Markov random fields and techniques for performing inference with them

UCSC CMPE 264 guest lecture
Mar. 13, 2012

James M. Coughlan, Ph.D.

THE SMITH-KETTLEWELL
EYE RESEARCH INSTITUTE

Energy functions

Energy (objective, cost, fitness, etc.) functions very useful in computer vision!

Allow you to formulate an inference problem in terms of prior constraints (hard or soft) and multiple forms of evidence

Natural connection with Bayesian statistics

Energy functions

Let *x* = unknown variables to be inferred, *d* = measured data (e.g. from image pixels)

$E(x) = U(x) + L(x,d)$

U(x) encodes prior constraints: what you know even before you see the image!

L(x,d) encodes information contained in data

Energy functions: performing inference

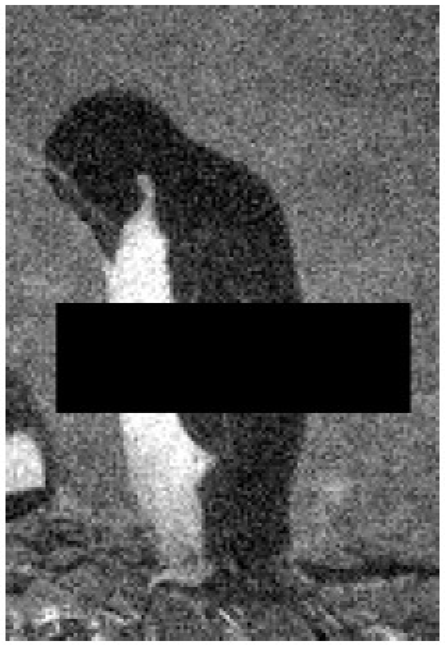
Given data *d*, the most likely interpretation, x^*, of the data is obtained by minimizing E(x):

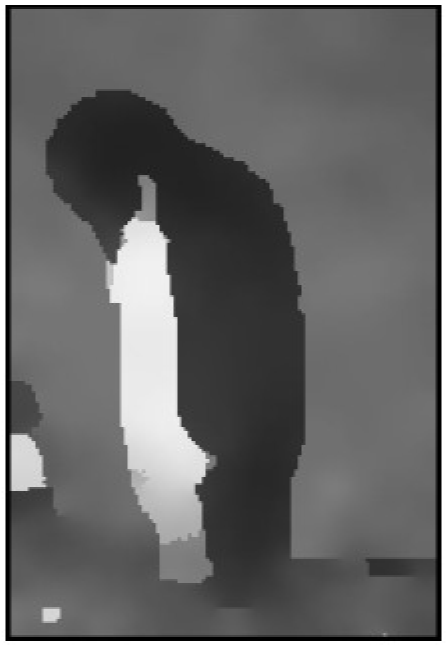
$$x^* = \arg\min_x E(x)$$

Since *E(x)* is the sum of *U(x)* and *L(x,d)*, minimizing *E(x)* is a compromise between minimizing *U(x)* and minimizing *L(x,d)*.

Sample application: energy function for image restoration

d is a noisy grayscale pixel map taken from a camera (plus added corruption), and *x* is the ideal pixel map that would result if there were no noise.


Corrupted


Restoration

Energy function for image restoration

Prior constraint: *U(x)* says that ideal pixel maps tend to be smooth (like a cartoon)

Evidence (likelihood) from image: *L(x,d)* says that the measured pixel intensity equals the ideal intensity corrupted by some error value (usually a *small* error).

In other words, *U(x)* is low when *x* is a smooth pixel map and high otherwise. Similarly, *L(x,d)* is low when the maps *x* and *d* are similar to each other and high if they are too different.

Energy function for image restoration

Notation:

x_i = ideal image intensity at pixel *i*

I_i = measured image (data) intensity at pixel *i*

x = entire ideal image

I = entire measured image

$$E(x) = U(x) + L(x, I)$$

Energy function for image restoration

Prior: $U(x) = \alpha \sum_{<ij>} B_{ij}(x_i, x_j)$

where $B_{ij}(x_i, x_j) = |x_i - x_j|$

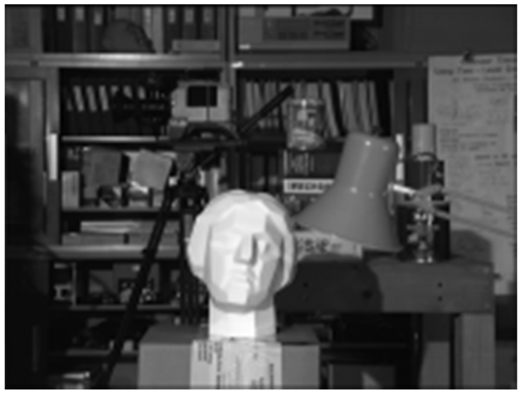
Likelihood: $L(x, I) = \beta \sum_i L_i(x_i, I_i)$

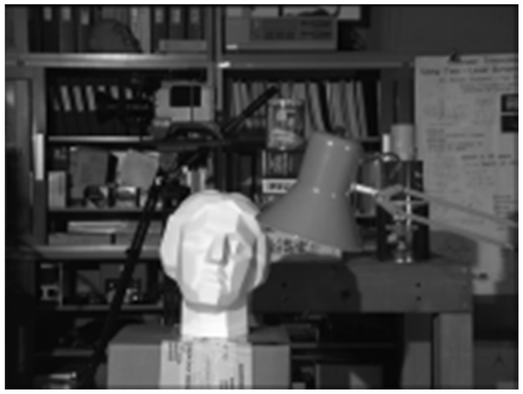
where $L_i(x_i, I_i) = |x_i - I_i|$

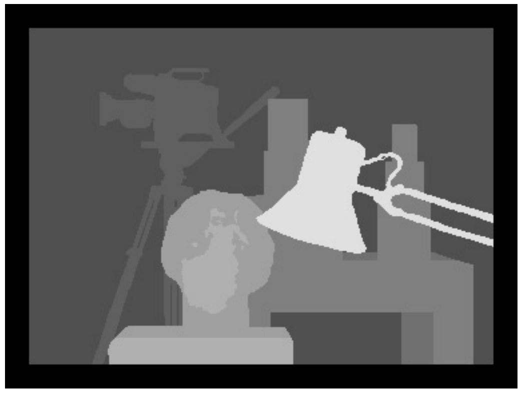
Here $\alpha > 0$ and $\beta > 0$ are constants; sum over <*ij*> refers to sum over all neighboring pixels *i* and *j*.

