Lecture 3

Exact Inference

- Exact inference on graphs is introduced using variable elimination.
- This produces cliques trees.
- Focus here on summation task. Others similar.

Overview

- **Linear Elimination**
- Clique Trees, Again
- Building Good Clique Trees
- Computation on Clique Trees

Variable Elimination, example

This is the "Asia Visit" graph. Consider summing all variables näively:

$$\sum_{x,toc,d,b,t,lc,s,av} f_1(x,toc)\, f_2(b,d,toc)\, f_3(toc,lc,t)\, f_4(t,av)\, f_5(b,s)\, f_6(lc,s)$$

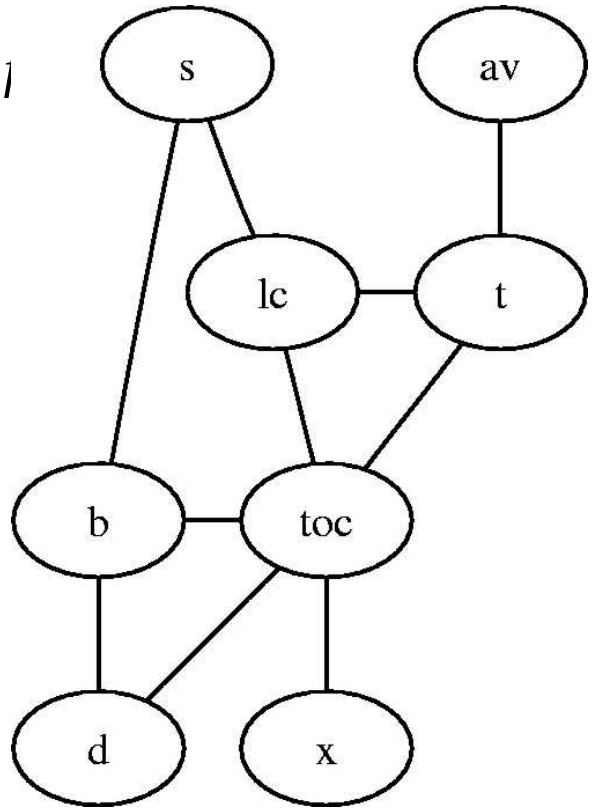
$$\sum_{toc,d,b,t,lc,s,av} f'_1(toc)\, f_2(b,d,toc)\, f_3(toc,lc,t)\, f_4(t,av)\, f_5(b,s)\, f_6(lc,s)$$

$$\sum_{d,b,t,lc,s,av} f_{1,2,3}(b,d,lc,t)\, f_4(t,av)\, f_5(b,s)\, f_6(lc,s)$$

$$\sum_{b,t,lc,s,av} f'_{1,2,3}(b,lc,t)\, f_4(t,av)\, f_5(b,s)\, f_6(lc,s)$$

$$\sum_{t,lc,s,av} f_{1,2,3,5}(lc,t,s)\, f_4(t,av)\, f_6(lc,s)$$

$$\sum_{lc,s,av} f_{1,2,3,4,5}(lc,s,av)\, f_6(lc,s)\ldots$$



where $f'_1(toc) = \sum_x f_1(x,toc)$,
$f_{1,2,3,5}(lc,t,s) = \sum_b f''_{1,2,3}(b,lc,t)\, f_5(b,s)$, etc.

Variable Elimination: Observations

- Eliminating toc binds all functions using toc into one new function, thus creating a new clique on nbrs(toc).
- Neighbors are thus continually bonded into cliques at each stage.
- Same happens if we compute a maximum instead of a sum.
- When computing marginals for all variables (our original tasks), things are a bit more complicated because this process is combined with a second sweep.

Single Variable Elimination

For an undirected graph on variables X, and $\mathscr{C}$ the set of cliques in the graph, with distribution

$$p(X) \;=\; \prod_{C \in \mathscr{C}} f_C(X_C) \;.$$

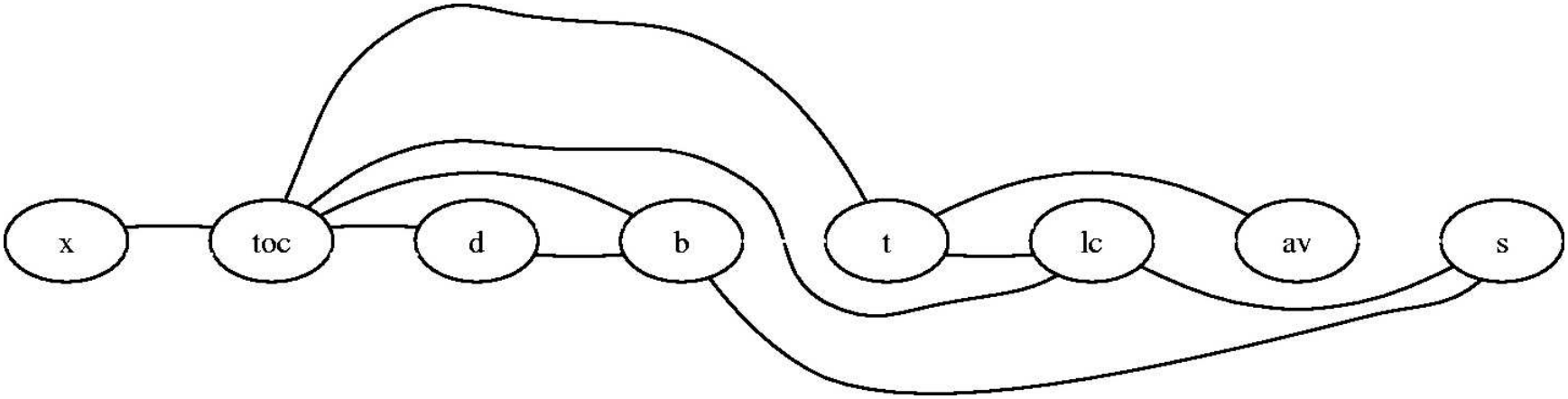
If $x \in X$ is eliminated with a $\sum$ operation, then the modifications are as follows (let $C_x = \text{nbrs}(x)$):

$$\begin{aligned} f_{C_x}(X_{C_x}) \;&=\; \sum_x \prod_{C \in \mathscr{C} : x \in C} f_C(X_C) \\ p(X - \{x\}) \;&=\; f_{C_x}(X_{C_x}) \prod_{C \in \mathscr{C} : x \notin C} f_C(X_C) \\ \mathscr{C}' \;&=\; \{C \in \mathscr{C} : x \notin C\} \cup \{C_x\} \end{aligned}$$

i.e. nbrs(x) becomes a clique in a revised graph

Variable Elimination, example cont.

If we arrange the variables linearly in the elimination ordering, we see the intermediate cliques that will be formed. *e.g.*, arcs between toc and d go to d,b,t,lc, the arguments to $f_{1,2,3}(\cdot)$.



If the variables E have been eliminated so far, then the clique set carried forward is $nbrs(E) = \cup_{x \in E} nbrs(x) - E$. This is the set present if we cut the linear layout after E.

