Scribe Notes of the class of 10/1/2009

Speakers: Chris and Peter

Scribe: Pingyu

Paper: Tree Clustering for Constraint Networks, by Rina Dechter and Judea Pearl

The two presentations and class discussion are summarized below.

Part 1

The first presentation was by Chris. Generally speaking, there are two ways to restrict CSPs in order to make finding the minimal CSP efficient:

1. either by restricting topology of the constraint graph, or
2. by restricting the types of constraints.

This paper follows the first strategy. It takes advantage of the desirable property of tree-shaped constraint graphs, which are solvable in polynomial time, and tries to propose different ways to structure problems into trees by creating a tree embedding of the constraint graph such as grouping variables into clusters. This paper proposes two techniques towards this goal and compares them: *tree clustering* and *adaptive consistency*.

Chris then discussed how a join graph can be obtained from the dual graph of a CSP:

- In a dual graph, the vertices are the constraints and edges are added between any two vertices that have common variables and annotated with the common variables.
- The join graph is obtained from the dual graph by removing redundant constraints while maintaining the connectedness property.
- A join graph that is a tree is called a join tree. When a CSP has a join tree, it is tractable. The instructor mentioned that recognizing a join tree can be done efficiently (see Dechter Section 9.2, page 249-250).

Chris also recalled the *connectedness property*.

Connectedness property: For each two nodes sharing a variable, there is at least one path of labeled arcs containing the shared variable. This property forces the same variable to carry the same value across clusters, which is crucial for T-C style reconstruction and problem solving.

He then presented the Tree Clustering (T-C) algorithm. The idea of this algorithm is to **s**imulate tree-solving algorithm on an embedding tree structure where each tree node is a cluster of original CSP nodes. Solve sub-problems in each cluster first, then combine them to form a complete solution. Chris explained the time complexity $O(nr \cdot k^r)$ and space complexity: $O(n \cdot k^r)$, where n is the number of variables, r the maximum arity of the constraints and k the domain size.

T-C Algorithm

1. Triangulate the primal graph
2. Identify all the maximal cliques in the primal chordal graph
3. Form a join tree
4. Solve the subproblems: Each cluster becomes single variable, its domain is the set of solutions to the subproblem induced by the variables in the cluster.
5. Solve the tree problem
   - Perform DAC from leaves to root
   - Instantiate the variables BT-free from root to leaves

Chris then presented the Adaptive Consistency (A-C) Algorithm. The idea of this algorithm is as follows: An ordered constraint graph is backtrack-free if the level of directional strong consistency along this order is greater than the width of the ordered graph. However, enforcing *i*-consistency for i>2 often requires adding new constraints that increase the width and violate the condition. Therefore, we define d-*i*-consistency recursively and let *i* change dynamically from node to node. Chris gave the time complexity as $O(n \cdot \exp(W^*(d) + 1))$ and the space complexity as $O(n \cdot k^{W^*(d)})$, where W* is defined as the width of the induced graph.

A-C Algorithm:

1. For i=n downto 1 do **Steps 2-4**
2. Compute PARENTS(X_i)
3. Connect all PARENTS(X_i)
4. Perform Consistency(X_i, PARENTS(X_i)) ≡ joining the constraints between X_i & its parents
5. Build a solution BT-free in the ordering (X_1, …, X_n)

Chris showed how the algorithm operates on a simple example.

The instructor asked as student, who was not paying attention to come up and repeat a run of the algorithm on the board.

Question by Robert: On Page 361, why the topology of the induced graph is identical to the one generated one by directional path-consistency? (Since it may introduce higher level of consistency)

Answer: Because the algorithm is working on the primal graph.

Chris then compared the two algorithms:

- The arcs resulting from triangulation in T-C match the arcs added by adaptive consistency, and every cluster in T-C is represented in A-C by a series of smaller constraints.
- T-C enumerates all solutions while A-C represents them via constraints. Therefore, A-C eliminates the redundancy of generating partial solutions that are to be discarded later. A-C can thus be considered superior to T-C.

Instructor asked: Why W*(d) + 1 = the size of the largest clique?

Wesley answered: The maximal size of the parents set is equal to the width of the induced graph, so the size of the largest clique would be W*(d) + 1.

Wesley: Any empirical study on T-C and A-C?

Answer: The instructor asked Shant to provide some insight. Shant explained that it is not practical execute T-C in practice because the cluster sizes can be large and enumerating all solutions to sub-problems is impractical. Hence, Jégou and Terrioux suggested using separators (variables shared by two adjacent clusters) and recording all good/no-good value pairs on them as they are discovered during

regular backtrack search on the clustered tree to avoid repeated checks. However, separators may grow bigger over time, eventually leads to large goods/no-goods which is also impractical.

