CSE 586 Final Programming Project **Spring 2011**
Due date: Tuesday, May 3

What I have in mind for our last programming project is to do something with either graphical models or random sampling. A few ideas are below, but I also hope you will put some thought into coming up with a project idea of your own that is of interest to you or is relevant to your research. Theses topics (graphical models; sampling) are pretty wide open, so should be able to encompass anything you can think of… recall that even averaging numbers together can be formulated as a graphical model; dynamic programming also could be considered to fall under the topic, for an appropriately defined model.

Also, unlike previous projects, it is OK for this project to use publicly available software packages/toolboxes that perform the underlying mechanics of graphical model reasoning (some examples are below). However, if you use one, I would then expect you would be spending relatively more effort on defining the graphical model (what are nodes; what is the graph topology; what are the appropriate potential functions) or on evaluating against alternative approaches.

Project idea 1: (MRFs)
The paper "A Comparative Study of Energy Minimization Methods for Markov Random Fields with Smoothness-Based Priors" by Rick Szeliski et.al. contains a number of experiments across different vision applications, formalized as MRF problems on a pixel grid graph. They compare three solution methods: belief propagation, graph cuts, and tree-reweighted message passing. The paper, along with all the data and code they used, is at http://vision.middlebury.edu/MRF/ . One project idea is to choose one of their applications, and try to replicate their results using their image data, and then extending to try on your own data or to solve a new image-based MRF problem that you formulate.
Also, recall that you watched a video lecture by Dan Huttenlocher on efficient methods for belief propagation in low-level vision MRFs. He has a paper (co-authored with Pedro Felzenswalb on this work, as well a C++ implementation of the methods for several vision problems at http://people.cs.uchicago.edu/~pff/bp/ . You could try to use that code instead, and replicate their results / extend to use on a problem of your own.

Project idea 2: (HMMs)
Kevin Murphy has a Matlab toolbox for learning and inference within Hidden Markov Models at http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html
A second project idea is to download and learn how to use this toolbox, and demonstrate HMM learning and inference on a problem of your choosing. (Note, recent versions of matlab also have HMM functionality implemented in the stats toolbox http://www.mathworks.com/help/toolbox/stats/f8368.html). As for applications ideas, here's one: Ara Nefian has a bunch of papers on using HMMs for face recognition, for example by slicing face images into horizontal regions and

matching the 1D sequence of regions to learned face models to achieve recognition. His papers on this are at http://www.anefian.com/research/face_reco.htm

Project idea 3: (HMMs again)
Another HMM-related idea is to try to implement the segmental K-means algorithm for HMM training, as laid out in our lecture notes
http://www.cse.psu.edu/~rcollins/CSE586/lectures/cse586HMM_6pp.pdf
(note: original slides were from Steve Mills at U.Nottingham.) For example, you could try to learn HMMs to represent common paths through a scene given a set of observed trajectories. One such sample dataset is something we put together here, from tracking people in the Hub. The dataset consists of two 15-minute sequences. The first sequence, called SU2-L has low-density crowds, and the second sequence, SU2-H, has higher crowd density crowd. Each sequence is broken into a series of short 20 second clips. All people in the middle frame of each clip were detected and tracked by human volunteers to collect ground truth trajectories for that clip.

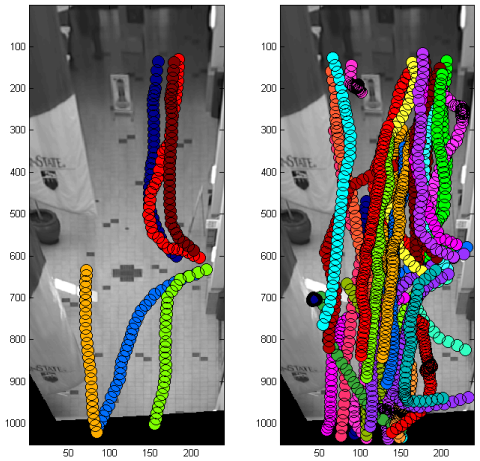


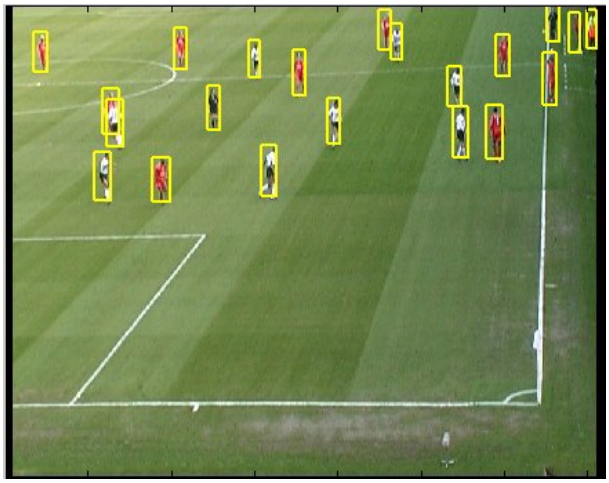
Figure 1: Sample ground truth trajectories of pedestrians tracked in the PSU Hub building. Left: from clip 50 of SU2-L (low density sequence). Right: Clip 20 of SU2-H (high density sequence).
http://www.cse.psu.edu/~rcollins/CSE586/HubTrajectories.zip - Hub trajectory data

