Interactive Image Segmentation with Multiple Linear Reconstructions in Windows

Shiming Xiang, Chunhong Pan, Feiping Nie, and Changshui Zhang, Member, IEEE

Abstract

This paper proposes an algorithm for interactive image segmentation. The task is formulated as a problem of graph-based transductive classification. Specifically, given an image window, the color of each pixel in it will be reconstructed linearly with those of the remaining pixels in this window. The optimal reconstruction weights will be kept unchanged to linearly reconstruct their class labels. The label reconstruction errors are estimated in each window. These errors are further collected together to develop a learning model. Then, the class information about the user specified foreground and background pixels are integrated into a regularization framework. Under this framework, a globally optimal labeling is finally obtained. The computational complexity is analyzed, and an approach for speeding up the algorithm is presented. Comparative experimental results illustrate the validity of our algorithm.

Index Terms

Interactive image segmentation, multiple linear reconstructions in windows, comparative study.

I. INTRODUCTION

Image segmentation is to partition the image grid into different regions such that the pixels in each region share the same visual characteristics. Although the past decades have yielded many approaches, automatically segmenting natural images is still a difficult task. The difficulties lie
in two aspects. On the low level, it is difficult to model properly the visual elements including colors, textures and other Gestalt characteristics in the image to be segmented. On the high level, it is difficult to group truthfully the visual patterns into the needed object regions. In the absence of prior knowledge about the image, none of these two aspects can be easily solved. In practice, such difficulties encourages the development of interactive image segmentation [2], [3], [9], [11], [16], [19], [24], [25]. With human computer interface, the user can label the foreground and background. In view of pattern classification, such a labeling is fundamentally important as it helps to reduce the complexity of pattern modeling as well as the ambiguity of pattern grouping.

In the past decade, some interactive image segmentation algorithms have been developed [2], [3], [9], [16], [4], [18], [19], [22]. Most of the early techniques such as intelligent scissors [11], [12], snapping [5] and jet-stream [13] require the user to label the pixels near the boundary of the desired objects. For example, when using the intelligent scissors, the user should gaze at the region near the boundary. Labeling in this way is not an easy work.

Recently, the style of user interaction has been significantly improved. Within the interface of the system, the user can drag the mouse to scribe zig-zag lines on the foreground and background regions. Such an improvement of interaction is beneficial from the development of the region-based algorithms. Typical algorithms in this family include magic wand, intelligent paint [1], [15], sketch-based interaction [20], Graph Cut (GC) [2], [3], Grabcut [16], lazy snapping [9], Random Walks (RW) [6], image matting [4], [7], [18], [19], [22], [27], distance-based interaction [14], and so on. Taking the pixels covered by the zig-zag lines as training examples, the segmentation task can be naturally addressed as a problem of pattern classification. This provides the work setting for applying statistical inference or machine learning algorithms to interactive image segmentation [9], [16], [22], [6].

Inferring on Markov Random Field (MRF) constructed on the image grid is a fundamental approach to pixel labeling [8]. The optimization task can be solved via maxflow/mincut [2], [3], [9], [16] or belief propagation [22]. The algorithm is effective in most cases. However, if the foreground and background regions have similar colors, the gap of the likelihood costs in these regions will be decreased. This will degrade the quality of segmentation. Fig. 1 gives an example. We see some background regions are incorrectly segmented.

Recently, Gray proposed Random Walks (RW) for interactive image segmentation [6]. In RW,
each unlabeled pixel will be assigned the same label of the seed point (one of the user labeled pixels) that a random walker starting from this unlabeled pixel reaches first [6]. RW is fast and can provide satisfactory segmentations for most natural images. However, for complex natural image, it may generate unsatisfactory segmentations. Fig. 2 gives an example, where the tail of the leopard is incorrectly segmented. Thus, more user interactions are needed to improve the quality of segmentation.

Fig. 1. Left: the mushroom image [10] with user specified strokes; Right: the segmentation obtained by GC. Some background regions are incorrectly segmented into the foreground object.

Fig. 2. Left: the leopard image [10] with user specified strokes; Right: the segmentation obtained by RW. The tail is incorrectly segmented.

Fig. 3. Left: the cat image [10] with user specified strokes; Right: the segmentation obtained by SR. The segmentation includes some noise.
Algorithms based on discriminative learning have also been introduced into interactive segmentation. Xiang et al. developed Spline Regression (SR) to directly map the pixel features to be class labels [25]. The spline is learned from the user-specified foreground and background pixels, and used as a prediction function for those unlabeled pixels. SR is fast and can generate satisfactory segmentations for most natural images with adequate user specified strokes. However, as it is a discriminative learning algorithm, the segmentation may include some noise. Fig. 3 illustrates an example.

In machine learning, transductive learning [21] is an important inferring method. The goal of the learner in transductive learning is to infer the class labels of the remaining unlabeled data points. Thus, it is suitable for the task of interactive image segmentation. In literature, Zhu et al. proposed an inferring approach with Gaussian Random Field (GRF) [29], and Zhou et al. developed an iterative framework of Learning with Local and Global Consistency (LLGC) [28]. These two algorithms are developed on the edge-weighted graph. Gaussian function are used to evaluate edge weights. However, the parameter of Gaussian function should be well tuned to data.

Later, Xiang et al. proposed Local Spline Regression (LSR) for semi-supervised learning [24]. In contrast, LSR does not contain parameters that should be well tuned to data. As one of its applications, LSR has been applied to interactive image segmentations [24]. But how to speed up LSR with unchanged segmentation accuracy is still a problem to be solved.

This paper presents a graph-based algorithm for interactive image segmentation. Specifically, given a $3 \times 3$ local window, the color of each pixel in it will be linearly reconstructed with those of the remaining eight pixels. The optimal weights will be transferred to linearly reconstruct its class label (foreground/background). This treatment is largely motivated from the manifold learning algorithm of Locally Linear Embedding (LLE) [17]. But beyond LLE where only one data point is reconstructed in each given data neighborhood, we will reconstruct all the pixels in each spatial window. In this process, the label reconstruction errors are estimated. Then, the information about the user specified foreground and background is introduced into a regularization framework. The segmentation task is finally solved via global optimization.

