CSCE921 Scribe Notes – January 30, 2013

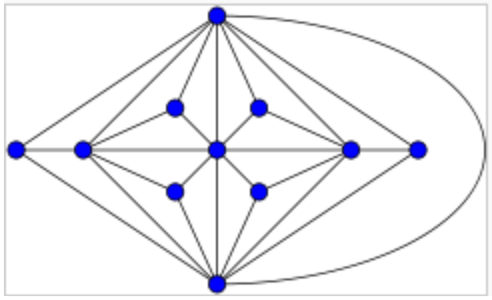
Olufikayo Adetunji

Answers to questions from Chapter 4 of "Constraint Processing" by Rina Dechter
Topic: Directional Consistencies

Daniel Dobo's questions:

Question: Section 4.1.3 covers *k*-trees, then the book never mentions them again. Is there anything else we should know about these trees, or was the section just included for completeness?

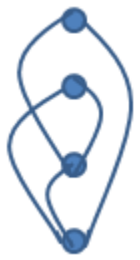
Answer: "In graph theory, a k-tree is a chordal graph all of whose maximal cliques are the same size k+ 1 and all of whose minimal clique separators are also all the same size k."[Wikipedia]



The Goldner–Harary graph, an example of a planar 3-tree.[Wikipedia]

K-tree – Structure of a tree where every node is connected to k nodes. To the best of the instructor's recollection, it has not been used/applied in contexts other than the one mentioned in Dechter's book. This statement does not however preclude finding an interesting use for it in the future. Otherwise, k-trees are an important concept in graph theory.

The K-tree structure guarantees a width of k, because when you order the vertices of a clique of size k+1 in a linear order, the bottom vertex is has k parents. For example, a 3-tree has a clique of size 4.



Question: On page 86 of the textbook, the author mentions tractability by restricted structure and tractability by restricted relations, each of which is covered in its own chapter. Can structures and relations always be exploited independently of each other, or are there some cases where an exploit depends on both the structure and the relations involved?

Answer: An example of tractability by restricted structure is tree decomposition, series-parallel networks (see [Montanri 74]). Examples of tractability by relations semantics is convex constraints, semi-convex constraints, restricted forms of Allen's algebra, etc.

The instructor seems to remember that some forms of causal reasoning (directed graphs) illustrate a case where both constraint semantics and network topology contribute to ensuring tractability. Causal networks express causal relations like in uncertainty (for more on causal networks, refer to Judea Pearl's book). They are typically directed graphs and their constraints typically have restricted.

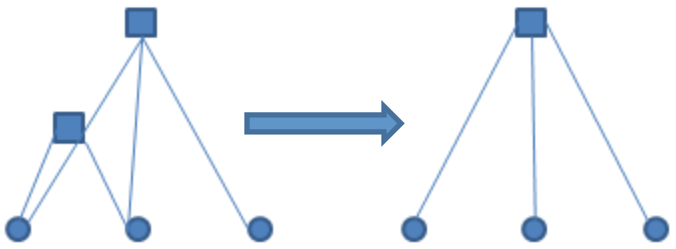
<u>Nathan Stender's question:</u>

Question: On page 110 of the book, it is mentioned that deciding if the induced width is less than a constant *b* can be done in polynomial time and that the class of constraint problems whose induced width is bounded by *b* is solvable in polynomial time and space *. I am wondering, how can the induced width go above the degree of the complete graph? Can multiple constraints exist between variables? Otherwise if would seem to me that the induced width is bounded by the degree of the complete graph.

Any problem that can be solved by an algorithm in $O(n^k)$ where k is fixed is by definition tractable because we have an algorithm whose cost is a polynomial of fixed degree. So, if you fix the width, then the problem becomes tractable because the problem can be solved by a polynomial-cost algorithm no matter how many variables there are.

In the case of a complete graph, the width is n-1, which is not a constant. The cost of solving the CSP is $O(n^{n-1})$, which is not a polynomial but grows exponentially with n.

Multiple constraints between variables are usually normalized: they are replaced by their intersection/join, at least as far as the theory is concerned. In practice, you may want to keep original constraints for preserving expressiveness.



<u>Olufikayo Adetunji's question:</u>

In what situation does inducing the width of a graph worsen the performance of running all types or a specific type of backtracking search algorithm?

What condition for induced widths guarantees backtrack free search?

All types of backtrack search are affected by the condition on the width, which is a worst-case figure. The idea of induced width is that a given variable has many parents, and, if you encounter a dead-end you may have to backtrack through all the parents and this effort could be as much as d^{w^*} *in the worst case. Theoretically, you have to enforce w*+1 consistency in order to ensure backtrack-free (*BT-free*) search. Now, this condition is a necessary condition and this bound is an upper bound, a worst-case figure. Some problems, fortunately, are solved with less effort and before we explore all partial solutions on the parents.*

So, the higher the level of consistency you enforce during lookahead the more you reduce the chances of backtracking (so full leakahead is desirable, but if the problem is too simple and a solution is easily found, then you may be losing time enforcing high consistency).

In practice, you do not always need to enforce such a high consistency level. It can be too costly. Furhter, you may change the topology of the constraint network if you are not careful, which has been the main reason for not enforcing higher level consistencies. Ideally, you want to examine the topology of the graph, estimate the difficulty of the problem and choose, perhaps even locally the level of consistency that needs to be enforced to get to BT-free search. We are currently working on this new research direction where we examine some structural parameters of the CSP and the tightness of the constraints to estimate the level of consistency that we need to enforce for backtrack free search. It is a dream.

