TREE-CLUSTERING SCHEMES FOR CONSTRAINT-PROCESSING *

Rina Dechter
Judea Pearl

Cognitive System Laboratory, Computer Science Department
University of California, Los Angeles, CA 90024
Net address: dechter@cs.ucla.edu
Net address: judea@cs.ucla.edu

ABSTRACT

The paper offers a systematic way of regrouping constraints into hierarchical structures capable of supporting information retrieval without backtracking. The method involves the formation and preprocessing of an acyclic database that permits a large variety of queries and local perturbations to be processed swiftly, either by sequential backtrack-free procedures, or by distributed constraint-propagation processes.

1. Introduction

Solving Constraint-Satisfaction Problems (CSP) usually involves two phases: a preprocessing phase that establishes local consistencies, followed by a backtracking procedure that actually produces the solution desired. While the preprocessing phase is normally accomplished by local, constraint-propagation mechanisms, the answer-producing phase occasionally runs into difficulties due to excessive backtrackings. If a given set of constraints is to be maintained over a long stream of queries, it may be advisable to invest more effort and memory space in restructuring the problem so as to facilitate more efficient answer-producing routines. This paper proposes such a restructuring technique, based on clique-tree clustering. The technique guarantees that a large variety of queries could be answered swiftly either by sequential backtrack-free procedures, or by distributed constraint propagation methods.

The technique proposed exploits the fact that the tractability of CSPs is intimately connected to the topological structure of their underlying constraint graphs [Freuder, 1982, Dechter, 1987a]. The simplest result in this regard asserts that if the constraint-graph is a tree then the corresponding CSP can be solved efficiently, in $O(nk^2)$ steps, where n is the number of variables and k is the number of values. Another important feature of tree topology lies in facilitating unsupervised, constraint-propagation mechanisms. Distributed relaxation algorithms applied to constraint trees reach equilibrium in time proportional to the tree's diameter and, more significantly, the local consistencies established by such algorithms also guarantee a global consistency, namely, any value in the resultant graph is guaranteed to participate in a solution.

A general strategy of utilizing these merits of tree topologies in non-tree CSPs is to form clusters of variables such that the interactions between the clusters is tree-structured, then solve the problem by efficient tree algorithms. This amounts to first, deciding which variables should be grouped together, finding the internally consistent values in each cluster and, finally, processing these sets of values as singleton variables in a tree.

In this paper we present a general and systematic method of accomplishing this strategy, applicable for both binary and non-binary CSPs. The method is based on a combination of the theory of **acyclic databases** [Beeri, 1983], Freuder's conditions for backtrack-free search [Freuder, 1982] and the notion of directional consistency [Dechter, 1987a]. Related methods were also used for structuring statistical databases [Malvestuto, 1987], Bayesian inferences [Lauritzen, 1988], and the analysis of belief functions [Shafer, 1988].

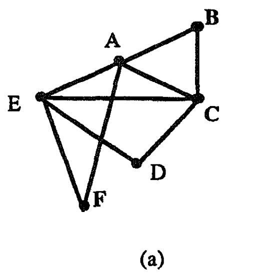
2. CSPs and their graph-representations

A constraint satisfaction problem involves a set of n variables $X_1,...,X_n$, each represented by its domain values, $R_1, \dots, R_n$ and a set of constraints. A constraint $C_i(X_{i_1}, \cdots, X_{i_j})$ is a subset of the Cartesian product $R_{i_1} \times \cdots \times R_{i_j}$ which specifies which values of the variables are compatible with each other. A solution is an assignment of values to all the variables which satisfy all the constraints and the task is to find one or all solutions. A **Binary CSP** is one in which all the constraints involve only pairs of variables. A binary CSP can be associated with a **constraint-graph** in which nodes represent variables and arcs connects pairs of constrained variables. Graph representations for high-order constraints can be constructed in two ways, **Primal-constraint-graph** and **Dual-constraint-graph**. A **Primal-constraint-graph** represents variables by nodes and associates an arc with any two nodes residing in the same constraint. A **Dual-constraint-graph** represents each constraint by a node (called a **c-variable**) and associates a labeled arc with any two nodes that share

* This work was supported in part by the National Science Foundation, Grant #DCR 85-01234 and by the Airforce Office of Scientific Research Grant #AFOSR-88-0177.

variables. The arcs are labeled by the shared variables.

