Probabilistic Graphical Models

Inference

1 Overview of Inference
2 Exact Inference Methods
  2.1 Variable Elimination
    2.1.1 Algorithm Overview
    2.1.2 Time Complexity
    2.1.3 Optimal Ordering of Variable Elimination
    2.1.4 Graph-Based Perspective on Variable Elimination
  2.2 Belief Propagation
    2.2.1 General Belief Propagation
    2.2.2 Belief Propagation in Clique Trees
3 Approximate Inference Methods
  3.1 Sampling Bounds
  3.2 Markov Chain Monte Carlo
    3.2.1 Markov Chain and Stationary Distribution
    3.2.2 MCMC in Practice
    3.2.3 MCMC Instances
      3.2.3.1 Metropolis-Hastings Algorithm
      3.2.3.2 Gibbs Sampling
4 MAP Inferences
  4.1 From Marginals to MAP
  4.2 MAP by Optimization Methods

1 Overview of Inference

Having a graphical model with all parameters complete, given some of variables observed, we may care about the distributions or the most possible values of other variables. This task is called **Inference**.

There are two common types of inference: Conditional Probability Queries(Marginal Probability) and MAP.

Conditional Probability Queries:

- Evidence: $E = e$;
- Query: a subset of variables $\boldsymbol{Y}$;
- Task: compute $P(\boldsymbol{Y}|E = e)$;
- Crucial step: compute $\sum_{\boldsymbol{W}} \prod_k \phi'_k(\boldsymbol{D}'_k)$ and renormalize, where $\boldsymbol{W} = \{X_1, X_2, \ldots, X_k\} - \boldsymbol{Y} - E$ and ϕ' is ϕ reduced by $E = e$.

MAP Inference:

- Evidence: $E = e$;
- Query: all other variables $\boldsymbol{Y} = \{X_1, X_2, \ldots, X_k\} - E$;
- Task: compute $\arg\max_{\boldsymbol{y}} P(\boldsymbol{Y} = \boldsymbol{y}|E = e)$;
- Crucial step: compute $\arg\max_{\boldsymbol{y}} \prod_k \phi'_k(\boldsymbol{D}'_k)$, where $\boldsymbol{W} = \{X_1, X_2, \ldots, X_k\} - \boldsymbol{Y} - E$ and ϕ' is ϕ reduced by $E = e$.

In both tasks, exact solution or non-trivial approximation is an NP-hard problem. But in practice, there exist algorithms to do it more efficiently.

2 Exact Inference Methods

2.1 Variable Elimination

As is shown above, in a inference task, the crucial step is to compute products of factors and reduce by marginalization or maximization. This can be done by Variable Elimination.

2.1.1 Algorithm Overview

- Reduce each factor ϕ_i by evidence $E = e$, and get a new set of factors $\Phi' = \{\phi'_i\}_{i=1,2,...,k}$.
- For each non-query variable Z, eliminate Z from Φ':
  - Find the set of factors containing Z: $\Phi'' = \{\phi'_i \in \Phi' : Z \in \boldsymbol{D}_i\}$, where $\boldsymbol{D}_i$ is the scope of ϕ'_i;
  - Multiple factors in Φ'' and marginalize it. Get a new factor $\tau = \sum_Z (\prod_{\phi'_i \in \Phi''} \phi'_i)$;
  - Add τ to Φ'.
- Multiply remaining factors and renormalize to get a distribution over the queried variables.

2.1.2 Time Complexity

The time complexity mainly comes from step 2, i.e., eliminating Z from Φ'.

For each eliminating step k, we first need to calculate factor product. Assume there are m_k factors in Φ''_k , and the result $\prod_{\phi'_i \in \Phi''_k} \phi'_i$ is a factor which has N_k values. Each value is calculated by multiplying $m_k - 1$ values, so we totally need to do $(m_k - 1)N_k$ multiplications. Then we need to calculate factor marginalization, where we need to do $\mathcal{O}(N_k)$ times summations. So totally, we need $\mathcal{O}(m_k N_k)$ times operations in each iteration. Totally we need $\mathcal{O}(\sum_k m_k N_k)$ operations.

