CSE921 Notes for Feb 18, 2013
Topic: Structure-Based Methods.

Daniel D.

February 21, 2013

1 Questions

Q: Daniel D.: *Is bucket elimination implemented on top of some simpler algorithm?*

A: Bucket elimination is its own technique and has no relation to search unless you are using search on each bucked to compute the join. It is instead a form of constraint compilation.

Q: Robert: *This strategy is not typically used on interval domains?*

A: (Revised answer.) Interval domains would be represented by unary constraints on discrete or continuous domains. If we are talking about constraints of linear equalities, then the technique corresponds to Guassian elimination. If the constraints are arbitrary, and the interval are continuous, computing the join of the constraints in the bucket may be too difficult. Indeed, AC on continuous domains is relaxed into bound consistency. Path consistency is $O(n^5)$, see PhD thesis of Djamila Sam Haroud at EPFL in 1995.

2 Graphical Representation of CSPs: a Review

The constraint network is a graphical representation of a CSP.

- For a binary CSP, this network is a constraint graph.
- For a nonbinary CSP, this network is a hyper-graph (see Figure 1).

Other graphical representations are:

- For binary CSPs: the microstructure, the macrostructure, and the co-microstructure.
- For non-binary CSPs: the primal graph, the dual graph (or the join-graph).

Note that the primal graph of a binary CSP is the constraint graph itself.

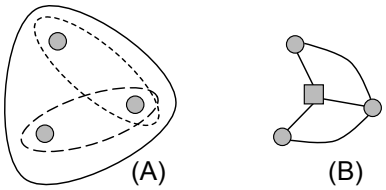

Figure 1: Two alternative graphical representations of a hypergraph.

3 Tree-Based Methods

When the constraint network (graph or hypergraph) is a tree, the CSP is easy to solve [3]. The problem is tractable. We can solve it directly in a backtrack-free manner as in the following general manner:

1. Propagate constraints directionally, up, from the leaves to the root, then,
2. Instantiate top down, from the root to the leaves, building a solution in a backtrack-free manner.

If a CSP does not have a tree structure, then we find some way to make it look like one. There are many ways to obtain a tree-structure or tree-like structure of a constraint network. In this lecture, we discussed a number of such techniques: cycle-cutset, dynamic dangle identification, bi-connected components, and tree clustering, bucket elimination, and tree decomposition (which defines a tree-embedding of the constraint network). The outline slide lists references with the name of the author and for further reading.

3.1 Cycle-cutset

Because trees are easy to solve, we prefer working with them. However, not every graph is a tree. One method of simulating a tree is to remove those vertices that cause cycles.

Definition 1 Cycle cutset *is a subset of a graph's vertices whose removal from the graph leave a tree. The cutset itself is not a tree in general.*

The main steps of this technique are:

1. Identify a cutset and partition the graph in two parts: a tree and the cycle cutest (see Figure 2).
2. Find a solution for the cycle cutset (by backtrack search or by local search).
3. Revise tree against the solution of the cycle cutset (directional arc-consistency).
4. Try to solve the tree (directional arc consistency).
5. If no solution is found, then the problem lies with the instantiation for the cycle cutset.
6. Get another solution for the cutset and try again from step one.

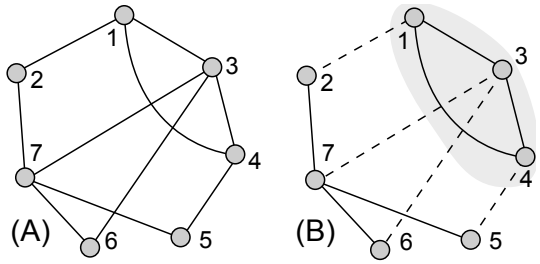

Figure 2: (A) The original graph, (B) graph with cutset (shaded region) removed, leaving behind a tree. The cutset may be, but is in general not, a tree.

The complexity is reduced from $O(d^n)$ to $O(nd^{c+2})$, where c is the size of the cycle cutset, so the bottleneck is the size of the cycle cutset. However, finding the smallest cycle cutset is NP-hard.

Q Robert: *Can the cycle-cutset tree be a forest?*

A Yes. We apply directional filtering on the nodes of the tree/forest when we have prospective solution to the cutset. Then we can solve each tree in the forest independently.

Q: Nate asked: *Could we solve the tree since c is harder?*

A: Yes, you could do DAC on the tree and test whether the problem can be discarded already because the tree does not have a solution (although, this may be unlikely to be the case). You could also start by enforcing AC (or any kind of pre-processing) on the cutset before you check the tree. There could also be other special properties that you may want to exploit. However, the point here is to show how the technique works in general. Then, you can go ahead and do any kind of refinement you wish.

