Probability Propagation

Steffen Lauritzen, University of Oxford

Graphical Models, Lecture 9, Michaelmas Term 2011

November 10, 2011

Local computation algorithms have been developed with a variety of purposes. For example:

- Kalman filter and smoother
- Solving sparse linear equations;
- Decoding digital signals;
- Estimation in hidden Markov models;
- Peeling in pedigrees;
- Belief function evaluation;
- Probability propagation.

Also dynamic programming, linear programming, optimizing decisions, calculating Nash equilibria in cooperative games, and many others. *List is far from exhaustive!*

All algorithms are using, explicitly or implicitly, a *graph decomposition* and *a junction tree* or similar to make the computations.

Factorizing density on $\mathcal{X} = \times_{v \in V} \mathcal{X}_v$ with V and $\mathcal{X}_v$ finite:

$$p(x) = \prod_{C \in \mathcal{C}} \phi_C(x).$$

The *potentials* $\phi_C(x)$ depend on $x_C = (x_v, v \in C)$ only.

Basic task to calculate *marginal* probability

$$p(x_E^*) = \sum_{y_{V \setminus E}} p(x_E^*, y_{V \setminus E})$$

for $E \subseteq V$ and fixed x_E^*, *but sum has too many terms.* *A second purpose* is to get the *prediction* $p(x_v \mid x_E^*) = p(x_v, x_E^*)/p(x_E^*)$ for $v \in V$.

If the initial model is based on a DAG $\mathcal{D}$, the first step is to form the *moral graph* $\mathcal{G} = \mathcal{D}^m$, exploiting that if P factorizes w.r.t. $\mathcal{D}$, it also factorizes w.r.t. $\mathcal{D}^m$.

A very simple example

Assume

$$p(x, y, z, w) = \phi(x, y)\psi(y, z)\eta(z, w)$$

and assume each of X, Y, Z, and W have, say, 100 states.

The joint state space has thus 10^8 states, and to calculate $p(x)$ directly from $p(x, y, z, w)$ by brute force involves 10^6 terms in the sum for every x, hence 10^8 arithmetic operations are needed.

A very simple example

From

$$p(x, y, z, w) = \phi(x, y)\psi(y, z)\eta(z, w)$$

we instead may do as follows:

1. Calculate $\eta^*(z) = \sum_w \eta(z, w)$, with 10000 additions;

A very simple example

From

$$p(x, y, z, w) = \phi(x, y)\psi(y, z)\eta(z, w)$$

we instead may do as follows:

1. Calculate $\eta^*(z) = \sum_w \eta(z, w)$, with 10000 additions;
2. Calculate $\psi^*(y, z) = \psi(y, z)\eta^*(z)$ with 10000 multiplications

A very simple example

From

$$p(x, y, z, w) = \phi(x, y)\psi(y, z)\eta(z, w)$$

we instead may do as follows:

1. Calculate $\eta^*(z) = \sum_w \eta(z, w)$, with 10000 additions;
2. Calculate $\psi^*(y, z) = \psi(y, z)\eta^*(z)$ with 10000 multiplications
3. Calculate $\psi^*(y) = \sum_z \psi^*(y, z)$, with 10000 additions;

A very simple example

From

$$p(x, y, z, w) = \phi(x, y)\psi(y, z)\eta(z, w)$$

we instead may do as follows:

1. Calculate $\eta^*(z) = \sum_w \eta(z, w)$, with 10000 additions;
2. Calculate $\psi^*(y, z) = \psi(y, z)\eta^*(z)$ with 10000 multiplications
3. Calculate $\psi^*(y) = \sum_z \psi^*(y, z)$, with 10000 additions;
4. Calculate $\phi^*(x, y) = \phi(x, y)\psi^*(y)$ with 10000 multiplications;

A very simple example

From

$$p(x, y, z, w) = \phi(x, y)\psi(y, z)\eta(z, w)$$

we instead may do as follows:

1. Calculate $\eta^*(z) = \sum_w \eta(z, w)$, with 10000 additions;
2. Calculate $\psi^*(y, z) = \psi(y, z)\eta^*(z)$ with 10000 multiplications
3. Calculate $\psi^*(y) = \sum_z \psi^*(y, z)$, with 10000 additions;
4. Calculate $\phi^*(x, y) = \phi(x, y)\psi^*(y)$ with 10000 multiplications;
5. Calculate $\phi^*(x) = \sum_y \phi^*(x, y)$, with 10000 additions.

