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

Shiming Xiang and Chunhong Pan are with the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, Beijing, 100190, China (e-mail: smxiang@gmail.com, chpan@nlpr.ia.ac.cn)

Feiping Nie, and Changshui Zhang are with the State Key Laboratory of Intelligent Technology and Systems, Tsinghua National Laboratory for Information Science and Technology (TNList), Department of Automation, Tsinghua University, Beijing 100084, China (feipingnie@gmail.com, zcs@mail.tsinghua.edu.cn)

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,

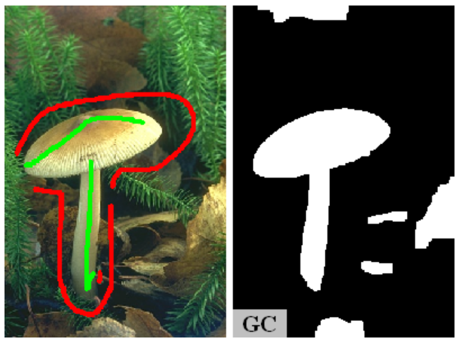


Fig. 1. Left: the mushroom image [10] with user specified strokes; Right: the segmentation obtained by GC. Some background regions are incorrectly segmented into the foreground object.

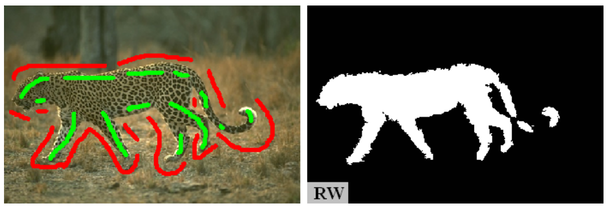


Fig. 2. Left: the leopard image [10] with user specified strokes; Right: the segmentation obtained by RW. The tail is incorrectly segmented.

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.

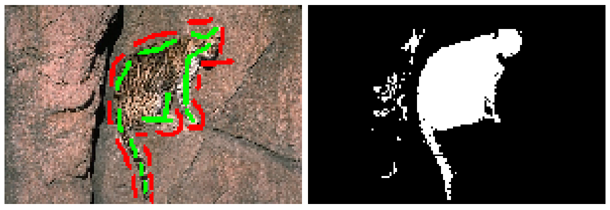


Fig. 3. Left: the cat image [10] with user specified strokes; Right: the segmentation obtained by SR. The segmentation includes some noise.

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 3×3 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 3×3 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

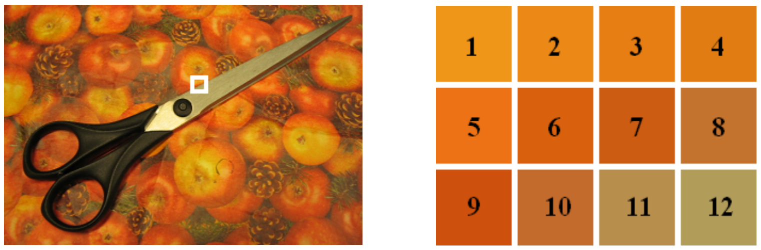
The problem of interactive image segmentation can be formulated as follows. Given an image $\mathcal{I}$ with $n = h \times w$ pixels $\{p_i\}_{i=1}^{n}$, two labeled pixel sets of foreground $\mathcal{F}$ and background $\mathcal{B}$, the task is to assign a label "foreground" or "background" to each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

Each pixel p_i can be described with a feature vector $\mathbf{x}_i = [r, g, b]^T$ 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 $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{n}$. Then, for $\mathcal{F}$ and $\mathcal{B}$, we can get two subsets $\mathcal{X}_{\mathcal{F}} = \{\mathbf{x}_i | p_i \in \mathcal{F}\}$ $(\subset \mathcal{X})$ and $\mathcal{X}_{\mathcal{B}} = \{\mathbf{x}_i | p_i \in \mathcal{B}\}$ $(\subset \mathcal{X})$. All the data points in $\mathcal{X}_{\mathcal{F}}$ will be labeled as "+1", and those in $\mathcal{X}_{\mathcal{B}}$ will be labeled as "-1". For each unlabeled data point $\mathbf{x}_i$, 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.



(a) (b)

Fig. 4. (a): Scissors image [16] with 337×225 pixels. (b): Twelve pixels located in the scissors image in (a) with coordinates: $\{(x,y)|x \in [183,186], y \in [74,76]\}$ (at the center of the white box in (a)).

This treatment is reasonable as in general the colors of the neighboring pixels are similar to each other.

Given pixel $p_i \in \mathcal{I}$ and its 3×3 spatial neighborhood with p_i at the center. We further denote the color set of these pixels by $\mathcal{N}_i = \{\mathbf{x}_{i_j}\}_{j=1}^{9}$, 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 $\mathbf{x}_i$:

$$\hat{\mathbf{x}}_i = w_{i,2}\mathbf{x}_{i_2} + \cdots + w_{i,9}\mathbf{x}_{i_9}, \tag{1}$$

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

$$\mathbf{w}_i = \frac{(\lambda \mathbf{I} + \mathbf{X}_i^T \mathbf{X}_i)^{-1}\mathbf{1}}{\mathbf{1}^T(\lambda \mathbf{I} + \mathbf{X}_i^T \mathbf{X}_i)^{-1}\mathbf{1}}, \tag{2}$$

where $\mathbf{w}_i = [w_{i,2}, \cdots, w_{i,9}]^T \in \mathbb{R}^8$, $\mathbf{X}_i = [\mathbf{x}_{i_2} - \mathbf{x}_i, \cdots, \mathbf{x}_{i_9} - \mathbf{x}_i] \in \mathbb{R}^{3\times 8}$, and $\mathbf{1} = [1, \cdots, 1]^T \in \mathbb{R}^{k-1}$. In (2), $\mathbf{I}$ is a 8×8 identity matrix, and λ is a small positive parameter introduced to avoid the (possible) singularity of $\mathbf{X}_i^T \mathbf{X}_i$.

