

# Grouping Using Factor Graphs: an Approach for Finding Text with a Camera Phone

Huiying Shen and James Coughlan

Smith-Kettlewell Eye Research Institute  
San Francisco, CA 94115  
hshen@ski.org

**Abstract.** We introduce a new framework for feature grouping based on factor graphs, which are graphical models that encode interactions among arbitrary numbers of random variables. The ability of factor graphs to express interactions higher than pairwise order (the highest order encountered in most graphical models used in computer vision) is useful for modeling a variety of pattern recognition problems. In particular, we show how this property makes factor graphs a natural framework for performing grouping and segmentation, which we apply to the problem of finding text in natural scenes. We demonstrate an implementation of our factor graph-based algorithm for finding text on a Nokia camera phone, which is intended for eventual use in a camera phone system that finds and reads text (such as street signs) in natural environments for blind users.

## 1 Introduction

The ability to read street signs and other informational signs would be very useful to people who have visual impairments that make it difficult or impossible to find and read signs. A growing body of work in computer vision tackles the problem of finding text in natural scenes [1,2,3,4], a task that is especially challenging in highly cluttered environments; once text is located, well-established OCR (optical character recognition) techniques can be used to read it. So far almost all of this work on finding text has been implemented on standard personal (e.g. desktop or laptop) computers. While computers are continually improving in terms of power and portability, they are still too heavy, bulky and expensive to be convenient for most visually impaired users.

An attractive hardware alternative is the camera cell phone (or smart phone), which is lightweight, inexpensive, multi-purpose and nearly ubiquitous. Since most people already carry a cell phone, it has the added benefit of requiring *no additional device to purchase or carry*.

However, an important limitation of the camera phone is that it has substantially less processing power than a standard computer. The camera phone CPU is significantly slower than the kind found in desktop computers; in addition, the camera phone lacks a floating point processing unit (FPU), which means that floating point calculations – a mainstay of most computer vision algorithms – are particularly slow. Integer arithmetic is faster on the camera phone, but it is still up to an order of magnitude slower than on a standard computer.

The need for an algorithm to find text efficiently enough to run on a camera phone has motivated us to develop a new framework for simple, fast text segmentation. To this end we have adapted a graphical model-based framework originally developed for finding pedestrian crosswalks in traffic intersections [5] to the problem of finding text in natural scenes [6]. In this approach, we cast text detection as a problem of segmenting edge-based text features extracted from an image into figure or ground. The signature of a text region is an abundance of text features that are aligned in a fairly regular way. By contrast, text features occur more sparsely outside of text regions, and are spaced less regularly. The purpose of the graphical model framework is to exploit this pattern to segment all the text features in an image into figure or ground, corresponding to text or non-text regions, respectively. The graphical model achieves the desired behavior by expressing appropriate grouping criteria among the text features.

In this paper we describe a new framework for segmentation that is an outgrowth of our previous work, which provides for more expressive grouping criteria, and which is simpler and faster. The framework is based on factor graphs [7], which provide a convenient way of expressing interactions of any order in a graphical model. We have used this framework to develop a text-finding algorithm that relies almost entirely on integer arithmetic calculations, which enables an efficient camera phone implementation. Preliminary experiments demonstrate the ability of the algorithm to segment text regions in an image in several seconds on a camera phone (Nokia 6681, 220 MHz ARM CPU).

## 2 Past Work on Text Detection

A large body of work addresses the problem of detecting and reading printed text, but so far this problem is considered solved only in the domain of OCR (optical character recognition). This domain is limited to the analysis of high-resolution, high-contrast images of printed text with little background clutter. The broader challenge of detecting and reading text in highly cluttered scenes, such as indoor or outdoor scenes with informational signs, is much more difficult and is a topic of ongoing research. (We focus on the problem of segmentation in this paper, leaving the task of reading segmented text for future research.)

