Contour cut: identifying salient contours in images by solving a Hermitian eigenvalue problem

Ryan Kennedy, Jean Gallier and Jianbo Shi
University of Pennsylvania
{kenry,jean,jshi}@cis.upenn.edu

Abstract

The problem of finding one-dimensional structures in images and videos can be formulated as a problem of searching for cycles in graphs. In [11], an untangling-cycle cost function was proposed for identifying persistent cycles in a weighted graph, corresponding to salient contours in an image. We have analyzed their method and give two significant improvements. First, we generalize their cost function to a contour cut *criterion and give a computational solution by solving a family of Hermitian eigenvalue problems. Second, we use the idea of a graph circulation, which ensures that each node has a balanced in- and out-flow and permits a natural random-walk interpretation of our cost function. We show that our method finds far more accurate contours in images than [11]. Furthermore, we show that our method is robust to graph compression which allows us to accelerate the computation without loss of accuracy.*

1. Introduction

Many visual perception problems involve finding salient one-dimensional structures in datasets, such as finding contours in images [1], finding object trajectories in video [3], or identifying meaningful sequence of events [9].

In [11], Zhu developed an approach for finding salient one-dimensional structures in a graph by finding persistent cycles and demonstrated that this can be used to extract contours from images. They were able to identify salient contours and their algorithm is one of the top-performing algorithms on the Berkeley Segmentation dataset [7]. In their algorithm, Zhu constructs a directed graph by defining a graph node for each image edge and connecting all nodes within a small radius. Weights are given to the graph edges based on the relative angles of the image edges such that edges with similar angles are strongly connected. To distinguish between contours and clutter, they consider a random walk on the graph. During a random walk, either a contour will be followed, or else the walk will diverge away from a contour due to clutter and gaps in the edges. The main insight of Zhu is that of *persistency*: for a salient contour, a random walk will tend to return to the same node consistently after a predictable amount of time (roughly the length of contour). It was shown this *peakedness* in the return time of a random walk is related to the eigenvectors of the random walk matrix. Zhu used this persistency measure to identify one-dimensional contours in the graph.

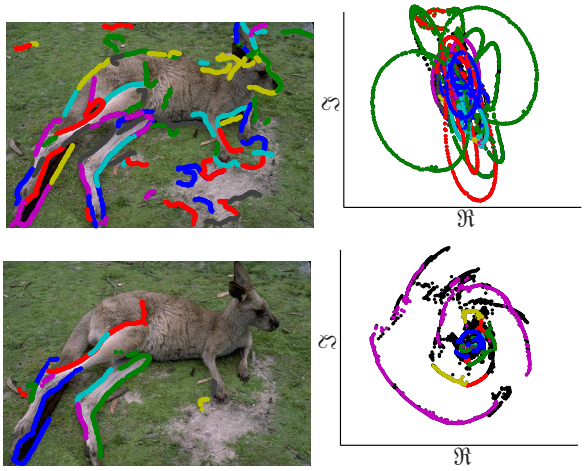
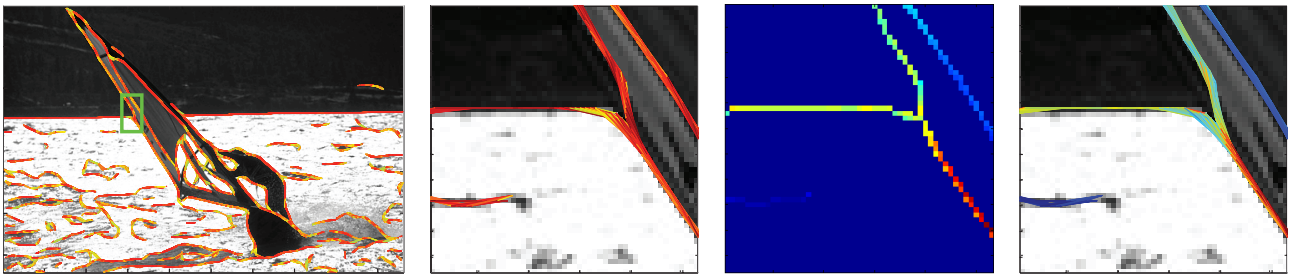


Figure 1: **Top:** image with all (possibly-overlapping) contours found by our algorithm using 30 eigenvectors. **right**, "most-representative" eigenvector with contours. **Bottom: left**, image with all (possibly-overlapping) contours found using Zhu's algorithm [11] using 30 eigenvectors. **right**, "most-representative" eigenvector with contours.

In this paper, we present an extension of the work of Zhu and provide two main contributions. First, we analyze the graph cycle cost function and introduce a flow on the graph which allows us to define various graph cuts. We show this leads to a simple modification of Zhu's method of computing the eigenvectors of the random walk matrix. This new algorithm significantly improves upon that of Zhu, while maintaining a low computation time. Second, we show that Zhu's solution only calculates the actual critical points of their cost function if the random walk matrix is a normal matrix. The exact optimal solution which we present re-



(a) Image with graph weights (W) superimposed.

(b) Zoomed in weights of W.

(c) Zoomed in stationary distribution of P.