Since the class labels are unknown, an alternative way to evaluate the quality of the model is to calculate the squared reconstruction error $e_i = ||\mathbf{x}_i - \hat{\mathbf{x}}_i||_2^2$. 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 3×3 spatial neighborhood, we can linearly reconstruct its color with the eight pixels "2, 3, 4, 6, 8, 10, 11, 12". 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 wights 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 $p_i \in \mathcal{I}$ and its 3×3 local window with nine color vectors in $\mathcal{N}_i = \{\mathbf{x}_{i_j}\}_{j=1}^{9} \subset \mathcal{X}$ (here $i_1 = i$). Now for each pixel p_{i_j}, we linearly reconstruct its color vector $\mathbf{x}_{i_j}$ with the remaining eight color vectors in $\mathcal{N}_i$:

$$\mathbf{x}_{i_j} \approx \sum_{s \neq j} w_{i,j,s}\, \mathbf{x}_{i_s}, \quad j = 1, 2, \cdots, 9, \tag{3}$$

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

Let $\mathbf{w}_{i,j} = [w_{i,j,1}, .., w_{i,j,j-1}, w_{i,j,j+1}, .., w_{i,j,9}]^T \in \mathbb{R}^8$ and $\mathbf{X}_{i,j} = [\mathbf{x}_{i_1} - \mathbf{x}_{i_j}, .., \mathbf{x}_{i_{j-1}} - \mathbf{x}_{i_j}, \mathbf{x}_{i_{j+1}} - \mathbf{x}_{i_j}, .., \mathbf{x}_{i_9} - \mathbf{x}_{i_j}] \in \mathbb{R}^{3 \times 8}$. Based on (2), then [17]

$$\mathbf{w}_{i,j} = \frac{(\mathbf{X}_{i,j}^T \mathbf{X}_{i,j} + \lambda \mathbf{I})^{-1} \mathbf{1}}{\mathbf{1}^T (\mathbf{X}_{i,j}^T \mathbf{X}_{i,j} + \lambda \mathbf{I})^{-1} \mathbf{1}}, \quad j = 1, 2, \cdots, 9. \tag{4}$$

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

After the optimal weights are estimated, they are transferred to linearly reconstruct the class label of $\mathbf{x}_{i_j}$. We have

$$f_{i_j} \approx \sum_{s \neq j} w_{i,j,s}\, f_{i_s}. \tag{5}$$

We see the class labels $\{f_{i_j}\}_{j=1}^{9}$ are all unknown. Now, we consider its squared reconstruction error:

$$e_{i,j} = (f_{i_j} - \sum_{s \neq j} w_{i,j,s} \, f_{i_s})^2. \tag{6}$$

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

$$e_{i,j} = \mathbf{f}_i^T \tilde{\mathbf{w}}_{i,j} \tilde{\mathbf{w}}_{i,j}^T \mathbf{f}_i = \mathbf{f}_i^T \mathbf{M}_{i,j} \mathbf{f}_i, \tag{7}$$

where

$$\mathbf{M}_{i,j} = \tilde{\mathbf{w}}_{i,j} \tilde{\mathbf{w}}_{i,j}^T \in \mathbb{R}^{9 \times 9}. \tag{8}$$

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

$$e_i = \sum_{j=1}^{9} e_{i,j} = \sum_{j=1}^{9} \mathbf{f}_i^T \mathbf{M}_{i,j} \mathbf{f}_i = \mathbf{f}_i^T \mathbf{M}_i \mathbf{f}_i, \tag{9}$$

where

$$\mathbf{M}_i = \sum_{j=1}^{9} \mathbf{M}_{i,j}. \tag{10}$$

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

$$E(\mathbf{f}) = \sum_{i=1}^{n} e_i = \sum_{i=1}^{n} \mathbf{f}_i^T \mathbf{M}_i \mathbf{f}_i. \tag{11}$$

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

$$E(\mathbf{f}) = \sum_{i=1}^{n} \mathbf{f}^T \mathbf{S}_i^T \mathbf{M}_i \mathbf{S}_i \mathbf{f} = \mathbf{f}^T \mathbf{M} \mathbf{f}, \tag{12}$$

where

$$\mathbf{M} = \sum_{i=1}^{n} \mathbf{S}_i^T \mathbf{M}_i \mathbf{S}_i. \tag{13}$$

B. Solving Class Labels for Interactive Image Segmentation

Our goal is to minimize $E(\mathbf{f})$ 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:

$$G(\mathbf{f}) = \mathbf{f}^T\mathbf{M}\mathbf{f} + \gamma(\sum_{p_i \in \mathcal{F}} (1 - f_i)^2 + \sum_{p_i \in \mathcal{B}} (-1 - f_i)^2), \tag{14}$$

where γ is a positive trade-off parameter.

Parameter γ in (14) has an explicit meaning. In the case of $\gamma = +\infty$, minimizing $G(\mathbf{f})$ 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 γ as a large positive number.

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

$$(\mathbf{M} + \gamma\mathbf{C})\mathbf{f} = \mathbf{y}, \tag{15}$$

where $\mathbf{C}$ is a diagonal matrix with diagonal elements:

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

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

$$y_i = \begin{cases} \gamma, & \text{if pixel } p_i \in \mathcal{F} \\ -\gamma, & \text{if pixel } p_i \in \mathcal{B} \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

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, $\mathbf{M}$ is a highly sparse matrix. This fact can be explained as follows. Note that the size of each $\mathbf{M}_i$ is 9×9 and the 3×3 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 480×320 pixels, we will get a matrix $\mathbf{M}$ 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 $\mathcal{I}$ with $n = w \times h$ pixels $\{p_i\}_{i=1}^n$ to be segmented; the set of the user specified foreground pixels $\mathcal{F}$ and the set of the user specified background pixels $\mathcal{B}$; two parameters λ and γ.