Assume we have n variables and start with m factors. Each step, we eliminate one variable, so at most we need to do n elimination iterations. And at each elimination step, we generate one new factor, so total number of factors $m^* \leq m + n$.

Set $N = \max_k(N_k)$, $\sum_k m_k N_k \leq N \sum_k m_k \leq N m^*$. So total time complexity is $\mathcal{O}(N m^*)$.

But that does not mean the time complexity is linear. Let's consider N. Assume $K = \arg\max_k N_k$, i.e. $N = N_K$. Set the number of variables of factor $\prod_{\phi'_i \in \Phi''_K} \phi'_i$ is r_K, and for each variable X_{Ki}, there are d_{Ki} values, in its scope (e.g., for binary variable, $d_{Ki} = 2$). Set $d_K = \max_i(d_{Ki})$. Then $N = N_K = \mathcal{O}({d_K}^{r_K})$.

So **main complexity comes from the largest factor** N_K.

So elimination ordering, which can change the size of the largest factor, makes a huge difference in time complexity.

2.1.3 Optimal Ordering of Variable Elimination

As discussed above, elimination ordering makes a huge difference on time complexity. We can use greedy search using heuristic cost function to do this. At each point, eliminate node with the smallest cost can give us a pretty good result. Here are some commonly used cost functions:

- min-neighbors: number of neighbors in current graph;
- min-weight: weight (number of values) of the factor formed after elimination;
- min-fill: number of new fill edges;
- weighted min-fill: total weight of new fill edges (edge weight = product of weight of the 2 nodes).

2.1.4 Graph-Based Perspective on Variable Elimination

We can also interpret VE and find optimal ordering from Graph-Based Perspective.

Induced Graph: The induced graph $I_{\Phi,\alpha}$ over factors Φ and ordering α is an undirected graph, where X_i and X_j are connected if they are in the scope of the same factor in a run of the VE algorithm using α as the ordering.

It turns out that induced graph has some properties we can take advantages of to find optimal VE ordering:

- Maximal cliques in induced graph is one-to-one corresponding to the factors produced during VE:
    - Every factor produced during VE is a maximal clique (maximal fully connected subgraph) in the induced graph.
    - Every maximal clique in the induced graph is a factor produced during VE.
- The induced graph is triangulated.

Induced Width: Define the width of an induced graph as the number of nodes in the largest clique in the graph minus 1(just by convention). Define minimal induced width of a graph over a set of factor Φ as $\min_\alpha width(I_{\Phi,\alpha})$. It provides a lower bound of the best performance of VE.

Therefore, we can find elimination ordering by finding a low-width triangulation of original graph H_Φ.

2.2 Belief Propagation

As discussed above, the crucial task in Inference is to calculate the product of factors. In VE, we successively eliminate variables and finally remains the target variables we care about. But if we now change our mind and ask about other variables, we need to redo VE again. To do it more cleverly, we can use **Belief Propagation** in a clique tree. We first discuss general Belief Propagation.

2.2.1 General Belief Propagation

Belief Propagation is executed on **Cluster Graph**. A Cluster Graph is an undirected graph such that:

- nodes are clusters $\boldsymbol{C}_i \subseteq X_1, X_2, \ldots, X_k$;
- edge between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ represents sepset $\boldsymbol{S}_{i,j} \subseteq \boldsymbol{C}_i \cup \boldsymbol{C}_j$;
- factors $\phi_k(\boldsymbol{D}_k)$ are assigned to clusters $\boldsymbol{C}_i$ s.t. $\boldsymbol{D}_k \subseteq \boldsymbol{C}_i$, define $\psi_i(\boldsymbol{C}_i) = \prod_k \phi_k$.

It turns out that a valid cluster graph needs to satisfy two properties:

- **Family Preservation**: Given a set of factors Φ, each factor in Φ need to be assigned to only one cluster which can accommodate it.
- **Running Intersection Property**: For any variable X, the set of clusters and sepsets containing X form a tree.