Can you do worse that w+1? Of course, if you take an ordering that yields a higher value of the treewidth. For example, assume you have a tree-structured CSP, if you choose the vertex of largest degree (not a leaf) to be at the bottom of the instantiation order, then you have the ordering of the worst possible width/induced width. You may have lots of backtracking although you could have solved the problem in a BT-free manner with directional arc-consistency.*

Reminder:

- *Width of a node in an ordering = # of its parents*
- *Width of an ordering = Max(Width of node)*
- *Width of a graph = Min(Width on all orderings)*

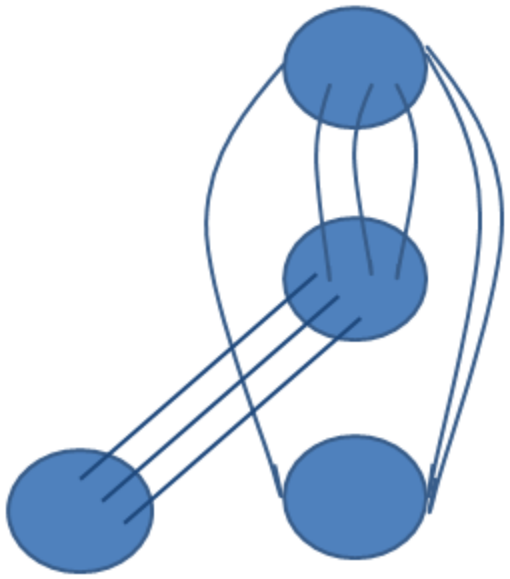
<u>Tony Schneider's question:</u>

The idea is to select an ordering with a small width, compute the induced width, then apply strong directional (w*(d)+1)-consistency (page 104). However, applying that level of consistency would likely alter the topology of the graph, as well as the induced width, correct? If that's the case (and it may not be) wouldn't the level of directional consistency to enforce increase with the induced width? If so, how is this method bounded (and is it still a viable algorithm), and if not, where did my understanding lapse?

When you are enforcing the consistency level required by the sufficient condition relating induced width and consistency level, you may have to add new constraints that modify the topology of the graph and increase the width. And it is true that enforcing the required consistency can increase the induced width. For this reason the result was not exploited in *practice except by Dechter in Tree Clustering (see below). Is the result still viable/useful? The instructor can think of two approaches:*

1- *Consistency algorithms that do not increase the width: Shant and Robert have recently been exploring consistency mechanisms that enforce higher levels of consistency without modifying the topology of the graph.*
2- *All the research done by Dechter since 1989: tree clustering, bucket elimination, etc.*

In the tree clustering algorithm, Dechter exploits a tree decomposition to solve each cluster independently (generating the global constraint on the cluster) then communicate the results among clusters. Consider the example of the tree decomposition below:



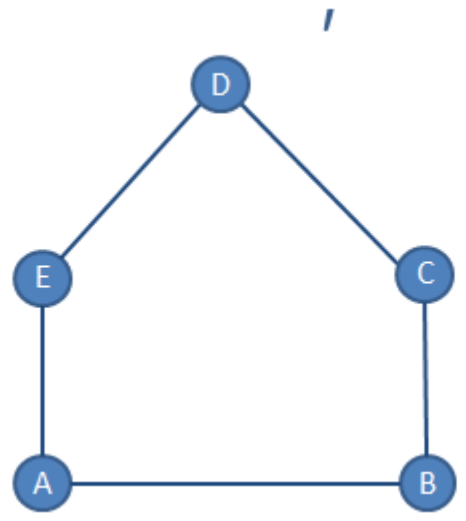
The above diagram shows a form of decomposition where each vertex has one parent and they communicate via common variables. The largest subproblem in the tree decomposition has w vertices, which means that in the worst case, the induced width there is w*.*
Consider each subproblem, solve it and list all its solutions, which require $O(d^{w})$ effort. Now take each subproblem as a variable and the solutions as values. Then run a directional `arc' consistency to ensure the solutions of a variable agree with those of a connected variable, filtering out the ones that are not consistent. This method guarantees backtrack-free search, it is not equivalent to enforcing w+1-consistency on the entire problem, but it does enforce w*-consistency on each subproblem (because it solves it!) and directional arc-consistency between subproblems.*

<u>Nathan Stender's question:</u>

In example 4.16 of the book demonstrating bucket elimination for two different orderings, both end up with a single unary relation on the final variable of the ordering, but the meaning of this relation is not mentioned (that I could find). I understand how/why the binary and ternary relations are created, but I cannot see what this unary relation should mean.

Short answer: The unary constraint(s) is the domain of the variable.
Bucket Elimination technique is: "Take the instantiation (top down) or the elimination ordering (bottom up). Start from the bottom variable and create a bucket for it, by listing all constraints

pertaining to it but have not yet appeared yet. By the time you get to the variable at the top, you will have just that variable and as such guarantee consistency." Consider the example below:



A All that is left now is R_A, which will produce a more filtered version when joint to R_A.

B Join R_{AB} and R_{AB} to get R_{AB}. Project this to A to get R_A

C Join R_{CB} and R_{ABC} to get R_{ABC}. Project this to B to get R_{AB}

D Join R_{AD} and R_{DCB} to get R_{ADBC}. Project this to C to get R_{ABC}

E Starting from E. Join the relations of E. $R_{ED} \bowtie R_{EC} \bowtie R_{EB} = R_{EDCB}$. Now project this on D to get R_{DCB}

This process is directly related to Guassian elimination, where you use a linear combination between one row and all other rows to eliminate the influence of one variable.

The final relation in A is just the filtered domain of A. The remaining values in A are guaranteed to participate in a solution.