Fast Generalized Belief Propagation for MAP Estimation on 2D and 3D Grid-Like Markov Random Fields

Kersten Petersen, Janis Fehr, and Hans Burkhardt

Albert-Ludwigs-Universität Freiburg, Institut für Informatik,
Lehrstuhl für Mustererkennung und Bildverarbeitung,
Georges-Koehler-Allee Geb. 052, 79110 Freiburg, Deutschland
petersep@informatik.uni-freiburg.de

Abstract. In this paper, we present two novel speed-up techniques for deterministic inference on Markov random fields (MRF) via generalized belief propagation (GBP). Both methods require the MRF to have a grid-like graph structure, as it is generally encountered in 2D and 3D image processing applications, e.g. in image filtering, restoration or segmentation. First, we propose a caching method that significantly reduces the number of multiplications during GBP inference. And second, we introduce a speed-up for computing the MAP estimate of GBP cluster messages by presorting its factors and limiting the number of possible combinations. Experimental results suggest that the first technique improves the GBP complexity by roughly factor 10, whereas the acceleration for the second technique is linear in the number of possible labels. Both techniques can be used simultaneously.

1 Introduction

Markov random fields [1] have become popular as probabilistic graphical models for representing images in image processing and pattern recognition applications, such as image filtering, restoration or segmentation. While an MRF provides a sound theoretical model for a wide range of problems and is easy to implement, inference on MRFs is still an issue. It is an NP hard problem [2] to explore all possible pixel combinations, and thus we have to resort to approximate inference algorithms, such as Monte Carlo chain methods (MCMC), or message passing algorithms. For a long time MCMC methods have been the common choice for inferring on MRFs, although they are non-deterministic and converge very slowly. But since the proclamation of message passing algorithms like Pearl's belief propagation (BP), they are no longer state-of-the-art. The BP algorithm has the advantage of being deterministic, fast and precise on many MRFs, especially if they are tree-structured [3]. However, BP proves to be inaccurate and unstable on graphs with many cycles, preventing its application to many image processing problems that are based on grid-like graphs. A more refined variant that generalizes the idea of the BP algorithm has been introduced by Yedidia

G. Rigoll (Ed.): DAGM 2008, LNCS 5096, pp. 41–50, 2008.

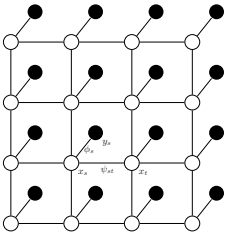

Fig. 1. A Markov random field with pairwise potential functions

et al. [4]. It is called *generalized belief propagation* and is far more stable and accurate. Yet, it is also computationally expensive, which limits its use to small graphs. In this paper, we present two techniques that accelerate the GBP algorithm on grid-like MRFs making GBP more suitable for 2D and 3D image processing tasks.

Related Work. In the literature, we have spotted only two papers, [5] and [6], that focus on speed-up techniques for the GBP algorithm, and both are guided by the same idea: Several edge potentials (see section 2) in an MRF can be divided into *compatible* pairs of labels whose values are label-dependent and *incompatible* pairs of labels that all have the same value. As the number of compatible pairs of labels n_c is usually much smaller than the number of incompatible labels n_i, we gain a speed-up of n_c/n_i by not computing redundant incompatible labels. Shental et al. [5] suggests this approach for the Ising model, and Kumar et al. [6] for the more general *robust truncated model*, comprising the piecewise constant prior and the piecewise smooth prior [7]. Thus, both techniques require a beneficial structure of the edge potentials for sparing redundant label configurations. In contrast, both our techniques accelerate the GBP algorithm on grid-like MRFs with arbitrary potential functions.

2 Belief Propagation and Its Generalization

The *belief propagation (BP)* algorithm is a message passing method that iteratively computes the marginal probabilities of a *Markov random field (MRF)*. An MRF is defined as a random vector $X = (X_s)_{s \in S}$ on the probability space $(\mathbf{X}, P)$ with respect to a neighborhood system ∂ such that P is strictly positive and fulfills the local Markov property. In this paper, we concentrate on *pairwise* MRFs, i.e. $s \in \partial\{t\}$ if and only if $t \in \partial\{s\}$ for two sites $s, t \in S$. For a given *labeling problem*, we have to divide the MRF nodes into two sets: The *observed* image nodes are denoted by y_s, whereas the *hidden* label nodes of the image are referred to as x_s. Figure 1 shows a pairwise MRF where y_s is depicted by a filled circle and x_s is indicated by an empty circle. Connections between a filled and an empty circle are weighted by *node potentials* to encode the similarity between the observed image and the hidden labeling. Connections among empty