Having built a cluster graph, we can do BP algorithm on the graph as follows:

- Calculate initial potentials $\psi_i(\boldsymbol{C}_i)$ and initialize all messages to 1;
- Repeat until converge (if can):
    - Select edge (i, j) and pass message:

$$\delta_{i \to j}(\boldsymbol{S}_{i,j}) = \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi_i \prod_{k \in \mathcal{N}_i - \{j\}} \delta_{k \to i}$$

- Compute cluster beliefs $\beta_i(\boldsymbol{C}_i) = \psi_i \prod_{k \in \mathcal{N}_i} \delta_{k \to i}$ as pseudo-marginals, where $\mathcal{N}_i$ is the set of all clusters connected to cluster $\boldsymbol{i}$.

BP algorithm has some good properties:

- **Calibration**: If convergence, every pair of adjacent clusters agrees on their sepset. And therefore, we can define sepset belief as the product of the messages in 2 directions, i.e., $\mu_{i,j}(\boldsymbol{S}_{i,j}) = \delta_{i \to j}\delta_{j \to i} = \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{C}_i) = \sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}} \beta_j(\boldsymbol{C}_j)$.
- **Reparameterization**: After BP, we do not lose information about the joint distribution, all is just reparameterization. The unnormalized joint distribution is just the product of all clique belief divided by the product of all sepset belief.

In general, BP algorithm does not guarantee convergence and correctness. Especially, **tight loops** and **strong potentials pulling in different directions** will lead to very bad performance.

To achieve better performance, we can have two main tricks:

- Choose a good message passing ordering
- Asynchronous BP is better
- Damping
    - Smooth the message, i.e.,

$$\delta_{i \to j} = \lambda \delta_{i \to j}^{current} + (1 - \lambda)\delta_{i \to j}^{old}$$

2.2.2 Belief Propagation in Clique Trees

Belief Propagation in clique trees guarantees to converge and gives us a correct marginal.

A **Clique Tree** is a tree form Clique Graph, and $\boldsymbol{S}_{i,j} = \boldsymbol{C}_i \cup \boldsymbol{C}_j$.

In a clique tree of K clusters, we only need to pass $2(K-1)$ messages, one from root to leaves, another from leaves back to root. And after that, all marginals are stored as cluster beliefs. After that, the marginal of every set of variables can be easily calculated by multiplying and renormalizing cluster beliefs.

Just like Variable Elimination, the computation costs is hidden: T **is a valid tree satisfying RIP(Running Intersection Property)** $\iff$ **for every edge** $(i,j), P_\Phi \models \boldsymbol{W}_{<(i,j)} \perp \boldsymbol{W}_{>(i,j)} | \boldsymbol{S}_{i,j}$**.** Therefore, each sepset needs to separate P_Φ into two conditional independent parts, which may implies a very large sepset.

3 Approximate Inference Methods

Exact inference may requires us to manipulate large factors, and thus can easily be intractable. So we need to turn to approximate inference methods. Here we focus on sampling methods.

3.1 Sampling Bounds

For a binomial random variable X, when estimating the parameter of the distribution p by M samples $x_1, x_2, \ldots, x_M$, we have two inequations to quantify the precision.

Hoeffding Bound:

$$P(\hat{p} \notin [p-\epsilon, p+\epsilon]) \leq 2e^{-2M\epsilon^2}$$

Chernoff Bound:

$$P(\hat{p} \notin [p(1-\epsilon), p(1+\epsilon)]) \leq 2e^{-Mp\epsilon^2/3}$$

3.2 Markov Chain Monte Carlo

To sample efficiently, we can use MCMC Sampling framework.

3.2.1 Markov Chain and Stationary Distribution

A Markov Chain defines a probabilistic transition model $T(x \to x')$ over current states x, and given a starting distribution $P^{(0)}$, $P^{(t+1)}(X^{(t+1)} = x') = \sum_x P^{(t)}(X^{(t)} = x)T(x \to x')$.

If the distribution at some time t is equal to the distribution at last step, i.e.

$$P^{(t)}(X^{(t)} = x') \approx P^{(t+1)}(X^{(t+1)} = x') = \sum_x P^{(t)}(X^{(t)} = x)T(x \to x')$$

, the Markov Chain is in **stationary distribution**, which is often written as $\pi(x)$.