(d) Zoomed in weights of F.

Figure 2: Difference between W and the circulation $F = \Pi P$. 2a: image with graph connections superimposed. 2b: a zoomed-in section. 2c: π, the stationary distribution of P. 2d: weights of F . Note that in 2b, the weights on all three contours of the junction are equally strong while in 2d, the flow constraint causes the weights to split at the junction. For all plots, the weight are averaged over both directions. Red indicates higher weights and blue indicates lower weights.

quires solving an eigenvalue problem of a parameterized matrix and we demonstrate that it significantly outperforms Zhu's algorithm on natural images.

2. Problem formulation

2.1. Graph construction

Let $G = (V, E)$ be a graph with vertex set V and edge set $E \subseteq V \times V$. Each edge is assigned a non-negative weight W_{ij}, with the matrix $W \in (\mathbb{R}^+ \bigcup \{0\})^{|V|\times|V|}$. A *contour* (C, O) is defined by a set of vertices $C \subseteq V$ and a function $O : C \to \{1, \ldots, |C|\}$ which specifies a unique ordering of these vertices. To construct a graph from an image, we follow the procedure of [11]. First, we find the edges in the image. Two graph vertices are created for each image edge (one for each direction) and vertices are connected if they are nearby in the image. Graph weights are set based on the relative angles of the image edges (Figure 3b).

Note that closed cycles in an image can be found directly by using only a single graph vertex per image edge and making sure that the orientations are locally consistent.

2.2. Graph circulations

In the normalized cut algorithm [10], the concepts of graph cuts and volume are defined for a symmetric graph weight matrix W. For asymmetric graphs, the cut from A to B is not in general equal to the cut from B to A, for $A, B \subseteq V$. We introduce a *graph circulation* matrix F. A circulation [2][5] is a matrix $F \in (\mathbb{R}^+ \bigcup \{0\})^{|V|\times|V|}$ which assigns a non-negative real value to each graph edge, satisfying

$$\sum_{i:(i,j)\in E} F_{ij} = \sum_{k:(j,k)\in E} F_{jk}, \tag{1}$$

so that each node has the same total incoming and outgoing weight. We use the circulation $F = \Pi P$, where $P = D^{-1}W$ with $D = \text{diag}(\sum_j W_{ij})$ is the row-normalized weight matrix and $\Pi = \text{diag}(\pi)$ is the diagonal matrix of the stationary distribution of P. This circulation is advantageous because if W is symmetric then the directed versions of cut and volume that we will define reduce to the original undirected versions [5]. It also admits a natural random-walk interpretation of our algorithm (Section 2.7). Figure 2 shows the difference between the weights of F and W.

2.3. Internal and external cuts

A contour is a one-dimensional structure in a graph. We use the *external cut* of a contour (C, O) to measure its separation from the rest of the graph, $V \setminus C$:

$$\text{Ecut}(C) = \sum_{i\in C, j\notin C} F_{ij}. \tag{2}$$

We use the *internal cut* to measure the entanglement caused by graph edges within the contour that violate the ordering O. Intuitively, the internal cut measures how much the contour deviates from an ideal one-dimensional contour toward a 2-dimensional clique. Let $k \in \mathbb{Z}^+$ be the *width* of the contour. Nodes $i, j \in C$ with $|O(i) - O(j)| > k$ are too distant and so are part of the internal cut:

$$\text{Icut}(C, O) = \sum_{(i,j)\in C, |O(i)-O(j)|>k} F_{ij}. \tag{3}$$

Figure 3c shows these two cuts on a weight matrix.

2.4. Contour cut cost

Given a contour (C, O), we define a cost function which takes into account both the internal and external cuts:

$$\boxed{\text{Ccut}(C, O) = \frac{\text{Icut}(C, O) + \text{Ecut}(C)}{\text{Vol}(C)},} \tag{4}$$

where $\text{Vol}(x) = \sum_{i\in C, j\in E} F_{ij}$ is the sum of the weights of all edges incident with the contour. This cost function will be small for contours having small internal and external cuts.

2.5. Circular embedding

To represent a contour (C, O), we must encode both *which* nodes are part of the contour as well as the *ordering* of these points. We do so using a *circular embedding* where each node of the contour is mapped to a point on a circle about the origin in the complex plane and all other points are mapped to the origin (Figure 3f). Each point is represented as as complex number

$$x_j = r_j \exp(i\theta_j) \tag{5}$$

with $r_j = 1$ if $j \in C$ and 0 otherwise, and $\theta_j = O(j)\delta$ with $\delta = \frac{2\pi}{|C|}$. The radius r_j of each point encodes whether it is part of the contour and the angle θ_j encodes the ordering.

