MIT CSAIL

6.869 Advances in Computer Vision

Spring 2011

Problem Set 6: Loopy BP and Image Segmentation

Posted: Wednesday, March 9, 2011	**Due:** Wednesday, March 16, 2011

You should submit a hard copy of your work in class, and upload your code (and all files needed to run it, images, etc) to stellar.
Your report should include images and plots showing your results, as well as pieces of your code that you find relevant.

Problem 6.1 *Loopy BP*

In this problem you will implement an algorithm for constructing dense stereo correspondence, following the work of Felzenswalb and Huttenlocher [2].

Stereo matching is one of the more studied problems in computer vision. It is the process of taking two or more images of a scene from different viewpoints and finding matching pixels between those images, that later allows reconstructing the 3D geometry of the scene. We'll cover this topic in an upcoming lecture, but you have the background now to implement a simple version of such algorithm.

Most of the recent stereo matching techniques focus on dense correspondences (finding matches for every pixel in the image), since it is required for applications such as image-based rendering or modeling. The input to our algorithm is a stereo image pair, and we would like to find disparities (shifts) between the pixels based on the two images alone.
On the course website you can find the file `tsukuba.zip`, which contains one such stereo pair, borrowed from the Middleburry stereo dataset [3], the standard dataset for evaluating stereo algorithms.

Many problems in computer vision can be formulated as discrete *multi-labeling* problems, were we would like assign each pixel one of L possible labels. Up till now, we have focused on formulating inference in terms of probability distributions, however in this problem set we will formulate our problem in terms of *energy* (minus log probability). For a given labeling f of the pixels, a common form of energy function that is usually encountered is

$$E(f) = \sum_p D_p(f_p) + \sum_{p,q \in N(p)} V_{pq}(f_p, f_q) \tag{1}$$

where D_p is the cost of assigning label f_p to pixel p, and is often referred to as the *data term*, V_{pq} is the cost of assigning labels f_p and f_q to neighboring pixels, and is frequently called the *smoothness or discontinuity term*. $N(p)$ is the set of neighbors of pixel p (4 immediate neighbors in grid graphs). For stereo, the labels will be disparity values that are assigned to

each pixel in the left image.
It is easy to show (you will do it in part (a) of this problem) that the minimal energy solution for Equation 1 corresponds the the MAP estimation of the corresponding MRF posterior distribution we have defined in class.

We can thus use Loopy Belief Propagation to solve for the optimal labeling of minimal energy. We'll use the max-product message update equation derived in class, where as we are working with minus log of the probabilities, the message update rule becomes what is called the *min-sum* rule, for which the message update equation is given by

$$m_{p\to q}(f_q) = \min_{f_p} \left(D_p(f_p) + V_{pq}(f_p, f_q) + \sum_{s\in N(p)\setminus q} m_{s\to p}(f_p) \right) \tag{2}$$

and the beliefs at each node can be computed as

$$b_p(f_p) = D_p(f_p) + \sum_{q\in N(p)} m_{q\to p}(f_p) \tag{3}$$

$D_p(f_p)$ can be thought of as a local evidence message to node p. We then select the label f_p^* that minimizes $b_p(f_p)$ locally at each node.

To solve for the disparities of the `tsukuba` image, we will assume only horizontal shifts of pixels. We will use the data and smoothness terms defined in [2]:

$$\begin{aligned} D_p(f_p) &= \lambda \min\left(|I_l(x,y) - I_r(x-f_p,y)|, \tau\right), \\ V_{pq}(f_p, f_q) &= \min\left(|f_p - f_q|, d\right) \end{aligned} \tag{4}$$

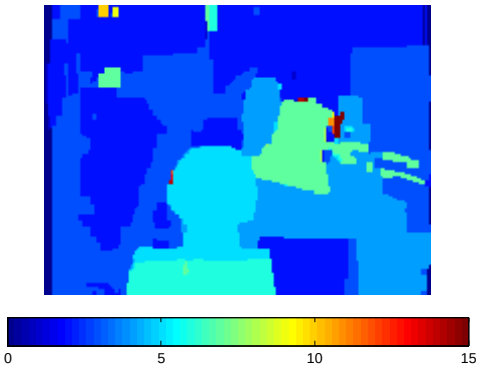
where I_l and I_r are the left and right images respectively. The data term enforces corresponding pixels (subject to our disparity estimates) to have similar intensities, and the smoothness penalizes different disparities at neighboring pixels, which will effectively lead to a smoother disparity map. λ, τ and d are parameters of the algorithm, which you can default to $\lambda = 0.3$, $\tau = 15$ and $d = 1.7$, and possibly tweak to improve your result.