Many text segmentation algorithms employ deterministic, bottom-up processes for grouping text features into candidate text regions using features such as edges, color or texture [1,2,3,4]; a recent and comprehensive survey is found in [8]. Statistical methods have recently been developed, such as an Adaboost-based algorithm [9] that uses a cascade of filters trained from a labelled data set of natural scenes containing text.

We build on our recent work [6] casting text detection as a figure-ground segmentation problem represented using a probabilistic graphical model. We now propose to use a novel factor graph grouping technique that permits a more expressive graphical model to be used – specifically, one that allows *higher-order* interactions among several features, rather than being restricted to pairwise interactions (between pairs of features). The advantage of the factor graph grouping framework is that it allows grouping to be performed on very simple features that can be extracted rapidly from an image. The simple features can be grouped according to complex criteria using higher-order fac-

tors. As we will see later, the computational complexity that could arise from the use of higher-order factors is avoided because of the particular form of the factor graph.

Recent work related to ours [10,11] also uses a graphical model framework. Unlike our approach, the former work tackles text detection solely in documents, and the latter work uses color to initiate the segmentation and requires images in which individual letters are clearly visible. By contrast, we have designed our algorithm to process natural grayscale images with letters that may be poorly resolved (e.g. Fig. 5). This allows us to segment text in images in which the letters appear small and/or process the images at coarser scales (which decreases the amount of computation required for segmentation).

### 3 Grouping with Factors

Most methods devised for clustering or grouping data (such as normalized cuts [12] and graphical-model based typical cuts [13]) rely on affinities defined on *pairs* of data points to express the likelihood that two points should be grouped together. However, many clustering problems necessitate the use of higher-order affinities; for instance, the problem of grouping points on a 2-D plane into lines requires an affinity defined on triplets of points, since every pair of points is trivially collinear. Some recent work [14] has investigated hypergraph partitioning techniques for handling these higher-order affinities.

We propose that a particular form of graphical model known as a *factor graph* [7] provides a natural framework for grouping with higher-order affinities that results in simple and efficient grouping algorithms. Our framework is well suited to modeling object-specific figure-ground segmentation, which is how we cast the problem of text detection. It is inspired by object-specific figure-ground segmentation work by [15] and from work on clustering using graphical models [13].

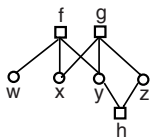
The factor graph provides a convenient way of representing interactions among arbitrary numbers of variables, generalizing the pairwise interactions often used in graphical models. An important property of factor graphs is that, as for all graphical models, rapid inference can be performed on them using a form of belief propagation (BP).

In the next subsection we introduce factor graphs and factor graph BP and demonstrate how they can be used to implement figure-ground segmentation. The application of figure-ground segmentation to text detection is described subsequently.

#### 3.1 Factor Graphs and Belief Propagation

Factor graphs [7] provide a convenient framework for representing graphical models in a way that shows interactions of any order in a visual format. Fig. 1 shows an example of a factor graph. Each square node represents a factor, or interaction, among one or more variables, depicted by circles, and the topology of the factor graph indicates how the joint distribution of all variables factors.

Belief propagation (BP) can be extended to factor graphs. Here we present a brief overview of factor graph BP, using notation similar to that of [7]. We note that factor graph BP is very similar to standard BP: messages are sent from one node to another, and each message is a function of the state of one node variable. However, a chief



**Fig. 1.** Factor graph. This graph represents a distribution on four variables  $w, x, y, z$  (drawn as circles) using three factors  $f, g, h$  (drawn as squares). Edges connect factors with the variables they influence. The joint distribution represented by this factor graph is  $P(w, x, y, z) = f(w, x, y)g(x, y, z)h(y, z)$ .

difference is that there are *two* types of messages in factor graph BP, those that are sent from variables to factors and those sent from factors to variables. We consider the max-product version of factor BP rather than the sum-product version because the former can be implemented very efficiently in the log domain using only addition and subtraction, without the need for multiplication. (We convert to the log domain by taking the log of the original max-product equations, and renaming the messages and factors to absorb the logs.) As we will see, this arithmetic simplification is key to our ability to implement factor BP efficiently on a camera phone CPU.