2.6. Contour cut

The internal and external cuts can be encoded with respect to the circular embedding. Given a the circular embedding of a contour, $x \in \mathbb{C}^{|C|}$, the external cut is:

$$\text{Ecut}(x) = \sum_{\substack{(i,j)\in E:\\ r_i=0\\ r_j\neq 0}} F_{ij} = \sum_{(i,j)\in E} F_{ij} r_i (1 - r_j). \tag{6}$$

Rather than using the hard-bound of k for the internal cut, we use a soft version of the internal cut by using the cosine function, as in [11]:

$$\text{Icut}(x) = \sum_{(i,j)\in C} F_{ij} r_i r_j [1 - \cos(\theta_j - \theta_i - \delta)]. \tag{7}$$

This cosine function has two desirable properties. First, the cosine will reach a maximum of 1 when two nodes are exactly δ apart (they are adjacent in the ordering) in which case the cut is zero, and it decreases as the nodes are farther away. Second, since longer contours are packed more tightly in the circle, δ will be smaller and this cost function will tend to regard nodes further away as "closer" as the contour becomes longer.

The volume of the contour is defined as

$$\text{Vol}(x) = \sum_{(i,j)\in E} F_{ij} r_i. \tag{8}$$

The contour cut in terms of the circular embedding x is then

$$\begin{aligned}
\text{Ccut}(x) &= \frac{\text{Icut}(x) + \text{Ecut}(x)}{\text{Vol}(x)} \\
&= 1 - \frac{\sum_{(i,j)\in E} F_{ij} r_i r_j \cos(\theta_j - \theta_i - \delta)}{\sum_{(i,j)\in E} F_{ij} r_i} \\
&= 1 - \frac{\Re\left\{x^* \left[F \exp(-i\delta)\right] x\right\}}{x^* \Pi x} \\
&= \frac{x^* \left[\Pi - \left[F \exp(-i\delta) + F^T \exp(i\delta)\right]/2\right] x}{x^* \Pi x} \\
&= \frac{x^* [\Pi - H(\delta)] x}{x^* \Pi x},
\end{aligned}$$

where

$$H(\delta) = \frac{F \exp(-i\delta) + F^T \exp(i\delta)}{2}. \tag{9}$$

Therefore, we can write the contour cut in terms of a generalized Rayleigh quotient:

$$\boxed{\text{Ccut}(x) = R_{\Pi - H(\delta), \Pi}(x) = \frac{x^* [\Pi - H(\delta)] x}{x^* \Pi x}.} \tag{10}$$

It follows that the problem of minimizing $\text{Ccut}(x) = R_{\Pi - H(\delta),\Pi}(x)$ is equivalent to maximizing $R_{H(\delta),\Pi}(x)$.

2.7. Interpretation

Following a similar derivation as in [2], the Rayleigh quotient $R_{\Pi - H(\delta),\Pi}(x)$ can be rewritten as

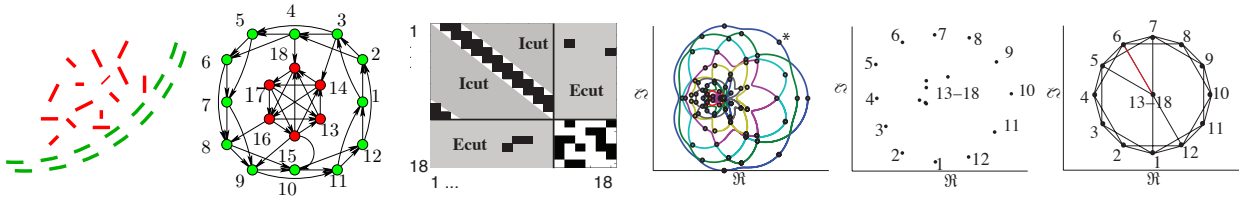
$$\begin{aligned}
&R_{\Pi - H(\delta),\Pi}(x) \\
&= \frac{1}{2} \frac{\sum_{i,j} P_{ij} \pi_i |x_i - x_j e^{i\delta}|^2}{\sum_i \pi_i |x_i|^2}. \\
&= \frac{1}{2} \frac{\sum_{i,j} P_{ij} \pi_i (r_i^2 + r_j^2 - 2 r_i r_j \cos(\theta_i - \theta_j - \delta))}{\sum_i \pi_i r_i^2}.
\end{aligned}$$

This Rayleigh quotient is small when nodes with large edge weights between them have a similar radius and an angle difference close to δ. Thus, by minimizing this Rayleigh quotient we are finding a subset of nodes and an ordering for which all nodes with high weights on the contour have a similar radius and an angle close to δ. Observe that for a clique of nodes there will necessarily be many strong links which span large angles (the internal cut), which will increase the value of the Rayleigh quotient; only for contours will the Rayleigh quotient be small.

The contour cut is also closely related to the normalized cut, which has has a natural random-walk interpretation. In [8], it was shown that minimizing the normalized cut minimizes the probability of jumping between the two partitions of nodes during a random walk. Like normalized cut, the contour cut also has a natural random-walk interpretation. Consider a discretized form of the internal cut (Equation 3) and let $B \subseteq E$ be the set of cut links (including both the internal and external cut) on the contour C which is represented by the circular embedding x. Then,

$$\begin{aligned}
\text{Ccut}(x) = \frac{x^*(\Pi - H(\delta))x}{x^* \Pi x} &= \frac{\sum_{(i,j)\in B} \pi_i P_{ij}}{\sum_{i\in C} \pi_i} \\
&= P((X_0, X_1) \in B | X_0 \in C),
\end{aligned}$$

where X_i is the present vertex at time i during a random walk. Therefore, the contour which minimizes the contour cut is the contour which also minimizes the probability of taking a bad link either too far forward on the contour (internal cut) or off of the contour (external cut) during a random walk. In other words, it maximizes the probability of taking a small step down the contour.