The main advantages or details of our algorithm can be highlighted as follows:

1. The segmentation results achieved with our algorithm are comparable to those with graph cut and random walks algorithms. With the same user strokes, our algorithm can generate more
accurate segmentations on most complex natural images where graph cut and random walks do. Experiments also indicate that our algorithm shows better adaptability to most natural images, compared with GRF and LLGC.

(2) Our algorithm has two parameters. Both of them have their own explicit meanings, which are all independent of data and need not be tuned well from image to image.

(3) The core computation can be easily implemented. The most complex computation is to solve a sparse symmetrical linear equations. In contrast, the main computation time will be taken to fulfil the linear reconstructions in the windows of $3 \times 3$ pixels. To reduce the computation, a fast approach is presented and tested via comparative experiments.

The remainder of this paper is organized as follows. Section II introduces our motivation. In Section III, the algorithm is presented and analyzed. Sections IV reports the experimental results. Section V gives a comparative study between our developed algorithms of SR, LSR and MLRW. The conclusions will be drawn in Section VI.

II. MOTIVATION

A. Problem Formulation

The problem of interactive image segmentation can be formulated as follows. Given an image $I$ with $n = h \times w$ pixels $\{p_i\}_{i=1}^{n}$, two labeled pixel sets of foreground $F$ and background $B$, the task is to assign a label “foreground” or “background” to each of the unlabeled pixel in $I$.

Each pixel $p_i$ can be described with a feature vector $x_i = [r, g, b]^T$ in $\mathbb{R}^3$, where $(r, g, b)$ is the normalized color of $p_i$ in RGB color space, namely, $0 \leq r, g, b \leq 1$. Thus we can get a data set $X = \{x_i\}_{i=1}^{n}$. Then, for $F$ and $B$, we can get two subsets $X_F = \{x_i | p_i \in F\}$ ($\subset X$) and $X_B = \{x_i | p_i \in B\}$ ($\subset X$). All the data points in $X_F$ will be labeled as “+1”, and those in $X_B$ will be labeled as “-1”. For each unlabeled data point $x_i$, we need to assign it a class label $f_i \in \{+1, -1\}$.

In machine learning, this task can be addressed as a problem of transductive classification [21]. In this work setting, we will develop a graph-based algorithm to solve this task.

B. Motivation

To develop a graph-based algorithm of transductive classification, the key is to properly represent the pixels in each window. To this end, we consider to linearly reconstruct their colors.
This treatment is reasonable as in general the colors of the neighboring pixels are similar to each other.

Given pixel \( p_i \in I \) and its \( 3 \times 3 \) spatial neighborhood with \( p_i \) at the center. We further denote the color set of these pixels by \( \mathcal{N}_i = \{x_{ij}\}_{j=1}^9 \), where \( i_j \in \{1, 2, \cdots, n\} \) is a unique index in \( \mathcal{X} \), and \( i_1 = i \). For pixel \( p_i \), we use its eight neighbors surrounding it to linearly reconstruct its color vector \( x_i \):

\[
\hat{x}_i = w_{i,2} x_{i2} + \cdots + w_{i,9} x_{i9},
\]

where \( w_{i,j} (j = 2, 3, \cdots, 9) \) are the reconstruction weights, \( \sum_{j=2}^{9} w_{i,j} = 1 \), and \( \hat{x}_i \) is the reconstructed color vector of \( x_i \). The optimal weights can be obtained by minimizing the squared reconstruction error \( ||x_i - \hat{x}_i||_2^2 \). We have \[17\]

\[
w_i = \frac{(\lambda I + X_i^T X_i)^{-1} 1}{1^T (\lambda I + X_i^T X_i)^{-1} 1},
\]

where \( w_i = [w_{i,2}, \cdots, w_{i,9}]^T \in \mathbb{R}^8 \), \( X_i = [x_{i2} - x_i, \cdots, x_{i9} - x_i] \in \mathbb{R}^{3 \times 8} \), and \( 1 = [1, \cdots, 1]^T \in \mathbb{R}^{k-1} \). In (2), \( I \) is a \( 8 \times 8 \) identity matrix, and \( \lambda \) is a small positive parameter introduced to avoid the (possible) singularity of \( X_i^T X_i \).

Since the class labels are unknown, an alternative way to evaluate the quality of the model is to calculate the squared reconstruction error \( e_i = ||x_i - \hat{x}_i||_2^2 \). Naturally, it is desired to obtain the minimum reconstruction error with its eight pixels surrounding it. However, this goal may not be achieved.

Fig. 4 gives an example. The scissors image [16] is shown in Fig. 4(a). Here we consider the pixel located at position (185, 75). For clarity, we label it as pixel “7” in Fig. 4(b). Within the \( 3 \times 3 \) spatial neighborhood, we can linearly reconstruct its color with the eight pixels “2, 3, 4, 6,
In the case of $\lambda = 0.1$ in (2), the squared reconstruction error is $e_1 = 0.0018386$. Since pixel “7” is a neighbor of pixel “6”, we can also employ “1, 2, 3, 5, 6, 9, 10, 11” to reconstruct it. In this case, the error is reduced to $e_2 = 0.0000373$. We see $e_2 < e_1$. This fact indicates that, the minimum color reconstruction error of a pixel may be obtained not by the eight pixels surrounding it. Such information can be considered in algorithm development.

To utilize the above information, for each pixel in a spatial window, we linearly reconstruct its color vector with those of the remaining pixels in it. The reconstruction weights will be kept to reconstruct their class labels. The label reconstruction errors will be minimized in way of global optimization.

