Algorithms for Graphical Models (AGM)

Decomposable models and join forests

$Date: 2008/10/16 10:33:15 $

AGM-09

In this lecture

- Triangulated (decomposable) graphs
- Maximum cardinality search
- Decomposable hypergraphs
- Join forests

The structure of variable elimination

- In variable elimination, to sum out a variable we have to first multiply all factors containing that variable. This corresponds to merging the associated hyperedges.

- If there is more than one factor this may produce a new hyperedge.

- We then sum out the variable: this corresponds to removing the variable from the (possibly new) hyperedge.

- But it's easier to 'see' this process by looking at the distribution's interaction graph.

Adding lines to interaction graphs by vertex elimination

- Recall that two variables are connected in the interaction graph if they both appear in a common hyperedge, i.e. they are both variables for some factor.

- Merging all hyperedges containing a given variable v corresponds to connecting—in the interaction graph—all variables in these hyperedges. A new connection is called a *fill-in*.

- Deleting a vertex from the only hyperedge to contain it corresponds to deleting that vertex from the graph.

Triangulation by variable elimination

- This process is known as *triangulation*.
- If carried out exactly as described we would end up with an empty graph (since all variables would eventually get deleted).
- Instead we just mark 'eliminated' vertices and treat them *as if* they were deleted.
- It's most easily understood graphically. Cue `triangulation_demo` from `gPy.Examples`

Complexity of variable elimination

- The triangulated graph gives a nice graphical representation of the complexity of variable elimination with a particular ordering.

- If there are n variables and N is the size (number of rows) of the biggest factor, then variable elimination is $O(nN)$.

- If this biggest factor has μ variables each with no more than v values then we only get an exponential upper bound: $N \leq v^{\mu}$.

- Need to keep μ, the size of the biggest clique, small!

Looking for good triangulations

- Even deciding whether there is an ordering which bounds the biggest clique by some value is NP-complete. Thus finding an ordering which minimises the biggest clique is a hard problem.

- Nonetheless, there are reasonable heuristic approaches.

- One *greedy* option is, at each point, to eliminate whichever vertex produces the fewest fill-in lines (preferably none).

- Another option is to compute an elimination ordering *backwards*, using *maximum cardinality search*.

Maximum cardinality search on graphs

Number vertices from $[n-1]$ to [0] in descending order. As the next vertex to number, select the vertex adjacent to the largest number of previously numbered vertices, breaking ties arbitrarily. (Tarjan and Yannakakis, Siam J. Computing, 13:3, 1984)

It's really that simple. Cue `max_card_search_demo.py` from `gPy.Examples`

Properties of maximum cardinality search

- If there are n vertices and m edges in the graph, then maximum cardinality search is $O(n + m)$, so it's linear :-).

- If there is a zero fill-in for a graph, maximum cardinality search will find one. :-)

- If there is not, then Kjærulff has shown that the order it finds is generally not that great. :-(

Key facts about triangulated graphs

- A triangulated graph is *defined* to be an undirected graph such that every cycle of length $n \geq 4$ has a *chord*: two non-consecutive vertices which are neighbours.

- This is why triangulated graphs are also called *chordal*.

- A graph is triangulated if and only if it has a zero fill-in.

Some definitions

- An *elimination ordering* for a graph is just an ordering of the vertices of the graph.

- The *fill-in* generated by an elimination ordering is the collection of *extra* lines added by running the vertex elimination algorithm with this ordering.

- An elimination ordering is a *zero fill-in* if its fill-in is empty.

Towards join forests

- The cliques of a triangulated graph can be arranged in an important data structure: a *join forest*. These are very closely related to the cluster trees we have seen previously.

- A join forest is a collection of trees—connected graphs without cycles—whose vertices are the cliques of the triangulated graphs.

- We will construct them by doing vertex elimination on a distribution's *hypergraph* rather than its interaction graph.

Decomposable hypergraphs

- A hypergraph is *decomposable* if its reduction is the clique hypergraph of a decomposable (i.e. triangulated) graph.

- This is not (usually) given as the *definition* of decomposable hypergraphs, but it may as well be.

- Note that decomposable hypergraphs are always graphical.

- In the maths literature decomposable hypergraphs are usually called *acyclic hypergraphs*.

Graham's algorithm: variable elimination on hypergraphs

Graham's algorithm: Repeat the following two operations on a hypergraph until neither apply:

1. Delete a vertex that appears in only one hyperedge.

2. Delete a hyperedge that is contained in another hyperedge

A hypergraph is decomposable iff Graham's algorithm deletes all vertices.

Join trees

A *join tree* $\mathcal{T}$ for a hypergraph $\mathcal{H}$ is a tree such that:

1. the vertices of $\mathcal{T}$ are the hyperedges of $\mathcal{H}$, and

2. for all $h_1, h_2 \in \mathcal{H}$ and any h on the unique path between h_1 and h_2 in $\mathcal{T}$, we have $h_1 \cap h_2 \subseteq h$.

The second condition is the *join* or *junction* property. Join trees are often called *junction trees*.

Cue `join_forest_demo` from `gPy.Examples`

Join forests

A *join forest* for a hypergraph $\mathcal{H}$ is simply a collection of join trees for hypergraphs $\mathcal{H}_i$ where $\mathcal{H}$ is the disjoint union of the $\mathcal{H}_i$ such that hyperedges in different trees are disjoint.

Almost always we will deal with join forests containing a single tree.

So, most of the literature talks about junction trees rather than join forests.

Decomposable hypergraphs and join forests

A fundamental theorem is that:

> A hypergraph is decomposable if and only if it has a join forest.

Constructing join forests

- There are number of different ways of constructing a join forest for a decomposable hypergraph.

- Recall that a decomposable hypergraph is always the clique hypergraph of some triangulated graph, and that each triangulated graph has an associated elimination order which is a zero fill-in.

- If we run variable elimination with this ordering then each elimination 'happens in a clique'. If clique C_i produces a message which is 'used' by clique C_j then connect these two cliques in the join forest.

Constructing join forests with Graham's algorithm

1. Delete a vertex that appears in only one hyperedge.

2. Delete a hyperedge that is contained in another hyperedge

- Graham's algorithm is just 'structural' variable elimination on a decomposable hypergraph using a zero fill-in ordering.

- When a hyperedge gets deleted just connect it to one of the hyperedges which contained it. Cue `grahams_demo` from `gPy.Examples`