Distributed probabilistic inference

Mark A. Paskin
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
mark@paskin.org

Carlos Guestrin
Berkeley Research Center
Intel Corporation
Berkeley, CA 94720
guestrin@intel-research.net

September 27, 2003

Abstract

In this paper we present the answer to life, the universe and everything.

1 Introduction

Recently there has been increased interest in using probabilistic inference algorithms in distributed settings. One example is sensor networks, which are wireless networks of tiny computing devices that can sense various aspects of their environment (e.g., temperature, light, sound, acceleration, etc.) (). Other examples come from mobile robotics, where several robots may work collectively to perform tasks such as localization, mapping, or tracking. In both examples, we have a network of nodes that can sense local aspects of their environment and communicate with each other.

Given a network of sensing and computing elements, we can adopt a centralized approach to probabilistic inference in which the sensor measurements are “downloaded” from the network and analyzed them at some central location. While this centralized approach is appropriate in some circumstances, there are a few reasons we may prefer a distributed approach to inference:

1. **Parallelism.** In many networks, the nodes are not simply sensors that can communicate; they may have significant computational resources which can be brought to bear in the inference problem. This may be especially important in real-time systems, when the results of inference must be available quickly.

2. **Reduced communication costs.** In many networks, the cost of communication (in terms of power consumption) must be minimized. If each node were required to transmit every measurement, the communication costs could become prohibitive. In a distributed implementation of inference, the messages can consist of probability distributions which summarize many measurements; thus the communication requirements can be reduced.

3. **Distributed action selection.** In a distributed implementation of inference, each node of the network can compute its own posterior distribution. This gives nodes the flexibility to choose their actions in a distributed fashion. Even nodes without actuators have decisions to make, including when to power on and off, which neighbors to communicate with, and when to send messages.

This paper describes a fully distributed implementation of the junction tree algorithm, the standard algorithm for efficient exact probabilistic inference (Cowell et al. 1999). The algorithm is suitable for deployment in many distributed settings, including sensor networks, because it is sensitive to limitations
on resources for computation and communication.

1.1 A representational challenge

In designing a distributed version of the junction tree algorithm, we are confronted with a representational challenge: we must reconcile three graphs:

1. the **network topology**, which represents which nodes can communicate with each other;

2. the **graphical model**, which encodes conditional independence relationships in the probability model; and

3. the **junction tree**, which organizes the computation and communication involved in inference.

Note that the third graph, the junction tree, is constrained by the first two: it is constrained by the graphical model, in that it must respect the conditional independence properties of the probability model; and it is constrained by the sensor network topology: the clusters must be assigned to nodes that can compute the messages, and there must be physical network links over which the messages can be transmitted.

In many problems, all three of these graphs can change over time. The network topology can change by the addition or removal (failure) of nodes or the addition or removal (failure) of communication links. The graphical model can change over time in dynamic probabilistic inference tasks such as filtering, smoothing, and prediction. Finally, the junction tree can change over time because it is constrained by these other two graphs.

1.2 A computational challenge

message passing is easy, but triangulation is NP-hard

1.3 Related work

One of the simplest inference algorithms is Loopy Belief Propagation (LBP), where variables in the network send messages to each other in an attempt to reach consistency (Murphy et al. 1999). LBP has recently been proposed as a natural algorithm to implement on sensor networks (Crick and Pfeffer 2003). However, LBP is an approximate inference algorithm with limited performance guarantees; furthermore, it assumes a very simple structure on the probability model (unary and binary potentials) which is overly restrictive.

2 Problem formulation

We begin by formalizing the problem of distributed probabilistic inference.

2.1 The network model

We will assume that our network has $k$ nodes $N = \{1, 2, \ldots, k\}$. The nodes are connected by a physical network that is represented by a graph $G = (N, L)$, whose edges we will call links. We assume that links are undirected, i.e., that messages can be sent in both directions. We will assume that the nodes and links are not subject to failure, and that messages are sent asynchronously and arrive in order without error.

2.2 The probability model

Let $X = (X_i : i \in V)$ be a vector of random variables indexed by $V$. For a subset of indices $A \subseteq V$, let $X_A \triangleq (X_i : i \in A)$ be the corresponding vector of random variables and let $X_C$ be its range.

We consider probability models that take the form

$$p(x) \propto \prod_{C \in \mathcal{C}} \psi_C(x_C)$$

where $\mathcal{C}$ is a set of subsets of $V$, each $\psi_C$ is a potential function over $C$ (i.e., a non-negative function of $X_C$), and $x_C$ is the subvector of $x$ indexed by $C$. This class includes directed graphical models (i.e., Bayesian networks) and undirected graphical models such as Markov random fields. Observed variables are reflected by evidence potentials. We use $p_A(\cdot)$ to denote the marginal density of $X_A$ and $p_{A|B}(\cdot | \cdot)$ to denote the conditional density of $X_A$ given $X_B$. We
use the notation of finite measure spaces for simplicity, but our approach extends easily to the continuous case.

Our first step towards a distributed implementation of probabilistic inference is to choose a distributed representation of the probability model. That is, each node of the sensor network will be responsible for only some of the potential functions that define the probability model (1). This is desirable for two reasons:

1. **Compactness.** By distributing the model across the network, the storage requirements at each node can be minimized.

2. **Modularity.** When nodes are responsible for restricted portions of the model, the model can be changed without informing the entire network. As we will see, this is especially useful in when nodes measure the same quantity many times.

