

Recitation 4: Inference – MAP, Loopy BP

Teaching Assistant: Eitan Richardson

Recommended reading:

- PGM Book, Chapters 11, 13 [1]

4.1 Reminder - Variable Elimination for Probability Queries

In class we saw how to perform probability queries such as:

- $P(\mathbf{Y})$, $\mathbf{Y} \subset \mathcal{X}$ – Marginal probability distribution
- $P(Y | \mathbf{E} = e)$, $\mathcal{X} = \mathbf{X} \cup \mathbf{Y} \cup \mathbf{E}$ – (Marginal) Posterior probability distribution
- $P(\mathbf{E} = e)$, $\mathbf{E} \subseteq \mathcal{X}$ – (Marginal) Likelihood of an observation (output is a scalar)

All marginal queries require summing-out a subset of variables, an operation that if done naively is exponential.

We saw in class how in some cases (e.g. chains, trees) we can exploit the graph structure to reduce the number of computations. We defined the *Sum-Product Variable Elimination* algorithm – an exact inference algorithm that eliminates (sums-out) variables one at a time in a given order. The computational cost depends on the original factors and selected elimination order.

We then defined the *clique tree* (CT) structure and the properties that must hold in it (tree structure, family preserving, running-intersection). We saw how a BN or MN can be represented as a CT. We saw the Sum-product-VE (or *Belief Propagation*) message-passing algorithm for clique trees, which outputs the marginal probability of every node – $P(C_i)$. We saw that the complexity depends on the size of the largest clique.

☞ Any graphical model can be represented as a CT, but the size of the largest clique could be high, making the solution expensive. In such cases, the approximate loopy algorithm can be used.

4.2 MAP Queries

In MAP queries, we want to find the most probable *joint assignment* to a set of random variables, usually given some evidence.

$$\text{MAP}(\mathbf{W} | e) = \arg \max_w P(w | e) = \arg \max_w P(w, e), \quad \mathcal{X} = W \cup \mathbf{E}$$

☞ Moving from conditional to joint is possible because we are only interested in the argmax.

In *marginal* MAP queries, we want to find the best joint assignment for only a subset:

$$\text{MAP}(\mathbf{Y} | e) = \arg \max_y P(y | e) = \arg \max_y \sum_z P(y, z | e), \quad \mathcal{X} = \mathbf{X} \cup \mathbf{Y} \cup \mathbf{E}$$

¹Original LaTeX template courtesy of UC Berkeley.

☞ *Not to be confused with MAP estimate – Maximum a posteriori estimation, where we estimate the most probable model parameters.*

Examples:

- Find the most probable combination of causes explaining a set of symptoms.
- Decoding a message (series of bits) over a noisy channel – we want to output a single (most probable) assignment

4.2.1 Performing MAP inference

Computational complexity (in the general case) is \mathcal{NP} -complete, like probability query. The graph structure can be exploited to reduce amount of calculation.

Unlike probability query, we can use the unnormalized product of factors:

$$\xi^{\text{map}} = \arg \max_{\xi} P_{\Phi}(\xi) = \arg \max_{\xi} \frac{1}{Z} \tilde{P}_{\Phi}(\xi) = \arg \max_{\xi} \tilde{P}_{\Phi}(\xi)$$

(For BNs, this is equivalent to moving from the conditional to the joint probability)

☞ *Computing the partition function is equivalent to performing marginal probability query – Sum-Product.*

The algorithm for MAP query is called Max-Product and is similar to Sum-Product.

For calculating a marginal MAP query, we perform both max and sum.

There are several algorithms for finding the maximum (optimization algorithms).

4.2.2 Max-Product Variable Elimination

In order to find the arg max we first need to find the max:

Example 4.1 *BN $A \rightarrow B$*

$$\begin{aligned} \max_{a,b} P(a,b) &= \max_{a,b} [P(a)P(a,b)] \\ &= \max_a \left[\max_b [P(a)P(a,b)] \right] \\ &= \max_a \left[P(a) \max_b P(a,b) \right] \\ &= \max_a [P(a)\phi(a)] \end{aligned}$$

Like in sum-product, we can push the max operation forward. We define new factors that are the result of the *factor maximization* operations.

