 2000. Figure source: [3].

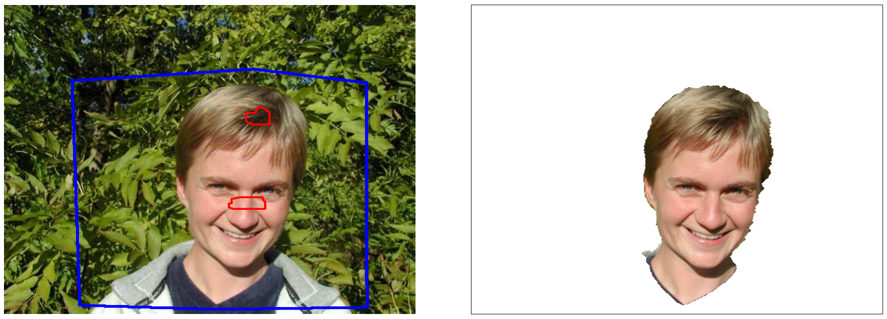


Fig. 13. Illustration of RBGAC algorithm. Left column: Images and seeds. The red curves are the initial contours and the blue polygons mark the regions of interest. Right column: Segmentation results.

VI. Conclusion

In this chapter, we have delineated two approaches to translate level set-based active contour models to the general case of arbitrary graphs, using either finite differences [2], [3] or the finite element method [4] for the required discretization of the employed PDEs. We have overviewed the approximations introduced in [3] for the gradient and curvature terms in active contour models for 2D graphs with arbitrary vertex and edge configuration and presented the main theoretical results of [3] regarding asymptotic consistency and accuracy of these approximations for the class of random geometric graphs. In addition, a detailed presentation of the formulation of the finite element method that is proposed in [4] for the approximate solution of active contour PDEs through discretization on arbitrary 2D Delaunay triangulations has been provided. The effectiveness of the two examined methods has been demonstrated experimentally by using them to apply the popular GAC and ACWE models on regular as well as arbitrary graphs and successfully segment regular images as well as unorganized 2D geographical data.

Although the framework of [3] applies to any type of graph, the presented theoretical analysis considers random geometric graphs, whose definition simplifies convergence proofs for the proposed approximations. However, judging from segmentation