The max-product version of factor graph BP is expressed in the log domain according to the following update equations:

$$m_{x \rightarrow f}(x) \leftarrow \sum_{h \in n(x) \setminus \{f\}} m_{h \rightarrow x}(x) \quad (1)$$

where  $n(x)$  is the set of neighbors of  $x$ , and

$$m_{f \rightarrow x}(x) \leftarrow \max_{\sim \{x\}} \left( f(X) + \sum_{y \in n(f) \setminus \{x\}} m_{y \rightarrow f}(y) \right) \quad (2)$$

where  $X = n(f)$  is the set of arguments of function  $f$ , and  $\sim \{x\}$  denotes the set of all arguments of  $f$  except for  $x$ . (The sums in these two equations correspond to products before converting to the log domain.) Note that, for each factor  $f$  and neighboring variable  $x$ , updating all messages  $m_{f \rightarrow x}(x)$  has worst-case complexity  $O(|S|^m)$ , where  $m$  is the number of variables coupled by factor  $f$  and  $|S|$  is the number of allowed states of each of the  $m$  variables (assuming the state spaces are the same size for each variable). This is because Eq. 2 must be iterated for each value of  $x$  on the left-hand side, and for each value of  $x$  the max must be evaluated over the remaining  $m - 1$  variables  $\sim \{x\}$ .

Once the messages have converged to some value (which, in general, we can only hope happens after enough message updates), we can calculate the “belief function” for each node:

$$b(x) = \sum_{f \in n(x)} m_{f \rightarrow x}(x) \quad (3)$$

In sum-product BP, the belief is an estimate of the marginal probability of each node. In max-product BP, the belief is a function with a weaker property: the state that maximizes the belief of a node is an estimate of the node’s state in the most likely global configuration of states across the entire graphical model (i.e. the MAP estimate if the graphical model is interpreted as representing a posterior distribution).

### 3.2 Factor Graphs for Figure-Ground Segmentation

We now specialize our discussion of factor graphs to the problem of figure-ground segmentation. In this context, each node variable  $x_i$  in the factor graph is binary-valued:  $x_i = 1$  and  $x_i = 0$  represent figure and ground states, respectively. A factor of  $m$  variables  $f(x_1, \dots, x_m)$  expresses the likelihood of every possible assignment of states to all  $m$  variables, irrespective of the other variables in the factor graph. A *unitary* factor of one node variable  $f(x)$  enforces a prior bias towards figure or ground, independent of other nodes.

A simple form of this type of factor graph – a graphical model with only unitary factors and pairwise interactions – was used in our previous work on figure-ground segmentation [6]. Generalizing from that form, we now stipulate that each factor in our graph has a special form:  $f(x_1, \dots, x_m)$  is non-zero *only when*  $x_1 = \dots = x_m = 1$ , i.e. when all the variables are in figure states, and  $f(x_1, \dots, x_m) = 0$  otherwise. In other words, the factor can only assume two possible values: one value when its arguments (variables that it influences) are in the “figure” state, and zero otherwise.

We make a further non-negativity requirement that  $K_f \doteq f(x_1 = 1, \dots, x_m = 1) \geq f(x_1 = 0, \dots, x_m = 0) = 0$ . This additional requirement implies great computational savings: factor BP for this factor graph will converge in *only one iteration* of message updates. It is straightforward to show this by first initializing all messages  $m_{f \rightarrow x}$  and  $m_{x \rightarrow f}$  to zero and noticing that the first update will yield  $m_{f \rightarrow x}(0) = 0$  and  $m_{f \rightarrow x}(1) = K_f$ . (The messages from variables to factors all remain zero:  $m_{x \rightarrow f}(x) = 0$  for  $x = 0$  and  $x = 1$ .) Thanks to the non-negativity requirement, subsequent message updates will leave the message values unaltered (provided the messages are “normalized” after each iteration by uniformly shifting them by an appropriate amount – an operation that does not affect the outcome of BP).

