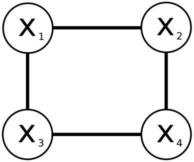
Bayesian Phase Unwrapping with Factor Graphs

Eric Jonas

6.556 Final Project

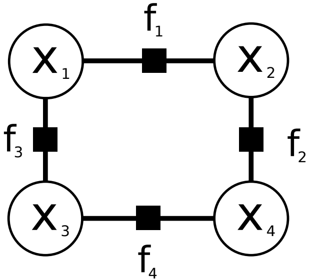
May 12, 2009

Markov Random Fields



Markov random field, undirected graphical model, etc.

Factor Graphs



- Factor Graphs [?] express the same concepts as MRFs but make the **factors** explicit.

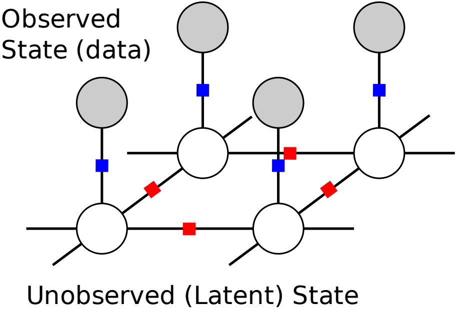
Factor Graph Probability

$$P(x_1, x_2, x_3, x_4) = f_1(x_1, x_2) \cdot f_2(x_2, x_3) \cdot f_3(x_3, x_4) \cdot f_4(x_4, x_1) \quad (1)$$

Ising Model: The original lattice MRF

Imagine you want to simulate a spin system Statistical physics people love doing this

Factor Graphs for Low-Level Vision



Properties of Image Factor Graphs [**?**]:

- Use observed state (data) to infer hidden state
- Have lattice structure like the ising
- Large number of vertices ($O(n)$ in number of pixels)
- $O(1)$ per-vertex connectivity
- Typically have homogeneous factors

Bayesian Factor Graphs

Bayes Rule

$$P(X|Y) = \frac{P(Y|X)P(X)}{\sum_X P(Y|X)P(X)}$$

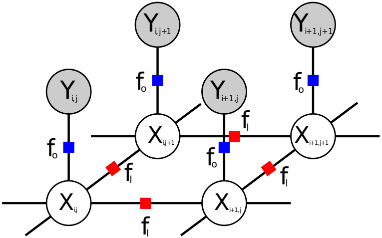
- $Y = y_{(i,j)}$: Observed Nodes
- $X = x_{(bi,j)}$: Hidden state we wish to estimate
- $P(Y|X)$: measurement model ("likelihood")
- $P(X)$: prior
-

Bayesian Factor Graphs For Low-Level vision

Our factor graph gives us P(X, Y)

MRFs for Phase: Frey's approach

In Ying and Frey's model [?] they formulate 2-D phase unwrapping as a low-level vision MRF problem.



- $Y_{(i,j)} \in \mathbb{R}$
- $x_{(i,j)} \in [0, 2\pi)$

Observation Potential

$$f_o = \delta((Y_{(i,j)} \mod 2\pi) - X_{(i,j)})$$

Latent potential

$$f_l(X_1, X_2) = (X_1 - X_2)^2$$

discrete latent state, uniform factors

discrete latent state, unique factors

My formulation

Inference in MRFs

- The MRF tells us how to compute $P(X, Y)$. Bayes rule tells us how to compute $P(X|Y)$. But the sum is awful.
- But it's easy to compute $P^*(Y|X)$

Two generic approaches:

- draw samples from $p(x|D)$ to empirically estimate
- optimize to find MAP solution

Inference in MRFs

- The MRF tells us how to compute $P(X, Y)$. Bayes rule tells us how to compute $P(X|Y)$. But the sum is awful.
- But it's easy to compute $P^*(Y|X)$

Two generic approaches:

- draw samples from $p(x|D)$ to empirically estimate
- optimize to find MAP solution

We focus on sampling (Why?)

Markov-Chain Monte Carlo

Markov Property: next state only depends on current state

$$p(x_{t+1}|x_{1:t} = p(x_{t+1}|x_t)$$

Ergodic markov chains have stationary distributions
Set up a state space so that the asymptotic limit is the target distribution
Used in situations where you want to sample from $\pi(x)$ but only can compute $\pi^*(x)$

Metropolis Hastings

Consider a proposal distribution, $q(x \rightarrow x^*)$ We draw x^* as a *new target state* from this distribution. We compute the "Acceptance" ratio :

$$a = \min(1, \frac{p(x^*)}{p(x)} \cdot \frac{q(x \rightarrow x^*)}{q(x^* \rightarrow x)})$$

[?]

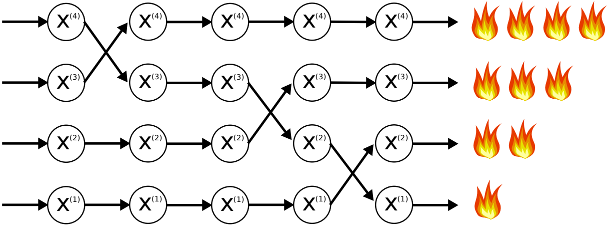
Gibbs Sampling

like MH but along an axis, useful when we can condition on other variables. Look, we can Gibbs sample in image MRFs with discrete state spaces [**?**]

Parallel Tempering

(aka "Replica Exchange Monte Carlo" [?], aka "Something to do with your 8 cores")

- Run N replicas of your chain, each at a different temperature
- Periodically propose MH-style swaps between adjacent chains
- Let's hot chains move around in flatter energy landscape



Swendsen-Wang

Work Through

Data-driven MCMC

- Any "move" is valid as long as it is reversable.
- We can cheat a little bit and construct moves based on the data to help the chains mix, without changing the target distribution

[?] Work Through

Partial Replica Exchange

[?]

MRFs and Parallelism

The conditional independence assumptions allow fine-grained parallelism

Our Implementation

use SW, etc. python, numpy, scipy, c++, boost, etc.
multithreaded

How to measure performance? I'm going to go for log-likelihood,

2-D Synthetic Data

3-D Synthetic Data

Div and Audrey

PRELUDE

Where to now?

Exact sampling using Systematic Stochsatic Search Better neighborhood connectivity / likelihood? GPU implementation Better visualization of posterior?

More information

Source is on github