circles correspond to *edge potentials* which capture the similarity of neighboring hidden nodes, or more general prior knowledge about the image. According to the Hammersley-Clifford theorem the joint probability function of this MRF is given by

$$P(x) = \frac{1}{Z} \prod_{s} \phi_s(x_s) \prod_{st} \psi_{st}(x_s, x_t) \tag{1}$$

where Z denotes a partition function, $\phi_s(x_s)$ a node potential, and $\psi_{st}(x_s, x_t)$ an edge potential. Note that we use shorthand notation $\phi_s(x_s)$ for $\phi_s(x_s, y_s)$, since the observed image can be regarded as fixed.

In the BP algorithm [3], *messages* are iteratively passed along the edges of the hidden image until their rate of change falls below a pre-set threshold. A message $m_{st}(x_t)$ from node s to node t is a one-dimensional vector that propagates the likeliest label probabilities for t from the view of s. Once the message values have converged, we can compute the *beliefs* for the hidden nodes, i.e. approximations for the marginal probabilities of the MRF.

The BP algorithm features many attractive properties, such as exact estimates in trees or fast execution time. On many loopy graphs, however, it shows poor convergence and delivers inaccurate estimates. The source of error is the circular message flow that distorts good message approximations by self-dependent probability values. Yedidia et al. [4] propose a generalization of the BP algorithm called *generalized belief propagation (GBP)* which alleviates this undesired behavior. The basic idea is to compute more informative messages between groups of nodes in addition to messages between single nodes. We obtain an algorithm that demonstrates improved convergence behavior and delivers accurate approximations to the exact estimates. It no longer tends to the stationary points of the Bethe energy but is proven to approximate the fixed points of the more precise Kikuchi energy [3]. The GBP algorithm can be formulated in different ways [8]; in this paper we refer to the *parent-to-child* variant in *max-product* form for calculating the MAP estimate on grid graphs. For two-dimensional grid graphs, we obtain the following formulas from [3] (see figures 2 and 3). The formula for computing *single-node beliefs* is given by

$$b_s(x_s) = k\phi_s m_{as} m_{bs} m_{cs} m_{ds} \tag{2}$$

where we use shorthand notation $m_{as} \equiv m_{as}(x_s)$ and $\phi_s = \phi_s(x_s)$. The variables a, b, c and d denote the neighbors of s. If we compute the formula at the border of the grid and a neighbor lies outside the grid, we can neglect corresponding factors or set them to 1. The message update rule for *edge messages*, i.e. messages between two single nodes, evaluates to

$$m_{su}(x_u) = \max_{x_s} \left(\phi_s \psi_{su} m_{as} m_{bs} m_{cs} m_{bdsu} m_{cesu}\right) \tag{3}$$

where we abbreviate $\psi_{sx} = \psi_{sx}(x_s, x_u)$. The message update rule for *cluster messages*, i.e. messages between two pairs of nodes, unfolds as

$$m_{stuv}(x_u, x_v) = \frac{\max_{x_s x_t} \left(\phi_s \phi_t \psi_{st} \psi_{su} \psi_{tv} m_{as} m_{cs} m_{bt} m_{dt} m_{abst} m_{cesu} m_{dftv}\right)}{m_{su} m_{tv}} \tag{4}$$

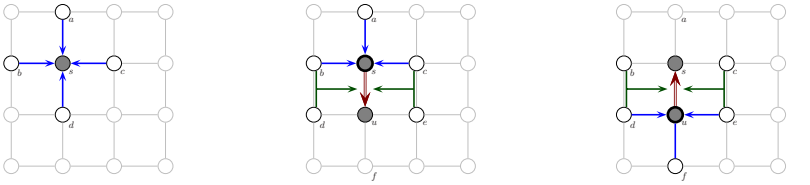

Fig. 2. LEFT: A diagram of the messages that influence the single-node belief at site s in a two-dimensional grid. CENTER and RIGHT: All edge messages (double-lined arrows) that are contained in the same two-node belief region $R = \{s, u\}$ (gray nodes). Note that the cluster messages from edges to edges are identical in both figures.

