Stereo Vision and Markov Random Fields

Alex Rudnick
School of Informatics and Computing, Indiana University
Bloomington, Indiana, USA
alexr@cs.indiana.edu

Abstract

Here we describe the stereo matching problem from computer vision, and some techniques for solving it as an optimization problem, including loopy belief propagation over Markov random fields. We also discuss some possible applications of these techniques to problems in natural language processing.

1 Introduction

In the stereo matching problem, we are given two images of the same scene, taken from two cameras set slightly apart, which is analogous to natural binocular vision with eyes. We must then estimate the horizontal disparity between a pixel in the image from the right camera and the corresponding pixel from the left camera. This disparity is a proxy for how close the relevant object is to the camera, as objects in the background will appear in roughly the same place in the two images, but objects closer to the camera will appear to shift more significantly.

Knowledge about the world helps us to sensibly constrain solutions to this problem. We know that objects in scenes are typically coherent wholes and so their pixels, being neighbors, are likely to have some locality in their disparity values. Thus we want to balance our desire to fit the parameters to the input images against our prior knowledge that objects tend to be smooth. Hopefully trying to satisfy both of these goals at once will lead to good reconstructions of 3D scenes.

We describe four different ways to frame this task as an optimization problem. First, we naïvely optimize each pixel independently, ignoring that pixel's neighbors. Then we try two approaches that consider a scanline (i.e., a row in the image) as a hidden Markov model: the familiar Viterbi algorithm, and belief propagation over a row of the image. These turn out to be mathematically equivalent. Finally, we consider loopy belief propagation over a grid-shaped Markov random field, which optimizes the disparity values for the image as a whole.

We will also briefly discuss image rectification using matrix transformations, and consider applications of belief propagation and other graphical models techniques to natural language processing.

2 Interlude on Image Rectification

Before using an image for a stereo vision, or for other reasons, we might wish to transform it. Particularly, in the stereo vision case, we might have a photograph that was taken from an inconvenient angle, such that an interesting surface was rotated partly away from the camera, but is still completely visible, or we may wish that two photographs were taken from the same perspective. We can perform a transformation that appears to rotate important surfaces in 3D space so that they are facing the camera at the same angle; this process is called *rectifying* the image.

With the use of homogeneous coordinates, many image transforms, including any combination of scaling, rotation, and translation, can be done with a single matrix multiplication. This is described in detail in (Foley et al., 1990, Chapter 5). Perspective transformations for image rectification can also be done with matrix multiplication. We do not address how to obtain the appropriate matrices, but one can manually find these matrices using image-manipulation applications such as The Gimp [1] or Photoshop.

[1] Available at http://www.gimp.org/ – particularly, The Gimp has a tool called the Perspective Tool.

Figure 1: A photograph of the Lincoln Memorial, before and after the perspective transform.

One can also do the rectification from within the application instead of just finding the perspective-transform matrix.

We were provided with an appropriate transformation matrix A to rectify a photo of the Lincoln memorial 1 (first image). To do this transformation mathematically, we might consider each pixel in the input image as a vector $\mathbf{p} = [x, y, 1]^T$ and then multiply $A\mathbf{p}$ to get a new vector $\mathbf{p}' = [x', y', w']^T$; the value from input pixel (x, y) is then mapped to output location $(x'/w', y'/w')$.

However, for ease of programming, we worked backwards, and looped over the pixels of the output image (x', y'), used the inverse transformation matrix A^{-1} to determine, for each output pixel, which input pixel would map onto it (if any), converting from rows and columns of a 2D image to homogeneous vector coordinates and back as necessary. Algebraically inverting a $3x3$ matrix is somewhat tedious, so we used SymPy [2] to build the inverse of a $3x3$ matrix full of variables, and then generated C++ code from that output, allowing us to invert any transformation matrix at run-time. The results of applying this transformation to the photograph of the Lincoln Memorial are in the second picture in Figure 1.

3 Stereo Matching Problem

In the stereo matching problem, we are given two images, I_R and I_L, each of which are comprised a grid of pixels. For each pixel from I_L, we want to assign a value of d, which selects some corresponding pixel in I_R as its best match. We want not just the individual pixels to be matched, but also the nearby pixels in some small square window. Once we pick the best value for d for every pixel from I_L, we generate an output image by setting the corresponding output pixel to have an intensity proportional to the value of d; thus objects in the foreground will appear brighter.