Energy function for stereo

Stereo – given left and right images, estimate unknown disparity field (see animated gif)


Left


Right


Ground truth disp.

<u>Prior constraint:</u> disparity field tends to be smooth

<u>Evidence from image:</u> given a pixel in left image, only some pixels in right image (along same row of pixels, i.e. epipolar constraint) have similar intensity

(Recall that *disparity is inversely proportional to depth* when two camera views are parallel and images are rectified.)

Energy function for stereo

Notation:

x = unknown disparity map, where x_i = disparity at pixel i.

The pixel coordinates of pixel i are (r_i, c_i).

Left and right images: $I_L(r,c)$ and $I_R(r,c)$.

Energy function for stereo

What does a disparity value x_i at pixel i mean?
→ (r_i, c_i) in right (reference) image matches with (r_i, c_i+x_i) in left image.

Given this match, we expect similar intensities at corresponding locations:

$$I_R(r_i, c_i) \approx I_L(r_i, c_i + x_i)$$

Energy function for stereo

Unary term (likelihood) at one pixel penalizes intensity difference between right pixel and corresponding left pixel predicted by disparity:

$$L_i(x_i, I_L, I_R) = |I_R(r_i, c_i) - I_L(r_i, c_i + x_i)|$$

Binary term (prior) between neighboring pixels *i* and *j* penalizes disparity differences:

$$B_{ij}(x_i, x_j) = |x_i - x_j|$$

This prior imposes a bias towards *fronto-parallel* surfaces.

Energy function for stereo

Putting it all together: $E(x) = U(x) + L(x,I_L,I_R)$

where $U(x) = \alpha \sum_{<ij>} B_{ij}(x_i, x_j)$

and $L(x, I_L, I_R) = \beta \sum_i L_i(x_i, I_L, I_R)$

$\alpha > 0$ and $\beta > 0$ are positive constants; sum over *<ij>* refers to sum over all neighboring pixels *i* and *j*.

How to minimize energy function?

Generic method: **gradient (steepest) descent**

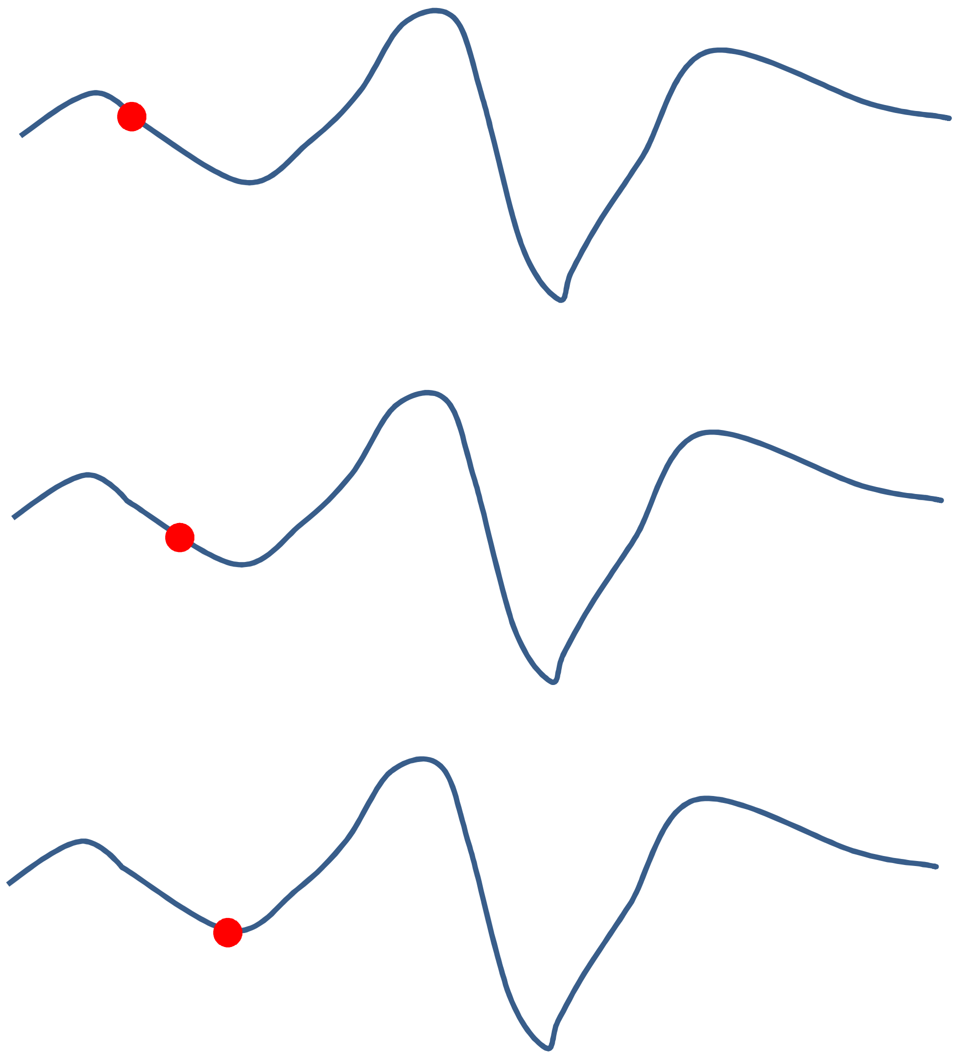
Given a value of *x* at iteration k, $x^{(k)}$, pick new value $x^{(k+1)}$ as follows:

$$x^{(k+1)} = x^{(k)} - k\nabla E(x^{(k)})$$

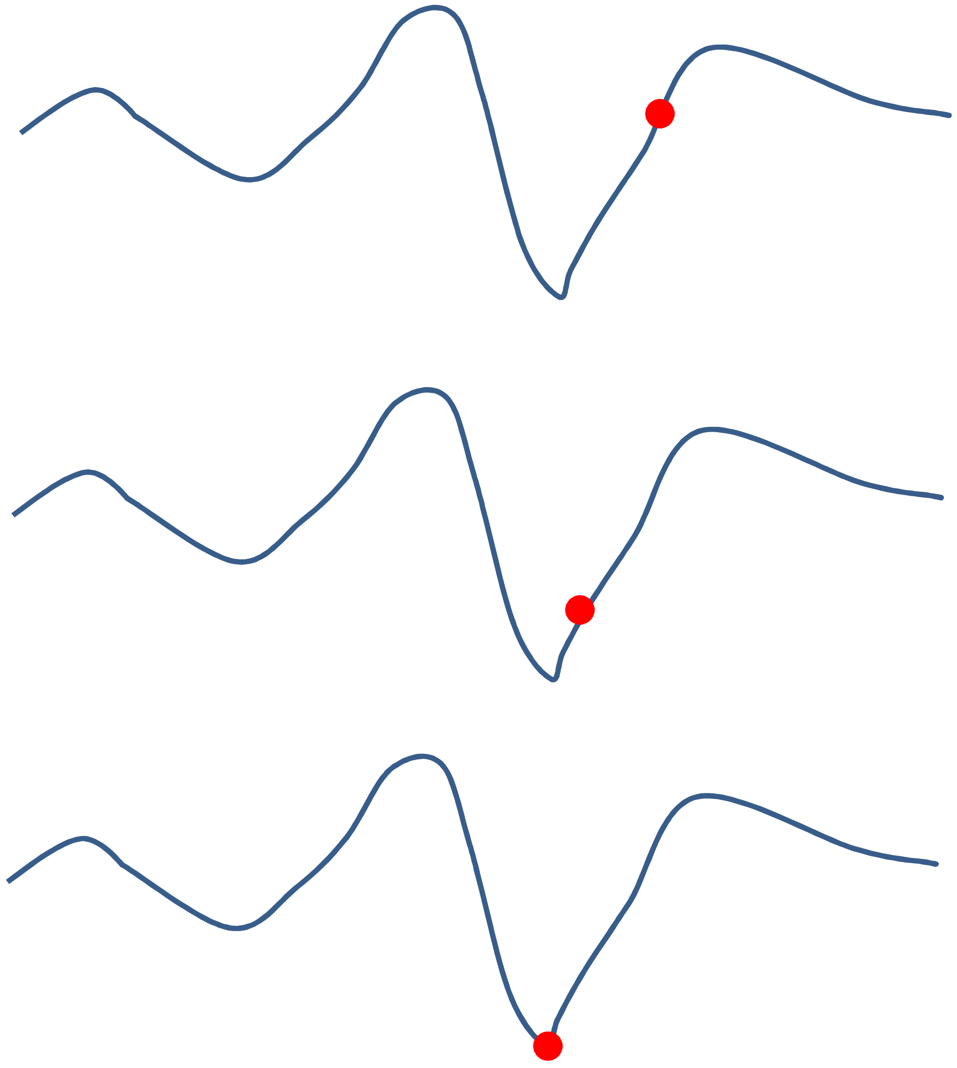
where *k* is a small positive constant. In other words, follow *E(x)* downhill!

Problem: gets trapped in local minima unless the initial condition is close to the correct solution!

Steepest descent: getting stuck in local minima



Steepest descent: getting stuck in local minima



Markov random field (MRF)

Notice: stereo energy function is a sum of local terms, each of which involves just the disparity at one pixel (unary terms) or the disparities at two neighboring pixels (binary interaction terms)

This is an example of a **Markov random field**.

<u>Defining feature of MRF:</u> total energy is a sum of terms; only a few variables interact in each term (**locality property**)

MRF optimization

Luckily, there are several optimization techniques that work well for MRFs. These techniques exploit locality property of energy function.

Even if the techniques aren’t guaranteed to find the very best minimum energy, they usually find a good approximation.

Today we will talk about two such techniques: **simulated annealing** and **belief propagation**.

Simulated annealing (SA)

SA is a stochastic approach to minimizing energy functions, either of continuous or discrete variables. (Not restricted to MRFs, but much more practical for MRFs in general.)

Basic idea:

- Steepest descent would work if there were a way to get unstuck out of local minima.

- To get unstuck, SA adds *random perturbations* to steepest descent (like going downhill in a wind storm).

- Over time, reduce the strength of the random perturbations (like “cooling” the system over time) so that you don’t inadvertently get knocked out of a good local (or even global) minimum towards the end.

SA algorithm

To minimize energy function *E(x)*:

1.) Pick an initial guess for *x*, and choose a starting "temperature" *T*.

2.) Propose a new state, called *x'*, which is a small perturbation of *x*. This **proposal** is obtained by making some sort of random, small change (a "move") to *x*.

3.) If *E(x')<E(x)*, i.e. the proposal improves the energy, then let *x'* be the new state (accept the proposal); otherwise, only accept *x'* as the new state with probability: $P = e^{(E(x)-E(x'))/T}$

4.) Lower the temperature *T* slightly (e.g. reduce it by a factor such as 0.9999). This defines the "annealing schedule."

5.) Go to step 2; repeat many times, until you are happy enough with the energy obtained.