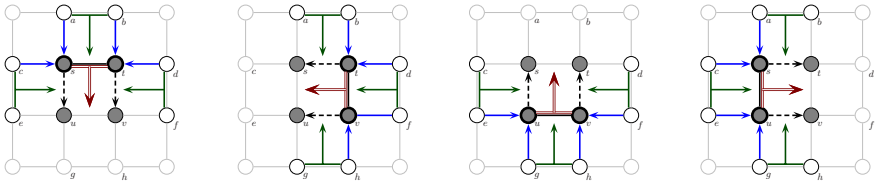

Fig. 3. A diagram of all cluster messages (double-lined arrows) that are contained in the same four-node belief region $R = \{s, t, u, v\}$ (gray nodes). Solid (blue) edges on the grid lines stand for edge messages in the nominator, whereas dashed edges are those in the denominator of the corresponding message update rule. (Green) messages in the centre of grid cells denote cluster messages that influence the value of the (double-lined) cluster message. We can observe that the same messages appear within several figures.

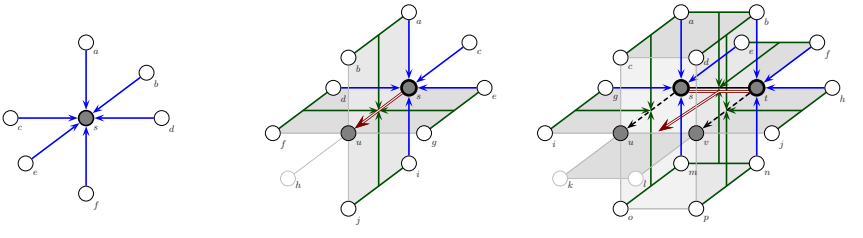

Fig. 4. LEFT: A diagram of the messages that influence the single-node belief at site s in a three dimensional grid. CENTER: The messages that influence the edge message from site s to site u. RIGHT: All messages that influence the (double-lined) cluster message from edge st to edge uv.

On three dimensional grid-graphs, the formulas for the GBP algorithm evaluate to (see figure 4):

$$b_s(x_s) = k\phi_s m_{as} m_{bs} m_{cs} m_{ds} m_{es} m_{fs} \tag{5}$$

$$m_{su}(x_u) = \max_{x_s} \left(\phi_s \psi_{su} m_{as} m_{cs} m_{ds} m_{es} m_{is} m_{absu} m_{dfsu} m_{egsu} m_{ijsu}\right) \tag{6}$$

$$m_{stuv}(x_u, x_v) = (m_{su}m_{tv})^{-1} \max_{x_s x_t} (\phi_s \phi_t \psi_{st} \psi_{su} \psi_{tv} M_1 M_2) \tag{7}$$

where

$$M_1 = m_{as}m_{es}m_{gs}m_{ms}m_{bt}m_{ft}m_{ht}m_{nt}$$

$$M_2 = m_{abst}m_{efst}m_{mnst}m_{acsu}m_{gisu}m_{mosu}m_{bdtv}m_{hjtv}m_{nptv}$$

3 Caching and Multiplication

Analyzing the messages that are computed within the same two- or four-node region, we notice that some messages appear repeatedly. As shown in figure 3, each cluster message computation involves four of the eight surrounding edge messages and three of the four surrounding cluster messages. Remarkably, the selection of the messages is not arbitrary but follows a simple pattern. In a cluster message m_{stuv} for instance, where s and t are its source nodes, and u and v are its target nodes, node potentials ϕ are only defined for the source nodes, while edge potentials ψ necessitate at least one involved node is a source node. Similarly, incoming edge messages lead from outside the basic cluster to a source node of m_{stuv}, while incoming cluster messages demand that at least one of its source nodes has to be a source node of m_{stuv}.

Also in edge messages, source nodes depend on data potentials and incoming edge messages, whereas pairwise node regions rely on edge potentials and incoming cluster messages. We subsume related factors of the message update rules into *cache products* and benefit in two ways:

1. *Caching*: Some cache products appear in several message update rules. We gain a speed-up if we pre-compute (cache) and use them for multiple message computations (see figure 2).
2. *Multiplication Order*: The multiplication order of the potentials and incoming messages plays a vital role. Node potentials and edge messages are represented by k-dimensional *vectors*, whereas edge potentials and cluster messages correspond to $k \times k$ *matrices*. Cache products comprise factors of either vector or matrix form which means that we need less computations than in the original formula where vectors and matrices are interleaved.