Linear Elimination: Summation

Suppose we have finite discrete variables X, and an undirected graph on X with clique set $\mathscr{C}$ and we wish to solve the all marginals problem. This more general recursive algorithm applies if $Z = \emptyset$ initially:

1. Have a pre-existing separating set Z representing $nbrs(E)$. For some $x \in X$, build the table $f_{C_x}(X_{C_x})$.
2. Consider the sub-problem on $X - \{x\}$ given by the graph with cliques $\mathscr{C}'$, new function $f_{C_x}(X_{C_x})$ and separating set $Z' = Z/\{x\} \cup \text{nbrs}(x)$. Return the joint marginal $p(Z')$.
3. From this, compute

$$p(Z' \cup \{x\}) \;=\; p(Z')p(x|nbrs(x)) \;=\; p(Z')/f_{C_x}(X_{C_x}) \prod_{C \in \mathscr{C} : x \in C} f_C(X_C)$$

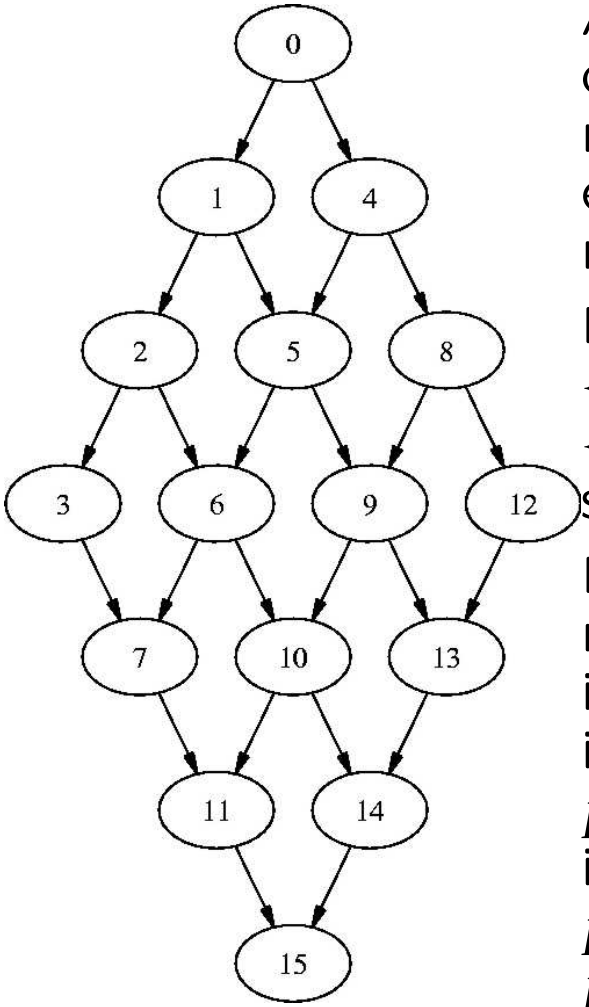
4. From $p(Z' \cup \{x\})$, compute $p(x)$ and report it.
5. Likewise compute $p(Z)$ by direct summation and pop this up to the next level.

Linear Elimination: example

This shows the key values during the algorithm's run on the linear layout. All sets are read directly off the linear layout.

Z	x	nbrs$(x)/Z$
$\emptyset$	x	$\{toc\}$
$\{toc\}$	toc	$\{b,d,t,lc\}$
$\{b,d,t,lc\}$	d	$\emptyset$
$\{b,t,lc\}$	b	$\{s\}$
$\{t,lc,s\}$	t	$\{av\}$
$\{lc,av,s\}$	lc	$\emptyset$
$\{av,s\}$	av	$\emptyset$
$\{s\}$	s	$\emptyset$

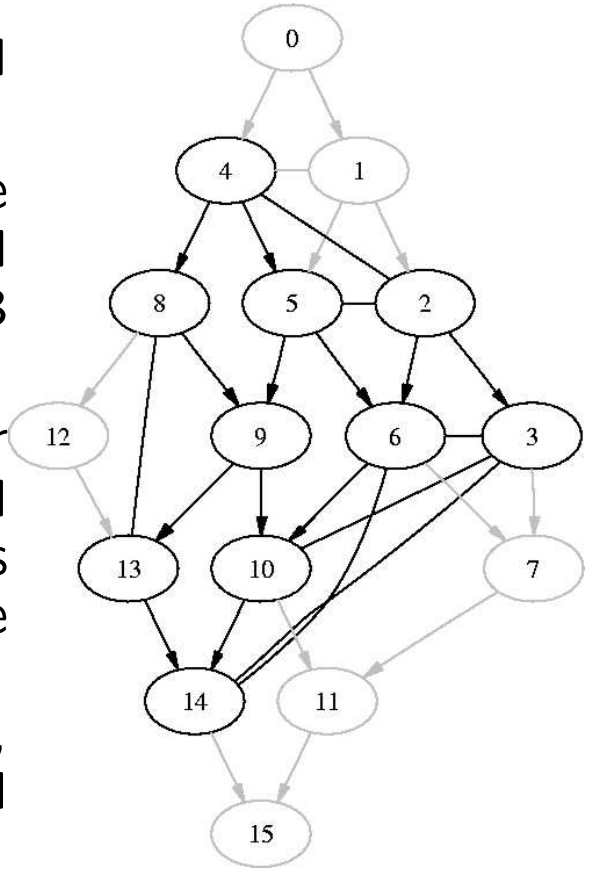
Linear Elimination, example inefficiency



As different parts of the graph are marginalized, their expanding cliques will not interact.

Here we marginalise $\{12\}$, $\{0,1\}$, and $\{7,11,15\}$, leaving 3 separate cliques.

But the linear model carries all independent parts in one big table $p(2,3,4,6,8,10,13,14)$ instead of $p(2,4)$, $p(3,6,10,14)$ and $p(8,13)$.



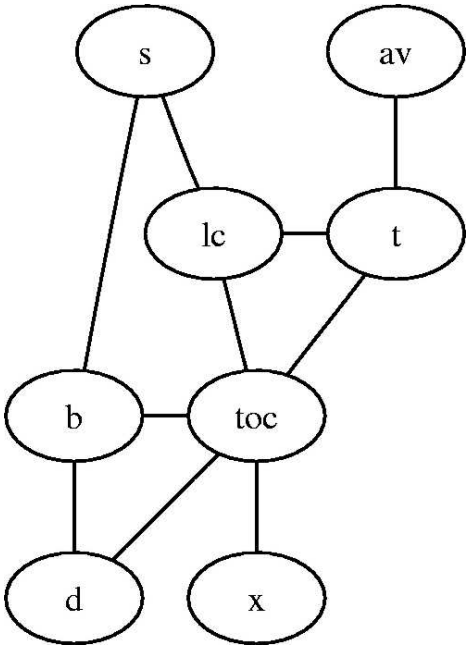
Linear Elimination, details

- Corresponds to a linear layout of the graph, whose *cut-width* or *path-width* gives the complexity of the operation.
- Electrical engineers like linear layouts (its a 1-D version of their 2-D layout problem). Most of their circuits have cut-width logarithmic in the size of the graph (see Prasad, Chong and Kuetzer, 1999), thus solution complexity polynomial.
- Inefficient: elimination should be organized in a tree or partial order.
- Note single variable elimination corresponds exactly to the problem decomposition step with $X_1 = \{x\}$, $Z = \text{nbrs}(x)$, $X_2 = X - \{x\} - \text{nbrs}(x)$.