SA: comments

For an MRF, a single proposal typically only changes one (or a few) variables. The corresponding energy change is thus quick and easy to compute – *this is the key to SA's speed/effectiveness for MRFs*.

Notice that proposals that "follow the gradient" are more likely to be accepted than those that don't.

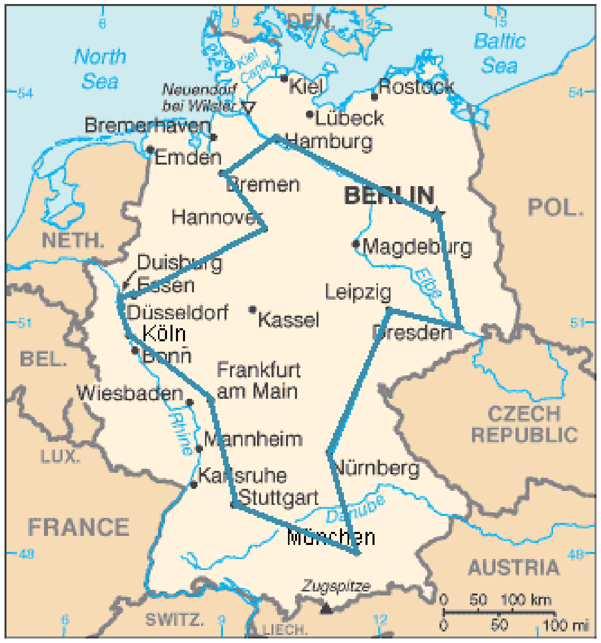
$P = e^{(E(x)-E(x'))/T}$ means that even a proposal that makes energy worse can be accepted; acceptance is more likely if energy isn't too much worse and if T is high.

Physics connection: annealing. To get a material (e.g. a metal) in a low energy state, start at high temperature and slowly decrease temperature over time.

SA: example

Traveling Salesman Problem (TSP)

Given *N* cities on a 2D map, find a closed path that takes a salesman through all the cities (starting at any arbitrary city A, hitting all the other cities exactly once, then returning to A), such that the total distance traveled is minimized.



TSP notation

N cities with coordinates (x_i, y_i) where *i = 1, 2, 3, …, N*.

Distance between city *i* and city *j* is denoted by *D(i,j)*, which is a symmetric matrix with zeros along the diagonal.

Definitions: a **path** *p* is a vector $p = (p_1, p_2, \ldots, p_N)$ where all p_i take integer values from 1 through N.
A path is **valid** if $p_i = p_j$ only when *i=j*, i.e. each city is hit exactly once and no cities are omitted along path.

TSP energy function

$$E(p) = \sum_{i=1}^{N} D(p_i, p_{i+1})$$

Note that indices "wrap around," so that p_{N+1} is the same variable as p_1

Technical aside: the TSP energy function isn't exactly an MRF, just because of the global (non-local) constraint that the path *p* must be valid. But it has the flavor of an MRF and is still a nice example for demonstrating SA.

TSP background

TSP is an NP-complete problem, which means it is impractical to find the globally minimum path when *N* is large enough. SA is a reasonable approach for finding a good (if sub-optimal) solution, but there is lots of work on better techniques for TSP.

Why is TSP so hard? There are *N!* possible paths to consider, of which *(N-1)!/2* them correspond to distinct physical paths. For instance, if *N=100*, then there are more than 10^{155} distinct paths.

SA for TSP

Start with any valid path.

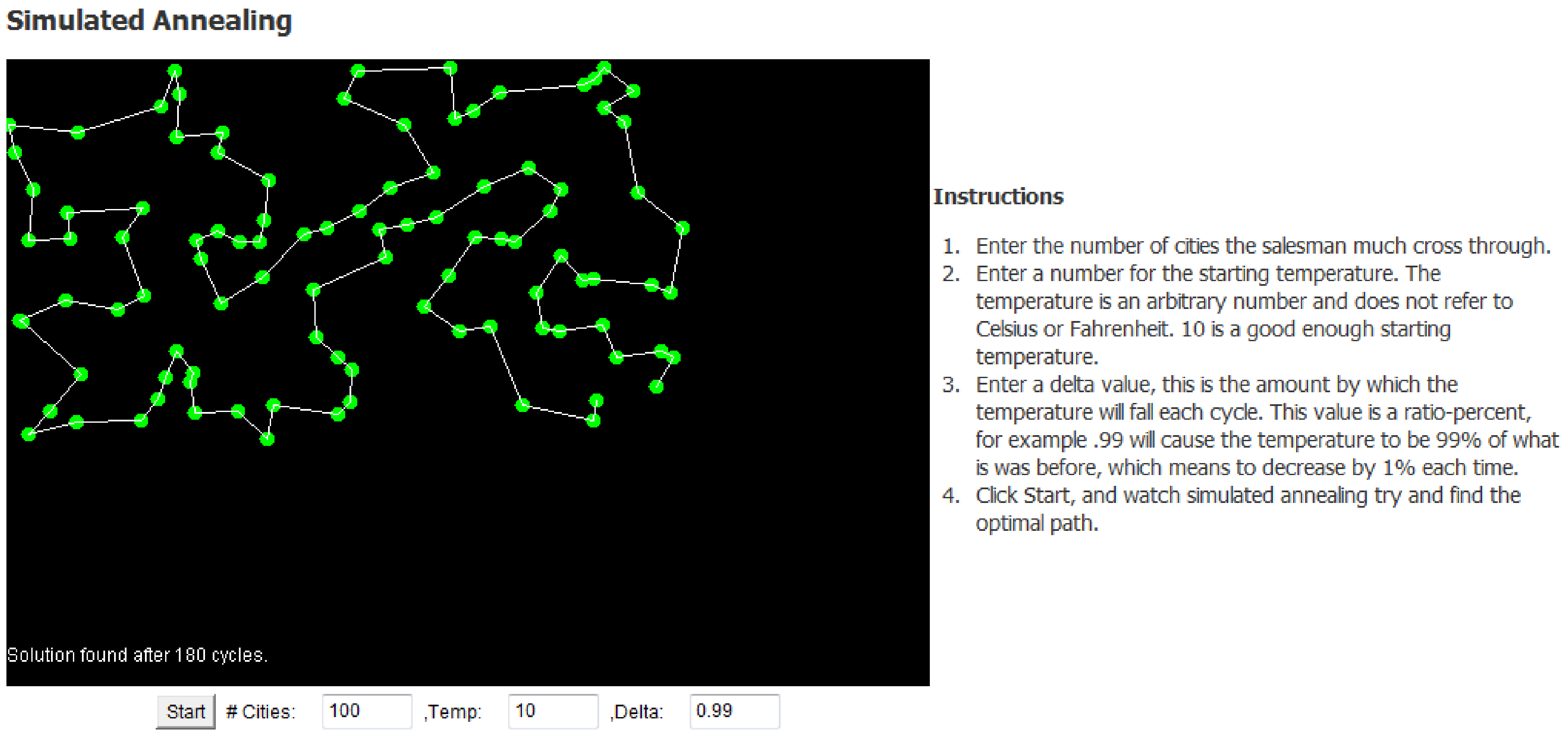
Proposal: pick two distinct cities at random, and propose switching them.