Output: The segmentation of $\mathcal{I}$.

```
1:  Construct $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^n$, where $\mathbf{x}_i = [r, g, b]^T \in \mathbb{R}^3$.
2:  Allocate a sparse matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$.
3:  for each pixel $p_i$, $i = 1, 2, \cdots, n$, do
4:      Allocate a zero matrix $\mathbf{M}_i \in \mathbb{R}^{9 \times 9}$.
5:      for $j = 1, 2, \cdots, 9$, do
6:          Calculate $\mathbf{M}_{i,j}$, according to (8).
7:          $\mathbf{M}_i \leftarrow \mathbf{M}_i + \mathbf{M}_{i,j}$.
8:      end for
9:      $\mathbf{M} \leftarrow \mathbf{M} + \mathbf{S}_i^T \mathbf{M}_i \mathbf{S}_i$, according to (13).
10: end for
11: Construct diagonal matrix $\mathbf{C}$, according to (16).
12: Construct vector $\mathbf{y} \in \mathbb{R}^n$, according to (17).
13: Solve $\mathbf{f}$, 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 5×5 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 5×5 window. For this point, we have the following conclusion: *the performance of reconstructing respectively all of the pixels in 3×3 windows will not be*

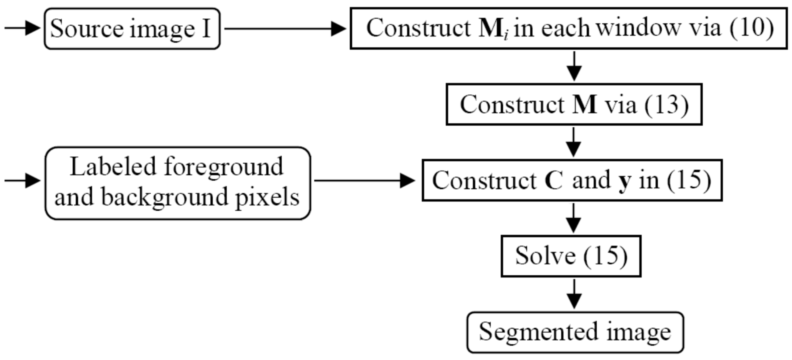

Source image I
Construct $\mathbf{M}_i$ in each window via (10)
Construct $\mathbf{M}$ via (13)
Labeled foreground and background pixels
Construct $\mathbf{C}$ and $\mathbf{y}$ in (15)
Solve (15)
Segmented image

Fig. 5. Flowchart of the algorithm.

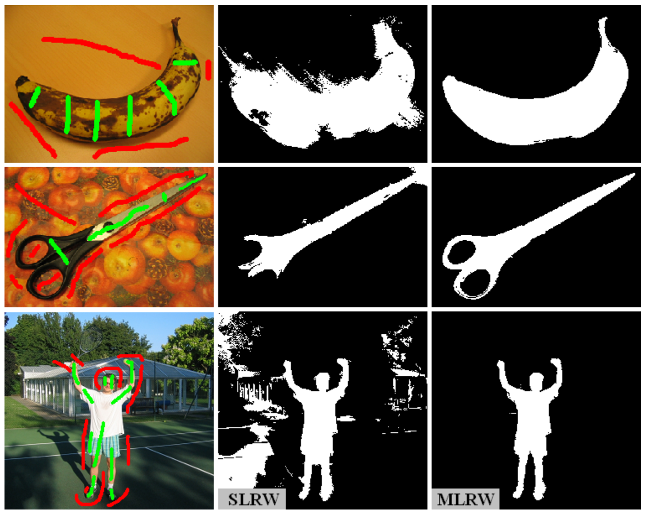


Fig. 6. Left column: source images with the user specified strokes; Middle column: results obtained by SLRW with 5×5 windows; Right column: results obtained by our MLRW with 3×3 windows.

equivalent to that of reconstructing only the center pixel in 5×5 *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 5×5 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 $\mathbf{w}_{i_j}$ in (4) will

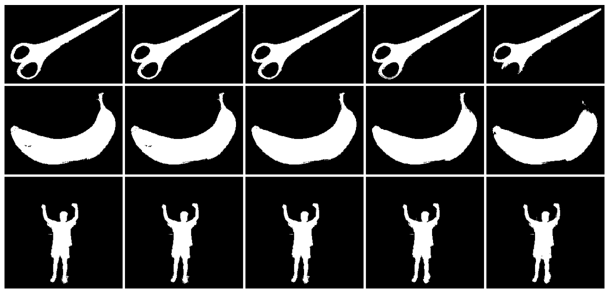


Fig. 7. From the left to the right columns are the segmentation results obtained by MLRW with $\lambda = 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}$, respectively.

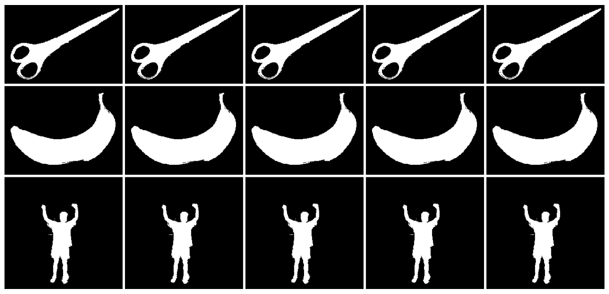


Fig. 8. From the left to the right columns are the segmentation results obtained by MLRW with $\gamma = 10^{2}, 10^{3}, 10^{4}, 10^{5}, 10^{6}$, respectively.

scale in about $O(3(k-1)^2 + (k-1)^2)$, including calculating $\mathbf{X}_{i_j}^T\mathbf{X}_{i_j}$ and solving equations $(\mathbf{X}_{i_j}^T\mathbf{X}_{i_j} + \lambda\mathbf{I})\mathbf{z} = \mathbf{1}$ to obtain $\mathbf{z} = (\mathbf{X}_{i_j}^T\mathbf{X}_{i_j} + \lambda\mathbf{I})^{-1}\mathbf{1}$. Note that $\mathbf{M}_{i_j}$ in (8) is a symmetric matrix and the computational complexity will be about $O(k(k+1)/2)$. Thus, calculating $\mathbf{M}_i$ in (10) will scale in about $O(4.5k^3)$. For n pixels, the computational complexity