I've supplied a function read_traj_block.m that reads in the trajectories for the people in each clip. See the comments at top of that file for more information. The routine returns arrays of X,Y coordinates for the trajectories of all people in the clip. Since the human volunteers only clicked on each person's location every 10 frames over the 20 second clip, there are 61 points for each trajectory. The X,Y locations are set to -1 for sampled frames where that person was not visible, otherwise they will be a positive numbers representing col and row in a top-down view (look at how the trajectories are displayed in read_traj_block.m for more details). Also returned in a third array are ID numbers for each person.

Project idea 4: (Kalman Filter)
As part of the VS-PETS 2003 workshop, some nice people hand-labeled 2500 video frames of a soccer sequence by drawing boxes around each person in every frame. There are 37,444 labeled boxes! This ground-truth box data is nice because they provide the raw observation or measurement data for a Kalman filter tracker, without having to get our hands dirty with detecting blobs in the raw image. So another project idea is to pick a box at random from this dataset, and start trying to track it from frame to frame, using a Kalman filter tracker. Recall that Kalman filters

have the same graphical model structure as HMMs, but they work with continuous variables and only Gaussian probabilities. Greg Welch has a web site devoted to Kalman filter resources http://www.cs.unc.edu/~welch/kalman/ , including tutorials, slides, tools, etc. Also note that Kevin Murphy (again) has a Kalman filter matlab toolbox, at
http://www.cs.ubc.ca/~murphyk/Software/Kalman/kalman_download.html .
Some design decisions you need to make to use a Kalman filter are what the state space should be (just x,y centroid of the box, or also width and height, or also include velocity on each of those terms, or…) and what the motion model should be (constant position plus noise, constant velocity plus noise, constant acceleration plus noise, or…). Another decision is how to decide which box in frame k+1 is the measurement that corresponds to your target. This is a data association task. A simple approach is to try to take the box that has a state vector "closest" to your motion-predicted state in frame k+1, taking into account the uncertainty in your prediction (it's covariance).

I have written some sample code to show how to read the box data and figure out what boxes are in each frame, as well as where and what size they are. This picture is produced by the sample code:



http://www.cse.psu.edu/~rcollins/CSE598B/Datasets/soccerboxes.mat - the box data
http://www.cse.psu.edu/~rcollins/CSE598B/Datasets/soccerboxesusage.m - sample code
http://www.cse.psu.edu/~rcollins/CSE598B/Datasets/genfilename.m - helper function
http://www.cse.psu.edu/~rcollins/CSE598B/Datasets/Soccer.zip - all the image data (big)

Project idea 5: (Quasi-Monte Carlo)
Recall that Monte Carlo integration is a numerical integration method that works by generating uniform random numbers in a volume that encloses the function you are trying to integrate, and then estimates the integral as being proportional to the percentage of point samples falling "within" your function times the area of the outer volume you are generating the samples in. It turns out that you can generate

even better (lower-variance) estimates if you use something called “quasi-random” numbers rather than regular pseudo-random numbers that most languages give.

Suppose you generate a set of 3000 pseudo-random 2D points using Matlab's rand function, and plot them as shown in Figure3a. Although the pseudo-random numbers are uniformly distributed, they will tend to form clusters and gaps, as can be seen in the figure. Figure3b shows the same number of points, but generated by the quasi-random Sobol generator. It looks much different! Indeed, the points seem to form a “smooth texture” that fills the whole space much more evenly. To make clear the correlated, texture-like nature of Sobol points, Figure3c shows a picture containing only a 1000 point samples. The pattern formed from the points looks like a regular lattice, which it pretty much is. Quasi-random number generators work in a coarse-to-fine manner, appearing to first fill space coarsely with a correlated pattern, then going back to fill in remaining gaps again and again at progressively finer scales.