Q: Tony asked: *Are there heuristics for the finding the cycle cutset?*

A: Rina must use one [2]. I do not remember what it is exactly, we must check her paper and textbook. Incidentally, Joel Gompert did an MS degree with me on a decomposition technique that is dual to the cycle cutset. He looks for (and exploits) the largest independant set (which is the complement in the graph of the minimal cycle cutset) [5, 6, 4].

Supplemental answer: Rina offers a clever solution that does not neccesarily 'look' for a cycle cutset, but takes advantage of one when it is encountered. From Dechter, pp. 275, *"Because backtracking works by progressively instantiating sets of variables, we only need to keep track of the connectivity status of the constraint graph. As soon as the set of instantiated variables constitutes a cycle-cutset, the search algorithm is switched to the tree-solving algorithm on the restricted problem..."* While this itself does not propose a heuristic, it builds up to the following on the same page *"Some simple heuristic ordering of the variables that aims at finding a small cycle-cutset could be very beneficial here. One is to order the variables in decreasing order of their degree in the constraint graph."*

3.2 Dynamic Dangle Identification

Based on Graham's graph reduction operation, which consists in iteratively removing from a graph a vertex of degree one and was first proposed to identify acyclic networks [10]. The strategy here is to identify a vertex of degree one, do DAC on its neighbor (i.e., revise the neighbor given the node), and remove it because it has no further effect on the network.

- Every vertex with degree 1 is a dangle.
- Revise its neighbor given the variable, and remove it (see Figure 3.)

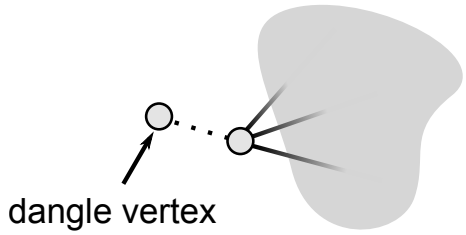

Figure 3: A dangle is a variable of degree one and can be eliminated once its neighbor has been made consistent with it.

- When a variable is instantiated during search, we do some kind of lookahead, then we remove the edges between the variable and its neighbors, reducing the degree of the variables, so we look again for dangles.

Q: Robert: *Can we estimate what instantiation would create the most dangles?*

A: (Revised answer) Yes, it is a good idea and would correspond to the degree ordering (see note in "Supplemental answer" above). Yaling Zheng had tested several orderings, wrote a paper, but it was never published. She tried several orderings.

3.3 Bi-Connected Components

Definition 2 Bi-connected component *If a simple graph has a vertex whose removal results in separating the graph into two or more independent graphs, then the induced subgraphs are called* bi-connected components *and their separator is called a* cut-vertex *or* articulation point*.*

We can overlay a tree-like structure atop a CSP by organizing it into independent subgraphs without explicitly altering the graph. To do this we identify portions of the graph separable by a single vertex, more formally we find the graph's bi-connected components (see Figure 4). The complexity of solving the CSP is $O(d^k)$, where k is the size of the largest biconnected component [3].

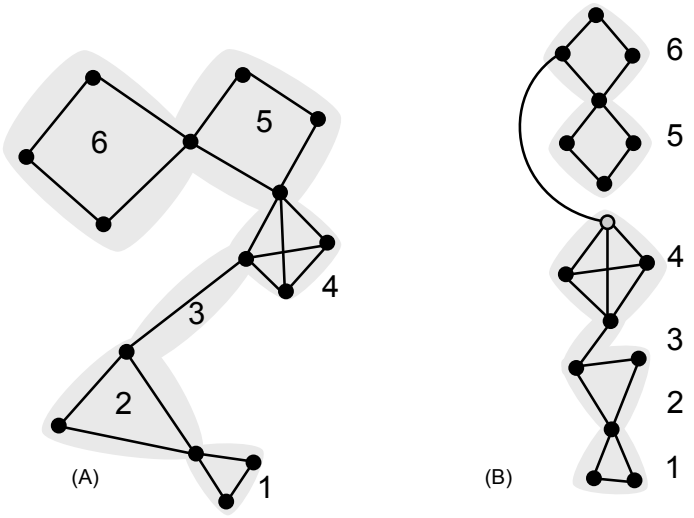

Figure 4: (A) A graph with identified bi-connected components, (B) Tree structure organization of bi-connected components.

3.4 Tree Clustering

For a non-binary CSP tree-clustering operates on the primal graph. The basic process follows these steps:

1. Apply MinFill to the primal graph to triangulate it.
2. Find maximal cliques of the primal graph...
3. and make a tree structure over them.
4. Find all the solutions of every clique, then
5. Do DAC on the solutions so that you can build a BT-free solution.