Definition 4.2 *Factor Maximization* (max-out a variable from a factor and create a new factor w/o it)

$$\psi(\mathbf{X}) = \max_y \phi(\mathbf{X}, y), \quad Y \notin \mathbf{X}$$

Useful (trivial) properties: When $X \notin \text{Scope}(\phi_1)$

$$\max_X(\phi_1 \cdot \phi_2) = \phi_1 \max_X \phi_2$$

$$\max_X(\phi_1 + \phi_2) = \phi_1 + \max_X \phi_2$$

Find the differences...

Algorithm 1 Sum-Product Variable Elimination (short version)

```

1: procedure SUM-PRODUCT-VE( $\Phi, Z_1, \dots, Z_k$ )
2:   for  $i = 1 \dots k$  do
3:      $\Phi = \text{SUM-PRODUCT-ELIMINATE-VAR}(\Phi, Z_i)$ 
4:   return  $\prod_{\phi \in \Phi} \phi$ 
5: procedure SUM-PRODUCT-ELIMINATE-VAR( $\Phi, Z$ )
6:    $\Phi_Z = \{\phi \in \Phi, Z \in \text{Scope}(\phi)\}$ 
7:    $\phi_Z = \prod_{\phi \in \Phi_Z} \phi$ 
8:    $\tau_Z = \sum_Z \phi_Z$ 
9:   return  $(\Phi - \Phi_Z) \cup \tau_Z$ 

```

Algorithm 2 Max-Product Variable Elimination (short version)

```

1: procedure MAX-PRODUCT-VE( $\Phi, X_1, \dots, X_k$ )
2:   for  $i = 1 \dots k$  do
3:      $\Phi, \phi_{X_i} = \text{MAX-PRODUCT-ELIMINATE-VAR}(\Phi, X_i)$ 
4:    $x^* = \text{TRACEBACK-MAP}(\phi_{X_1} \dots \phi_{X_k})$ 
5:   return  $x^*, \Phi$ 
6: procedure MAX-PRODUCT-ELIMINATE-VAR( $\Phi, X$ )
7:    $\Phi_X = \{\phi \in \Phi, X \in \text{Scope}(\phi)\}$ 
8:    $\phi_X = \prod_{\phi \in \Phi_X} \phi$ 
9:    $\tau_X = \max_x \phi_X$ 
10:  return  $(\Phi - \Phi_X) \cup \tau_X, \phi_X$ 
11: procedure TRACEBACK-MAP( $\phi_{X_1} \dots \phi_{X_k}$ )
12:  for  $i = k \dots 1$  do
13:     $x_i^* = \arg \max \phi_{X_i}(x_i, x_{i+1}^* \dots x_k^*)$ 

```

Example 4.3 *The Stopped Car (Figure 4.1)*

Lets calculate the MAP assignment for all variables: $\text{MAP}(M, F, S)$

Our factors are: $\phi_1(M) = P(M)$, $\phi_2(F) = P(F)$, $\phi_3(S, M, F) = P(S | M, F)$

The algorithm input (with some order): $\Phi = \{\phi_1, \phi_2, \phi_3\}, (S, M, F)$

Run:

1. Eliminating S :

$$\Phi_S = \phi_3, \phi_S = \phi_3, \tau_S =$$

| M | F | $\max_s(\phi_S)$ |
|-----|-----|------------------|
| 0 | 0 | 1 |
| 0 | 1 | 0.5 |
| 1 | 0 | 0.8 |
| 1 | 1 | 0.9 |

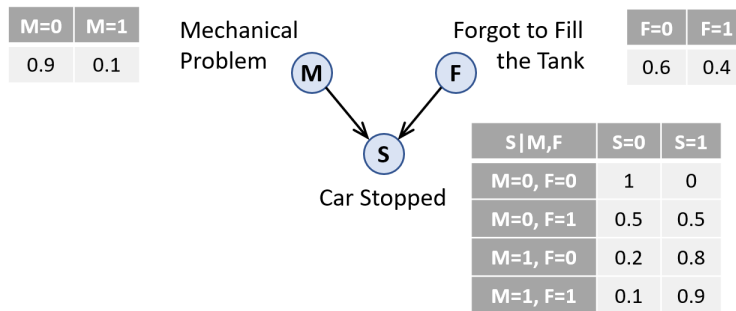


Figure 4.1: Bayesian Network example – *The Stopped Car*

2. Eliminating M :

$$\Phi_M = \{\phi_1, \tau_S\}, \phi_M = \begin{array}{c|c|c} M & F & \phi_1 \cdot \tau_S \\ \hline 0 & 0 & 0.9 \\ 0 & 1 & 0.45 \\ 1 & 0 & 0.08 \\ 1 & 1 & 0.09 \end{array}, \tau_M = \begin{array}{c|c} F & \max_m(\phi_M) \\ \hline 0 & 0.9 \\ 1 & 0.45 \end{array}$$