A very simple example

From

$$p(x, y, z, w) = \phi(x, y)\psi(y, z)\eta(z, w)$$

we instead may do as follows

1. Calculate $\eta^*(z) = \sum_w \eta(z, w)$, with 10000 additions;
2. Calculate $\psi^*(y, z) = \psi(y, z)\eta^*(z)$ with 10000 multiplications
3. Calculate $\psi^*(y) = \sum_z \psi^*(y, z)$, with 10000 additions;
4. Calculate $\phi^*(x, y) = \phi(x, y)\psi^*(y)$ with 10000 multiplications;
5. Calculate $\phi^*(x) = \sum_y \phi^*(x, y)$, with 10000 additions.

Now $p^*(x) = \phi^*(x)$ so *we have done this with only 50000 operations,* rather than a million.

Note we have never explicitly formed the product
$p(x, y, z, w) = \phi(x, y)\psi(y, z)\eta(z, w)$

Starting from a DAG $\mathcal{D}$, the computational structure is set up in several steps:

1. *Moralisation:* Constructing $\mathcal{D}^m$, exploiting that if P factorizes over $\mathcal{D}$, it factorizes over $\mathcal{D}^m$. Skip if starting from an undirected graph.

Starting from a DAG $\mathcal{D}$, the computational structure is set up in several steps:

1. *Moralisation:* Constructing $\mathcal{D}^m$, exploiting that if P factorizes over $\mathcal{D}$, it factorizes over $\mathcal{D}^m$. Skip if starting from an undirected graph.
2. *Triangulation:* Adding edges to find chordal graph $\tilde{\mathcal{G}}$ with $\mathcal{G} \subseteq \tilde{\mathcal{G}}$. This step is non-trivial (NP-complete) to optimize;

Starting from a DAG $\mathcal{D}$, the computational structure is set up in several steps:

1. *Moralisation:* Constructing $\mathcal{D}^m$, exploiting that if P factorizes over $\mathcal{D}$, it factorizes over $\mathcal{D}^m$. Skip if starting from an undirected graph.
2. *Triangulation:* Adding edges to find chordal graph $\tilde{\mathcal{G}}$ with $\mathcal{G} \subseteq \tilde{\mathcal{G}}$. This step is non-trivial (NP-complete) to optimize;
3. *Constructing junction tree:* Using MCS, the cliques of $\tilde{\mathcal{G}}$ are found and arranged in a junction tree.

Starting from a DAG $\mathcal{D}$, the computational structure is set up in several steps:

1. *Moralisation:* Constructing $\mathcal{D}^m$, exploiting that if P factorizes over $\mathcal{D}$, it factorizes over $\mathcal{D}^m$. Skip if starting from an undirected graph.
2. *Triangulation:* Adding edges to find chordal graph $\tilde{\mathcal{G}}$ with $\mathcal{G} \subseteq \tilde{\mathcal{G}}$. This step is non-trivial (NP-complete) to optimize;
3. *Constructing junction tree:* Using MCS, the cliques of $\tilde{\mathcal{G}}$ are found and arranged in a junction tree.
4. *Initialization:* Assigning potential functions ϕ_C to cliques.

Starting from a DAG $\mathcal{D}$, the computational structure is set up in several steps:

1. *Moralisation:* Constructing $\mathcal{D}^m$, exploiting that if P factorizes over $\mathcal{D}$, it factorizes over $\mathcal{D}^m$. Skip if starting from an undirected graph.
2. *Triangulation:* Adding edges to find chordal graph $\tilde{\mathcal{G}}$ with $\mathcal{G} \subseteq \tilde{\mathcal{G}}$. This step is non-trivial (NP-complete) to optimize;
3. *Constructing junction tree:* Using MCS, the cliques of $\tilde{\mathcal{G}}$ are found and arranged in a junction tree.
4. *Initialization:* Assigning potential functions ϕ_C to cliques.

Computations are executed by *message passing*.