Complexity of tree clustering is $O(nd^k)$, where d is the domain size, k is the size of the largest max clique.

3.5 Bucket Elimination

Bucket elimination propagates constraints up through the network in a way similar to how tree clustering projects across the separator [1].

1. Choose a variable ordering.
2. Create a bucket for each variable.
3. Place each constraint in a bucket: the deepest one of any of its variables.
4. Starting from the bottom of the ordering, join the relations in the bucket, project out the bucket's variable from the join, and place the projection in the deepest bucket of any of its variables.

5. From the top down, find solutions backtrack free.

3.6 Tree Decomposition

This method is the most abstract view of trees in the CSP formalism. Tree decomposition defines a tree embedding of the constraint network of a CSP. Because it is so high level there are many ways a tree-decomposition scheme may be realized.

Definition 3 Tree Decomposition *Given a CSP $P = \langle X, D, C \rangle$, a tree $T = \langle V, E \rangle$ with the two labeling functions χ and ψ is a tree embedding of P (see Figure 5):*

- *$\chi(v)$: Associates to each $v \in V$ some subset of X.*
- *$\psi(v)$: Associates ti each $v \in V$ some subset of C.*

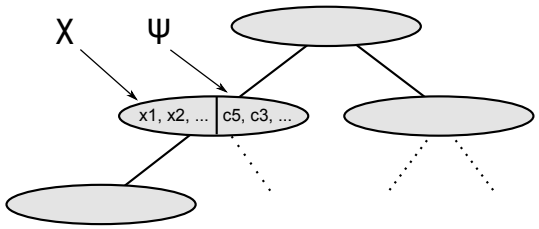

Figure 5: χ and ψ associate to every tree vertex a subset of the variables and constraints, respectively.

This embedding is a tree decomposition if the functions χ and ψ obey the following two rules:

1. *Every constraint appears with its full scope in some tree node.*
2. *Every appearance of a variable $x \in X$ complies with the connectedness property, which is illustrated in Figure 6.*

The decomposition methods based on bi-connected components, tree clustering, and bucket elimination are special cases of a tree decomposition. Other tree decomposition techniques exist [9, 11, 7, 8].

References

[1] Rina Dechter. Bucket Elimination: A Unifying Framework for Reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[2] Rina Dechter and Judea Pearl. The Cycle-Cutset Method for improving Search Performance in AI Applications. In *Third IEEE Conference on AI Applications*, pages 224–230, Orlando, FL, 1987.

[3] Eugene C. Freuder. A Sufficient Condition for Backtrack-Free Search. *JACM*, 29 (1):24–32, 1982.

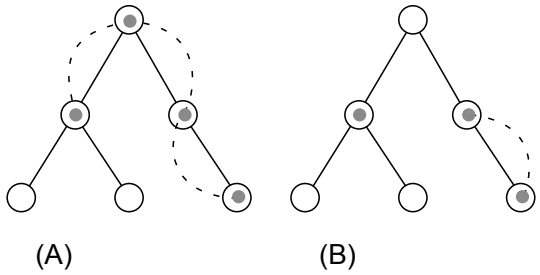

Figure 6: If a variable appears in two nodes of the tree, it must appear in every node along the path between the two nodes, inducing a single subtree. Here (A) displays the connectedness property while (B) does not.

[4] Joel Gompert. IndSet: A Decomposition Technique for CSPs Using Maximal Independent Sets and Its Integration with Local Search. Master's thesis, Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, May 2005.

[5] Joel M. Gompert. Local Search With Maximal Independent Sets. In Mark Wallace, editor, *Proceedings of* 10^{th} *International Conference on Principle and Practice of Constraint Programming (CP 04)*, volume 3258 of *LNCS*, page 795, Toronto, Canada, 2004.

[6] Joel M. Gompert and Berthe Y. Choueiry. A Decomposition Technique for CSPs Using Maximal Independent Sets and Its Integration with Local Search. In *International FLAIRS conference (FLAIRS'05)*, pages 167–174, 2005.

[7] Georg Gottlob, Nicola Leone, and Francesco Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124(2):243–282, 2000.

[8] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree Decompositions and Tractable Queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

[9] Marc Gyssens, Peter G. Jeavons, and David A. Cohen. Decomposing Constraint Satisfaction Problems Using Database Techniques. *Artificial Intelligence*, 66:57–89, 1994.

[10] David Maier. *The Theory of Relational Databases*. Pitman Publishing Limited, 1983.

[11] Yaling Zheng and Berthe Y. Choueiry. New Structural Decomposition Techniques for Constraint Satisfaction Problems. In Boi Faltings et al., editor, *Recent Advances in Constraints*, volume 3419 of *Lecture Notes in Artificial Intelligence*, pages 113–127. Springer, 2005.