of constructing $\mathbf{M}$ 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, λ in (4) and γ in (14). As explained in Section III, λ is introduced to avoid the (possible) singularity of matrix $\mathbf{X}_{i_j}^T\mathbf{X}_{i_j}$. Thus we can take it as a small positive number. Fig. 7 illustrates the segmentations obtained by MLRW with different λ on the three images used in Fig. 6. In Fig. 7, from the left column to the right column are the results

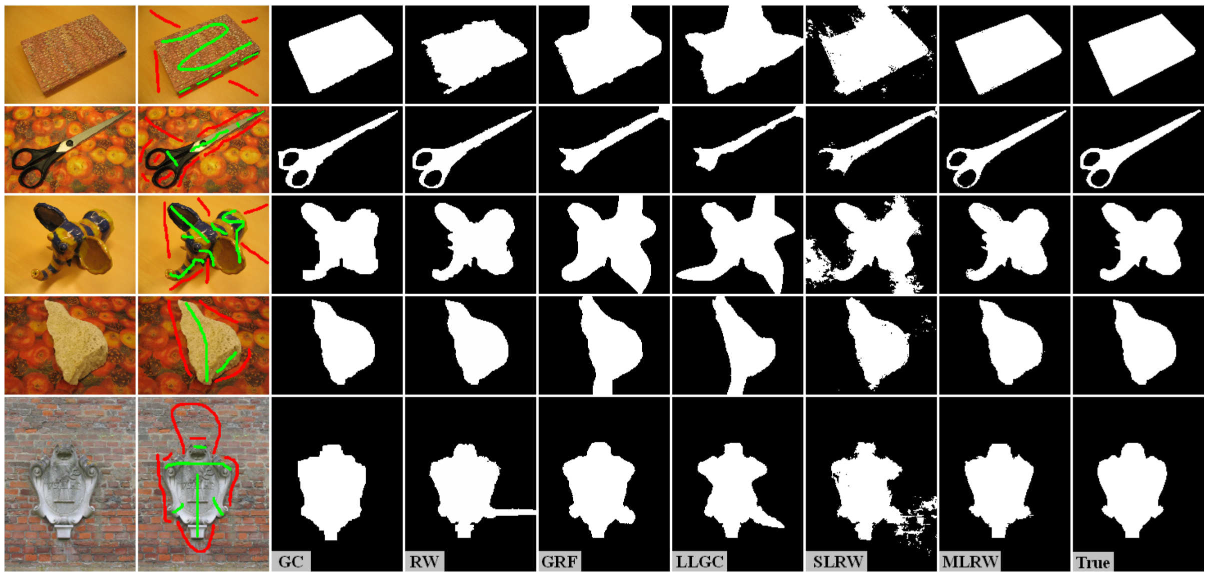


Fig. 9. Demo I: Segmentation results of the images from Grabcut image database. The images are scaled for arrangement.

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 λ is a small positive number. In the next experiments, we fix $\lambda = 0.0001$.

Parameter γ is introduced to leverage the contributions of the pixels with known class labels. As stated previously, γ can be taken as a large number. Fig. 8 shows the results obtained with different γ 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 γ 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

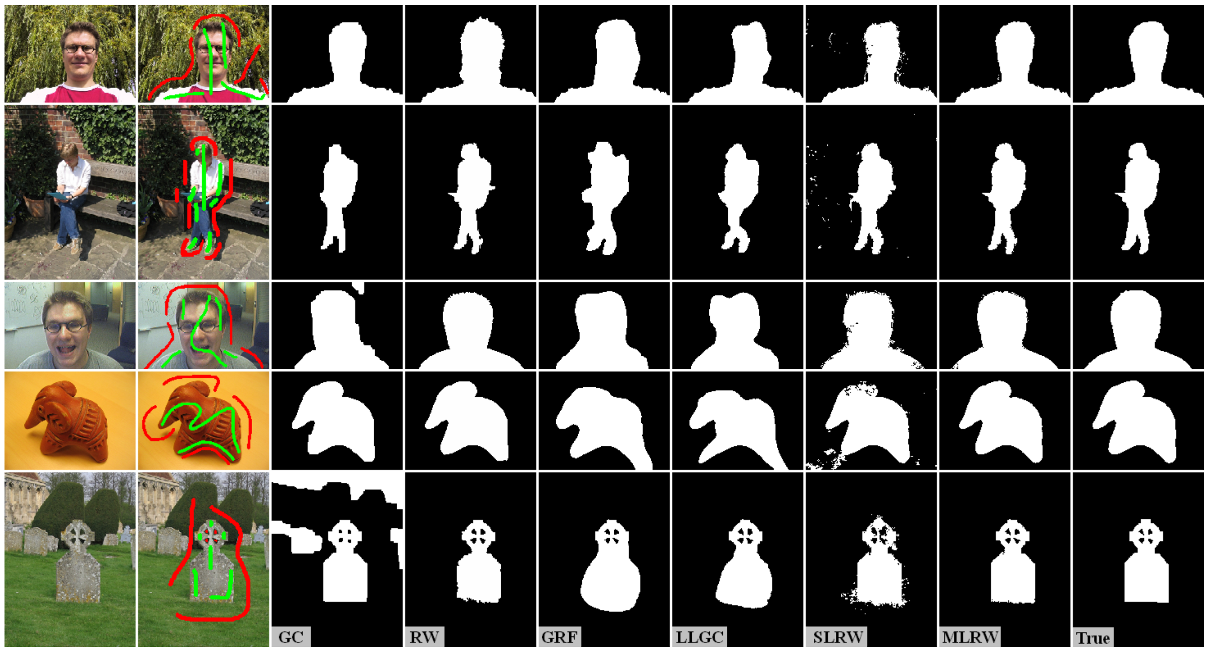


Fig. 10. Demo II: Segmentation results of the images from Grabcut image database. The images are scaled for arrangement.

pixels [9].

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, repsectively. 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 3×3 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 α [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

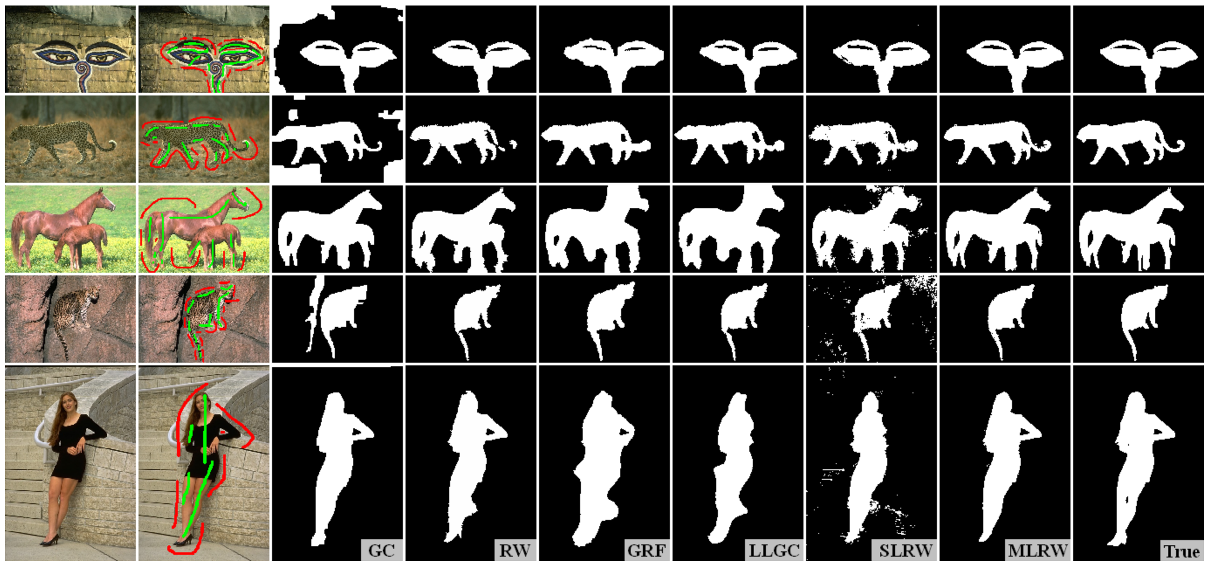


Fig. 11. Demo III: Segmentation results of the images from Berkeley image database. The images are scaled for arrangement.

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 481×321 pixels, finishing all the computations with GC, RW, GRF, LLGC and MLRW will take about 79.0, 78.0, 43.0, 3.8 and

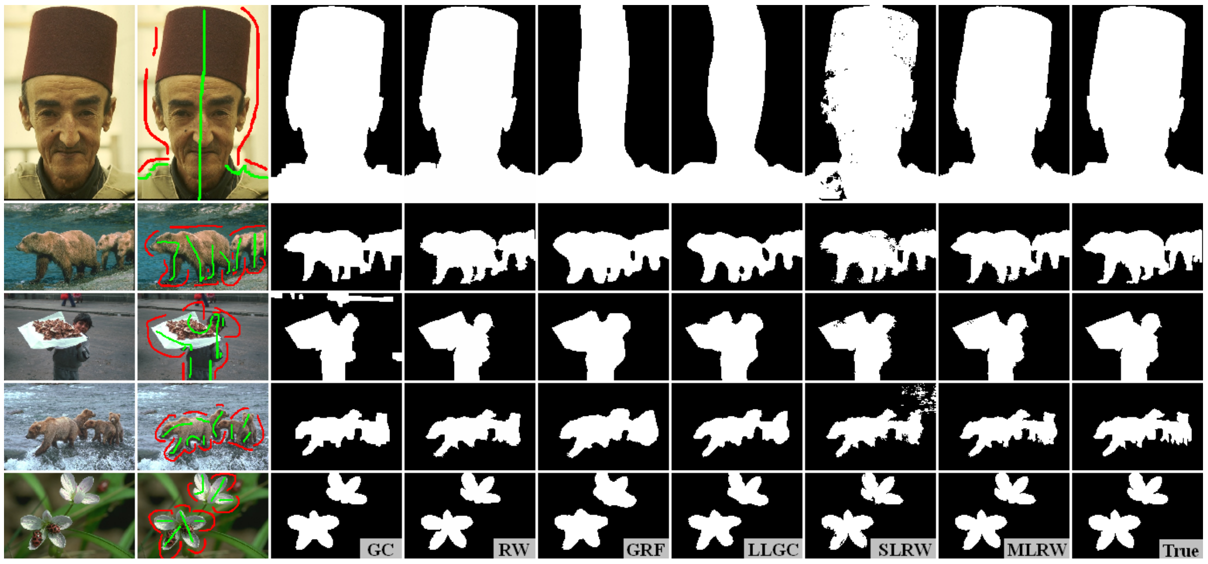


Fig. 12. Demo IV: Segmentation results of the images from Berkeley image database. The images are scaled for arrangement.

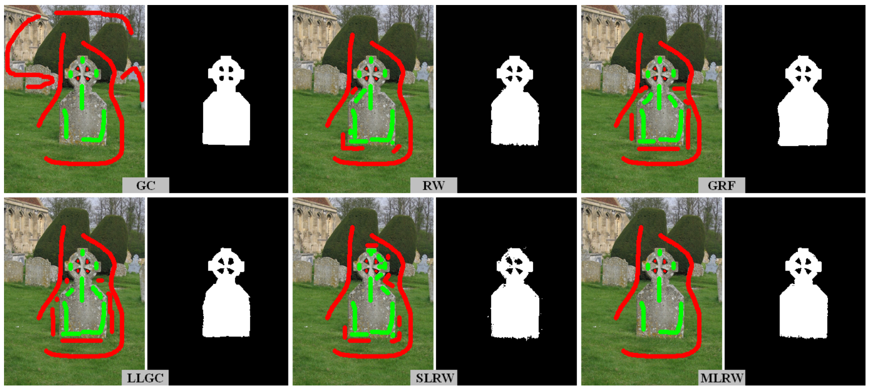


Fig. 13. Segmentation results obtained by GC, RW, GRF, LLGC, SLRW and MLRW, with different user-specified strokes. MLRW generates satisfactory segmentation with a small number of user specified strokes.

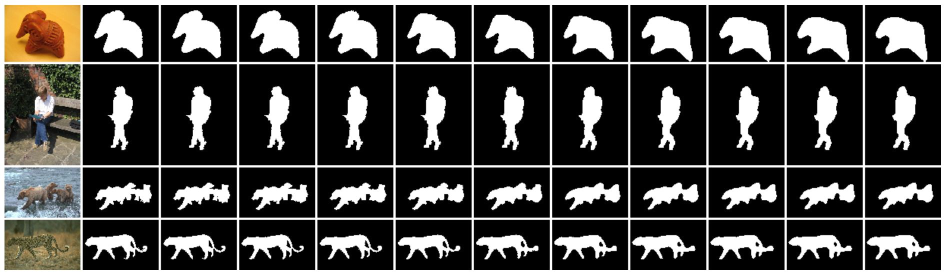


Fig. 14. Segmentation results obtained by MLRW with different α. From the second to the last columns are those with $\alpha = 0, 0.02, \cdots, 0.2$.

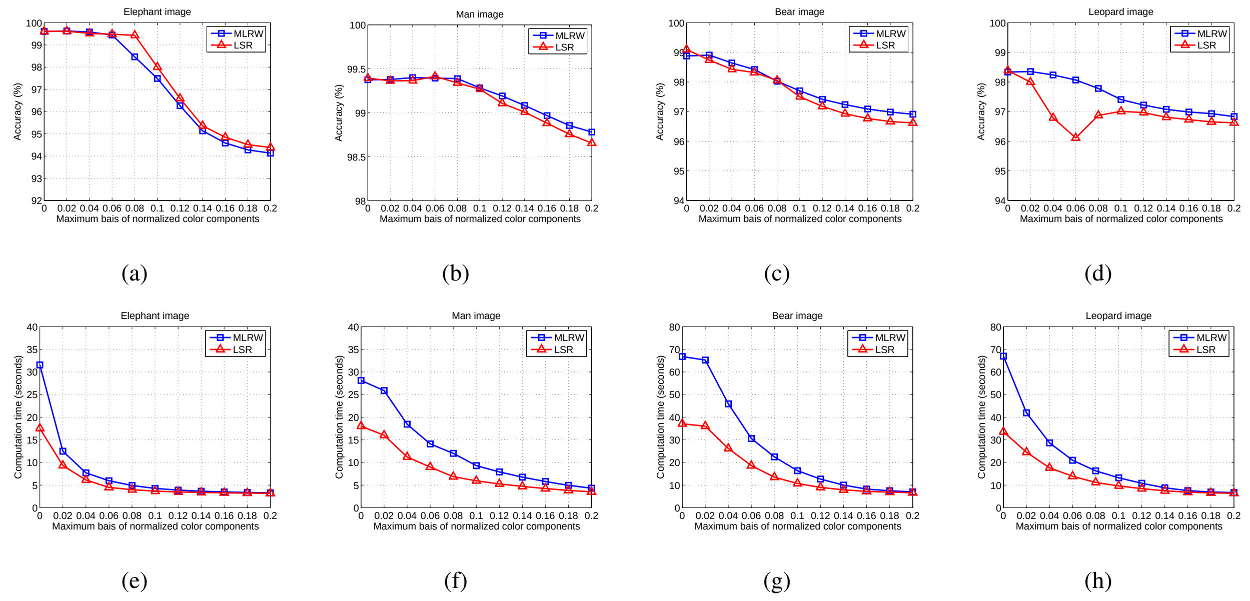


Fig. 15. The accuracy curves and computation time curves with different α.

119.0 seconds respectively, using Matlab 7.0 on a PC with 3.0 GHz CPU and 4.0G RAM (Note that, assembling matrix $\mathbf{M}$ 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, calculating 481×321 matrices $\{\mathbf{M}_i\}$ in (10) will need about 67.0 seconds. We see more than a 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.