To pick the best d values for each pixel, we begin by computing a penalty for assigning a given disparity d to a certain pixel (i, j) from image I_L. This is done with the following function D, which calculates the sum of the squared difference in intensity values over all of the corresponding pixels in the window; here we ignore colors and only consider the intensity values of each pixel, so each pixel can be represented with a single number in the range $[0, 255]$. Notably, this formula does not consider the relationships between nearby pixels; it simply assigns a score for a particular pixel to take a given value of d. We will later add penalties for disagreeing with neighboring pixels, to encourage smoothness. If a block of pixels were copied verbatim from I_L to I_R and shifted exactly 20 pixels to the right, then that window could achieve a penalty of 0 by setting $d = 20$. As an artificially generated example, we took a picture of a gopher and shifted it 40 pixels to the right: the baseline stereo-matching algorithm was easily able to find the foreground object, as seen in Figure 2.

$$D(i, j, d) = \sum_{u=-W}^{u=W} \sum_{v=-W}^{v=W} (I_L(i + u, j + v) - I_R(i + u + d, j + v))^2 \quad (1)$$

[2] `http://sympy.org`

Figure 2: Gophers: the left image, right image, and a disparity map calculated with the baseline algorithm. The right image was created artificially by moving the left image 40 pixels. A d value of 40 is mapped to maximum brightness.

In cases where the window was displaced off the right edge of I_R, we considered this to have an infinite penalty, effectively disallowing certain settings to d. To correct for cases where the window was *partially* outside of an image, which includes cases along any edge, we divide penalties by the number of pixels that were inspected, effectively returning the average D-penalty per pixel.

There are a number of free parameters that we must set for each of the following algorithms. For all of them, we must choose some set of possible values for d. To simplify search, we restrict d to the integers, and give it a maximum value. The maximum possible value of d should vary with the size of the input images and the distance between the cameras that took them; with the test images given for this project, which are described in (Scharstein, 2007), we found that a maximum d of 40 gave sensible output images for some pictures, although we got better results with 100 for others. We must also choose a value of W, to determine the size of the window; here we use $W = 5$.

All of the approaches but the simple independent optimization algorithm require us to pick some function V which will describe the relationship between two neighboring pixels and the penalty for having different disparity values. Here we choose, fairly arbitrarily, a quadratic function of the difference between two d values, capped at 1000. An absolute value might have also been a good choice, instead of a quadratic function,

$$V = \min(40 * (d_1 - d_2)^2, 1000) \tag{2}$$

4 Algorithms for Stereo Matching

Here we will describe the four approaches that we have explored in this work, in increasing order of sophistication.

4.1 Independent Optimization

In the naïve baseline for stereo matching, we can simply choose a value of d for each pixel that minimizes the cost function $D(i, j, d)$ without regard to its neighbors. In the implementation, we use a single thread to loop over each pixel (i, j) from the left image I_R, then consider each possible integer value of d for that pixel, picking the one that minimizes the penalty.

This yields a sensible image, although it is fairly noisy; in the aloe plant picture, for example, patches of the leaves of the plant have low disparity values, even though they are part of a foreground object. See the first entry in Figure 3. We would like to encourage smoother images.

4.2 Viterbi Algorithm on Hidden Markov Models

We can use the familiar hidden Markov model formalism for stereo matching, if we are willing to make the simplifying assumption that we only need to smooth di parities across a single scan-line of the image at a time.

For readers from a natural language processing background, we can easily analogize this problem to that of building a part of speech tagger with HMMs. In the tagging case, we have a number of hidden variables, the parts of speech for each word in the sentence, and we imagine that the surface words are caused, through some stochastic process, by the underlying parts of speech. The words are distributed according to some "emission probability" distribution $P(w|t)$, the probability of the word w given the tag t. For the simplest, first-order Markov model, the transition probabilities are just the probability of a given tag conditioned on the previous tag, $P(t_i|t_{i-1})$.

However, in the stereo matching case, we are not working with probabilities, but with penalties, which we could interpret as negative log-probabilities, if necessary. Thus in places where we would normally search for a maximum probability, we simply search for a minimum penalty; the Viterbi algorithm is otherwise unchanged. By analogy with part-of-speech tagging, we now have a distribution for the emissions, defined by the D function, and also for the transitions, in the form of the V function.