In other words: pick *i* and *j* at random, so that $i \neq j$, and swap values of p_i and p_j

In this way the path explored by SA will *always* remain valid.

SA for TSP: interactive demo

http://www.heatonresearch.com/articles/64/page1.html



Simulated Annealing

Solution found after 180 cycles.

Start # Cities: 100 ,Temp: 10 ,Delta: 0.99

Instructions

1. Enter the number of cities the salesman much cross through.
2. Enter a number for the starting temperature. The temperature is an arbitrary number and does not refer to Celsius or Fahrenheit. 10 is a good enough starting temperature.
3. Enter a delta value, this is the amount by which the temperature will fall each cycle. This value is a ratio-percent, for example .99 will cause the temperature to be 99% of what is was before, which means to decrease by 1% each time.
4. Click Start, and watch simulated annealing try and find the optimal path.

Belief propagation (BP)

SA is general purpose but can be prohibitively slow.

For a variety of problems, BP is much faster (and in some cases it even gives exact solutions in a finite number of iterations).

(Standard BP only works when variables are discrete, though some extensions to continuous variables have been devised.)

BP overview

BP is an iterative, deterministic algorithm in which neighboring variables “talk” to each other, passing messages such as:

“I (variable x_3) think that you (variable x_2) belong in these states with various likelihoods…”

BP convention

Usually we want to minimize *E(x)*

For BP discussion, let’s assume the opposite:

we want to <u>maximize</u> *E(x)*

BP overview (con't)

After enough iterations, this series of conversations is likely to converge to a consensus that determines the value of *x* that maximizes the energy function *E(x)*.

For each variable x_i, a score for all possible states is calculated called the **belief** function.

<u>BP algorithm summary:</u> (a) update messages until convergence; (b) calculate beliefs; (c) for each variable, choose the state with maximum belief.

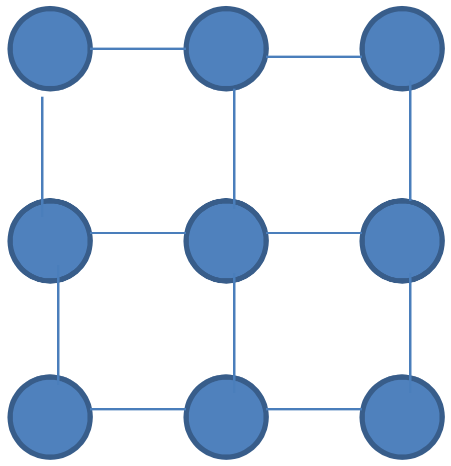
BP overview (con’t)

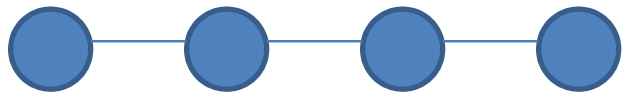
Property: if MRF has no loops in connectivity, then BP messages are guaranteed to converge in a finite number of steps, and the resulting beliefs will indicate the precise state *x* that maximizes *E(x)*.

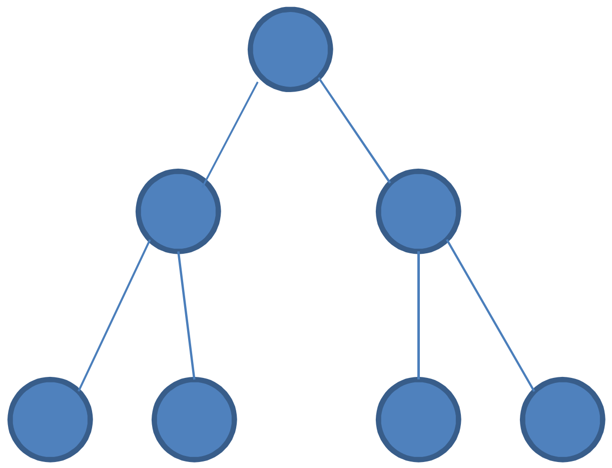
(BP was invented for use with MRFs with tree structure, i.e. no loops.)

But empirically it also works well even when there are loops, even though message convergence is no longer guaranteed and the resulting beliefs are approximate.

BP: loopy or not?







BP notation

Here we restrict ourselves to **pairwise** MRFs – MRFs in which all interactions involve pairs of variables (no triplets or higher):

$$E(x) = \sum_i f_i(x_i) + \sum_{<ij>} g_{ij}(x_i, x_j)$$

(For simplicity we don't mention data variables explicitly, though they appear implicitly.)

BP can be extended to higher-order interactions, but the speed/memory requirements scale *exponentially* with the interaction order.