3. Eliminating F :

$$\Phi_F = \{\phi_2, \tau_M\}, \phi_F = \begin{array}{c|c} F & \phi_2 \cdot \tau_M \\ \hline 0 & 0.54 \\ 1 & 0.18 \end{array}, \tau_F = \max_f \phi_F = 0.54$$

Traceback (F, M, S):

$$f^* = \arg \max_f \phi_F(f) = f^0$$

$$m^* = \arg \max_m \phi_M(m, f^0) = m^0$$

$$s^* = \arg \max_s \phi_S(s, m^0, f^0) = s^0$$

☞ Since our factors were normalized, the final Φ is the probability of the MAP assignment – 0.54 in our example.

4.2.3 MAP query conditioned on evidence

Let's calculate $\text{MAP}(M, F | S = 1)$:

Definition 4.4 *Reduced Factor*

A factor ϕ with scope Y , reduced to an assignment (context) of a subset of its scope $U = u$, $U \subseteq Y$, is a new factor denoted $\phi[u]$ with scope $Y' = Y - U$, such that: $\phi[u](y') = \phi(y', u)$

☞ For BNs, to re-normalize (not required for the MAP task), we need to divide the new factor by $P(u)$.

The algorithm input (with some order): $\Phi = \{\phi_1(M), \phi_2(F), \phi_3[s^1](M, F)\}, (M, F)$

1. Eliminating M :

$$\Phi_M = \{\phi_1, \phi_3[s^1]\}, \phi_M = \begin{array}{c|c|c} M & F & \phi_1 \cdot \phi_3[s^1] \\ \hline 0 & 0 & 0 \\ 0 & 1 & 0.45 \\ 1 & 0 & 0.08 \\ 1 & 1 & 0.09 \end{array}, \tau_M = \begin{array}{c|c} F & \max_m(\phi_M) \\ \hline 0 & 0.08 \\ 1 & 0.45 \end{array}$$

2. Eliminating F :

$$\Phi_F = \{\phi_2, \tau_M\}, \phi_F = \begin{array}{c|c} F & \phi_2 \cdot \tau_M \\ \hline 0 & 0.048 \\ 1 & 0.18 \end{array}, \tau_F = \max_f \phi_F = 0.18$$

Traceback (F, M):

$$f^* = \arg \max_f \phi_F(f) = f^1$$

$$m^* = \arg \max_m \phi_M(m, f^0) = m^0$$

Given evidence that our car stopped, the most probable explanation is that we didn't fill the tank (f^1) and we have no mechanical problem (m^0).

4.2.4 MAP Inference using Graph Cuts


For a special class of MNs, there is a polynomial-time algorithm for exact MAP queries – *graph cut*.

Definition 4.5 Graph Min-Cut

Given a set of nodes \mathcal{Z} , two additional special nodes s and t and directed edges \mathcal{E} associated with non-negative weight $w(z_1, z_2)$, we define a valid graph cut as a disjoint partition of the nodes $\mathcal{Z} + \{s, t\} = \mathcal{Z}_s \cup \mathcal{Z}_t$ such that $s \in \mathcal{Z}_s$ and $t \in \mathcal{Z}_t$. The cut cost is defined as:

$$\text{cost}(\mathcal{Z}_s, \mathcal{Z}_t) = \sum_{z_1 \in \mathcal{Z}_s, z_2 \in \mathcal{Z}_t, (z_1, z_2) \in \mathcal{E}} w(z_1, z_2)$$

The graph minimal cut is the partition $\mathcal{Z}_s, \mathcal{Z}_t$ with the minimal cost.

 There is a known algorithm for finding the min cut in polynomial time.

What class of MNs can be (trivially) reduced to a min-cut problem?