(a) (b) (c) (d) (e) (f)

Figure 3: Our algorithm run on a synthetic example. The synthetic image (3a) is used to construct a graph (3b) represented by a weight matrix (3c). The eigenvalues of $\Pi^{-1}H(\delta)$ over all δ are plotted in 3d with local maxima denoted by black circles. The best embedding, denoted by an asterisk, is shown in 3e with the ideal circular embedding with graph connections shown in 3f.

3. Computational solution

To minimize the contour cut, we want to solve

$$\max_x \frac{x^* H(\delta) x}{x^* \Pi x} \tag{11}$$
$$\text{s.t.} \quad x_i = r_i \exp(i\theta_i), \quad r_i \in \{0, 1\}, \quad \theta_i = O(i)\delta,$$

where we seek not just the global maximum but all critical points which correspond to different contours. By requiring r_i and θ_i to take on discrete values the problem is computationally infeasible since it requires searching over not only an exponential number of subsets sets of vertices but also over orderings on each of these sets. We relax the problem by allowing x to take on arbitrary complex values: $x \in \mathbb{C}^{|C|}$. Note that we now must maximize over δ in addition to x. We thus seek the critical points of Equation 12 with respect to both x and δ. Our main result for solving Equation 12 is the following theorem:

Theorem 3.1. *The critical points of the relaxed contour cut problem*

$$\max_{x,\delta} \frac{x^* H(\delta) x}{x^* \Pi x} \quad \textit{s.t.} \quad x_i \in \mathbb{C} \tag{12}$$

can be found by searching over δ and finding the eigenvectors of the corresponding matrices $\Pi^{-1}H(\delta)$; any eigenvectors for which $x^*H(\delta)x = |x^*Fx|$ are critical points with respect to both x and δ.

Proof. This immediately follows from Lemmas 3.2 and 3.3, with proofs in the Supplementary Material. □

For a fixed δ, we have:

Lemma 3.2. *For fixed δ, the critical values of $\frac{x^* H(\delta) x}{x^* \Pi x}$ are equal to the eigenvalues of the Hermitian matrix $\Pi^{-1/2}H(\delta)\Pi^{-1/2}$ and are achieved for all vectors of the form $x = \Pi^{-1/2}y$, where y is the associated unit eigenvector of $\Pi^{-1/2}H(\delta)\Pi^{-1/2}$, with the maximum being the top eigenvalue λ_1. Furthermore, $\Pi^{-1/2}H(\delta)\Pi^{-1/2}$ and $\Pi^{-1}H(\delta)$ have the same eigenvalues and every eigenvector, x, of $\Pi^{-1}H(\delta)$ is of the form $x = \Pi^{-1/2}y$, where y is an eigenvector of $\Pi^{-1/2}H(\delta)\Pi^{-1/2}$.*

For a fixed x we have:

Lemma 3.3. *For a fixed x, the unique local maximum (and thus the global maximum) of $\frac{x^* H(\delta) x}{x^* \Pi x}$ is attained for $\delta = \arg(x^* F x)$ for the function value $\frac{|x^* F x|}{x^* \Pi x}$. Furthermore, we have*

$$\frac{x^* H(\delta) x}{x^* \Pi x} \leq \frac{|x^* F x|}{x^* \Pi x} \quad \forall\, \delta. \tag{13}$$

3.1. Normal graphs

For a normal matrix P (or equivalently, F), we have the following result:

Claim 3.4. *If $P = D^{-1}W$ is a normal matrix, the critical points of Equation 12 are exactly the eigenvectors of P.*

Proof. See supplementary material[1]. □

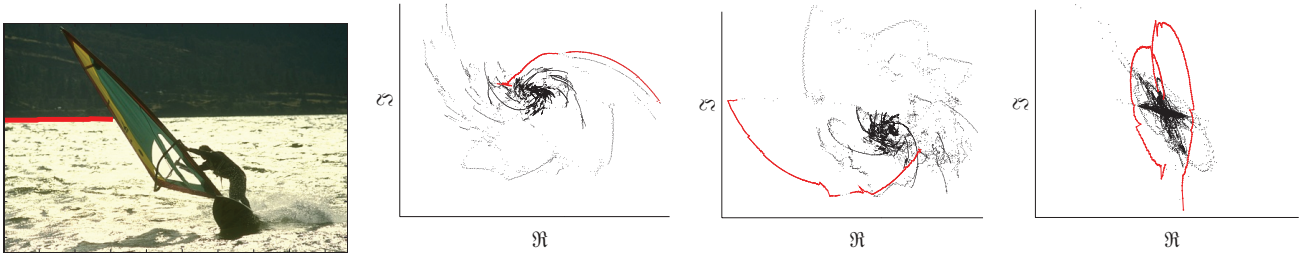
This means that, for normal P, we can find the critical points of Equation 12 by directly finding the eigenvalues of P without searching over all δ.