3.1 Caching and Multiplication in 2D

Edge Messages in 2D. If we subsume all edge message factors that depend on the same source node s into a cache product P_s, we obtain

$$P_s = \phi_s m_{as} m_{bs} m_{cs}, \tag{8}$$

whereas edge message factors that depend on two nodes s and u can be summarized as a cache product P_{su}

$$P_{su} = \psi_{su} m_{bdsu} m_{cesu}. \tag{9}$$

Combining both cache products, we can rewrite (3) as

$$m_{su}(x_u) = \max_{x_s} P_s P_{su}. \tag{10}$$

Cluster Messages in 2D. In analogy to the case of edge messages, we can define cache products within the message update rule for cluster messages. If we for instance compute the message update rule for the first cluster message of figure 3, the required cache products for source nodes are given by

$$P_s = \phi_s m_{as} m_{cs}, \qquad P_t = \phi_t m_{bt} m_{dt}. \tag{11}$$

and the cache products for pairs of nodes can be written as

$$P_{st} = \psi_{st} m_{abst}, \qquad P_{su} = \psi_{su} m_{cesu}, \qquad P_{tv} = \psi_{tv} m_{dftv}. \tag{12}$$

Substituting these expressions into (4), we obtain

$$m_{stuv}(x_u, x_v) = (m_{su} m_{tv})^{-1} \max_{x_s, x_t} P_s P_t P_{st} P_{su} P_{tv}. \tag{13}$$

3.2 Caching and Multiplication in 3D

We can extend the caching and multiplication technique to three-dimensional grids with a six-connected neighborhood system. The only difference is that the products of nodes and edges involve more terms than in the two-dimensional case, thereby increasing the speed-up.

Edge Messages in 3D. The cache product over the source variable of edge messages is computed by

$$P_s = \phi_s m_{as} m_{cs} m_{ds} m_{es} m_{is} \tag{14}$$

and the corresponding product over the pairs of nodes is described by

$$P_{su} = \psi_{su} m_{absu} m_{dfsu} m_{egsu} m_{ijsu} \tag{15}$$

Using these definitions of the cache products, (6) takes the same form as in the 2D case (see formula (10)).

Cluster Messages in 3D. For cluster messages, we define the cache products for the source nodes as

$$P_s = \phi_s m_{as} m_{es} m_{gs} m_{ms} \tag{16}$$

and the cache products on pairs of nodes as

$$P_{st} = \psi_{st} m_{abst} m_{efst} m_{mnst}. \tag{17}$$

The explicit formula (7) transforms to the same formula as in the 2D case (see formula (13)).

4 Accelerating MAP Estimation

According to (13) the GBP algorithm grows with the power of four in the number of labels, as a cluster message computation involves the traversal of all label combinations of x_s and x_t for each combination of x_u and x_v. Compared to edge messages that require quadratic computation time (see (10)), the update rule for cluster messages consumes most of the time for inference problems with multiple labels. In this section, we therefore pursue the question if it is necessary to explore all possible label combinations of x_s and x_t for determining the maximum.
In the spirit of [9] [6] [10], we sort the terms of (13) by *source variables* x_s and x_t, yielding

$$m_{stuv}(x_u, x_v) = (m_{su} m_{tv})^{-1} \max_{x_s, x_t} (P_{st} M_{su} M_{tv}) \tag{18}$$

where we define

$$M_{su} = P_s P_{su}, \qquad M_{tv} = P_t P_{tv}.$$

We observe that the maximum message value is likely to consist of relatively large factors P_{st}, M_{su} and M_{tv}. Thus, for each combination of x_u and x_v the basic idea is to start at the maximum values of M_{su} and M_{tv} in the respective columns and then systematically decrease the factors until the product of both entries and the corresponding value in P_{st} is assured to be maximal. Thus, we have to answer two questions: (1) How do we traverse the label combinations for x_s and x_t such that the product of M_{su} and M_{tv} monotonically decreases? (2) Under which conditions can we terminate the search for the maximum?