MATLAB is far for being the ideal environment to solve multi-label optimizations on large grid graphs. Such algorithms are quite computationally intensive, and usually require a lower-level programming environment such as C or C++. To make our code tractable, we will additionally make the following design decisions: (a) We will reduce the size of our state space by considering only disparities in range $[0, 10]$; (2) We will pre-compute the data and smoothness terms (part (b)); and (3) we will use the message update schedule by Tappen and Freeman [4] to propagate information faster through the grid, which will reduce the number of BP iterations required. The message schedule is as follows: first, messages are sent along rows, forward and then backward at each row. Then messages are sent along columns, downwards and then upwards at each column. We consider one complete iteration of BP after traversing (forth and back) all rows and columns of the image.

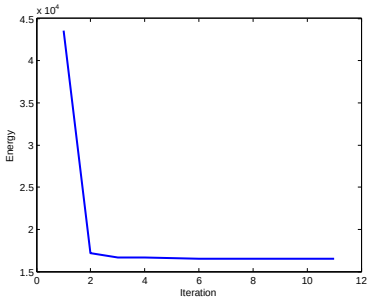
Instructions

(a) Prove that the minimal energy solution of Equation 1 is equivalent to the MAP estimate of the joint probability

$$p(f|y) = \frac{1}{Z} \prod_p \phi_p(f_p, y) \prod_{p,q\in N(p)} \psi_{pq}(f_p, f_q) \tag{5}$$



(a) Result disparity map



(b) BP energy

Figure 1: Our result for the `tsukuba` image, using 10 BP iterations and the parameters specified above. The computation time was 258 seconds.

where y stands for the observed data, and Z is the normalization factor, and define ϕ_p and ψ_{pq} in terms of D_p and V_{pq}.

(b) Write the function `PrecompDataTerm` that produces a $H \times W \times L$ matrix $\mathbf{D}$, such that $\mathbf{D}(y, x, f_p) = D_p(f_p)$ $(p = (x, y))$. Similarly, write the function `PrecompSmoothTerm` that produces the $L \times L$ matrix $\mathbf{V}$, such that $\mathbf{V}(f_p, f_q) = V_{pq}(f_p, f_q)$.

(c) Write the function `TotalEnergy`, that evaluates a solution labeling f according to Equation 1 and our choice of data and smoothness terms (Equation 4).

(d) Compute the disparities for the `tsukuba` image using Loopy Belief Propagation and the above message update schedule. Add to your report your estimated disparity map (see Figure 1(a)) after the first iteration, as well as the final result. In addition, after each complete iteration of message updates, compute the beliefs, and evaluate the energy of the current estimate. Plot the energy convergence curve throughout the BP iterations as shown in Figure 1(b). Note that although convergence is not guaranteed theoretically, Loopy BP energy curves usually have similar characteristics to the one shown in that figure: they are mostly monotonically decreasing, with sharp decrease in energy in the first 2-3 iterations. Keeping track of the energy when working with such optimizations is very useful to test your implementation. If your energy does not converge in a similar manner – it is very likely you have a bug.

Final hint: messages need to be normalized to prevent overflow *(remember we are summing energies here, as opposed to multiplying probabilities in previous problem sets). You can use the normalization method of [2]:* $m_{p\to q}(:) = m_{p\to q}(:) - mean(m_{p\to q}(:))$. Note moreover that the message equation can be written in MATLAB in vectorized form using the pre-computed matrices (part (b)). Doing that will improve the performance of your code significantly.

Problem 6.2 *Image Segmentation*

In this problem you will implement the mean-shift image segmentation algorithm presented by Comaniciu and Meer [1]. This problem is structured into two parts. The first part focuses on the implementation of the mean-shift algorithm as general nD-data point clustering algorithm.

The second part applies this algorithm to image segmentation.

On the class website you will find the MATLAB data file `pts.mat`, which stores a set of 3D points belonging to two 3D Gaussians. Test and debug your algorithm using this data set as described below. Plot the resulting clusters using the provided function `plot3dclusters`.

The mean-shift algorithm is discussed in the paper by Comaniciu and Meer, and we also provide some additional implementation details. As discussed in the paper, the mean-shift algorithm clusters an n-dimensional data set by associating each point to a peak of the data set's probability density. For each point, mean-shift computes its associated peak by first defining a spherical window at the data point of radius r and computing the mean of the points that lie within the window. The algorithm then shifts the window to the mean and repeats until convergence, i.e. the shift is under some threshold ϵ (we found $\epsilon = 0.01$ to work well). With each iteration the window will shift to a more densely populated portion of the data set until a peak is reached, where the data is equally distributed in the window.

(a) Implement the peak searching processes as the function

```
peak = findpeak(data, idx, r)
```

where `data` is the n-dimensional data set, `idx` is the column index of the data point for which we wish to compute its associated density peak, and `r` is the search window radius. The algorithm's dependence on `r` will become apparent from the experiments performed below.