To optimize the values of d for each row in I_L, we treat it as a separate HMM, filling in a chart with penalties in the familiar dynamic programming style, described in detail in (Jurafsky and Martin, 2009, Chapter 5), and then reading the back-pointers from that chart to recover the most likely sequence of d values over the row of the image, just as we would usually recover the most likely sequence of tags.

The images produced by this technique are somewhat less noisy than those from the independent optimization approach, although they show noticeable horizontal streaking. The picture of the aloe plant processed with this algorithm is shown in the upper-righthand corner of Figure 3. This approach performs very quickly: the chart is filled in left-to-right in one pass, and then in linear time we can read out the most likely sequence of d values for each row. It also parallelizes well, as is described in a later section, since each row is processed independently.

4.3 Belief Propagation on HMMs

Belief propagation was originally described in (Pearl, 1982) as a way to do inference over tree-shaped Bayes networks. It has since been extended for use with Bayes networks in general, as well as Markov networks (also known as Markov random fields). In the case where it is used to find marginal distributions given some evidence, it is known as the *sum-product* algorithm, but in this case, we are using it to find the MAP assignment of variables given some evidence, so this is an instance of the *min-sum* algorithm. Belief propagation is described in great detail in (Koller and Friedman, 2009, Chapters 11 and 13).

Belief propagation over many shapes of graphical models, including chains, trees, and polytrees, allows exact inference, such that the algorithm is guaranteed to converge on an optimum assignment of the unobserved variables. This is described in more detail in (Bishop, 2006, Chapter 8), Chapter 8, and (Koller and Friedman, 2009, Chapter 13), specifically deals with inference procedures to find MAP assignments.

In the general case of belief propagation, we typically construct what is called a *factor graph*, although in the computer vision literature, this does not seem to be common. The approach taken here may be equivalent to constructing a pairwise factor graph, where there is a factor node between every pair of interacting variables. For numerical stability and easy of implementation, as in the Viterbi case, we work with positive penalties (interpretable as negative log-probabilities) instead of positive probabilities; this avoids floating-point underflows.

The BP algorithm operates by sending "messages" between nodes in a graph, where each node, in this work, corresponds to a variable that must have a value assigned to it. The messages inform a given node's neighbors about that node's preferences for the neighbor's assignments, with the assumption that all of the nodes would like to minimize the global penalty for the graph. Operationally, each message contains a node's current estimate of the penalty that an assignment would incur if its neighbor were to take a certain value, so it must communicate a number for each possible setting that the neighboring node