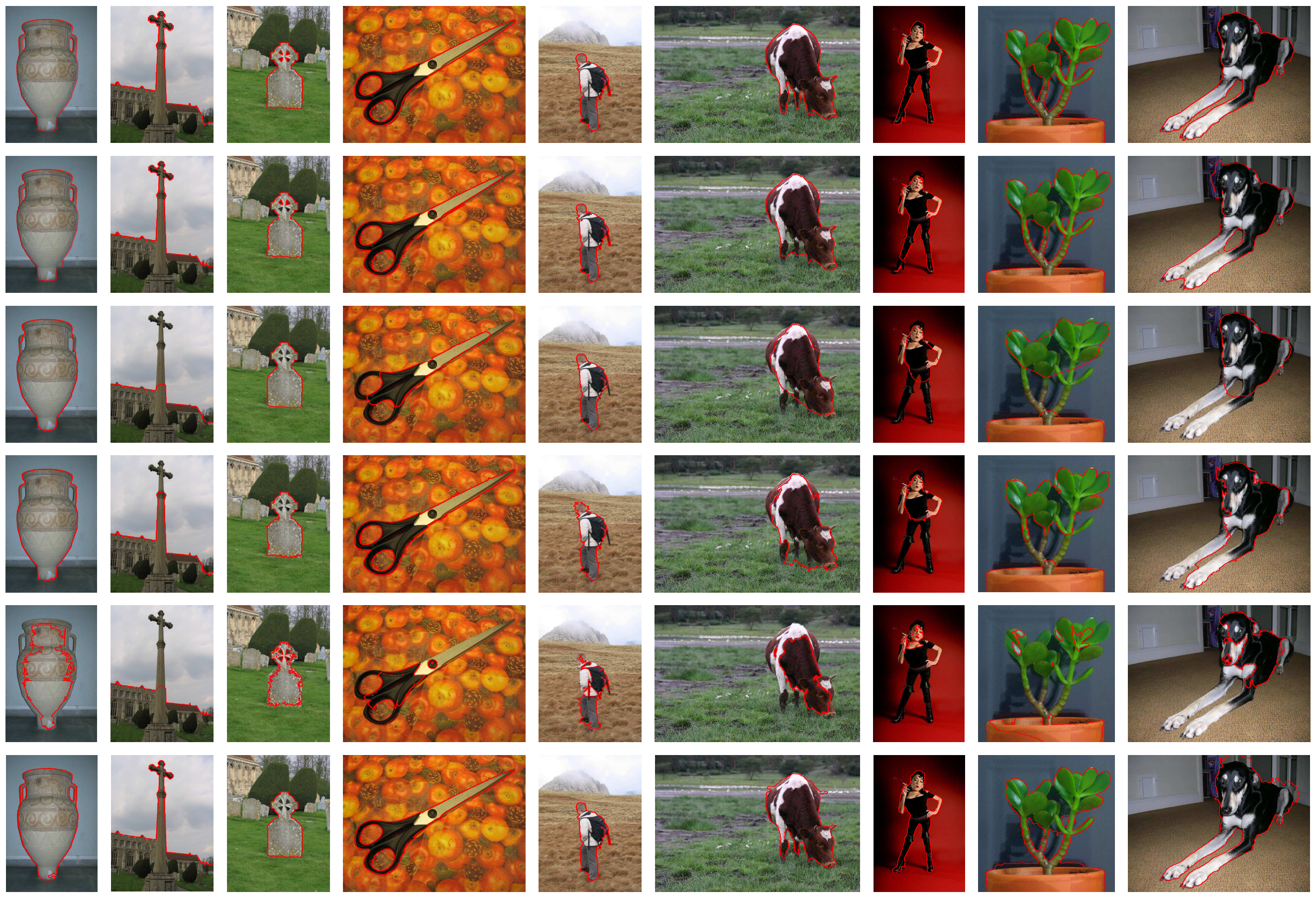


Fig. 14. Segmentation Results. Columns 1-5: GrabCut Dataset. Columns 6:9: grabcut Dataset. From top to bottom: Ground Truth, segmentation results from GrabCut, Laplacian Coordinates, Power Watershed, ACWE and RBGAC. Figure source: [4].

quality, more regular graph structures, such as Delaunay triangulations, lead to more accurate results. Therefore, an interesting direction in this line is the theoretical study of the approximations of [3] on graphs formed as Delaunay triangulations, ideally establishing similar asymptotic consistency results to those shown in [3].

Last but not least, both methods [3], [4] are formulated for the case of 2D graphs. While this type of input is sufficient to model signals such as regular images and geographical profiles, there are several areas of application that involve signals defined in higher-dimensional—often 3D—Euclidean spaces, including medical image processing and geology. Thus, it is worth generalizing the above methods to more dimensions. For the approach of [3] in particular, this requires generalizing the approximation formulae for gradient and curvature. With regard to gradient, the main challenge is the calculation of neighbor angles (cf. (5)) in higher dimensions. For each vertex v, this would involve the calculation of solid angles which are induced by the Voronoi tessellation of the ambient d-dimensional space that is defined by the set of neighbors $\mathcal{N}(v)$, analogously to Fig. 1. The general formula of [68] for measuring simplicial solid angles in Euclidean spaces of arbitrary dimensionality would likely have to be utilized to this end. As far as curvature is concerned, the quantity of interest is the mean curvature and we foresee that the geometric type of approximation we highlighted in Section III-B will require meticulous calculations in order to be extended to more dimensions, whereas the gradient-based approximation we briefly mentioned, which uses the differential definition of divergence, is easier to extend.

References

[1] S. Osher and J. A. Sethian, “Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations,” *Journal of Computational Physics*, vol. 79, pp. 12–49, 1988.

[2] K. Drakopoulos and P. Maragos, “Active Contours on Graphs: Multiscale Morphology and Graphcuts,” *IEEE J. Sel. Topics Signal Process.*, vol. 6, no. 7, pp. 780–794, Nov. 2012.

[3] C. Sakaridis, K. Drakopoulos, and P. Maragos, “Theoretical Analysis of Active Contours on Graphs,” *SIAM Journal on Imaging Sciences*, vol. 10, no. 3, pp. 1475–1510, 2017.

[4] N. Kolotouros and P. Maragos, “A Finite Element Computational Framework for Active Contours on Graphs,” *ArXiv e-prints*, 2017.

[5] V. Caselles, R. Kimmel, and G. Sapiro, “Geodesic Active Contours,” *Int. J. Comput. Vis.*, vol. 22, no. 1, pp. 61–79, 1997.