The beliefs then have the following simple form:  $b_x(x = 1) = \sum_{f \in n(x)} K_f$  and  $b_x(x = 0) = 0$ . Since  $b_x(x = 0) = 0$  is fixed and only the difference  $b_x(x = 1) - b_x(x = 0)$  matters, we use a simplified notation to represent the beliefs:

$$B_x = \sum_{f \in n(x)} K_f \quad (4)$$

This result shows that the beliefs can be calculated without the need for any message updates! This computational savings comes at a price, however. First, the non-negativity requirement means that a unitary prior favoring ground over figure (e.g.  $f(x = 0) = 0, f(x = 1) < 0$ ) is not allowed. As a result, the state configuration that maximizes the sum of all the (non-negative) factors in the graph is simply  $x = 1$  for all node variables  $x$ , which is a degenerate result. A simple way to work around this problem is to omit all unitary factors but to assign each node variable  $x$  to figure only if its belief  $B_x$  is

sufficiently large. In this way, only nodes with sufficient support from other nodes will be assigned to figure, and the rest will be assigned to ground.

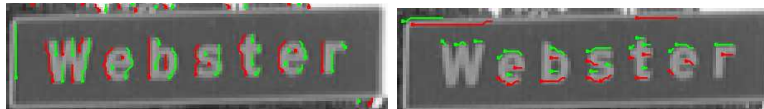
Second, the fact that BP converges in only one iteration means that information will not be propagated over long distances in the factor graph. Such propagation can be very useful for “filling in” an image region that has weak evidence with stronger evidence outside the region. However, the benefit of our simple factor graph is that it is easy to include *factors of arbitrarily high order*, whereas in general the computational complexity increases exponentially with the factor order (see the complexity analysis immediately following Eq. 2). Since information is still propagated within overlapping factors – which can be of very high order and thus encompass large regions of an image – this means that information can still be propagated over long distances.

## 4 Grouping Text Features

We have devised a bottom-up procedure for grouping edges into composite features that are signatures of regions containing text. The next subsection describes how these features are constructed, and the subsequent subsection explains how the features are grouped into factors.

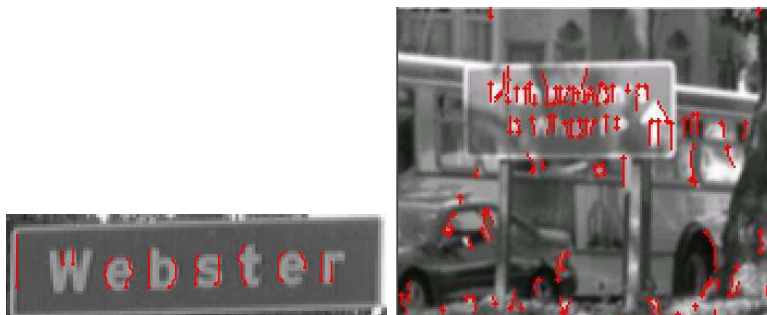
### 4.1 Constructing Features

Speed is a major consideration, so we used a very simple edge detector to provide the basic elements to be grouped. First, the image is blurred slightly, converted to grayscale and decimated to  $640 \times 480$  pixels or smaller. Two kinds of edges are detected, corresponding to local maxima or minima of the horizontal and vertical image intensity derivatives. The edges are grouped into line segments, which are approximately straight and fully connected sequences of edge pixels (with between 3 and 20 pixels, which sets an appropriate range of scales for text detection). There are two kinds of line segments, those that are oriented (approximately) vertically and those that are oriented (approximately) horizontally. Vertical segments that are sufficiently close together and have opposite polarity are grouped into “weakly matched vertical edges”, shown in Fig. 2(a). “Weakly matched horizontal edges” are determined in a similar way (see Fig. 2(b)). As the figure suggests, weakly matched edges are features designed to be prevalent along the borders of letter strokes.