- Pairwise (i.e. factors over single and pairs of variables)
- Binary variables
- Factors defined as log-space energy functions with zero energy on equal-valued pairs:

$$E(x_1, \dots, x_n) = \sum_i \epsilon_i(x_i) + \sum_{(i,j) \in \mathcal{E}} \epsilon_{i,j}(x_i, x_j)$$

$$\epsilon_{i,j}(x_i, x_j) = \begin{cases} 0, & x_i = x_j \\ \lambda_{i,j} \geq 0, & x_i \neq x_j \end{cases}$$

Reminder:

$$P_{\Phi}(X_1, \dots, X_n) = \frac{1}{Z} e^{-E(X_1, \dots, X_n)}$$

How do we construct the graph?

- A nodes for each random variables
- s indicates 0 assignment and t indicates 1 assignment
- Each variable connected either to s or t with cost ϵ_i (the other cost is assumed 0).
- Each pair of nodes are connected with edges with weight $w_{i,j} = \lambda_{i,j}$

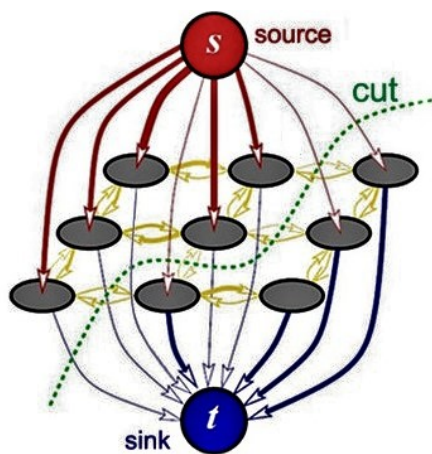


Figure 4.2: Graph cut on a pairwise regular graph – image segmentation

☞ It is easy to show that the cut cost is the energy function we want to minimize.

☞ There are extensions to additional classes e.g. non-binary discrete variables.

4.3 Loopy Belief Propagation

4.3.1 Cluster Graphs

The clique tree representation of some graphs would have a very large clique. For example, a regular pairwise MN (grid of $n \times n$ variables) will have a cluster of size n , making the computation cost exponential in n .

To avoid this, we define a structure with relaxed requirements – the *cluster graph* (CG) and a suitable variant of the BP message-passing algorithm.

Definition 4.6 Cluster Graph (For Loopy BP)

An undirected graph (not necessarily a tree) where each node C_i represents a set of RVs.

Each edge $C_i - C_j$ represents a sepset $S_{ij} \subseteq C_i \cap C_j$ (note the difference from CT sepsets)

Each factor is a subset of at least one cluster $\forall j \exists i$ s.t. $\phi_j \subseteq C_i$ – family preserving.

Whenever an RV X is in two clusters C_i, C_j , there exists a single path $C_i - \dots - C_j$ in which X is in every node and every edge (sepset) along the path – running-intersection for CGs.

☞ *The nodes and edges containing each RV form a tree.*

☞ *Two clusters can be connected by two different paths (i.e. a loop), but each path will be associated with a different set of variables.*

☞ *Like with CTs, there might be many valid CGs for a given graphical model.*

4.3.2 Loopy BP on Cluster Graphs

The BP message-passing algorithm we saw for clique trees (BP-CT) is applicable to cluster graphs with the following changes:

- Scheduling: In BP-CT, a node can send a message once it received all incoming messages. Because of this, we started from the leaves. In CGs, because of the loops, we need another initialization solution (otherwise we'll have deadlocks). The solution: Each node sends initial all-one messages.
- Convergence: We run the process (select an edge each time) for several iteration.

At the end of the process, the beliefs are an **approximation** of the marginal probabilities (see Fig 11.2 in the book).

☞ *As we run more iteration, the effect of the original factors increase, but the marginal probabilities converge to some value.*

Concerns:

- Does the algorithm always converge? (No guarantee)
- How do we know when to stop?
- Error bounds?

☞ *Different CG representations of the same model might output a different answer.*

There are several practical methods to speed-up convergence and/or reduce risk of non-convergence:

- Ignore specific beliefs that didn't converge
- Damping – average the new message with the previous one (reduces chance for oscillation)
- Smart scheduling, for example, send more messages in areas where there is more disagreement

☞ *An alternative solution is to treat the problem as an optimization problem and solve it using methods such as gradient descent.*

References

- [1] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.