(b) Implement the mean-shift function, which calls `findpeak` for each point and then assigns a label to each point according to its peak. This function should have the syntax

```
[labels, peaks] = meanshift(data, r)
```

The cluster labels are returned as `labels`, a vector that has an entry for each data column of `data` storing for each data point an associated cluster label. `peaks` is a matrix storing the density peaks found using meanshift as its columns. Note the mean-shift algorithm requires that peaks are compared after each call to `findpeak` and for similar peaks to be merged. For our implementation of mean-shift, we will consider two peaks to be the same if the distance between them is $\leq r/2$. Also, if the peak of a data point is found to already exist in `peaks` then for simplicity its computed peak is discarded and it is given the label of the associated peak in `peaks`.

(c) Using the data set stored by `pts.mat` run mean-shift with $r = 1, 2, 4, 5, 10$. Plot the results in separate subplots labeling each plot with its corresponding r value. How does varying the search window radius r affect the result?

(d) As implemented, the mean-shift algorithm of part (b) is too slow to be realized for image segmentation. We will therefore incorporate the following two speedups into our implementation. Upon finding a peak, the first speedup will be to associate each data point that is at a distance $\leq r$ from the peak with the cluster defined by that peak. This speedup is known as *basin of attraction* and is based on the intuition that points that are within one window size distance from the peak will with high probability converge to that peak (see Figure 2(a)). The second speedup is based on a similar principle, where points that are within a distance

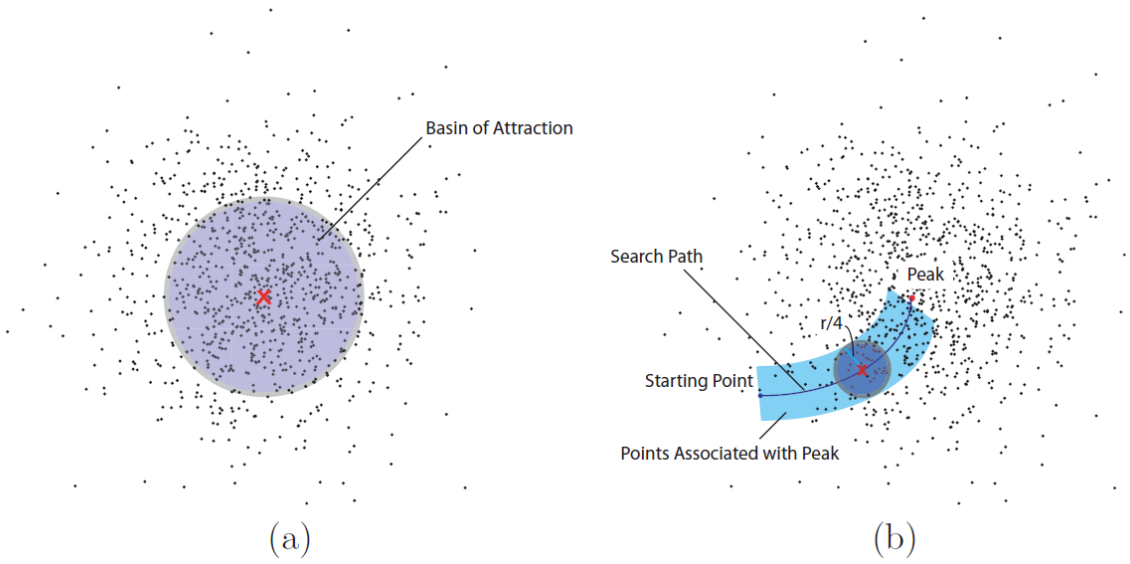


Figure 2: Speedups incorporated into the mean-shift algorithm: (a) basin of attraction, (b) points along the search path are associated with the converged peak.

of r/c of the search path are associated with the converged peak, where c is some constant value (see Figure 2(b)). We will choose $c = 4$ for this problem.

Incorporate the above speedups into your mean-shift implementation by modifying your implementation from part (b). The resulting modified function should have the syntax

```
[labels, peaks] = meanshift_opt(data, r)
```

To realize the second speedup you will need to modify `findpeak` as follows:

```
[peak, cpts] = findpeak_opt(data, idx, r)
```

where `cpts` is a vector storing a 1 for each point that is a distance of $r/4$ from the path, and 0 otherwise.

(e) Write the function `segmIm = imsegment(im)` that takes an image and returns the segmented image, where each pixel has the value of its cluster's center. Run your function on the images `sunset.bmp` and `terrain.bmp` available on the class website. Submit all your code, and add the resulting segmented images to your report.

References

[1] Dorin Comaniciu and Peter Meer. Robust analysis of feature spaces: Color image segmentation. In *CVPR*, pages 750–755, 1997. `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=609410&tag=1`.

[2] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient belief propagation for early vision. *International Journal of Computer Vision*, (70), 2003. `http://people.cs.uchicago.edu/~pff/bp/`.

[3] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, (47):7–42. `http://vision.middlebury.edu/stereo/data/scenes2001/`.

[4] M.F. Tappen and W.T. Freeman. Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters. In *ICCV*, pages 900–907, 2003. `http://www.eecs.ucf.edu/~mtappen/iccv03.pdf`.