**Fig. 2.** Edge features used to construct text features shown on cropped image of street sign. Left, weakly matched vertical edge segments. Right, weakly matched horizontal edge segments. Edges shown in red and green to indicate opposite polarities.

Next we prune the set of weakly matched vertical segments to obtain our final features, “anchored verticals” (see Fig. 3). An anchored vertical feature is a weakly matched vertical segment whose topmost or bottommost pixel lies sufficiently close to the leftmost or rightmost pixel of a weakly matched horizontal segment. By “sufficiently close” pixels we mean that they are either identical or one pixel is one of the eight nearest neighbors of the other.



**Fig. 3.** Anchored verticals. Top, anchored verticals shown for image in previous figure. Bottom, shown for an image. Note density and regularity of anchored verticals in text region, where bottoms and tops tend to be at the same level. By contrast, anchored verticals are scattered sparsely and irregularly throughout rest of the image.

As Fig. 3 shows, anchored verticals have a distribution on text regions that is significantly different from the distribution outside of text regions. Anchored verticals are distributed densely on text regions, and their bottoms and tops tend to be aligned to the same level. By contrast, outside of text regions, anchored verticals are distributed more sparsely and irregularly. We will exploit this differential distribution of anchored verticals to segment out text regions in an image.

## 4.2 Building Factors

Having constructed a set of useful anchored vertical features that have a distinctive signature in text regions, we now proceed to construct a factor graph based on these features. In the factor graph, each anchored vertical is a variable node. Factor nodes are defined as groups of anchored verticals that may plausibly belong to one text region. As we have shown, our particular factor graph has the advantage that the results of BP can be calculated very simply without any iterative message updating. However, the trade-off is that the search for building suitable factor nodes is computationally intensive, since many of these factors are high-order (i.e. they bind many anchored verticals).

The search for factors is conducted by means of a sliding window of size 5 pixel rows by 30 pixel columns. Rather than having to slide the window pixel by pixel across each column and row of the image, the left side of the window is aligned to the top or bottom of each node (anchored vertical), one after the other. For each node the window

is aligned to (which we call the “reference” node), all node tops or bottoms to its right are found that lie in the window. Two kinds of factors are constructed, one linking the tops of the nodes and the other linking the bottoms. For simplicity we will describe the case of linking node tops; the same procedure is followed for linking node bottoms.

For each candidate factor  $f$ , we calculate a measure of alignment with respect to a horizontal line at the level of the top of the reference node. This alignment measure is the maximum of the absolute value of the vertical pixel distance between each node top and the horizontal line, where the maximum is taken over all nodes in the window. To accommodate text that may be slightly off-horizontal, we also calculate similar alignment measures for two other lines, one with a slope of  $1/10$  and the other with slope  $-1/10$ . The final error measure  $E_f$  corresponding to factor  $f$  equals the minimum of the three alignment errors, divided by the number of nodes in the sliding window.

Empirically we find that error measures  $E_f$  of 5 or more correspond to very weak factors, so we discard any factor  $f$  for which  $E_f \geq 5$ . Otherwise, we define  $K_f = 5 - E_f$ , so that a lower error equates to a stronger factor.

One of the strengths of our max-product factor BP approach is that almost all calculations can be done in simple integer arithmetic, which greatly speeds up the algorithm on a camera phone CPU (which lacks a floating-point unit). Rather than using floating-point to calculate and represent the error measure  $E_f$  (which is defined above using division), we rescale it by a factor of 100 into a suitable range of integers. Obviously, this rescaling is implicit in all subsequent calculations.

## 5 Experimental Results

We implemented our algorithm in Symbian C++ on a Nokia 6681 camera phone. Although the built-in camera has a resolution of approximately 1 megapixel, our algorithm decimated the images to  $640 \times 480$  or smaller to speed execution.



**Fig. 4.** Experimental results. Left, text segmentation from Fig. 3(b) demonstrating robustness to non-uniform lighting. Right, result demonstrating performance in high clutter.