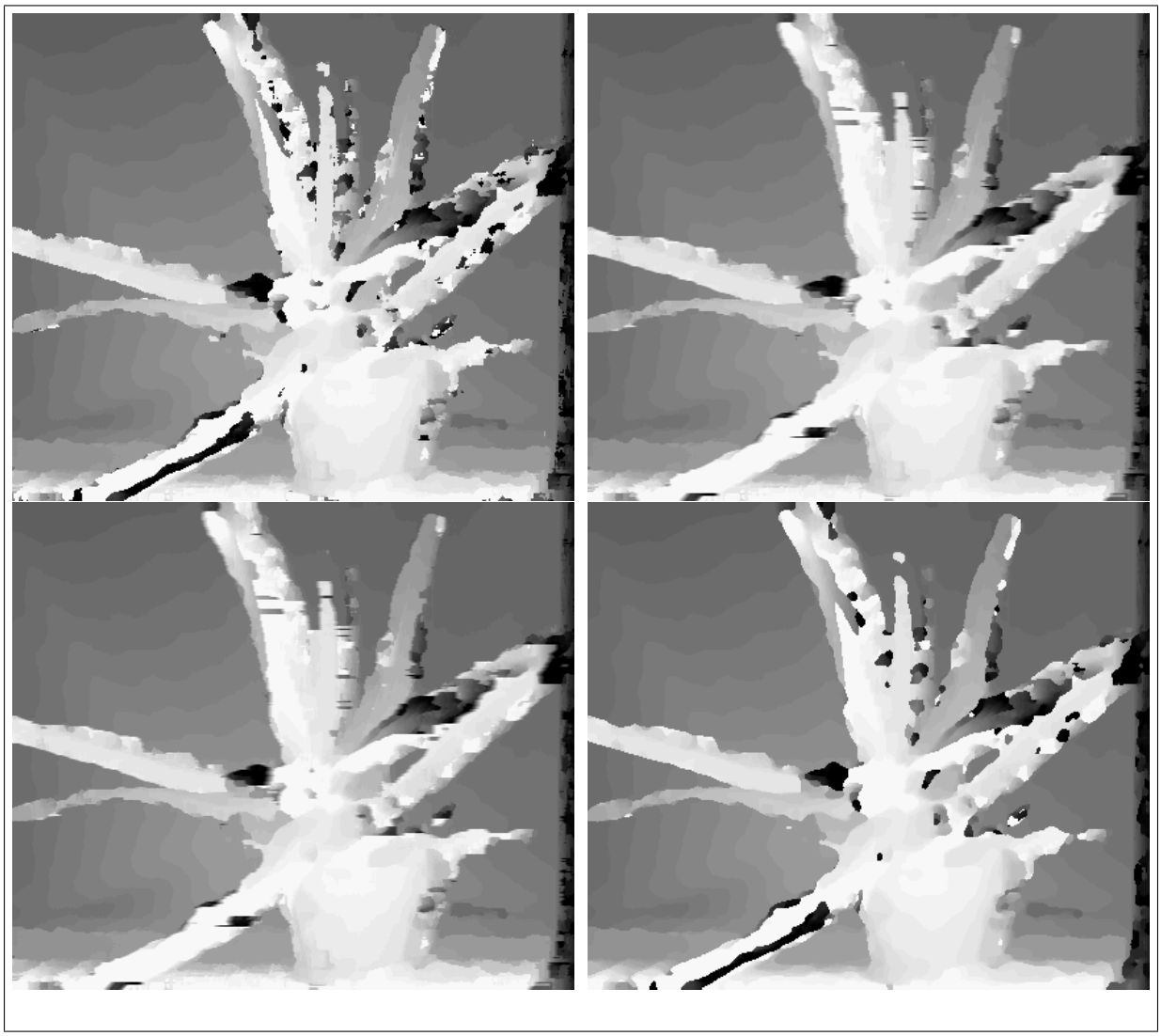

Figure 3: Disparity maps of the Aloe image. In order, these images were produced with the naïve independent approach, the Viterbi algorithm, belief propagation over chain HMMs, and loopy belief propagation over the grid MRF.

```
for each possible value your_d:
    lowest_penalty = INFINITY  // we'll pick one lower than this
    for each possible value of my_d:
        unary_cost = D(i, j, d)    // see Equation 1
        penalty = unary_cost
        penalty += V(d, your_d)  // see Equation 2
        for each neighbor not the recipient of this message:
            penalty += (previous message from neighbor)[my_d]
        lowest_penalty = min(penalty, lowest_penalty)
    message[your_d] = lowest_penalty
```

Figure 4: Pseudocode for computing a message to a given neighbor at a given timestep during belief propagation

could take.

Operationally, in this work messages are vectors of numbers, where the number at index i is the estimated penalty for the neighbor taking value i. At every timestep t, every node calculates a message for each of its neighbors, making use of its own ideas about what its d value should be from the D function, and also the messages that it received during the previous timestep $t-1$, from all of the neighbors that aren't the recipient of that message. Figure 4 gives pseudocode for the computation of outgoing messages. This procedure is run at every timestep, for every node, for each of the neighbors of that node.

In the case where we model each scanline of an image as an HMM, the belief propagation algorithm is run over a chain-shaped network – nodes have neighbors to the left and the right, unless they are on an edge. It is guaranteed to converge on messages that will allow us to compute the true MAP assignment, within a number of iterations proportional to the length of the chain. The final assignment for each node is simply the value of d that minimizes the sum of the D penalty and all of the final time step's messages from that node's neighbors, for that value d. Although we will not prove it here, a message-passing schedule in which we only pass a single message at each time step, going from left to right across the chain, and then back from right to left is mathematically equivalent to the Viterbi algorithm (Bishop, 2006, section 8.4.5). Intuitively, this is similar to the way information flows in the Viterbi algorithm; we fill in the chart from left to right, then read back-pointers from right to left.