[6] T. F. Chan and L. A. Vese, “Active Contours Without Edges,” *IEEE Trans. Image Process.*, vol. 10, no. 2, pp. 266–277, Feb. 2001.

[7] M. Penrose, *Random Geometric Graphs*. Oxford University Press, 2003.

[8] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active Contour Models," *Int. J. Comput. Vis.*, vol. 1, no. 4, pp. 321–331, 1988.

[9] V. Caselles, F. Catté, T. Coll, and F. Dibos, "A geometric model for active contours in image processing," *Numerische Mathematik*, vol. 66, pp. 1–31, 1993.

[10] D. Mumford and J. Shah, "Optimal approximations by piecewise smooth functions and associated variational problems," *Communications on Pure and Applied Mathematics*, vol. 42, no. 5, pp. 577–685, 1989.

[11] T. F. Chan, S. Esedoglu, and M. Nikolova, "Algorithms for Finding Global Minimizers of Image Segmentation and Denoising Models," *SIAM Journal on Applied Mathematics*, vol. 66, no. 5, pp. 1632–1648, 2006.

[12] L. Vincent, "Graphs and Mathematical Morphology," *Signal Processing*, vol. 16, pp. 365–388, 1989.

[13] J. Jaromczyk and G. Toussaint, "Relative Neighborhood Graphs and Their Relatives," *Proceedings of the IEEE*, vol. 80, no. 9, pp. 1502–1517, Sep. 1992.

[14] H. J. A. M. Heijmans, P. Nacken, A. Toet, and L. Vincent, "Graph Morphology," *Journal of Visual Communication and Image Representation*, vol. 3, no. 1, pp. 24–38, Mar. 1992.

[15] J. Cousty, L. Najman, and J. Serra, "Some Morphological Operators in Graph Spaces," in *Proc. International Symposium on Mathematical Morphology*, 2009.

[16] G. Gilboa and S. Osher, "Nonlocal Linear Image Regularization and Supervised Segmentation," *Multiscale Model. Simul.*, vol. 6, no. 2, pp. 595–630, 2007.

[17] ——, "Nonlocal Operators with Applications to Image Processing," *Multiscale Model. Simul.*, vol. 7, no. 3, pp. 1005–1028, 2008.

[18] A. L. Bertozzi and A. Flenner, "Diffuse Interface Models on Graphs for Classification of High Dimensional Data," *Multiscale Model. Simul.*, vol. 10, no. 3, pp. 1090–1118, 2012.

[19] E. Merkurjev, T. Kostic, and A. L. Bertozzi, "An MBO Scheme on Graphs for Classification and Image Processing," *SIAM Journal on Imaging Sciences*, vol. 6, no. 4, pp. 1903–1930, 2013.

[20] Y. van Gennip, N. Guillen, B. Osting, and A. L. Bertozzi, "Mean Curvature, Threshold Dynamics, and Phase Field Theory on Finite Graphs," *Milan Journal of Mathematics*, vol. 82, no. 1, pp. 3–65, 2014.

[21] E. Merkurjev, E. Bae, A. L. Bertozzi, and X.-C. Tai, "Global Binary Optimization on Graphs for Classification of High-Dimensional Data," *J. Math. Imaging Vision*, vol. 52, no. 3, pp. 414–435, 2015.

[22] V.-T. Ta, A. Elmoataz, and O. Lézoray, "Nonlocal PDEs-Based Morphology on Weighted Graphs for Image and Data Processing," *IEEE Trans. Image Process.*, vol. 20, no. 6, pp. 1504–1516, Jun. 2011.

[23] O. Lézoray, A. Elmoataz, and V.-T. Ta, "Nonlocal PdEs on graphs for active contours models with applications to image segmentation and data clustering," in *ICASSP*, 2012, pp. 873–876.

[24] F. Lozes, A. Elmoataz, and O. Lézoray, "Partial difference operators on weighted graphs for image processing on surfaces and point clouds," *IEEE Trans. Image Process.*, vol. 23, no. 9, pp. 3896–3909, Sep. 2014.

[25] X. Desquesnes, A. Elmoataz, and O. Lézoray, "Eikonal Equation Adaptation on Weighted Graphs: Fast Geometric Diffusion Process for Local and Non-local Image and Data Processing," *J. Math. Imaging Vision*, vol. 46, no. 2, pp. 238–257, 2013.