Overview

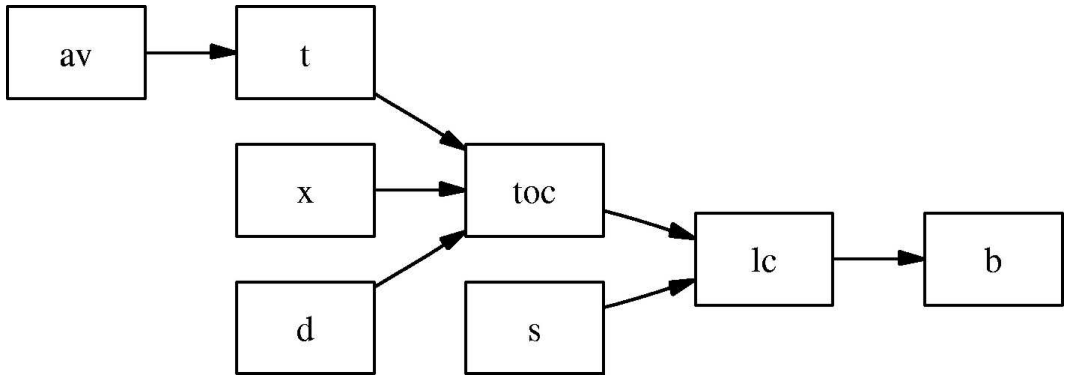
- Linear Elimination
- **Clique Trees, Again**
- Building Good Clique Trees
- Computation on Clique Trees

Tree-based Elimination, example



Use a different variable elimination, but separate non-interacting parts, so elimination graph is now a tree or partial order, not a single path.

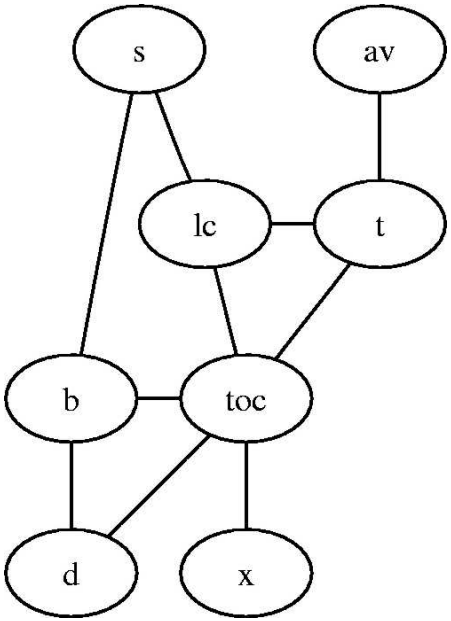
Elimination ordering below is a partial order. *e.g.* only eliminate *toc* after all t, x and d are eliminated.

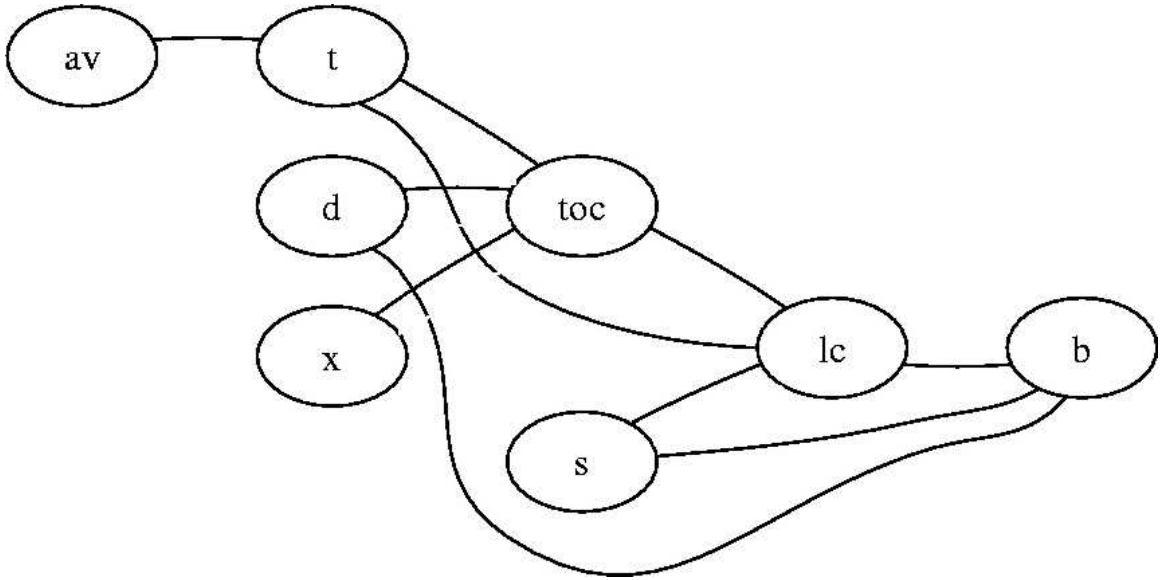


NB. Not all trees are valid elimination orderings.

Tree-based Elimination, example cont.

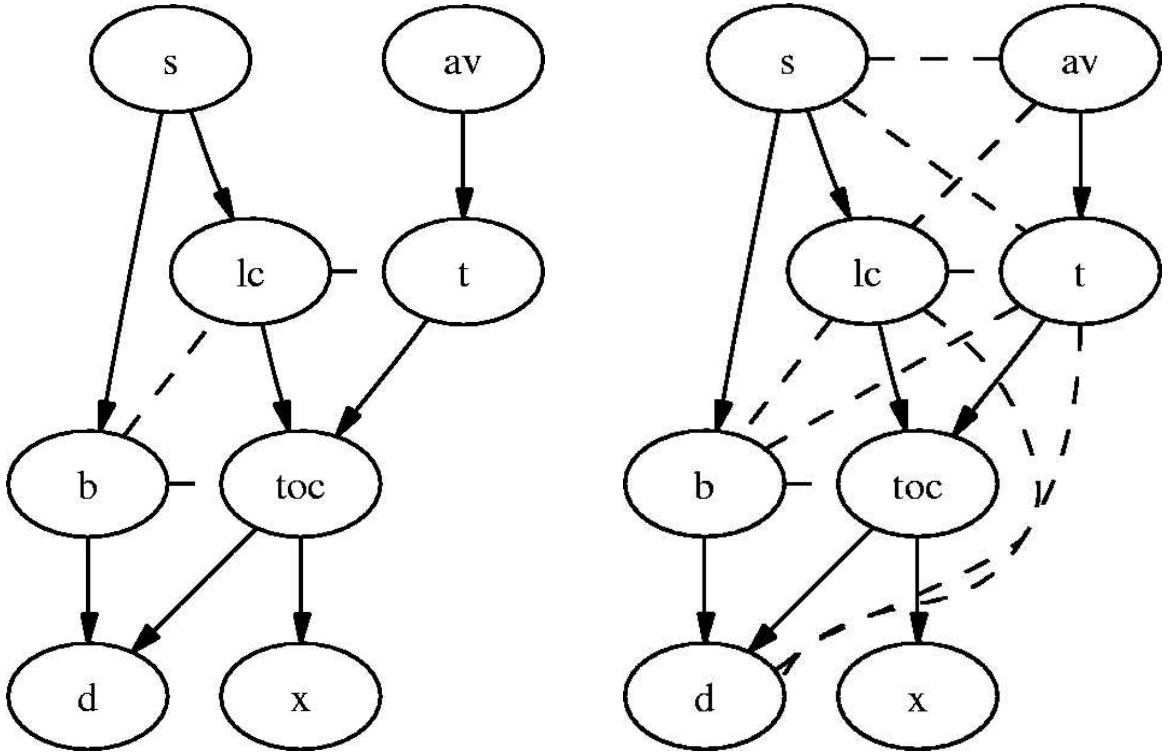
Again, a layout of the moralized graph, in a tree shape corresponding to the elimination partial order, shows the cliques induced during elimination. As for the linear case, neighbors of the subtree (which are turned into a clique) correspond to arcs travelling out of the subtree.