This is, in fact, the algorithm given by Zhu [11], where it is claimed that the critical points of their cost function are obtained at the eignenvectors of P. As their cost function differs from ours only in that we use the normalized matrix $F = \Pi P$ rather than P directly and divide by $x^*\Pi x$, it holds that the eigenvectors of P are only the critical points of their cost function when P is normal, contrary to the claim of Zhu. This result also motivates an approximation of our algorithm, given in Section 3.6

3.2. General graphs

In practice, P is rarely a normal matrix and we cannot simply calculate the eigenvectors of P. For general graphs, we can use Theorem 3.1 to create an algorithm which finds the actual critical points of Equation 12. However, in practice it might be difficult to search over all values of δ with a small enough step size such that $x^*H(\delta)x = |x^*Fx|$ to a high enough precision. Instead we propose a different

[1] Available at http://www.seas.upenn.edu/∼ kenry/ccut.html



(a) Image with a contour traced (red).

(b) Contour in embedding space for x a solution to $Px = \lambda x$.

(c) Contour in embedding space for $x = \Pi^{-1}y$, y a solution to $P^T y = \lambda y$.

(d) Contour in embedding space for x a solution to $\Pi^{-1}H(\delta)x = \lambda x$.

Figure 4: Comparison between approximate solutions (4b, 4c) and an exact solution (4d) to the eigenvalue problem $\Pi^{-1}H(\delta)x = \lambda x$. By using $H(\delta)$, we take both P and P^T into account and are able to achieve a more circular embedding. Using just P or P^T by itself can lead to an ordering which spirals out rather than creating a circle.

algorithm which is motivated by the observation that as δ changes, the eigenvalues of $H(\delta)$ rarely cross. This has been observed previously [6] and it has been true in our experience as well. To determine the critical points with respect to δ, we make the assumption that the k^{th} largest eigenvalue of $H(\delta)$ remains the k^{th} largest over all values of δ, in which case it suffices to directly find the local maxima of the k^{th} eigenvalue over all δ. This gives rise to the following algorithm:

Algorithm 1 Contour cut

Calculate matrices P and Π from W.
for $\delta = \delta_{\text{min}}$ to δ_{max} **do**
  $H(\delta) \leftarrow (\Pi P \exp(-i\delta) + P^T \Pi \exp(i\delta))/2$
  Solve $\Pi^{-1}H(\delta)x = \lambda x$ for top k eigenvectors.
end for
Find local maxima of each λ_i over all δ. The associated x's are critical points with respect to both x and δ.

In practice, we found it sufficient to search over a small range for δ, such as $\delta \in [0, \pi/4]$ with a step size around 0.025.

3.3. Examples

Figure 3 shows the result of running our algorithm on a synthetic image. All eigenvalues for all δ were computed and plotted in Figure 3d. The top solution to the relaxed optimization problem is shown in Figure 3e with the ideal circular embedding in Figure 3f.

Results on a real image are shown in Figure 4. The image and a contour are shown in Figure 4a, with the corresponding contour in the embedding space in Figure 4b-4d. Another comparison between the embeddings for each algorithm is shown in Figure 8

3.4. Approximations

For full images the resulting graph can have on the order of 10,000 nodes, making calculation of our algorithm slow. To improve speed, there are two approximations: approximating the graph and an approximating the algorithm.

3.5. Approximating the full graph

In the middle of a strong contour where there is no junction or clutter, there is very little ambiguity about which nodes should be grouped together into a potential contour. We found that constructing the full weight matrix and then using normalized cut [10] to cluster nodes into $n = 500$ or $n = 1000$ small contour segments worked well in practice. Given these contour segments, a new weight matrix is created by treating each fragment as a supernode in a new graph with the weights between fragments set to be the sum of the weights between nodes in each fragment in the original graph. Our algorithm can be run on the smaller graph.

3.6. Approximating the algorithm

The matrix $\Pi^{-1}H(\delta)$ can be written as

$$\Pi^{-1}H = \frac{P\exp(-i\delta) + \Pi^{-1}P^T\Pi\exp(i\delta)}{2}. \qquad (14)$$

The solutions of $\Pi^{-1}H(\delta)x = \lambda x$ are thus a combination of the eigenvectors of P and of $\Pi^{-1}P^T\Pi$, modulated by δ. As an approximation, we can solve just the eigenvalue problem $Px = \lambda x$. Alternatively, another approximation is had by solving the eigenvalue problem $P^T y = \lambda y$ and performing the variable transformation $x = \Pi^{-1}y$. These approximations can also be interpreted as assuming that P is normal, in which case either one provides the exact solution (Section 3.1).

We will demonstrate experimentally that (1) approximating the graph by clustering high-confidence nodes does not

reduce the accuracy significantly and that (2) our exact algorithm gives significantly better results than using the eigenvectors of P or P^T.