BP messages

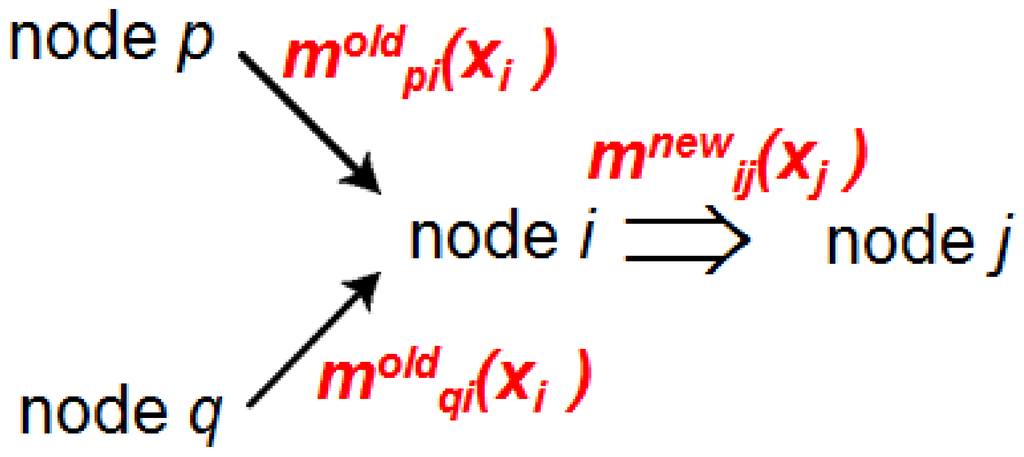
Message from node *i* to node *j*:
$m_{ij}(x_j)$

High value of $m_{ij}(x_j)$
means that node *i* "believes"
that node *j* is likely to be in state x_j

Usually initialize all message values to zeros, or else random values

BP message updates

To update message from *i* to *j*, consider all messages flowing into *i* (*except* for message from *j* itself):



BP message update equation

Messiest equation in entire talk:

$$m_{ij}^{new}(x_j) = \max_{x_i} g_{ij}(x_i, x_j) + \underbrace{f_i(x_i) + \textstyle\sum_{k \in Nbd(i) \setminus j} m_{ki}^{old}(x_i)}_{h_i(x_i)}$$

Here *$h_i(x_i)$* is a function of *x_i* that includes local evidence for *x_i* plus all messages flowing into *x_i* (excluding message from *x_j* to *x_i*).

BP message update schedule

Synchronous: update all messages in parallel

Asynchronous: update one message at a time

With luck, messages will converge after enough updates.

Which schedule to choose?

For MRF with chain structure, asynchronous is most efficient for a serial computer (up and down chain once guarantees convergence). Similar procedure for a tree.

For a grid (e.g. stereo on pixel lattice), people often sweep in an “up-down-left-right” fashion.

Choosing a good schedule requires experimentation.

BP belief read-out

Once message updates have converged, use belief read-out equation:

$$b_i(x_i) = f_i(x_i) + \sum_{k \in Nbd(i)} m_{ki}(x_i)$$

The *x* that maximizes *E(x)*, *x**, is then given by:

$$x_i^* = \arg\max_{x_i} b_i(x_i)$$

(Caveat: problems arise if two or more states tie for maximum energy.)

BP computational complexity

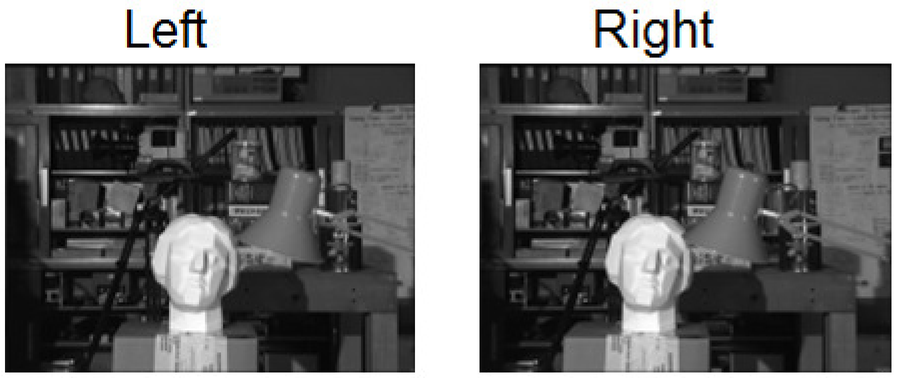
The main cost is the message updates.

If each variable x_i can have one of S states, and there are N variables in the MRF…

then each message update (across entire MRF) has $O(NS^2)$ complexity.

For comparison, brute-force exhaustive search would have $O(S^N)$ complexity.

BP applied to stereo



Message update schedule: “left-right-up-down”

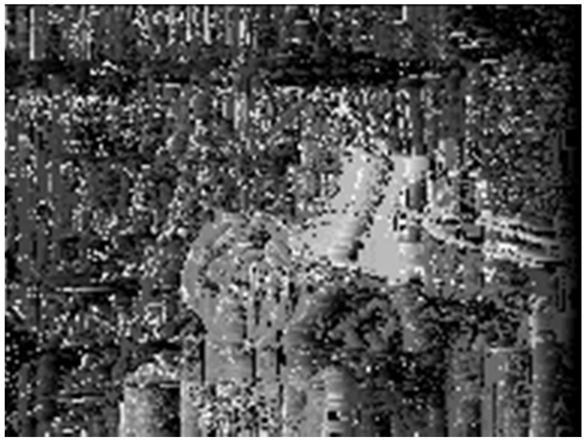
“Left” means an entire sweep that updates messages from all pixels to their left neighbors, etc.

One iteration consists of a sweep left, then right, then up, then down.