Starting from a DAG $\mathcal{D}$, the computational structure is set up in several steps:

1. *Moralisation:* Constructing $\mathcal{D}^m$, exploiting that if P factorizes over $\mathcal{D}$, it factorizes over $\mathcal{D}^m$. Skip if starting from an undirected graph.
2. *Triangulation:* Adding edges to find chordal graph $\tilde{\mathcal{G}}$ with $\mathcal{G} \subseteq \tilde{\mathcal{G}}$. This step is non-trivial (NP-complete) to optimize;
3. *Constructing junction tree:* Using MCS, the cliques of $\tilde{\mathcal{G}}$ are found and arranged in a junction tree.
4. *Initialization:* Assigning potential functions ϕ_C to cliques.

Computations are executed by *message passing*.

The complete process above is known as *compilation*.

Initialization

1. For every vertex $v \in V$ we find a clique $C(v)$ in the triangulated graph $\tilde{\mathcal{G}}$ which contains $\mathrm{pa}(v)$. Such a clique exists because $v \cup \mathrm{pa}(v)$ are complete in $\mathcal{D}^m$ by construction, and hence in $\tilde{\mathcal{G}}$;

Initialization

1. For every vertex $v \in V$ we find a clique $C(v)$ in the triangulated graph $\tilde{\mathcal{G}}$ which contains $\mathrm{pa}(v)$. Such a clique exists because $v \cup \mathrm{pa}(v)$ are complete in $\mathcal{D}^m$ by construction, and hence in $\tilde{\mathcal{G}}$;
2. Define potential functions ϕ_C for all cliques C in $\tilde{\mathcal{G}}$ as

$$\phi_C(x) = \prod_{v:C(v)=C} p(x_v \,|\, x_{\mathrm{pa}(v)})$$

where the product over an empty index set is set to 1, i.e. $\phi_C \equiv 1$ if no vertex is assigned to C.

Initialization

1. For every vertex $v \in V$ we find a clique $C(v)$ in the triangulated graph $\tilde{\mathcal{G}}$ which contains $\mathrm{pa}(v)$. Such a clique exists because $v \cup \mathrm{pa}(v)$ are complete in $\mathcal{D}^m$ by construction, and hence in $\tilde{\mathcal{G}}$;
2. Define potential functions ϕ_C for all cliques C in $\tilde{\mathcal{G}}$ as
$$\phi_C(x) = \prod_{v:C(v)=C} p(x_v \mid x_{\mathrm{pa}(v)})$$
where the product over an empty index set is set to 1, i.e. $\phi_C \equiv 1$ if no vertex is assigned to C.
3. It now holds that
$$p(x) = \prod_{C \in \mathcal{C}} \phi_C(x).$$

Overview

This involves following steps

1. *Incorporating observations:* If $X_E = x_E^*$ is observed, we modify potentials as

$$\phi_C(x_C) \leftarrow \phi_C(x) \prod_{e \in E \cap C} \delta(x_e^*, x_e),$$

with $\delta(u, v) = 1$ if $u = v$ and else $\delta(u, v) = 0$. Then:

$$p(x \mid X_E = x_E^*) = \frac{\prod_{C \in \mathcal{C}} \phi_C(x_C)}{p(x_E^*)}.$$

2. Marginals $p(x_E^*)$ and $p(x_C \mid x_E^*)$ are then calculated by a local *message passing* algorithm.

Separators

Between any two cliques C and D which are neighbours in the junction tree their intersection $S = C \cap D$ is called a *separator*. In fact, *the sets S are the minimal separators appearing in any decomposition sequence.*

We also assign potentials to separators, initially $\phi_S \equiv 1$ for all $S \in \mathcal{S}$, where $\mathcal{S}$ is the set of separators.

Finally let

$$\kappa(x) = \frac{\prod_{C \in \mathcal{C}} \phi_C(x_C)}{\prod_{S \in \mathcal{S}} \phi_S(x_S)}, \tag{1}$$

and *now it holds that $p(x \mid x_E^*) = \kappa(x)/p(x_E^*)$.*

The expression (1) will be *invariant* under the message passing.

Marginalization

The *A-marginal* of a potential ϕ_B for $A \subseteq V$ is

$$\phi_B^{\downarrow A}(x) = \phi_B^{\downarrow A}(x_A) = \sum_{y_{A\cap B}:y_{A\cap B}=x_{A\cap B}} \phi_B(y)$$

Since ϕ_B depends on x through x_B only it is true that if $B \subseteq V$ is 'small', marginal can be computed easily.