TABLE I

THE SEGMENTATION ACCURACY (%) OF THE 20 IMAGES ORDERED IN FIG. 9, FIG. 10. FIG. 11 AND FIG. 12.

	GC	RW	GRF	LLGC	SLRW	MLRW
1	99.1237	95.5690	93.5651	91.6081	93.4961	**99.5638**
2	97.9479	98.6284	94.8750	93.8701	96.1372	**98.9700**
3	97.8086	99.1341	88.1380	89.1849	88.0169	**99.3411**
4	**99.7383**	99.6823	96.9245	94.3073	98.6719	99.6576
5	98.3460	97.7481	98.3102	96.2405	95.4303	**98.9827**
6	99.4252	96.3348	95.6978	96.9926	97.6400	**99.5719**
7	98.8904	99.2844	97.3319	98.6889	98.7526	**99.3748**
8	94.4543	98.8276	95.9512	95.3788	97.3201	**98.8276**
9	99.4336	99.4531	93.9714	93.1992	97.0104	**99.6081**
10	84.1156	99.2726	95.9304	97.4504	98.3704	**99.6711**
11	86.4399	99.4514	96.6704	98.2371	98.8439	**99.4754**
12	83.4476	97.2422	97.2371	97.1419	97.3316	**98.3349**
13	98.5648	95.5745	89.6937	89.4800	93.8673	**98.6781**
14	95.0337	99.0447	98.2947	98.9352	94.8491	**99.0939**
15	98.4916	98.6088	94.9016	96.1432	96.3757	**99.0959**
16	99.4229	**99.5745**	78.1044	80.1187	96.9638	99.4644
17	96.5525	97.0305	94.5603	94.4165	96.8491	**98.1114**
18	93.1393	99.3141	97.5033	98.2124	98.8692	**99.3679**
19	98.6503	97.8446	96.2338	97.3517	96.2546	**98.8782**
20	99.4087	99.2727	97.4689	98.3711	99.0531	**99.5045**

C. Speeding up MLRW

This subsection will introduce a method to reduce the calculations of matrices $\{\mathbf{M}_i\}$ in (10). Note that natural images usually contain small regions with the same pixel colors. That is, if the pixels in a 3×3 window have the same color, matrix $\mathbf{X}_{i,j}^T\mathbf{X}_{i,j}$ in (4) will be zero. In this case, all the elements in $\mathbf{w}_{i,j}$ will be equal to each other. As a result, $\mathbf{M}_i$ 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 $\mathbf{M}_i$. This condition

TABLE II

THE SEGMENTATION ACCURACY (%) OF THE 20 IMAGES, OBTAINED BY SR, LSR AND MLRW.

	1	2	3	4	5	6	7	8	9	10
SR	99.5664	97.7949	96.7031	99.5951	97.7774	99.5644	98.7793	98.3409	99.2396	99.2326
LSR	99.4870	99.0386	99.3294	99.6719	98.9835	99.6193	99.3941	99.0399	99.6087	99.6859
MLRW	99.5638	98.9700	99.3411	99.6576	98.9827	99.5719	99.3748	98.8276	99.6081	99.6711

	11	12	13	14	15	16	17	18	19	20
SR	97.4553	96.0214	98.6121	97.2429	98.7753	97.8102	95.4806	99.2526	99.0680	98.9586
LSR	99.4761	98.4383	98.7550	99.1416	99.1048	99.2564	98.0613	99.2260	99.1119	99.2276
MLRW	99.4754	98.3349	98.6781	99.0939	99.0959	99.4644	98.1114	99.3679	98.8782	99.5045

can be formulated as follows:

$$\max(\text{std}(\mathbf{r}), \text{std}(\mathbf{g}), \text{std}(\mathbf{b})) \leq \alpha, \tag{18}$$

