Probabilistic Graphical Models

Raquel Urtasun and Tamir Hazan

TTI Chicago

April 22, 2011

This week we saw...

- Variable elimination algorithm can be used to compute $P(\mathbf{Y})$, $P(\mathbf{Y}, \mathbf{e})$ and $P(\mathbf{Y}|\mathbf{e})$.
- Finding the optimal ordering for VE is NP-hard.
- For chordal graphs, we can construct an optimal ordering via de clique tree or the max-cardinality algorithm.

Algorithm 9.3 Maximum Cardinality Algorithm for constructing an elimination ordering

```
Procedure Max-Cardinality (
    H    // An undirected graph over X
  )
  Initialize all nodes in X as unmarked
  for k = |X| ... 1
    X ← unmarked variable in X with largest number of marked neighbors
    π(X) ← k
    Mark X
  return π
```

- If the graph is non-chordal, then we can use heuristics (i.e., width, weight)

Conditioning Algorithm

- It uses the fact that observing certain variables can simplify the elimination process.
- If the variable was not observed, we can
  - Use a case analysis to enumerate all the possibilities.
  - Perform the VE on the simplified graphs.
  - Aggregate the results for the different values.
- This offers no advantage with respect to the VE algorithm in terms of operations.
- But it does in terms of memory.
- Time-Space tradeoff.

More formally...

- Let's consider the case of a Markov network.
- Let Φ be the set of factors over $\mathcal{X}$ and P_ϕ the associated distribution.
- All the observations already assimilated in Φ.
- The goal is to compute $P_\Phi(\mathbf{Y})$ for some query $\mathbf{Y}$.
- Let $\mathbf{U} \subseteq \mathcal{X}$ be a set of variables, then

$$\hat{P}_\Phi(\mathbf{Y}) = \sum_{\mathbf{y} \in Val(\mathbf{U})} \hat{P}_\Phi(\mathbf{Y}, \mathbf{u})$$

- Each term $\hat{P}_\Phi(\mathbf{Y}, \mathbf{u})$ can be computed by marginalizing out $\mathcal{X} - \mathbf{U} - \mathbf{Y}$ in the unnormalized measure $\hat{P}_\Phi[\mathbf{u}]$.
- The reduce measure is obtained by reducing the factors to the context $\mathbf{u}$.
- The reduced process has smaller cost in general.

Conditioning Algorithm

- We construct a network $\mathcal{H}_\Phi[\mathbf{u}]$ for each assignment $\mathbf{u}$.
- Note that these networks have the same structure, but different parameters.
- Run sum-product VE for each of the networks.
- Sum the results to obtain $\hat{P}_\Phi(\mathbf{Y})$.

Algorithm 9.5 Conditioning algorithm

Procedure Sum-Product-Conditioning (
  Φ, // Set of factors, possibly reduced by evidence
  $\boldsymbol{Y}$, // Set of query variables
  $\boldsymbol{U}$ // Set of variables on which to condition
)
  for each $\boldsymbol{u} \in Val(\boldsymbol{U})$
    $\Phi_{\boldsymbol{u}} \leftarrow \{\phi[\boldsymbol{U} = \boldsymbol{u}] \; : \; \phi \in \Phi\}$
    Construct $\mathcal{H}_{\Phi_{\boldsymbol{u}}}$
    $(\alpha_{\boldsymbol{u}}, \phi_{\boldsymbol{u}}(\boldsymbol{Y})) \leftarrow$ Cond-Prob-VE$(\mathcal{H}_{\Phi_{\boldsymbol{u}}}, \boldsymbol{Y}, \emptyset)$
  $\phi^*(\boldsymbol{Y}) \leftarrow \frac{\sum_{\boldsymbol{u}} \phi_{\boldsymbol{u}}(\boldsymbol{Y})}{\sum_{\boldsymbol{u}} \alpha_{\boldsymbol{u}}}$
  Return $\phi^*(\boldsymbol{Y})$

Sum-product VE for conditional distributions

Algorithm 9.2 **Using** Sum-Product-Variable-Elimination **for computing conditional probabilities.**

Procedure Cond-Prob-VE (
  $\mathcal{K}$, // A network over $\mathcal{X}$
  $\boldsymbol{Y}$, // Set of query variables
  $\boldsymbol{E} = \boldsymbol{e}$ // Evidence
)