Fig.'s 4-5 show results of the algorithm for pictures taken by the camera phone of local street scenes. The algorithm took several seconds per image to execute. Note the algorithm's ability to handle considerable amounts of scene clutter. The algorithm's robustness to non-uniform lighting conditions is shown in Fig. 4. However, false positives still occur, especially in image regions characterized by periodic structures such as windows or fences that resemble text at a local scale (Fig. 5).



Fig. 5. More experimental results.

## 6 Discussion

We have demonstrated the feasibility of a novel grouping framework, based on factor graphs, which we have applied to the problem of segmenting text in natural scenes. The algorithm is simple and fast enough to implement on a camera phone. Future work will focus on improving the false positive and false negative rates, which may be accomplished in part by learning the potentials [13] from training samples of manually segmented text. In order for our algorithm to function as part of a practical system for finding and reading text, we will also have to use the text features output by it to determine appropriate bounding boxes to enclose the text, and use OCR to actually read the text. We note that the OCR stage will have the benefit of discarding some false positives which cannot be ruled out by our algorithm alone.

## 7 Acknowledgment

The authors would like to acknowledge support from NIH grant NIH EY015187-01A2, and JMC also acknowledges support from NIH grant 1R01EY013875 for preliminary work on computer vision-based wayfinding systems.

## References

1. Wu, V., Manmatha, R., Riseman, E.M.: Finding text in images. In: ACM DL. (1997) 3–12 [1](#), [2](#)
2. Jain, A., Tu, B.: Automatic text localization in images and video frames. *Pattern Recognition* **31**(12) (1998) 2055–2076 [1](#), [2](#)
3. Li, H., Doermann, D., Kia, O.: Automatic text detection and tracking in digital videos. *IEEE Transactions on Image Processing* **9**(1) (2000) 147–156 [1](#), [2](#)
4. Gao, J., Yang, J.: An adaptive algorithm for text detection from natural scenes (2001) [1](#), [2](#)
5. Coughlan, J., Shen, H.: A fast algorithm for finding crosswalks using figure-ground segmentation. In: Proc. 2nd Workshop on Applications of Computer Vision, in conjunction with ECCV 2006. (2006) [2](#)
6. Shen, H., Coughlan, J.: Finding text in natural scenes by figure-ground segmentation. In: ICPR (4). (2006) 113–118 [2](#), [5](#)
7. Kschischang, Frey, Loeliger: Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory* **47** (2001) [2](#), [3](#)
8. Liang, J., Doermann, D., Li, H.: Camera-based analysis of text and documents: a survey. *International Journal on Document Analysis and Recognition* **7** (2005) 84–104 [2](#)
9. Chen, X., Yuille, A.L.: Detecting and reading text in natural scenes. In: Proc. Computer Vision and Pattern Recognition 2004. (2004) [2](#)
10. Zheng, Y., Li, H., Doermann, D.: Text identification in noisy document images using markov random field. In: International Conference on Document Analysis and Recognition. (2003) [3](#)
11. Zhang, D., Chang, S.: Learning to detect scene text using a higher-order mrf with belief propagation. In: Computer Vision and Pattern Recognition (CVPR). (2004) [3](#)
12. Shi, J., Malik, J.: Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**(8) (2000) 888–905 [3](#)
13. Shental, N., Zomet, A., Hertz, T., Weiss, Y.: Pairwise clustering and graphical models. In: NIPS. (2003) [3](#), [9](#)
14. Agarwal, S., Lim, J., Zelnik-Manor, L., Perona, P., Kriegman, D., Belongie, S.: Beyond pairwise clustering. In: Proceedings of Computer Vision and Pattern Recognition (CVPR 2005). Volume 2. (2005) 838–845 [3](#)
15. Yu, S.X., Shi, J.: Object-specific figure-ground segregation. In: Computer Vision and Pattern Recognition (CVPR). (2003) [3](#)