4.4 Belief Propagation on Markov Random Fields

Markov Random Fields (also known as Markov Networks) are probabilistic graphical models that use arbitrary undirected graphs to express relationships between the variables in the network, in contrast to the directed graphs used to encode conditional probabilities in Bayes networks. There are algorithms for exact inference of MAP assignments over Markov Random Fields, but they are in general NP-Complete. We can use Belief Propagation, as described above, on generalized MRFs, but we have no guarantee that it will converge, and if it does converge, it may not approach an optimum assignment. This approach is called "loopy belief propagation", since it operates over a graph with loops in it; while inexact, it is often effective in practice.

We would like to encourage smooth transitions over d values in our disparity maps, not only along scanlines of the image, but over all neighboring pixels, so we consider nodes to have neighbors not only along scanlines, but also above and below. With this interpretation, we compute messages for all of a node's neighbors using the algorithm in Figure 4 unchanged. At each timestep, for every pixel in the image, we compute a message for all four neighbors of that pixel, and then these messages are used to compute later messages at subsequent timesteps. We continue running this algorithm for as long as we like, perhaps for hundreds of iterations, and eventually we stop iterating and compute d values for each pixel.

Notably, since the penalties in messages only ever increase, and the messages in a given node are

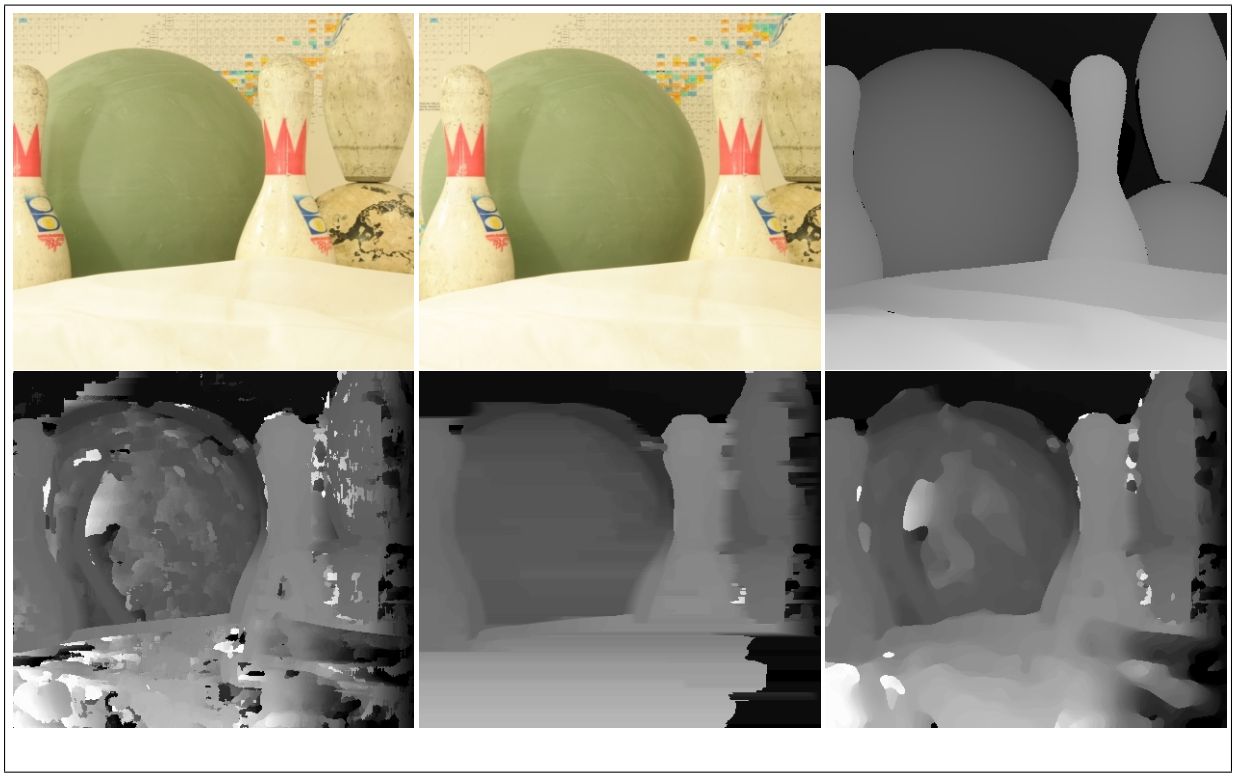

Figure 5: The Bowling image, from the set of test images. On the top row, the left image, the right image, and the correct displacement image from the set of images. On the bottom row, the independent optimization, the results from using the Viterbi algorithm, and finally the results from 100 iterations of belief propagation on an MRF. For the bottom row, the maximum d is set to 100.