Tree-based Elimination, example cont.

As elimination proceeds, the graph is augmented with arcs to fill in new cliques. Full elimination by the good partial order induces the undirected graph on the left. The original linear order induces the one on the right.



Revision: Clique Trees

A clique tree $(V.E)$ is an undirected graph with no cycles whose nodes have sets of variables $C \subseteq V$, not single variables. By convention separating set between nodes X_j and X_l has variables $X_j \cap X_l$.

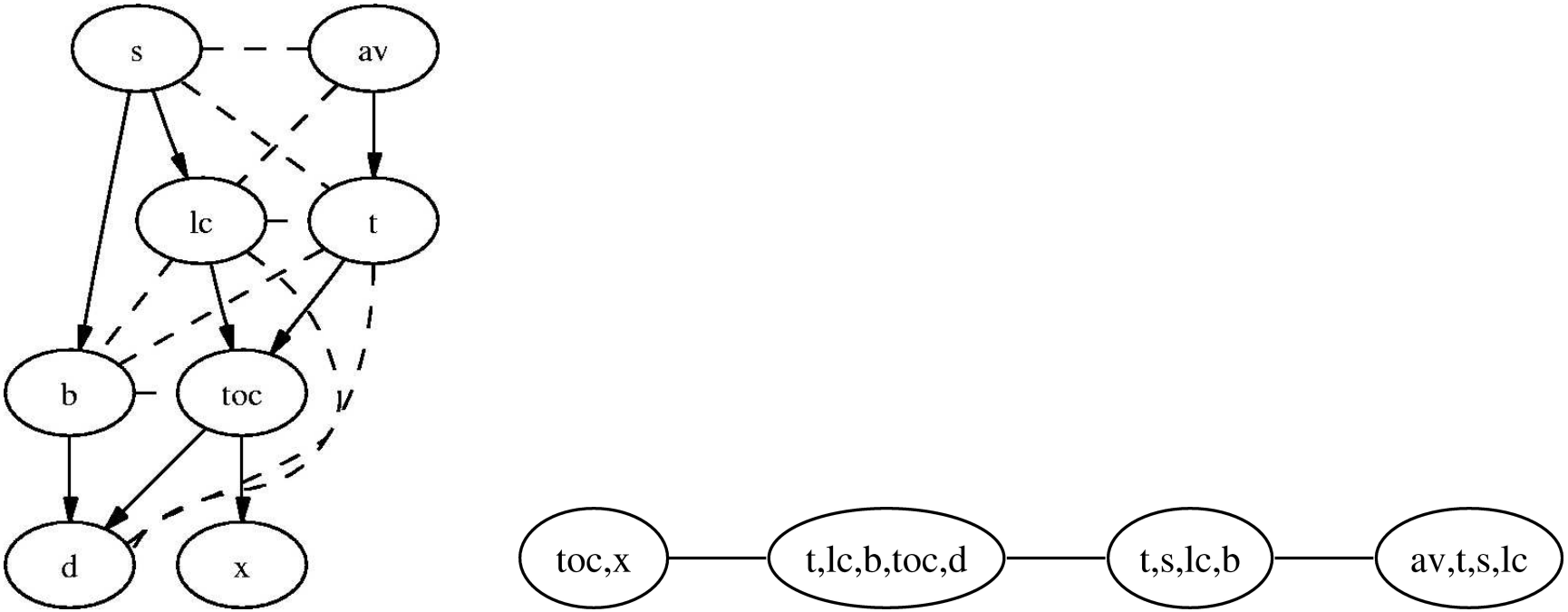
To prevent illegal independence statements, for any connected subtree in the clique tree, each separating set must condition a valid independence statement for its two sides (*e.g.* $\{a\} \perp\!\!\!\perp \{a,b\} | \{c,d\}$ is not valid). The necessary and sufficient condition is:

> if node X_j is on the path between nodes X_i and X_k then $X_i \cap X_k \subseteq X_j$

A clique tree corresponding to a directed graph (V,E) should have every clique in the moralized version of (V,E) contained in one of the cliques in the clique tree.

Tree-based Elimination, example cont.

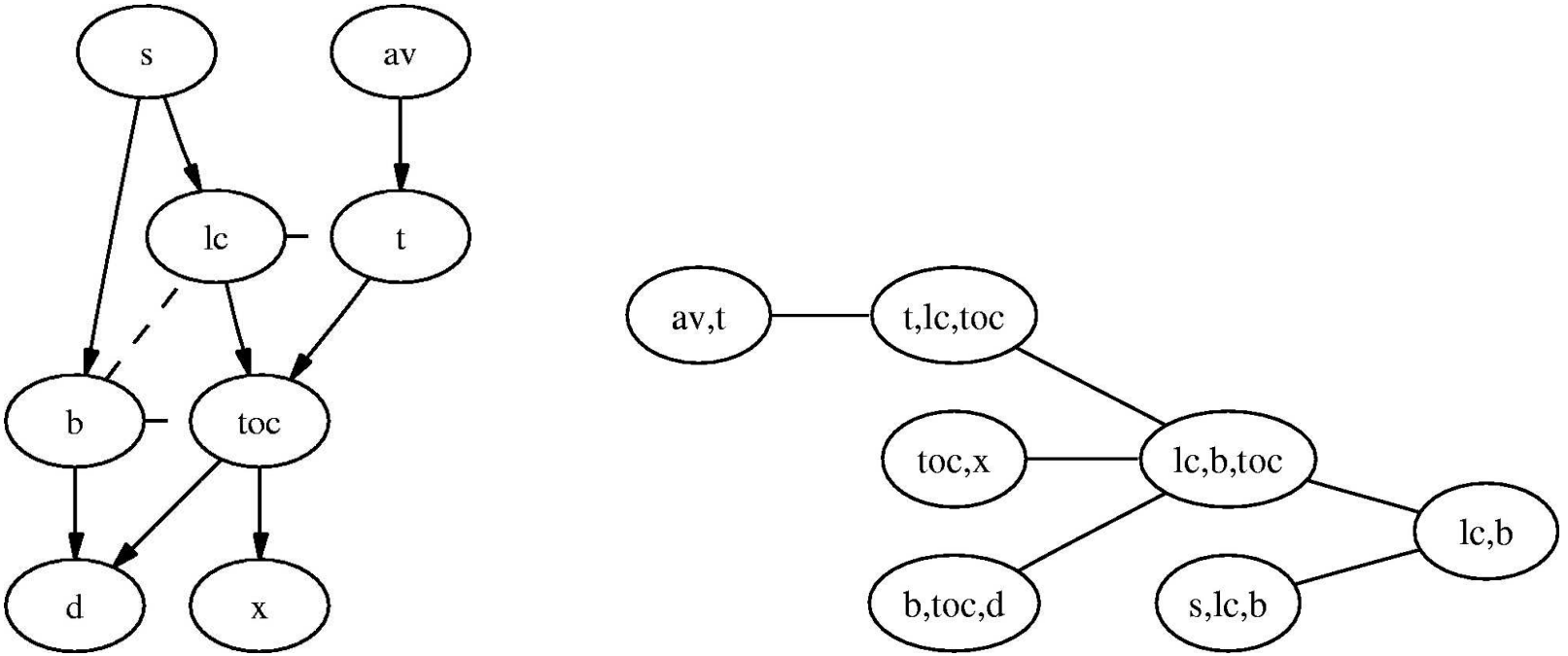
A corresponding clique tree.



