Summary of Class Discussion on Directional Consistency (Sections 4.2 and 4.3)

Scribe: Anagh Lal

February 21, 2003

1 Discussion on backtrack-free network

The class discussed the definition of backtrack-free network, which was initially unclear. A BT-free network is that any partial *consistent* instantiation is guaranteed to be extendible to a complete solution. This does *not* mean that any combination of values for variables is a solution. So, the problem may still have no-goods (i.e., combinations of values to variables that are not solutions). A BT-free network is said to be consistent in the sense that it is guaranteed to have a solution.

2 Section 4.2: Graph concepts

1. The following concepts were introduced and discussed: an ordering (d), the width of a node, the width of an ordering and the width of a graph. Examples were reviewed. As a reminder:

   - The width of a node in an ordering is the number of its parents in the ordering.
   - The width of an ordering is the maximum of the widths of its nodes.
   - The width of graph is the minimum of the widths of all its orderings.
   - To find the width of a graph it is not feasible to list out all orderings (there is a factorial number of them).
   - Fortunately, a greedy algorithm to find the width of a graph and the corresponding exists, the Minimum Width algorithm (MW). It is shown in Figure 4.2, page 96 of [1] along with an example and it was discussed in CSCE 421/821 (see instructor notes #9).

   Dechter adopts the convention of drawing an ordering in the reverse order of the search: the last node to be instantiated is listed on top.

2.1 Induced width and induced graph

(a) The induced width of a graph under ordering d is the width of its induced graph.

(b) The induced graph of an ordering d is obtained by moralizing the original graph. The term moralizing the graph was revisited, as it was introduced CSCE 421-821.

Given an ordering d, the induced width of the ordering and the corresponding induced graph can be found as follows. The original graph is processed from last node to first node in the ordering. For every node, the parents of the node are moralized, that is connected.

(c) The induced width of a graph is the minimal induced width over all its orderings.

(d) Finding the *minimum* induced width of a graph is **NP**-hard however, a linear-time approximation exists: the Minimum Induced Width (MIW) algorithm, Fig. 43 page 97 of [1].

(e) The MIW algorithm is a modification of the min-width (MW) algorithm and works as follows. From the input graph the algorithm selects a node with the minimum degree and places it last in the ordering. The neighbors of the node are connected, after which the node and the edges adjacent to it are removed from the graph. The node with the minimum degree in the resulting graph is then selected, and this process is applied recursively till all the nodes have been processed and placed in the ordering list. Ties (i.e., two or more nodes having the same degree) are broken arbitrarily. The width of this ordering gives an approximation of the minimum-induced-width of the graph.

(f) Eric Moss reinforced the objective of studying the width and induced width as being to find graphs with special topologies that lend themselves to quick solutions.

2.2 Chordal graphs

(a) Many difficult graph problems (e.g., like finding maximal cliques) become easy on chordal graphs. Chordal graphs are a sub-class of perfect graphs. Finding max-cliques is **NP**-hard in general, but for chordal graphs, it can be done in linear time.

(b) It was noted that finding max-cliques was not approximable, i.e. there is no polynomial-time algorithm that can find max-cliques within a given threshold, for all possible thresholds.

(c) The Maximal-Cardinality Ordering algorithm of Fig 4.4 on page 98 on [1] is

(d) In chordal graphs, maximal cliques can be found with the algorithm Maximal-Cardinality (MC) Ordering given in Fig 4.4, page 98 of [1].

(e) MC is in fact the algorithm one uses to *recognize* chordal graphs. A graph is chordal iff in the max-cardinality ordering graph each vertex forms a clique with its parents. Eric Moss discussed an algorithm for making a graph chordal (which does not guarantee the minimum number of added edges).

(f) We can enumerate all max-cliques associated with each vertex by listing the sets of each vertex and its parents, and then identifying the maximal size of a clique.

(g) Note that, for a chordal graph, MIW generates the same graph as MC.

2.2.1 K-trees

Dechter mentions that a graph can be embedded in a k-tree iff it has an induced width $w^* \leq k$. This triggered an investigation of what is a k-tree. We try below to answer this question.

A k-tree can be defined inductively in the following way:

(a) A k-vertex complete graph is a k-tree.

(b) The graph obtained from a k-tree by adding a vertex adjacent to *each* vertex of a k-clique is also a k-tree.

We had hard time to parse this compact, but correct, definition. Consider the graph shown in Figure 1a. It is a four vertex complete graph, hence a 4-tree. Going by the definition, to add a vertex V5 to the graph and keep the graph a 4-tree, we need to connect V5 to 4 vertices that form a clique of size 4 (since, here, k = 4). These vertices are V1, V2, V3 and V4. The vertices $\{$V1, V2, V3, V4, V5$\}$ form the max-clique of the graph and the size is 5, which is equal to $(k + 1)$. The vertices $\{$V5, V4, V3, V2$\}$ form a clique of size 4 in the new graph, as shown in Figure 1b. Addition of a

new variable V6 to the graph of Figure 1b is shown in Figure 1c. Here, the variable V6 is connected to V5, V4, V3 and V2 (which form a clique of size 4), the new resulting clique is of size 5. The maximal clique size in this final graph is also 5, which is equal to $(k + 1)$.

