CSCE921 Scribe Notes: April 22, 2013

Zion Schell

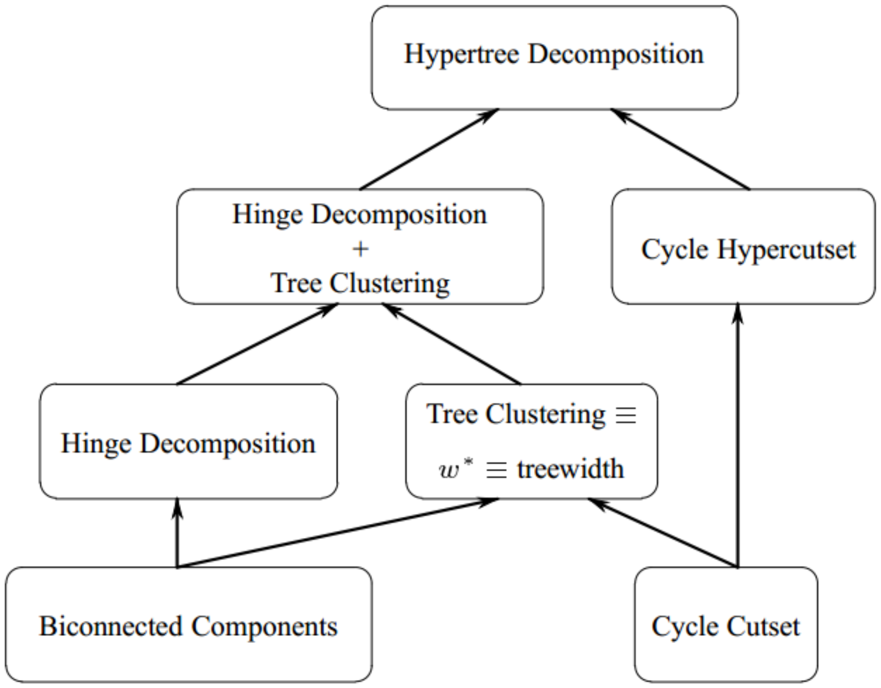
1 Class Overview

We reviewed the methods of tree decomposition.

2 The Beginning Bit (Motivation for Tree Decomposition)

The reason why we might want to tree-decompose a CSP instance is because recognizing an underlying tree structure allows us to solve it from a tree-ey perspective. More specifically, this helps to recognize cyclic dependencies, allowing backtracking to be confined to each individual node of the tree structure. With an appropriate tree decomposition, it may even prove possible to solve the CSP in a functionally backtrack-free fashion.

To illustrate the dichotomy of tree decomposition methods, Daniel showed us this image, to which Dr. Choueiry responded with feigned Socratic confusion. Daniel's initial attempt at elaboration was that the "higher" elements of the graph were "more general" than the lower ones. Dr Choueiry's response was that the higher ones in the graph have the lower (less-general, if we follow Daniel's assertion to its logical conclusion) ones as subsets of decompositions they can provide. Everyone seemed to understand, so we moved on.



To reiterate, the higher methods of tree decomposition in this graph are able to create any decompositions created by the lower ones.

3 The Middle Bit (The Tree Decompositions We Know)

The following section enumerates and elaborates the various varieties of tree decomposition discussed throughout the lecture.

3.1 The Front Bit of the Middle Bit (Tree Decomposition)

A tree decomposition is just that same thing we all know ~~and love~~ how to use: A tree T, with vertices and edges, and two functions, χ and λ, such that for each vertex v in T, $\chi(v)$ is a set of vertices from the CSP and $\lambda(v)$ is a set of edges from the CSP, with χ and λ having the properties that A) no variable from the CSP is unconstrained, B) each constraint appears alongside its scope at least once, and C) the connectedness property over tree decompositions (there must be a path in T between any two instances of a variable such that every vertex in the tree along that path contains the variable).

But let's be honest: you already know this. Let's move on from this reiteration of trivia.

The new stuff is that every tree decomposition method does all of these things, but also may impose more restrictions on the tree T and functions χ and λ.

Furthermore, the **width** of a tree decomposition is the minimum width of any tree decomposition possibly have. Each tree decomposition method can have its own definition of a separate but similar idea: *treewidth*.

3.2 The Front Middle Bit of the Middle Bit (Biconnected Components)

A **biconnected component** C is maximal set of vertices such that C is a tree node, the subgraph $\{C\}$ is connected, and the removal of no vertex in C would disconnect the original graph.

We only care about this because it is "well known" that a tree decomposition can be formed in linear time if every node in the tree is either a biconnected component, or contains a single vertex, the removal of which would disconnect the tree. You find the components in linear time $O(|V|+|E|)$, stick it in a tree in constant time, and find a perfect elimination ordering pretty quickly, and you've got yourself a tree decomposition (which you can probably solve in no time flat (speaking strictly, it can be solved in exponential time on the largest connected component)).

The width of a biconnected component tree decomposition is the maximum number of vertices in any node of the tree minus 1.

3.3 The Back Middle Bit of the Middle Bit (Hinge Decompositions)

Hinge decompositions form the general case of biconnected component decompositions; so given two disjoint sets of edges (A and B), A is "connected" with regards to B iff for any pair of edges in A, there is a path between them that doesn't go through any element of B. The hinge decomposition of a CSP just uses maximal "connected" sets of edges to form λ.

The width of a hinge decomposition is the cardinality of the largest minimal "connected" set of edges.

3.4 The Back Bit of the Middle Bit (Tree Clustering)

Tree clustering triangulates the graph of the CSP, finds the maximal cliques, and generates a tree structure over these (then it of course solves them and off we go).

4 The Back Bit (Hypertree Decomposition)

Remember tree decompositions? If so, then I bet you understand at least partially the ideas behind hypertree decompositions.

We're just going to add two things to the definition (aside from allowing it to be a hypertree instead of merely a tree): all variables in the tree nodes must be in the scope of one of the constraints in the same tree node (so add constraints to the tree nodes if necessary), AND for every tree node, variables in the scope of one of the constraints and also present in the subtree of that tree node must be placed in the tree node. In Daniel's somewhat confusing initial phrasing, variables are propagated up the tree as far as they can be if they are under the scope of the constraints.

The treewidth parameter for the hypertree decomposition is called *hyperwidth* and and is the smallest largest constraint cardinality of any tree node in any hypertree decomposition.

In summary of this mystical hypertree decomposition idea, a CSP whose width is bounded to k can be solved in $O(||I||^{k+1} log||I||)$, which is kind of nifty, although, as Daniel pointed out, this is mostly for theoretical purposes to allow Gottlob to prove what he was interested in: Hypertree decompositions are the most general of all of the discussed tree decompositions.