BP applied to stereo

Winning disparities shown by grayscale levels (lighter pixels have higher estimated disparity)

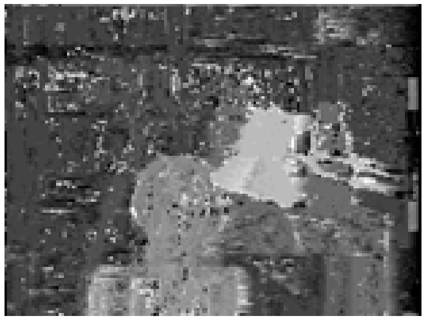
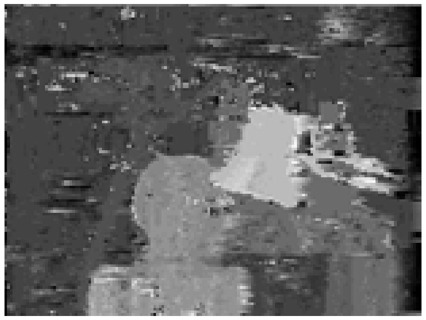
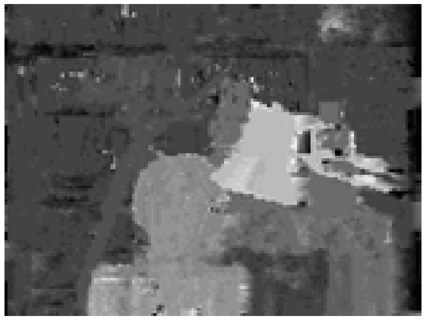
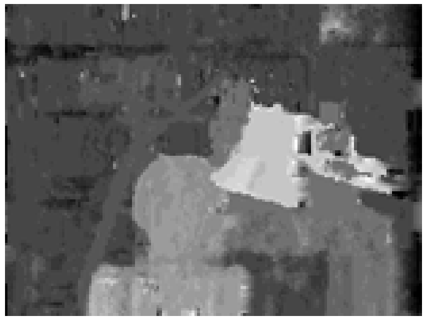
Before BP (i.e. disparities estimated solely by pixel-wise evidence):



BP applied to stereo

First iteration: disparities shown after

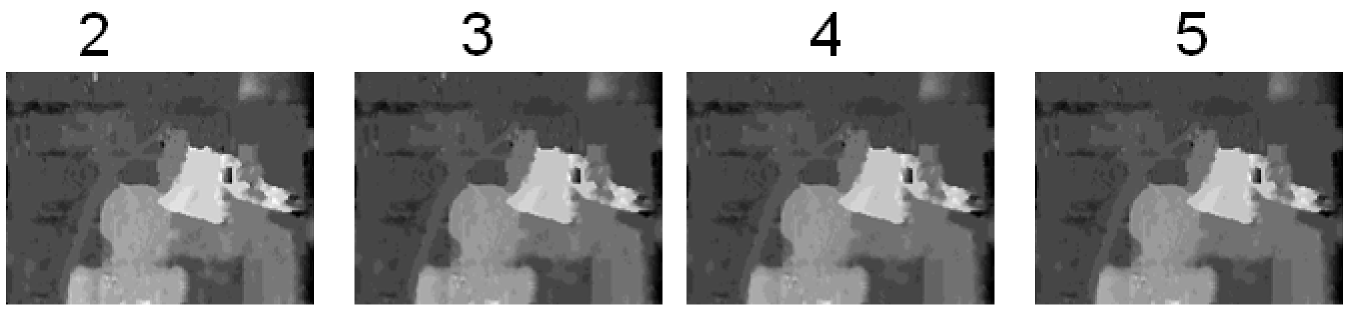
left, right, up, down sweeps

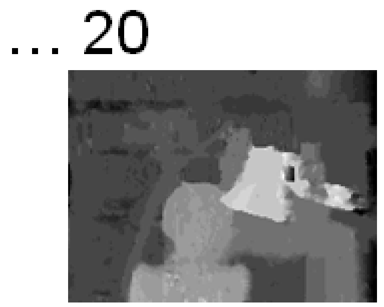





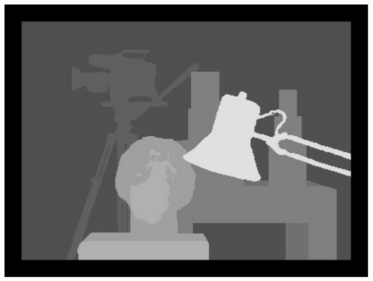
Noticeable “streaking” after left and right sweeps, mostly erased by up sweep.

BP applied to stereo

Subsequent iterations:







Ground truth disparity

Note:

Little change after first few iterations.

Model can be improved to give better results -- this is just a simple example to illustrate BP.

Improving BP stereo

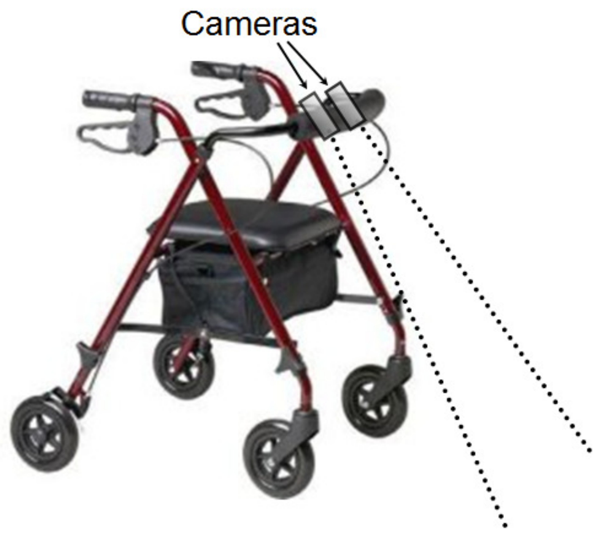
Problem: to find subtle deviations from a ground plane (e.g., a curb viewed from a distance), which could be obscured by noisy 3D stereo reconstruction.

Approach: use the most appropriate prior (smoothness) constraint for the application.