III. MULTIPLE LINEAR RECONSTRUCTIONS IN WINDOWS FOR INTERACTIVE IMAGE SEGMENTATION

A. Multiple Linear Reconstructions in Windows

Given pixel $p_i \in I$ and its $3 \times 3$ local window with nine color vectors in $\mathcal{N}_i = \{x_{ij}\}_{j=1}^9 \subset \mathcal{X}$ (here $i_1 = i$). Now for each pixel $p_{ij}$, we linearly reconstruct its color vector $x_{ij}$ with the remaining eight color vectors in $\mathcal{N}_i$:

$$x_{ij} \approx \sum_{s \neq j} w_{i,j,s} x_{is}, \quad j = 1, 2, \cdots, 9, \quad (3)$$

where $w_{i,j,s}$ is a reconstruction weight, and $\sum_{s \neq j} w_{i,j,s} = 1$.

Let $w_{i,j} = [w_{i,j,1}, \cdots, w_{i,j,j-1}, w_{i,j,j+1}, \cdots, w_{i,j,9}]^T \in \mathbb{R}^8$ and $X_{i,j} = [x_{i1} - x_{ij}, \cdots, x_{ij-1} - x_{ij}, x_{ij+1} - x_{ij}, \cdots, x_{i9} - x_{ij}] \in \mathbb{R}^{3 \times 8}$. Based on (2), then [17]

$$w_{i,j} = \frac{(X_{i,j}^T X_{i,j} + \lambda I)^{-1} 1}{1^T (X_{i,j}^T X_{i,j} + \lambda I)^{-1} 1}, \quad j = 1, 2, \cdots, 9. \quad (4)$$

Now we explain why a term $\lambda I$ is introduced in (4). Actually, the translated data matrix $X_{i,j}$ may equal zero if the pixels in the window all have the same color. Thus, $\lambda I$ can help avoid the (possible) singularity of $X_{i,j}^T X_{i,j}$.

After the optimal weights are estimated, they are transferred to linearly reconstruct the class label of $x_{ij}$. We have

$$f_{ij} \approx \sum_{s \neq j} w_{i,j,s} f_{is}. \quad (5)$$
We see the class labels \( \{ f_{ij} \}_{j=1}^9 \) are all unknown. Now, we consider its squared reconstruction error:

\[
e_{ij} = (f_{ij} - \sum_{s \neq j} w_{i,j,s} f_{is})^2. \tag{6}
\]

Let \( \tilde{w}_{i,j} = [-w_{i,j,1}, \ldots, -w_{i,j,j-1}, 1 - w_{i,j,j+1}, \ldots, -w_{i,j,9}]^T \in \mathbb{R}^9 \), and \( f_i = [f_{i1}, f_{i2}, \ldots, f_{i9}]^T \in \mathbb{R}^9 \) be a sub-vector of \( f \) indexed by \( i, i_2, \ldots, i_9 \). Then, the error in (6) can be rewritten as [26]

\[
e_{i,j} = f_i^T \tilde{w}_{i,j} \tilde{w}_{i,j}^T f_i = f_i^T M_{i,j} f_i, \tag{7}
\]

where

\[
M_{i,j} = \tilde{w}_{i,j} \tilde{w}_{i,j}^T \in \mathbb{R}^{9 \times 9}. \tag{8}
\]

In window \( \mathcal{N}_i \), we get nine squared errors to be minimized. To this end, we first consider their sum as follows:

\[
e_i = \sum_{j=1}^{9} e_{i,j} = \sum_{j=1}^{9} f_i^T M_{i,j} f_i = f_i^T M_i f_i, \tag{9}
\]

where

\[
M_i = \sum_{j=1}^{9} M_{i,j}. \tag{10}
\]

Note that the image has \( n = h \times w \) pixels, and for each pixel we can get a window of \( 3 \times 3 \) pixels. Thus, totally we can get \( n \) errors evaluated respectively from \( n \) windows. Adding them together, then we have

\[
E(f) = \sum_{i=1}^{n} e_i = \sum_{i=1}^{n} f_i^T M_i f_i. \tag{11}
\]

Actually, \( f_i \) is a sub-vector of \( f = [f_1, f_2, \ldots, f_n]^T \), which can be selected out from \( f \) with indices \( i, i_2, \ldots, i_9 \). That is, with a selection matrix \( S_i \in \mathbb{R}^{9 \times n} \), we have \( f_i = S_i f \). Here the \( r \)-th row and \( c \)-th column element \( s_i(r, c) \) of \( S_i \) is defined as follows: \( s_i(r, c) = 1 \) if \( c = i_r \); 0, otherwise. Then,

\[
E(f) = \sum_{i=1}^{n} f^T S_i^T M_i S_i f = f^T M f, \tag{12}
\]

where

\[
M = \sum_{i=1}^{n} S_i^T M_i S_i. \tag{13}
\]
B. Solving Class Labels for Interactive Image Segmentation

Our goal is to minimize $E(f)$ evaluated on the grid graph. Moreover, to achieve the goal of interactive image segmentation, it is also necessary to minimize the label prediction errors of the pixels specified by the user in the human-computer interface. By summing these errors together, an objective function can be constructed as follows:

$$G(f) = f^T M f + \gamma \left( \sum_{p_i \in F} (1 - f_i)^2 + \sum_{p_i \in B} (-1 - f_i)^2 \right),$$  \hspace{1cm} (14)

where $\gamma$ is a positive trade-off parameter.

Parameter $\gamma$ in (14) has an explicit meaning. In the case of $\gamma = +\infty$, minimizing $G(f)$ will output “+1” for each of the user specified foreground pixels, and “-1” for each of the user specified background pixels. Thus in this case, the class labels of the user labeled pixels will be exactly satisfied. In computation, we can take $\gamma$ as a large positive number.