The maximum clique size is 5.

Tree-based Elimination, example cont.

A corresponding clique tree.

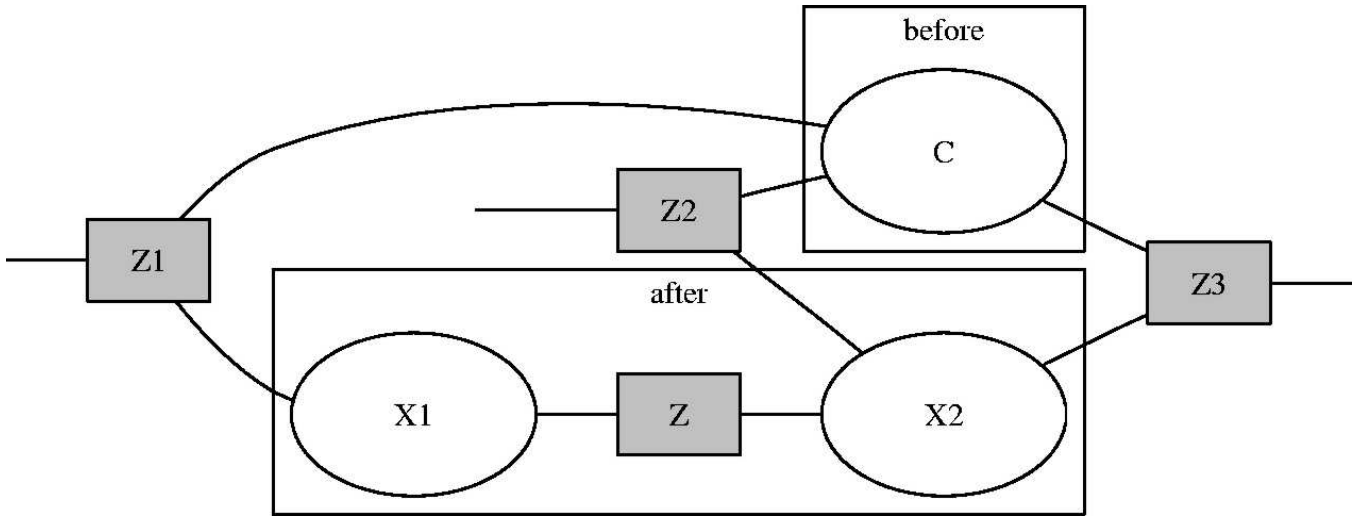


The maximum clique size is 3.

Overview

- Linear Elimination
- Clique Trees, Again
- **Building Good Clique Trees**
- Computation on Clique Trees

Building Clique Trees



- We'll build clique trees top down, splitting a clique C at each stage and reassigning its neigbors to one part of the split.
- Involves using the standard independence operation and graph partitioning.
- Involves insuring neighbors can be adequately reassigned: each separating set must be wholly contained in one side of the split.

Building Clique Trees

Here's a general purpose algorithm.

Repeat until no clique in the tree can be further split (at step 2).

1. Choose a clique C and form $\mathscr{Z}$ the set of all existing separating sets connected to the clique.
2. Find a decomposition/independence in C, $X_1 \perp\!\!\!\perp X_2 | Z$ where $Z = X_1 \cap X_2$, $C = X_1 \cup X_2$ and such that for all $Z' \in \mathscr{Z}, Z' \subseteq X_1$ or $Z' \subseteq X_2$.
3. Split the clique C into X_1 and X_2 and attach the separating sets to either according to their membership, assigning arbitrarily if in both.

NB. related theory in Bodlaender, 1997. **NB.** One method for choosing independent sets would be to use good hypergraph partitioning software with appropriate constraints on partitions.

Tree-based Elimination also Builds Clique Trees

- Note the algorithm for building clique trees is symmetric in X_1 and X_2 in step 2 and any clique can be selected for further expansion.
- We merge the previous Linear Elimination algorithm and the Clique Tree Building algorithm into a Tree-Based Elimination algorithm as follows: In the clique tree algorithm at step 2, force $X_1 / \cup_{Z' \in \mathscr{Z}} Z'$ to be a singleton set.
  - The algorithm eliminates a single variable in each cycle.
  - X_2 must be chosen in the next loop at step 1.
- This spawns off cliques containing only a single variable in addition to separating variables, thus "eliminating" the variable.

Tree-based Elimination, cont

The algorithm works with a main clique C_0, (uneliminated variables) initialized to the set of all variables X, and updates the undirected graph with extra arcs.

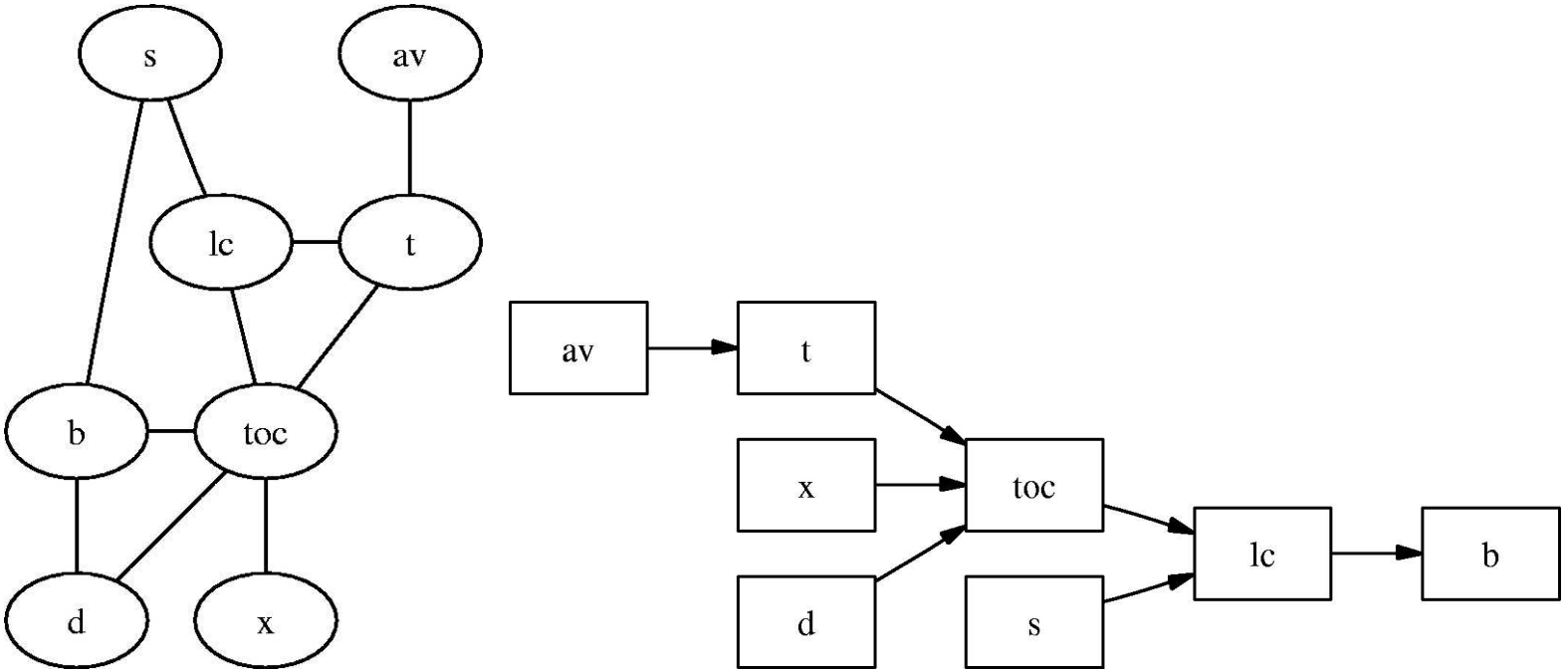
Repeat until C_0 a singleton.