Standard smoothness constraint penalizes any change in disparity between neighboring pixels. (Fronto-parallel bias.)

Improving BP stereo

But, if camera is at an angle w.r.t. ground, even a perfectly flat ground plane will not be perfectly "smooth," since disparity varies over ground.



Improving BP stereo

Solution: re-formulate smoothness prior to fit problem domain.

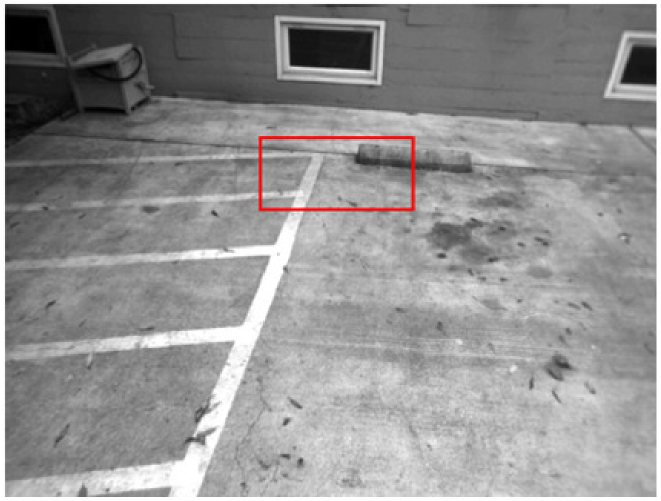
The quantity that is perfectly smooth for a perfectly flat ground plane is the *elevation* (above the ground) at each pixel.

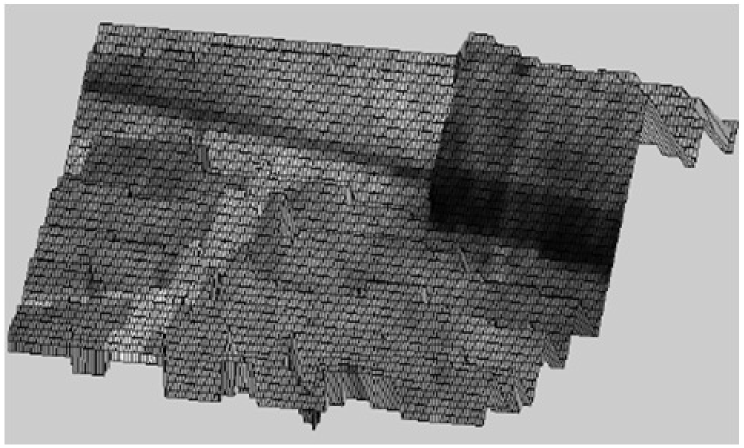
Convert disparity to elevation. (Given a pixel location and an elevation value, the corresponding disparity is easily calculated given knowledge of dominant ground plane.)

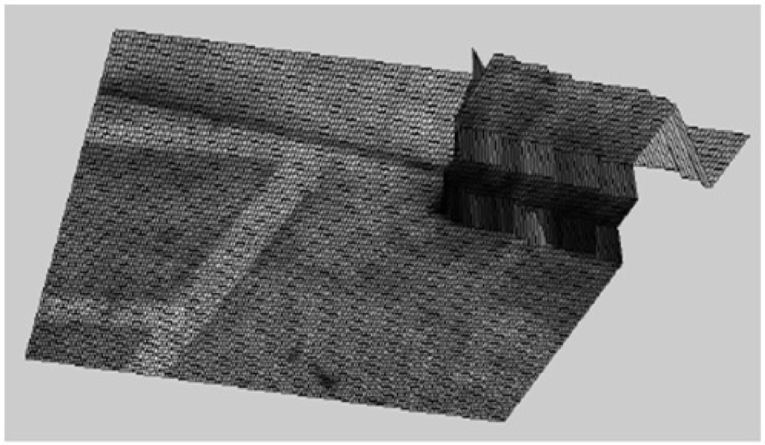
Stereo MRF variables to infer are then elevation variables, and smoothness is imposed on elevation instead of on disparity.

Improving BP stereo

Red rect. = region of interest







Disparity smoothness

Elevation smoothness

Aside: energy functions and Bayesian inference

Bayes theorem: $P(x|d) = P(x)\, P(d|x)/P(d)$

This is just a re-statement of conditional probability, since for any variables *a, b* we define

$P(a|b) = P(a,b)/P(b)$

Aside: energy functions and Bayesian inference

Terms:

$P(x)$ is the “prior” on *x*
$P(d|x)$ is the “likelihood” function
$P(x|d)$ is the “posterior” function

Maximum a posterior (MAP) estimate of *x* is the value that maximizes *$P(x|d)$* given *d:* the most likely interpretation given the data. The MAP value is the same as the value that maximizes *$P(x)\ P(d|x)$.*

Connection with energy function:

MAP can also be obtained by maximizing *$E(x) = U(x) + L(x,d)$*
where *$U(x) = \log(P(x))$* and *$L(x,d) = \log(P(d|x))$*

Aside: energy functions and Bayesian inference

Advantages of Bayesian framework:

- Prior and likelihood can be learned from training data, rather than just guessing functions U(x) and L(x,d)
- If prior and likelihood are correctly learned (exactly), the Bayesian model is optimal: it can be proved that *there is no better way to do inference!*
- Provides probabilistic answers, such as "how often does image intensity difference between neighboring pixels in image exceed the value 10?"

Disadvantages:

- Given finite amount of training data, you still have to *guess* the *form* of the prior and likelihood functions, even if the function *parameters* are learned from data
- If training data is insufficient, Bayesian model may lead to worse inferences than other techniques
- Calculations may be too burdensome in practice even if they are straightforward on paper

Summary

Energy functions: powerful, but can be hard to optimize and/or slow.

MRFs strike a good balance between power and ease of optimization.

Optimization techniques are constantly improving, and faster hardware (e.g. GPUs) is making them increasingly practical.