[26] A. Elmoataz, M. Toutain, and D. Tenbrinck, "On the p-Laplacian and ∞-Laplacian on Graphs with Applications in Image and Data Processing," *SIAM Journal on Imaging Sciences*, vol. 8, no. 4, pp. 2412–2451, 2015.

[27] A. Elmoataz, F. Lozes, and M. Toutain, "Nonlocal PDEs on Graphs: From Tug-of-War Games to Unified Interpolation on Images and Point Clouds," *J. Math. Imaging Vision*, vol. 57, no. 3, pp. 381–401, 2017.

[28] S.-Y. Chung and C. A. Berenstein, "ω-Harmonic Functions and Inverse Conductivity Problems on Networks," *SIAM J. Appl. Math.*, vol. 65, no. 4, pp. 1200–1226, Apr. 2005.

[29] A. Bensoussan and J.-L. Menaldi, "Difference Equations on Weighted Graphs," *J. Convex Analysis*, vol. 12, no. 1, pp. 13–44, 2005.

[30] L. Grady and J. Polimeni, *Discrete Calculus: Applied Analysis on Graphs for Computational Science*. Springer Science & Business Media, 2010.

[31] A. N. Hirani, "Discrete Exterior Calculus," Ph.D. dissertation, Cal. Tech, 2003.

[32] C. Couprie, L. Grady, L. Najman, J.-C. Pesquet, and H. Talbot, "Dual Constrained TV-based Regularization on Graphs," *SIAM Journal on Imaging Sciences*, vol. 6, no. 3, pp. 1246–1273, 2013.

[33] C. Couprie, L. Grady, H. Talbot, and L. Najman, "Combinatorial Continuous Maximum Flow," *SIAM Journal on Imaging Sciences*, vol. 4, no. 3, pp. 905–930, 2011.

[34] O. Lézoray and L. Grady, *Image Processing and Analysis with Graphs: Theory and Practice*, ser. Digital Imaging and Computer Vision. CRC Press, 2012.

[35] L. Grady and C. Alvino, "The Piecewise Smooth Mumford-Shah Functional on an Arbitrary Graph," *IEEE Trans. Image Process.*, vol. 18, no. 11, pp. 2547–2561, Nov. 2009.

[36] S. Bougleux, A. Elmoataz, and M. Melkemi, "Local and Nonlocal Discrete Regularization on Weighted Graphs for Image and Mesh Processing," *Int. J. Comput. Vis.*, vol. 84, pp. 220–236, 2009.

[37] A. Elmoataz, O. Lézoray, and S. Bougleux, "Nonlocal Discrete Regularization on Weighted Graphs: A Framework for Image and Manifold Processing," *IEEE Trans. Image Process.*, vol. 17, no. 7, pp. 1047–1060, Jul. 2008.

[38] R. Courant, K. Friedrichs, and H. Lewy, "On the Partial Differential Equations of Mathematical Physics," *IBM Journal*, pp. 215–234, Mar. 1967.

[39] J. Shi and J. Malik, "Normalized Cuts and Image Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, Aug. 2000.

[40] Y. Boykov, O. Veksler, and R. Zabih, "Fast Approximate Energy Minimization via Graph Cuts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 11, pp. 1222–1239, Nov. 2001.

[41] Y. Boykov and V. Kolmogorov, "Computing Geodesics and Minimal Surfaces via Graph Cuts," in *Proc. Int. Conf. Comput. Vis.*, 2003.

[42] L. Grady, "Random Walks for Image Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 11, pp. 1768–1783, Nov. 2006.

[43] J. Cousty, G. Bertrand, L. Najman, and M. Couprie, "Watershed Cuts: Minimum Spanning Forests and the Drop of Water Principle," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 8, pp. 1362–1374, Aug. 2009.

[44] ——, "Watershed Cuts: Thinnings, Shortest Path Forests, and Topological Watersheds," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 5, pp. 925–939, May 2010.

[45] C. Couprie, L. Grady, L. Najman, and H. Talbot, "Power Watershed: A Unifying Graph-Based Optimization Framework," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 7, pp. 1384–1399, Jul. 2011.

[46] D. Pollard, "Strong Consistency of k-Means Clustering," *Ann. Statist.*, vol. 9, no. 1, pp. 135–140, 1981.