where α is a given threshold, and std($\mathbf{r}$), std($\mathbf{g}$) and std($\mathbf{b}$) are the standard deviations of the normalized red, green and blue color components in the 3×3 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 α. 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 $\{\mathbf{M}_i\}$ is drastically reduced even with $\alpha = 0.02$. For other three images, when α 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.

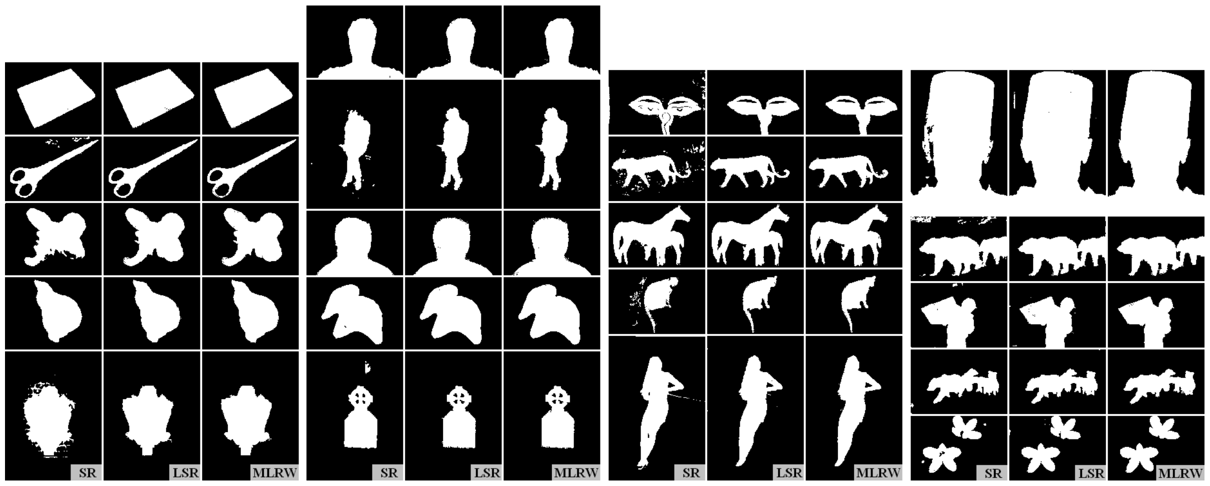


Fig. 16. Segmentation results of the images obtained by SR, LSR and MLRW, with the same user specified strokes illustrated in Fig. 9, Fig. 10, Fig. 11 and Fig. 12.

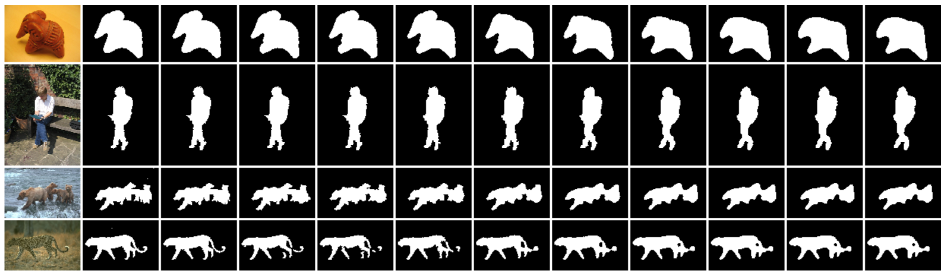


Fig. 17. Segmentation results obtained by LSR with different α. From the second to the last columns are those with $\alpha = 0, 0.02, \cdots, 0.2$.

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 3×3 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 481×321 such matrices will need about

TABLE III

COMPARISON BETWEEN SR, LSR AND MLRW.

	category of algorithm	local function used in each window	accuracy	speed	memory	speeding up under (18)
SR	discriminative learning	–	low	fast	low	–
LSR	transductive learning	nonlinear (one spline)	high	slow	high	not recommended
MLRW	transductive learning	linear (multiple linear reconstructions)	high	slow	high	recommended

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 3×3 *(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 α. 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 3×3 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 $\mathbf{M}_1$, and the constant matrix by $\mathbf{C}_1$. We further denote the Laplacian matrix in LSR by $\mathbf{M}_2$, and the constant Laplacian matrix by $\mathbf{C}_2$. Then, we use Frobenius norm to measure the difference between matrices. We have $||\mathbf{M}_1 - \mathbf{C}_1||_F = 5.5164$ and $||\mathbf{M}_2 - \mathbf{C}_2||_F = 1.7321 \times 10^4$. 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 3×3 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 3×3 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

$$\mathbf{A} = \begin{pmatrix} \lambda\mathbf{I} & \mathbf{U} \\ \mathbf{U}^T & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{13\times 13}, \tag{19}$$

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

The Laplacian matrix deduced by LSR is the 9×9 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 9×9 top-left sub-matrix of $\mathbf{A}^+$, namely, the Laplacian matrix $\mathbf{C}$ (the matrix $\mathbf{M}_i$ in Equation (11) in [24]), equals to

$$\mathbf{C} = (\mathbf{I} - \mathbf{e}\mathbf{e}^T/9)/\lambda. \tag{20}$$

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 $\mathbf{U} = \mathbf{e}\mathbf{u}^T$. Further let $\mathbf{T} = \mathbf{U}^T\mathbf{U} = \mathbf{u}\mathbf{e}^T\mathbf{e}\mathbf{u}^T = 9\mathbf{u}\mathbf{u}^T \in \mathbb{R}^{4\times 4}$. 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