3.7. Discretization

Given a solution to our relaxed problem, we must find a discrete set of vertices and a discrete ordering on those vertices. To do so, we follow the procedure of [11] and discretize the relaxed solutions by finding a cycle around the origin of maximum area. However, we make one change to this algorithm. In [11], Zhu finds the maximum-area cycle by using the shortest path algorithm. The best cycle is actually the *longest* path, but since the longest path problem is NP-complete, Zhu transforms the longest path problem into a shortest path problem by subtracting the graph weights from a large constant, giving an approximate longest path. We make the observation that the graph induced by the relaxed solution can be constrained such that the links go around the origin in only one direction, in which case the graph can be split at some angle and then forms a directed, acyclic graph (DAG). The longest path through a DAG can be found in polynomial time by negating the weights and using the Bellman-Ford shortest-path algorithm.

4. Experiments

4.1. Algorithms

We compare the algorithm which finds the exact critical points of Equation 12 by searching over all δ to the algorithms that give approximations by finding the right eigenvectors of P and the left eigenvectors of P scaled by Π^{-1}. The algorithms we compare are '$H(F)$', '$H(P)$', 'P', and 'P^T', where the name designates which matrix is used in the eigenvector computation. Specifically, '$H(F)$' is the algorithm which finds the exact critical points of Equation 12 using the circulation matrix $F = \Pi P$; '$H(P)$' is the exact algorithm using P rather than F; 'P' is the approximation given by the right eigenvectors of P (Section 3.5); 'P^T' is the approximation given by the left eigenvectors of P and scaled by Π^{-1} (Section 3.6). Note that 'P' is the algorithm given by Zhu [11].

We also compare between using the full graph and clustering the nodes to $n = 500$ and $n = 1000$ clusters and running the algorithms on this reduced graph (Section 3.5). Finally, we compare to the probability of boundary (Pb) algorithm [7] as a baseline. Since Pb is the input to our algorithm, this comparison gives an indication of whether we are identifying the contours of Pb that are most salient.

We show that reducing the graph size to $n = 1000$ or 500 has little effect on the quality of the results but significantly reducing the running time, while our exact algorithm performs significantly better than the approximate algorithms.

4.2. Berkeley Segmentation Dataset

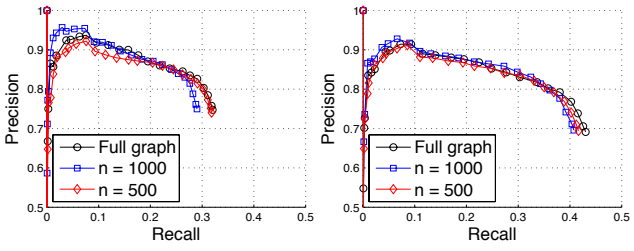
4.2.1 Evaluation

We use two different measures to evaluate the algorithms on the Berkeley Segmentation Dataset [7]. First, we compare the precision-recall (PR) curves from the Berkeley Segmentation dataset. Because different algorithms may yield different numbers of contours, we generate PR curves using the top 10 and top 20 contours from each algorithm that were found from the top 30 eigenvectors, as ranked by our cost function $\frac{x^* H(\delta) x}{x^* \Pi x}$. We also prune contours that overlap a better one by more than 25%. For each contour, the average of the Pb along the contour is used as input to the Berkeley benchmark, as was done in [4].

As noted in [4], the Berkeley benchmark is not well-suited for contour algorithms because it treats each pixel independently, and so we also compare the algorithms using the average value of the cost function $\frac{x^* H(\delta) x}{x^* \Pi x}$ with respect to the full graph, for each algorithm and graph size. Under this measure, the best possible value is obtained by the true, discrete local maxima of Equation 11. This is therefore a measurement of how close each algorithm gets to finding the true local optima, relative to each other. This measure is also independent of our application to images since it measures with respect to the cost function without translating back to the image domain.

4.2.2 Results

The effect of using a reduced-size graph is shown in Figure 5. When using both the top 10 and top 20 contours, there is very little if any decrease in the quality of the results by using a reduced-size graph. This is also seen in average cost function value for the contours (Table 1). Thus, we can use a smaller graph to speed up the algorithm without reducing the quality of the results.



(a) Top 10 contours (b) Top 20 contours

Figure 5: Effect of using a reduced-size matrix on the results of the exact algorithm $H(F)$ for the Berkeley Segmentation dataset.

A comparison between algorithms is shown in Figure 7. Using a graph of size $n = 500$, the precision values are

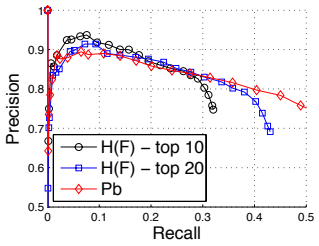
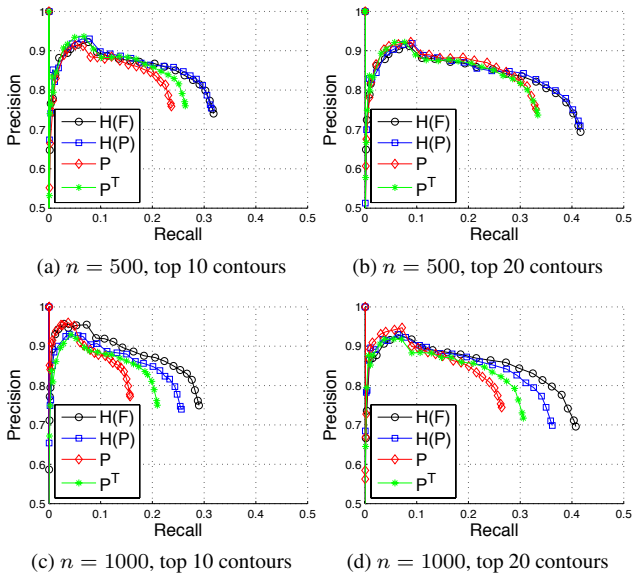