  $\Phi \leftarrow$ Factors parameterizing $\mathcal{K}$
  Replace each $\phi \in \Phi$ by $\phi[\boldsymbol{E} = \boldsymbol{e}]$
  Select an elimination ordering $\prec$
  $\boldsymbol{Z} \leftarrow \; = \mathcal{X} - \boldsymbol{Y} - \boldsymbol{E}$
  $\phi^* \leftarrow$ Sum-Product-Variable-Elimination($\Phi, \prec, \boldsymbol{Z}$)
  $\alpha \leftarrow \sum_{\boldsymbol{y} \in Val(\boldsymbol{Y})} \phi^*(\boldsymbol{y})$
  return α, ϕ^*

Conditioning Algorithm

- We construct a network $\mathcal{H}_\Phi[\mathbf{u}]$ for each assignment $\mathbf{u}$.
- Note that these networks have the same structure, but different parameters.
- Run sum-product VE for each of the networks.
- Sum the results to obtain $\hat{P}_\Phi(\mathbf{Y})$.

Algorithm 9.5 Conditioning algorithm

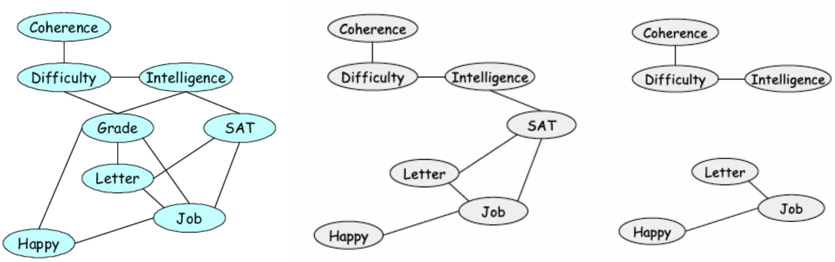
```
Procedure Sum-Product-Conditioning (
    Φ,   // Set of factors, possibly reduced by evidence
    Y,   // Set of query variables
    U    // Set of variables on which to condition
)
  for each u ∈ Val(U)
    Φ_u ← {φ[U = u] : φ ∈ Φ}
    Construct H_{Φ_u}
    (α_u, φ_u(Y)) ← Cond-Prob-VE(H_{Φ_u}, Y, ∅)
  φ*(Y) ← Σ_u φ_u(Y) / Σ_u α_u
  Return φ*(Y)
```

Normalization and an Example

- To get $P_\Phi(\mathbf{Y})$ we have to normalize.
- The partition function is the sum of partition functions

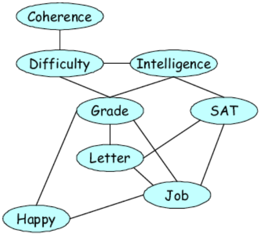
$$Z_\Phi = \sum_{\mathbf{u}} Z_{\Phi[\mathbf{u}]}$$

- We want to obtain $P(J)$ with evidence $G = g^1$.
- Apply the conditioning algorithm to S.



Complexity of conditioning

- It seems more efficient than VE, i.e., smaller network.
- However, we need to compute multiple VE, one for each $\mathbf{u} \in Val(\mathbf{U})$.

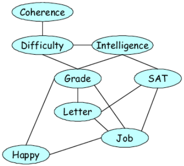


- We know that $P(J) = \sum_{C,D,I,S,G,L,H} P(C,D,I,S,G,L,H,J)$.
- Reorder $P(J) = \sum_g \left[\sum_{C,D,I,S,L,H} P(C,D,I,S,g,L,H,J)\right]$.
- The inside of the parenthesis is the $P(J)$ in the network $\mathcal{H}_{\Phi_{G=g}}$.

Conditioning vs Variable elimination

- Conditioning algorithm executes parts of the summation, enumerating all possible values of the conditioning variables.
- The VE algorithm does the same summation from the inside out using dynamic programming to reuse computation.
- **Theorem:** Let Φ be a set of factors, $\mathbf{Y}$ a query, and $\mathbf{U}$ a set of conditioning variables with $\mathbf{Z} = \mathcal{X} - \mathbf{Y} - \mathbf{U}$. Let $\prec$ be an elimination ordering over $\mathbf{Z}$ used by the VE over the network $\mathcal{H}_{\Phi_\mathbf{u}}$ in the conditioning algorithm. Let $\prec^+$ be an ordering consistent with $\prec$ over the variables $\mathbf{Z}$, and where for each $U \in \mathbf{U}$ we have $\mathbf{Z} \prec^+ U$. Then the number of operations performed by the conditioning is no less than the number of operations performed by VE with ordering $\prec^+$.
- Conditioning never performs less operations than VE, as it uses sums and products but it does not reuse the computation for the conditioning variables.
- If $\prec$ and $\prec^+$ are not consistent, we cannot say anything.