$$\mathbf{T}^+ = \frac{1}{9(\mathbf{u}^T\mathbf{u})^2}\mathbf{u}\mathbf{u}^T \in \mathbb{R}^{4\times 4}. \tag{21}$$

Now we can prove the above theorem as follows:

Proof: According to matrix theory, formally we have

$$\mathbf{A}^{-1} = \begin{pmatrix} (\mathbf{I} - \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T)/\lambda & \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1} \\ (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T & -\lambda(\mathbf{U}^T\mathbf{U})^{-1} \end{pmatrix}. \tag{22}$$

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

$$\mathbf{A}^{+} = \begin{pmatrix} (\mathbf{I} - \mathbf{e}\mathbf{e}^T/9)/\lambda & \mathbf{e}\mathbf{u}^T/(9\mathbf{u}^T\mathbf{u}) \\ \mathbf{u}\mathbf{e}^T/(9\mathbf{u}^T\mathbf{u}) & -\lambda\mathbf{u}\mathbf{u}^T/(9(\mathbf{u}^T\mathbf{u})^2) \end{pmatrix}. \tag{23}$$

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

Based on (20), $\mathbf{C}$ is independent of color vector $\mathbf{p}$. 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

[1] W. A. Barrett and A. S. Cheney, “Object-based image editing,” in *SIGGRAPH*, San Antonio, USA, 2002, pp. 777–784.

[2] A. Blake, C. Rother, M. Brown, P. Perez, and P. Torr, “Interactive image segmentation using an adaptive gmmrf model,” in *European Conference on Computer Vision*, Prague, Czech, 2004, pp. 428–441.

[3] Y. Boykov and M. Jolly, “Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images,” in *International Conference on Computer Vision*, Vancouver, Canada, 2001, pp. 105–112.

[4] Y. Y. Chuang, B. Curless, and D. S. abd Richard Szeliski, “A bayesian approach to digital matting,” in *International Conference on Computer Vision and pattern recognition*, vol. 2, Hawaii, USA, 2001, pp. 264–271.

[5] M. Gleicher, “Image snapping,” in *SIGGRAPH*, Los Angeles, USA, 1995, pp. 183–190.

[6] L. Grady, “Random walks for image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1768–1783, 2006.

[7] A. Levin, D. Lischinski, and Y. Weiss, “A closed-form solution to natural image matting,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 228–242, 2008.

[8] S. Li, *Markov Random Field Modeling in Image Analysis*, 3rd ed. Berlin, Germany: Springer, 2009.

[9] Y. Li, J. Sun, C. Tang, and H. Shum, “Lazy snapping,” in *SIGGRAPH*, Los Angeles, USA, 2004, pp. 303–307.

[10] D. R. 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 *IEEE International Conference on Computer Vision*, Vancouver, Canada, 2001, pp. 416–425.

[11] E. Mortensen and W. Barrett, "Intelligent scissors for image composition," in *SIGGRAPH*, Los Angeles, USA, 1995, pp. 191–198.

[12] E. Mortensen and W. Barrett, "Toboggan-based intelligent scissors with a four-parameter edge model," in *International Conference on Computer Vision and pattern recognition*, Fort Collins, CO, USA, 1999, pp. 452–458.