A sufficient (not necessary) condition for a Markov Chain to converge to a unique stationary distribution is **regularity**: A Markov Chain is regular if there exists k such that for every 2 states x and x', the probability of getting from x to x' in exactly k steps is positive.

A stronger property is **reversibility**: a Markov Chain is reversible, if **detailed balance** $\pi(x)T(x \to x') = \pi(x')T(x' \to x)$ holds. If a Markov Chain is reversible, then it is regular, and has a unique stationary distribution π.

So here comes the intuition: if we can design a Markov Chain whose transition probability T satisfies detailed balance with π equal to the target distribution. We can roam randomly along this Markov Chain until converge, then the sample generated at every single step follows the target distribution. This is the basic idea of MCMC.

3.2.2 MCMC in Practice

Having this basic intuition, we can perform MCMC for sampling. But there remains two question to be answered:

- How do we know the Markov Chain has converged
- How to choose T

Unfortunately, as for the first question, the answer is that we don't. But in practice, we can perform multiple tests, if non of them shows the Markov Chain has not converged, we can somehow believe it has converged.

Specifically, we can track some statistics along the time, to see if they have converged, and we can also run multiple chains from different starting points and check if these chains agree on some statistics.

3.2.3 MCMC Instances

As for the second question, we have some commonly used methods to design such a chain:

3.2.3.1 Metropolis-Hastings Algorithm

For each assignment $\boldsymbol{x}$ in our Graphical Model, we assign a state in our Markov Chain. Having this Markov Chain, we can perform MH sampling:

- First propose a proposal distribution $Q(\boldsymbol{x} \to \boldsymbol{x}')$.
- At each step, sample $\boldsymbol{x}'$ from $Q(\boldsymbol{x} \to \boldsymbol{x}')$ and calculate acceptance probability $\alpha(\boldsymbol{x} \to \boldsymbol{x}') = \min\left\{1, \frac{P(\boldsymbol{x}')Q(\boldsymbol{x}' \to \boldsymbol{x})}{P(\boldsymbol{x})Q(\boldsymbol{x} \to \boldsymbol{x}')}\right\}$. Then sample a value a from uniform distribution $U(0, 1)$. If $a < \alpha$, we move to $\boldsymbol{x}'$; otherwise, we stay at $\boldsymbol{x}$. Repeat this step until mix.
- Use the same process to generate samples after mix. Collect the samples at each step(if we stay at $\boldsymbol{x}$, we just re-collect it again), until we have enough samples.

NB. we know all parameters in our Graphical Model, so even we do not know the parametric form of α, $P(\boldsymbol{x})$, $P(\boldsymbol{x}')$ and thus $\alpha(\boldsymbol{x} \to \boldsymbol{x}')$can be easily calculated at each step.

This process means that T in our Markov Chain is:

$$T(\boldsymbol{x} \to \boldsymbol{x}') = \begin{cases} Q(\boldsymbol{x} \to \boldsymbol{x}')\alpha(\boldsymbol{x} \to \boldsymbol{x}'), & \boldsymbol{x} \neq \boldsymbol{x}'; \\ Q(\boldsymbol{x} \to \boldsymbol{x}') + \sum_{\boldsymbol{x} \to \boldsymbol{x}'} Q(\boldsymbol{x} \to \boldsymbol{x}')(1 - \alpha(\boldsymbol{x} \to \boldsymbol{x}')), & \boldsymbol{x} = \boldsymbol{x}'. \end{cases}$$

And we can prove Metropolis-Hastings Sampling satisfies detailed balance:

- For the case $\boldsymbol{x} \neq \boldsymbol{x}'$,

$$
\begin{aligned}
\pi(\boldsymbol{x})T(\boldsymbol{x} \to \boldsymbol{x}') &= P(\boldsymbol{x})Q(\boldsymbol{x} \to \boldsymbol{x}') \min\Big\{1, \frac{P(\boldsymbol{x}')Q(\boldsymbol{x}' \to \boldsymbol{x})}{P(\boldsymbol{x})Q(\boldsymbol{x} \to \boldsymbol{x}')}\Big\} \\
&= \min\Big\{P(\boldsymbol{x})Q(\boldsymbol{x} \to \boldsymbol{x}'), P(\boldsymbol{x}')Q(\boldsymbol{x}' \to \boldsymbol{x})\Big\} \\
&= P(\boldsymbol{x}')Q(\boldsymbol{x}' \to \boldsymbol{x}) \min\Big\{\frac{P(\boldsymbol{x})Q(\boldsymbol{x} \to \boldsymbol{x}')}{P(\boldsymbol{x}')Q(\boldsymbol{x}' \to \boldsymbol{x})}, 1\Big\} \\
&= \pi(\boldsymbol{x}')T(\boldsymbol{x}' \to \boldsymbol{x})
\end{aligned}
$$

- For the case $\boldsymbol{x} = \boldsymbol{x}'$, the proof is trivial.

3.2.3.2 Gibbs Sampling

A variation of MH sampling is a method called Gibbs Sampling, where
$Q(\boldsymbol{x} \to \boldsymbol{x}') = P(x_k'|\boldsymbol{x}_{-k})$, and thus
$\alpha = \min\{1, \frac{P(x_k', \boldsymbol{x}_{-k})P(x_k|\boldsymbol{x}_{-k})}{P(x_k, \boldsymbol{x}_{-k})P(x_k'|\boldsymbol{x}_{-k})}\} = \min\{1, \frac{P(x_k'|\boldsymbol{x}_{-k})P(x_k|\boldsymbol{x}_{-k})P(\boldsymbol{x}_{-k})}{P(x_k|\boldsymbol{x}_{-k})P(x_k'|\boldsymbol{x}_{-k})P(\boldsymbol{x}_{-k})}\} = 1$, which means we accept every sample.

The sampling procedure is as follows:

- At each step, for $i = 1, 2, \ldots, n$, sample x_i from $P(X_i|\boldsymbol{x}_{-i})$ and update the x_i value in $\boldsymbol{x}$, then after updating every dimension(node in Graphical Model), set $\boldsymbol{x}' = \boldsymbol{x}$.

$P(X_i|\boldsymbol{x}_{-i})$ is not very hard to sample, because

$$
\begin{aligned}
P(X_i|\boldsymbol{x}_{-i}) &= \frac{P(X_i, \boldsymbol{x}_{-i})}{P(X_i)} \\
&= \frac{\tilde{P}(X_i, \boldsymbol{x}_{-i})}{\tilde{P}(X_i)} \\
&\propto \prod_{j: X_i \in \boldsymbol{D}_j} \phi_j(\boldsymbol{D}_j)
\end{aligned}
$$

, which means we only multiply the factors that involve X_i.

But it turns out the Markov Chain designed in Gibbs Sampling is not always regular, because when derive the detailed balance, we implicitly require all factors in P to be non-zero. So only when all factors are positive, Gibbs chain is regular.

Some exception is XOR Bayesian network, where Y is a deterministic function of X_1 and X_2. In this case, e.g., $P(Y = 1|X_1 = 1, X_2 = 1) = 0$, thus not regular.

NB. In MH Sampling, no matter we move to a new $\boldsymbol{x}$ or not, we collect it as a new sample, even it maybe just the same as the last sample.

In Gibbs Sampling, by convention, even if we get a new sample which meets the detailed balance after just updating one single dimension, we only collect a new sample when every dimension has been updated.

4 MAP Inferences

4.1 From Marginals to MAP

In both VE and BP algorithm, just by replacing summations by maximizations, (and to avoid numerical underflow, we usually also take the logarithm of the probabilities, and replacing multiplications by summations), we convert marginal solutions to max-posterior solutions. And then the only thing we need to do is to find the MAP assignments.

In Clique Tree, if the assignment is unique, it is easy because we can just get the unique assignment from each clique. But if not, we have two options:

- Perturb parameters slightly to make MAP unique;
- Use traceback procedure that incrementally finds an MAP assignment.

4.2 MAP by Optimization Methods

Finding an MAP assignment, unlike marginals where we need to consider the distribution over the entire space, what we really care about is just the optimal assignment that maximizes the posterior probability. Hence finding MAP assignment is also an optimization task and we can use optimization methods like Dual Decomposition to solve it.