1. Choose a variable $x \in C_0$, *i.e.*, not yet eliminated.
2. The new separating set $Z = \text{nbrs}(x) \cap C_0$, the new clique is $C_x = \{x\} \cup Z$, remove x from C_0, and "fill in" the graph to make C_x a clique (may affect subsequent use of $\text{nbrs}(\cdot)$).
3. For all cliques connected to C_0, split them into those with separating sets in C_x and those not. Spawn the new clique C_x off of C_0 and allocate the connected cliques to C_x or C_0.

NB. C_x formed from neighbors of x, thus any existing separating set either contains x and is wholly absorbed in C_x, or does not contain x and thus is contained in the new C_0.

Tree-based Elimination, cont

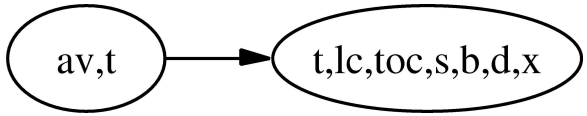
Here's an example on our favorite DAG using elimination order:



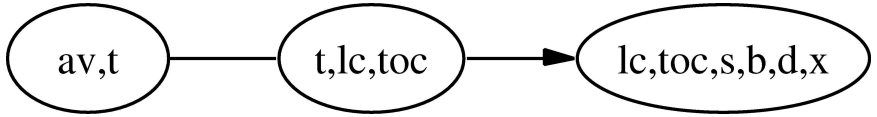
Tree-based Elimination, cont

The first few steps:

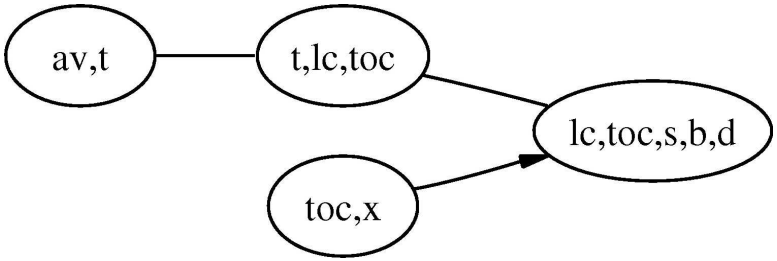
(1) eliminate *av*



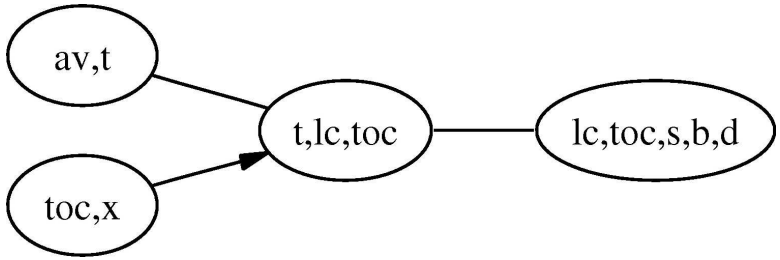
(2) eliminate *t*



(3) eliminate *x*



or

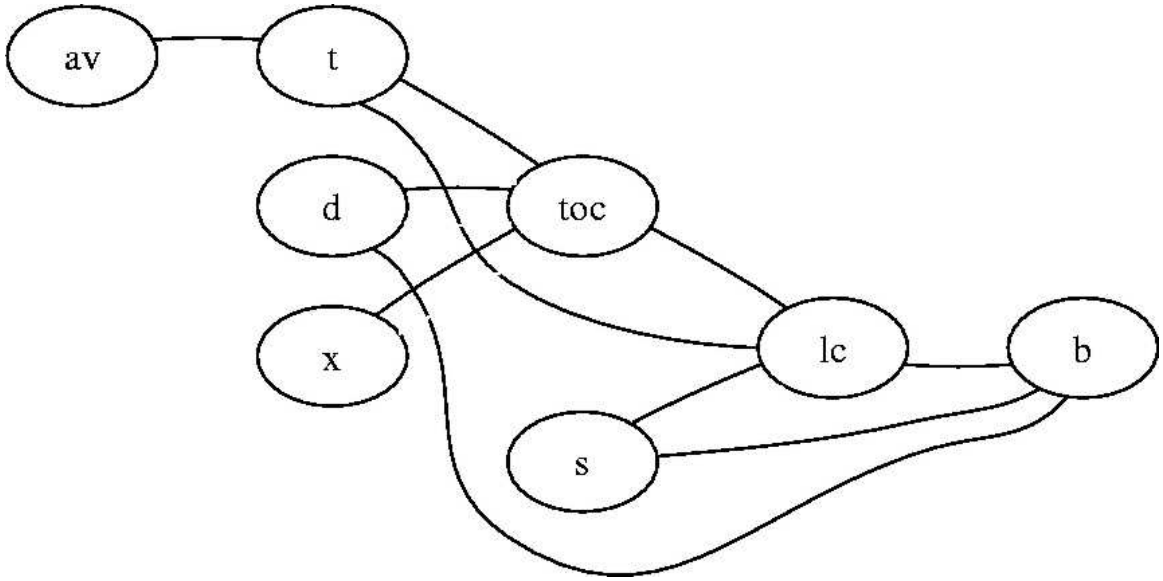


NB. new clique can attach to either of two nodes.

Tree-based Elimination, heuristic

- While several suggestions exist for selecting x, the next variable to eliminate, a common one is: choose x to minimize the number of arcs filled in at that step. i.e., the number of arcs added to make Z or C_x a clique.
- Called *minimum fill* by Kjaerulff 1990 and attributed to Rose, 1973.
- Kjaerulff also suggests a method to removed unnecessary fill-ins after completion.