By differentiating the objective function $G(f)$ with respect to $f$ and setting the derivative to be zero, it follows

$$ (M + \gamma C) f = y, $$  \hspace{1cm} (15)

where $C$ is a diagonal matrix with diagonal elements:

$$ C(i,i) = \begin{cases} 
1, & \text{if pixel } p_i \text{ is labeled by the user} \\
0, & \text{otherwise.} 
\end{cases} $$  \hspace{1cm} (16)

In (15), $y$ is a known vector. Let $y = [y_1, y_2, \cdots, y_n]^T \in \mathbb{R}^n$. Then, the element $y_i$ gives

$$ y_i = \begin{cases} 
\gamma, & \text{if pixel } p_i \in F \\
-\gamma, & \text{if pixel } p_i \in B \\
0, & \text{otherwise.} 
\end{cases} $$  \hspace{1cm} (17)

Finally, after $f_i$ is solved, the class label of pixel $p_i$ can be assigned as “+1”, if $f_i \geq 0$; “-1”, otherwise.

C. The Algorithm

The steps of the algorithm, Multiple Linear Reconstructions in Windows (MLRW), are listed in Table I. The flowchart is shown in Fig. 5. We see, except solving the linear equations in (15), there are no complex computations in MLRW. In addition, $M$ is a highly sparse matrix. This
fact can be explained as follows. Note that the size of each $M_i$ is $9 \times 9$ and the $3 \times 3$ windows are overlapped with each other. Thus, for image with $n = w \times h$ pixels, we only need to allocate about $[h/3] \times [w/3] \times 81$ non-zero elements, where $[\cdot]$ stands for the integer not greater than the number. For example, for images with $480 \times 320$ pixels, we will get a matrix $M$ in $\mathbb{R}^{153600 \times 153600}$. But the sparsity ratio will be up to about $153600 \times (153600 - 9) / 153600^2 \approx 99.99\%$. Sparsity will facilitate the storage and help to reduce the computational complexity from $O(n^2)$ to $O(n)$. As a result, the linear equations in (15) can be solved efficiently.

**Algorithm 1 Algorithm of MLRW**

**Input**: Image $I$ with $n = w \times h$ pixels $\{p_i\}_{i=1}^{n}$ to be segmented; the set of the user specified foreground pixels $F$ and the set of the user specified background pixels $B$; two parameters $\lambda$ and $\gamma$.

**Output**: The segmentation of $I$.

1: Construct $X = \{x_i\}_{i=1}^{n}$, where $x_i = [r, g, b]^T \in \mathbb{R}^3$.

2: Allocate a sparse matrix $M \in \mathbb{R}^{n \times n}$.

3: for each pixel $p_i$, $i = 1, 2, \cdots, n$, do

4: Allocate a zero matrix $M_i \in \mathbb{R}^{9 \times 9}$.

5: for $j = 1, 2, \cdots, 9$, do

6: Calculate $M_{i,j}$, according to (8).

7: $M_i \leftarrow M_i + M_{i,j}$.

8: end for

9: $M \leftarrow M + S_i^T M_i S_i$, according to (13).

10: end for

11: Construct diagonal matrix $C$, according to (16).

12: Construct vector $y \in \mathbb{R}^n$, according to (17).

13: Solve $f$, according to (15).

14: for $i = 1, 2, \cdots, n$, do

15: Label $p_i$ as “+1”, if $f_i \geq 0$; “-1”, otherwise.

16: end for

Now we further explain the performance of our algorithm. In Fig. 4, in our way, pixel “9” will be employed to reconstruct pixel “7”. Note that pixel “9” is also in the $5 \times 5$ window with pixel “7” at the center. Then a question is that if it is enough to reconstruct pixel “7” only once with the pixels in the $5 \times 5$ window. For this point, we have the following conclusion: **the performance of reconstructing respectively all of the pixels in $3 \times 3$ windows will not be**
Fig. 5. Flowchart of the algorithm.

Fig. 6. Left column: source images with the user specified strokes; Middle column: results obtained by SLRW with $5 \times 5$ windows; Right column: results obtained by our MLRW with $3 \times 3$ windows.

\textit{equivalent to that of reconstructing only the center pixel in $5 \times 5$ windows.}

As for this point, Fig. 6 gives three examples. In Fig. 6, the left column illustrates the user specified strokes. The middle column shows the segmentation results by only reconstructing the center pixels of $5 \times 5$ windows, namely, using Single Linear Reconstructions in Windows (SLRW). The right column shows the results obtained by MLRW. We see only reconstructing the center pixels even with large image windows may still generate unsatisfactory results. This in turn indicates that MLRW is not equivalent to SLRW with large windows.

Finally, we analyze the computational complexity. Let $k = 9$, then calculating $w_{ij}$ in (4) will
Fig. 7. From the left to the right columns are the segmentation results obtained by MLRW with $\lambda = 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}$, respectively.

Fig. 8. From the left to the right columns are the segmentation results obtained by MLRW with $\gamma = 10^{2}, 10^{3}, 10^{4}, 10^{5}, 10^{6}$, respectively.

scale in about $O(3(k - 1)^2 + (k - 1)^2)$, including calculating $X_{ij}^T X_{ij}$ and solving equations $(X_{ij}^T X_{ij} + \lambda I)z = 1$ to obtain $z = (X_{ij}^T X_{ij} + \lambda I)^{-1}1$. Note that $M_{ij}$ in (8) is a symmetric matrix and the computational complexity will be about $O(k(k + 1)/2)$. Thus, calculating $M_i$ in (10) will scale in about $O(4.5k^3)$. For $n$ pixels, the computational complexity of constructing $M$ in (13) will be up to about $O(4.5nk^3)$. As a summary, the computational complexity of our algorithm is about $O((4.5k^3 + 1)n)$, linear in the number of pixels to be segmented.