Note that the marginal $\phi^{\downarrow A}$ depends on x_A only.

Marginalization satisfies

Consonance For subsets A and B: $\phi^{\downarrow(A\cap B)} = \left(\phi^{\downarrow B}\right)^{\downarrow A}$

Distributivity If ϕ_C depends on x_C only and $C \subseteq B$:
$(\phi\phi_C)^{\downarrow B} = \left(\phi^{\downarrow B}\right)\phi_C.$

Essentially the distributivity ensures that we can move factors in a sum outside of the summation sign.

Messages

When C *sends message* to D, the following happens:

$$\boxed{\phi_C}\;\boxed{\phi_S}\;\boxed{\phi_D}$$

$$\longrightarrow$$

$$\boxed{\phi_C}\;\boxed{\phi_C^{\downarrow S}}\;\boxed{\phi_D \frac{\phi_C^{\downarrow S}}{\phi_S}}$$

Before *After*

Computation is *local*, involving only variables within cliques.

The expression

$$\kappa(x) = \frac{\prod_{C \in \mathcal{C}} \phi_C(x_C)}{\prod_{S \in \mathcal{S}} \phi_S(x_S)}$$

is *invariant under the message passing* since $\phi_C \phi_D / \phi_S$ is:

$$\frac{\phi_C \, \phi_D \frac{\phi_C^{\downarrow S}}{\phi_S}}{\phi_C^{\downarrow S}} = \frac{\phi_C \phi_D}{\phi_S}.$$

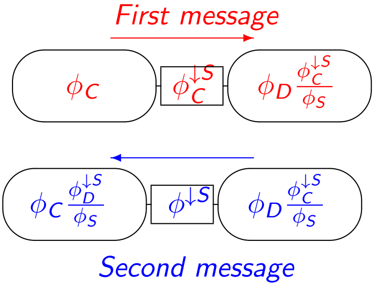
After the message has been sent, D *contains the* D*-marginal of* $\phi_C \phi_D / \phi_S$.

To see this, calculate

$$\left(\frac{\phi_C \phi_D}{\phi_S}\right)^{\downarrow D} = \frac{\phi_D}{\phi_S} \phi_C^{\downarrow D} = \frac{\phi_D}{\phi_S} \phi_C^{\downarrow S}.$$

Second message

If D *returns message* to C, the following happens:



Now all sets contain the relevant marginal of $\phi = \phi_C \phi_D / \phi_S$:

The separator contains

$$\phi^{\downarrow S} = \left(\frac{\phi_C \phi_D}{\phi_S}\right)^{\downarrow S} = (\phi^{\downarrow D})^{\downarrow S} = \left(\phi_D \frac{\phi_C^{\downarrow S}}{\phi_S}\right)^{\downarrow S} = \frac{\phi_C^{\downarrow S}\, \phi_D^{\downarrow S}}{\phi_S}.$$

C contains

$$\phi_C \frac{\phi^{\downarrow S}}{\phi_C^{\downarrow S}} = \frac{\phi_C}{\phi_S} \phi_D^{\downarrow S} = \phi^{\downarrow C}$$

since, as before

$$\left(\frac{\phi_C \phi_D}{\phi_S}\right)^{\downarrow C} = \frac{\phi_D}{\phi_S} \phi_C^{\downarrow D} = \frac{\phi_C}{\phi_S} \phi_D^{\downarrow S}.$$

Further messages between C and D are neutral! Nothing will change if a message is repeated.

Two phases:

- COLLINFO: messages are sent from leaves towards arbitrarily chosen root R.
  After COLLINFO*, the root potential satisfies* $\phi_R(x_R) = \kappa^{\downarrow R}(x_R) = p(x_R, x_E^*)$.