Tree-based Elimination, cont

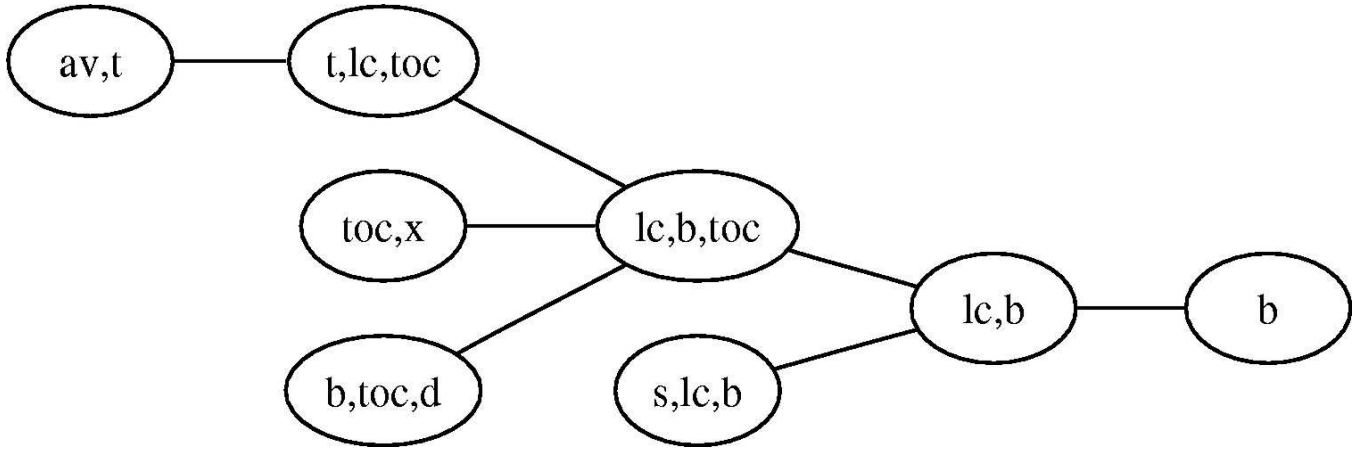


At each cycle of the algorithm, the new Z is also the cut set on the hierarchical layout to the right of the node just eliminated. Therefore:

> A tree on variables X is a valid elimination order for an undirected graph if and only if no variables in different branches are connected by an arc.

Improving Clique Trees

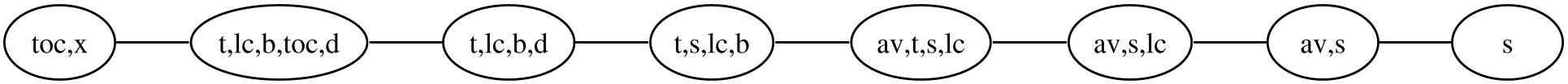
Final clique tree is exactly the same as the partial order, with the variables to eliminate replace by their clique. But elimination builds messy trees.



Here we can merge the nodes for b and lc, b into s, lc, b.

Improving Clique Trees, cont

Here's the clique true built using elimination by the linear order.



Here we remove the third node for t, lc, b, d as well as the three on the right.

Reduction Algorithm: for every clique that is a subset of its neighbors, merge it with one neighbor. Break ties arbitrarily.

Clique Trees and Elimination Orderings

A clique tree that cannot be reduced is called *irreducible*. Every irreducible clique tree can be generated by a tree-based elimination ordering following by a reduction step.

Proof Sketch:

> Orient the clique tree with a root. Every child node must contain a new variable not contained in one of its ancestors in the oriented clique tree. If it contains several, add extra cliques between the two until the children contain just one additional variable. Call this the intermediate clique tree. Create a matching undirected graph and run the tree-based elimination using the variable ordering given by the tree. This will reproduce the intermediate clique tree. Now reduce the added cliques by merging them into the original child. Thus reproducing the original clique tree.

Overview

- Linear Elimination
- Clique Trees, Again
- Building Good Clique Trees
- **Computation on Clique Trees**

Computation on Clique Trees

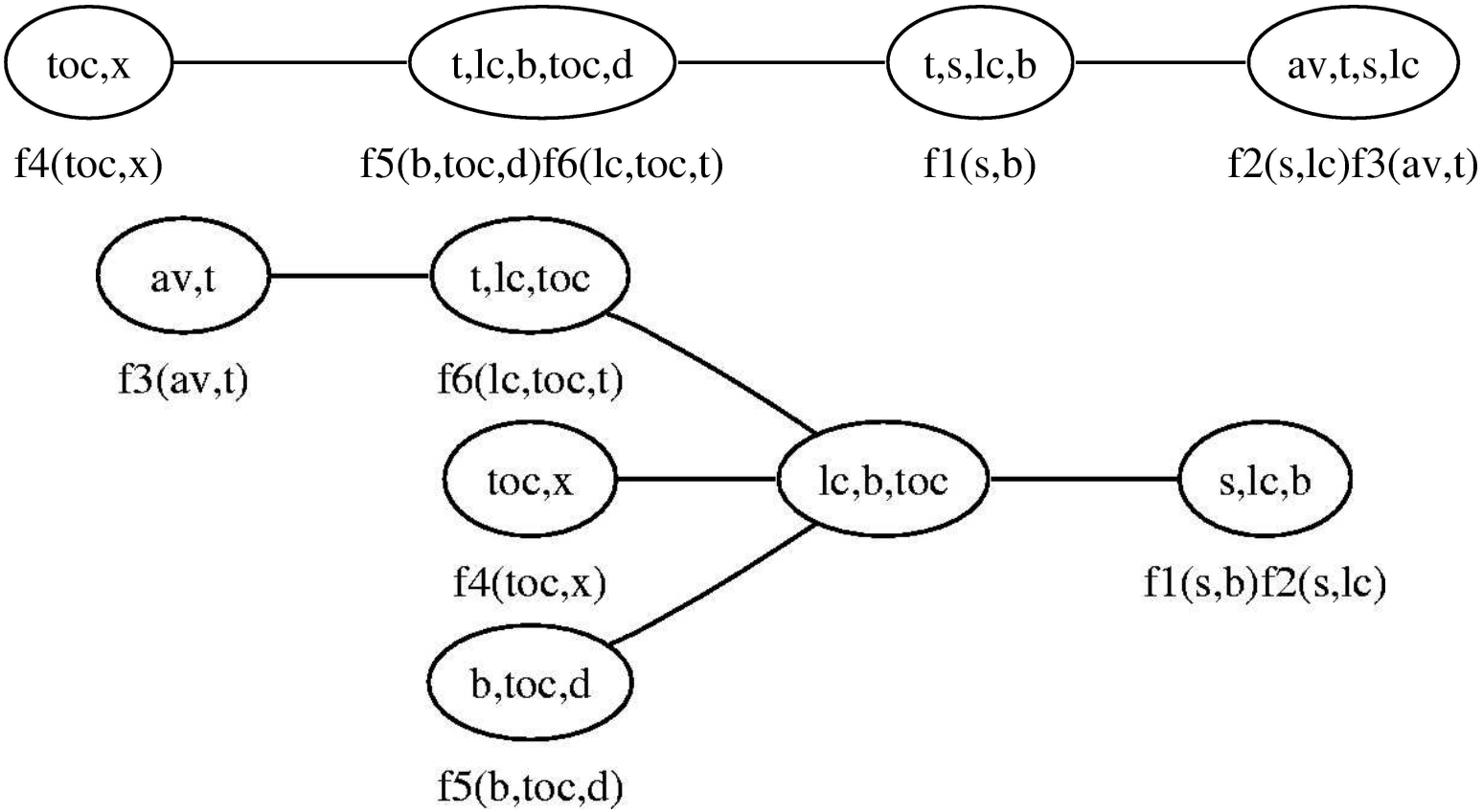
- Covered two node case in discussion of independence and problem decomposition. Here we generalize.
- In the general case, the clique tree starts out without properly normalized probabilities and the effort is to create these at each clique.
- **Step 1: Initialize:**

  Assign each clique function $f_D(X_D)$ (for D a clique in the undirected graph) arbitrarily to a clique in the clique tree that contains D. Aggregrate these so each clique C in the clique tree has its function $g_C(X_C)$.

