CSCE921 Scribe Notes: February 20, 2013

Zion Schell

1 Class Overview

Class began with the continuation of a conversation between Dr. Choueiry and Shant regarding an error in the Join-Tree Clustering (JTC) algorithm's pseudocode. After class, Dr. Choueiry provided the following notes on this discussion:

> JTC takes as input a variable ordering, moralizes the ordering, finds the maximal cliques corresponding to the ordering, then builds the join tree based on the considered maximal cliques ordering. Shant found, while doing his experiments, that, if he chooses the following ordering, the resulting join tree does not necessarily have the connectedness property: Apply MinFill, using the PEO build Max-Cliques, then build the JoinTree. Shant says that the problem does not show up if the JTC is given as input the ordering obtained by by the MaxCardinality algorithm after applying MinFill.

Next, Robert Woodward presented the pseudocode for the MaxCliques algorithm (see pseudocode in Appendix A and animation of the code on Slide 13 of the slides). Finally, we began a quick by critical examination of Dechter's own slides for chapter 9.

2 MaxCliques Algorithm (Robert Woodward)

Robert Woodward presented the algorithm as follows. Comparable commentary regarding the purpose of each line is included.

Input: A triangulated graph $G = (V, E)$ and a perfect elimination order σ.

Output: A vector of cliques $\mathcal{C}$.

```
1 $j \leftarrow 0$
```

The variable j is used as an index in the vector $\mathcal{C}$ so that the vector can be used as a list; every time something is added to $\mathcal{C}$, it is added at the jth location, and j is incremented immediately afterwards in all cases.

```
2 foreach $v \in V$ do $S(v) \leftarrow 0$
```

This step initializes the vector S. S is used to track the size of the largest clique in which a node is involved.

```
3 for $i \leftarrow 1$ to $n$ do
```

The variable i is used as the incrementor to walk through the perfect elimination ordering σ.

```
4     $v \leftarrow \sigma(i)$
```

As previously described, the i^{th} element of σ is stored as v for succinctness' sake.

```
5     $X \leftarrow \{x \in Adj(v) \mid \sigma^{-1}(v) < \sigma^{-1}(x)\}$
```

The set X contains the set of parents of v (elements later in the perfect elimination ordering that are also found in the adjacency list of v).

```
6     if $Adj(v) = \emptyset$ then
```

```
7          $\mathcal{C}(j) \leftarrow v$
8          $j \leftarrow j + 1$
9      **end**
```
(6-9): If a vertex has absolutely nothing adjacent to it (to say nothing of the presence of parents in the perfect elimination ordering), then it forms its own clique.

```
10     **if** $X = \emptyset$ **then return** $\mathcal{C}$
```
If a vertex has no parents, then we have reached the end of the first connected component of the perfect elimination ordering. As there should only be one connected component in a perfect elimination ordering on a triangulated graph, the algorithm should therefore terminate.

```
11     $u \leftarrow \sigma(\min\{\sigma^{-1}(x) \mid x \in X\})$
```
The variable u will contain the closest parent of v (the member of X that appears earliest in the perfect elimination ordering).

```
12     $S(u) \leftarrow \max\{S(u), |X| - 1\}$
```
The value in $S(u)$ either remains the same or is replaced with the number of parents of v (other than u). The objective of this step is to track the largest size of a clique that may exist later in the perfect elimination ordering.

```
13     **if** $S(v) < |X|$ **then**
14         $\mathcal{C}(j) \leftarrow \{v\} \cup X$
15         $j \leftarrow j + 1$
16     **end**
```
(13-16): If the vertex v has more parents than previous vertices have suggested possible, then the largest clique of which it is a member is the one containing itself and its parents. That clique should therefore be added to the set of cliques.

```
17 **end**
18 **return** $\mathcal{C}$
```

In short, the algorithm operates by walking through each node in the perfect elimination ordering, finding its closest parent, and informing that parent of the number of other parents it has. If, when the parent is reached, this parent has a child that had a larger number of parents than the parent did, then the group containing the parent and its parents is the largest existing clique in which the parent lies. The algorithm was deemed "beautiful" by Dr. Choueiry because it functions in what is essentially a single pass over the nodes, accumulating the required information and concluding the cliques in question in a simple, elegant, and effective manner.

3 Tree Clustering Algorithm

Dr. Choueiry then went back two slides to Slide 11 with the flow chart of code of the Tree Clustering algorithm. The sequence of operations is:

1 Triangulate the graph (with MinFill)

2 Find the maximal cliques by:

  2.1 Using MaxCardinality to generate an elimination ordering

  2.2 and using MaxCliques with that elimination ordering to generate a set of maximal cliques

3 Generating a join tree from these cliques by connecting a clique C_i with the deepest clique C_j in the ordering such that C_j shares the largest number of variables with C_i for $1 \leq j < i$).