composed of sums from several other nodes, these penalties can become very large after many iterations. We found that we got better results normalizing the outgoing messages from each node, such that they summed to some large number – otherwise, the costs for differing from one's neighbors would rapidly outstrip the "unary" cost of having a bad value of d, and since the nodes along the right-hand side of an image must have a value of 0 to avoid an infinite penalty (so that they do not reach off the edge), we were seeing entirely black images in early development.

As we can see in Figures 3 and 5, the disparity maps computed with BP over MRFs are relatively less noisy than those produced by independent optimization, and have less of the horizontal streaking produced by the scanline HMM approaches. We show in Figure 6, a few different settings of parameters of the MRF BP algorithm, particularly varying the normalization constant for outgoing messages and the number of iterations; the results are not drastically different.

5 Implementation Notes: Parallelism

In the implementation, we parallelized the Viterbi and MRF belief propagation algorithms, using the new C++11 threads standard and the multiple cores available on modern machines. In both cases, we created threads to handle computations for a row at a time. In the Viterbi case, the complete optimization for a row of the image is done in a thread, which is straightforward since that optimization does not depend on the computation for the other rows in the image. In the case of belief propagation, during each iteration, k threads are created (k should be set to the number of cores on the computer), and each thread t would compute the outgoing messages for a row r if $r \mod k == t$.

For performance purposes, we memoize $D(i, j, d)$, noting that since its output values are constant once computed (for a fixed pair of input images), threads may access the cached values of D without locking. This insight sped up processing considerably, allowing concurrent reads from many threads simultaneously.

Figure 6: Disparity maps of the Aloe image, produced with different settings for BP over the grid-shaped MRF. First, 100 iterations with messages normalized to sum to 1k, then normalized to 10k, then normalized to 100k, then finally 400 iterations, with messages normalized to 100k.

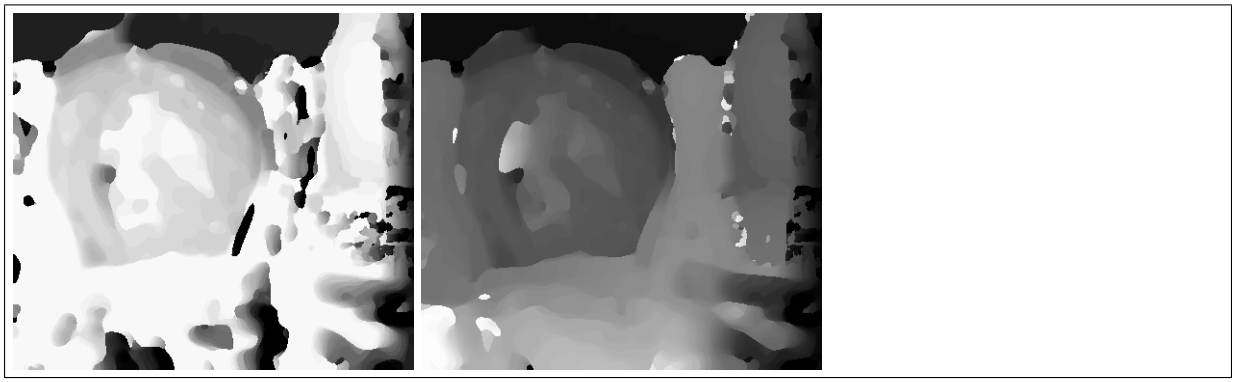

Figure 7: Disparity maps for the the Bowling image, from the set of test images. On the left, we tried computing the disparity map with belief propagation over the MRF, with a maximum d of 40. On the right, we used a maximum d of 100; we can verify with an image processing program that 40 pixels is slightly too narrow for these images. Note how much less splotchy the bowling pins are in the right-hand image.