IV. EXPERIMENTAL RESULTS

A. Parameter Setting of MLRW

MLRW has two parameters, $\lambda$ in (4) and $\gamma$ in (14). As explained in Section III, $\lambda$ is introduced to avoid the (possible) singularity of matrix $X_{ij}^T X_{ij}$. Thus we can take it as a small positive number. Fig. 7 illustrates the segmentations obtained by MLRW with different $\lambda$ on the three images used in Fig. 6. In Fig. 7, from the left column to the right column are the results.
obtained with $\lambda = 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}$, respectively, by fixing $\gamma = 10000$. We see there are not significant changes when $\lambda$ is a small positive number. In the next experiments, we fix $\lambda = 0.0001$.

Parameter $\gamma$ is introduced to leverage the contributions of the pixels with known class labels. As stated previously, $\gamma$ can be taken as a large number. Fig. 8 shows the results obtained with different $\gamma$ on the three images used in Fig. 6. From the left to the right columns are the results obtained with $\gamma = 10^2, 10^3, 10^4, 10^5, 10^6$, respectively. We see there are almost no changes between the segmentations. This indicates that $\gamma$ can be selected from a very large interval.

B. Comparisons

Here we compare MLRW with the commonly-used algorithms of Graph Cut (GC) [2], [3], [9] and Random Walks (RW) [6] in interactive image segmentation. We also compare it with the classical transductive algorithms of GRF [29] and LLGC [28]. In addition, SLRW will be also compared to illustrate the effectiveness of our algorithm.

In GC, the algorithm in [3] is implemented. The label likelihoods of pixels are calculated via the approach used in [9]. To speed up the calculation, Kmeans clustering algorithm with 20 clusters is run to cluster respectively the colors of the user specified foreground and background.
The Berkeley database [10] and Grabcut database [16] are used to conduct the experiments. Segmentations on twenty images are reported here. For clarity, we use Fig. 9, Fig. 10, Fig. 11 and Fig. 12 to illustrate the results obtained by different algorithms. In each figure, in the first and second columns are the source images and the user specified strokes. From the third to the eighth column are the results obtained by GC, RW, GRF, LLGC, SLRW and MLRW, respectively. The last column lists the ground truth for comparison.

To run RW, we downloaded the source codes from the author’s homepage, and kept all the default parameters unchanged. In GRF and LLGC, the graph is constructed with $3 \times 3$ local windows. Both GRF and LLGC employ Gaussian weighting function to evaluate the affinity matrix. For each image, we calculate the mean distance $d_m$ between the color vectors of pixels. Then the Gaussian parameter is taken as $0.5d_m$. In addition, LLGC has a normalized parameter $\alpha$ [28]. In experiments, we fix it to be 0.99. When running SLRW and MLRW, we set $\lambda = 0.0001$ and $\gamma = 10000$.

As can be seen, GC can generate satisfactory segmentations where the foreground and background pixels have different colors. If the foreground and background have similar colors and
those colors are not labeled by the user, GC may generate unsatisfactory segmentations. This can be witnessed from the last image in Fig. 10, the first and the second images in Fig. 11, and the third image in Fig. 12.

RW is a powerful algorithm. However, it may also generate unsatisfactory results for complex natural images. This can be observed from the last image in Fig. 9 and the second image in Fig. 11. More user specified strokes are needed to guarantee that the random walk starting from an unlabeled pixel meets first the labeled pixel belonging to its own class.

In most experiments, GRF and LLGC generate unsatisfactory results. More user specified strokes are needed to block the leaking of label propagation into the unwanted regions (see Fig. 13). In addition, the segmentations also indicate that MLRW significantly outperforms SLRW.

Table I gives a quantitative comparison. The segmentation accuracy is calculated as the ratio of correct segmented pixels with respect to the ground truth segmented by hand. The numbers in the first rows correspond orderly to those twenty images. In contrast, in most experiments, our algorithm achieves the highest accuracy.

Now we report the computation time. For image with $481 \times 321$ pixels, finishing all the computations with GC, RW, GRF, LLGC and MLRW will take about $79.0, 78.0, 43.0, 3.8$ and
Fig. 12. Demo IV: Segmentation results of the images from Berkeley image database. The images are scaled for arrangement.

Fig. 13. Segmentation results obtained by GC, RW, GRF, LLGC, SLRW and MLRW, with different user-specified strokes. MLRW generates satisfactory segmentation with a small number of user specified strokes.

Fig. 14. Segmentation results obtained by MLRW with different $\alpha$. From the second to the last columns are those with $\alpha = 0, 0.02, \cdots, 0.2$. 

October 20, 2011 DRAFT
a half of the computation time will be taken to calculate these matrices. Calculating $481 \times 321$ matrices \{M$_i$\} in (10) will need about 67.0 seconds. We see more than half of the computation time will be taken to calculate these matrices.

Finally, we point out the fact that, with enough interactions, the algorithms of GC, RW, GRF, LLGC and SLRW all have the ability to generate satisfactory results (see Fig. 13). This can be achieved by adding more and more user-specified strokes. However, in usage, there are two natural criteria to justify the performance of the algorithms. On the one hand, we hope the algorithm can get better segmentation with the same user specified strokes. On the other hand, to achieve the same segmentation accuracy, we hope the number of the user specified strokes is as small as possible. Fig. 13 gives an example. The image is also shown in the last row in Fig. 10, where we see with a small number of strokes the above five algorithms all fail to cut out the desired foreground. The results in Fig. 13 indicate that the segmentation quality can be significantly improved when more user specified strokes are added. In contrast, MLRW generates satisfactory segmentation, but with a small number of user specified strokes.