Accordingly, we assume that the potential functions in (1) are partitioned into subsets, one per node of the network. This allows us to write

\[ p(x) \propto \prod_{i=1}^{n} \psi_i(x_{C_i}) \tag{2} \]

where \( \psi_i \) is the potential function at node \( i \) and \( C_i \subset V \) is the set of variables that are inputs to \( \psi_i \); we can regard \( C_i \) as the minimal set of variables that node \( i \) must reason about.

We can use plates or probabilistic relational models to specify these models in a modular way…

### 2.3 Mapping the junction tree to the sensor network

There are three ways we can relate the clusters of the junction tree to the network nodes:

1. **One cluster to many nodes.** This option is difficult to implement, since distinct nodes must coordinate to perform computations like products and marginalization. However, when the junction tree has clusters that are too large for a single node (because the cluster belief is too large to represent, or marginalization is too expensive), this option allows us to relax the computational and storage requirements of the nodes. We do not pursue this path here, preferring to deal with this problem by using approximate inference.

2. **One cluster to one node.** This is the simplest and most direct mapping, where there is a one-to-one correspondence between the clusters of the junction tree and the nodes of the sensor network. In this case, the messaging metaphor that is inherent in the language used to describe junction tree algorithm become literal: the cluster (at each sensor node) sends messages its neighbors over the network.

3. **Many clusters to one node.** This case can be viewed as a special case of the one-to-one relationship above, since we can view the union of the clusters assigned to a node as a single cluster. If we leverage the internal structure of this union, then we obtain an algorithm that is similar to the “nested” junction tree technique of (Kjærulff 1997), where each cluster of a junction tree can be represented by an internal junction tree that is used to carry out message computations.

In this work, we will choose the second option, and assign to each node of the sensor network a cluster of the junction tree. We can view the third option as an optimization that uses a different internal representation to gain a performance advantage in certain models.

Given our choice to identify clusters of the junction tree with sensor nodes, it is natural to choose the junction tree’s edges from among the communication links of the sensor network. That is, the junction tree will be chosen as a spanning tree of the sensor network. We will denote this spanning tree by \( T = (N, E) \); \( E \) is the subset of the links \( L \) that are used as edges in the junction tree.
3 Distributed implementation of the junction tree algorithm

Our general algorithm has the following steps:

1. Choose an arbitrary spanning tree of the network using a distributed spanning tree algorithm such as (Gallagher et al. 1983).

2. Construct the triangulated cliques by propagating messages on this spanning tree.

3. Until there is no further improvement in the cost of message passing, perform greedy optimal alterations to the junction tree.

4. Perform inference via message passing on the final junction tree.

3.1 Distributed triangulation for a fixed spanning tree

Imagine now that we have selected a particular spanning tree of the sensor network. We now have a cluster tree: it is singly connected and it has clusters that cover the cliques of the graphical model, but it does not (necessarily) have the junction tree property. We will now describe a distributed algorithm for ensuring the junction tree property.

The basic form of the algorithm closely resembles the message passing updates involved in inference: the clusters send messages along each edge of the cluster tree in both directions, and the ordering of the messages obeys the same (Collect then Distribute) message passing protocol. The difference is that in this case, the messages consist of sets of variables, rather than potential functions.

Let $T = (N, L)$ be the tree topology. For each (directed) edge $(j, k)$, define the variables behind $(j, k)$ to be

$$V_{jk} \triangleq C_j \cup \bigcup_{(i,j) \in E \atop i \neq k} V_{ij}$$

Intuitively, $V_{jk}$ is the set of variables that are in clusters that are on the $j$ side of the $(j, k)$ edge. Thus, if $j$ is a leaf, then $V_{jk} = C_j$ and $V_{kj} = V$.

When a node has received all of its variable messages, it can select which variables must be added to its cluster to ensure the junction tree property. In particular, if it receives more than one message that contains some variable $a$ but its cluster does not contain $a$, then it must add $a$ to its cluster to ensure the junction tree property. More formally, we must update

$$C_j^* \triangleq C_j \cup \bigcup_{(i,j) \in E \atop (k,j) \in E \atop i \neq k} V_{ij} \cap V_{jk}$$

Figure 1 illustrates this process for two different cluster trees.

[Query routing: each node $j$ can use the variable messages it receives to route queries about variables it does not possess.]

3.2 Distributed optimization of the triangulation

Both of the junction trees in Figures 1(a) and 1(b) are valid; which one should we prefer? In the standard, centralized setting, the “best” junction tree for a given probability model is usually the junction tree with the smallest maximum cluster size, which determines the total computational cost of inference. Unfortunately, finding this junction tree is NP-hard.

In the distributed setting, the problem of selecting a good junction tree is yet more complicated, because instead of simply minimizing the total computational cost, we have to take the following into account:

1. The sensor nodes may have different computational power. A junction tree with one large cluster and many small clusters may be preferred to one with many medium-sized clusters, if the large cluster is located at a sensor node with extra computational resources.

2. The separators should scale with the communication cost. The representational size of the messages that are transmitted over the edges of the junction tree scale with the number of variables in the separator. Because large separators
Figure 1: Illustration of constructing embedded junction trees. Each node $i$ is annotated with its original variables $C_i$ (top, blue) and the set of variables $C_i^*$ it must carry to maintain the junction tree property (bottom, purple). The messages $V_{ij}$ are indicated in red.

4 Coping with link failures

References