Traversal Order. We have to proceed for each combination of x_u and x_v in separation. Assume we set $x_u = u$ and $x_v = v$. Then we obtain the maximum product of the first two factors in (18) by determining the maximum entry s_{max} in the u-th column of M_{su} and the maximum entry t_{max} in the v-th column of M_{tv}. We multiply this product with the entry at position (s_{max}, t_{max}) in P_{st} and store the result as the first *temporary* maximum product value r_{max}. Suppose that the entry (s_{max}, t_{max}) is the i_{left} biggest value of all entries in P_{st}. Then all combinations of x_s and x_t, whose entry in P_{st} is smaller than the i_{left} biggest value of P_{st}, are not eligible to be the *final* maximum product value. For this reason we can save time by solely computing the products for combinations of x_s and x_t with a bigger value than the i_{left} biggest value of P_{st}.

Unfortunately, our speed-up is relatively small if i_{left} is large. For decreasing i_{left}, we examime which label combination of x_s and x_t leads to the next biggest product of M_{su} and M_{tv}. We *sort* M_{su} and M_{tv} column by column and refer to them as S_{su} and S_{tv}. Then s_{max} and t_{max} correspond to the positions $(1, u)$ in S_{su} and $(1, v)$ in S_{tv}, and the candidates for the next biggest combination have to be either $(1, u)$ in S_{su} and $(2, v)$ in S_{tv} or $(2, u)$ in S_{su} and $(1, v)$ in S_{tv}. We compute both products and take the bigger one. In general, the set of candidates for the next biggest product value of M_{su} and M_{tv} constitutes from label combinations for x_s and x_t that are adjacent to the already visited ones. Compare figure 5 where s_S and t_S refer to the row positions in S_{su} and S_{tv}.

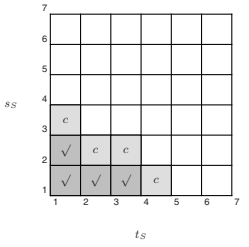

Fig. 5. A graphical depiction of the candidates for the next combination of s_S and s_T. Visited combinations are marked with a tick on dark gray background. The light gray fields with a c denote possible candidates for the maximal unvisited combination. White fields are unvisited and are not eligible as the next possible combination.

Thus, we gradually determine the next biggest products of M_{su} and M_{tv} and multiply it with the corresponding entry of s and t in P_{st}. We compare the result with the temporary maximum product value r_{max} and replace it if the new value is bigger. In this case we also update i_{left}. This pattern is repeated until i_{left} is considerably small, i.e. smaller than a pre-set threshold β. Once i_{left} falls below β, we can trigger the traversal of less than i_{left} combinations whose entries in P_{st} are bigger than the entry of the current maximum product value $(s_{\max}, t_{\max})$.

Termination Conditions. We have found the maximum product if any of the following two conditions is satisfied:

1. We have visited all entries in P_{st} that are bigger than the entry at position (s_{max}, t_{max}) in P_{st}.
2. The product of M_{su} and M_{tv}, multiplied with the maximal unvisited entry in P_{st}, is smaller than r_{max}.

5 Experiments

In our experiments on two-dimensional images, the caching and multiplication technique improves on the standard implementation by roughly factor 10. Several optimizations contribute to this result. First, we gain most of the speed-up by exploiting cache products that inherently use a beneficial multiplication order. Second, compared to the standard form of the message update rule in [8], the direct evaluation of the message update rules avoids the initial recursive traversal of the region graph for determining all incoming messages in the formulas. And finally, we do not have to evaluate the variable order of factors before multiplication. An additional benefit of the caching and multiplication technique is that we can reduce the storage costs, since we do not have to store references from a message to its dependent messages. On three-dimensional images, we can expect even higher speed-ups, as the cache products consist of more factors.