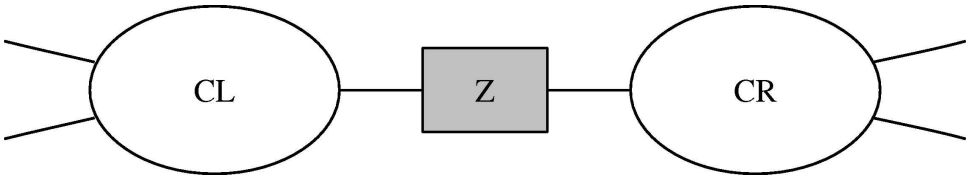
Initialization, example

Here's two examples on our favorite DAG.

$$f_1(s,b)\, f_2(s,lc)\, f_3(av,t)\, f_4(toc,x)\, f_5(b,toc,d)\, f_6(lc,toc,t)$$



Computation, cont.



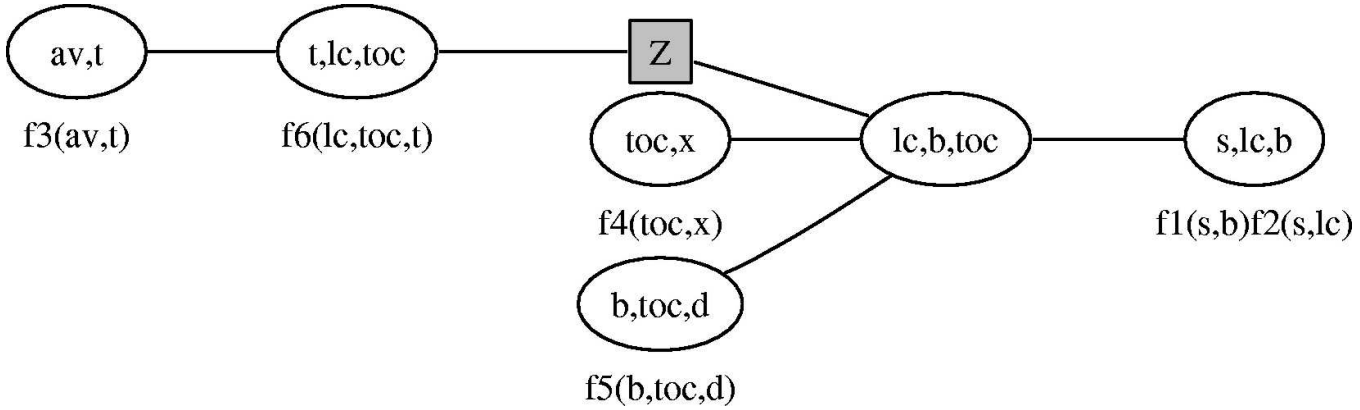
- The algorithm sends messages across each separating set in both directions in a bidirectional sweep.
- Have separating set Z between cliques C_R on the right and C_L on the left. Let X_R be all variables in cliques to the right (C_R and others). The message (computed recursively in the algorithm)

$$m_{C_R,C_L}(X_Z) = \sum_{X_R/Z} \prod_{C \text{ on right of } Z} g_C(X_C)$$

  is the message from C_R to C_L.
- These contribute to a solution as follows:

$$\begin{aligned} p(Z) &= m_{C_R,C_L}(X_Z) m_{C_L,C_R}(X_Z) \\ p(C) &= g_C(X_C) \prod_{C' \text{ connects to } C \text{ with sep.set } Z} m_{C',C}(X_Z) \end{aligned}$$

Computation, example



The message going to the left for separating set Z

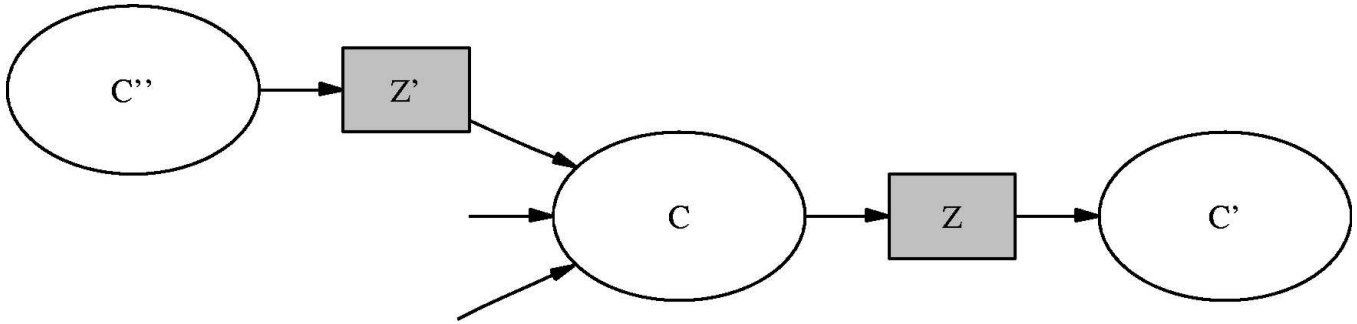
$$m_L(lc,toc) = \sum_{x,b,d,s} f_1(s,b)\, f_2(s,lc)\, f_4(toc,x)\, f_5(b,toc,d)$$

The message going to the right

$$m_R(lc,toc) = \sum_{av,t} f_3(av,t)\, f_6(lc,toc,t)$$

Finally, $p(lc,toc) = m_L(lc,toc) m_R(lc,toc)$.

Computation, cont.



- **Step 2: Make Consistent.**
  Do in parallel for each clique C and each separating set Z to another clique C': when all other separating sets Z' connecting C'' for C have received messages $m_{C'',C}(X_{Z'})$, then send message

$$m_{C,C'}(X_Z) = \sum_{C/Z} g_C(X_C) \prod_{\text{sep.set } (Z',C'')\neq(Z,C')} m_{C'',C}(X_{Z'})$$

- Computation in the inner loop is $O(2^{|C|})$ for boolean variables.

Computation, cont.

- **Step 3: Solve.**
  When everything has settled, the local clique probabilities can be computed from:

$$p(C) = g_C(X_C) \prod_{C' \text{ connects to } C \text{ with sep.set } Z} m_{C',C}(X_Z)$$

  so can be solved to give an answer.
- The *tree-width* of the clique tree is $T = \max_j |X_j| - 1$, one off the size of the largest clique.
- Many NP-complete problems solvable in $O(C2^T)$ for C the number of cliques in the tree for the problem.

Implementation Notes

- Parallel message passing made sequential as follows:

  Choose a single node as root and orient tree accordingly. Start message passing upwards from leaves by postorder traversal to pass messages up the tree, to root, and then preorder traversal to pass messages in opposite direction down to leaves.
- The algorithm relies on repeatedly summing over large multi-dimensional tables.
- Thus construction of the loops so they operate quickly is the key step. *cf.* Lapack.
- The tables in each clique need to be marginalized in different orders, for different separating sets, so the ordering of variables in the table is important.

Next Week

- Review Prof. Tirri's notes on Proportions.