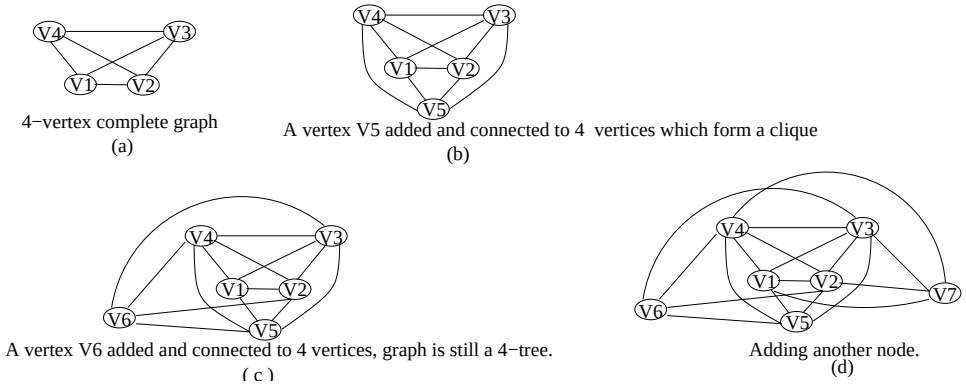

Figure 1: *Various stages of a k-tree (a) A k-tree initially having 4 vertices (b),(c) and (d) show addition of new vertices so that the graph remains a k-tree. Here k = 4.*

Such a graph is called a k tree for the following reason. Each maximal cliques (which have size $(k + 1)$) can be grouped under a single node (representing $(k + 1)$ of the original vertices). These new nodes can be arranged in a tree structure in which the neighbors (i.e., parent and children) of each node share exactly k of the original vertices with the node. In Figure 2 we show the k-tree corresponding to the example of Figure 1d.

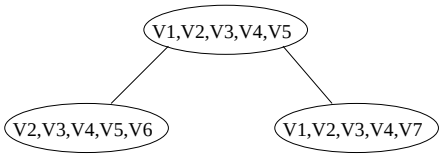

Figure 2: *Tree decomposition of Figure 1d.*

3 Section 4.33: Directional consistency

Next, Eric introduced Directional Arc-Consistency (DAC), Directional Path-Consistency (DPC), and Directional i-Consistency (DIC).

3.1 Directional arc consistency

1. Directional Arc-Consistency (DAC) ensures that in a given ordering, you can pick a value from any variable and know that it has support from variables that come later in the ordering. Hence, while enforcing this property, we start from the last variable V_i in the ordering and we update the domains (or higher order relations) of the variables earlier in the search ordering ($V_{j<i}$) to keep only those values (or tuples) that are supported by V_i.

2. Directional Arc-Consistency does not guarantee BT-free search but is a cheaper alternative to arc-consistency as we revise, for every variable, the domains of only those variables that appear earlier in the search order and not those that appear later.

3. The algorithm for enforcing DAC is given in Figure 4.6, page 101 of [1]. It updates the *domain* of variables.

3.2 Directional path-consistency

1. The concept of constraint composition was revisited and its relevance to path-consistency was explained. Let V_1, V_2, V_3 be three variables, and let R_{12} and R_{23} be the two constraints defined on these variables. We can see that there is one common variable, V_2, between the two constraints. The two constraint restricts the values the variables V_1 and V_3 can take at the same time, even though they do not have an explicit constraint between them. Based on the restrictions imposed by the constraints R_{12} and R_{23}, we can impose a new, induced constraint R_{13}. The constraint R_{13} can be found by first, listing the values the common variable V_2 can take and be consistent with the two constraints.

2. The algorithm for Directional Path-Consistency (DPC) maintains a list of edges to be visited for constraint propagation. It adds these edges to the constraint graph. The algorithm for Path-Consistency (PC-1) would visit every combination of the three nodes, whereas in DPC fewer combinations are visited: for a given variable we only consider those that appear earlier in the search ordering.

3. The DPC algorithm of Figure 4.6 results in a strongly directional path-consistent graph, i.e. it also makes the graph directional arc-consistent.

4. DPC results in a moralized graph, which has the min-induced-width w^*. Importantly, while MIW changes the topology of the graph, DPC also tightens the binary constraints, which label the old and new edges.

5. Eric Moss suggested a possible optimization to the DPC algorithm. If we know that there is only one parent for every node and that the given network is arc-consistent then the arc-consistency portion of the DPC algorithm can be skipped.

6. Typo: The Figure 4.8, page 104, of [1] caused confusion by not having the reference to the `Revise-3()` function properly commented.

3.3 Directional i-consistency

1. The algorithm for directional i-consistency uses a generalized version of the `Revise` algorithm. The generic algorithm takes, as a its first parameter, a set of any size. The algorithm for directional i-consistency picks up variables in the reverse order of the ordering d. For each variable it finds the set of parent variables. If the set of parents has exactly $(i-1)$ elements, then the algorithm induces just one new constraint of arity $(i-1)$ over the set of parents. Otherwise, if the set of parents has more that $(i-1)$ elements, then a constraint is recorded over every combination of $(i-1)$ subset of variables in this set.

2. If we have a CSP with constraint arities $\leq i$, then applying the directional i-consistency algorithm produces a graph that is subsumed by (or is a part of) the induced graph for the same ordering d.

4 Overall objective of the sections

In the sections ahead we will study that if an ordered constraint graph (ordering d) has width $(w-1)$ and if it is strong directional i-consistent then the constraint graph is BT-free for the ordering d. Thus

by studying the concepts of directional consistency we are able to exploit the structural properties of a constraint network to obtain solutions within a polynomially bounded amount of effort.

References

[1] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2002.