[47] U. von Luxburg, M. Belkin, and O. Bousquet, "Consistency of Spectral Clustering," *Ann. Statist.*, vol. 36, no. 2, pp. 555–586, 2008.

[48] M. Belkin and P. Niyogi, "Convergence of Laplacian Eigenmaps," in *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[49] D. Ting, L. Huang, and M. I. Jordan, "An Analysis of the Convergence of Graph Laplacians," in *Proc. International Conference on Machine Learning*, 2010.

[50] E. Arias-Castro, B. Pelletier, and P. Pudlo, "The Normalized Graph Cut and Cheeger Constant: From Discrete to Continuous," *Adv. in Appl. Probab.*, vol. 44, no. 4, pp. 907—937, 2012.

[51] M. Maier, U. von Luxburg, and M. Hein, "How the result of graph clustering methods depends on the construction of the graph," *ESAIM: Probability and Statistics*, vol. 17, pp. 370–418, 2013.

[52] N. García Trillos and D. Slepčev, "Continuum Limit of Total Variation on Point Clouds," *Arch. Ration. Mech. Anal.*, vol. 220, no. 1, pp. 193–241, 2016.

[53] Y. van Gennip and A. L. Bertozzi, "Γ-convergence of Graph Ginzburg-Landau Functionals," *Advances in Differential Equations*, vol. 17, no. 11/12, pp. 1115–1180, 2012.

[54] D. D. Lee and H. S. Seung, "Algorithms for Non-negative Matrix Factorization," in *Advances in Neural Information Processing Systems (NIPS)*, 2001.

[55] Z. Yuan and E. Oja, "Projective Nonnegative Matrix Factorization for Image Compression and Feature Extraction," in *Proc. Scandinavian Conf. Image Anal.*, 2005.

[56] Z. Yang, T. Hao, O. Dikmen, X. Chen, and E. Oja, "Clustering by Nonnegative Matrix Factorization Using Graph Random Walk," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[57] T. Snijders and K. Nowicki, "Estimation and Prediction for Stochastic Blockmodels for Graphs with Latent Block Structure," *Journal of Classification*, vol. 14, no. 1, pp. 75–100, 1997.

[58] H. M. Mourad, J. Dolbow, and K. Garikipati, "An assumed-gradient finite element method for the level set equation," *International Journal for Numerical Methods in Engineering*, vol. 64, no. 8, pp. 1009–1032, 2005.

[59] S. Valance, R. De Borst, J. Réthoré, and M. Coret, "A partition-of-unity-based finite element method for level sets," *International journal for numerical methods in engineering*, vol. 76, no. 10, pp. 1513–1527, 2008.

[60] T. J. Barth and J. A. Sethian, "Numerical schemes for the hamilton–jacobi and level set equations on triangulated domains," *Journal of Computational Physics*, vol. 145, no. 1, pp. 1–40, 1998.

[61] P.-O. Persson and G. Strang, "A Simple Mesh Generator in MATLAB," *SIAM Review*, vol. 46, no. 2, pp. 329–345, Nov. 2004.

[62] C. Li, C. Xu, C. Gui, and M. D. Fox, "Level set evolution without re-initialization: a new variational formulation," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'05)*, 2005.

[63] A. George and J. Liu, *Computer Solution of Large Sparse Positive Definite Systems*. Prentice Hall, 1981.

[64] D. Adalsteinsson and J. A. Sethian, "A Fast Level Set Method for Propagating Interfaces," *J. Comput. Phys.*, vol. 118, no. 2, pp. 269–277, May 1995. [Online]. Available: http://dx.doi.org/10.1006/jcph.1995.1098

[65] R. T. Whitaker, "A Level-Set Approach to 3D Reconstruction from Range Data," *Int. J. Comput. Vision*, vol. 29, no. 3, pp. 203–231, Sep. 1998. [Online]. Available: https://doi.org/10.1023/A:1008036829907

[66] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, "A PDE-Based Fast Local Level Set Method," *Journal of Computational Physics*, vol. 155, no. 2, pp. 410 – 438, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999199963453

[67] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics," in *Proc. Int. Conf. Comput. Vis.*, vol. 2, Jul. 2001, pp. 416–423.

[68] J. M. Ribando, "Measuring Solid Angles Beyond Dimension Three," *Discrete & Computational Geometry*, vol. 36, no. 3, pp. 479–487, 2006.