Finally, Chris discussed the issue of graph triangulation, which is essential for the T-C algorithm. The problem is to find the triangulated graph with smallest maximum clique, but the problem is NP-hard. Instead, we have many heuristics. The basic operation is to choose a vertex in the graph, eliminate it, and connect all its neighbors in order to form a clique. The edges added to connect the non-connected neighbors are called the *fill edges*. Various heuristics exist:

- H1: choose the node w/ smallest degree
- H2: choose the node that, after elimination, yields the smallest number of fill edges
- H3: Given any ordering (e.g., maximal cardinality ordering), moralize the graph

Shant: We may try different heuristics and for each resulting order, compute the induced width, then choose the order with the minimum induced width.

Chris then reminded the class of the maximal cardinality ordering used to order nodes in A-C. We had seen this algorithm in 421/821 (as an approximation of minimum width ordering).

1. Choose a simplicial node (the algorithm in general uses an arbitrary node)
2. Among the remaining nodes, choose the one that is connected to the maximum number of already chosen nodes, break ties arbitrarily
3. Repeat until empty
4. Output in reverse order

Part 2

The second portion of the presentation was on finding the maximal cliques of a triangulated grapg and on building the join tree. It was given by Peter.

Algorithm for Maximal Cliques of the triangulated graph (Peter)

Input: A triangulated graph and a perfect elimination order for the nodes

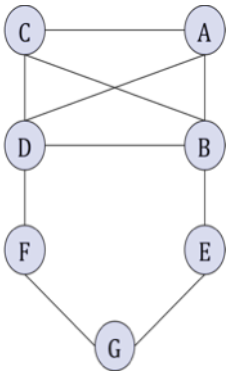
Output: Maximal cliques of the graph

- Initialize S(v) = 0 (where v is any variable in V)
- **For** i = 1 up to n

–Let v be the i^{th} variable in the ordering
–Let X be the set of variables adjacent to v that come **after** v in the ordering.
–**If** v has no neighbors, **then** put it in its own clique.
–**If** v has neighbors but all of them come before v in the ordering, **then** stop the algorithm – we're finished.
–Let u be the member of X that's earliest in the ordering.
–Let S(u) be the maximum of S(u) and $|X|$-1.
–**If** S(v) < $|X|$, **then** add a clique: v (union) X.

Peter demoed the algorithm with an example on the board.

For the graph below:



One possible perfect elimination order is A, C, B, D, E, F, G

Applying the above algorithm to the graph, we have the following results.

C_1={C, A, B, D}

C_2={B, D, E}

C_3={D, E, F}

C_4={E, F, G}

Algorithm for Join Tree of the triangulated graph (Peter)

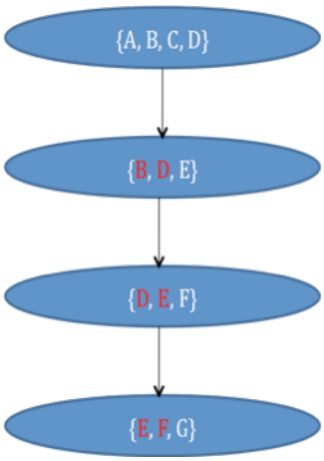
Input: A vector of cliques.

Output: A join-tree of cliques.

> •**For** every clique C_i in our vector C:
>
> –Connect C_i to the C_j ($j < i$) with which it shares the most variables

Peter continued with the previous example.

If we use the order as specified above, the resulting tree will be a chain



NOTE: The red variable names indicates nodes that are shared by a cluster and its parent. We can see from the graph that for each two clusters charing a variable, there is at least one path of labeled arcs containing the shared variable. Thus the connectedness property is maintained.

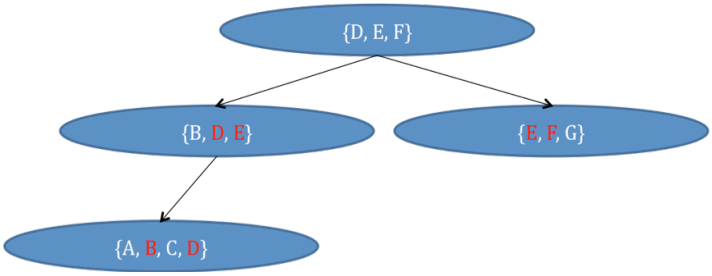
However, if we choose a different order, the shape of the resulting tree changes accordingly

C_1={D, E, F}

C_2={B, D, E}

C_3={E, F, G}

C_4={C, A, B, D}



The instructor further stressed the importance of the connectedness property, illustrating it on the generated graphs. This property forces the same variable to carry the same value across clusters, which is crucial to maintain the correctness of solution in T-C style reconstruction and problem solving.