[13] P. Perez, A. Blake, and M. Gangnet, "Jetstream: probabilistic contour extraction with particles," in *International Conference on Computer Vision*, vol. 2, Vancouver, Canada, 2001, pp. 524–531.

[14] A. Protiere and G. Sapiro, "Interactive image segmentation via adaptive weighted distances," *IEEE Transactions on Image Processing*, vol. 16, no. 4, pp. 1046–1052, 2008.

[15] L. Reese and W. Barrett, "Image editing with intelligent paint," in *Proceedings of Eurographics*, Saarbrucken, Germany, 2002, pp. 714–724.

[16] C. Rothera, V. Kolmogorov, and A. Blake, ""grabcut" —interactive foreground extraction using iterated graph cuts," in *SIGGRAPH*, Los Angeles, USA, 2004, pp. 309–314.

[17] S. Roweis and L. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, pp. 2323–2326, 2000.

[18] M. A. Ruzon and C. Tomasi, "Alpha estimation in natural images," in *International Conference on Computer Vision and Pattern Recognition*, vol. 1, Hilton Head, SC, USA, 2000, pp. 18–25.

[19] J. Sun, J. Jia, C. Tang, and H. Shum, "Poisson matting," in *SIGGRAPH*, Los Angeles, USA, 2004, pp. 315–321.

[20] K.-H. Tan and N. Ahuja, "Selecting objects with freehand sketches," in *International Conference on Computer Vision*, vol. 1, Vancouver, Canada, 2001, pp. 337–344.

[21] V. Vapnik, *Statistical Learning Theory*. New York, USA: Wiley, 1998.

[22] J. Wang and M. Cohen, "An iterative optimization approach for unified image segmentation and matting," in *International Conference on Computer Vision*, Beijing, China, 2005, pp. 936–943.

[23] S. Xiang, F. Nie, and C. Zhang, "Learning a mahalanobis distance metric for data clustering and classification," *Pattern Recognition*, vol. 41, no. 12, pp. 3600–3612, 2008.

[24] S. Xiang, F. Nie, and C. Zhang, "Semi-supervised classification via local spline regression," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 11, pp. 2039–2053, 2010.

[25] S. Xiang, F. Nie, C. Zhang, and C. Zhang, "Interactive natural image segmentation via spline regression," *IEEE Transactions on Image Processing*, vol. 18, no. 7, pp. 1623–1632, 2009.

[26] D. Zhao, "Formulating lle using alignment technique," *Pattern Recognition*, vol. 39, no. 11, pp. 2233–2235, 2006.

[27] Y. Zheng and C. Kambhamettu, "Learning based digital matting," in *International Conference on Computer Vision*, Kyoto, Japan, 2009, pp. 889–896.

[28] D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Scholkopf, "Learning with local and global consistency," in *Advances in Neural Information Processing Systems 16*, Vancouver, Canada, 2003, pp. 321–328.

[29] X. J. Zhu, Z. Ghahramani, and J. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of International Conference of Machine Learning*, Washington DC, USA, 2003, pp. 912–919.