4 Solving each clique in the tree

5 Applying directional arc-consistency from the bottom to top

and solutions in the resulting join tree can be generated in a backtrack-free manner from top to bottom.

A general elaboration of the MaxCardinality algorithm was requested by Nate Stender and provided by Dr. Choueiry. I did not perceive any new information in this discussion, and therefore will not reiterate it here. Suffice it to say that MaxCardinality can generate an instantiation ordering rapidly starting from an arbitrary node.

We then proceeded into Dechter's slides.

4 Dechter's Slides

The slides in question are of chapter 9 of her book, regarding tree decomposition methods. In the first few slides, we reviewed relevant terminology as follows:

- A hypergraph is, informally, a graph where edges need not have arity 2. Formally, a hypergraph is $H = (V, S)$ such that $V = \{v_1, v_2, \ldots, v_n\}$ and $S = \{S_1, S_2, \ldots, S_l\}$, with each S_i being a *hyperedge* defined over a subset of the vertices V.
- The dual graph of a hypergraph is a graph in which each hyperedge is represented by a vertex, and two vertices of the dual graph are connected by an edge if the hyperedges they represent share a vertex in the original hypergraph. Those edges are labeled with the shared vertices.
- The primal graph of a hypergraph is a graph whose vertices are the vertices of the hypergraph and in which two vertices are connected if an only if they appear in some hyperedge of the original hypergraph. More graphically, the vertices that appear in a hyperedge form a clique in the primal graph.
- The connectedness property (or the "running intersection property") of a dual graph states that if a vertex v_i of the hypergraph appears in two vertices S and S' of its dual graph, then v_i must appear in the edge labelings of some path of the dual graph from S to S'.
- A redundant edge in the dual graph is an edge that can be removed without breaking the connectedness property.
- A join graph is an edge-subgraph of a dual graph of a hypergraph that has maintained the connectedness property.
- A join tree is a join graph that is also a tree.
- A hypertree is a hypergraph whose dual graph has a join tree (i.e., can be coerced into a join-tree by removing redundant edges).
- An acyclic network is a network whose hypergraph is a hypertree.

Next, we discussed how an acyclic network can be solved by applying directional relational arc-consistency from the leaves to the root in $O(rl \log l)$ where r is the number of constraints (hyperedges) and l is the largest number of tuples in a constraint. The example on the following slide had a typo, previously identified by Nate Stender, and was skipped.

Finally, the last slide was the recognition of acyclic networks. Two methods were proposed. The first, which runs $O(e^3)$, is based on the dual graph of the network: namely, to form the maximal spanning tree over the dual graph, then check the resulting graph for *the connectedness property.* (Careful not to confuse the connectedness *property* with the definition of connectedness in graph theory.)

The second method is based on the primal graph of the network. A theorem by Maier states that a hypergraph is a hypertree iff its primal graph is both chordal (triangulated) and conformal (all maximal cliques in the primal graph correspond to hyperedges in the hypergraph). So, the method starts by testing whether the primal graph is triangulated (using the MaxCardinality algorithm). If it is, then, it identifies the max cliques (using the MaxCliques algorithm). Then, it tests whether the maximal cliques correspond exactly to the scopes of the constraints (one-to-one correspondence or bijective mapping). If they are, then we build the join tree using the JoinTree algorithm of Step 3 of Section 3 above.

Thus terminated our class on February 20th, 2013.

A MaxCliques Pseudocode

Algorithm 7: CLIQUES(G, σ)

```
Input: A triangulated graph $G = (V, E)$ and a perfect elimination order
       $\sigma$.
Output: A vector of cliques $\mathcal{C}$.
$j \leftarrow 0$
foreach $v \in V$ do $S(v) \leftarrow 0$
for $i \leftarrow 1$ to $n$ do
    $v \leftarrow \sigma(i)$
    $X \leftarrow \{x \in \mathrm{Adj}(v) \mid \sigma^{-1}(v) < \sigma^{-1}(x)\}$
    if $\mathrm{Adj}(v) = \emptyset$ then
        $\mathcal{C}(j) \leftarrow v$
        $j \leftarrow j + 1$
    end
    if $X = \emptyset$ then return $\mathcal{C}$
    $u \leftarrow \sigma(\min\{\sigma^{-1}(x) \mid x \in X\})$
    $S(u) \leftarrow \max\{S(u), |X| - 1\}$
    if $S(v) < |X|$ then
        $\mathcal{C}(j) \leftarrow \{v\} \cup X$
        $j \leftarrow j + 1$
    end
end
return $\mathcal{C}$
```