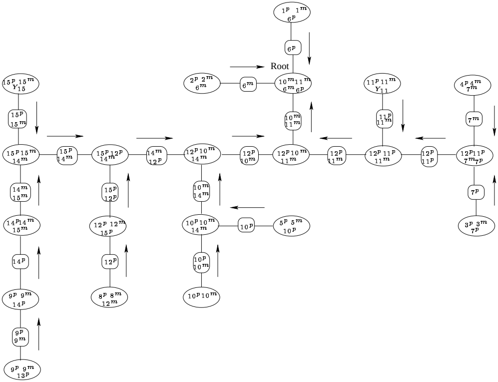
Two phases:

- COLLINFO: messages are sent from leaves towards arbitrarily chosen root R.
  After COLLINFO, *the root potential satisfies* $\phi_R(x_R) = \kappa^{\downarrow R}(x_R) = p(x_R, x_E^*)$.
- DISTINFO: messages are sent from root R towards leaves.
  After COLLINFO *and subsequent* DISTINFO, *it holds for all* $B \in \mathcal{C} \cup \mathcal{S}$ *that* $\phi_B(x_B) == \kappa^{\downarrow B}(x_B) = p(x_B, x_E^*)$.

Two phases:

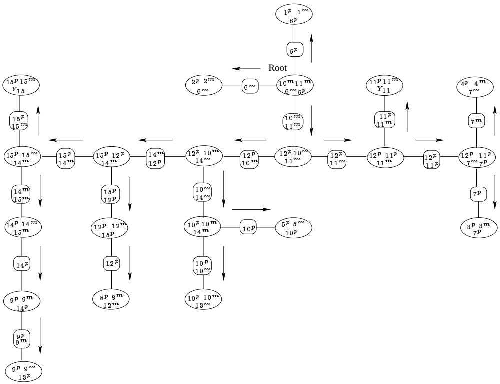
- COLLINFO: messages are sent from leaves towards arbitrarily chosen root R.
  After COLLINFO*, the root potential satisfies* $\phi_R(x_R) = \kappa^{\downarrow R}(x_R) = p(x_R, x_E^*)$.
- DISTINFO: messages are sent from root R towards leaves.
  After COLLINFO *and subsequent* DISTINFO*, it holds for all* $B \in \mathcal{C} \cup \mathcal{S}$ *that* $\phi_B(x_B) == \kappa^{\downarrow B}(x_B) = p(x_B, x_E^*)$.
- Hence $p(x_E^*) = \sum_{x_S} \phi_S(x_S)$ for any $S \in \mathcal{S}$ and $p(x_v \mid x_E^*)$ can readily be computed from any ϕ_S with $v \in S$.

CollInfo



Messages are sent from leaves towards root.

DistInfo



After CollInfo, messages are sent from root towards leaves.

The correctness of the algorithm is easily established by induction:

We have on the previous overheads shown correctness for a junction tree with only two cliques.

Now consider a leaf clique L of the juction tree and let $V^* = \cup_{C:C\in\mathcal{C}\setminus\{L\}} C$.

Because the tree is a junction tree, we have $S^* = L \cap C^* = L \cap V^*$ where C^* is the neighbour of L in the junction tree. Thus L and V^* form a junction tree of two cliques with separator S^*

After a message has been sent from L to V^* in the CollInfo phase, ϕ_{V^*} is equal to the V^*-marginal of κ.

By induction, when all messages have been sent except the one from the neighbour clique C^* to L, all cliques other than L contain the relevant marginal of κ, and

$$\phi_{V^*} = \frac{\prod_{C:C\in\mathcal{C}\setminus\{L\}} \phi_C}{\prod_{S:S\in\mathcal{S}\setminus\{S^*\}} \phi_S}.$$

Now let, V^* send its message back to L. To do this, it needs to calculate $\phi_{V^*}^{\downarrow S^*}$. But since $S^* \subseteq C^*$, and $\phi_{C^*} = \phi_{V^*}^{\downarrow C^*}$ we have

$$\phi_{V^*}^{\downarrow S^*} = \phi_{C^*}^{\downarrow S^*}$$

and sending a message from V^* to L is thus equivalent to sending a message from C^* to L. Thus, after this message has been sent, $\phi_L = \kappa^{\downarrow L}$ as desired.

Alternative scheduling of messages

Local control:

Allow clique to send message if and only if it has already received message from all other neighbours. Such messages are *live.*

Using this protocol, there will be one clique who first receives messages from all its neighbours. This is effectively the root R in CollInfo and DistInfo.

Additional messages never do any harm (ignoring efficiency issues) as κ is invariant under message passing.

Exactly two live messages along every branch is needed.