Example



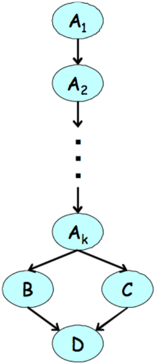
Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C^+(C,G)$, $\phi_D^+(D,C,G)$	C,D,G	$\tau_1(D,G)$
2	D	$\phi_G^+(G,I,D)$, $\tau_1(D,G)$	G,I,D	$\tau_2(G,I)$
3	I	$\phi_I^+(I,G)$, $\phi_S^+(S,I,G)$, $\tau_2(G,I)$	G,S,I	$\tau_3(G,S)$
4	H	$\phi_H^+(H,G,J)$	H,G,J	$\tau_4(G,J)$
5	S	$\tau_3(G,S)$, $\phi_J^+(J,L,S,G)$	J,L,S,G	$\tau_5(J,L,G)$
6	L	$\tau_5(J,L,G), \phi_L^+(L,G)$	J,L	$\tau_6(J)$
7	—	$\tau_6(J), \tau_4(G,J)$	G,J	$\tau_7(G,J)$

- Elimination ordering is $\{C, D, I, H, S, L\}$ and conditioning on G

Step	Variable eliminated	Factors used	Variables involved	New factor
1	C	$\phi_C(C)$, $\phi_D(D,C)$	C,D	$\tau_1(D)$
2	D	$\phi_G(G,I,D)$, $\tau_1(D)$	G,I,D	$\tau_2(G,I)$
3	I	$\phi_I(I)$, $\phi_S^+(S,I)$, $\tau_2(G,I)$	G,S,I	$\tau_3(G,S)$
4	H	$\phi_H^+(H,G,J)$	H,G,J	$\tau_4(G,J)$
5	S	$\tau_3(G,S)$, $\phi_J^+(J,L,S)$	J,L,S,G	$\tau_5(J,L,G)$
6	L	$\tau_5(J,L,G), \phi_L(L,G)$	J,L	$\tau_6(J)$
7	G	$\tau_6(J), \tau_4(G,J)$	G,J	$\tau_7(J)$

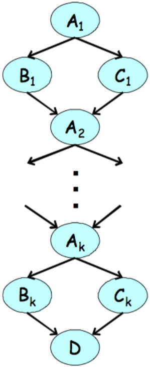
- Run of $P(J)$ with G eliminated last.
- We have defined ϕ^+ the augmented factor that contains G in the scope.

More examples



- If we conditioned on A_k in order to cut the loop, we will perform he entire elimination of the chain $A_1 \rightarrow \cdots \rightarrow A_{k-1}$ one time for each value of A_k.

More examples



- We want to cut every other A_i, e.g., $A_2, A_4, \cdots$.
- The cost of conditioning is exponential in k.
- The induced width (i.e., number of nodes in the largest clique -1) of the network is 2.
- The cost of VE is linear in k.

Conditioning is still useful

When is conditioning still useful?

1. VE can have very large factors that are too memory consuming in large networks. Conditioning gives a continuous time-space tradeoff.
2. Conditioning is the basis for useful approximate inference algorithms where for example we only enumerate a small set of possible $\mathbf{u} \in Val(\mathbf{U})$.

Effects in the graph

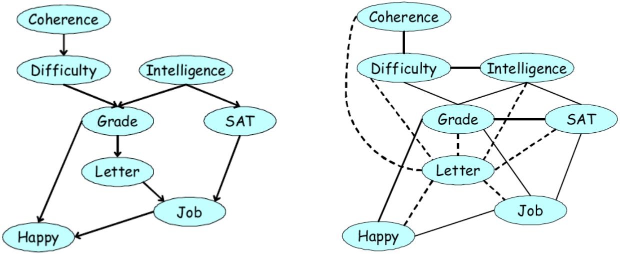
- Conditioning on U introduces U on every factor in the graph.
- We should connect U to every node in the graph.
- When eliminating U we remove it from the graph.
- We can defined an induced graph for the conditioning algorithm, which has two type of filled edges.