Figure 6: Comparison of our algorithm to Pb [7] algorithm on the Berkeley Segmentation dataset. Our algorithm outperforms Pb in the low-recall range



(a) $n = 500$, top 10 contours (b) $n = 500$, top 20 contours

(c) $n = 1000$, top 10 contours (d) $n = 1000$, top 20 contours

Figure 7: Comparison of algorithms for the Berkeley Segmentation dataset.

very similar for low recall, but for higher recall the algorithms '$H(F)$' and '$H(P)$' have better precision. Because all algorithms are producing the same number of contours, this indicates that '$H(F)$' and '$H(P)$' are producing longer contours, which is a better result. This can also be seen in the cost function values in Table 1, where '$H(F)$' and '$H(P)$' outperform 'P' and 'P^T'.

A similar result holds for the graph of size $n = 1000$, except that in this case the algorithm '$H(F)$' significantly outperforms '$H(P)$', which can also be seen in Table 1.

Results on some of the Berkeley Segmentation dataset are given in Figure 9. Figures 9c and 9d show that '$H(F)$' finds significantly longer contours than either 'P' or 'P^T'.

Figure 6 shows a comparison between our algorithm and Pb [7]. Since our algorithm begins with Pb as an input, it will converge to Pb as more contours are used. Our algorithm outperforms Pb in the low-recall range and so is able to pick out the best salient contours in an image.

Graph	Algorithm	Top 10 contours	Top 20 contours
$n =$ 500	$H(F)$	**0.8312**	**0.7008**
	P	0.6047	0.5003
	P^T	0.7033	0.5098
	$H(P)$	0.8246	0.6896
$n =$ 1000	$H(F)$	**0.8292**	**0.7403**
	P	0.4569	0.4277
	P^T	0.6470	0.5460
	$H(P)$	0.7887	0.6743
Full graph	$H(F)$	**0.8411**	**0.7452**
	P	0.6833	0.4970
	P^T	0.7391	0.6037

Table 1: Average value of $\frac{x^* H(\delta) x}{x^* \Pi x}$ with respect to the full graph, for each variation of our algorithm.

References

[1] T. Alter and R. Basri. Extracting salient curves from images: An analysis of the saliency network. *International Journal of Computer Vision (ICCV)*, 27(1):51–69, 1998. 2065

[2] F. Chung. Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9(1):1–19, 2005. 2066, 2067

[3] I. Cox, J. Rehg, and S. Hingorani. A Bayesian multiple-hypothesis approach to edge grouping and contour segmentation. *International Journal of Computer Vision (ICCV)*, 11(1):5–24, 1993. 2065

[4] P. Felzenszwalb and D. McAllester. A min-cover approach for finding salient curves. In *IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, page 185. IEEE, 2006. 2070

[5] D. Gleich. Hierarchical Directed Spectral Graph Partitioning. 2066

[6] P. Lax. *Linear algebra and its applications*. Wiley-Interscience, 2007. 2069

[7] 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 *International Conference on Computer Vision*, volume 2, pages 416–423, July 2001. 2065, 2070, 2071

[8] M. Meila and J. Shi. A random walks view of spectral segmentation. 2001. 2067

[9] K. Prabhakar, S. Oh, P. Wang, G. Abowd, and J. Rehg. Temporal causality for the analysis of visual events. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1967–1974. IEEE, 2010. 2065

[10] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2002. 2066, 2069

[11] Q. Zhu, G. Song, and J. Shi. Untangling cycles for contour grouping. In *International Conference on Computer Vision (ICCV)*, pages 1–8. IEEE, 2007. 2065, 2066, 2067, 2068, 2070