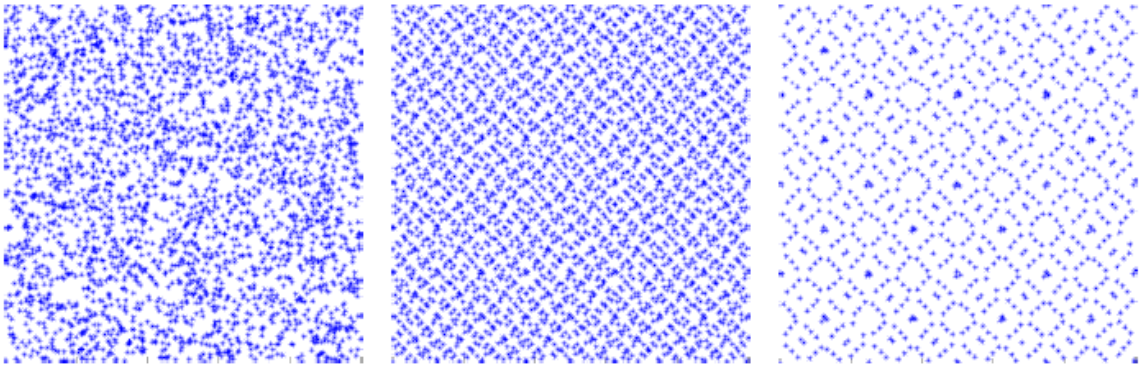


Figure 3: (a) 3000 points from a pseudo-random number generator. Note the clustering and gaps. (b) 3000 points from the Sobol quasi-random number generator, which fills space more evenly. (c) 1000 points from the Sobol generator, clearly displaying the lattice-like structure of the point generation process.

One project idea then, is to implement Monte Carlo Integration using pseudo-random numbers, and also using quasi-random numbers, and compare the results for some functions that you know the analytic answer for. You would want to compare for different numbers of sample points, and over many independent runs, so that you could empirically compare the variance in the estimates, for different numbers of points N. For more on quasi-random Monte-Carlo integration, see the page http://www.puc-rio.br/marco.ind/quasi_mc.html (also there are pages on quasi-Monte Carlo at wiki and wolfram mathworld). An implementation of one kind of quasi-random point generator can be found at http://people.sc.fsu.edu/~jburkardt/m_src/sobol/sobol.html .

Project idea 6: (Sampling-based Graphical Model Inference)
We've seen how to use belief propagation to estimate either the marginal distribution or expected value of a variable (node) within a graphical model. We should be able to do the same thing with random sampling methods. The idea is to "simulate" many different likely configurations of variable values in the graphical model, and estimate the marginal or expected value using those configurations (this is basically a process of numeric integration via Monte Carlo). Do do the simulation within the graph-structured model, we use something called "ancestral sampling". Here is what Chris Bishop has to say about ancestral sampling in his PRML book

There are many situations in which we wish to draw samples from a given probability distribution. Although we shall devote the whole of Chapter 11 to a detailed discussion of sampling methods, it is instructive to outline here one technique, called *ancestral sampling*, which is particularly relevant to graphical models. Consider a joint distribution $p(x_1, \ldots, x_K)$ over K variables that factorizes according to (8.5) corresponding to a directed acyclic graph. We shall suppose that the variables have been ordered such that there are no links from any node to any lower numbered node, in other words each node has a higher number than any of its parents. Our goal is to draw a sample $\widehat{x}_1, \ldots, \widehat{x}_K$ from the joint distribution.

To do this, we start with the lowest-numbered node and draw a sample from the distribution $p(x_1)$, which we call $\widehat{x}_1$. We then work through each of the nodes in order, so that for node n we draw a sample from the conditional distribution $p(x_n|\mathrm{pa}_n)$ in which the parent variables have been set to their sampled values. Note that at each stage, these parent values will always be available because they correspond to lower-numbered nodes that have already been sampled. Techniques for sampling from specific distributions will be discussed in detail in Chapter 11. Once we have sampled from the final variable x_K, we will have achieved our objective of obtaining a sample from the joint distribution. To obtain a sample from some marginal distribution corresponding to a subset of the variables, we simply take the sampled values for the required nodes and ignore the sampled values for the remaining nodes. For example, to draw a sample from the distribution $p(x_2, x_4)$, we simply sample from the full joint distribution and then retain the values $\widehat{x}_2, \widehat{x}_4$ and discard the remaining values $\{\widehat{x}_{j\neq 2,4}\}$.

So one project idea is to define a directed graphical model, perhaps having either a chain or tree structure (so that belief propagation yields exact solutions for the marginal using sum-product algorithm), and compute marginals for various nodes using BP, and also using ancestral sampling / monte carlo integration, and compare the results. If you are ambitious, you could also try computing MAP estimates using BP and using sampling (a MAP estimate computed by sampling is formed simply by remembering the sampled configuration that yields the highest joint probability seen so far… super easy!).