119.0 seconds respectively, using Matlab 7.0 on a PC with 3.0 GHz CPU and 4.0G RAM (Note that, assembling matrix M in step 9 of Algorithm 1 is done in Matlab setting, by calling a DLL of C codes). In contrast, our algorithm will take more computation time. Specifically for MLRW, $M \times 321$ matrices \{M$_i$\} in (10) will need about 67.0 seconds. We see more than half of the computation time will be taken to calculate these matrices.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{accuracy_time_curves.png}
\caption{The accuracy curves and computation time curves with different $\alpha$.}
\end{figure}
### TABLE I

<table>
<thead>
<tr>
<th></th>
<th>GC</th>
<th>RW</th>
<th>GRF</th>
<th>LLGC</th>
<th>SLRW</th>
<th>MLRW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>99.1237</td>
<td>95.5690</td>
<td>93.5651</td>
<td>91.6081</td>
<td>93.4961</td>
<td><strong>99.5638</strong></td>
</tr>
<tr>
<td>2</td>
<td>97.9479</td>
<td>98.6284</td>
<td>94.8750</td>
<td>93.8701</td>
<td>96.1372</td>
<td><strong>98.9700</strong></td>
</tr>
<tr>
<td>3</td>
<td>97.8086</td>
<td>99.1341</td>
<td>88.1380</td>
<td>89.1849</td>
<td>88.0169</td>
<td><strong>99.3411</strong></td>
</tr>
<tr>
<td>4</td>
<td><strong>99.7383</strong></td>
<td>99.6823</td>
<td>96.9245</td>
<td>94.3073</td>
<td>98.6719</td>
<td>99.6576</td>
</tr>
<tr>
<td>5</td>
<td>98.3460</td>
<td>97.7481</td>
<td>98.3102</td>
<td>96.2405</td>
<td>95.4303</td>
<td><strong>98.9827</strong></td>
</tr>
<tr>
<td>6</td>
<td>99.4252</td>
<td>96.3348</td>
<td>95.6978</td>
<td>96.9926</td>
<td>97.6400</td>
<td><strong>99.5719</strong></td>
</tr>
<tr>
<td>7</td>
<td>98.8904</td>
<td>99.2844</td>
<td>97.3319</td>
<td>98.6889</td>
<td>98.7526</td>
<td><strong>99.3748</strong></td>
</tr>
<tr>
<td>8</td>
<td>94.4543</td>
<td>98.8276</td>
<td>95.9512</td>
<td>95.3788</td>
<td>97.3201</td>
<td><strong>98.8276</strong></td>
</tr>
<tr>
<td>10</td>
<td>84.1156</td>
<td>99.2726</td>
<td>95.9304</td>
<td>97.4504</td>
<td>98.3704</td>
<td><strong>99.6711</strong></td>
</tr>
<tr>
<td>11</td>
<td>86.4399</td>
<td>99.4514</td>
<td>96.6704</td>
<td>98.2371</td>
<td>98.8439</td>
<td><strong>99.4754</strong></td>
</tr>
<tr>
<td>12</td>
<td>83.4476</td>
<td>97.2422</td>
<td>97.2371</td>
<td>97.1419</td>
<td>97.3316</td>
<td><strong>98.3349</strong></td>
</tr>
<tr>
<td>13</td>
<td>98.5648</td>
<td>95.5745</td>
<td>89.6937</td>
<td>89.4800</td>
<td>93.8673</td>
<td><strong>98.6781</strong></td>
</tr>
<tr>
<td>14</td>
<td>95.0337</td>
<td>99.0447</td>
<td>98.2947</td>
<td>98.9352</td>
<td>94.8491</td>
<td><strong>99.0939</strong></td>
</tr>
<tr>
<td>15</td>
<td>98.4916</td>
<td>98.6088</td>
<td>94.9016</td>
<td>96.1432</td>
<td>96.3757</td>
<td><strong>99.0959</strong></td>
</tr>
<tr>
<td>16</td>
<td>99.4229</td>
<td><strong>99.5745</strong></td>
<td>78.1044</td>
<td>80.1187</td>
<td>96.9638</td>
<td>99.4644</td>
</tr>
<tr>
<td>17</td>
<td>96.5525</td>
<td>97.0305</td>
<td>94.5603</td>
<td>94.4165</td>
<td>96.8491</td>
<td><strong>98.1114</strong></td>
</tr>
<tr>
<td>18</td>
<td>93.1393</td>
<td>99.3141</td>
<td>97.5033</td>
<td>98.2124</td>
<td>98.8692</td>
<td><strong>99.3679</strong></td>
</tr>
<tr>
<td>19</td>
<td>98.6503</td>
<td>97.8446</td>
<td>96.2338</td>
<td>97.3517</td>
<td>96.2546</td>
<td><strong>98.8782</strong></td>
</tr>
<tr>
<td>20</td>
<td>99.4087</td>
<td>99.2727</td>
<td>97.4689</td>
<td>98.3711</td>
<td>99.0531</td>
<td><strong>99.5045</strong></td>
</tr>
</tbody>
</table>

---

### C. Speeding up MLRW

This subsection will introduce a method to reduce the calculations of matrices \( \{M_i\} \) in (10).

Note that natural images usually contain small regions with the same pixel colors. That is, if the pixels in a \( 3 \times 3 \) window have the same color, matrix \( X_i^T X_i \) in (4) will be zero. In this case, all the elements in \( w_{i,j} \) will be equal to each other. As a result, \( M_i \) in (10) will turn out to be a constant matrix. This can be utilized to reduce the computations.

Actually, we can relax the above condition. In other words, if the standard deviation of the color is less than a given threshold, a constant matrix can be used to replace \( M_i \). This condition...
TABLE II
THE SEGMENTATION ACCURACY (%) OF THE 20 IMAGES, OBTAINED BY SR, LSR AND MLRW.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>97.4553</td>
<td>96.0214</td>
<td>98.6121</td>
<td>97.2429</td>
<td>98.7753</td>
<td>97.8102</td>
<td>95.4806</td>
<td>99.2526</td>
<td>99.0680</td>
<td>98.9586</td>
</tr>
</tbody>
</table>

can be formulated as follows:

$$\max(\text{std}(r), \text{std}(g), \text{std}(b)) \leq \alpha,$$

where $\alpha$ is a given threshold, and $\text{std}(r)$, $\text{std}(g)$ and $\text{std}(b)$ are the standard deviations of the normalized red, green and blue color components in the $3 \times 3$ window.