♯ Labels	Standard [s]	MAP accelerated [s]	Speed-up
8	0.03	0.05	0.61
16	0.15	0.15	1.03
32	1.21	0.59	2.05
64	15.43	2.92	5.28
128	243.03	19.29	12.60
256	4845.19	183.36	26.42

Fig. 6. The average iteration time of the standard and the accelerated MAP implementation for various label sizes

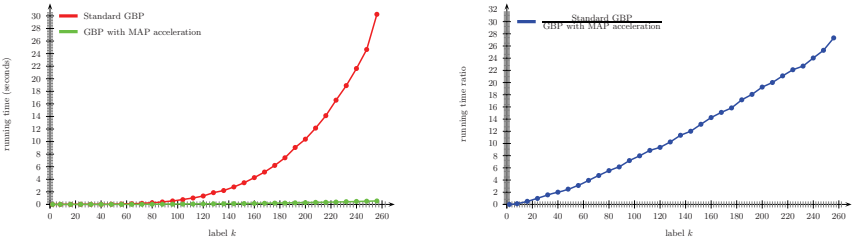

Fig. 7. LEFT: Running time as a function of the number of labels k. RIGHT: Ratio of standard GBP to MAP accelerated GBP as a function of k.

We estimate the effect of the accelerated MAP computation with two experiments[1]: First, we measure the speed-up for different numbers of labels on a 8×8 grid-like MRF, where we use the Potts model[2] for the edge potentials. In figure 6, we contrast how the average running time per iteration varies between the standard implementation and our technique. Note that the running time in the first iterations is often much higher than in later iterations. The reason could be that the message values contain more candidate maxima, which have to be evaluated in our optimized algorithm. After several iterations edge messages seem to attribute relatively high probabilities to a small number of labels.

And second, we demonstrate the effectiveness of our technique for other potentials by computing the average running time of 100 random edge potentials for various sizes of k (see figure 7). Opposed to the first experiment, we do not evaluate the computation cost of the whole GBP algorithm, but solely the elapsed time for computing the MAP estimate for cluster messages. Figure 7 shows the ratio of these running times. We can observe that the speed-up of our implementation grows almost linear.

While we always benefit from using cache products, we recommend to use the second optimization technique solely for label sizes bigger than 15. For smaller

[1] All experiments are conducted on a 3 GHz CPU with 2 GB RAM. The caching and multiplication technique is enabled.

[2] The *Potts model*[1] is the generalization of the Ising model and is used for various image processing tasks.

labels sizes, the computational overhead per combination outweighs the reduced number of visited combinations.

6 Summary and Conclusion

In this paper, we have presented two novel techniques for reducing the computational complexity of the GBP algorithm on grid-like graphs without losing any precision. Both techniques are independent on the values of the potentials and can be used simultaneously. Thus, our accelerated GBP algorithm may solve an inference problem on a 2D grid-like MRF with 256 labels more than 250 times faster than the standard version. In the future, we are going to investigate how to generalize these techniques for other MRFs, and how we can integrate them with other acceleration techniques that are currently under development.

References

1. Winkler, G.: Image Analysis, Random Fields and Markov Chain Monte Carlo Methods. Springer, Heidelberg (2006)
2. Cowell, R.G., Dawid, A.P., Lauritzen, S.L., Spiegelhalter, D.J.: Probabilistic Networks and Expert Systems. Springer, New York (1999)
3. Yedidia, J.S., Freeman, W.T., Weiss, Y.: Bethe free energy, kikuchi approximations and belief propagation algorithms. Technical Report TR-2001-16, Mitsubishi Electric Research Laboratories (2001)
4. Yedidia, J.S., Freeman, W.T., Weiss, Y.: Generalized belief propagation. In: NIPS, pp. 689–695 (2000)
5. Shental, N., Zomet, A., Hertz, T., Weiss, Y.: Learning and inferring image segmentations using the gbp typical cut algorithm. In: ICCV 2003: Proceedings of the Ninth IEEE International Conference on Computer Vision, Washington, DC, USA, p. 1243. IEEE Computer Society, Los Alamitos (2003)
6. Kumar, M.P., Torr, P.H.S.: Fast Memory-Efficient Generalized Belief Propagation. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3954, pp. 451–463. Springer, Heidelberg (2006)
7. Veksler, O.: Efficient graph-based energy minimization methods in computer vision. PhD thesis, Cornell University (1999)
8. Yedidia, J.S., Freeman, W.T., Weiss, Y.: Constructing free energy approximations and generalized belief propagation algorithms. Technical Report TR-2004-40, Mitsubishi Electric Research Laboratories (2004)
9. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient belief propagation for early vision. Int. J. Comput. Vision 70, 41–54 (2006)
10. Kumar, M.P., Torr, P.H.S., Zisserman, A.: Obj cut. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005, vol. 1, pp. 18–25 (2005)