Induced Graph for Conditioning

Def: Let Φ be a set of factors over $\mathcal{X} = \{X_1, \cdots, X_n\}$ with $\mathbf{U} \subseteq \mathcal{X}$ conditional variables, and $\prec$ an elimination ordering over the set $\mathbf{X} \subseteq \mathcal{X} - \mathbf{U}$. The induced graph $\mathcal{I}_{\mathbf{\Phi},\prec,\mathbf{U}}$ is an undirected graph over $\mathcal{X}$ with edges.

1. A **conditioning edge** between every $U \in \mathbf{U}$ and every other variable.
2. A **factor edge** between every pair of variables $X_i, X_j \in \mathbf{X}$ that both appear in some intermediate factor ψ generated by the VE using $\prec$ as elimination ordering.

Example



(BN) (Induced Graph)

- Query: $P(J)$, we conditioned on L, and eliminate $\{C, D, I, H, G, S\}$.
- Conditioning edges shown in dashed and factor edges as regular edges.

Complexity Analysis I

Theorem: Consider conditioning algorithm to a set of factors Φ, with conditioning variables $\mathbf{U} \subset \mathcal{X}$, and elimination ordering $\prec$. Then the running time of the algorithm is $\mathcal{O}(nv^m)$, where v is a bound on the domain size of any variable and m is the size of the largest clique in the induced graph.

- Conditioning adds edges between the conditioning variables and all other variables in the graph.
- VE only does between those that are neighbors of the variable to eliminate.

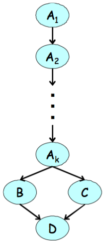
Complexity Analysis II

Theorem: Consider conditioning algorithm to a set of factors Φ, with conditioning variables $\mathbf{U} \subset \mathcal{X}$, and elimination ordering $\prec$ to eliminate $\mathbf{X} \subseteq \mathcal{X} - \mathbf{U}$. The space complexity of the algorithm is $\mathcal{O}(nv^{m_f})$, where v is a bound on the domain size of any variable and m is the size of the largest clique in the graph using only factor edges.

- For VE the space and time complexity are exponential in the size of the largest clique of the induced graph.

Improving conditioning: alternating VE and cond.

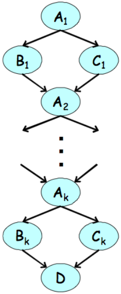
- In terms of operations, conditioning is always equal or worst than VE.
- The main problem is that computations are repeated for all values, and sometimes we do the computations multiple times, even if they are the same.



- We are interested in $P(D)$.
- Better to eliminate first $A_1, \cdots, A_{k-1}$ before conditioning on A_k.
- Then only a network involving A_k, B, C, D.
- We can then condition on any of these variables, e.g., A_k, and eliminate the others B, C.

- We first did elimination, then condition, then elimination.

Improving conditioning: network decomposition



- Efficient when conditioning splits the graph into independent pieces.
- If we conditioned on A_2 this splits the network.
- If we conditioned on A_3, there is no need to take into account the top part of the network, i.e., $\{A_1, B_1, C_1\}$, as we will repeat the same computation

- Since we partitioned the network into individual pieces, we can now perform the computation on each of them separately, and then combine the results.
- The conditioning variables used in one part will not be used in the other.
- Build an algorithm that checks wether after conditioning the graph is disconnected or not.
- If it has, then splits computation into the disjoint sets recursively.

Summary of this week

- Variable elimination algorithm can be used to compute $P(\mathbf{Y})$, $P(\mathbf{Y}, \mathbf{e})$ and $P(\mathbf{Y}|\mathbf{e})$.
- Finding the optimal ordering for VE is NP-hard.
- For chordal graphs, we can construct an optimal ordering via de clique tree or the max-cardinality algorithm.
- If the graph is non-chordal, then we can use heuristics (i.e., width, weight) in a deterministic or stochastic fashion.
- Today we saw the **conditioning algorithm**.
- Improving conditioning via alternating VE and conditioning and via network decomposition.
- Next week we will see clique trees and message passing.