Here we report the experiments on four images. The first column in Fig. 14 illustrates these images. They are reported again here since they have different types of visual appearances. In Fig. 14, from the second to the last column are the results obtained with $\alpha = 0, 0.02, 0.04, \cdots, 0.2$, respectively. Fig. 15(a), Fig. 15(b), Fig. 15(c) and Fig. 15(d) illustrate the accuracy curves, with different $\alpha$. As can be seen, in the case that $0 < \alpha \leq 0.06$, the segmentation accuracy keeps at the same level as that obtained with the original algorithm ($\alpha = 0$). This indicates that the above fast implementation can allow about 15 ($\approx 255 \times 0.06$) deviation of intensity. Correspondingly, Fig. 15(e), Fig. 15(f), Fig. 15(g) and Fig. 15(h) show the computation time. For the elephant image, the computation time of constructing $\{M_i\}$ is drastically reduced even with $\alpha = 0.02$. For other three images, when $\alpha$ is up to 0.06, the computation time is reduced to about a half of the original computation time.

V. COMPARATIVE STUDY ON SR, LSR, AND MLRW

In this section, we will conduct a comparative study on our developed algorithms of SR [25], LSR [24] and MLRW.
Fig. 16 illustrates the segmentations of the 20 images obtained respectively by these three algorithms. The user-specified strokes about the background and foreground are shown in Fig. 9, Fig. 10, Fig. 11, and Fig. 12. The same parameter settings reported in [25], [24] are taken here for implementation. Table II lists the segmentation accuracy. We see both LSR and MLRW generate better segmentations, compared with SR. We also see that both LSR and MLRW can generate results with the same level of segmentation quality.

Given a $3 \times 3$ local image window, LSR also constructs a Laplacian matrix. The computational complexity will be up to about $O((k + d + 1)^3)$. In the case of $k = 9$ and $d = 3$, it will slightly smaller than $O(4.5k^3)$. Specifically, for LSR, calculating $481 \times 321$ such matrices will need about
### TABLE III
Comparison between SR, LSR and MLRW.

<table>
<thead>
<tr>
<th>category of algorithm</th>
<th>local function used in each window</th>
<th>accuracy</th>
<th>speed</th>
<th>memory</th>
<th>speeding up under (18)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR</td>
<td>discriminative learning</td>
<td>–</td>
<td>low</td>
<td>fast</td>
<td>low</td>
</tr>
<tr>
<td>LSR</td>
<td>transductive learning</td>
<td>nonlinear (one spline)</td>
<td>high</td>
<td>slow</td>
<td>high</td>
</tr>
<tr>
<td>MLRW</td>
<td>transductive learning</td>
<td>linear (multiple linear reconstructions)</td>
<td>high</td>
<td>slow</td>
<td>high</td>
</tr>
</tbody>
</table>

35.0 seconds on average, while MLRW will cost about 67.0 seconds. However, speeding up LSR in the way introduced in Subsection IV-C may decrease the segmentation quality, compared with MLRW. Explanations are given below.

First, for LSR, we have the following conclusion (proof is given in Appendix):

*For any $3 \times 3$ (or any other size) image window with the same pixel color, the Laplacian matrix deduced by LSR will equal to a unique constant Laplacian matrix.*

Then, like MLRW, we can use this conclusion to speed up LSR. Specifically, if the condition (18) is satisfied, the Laplacian matrix in LSR will be replaced by a constant Laplacian matrix. Fig. 17 reports the segmentations of the images used in Fig. 14, with different parameter $\alpha$. The accuracy and computation time curves are shown in Fig. 15. We see that employing the above speed-up approach for LSR may significantly degrade the segmentation quality.

Finally, we give an example to explain the reason. Suppose we are given a $3 \times 3$ window with nine color vectors: $(0.274, 0.620, 0.563)$, $(0.267, 0.612, 0.525)$, $(0.271, 0.605, 0.529)$, $(0.276, 0.631, 0.524)$, $(0.253, 0.614, 0.534)$, $(0.247, 0.627, 0.549)$, $(0.263, 0.603, 0.549)$, $(0.293, 0.647, 0.535)$, and $(0.300, 0.614, 0.554)$. Here the minimum deviation of the components only equals to 0.0171. We denote the matrix calculated according to (10) by $M_1$, and the constant matrix by $C_1$. We further denote the Laplacian matrix in LSR by $M_2$, and the constant Laplacian matrix by $C_2$. Then, we use Frobenius norm to measure the difference between matrices. We have $||M_1 - C_1||_F = 5.5164$ and $||M_2 - C_2||_F = 1.7321 \times 10^4$. In contrast to MLRW, we see simplifying the computation in LSR will cause larger difference. This explains why LSR may significantly degrade the segmentation quality when performing the above simplification of computation.
Now we summarize SR, LSR, and MLRW as follows (see Table III):

(1) SR is developed in view of discriminative learning. That is, the features of the user specified foreground and background pixels are employed to train a spline, which is used as a prediction function for those unlabeled pixels. SR need not solve a large group of linear equations. Thus, it is fast and can run with low memory. However, SR may generate segmentations with noises (see Fig. 16). The reason is that the pixels are segmented one-by-one by the learned spline, without considering their spatial relations on the image grid.

(2) LSR is developed SR from discriminative learning to transductive learning. Differently, SR learns a unique spline for all of the pixels to be segmented, while LSR employs a group of splines, each of which is used to only map the pixels in a $3 \times 3$ window. As a graph-based learning algorithm, LSR explicitly utilizes the spatial relations between pixels when it is applied to image segmentation. As a result, the segmentation quality is significantly improved. However, LSR needs to construct the Laplacian matrices in image windows and solve large-scale linear equations. More time will be taken and large memory will be required to fulfill the segmentation.