6 Related Work

Stereo vision is a very well-studied field, and the stereo matching problem has been approached with much more sophisticated models than are discussed in this work, using features other than the simple squared differences between intensity values. Graphical models have been used extensively for stereo matching for at least about a decade; the earliest work we found using belief propagation was (Sun et al., 2003); Sun *et al* explicitly model occlusion and have a richer model of the image. Work on stereo matching with graphical models has continued, including work by (Srivastava et al., 2009), who manage their message-passing schedule to target ambiguous pixels eagerly, and also (Scharstein, 2007), who use fewer hard-coded heuristics and learn models of differences between disparities from data.

The natural language processing community has also been using graphical models; recent work in named entity recognition particularly uses conditional random fields (a variant of MRFs), such as (Finkel et al., 2005). This work is the basis for the NER system distributed with Stanford's Core NLP tools[3].

In general, graphical models are being used in NLP to solve problems where we have a number of interacting variables that we would like to solve for jointly because they are in some sense mutually constraining. As a straightforward example, in (Lee et al., 2011), a parsing system must address not only syntactic ambiguity, but also morphological ambiguity, and making firm morphological disambiguation decisions often leads to impossible syntactic situations later on, for systems built in a traditional "pipeline" approach. However, by solving morphological and syntactic ambiguities jointly using a single graphical model, Lee *et al* avoid such contradictions and encourage internally consistent linguistic analyses. The problem of early decisions leading to cascading errors in pipeline systems has been discussed in a number of places, but a particularly novel solution is provided in (Finkel et al., 2006); pipeline systems are here adapted into Bayes nets, and instead of the one-best solution being passed forward, solutions are probabilistically sampled.

7 Conclusions and Future Work

Here we have discussed the stereo matching problem from computer vision and described four different approaches for solving it, with a brief interlude on image rectification. Of particular interest to us is belief propagation for approximate inference on graphical models, because it allows us to tractably solve systems with large numbers of mutually constraining variables, even though solving them exactly would be NP-complete.

In the near future, we would like to use MRFs and approximate inference to solve problems in word sense disambiguation, and more generally in machine translation. We find the work on jointly parsing and performing morphological disambiguation especially encouraging. We (the single author) have a lot to learn about probabilistic graphical models; some of the things we would like to understand include different message passing schedules for belief propagation, other inference/optimization algorithms including graph cuts and simulated annealing, when exactly one must construct a factor graph, and the situations in which MAP assignments can be attained exactly in a tractable way. Finally, we suspect that work that uses exact constraint solvers, such as (Gasser, 2011), could be framed in terms of approximate solutions using trainable, flexible, and possibly faster probabilistic graphical models.

References

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 363–370, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Jenny Rose Finkel, Christopher D. Manning, and Andrew Y. Ng. 2006. Solving the problem of cascading errors: Approximate bayesian inference for linguistic annotation pipelines. In *Proceedings of the 2006 Conference on*

[3] `http://nlp.stanford.edu/software/corenlp.shtml`

Empirical Methods in Natural Language Processing, pages 618–626, Sydney, Australia, July. Association for Computational Linguistics.

James D. Foley, Andries van Dam, Steven Feiner, and John F. Hughes. 1990. *Computer graphics - principles and practice, 2nd Edition*. Addison-Wesley.

Michael Gasser. 2011. Toward synchronous extensible dependency grammar. In *Second International Workshop on Free/Open-SourceRule-Based Machine Translation*, pages 3–10, Barcelona, Spain, January. Universitat Oberta de Catalunya.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing*. Prentice Hall.

Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

John Lee, Jason Naradowsky, and David A. Smith. 2011. A discriminative model for joint morphological disambiguation and dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 885–894, Portland, Oregon, USA, June. Association for Computational Linguistics.

Judea Pearl. 1982. Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In David L. Waltz, editor, *AAAI*, pages 133–136. AAAI Press.

Daniel Scharstein. 2007. Learning conditional random fields for stereo. In *CVPR*.

Sumit Srivastava, Seong Jong Ha, Sang Hwa Lee, and Nam Ik Cho. 2009. Stereo matching using hierarchical belief propagation along ambiguity gradient. In *Proceedings of the 16th IEEE international conference on Image processing*, ICIP'09, pages 2061–2064, Piscataway, NJ, USA. IEEE Press.

Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. 2003. Stereo Matching Using Belief Propagation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(7):787–800, July.