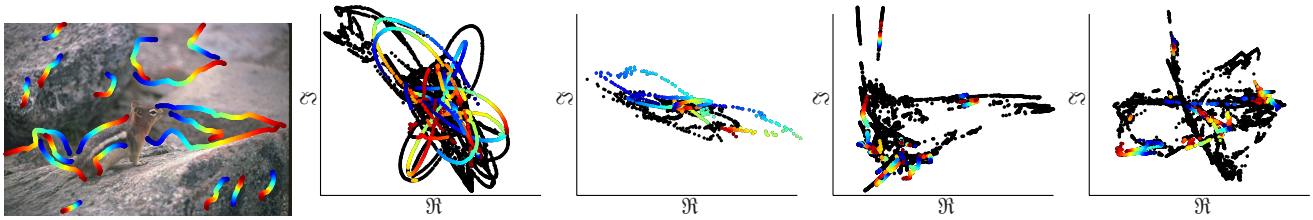
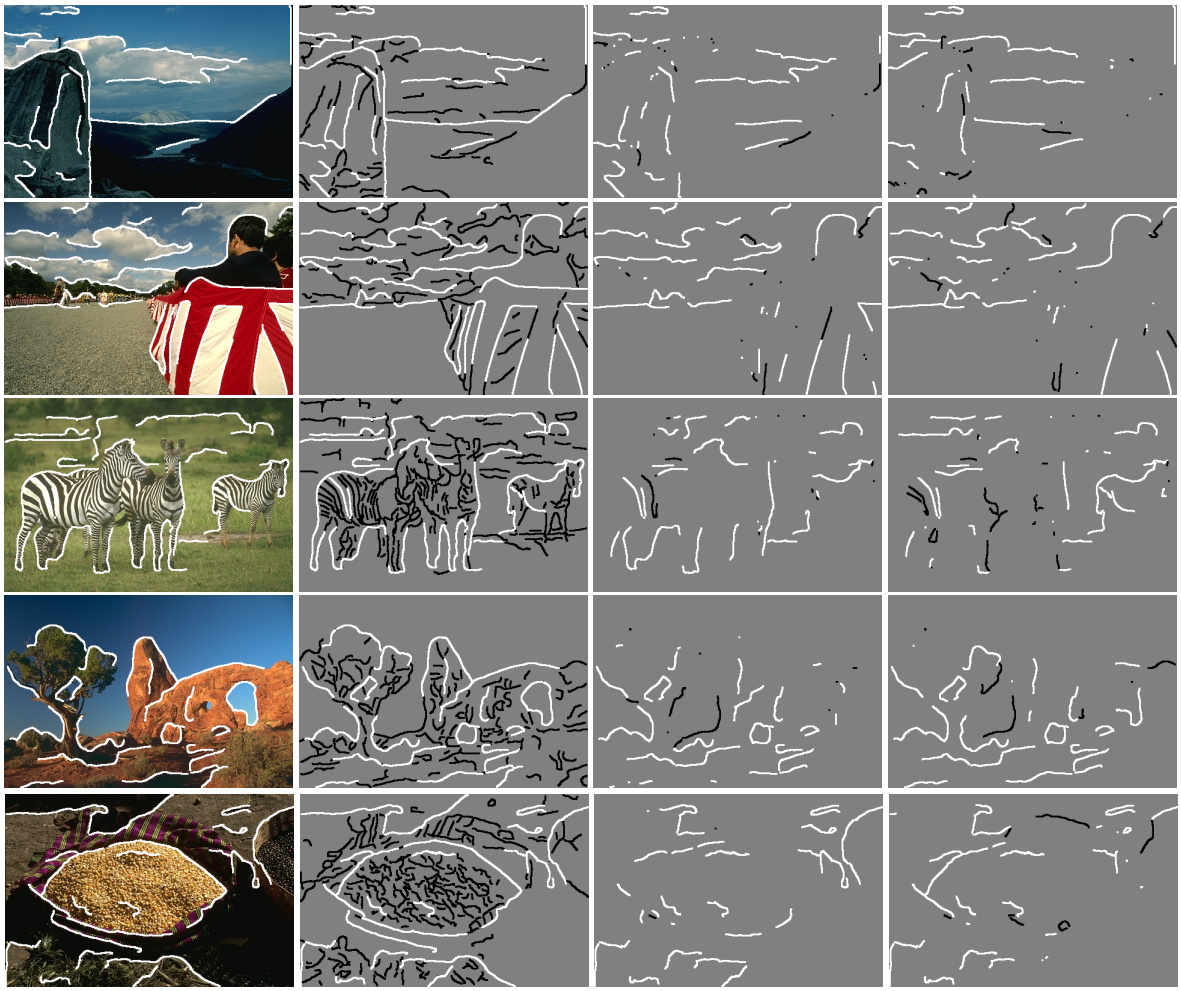


Figure 8: The top 20 contours were found for an image and the contours were traced on the eigenvector for each algorithm that was "most representative", calculated as the one that maximized mean contour cut score of the un-discretized contours as plotted in each embedding. **Left to right:** Image with contours overlaid (color indicates the ordering), most representative eigenvector for '$H(F)$', '$H(P)$', 'P', and 'P^T'. Best viewed in color.



(a) Image with extracted contours (white).

(b) All thresholded Pb edges (black) and contours (white).

(c) Difference between '$H(F)$' and 'P'. Contours only '$H(F)$' found are white and contours only 'P' found are black.

(d) Difference between '$H(F)$' and 'P^T'. Contours only '$H(F)$' found are white and contours only 'P^T' found are black.

Figure 9: Results of '$H(F)$' on various images using an aggregated graph with $n = 1000$ vertices. The top 20 contours, as ranked by our cost function (Equation 10), are plotted. Observe that our algorithm ('$H(F)$') finds significantly longer contours than Zhu's algorithm ('P').