(3) MLRW is also a transductive learning algorithm. Differently, LSR uses spline to map the pixels in each window, nonlinearly, into their classes labels, while MLRW linearly reconstructs them. Experiments shows that MLRW and LSR can output segmentation with high accuracy. However, as analyzed above, speeding up for LSR under condition (18) may significantly degrade the segmentation quality.

Based on SR, LSR and MLRW, one future work can focus on developing a segmentation system for very large images. To this end, the large-scale sparse linear equations in LSR and MLRW can be solved iteratively by combining conjugate gradient, image pyramid and multi-grid methods. In this process, SR can be used to provide an initial solution.

VI. CONCLUSIONS

We presented a graph-based classification algorithm for interactive image segmentation. It is developed with multiple linear reconstructions in image windows. The key idea is to linearly reconstruct the color vector of each pixel with those of the remaining pixels also in the window. The estimated optimal reconstruction weights are transferred to linearly reconstruct the class label of each pixel. In this way, the label reconstruction errors are estimated and minimized to obtain the final segmentation. We analyzed the proposed algorithm, and reported the experiments...
on many types of natural images. A speeding up approach is presented. We also conducted a comparative study between our developed algorithms. Comparative experimental results illustrate the validity of our algorithm.

**APPENDIX**

In this appendix, we will give a proof about the property mentioned in Section V.

Without loss of generality, we consider $3 \times 3$ windows. Suppose the nine pixel has the same color vector $\mathbf{p} = [r, g, b]^T \in [0, 1]^3$. In this case the coefficient matrix for solving the spline (Equation (9) in [24]) will turn out to be

$$
\mathbf{A} = \begin{pmatrix}
\lambda \mathbf{I} & \mathbf{U} \\
\mathbf{U}^T & 0
\end{pmatrix} \in \mathbb{R}^{13 \times 13},
$$

(19)

where $\lambda$ is a small positive number, $\mathbf{I}$ is a $9 \times 9$ identity matrix, $\mathbf{U} = [\mathbf{e}, \mathbf{ep}^T] \in \mathbb{R}^{9 \times 4}$, and $\mathbf{e} = [1, 1, \cdots, 1] \in \mathbb{R}^9$.

The Laplacian matrix deduced by LSR is the $9 \times 9$ top-left sub-matrix of $\mathbf{A}^{-1}$ [24]. In the case that the image window has the same color vector $\mathbf{p}$, we see $\mathbf{A}$ is a singular matrix. Thus here $\mathbf{A}^{-1}$ is actually calculated as its Moore-Penrose inverse matrix $\mathbf{A}^+$. Now we can re-state the property in Section V as a theorem:

**Theorem.** For any $\mathbf{p} = [r, g, b]^T \in [0, 1]^3$, the $9 \times 9$ top-left sub-matrix of $\mathbf{A}^+$, namely, the Laplacian matrix $\mathbf{C}$ (the matrix $\mathcal{M}_i$ in Equation (11) in [24]), equals to

$$
\mathbf{C} = (\mathbf{I} - \mathbf{ee}^T/9)/\lambda.
$$

(20)

To prove this theorem, we have the following lemma [23]:

**Lemma.** Given $\mathbf{P} \in \mathbb{R}^{n \times N}$ and $\mathbf{Q} \in \mathbb{R}^{N \times n}$, then $\mathbf{PQ}$ and $\mathbf{QP}$ have the same nonzero eigenvalues. For each nonzero eigenvalue of $\mathbf{PQ}$, if the corresponding eigenvector of $\mathbf{PQ}$ is $\mathbf{v}$, then the corresponding eigenvector of $\mathbf{QP}$ is $\mathbf{w} = \mathbf{Qv}$.

Now let $\mathbf{u} = [1, \mathbf{p}^T]^T \in \mathbb{R}^4$. Then we have $\mathbf{U} = \mathbf{eu}^T$. Further let $\mathbf{T} = \mathbf{U}^T\mathbf{U} = \mathbf{ee}^T\mathbf{eu}^T = 9\mathbf{uu}^T \in \mathbb{R}^{4 \times 4}$. We see $\mathbf{T}$ is a rank-one matrix. Based on the above lemma, it has an eigenvalue $\mathbf{u}^T\mathbf{u}$ and an eigenvector $\mathbf{u}$. According to the definition of Moore-Penrose inverse matrix, we have

$$
\mathbf{T}^+ = \frac{1}{9(\mathbf{u}^T\mathbf{u})^2}\mathbf{uu}^T \in \mathbb{R}^{4 \times 4}.
$$

(21)
Now we can prove the above theorem as follows:

**Proof:** According to matrix theory, formally we have

$$
A^{-1} = \begin{pmatrix}
(I - U(U^TU)^{-1}U^T)/\lambda & U(U^TU)^{-1} \\
(U^TU)^{-1}U^T & -\lambda(U^TU)^{-1}
\end{pmatrix}.
$$

(22)

Using $T^+$ to calculate $(U^TU)^{-1}$ and substituting (21) into (22), we have

$$
A^+ = \begin{pmatrix}
(I - ee^T/9)/\lambda & eu^T/(9u^Tu) \\
eu^T/(9u^Tu) & -\lambda uu^T/(9(u^Tu)^2)
\end{pmatrix}.
$$

(23)

We see the $9 \times 9$ top-left sub-matrix of $A^+$ just equals $(I - ee^T/9)/\lambda$. It is easy to justify that it is a Laplacian matrix. In this way, the conclusion is proved.

Based on (20), $C$ is independent of color vector $p$. When speeding up LSR as mentioned in Section V, it will be used to simplify the computation if the condition in (18) is held.

**ACKNOWLEDGMENT**

This work was supported by the Hi-Tech Research and Development (863) Program of China (Grant No. 2009AA012104), the National Natural Science Foundation of China (Grant No. 60873161, 60975037), and the National Basic Research Program of China (Grant No. 2009CB320602).

**REFERENCES**