For example, Figure 1a and 1b depict the primal and dual constraint-graph respectively, of a CSP with variables A, B, C, D, E, F and constraints on the subsets (ABC),(AEF), (CDE) and (ACE) (the constraints themselves are not explicitly given).



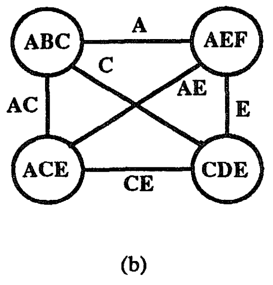


Figure 1: A primal and dual constraint graphs of a CSP.

Since trees are desirable structures we want to transform any constraint-graph into a tree. One way of doing it is to form larger clusters of c-variables, another is to identify and remove **redundant arcs**. A constraint is considered redundant if its elimination from the problem does not change the set of solutions. Since all constraints in the dual-graph are equalities, an arc can be deleted if its variables are shared by every arc along an alternative path between the two end points. The subgraph resulting from the removal of redundant arcs is called a **join graph**, and it has the following property: for each two nodes that share a variable there is at least one path of labeled arcs, each containing the shared variable. For example, in figure 1b, the arc between (AEF) and (ABC) can be eliminated because the variable A is common along the cycle (AEF) -- A -- (ABC) -- AC -- (ACE) -- AE -- (AEF) and, so, a consistent assignment to A is ensured by the remaining arcs. By a similar argument we can remove the arcs labeled C and E, thus turning the join-graph into a tree, called **join-tree**.

A CSP organized as a join-tree can be solved efficiently. If there are p constraints in the join-tree, each with at most l subtuples, then, a straight application of the algorithm developed for a tree of singletons (i.e., $O(nk^2)$) would yield a solution in $O(pl^2)$. However, by ordering the tuples of each constraint lexicographically, the task of matching two tuples can be reduced to $O(\log l)$ steps, and so arc-consistency between two constraints, (which is $O(k^2)$ for binary constraints), can be enforced in $O(l \log l)$ steps, thus reducing the overall complexity to $O(p \cdot l \cdot \log l)$. The set of CSPs that possess a join-tree is called **acyclic-databases** (called **Acyclic-CSPs** here), and their desirable properties are discussed at length in [Beeri, 1983].

3. The Tree-Clustering Scheme

Our aim is to transform any CSP into an acyclic representation, even when the dual constraint graph of the original representation of the problem cannot be reduced to a join-tree. We do it by systematically forming larger clusters than those given in the dual constraint graphs.

A CSP is acyclic iff its primal graph is both chordal and conformal [Beeri, 1983]. A graph G is **chordal** if every cycle of length at least four has a chord, i.e., an edge joining two nonconsecutive vertices along the cycle. A graph is **conformal** if each of its maximal cliques corresponds to a constraint in the original CSP.

The clustering scheme is based on an efficient triangulation algorithm [Tarjan, 1984] which transforms any graph into a chordal graph by adding edges to it. It consists of two steps: 1. Compute an ordering for the nodes, using a **maximum cardinality** search. 2. Fill-in edges between any two non-adjacent nodes that are connected via nodes higher up in the ordering. The maximal cliques of the resulting chordal graph are the clusters necessary for forming an acyclic CSP.

The **maximum-cardinality-search** numbers vertices from 1 to n, in increasing order, always assigning the next number to the vertex having the largest set of previously numbered neighbors, (breaking ties arbitrarily). Such ordering will be called **m-ordering**. If no edges are added in step two, the original graph is chordal, otherwise the new filled graph is chordal.

The above theory suggests the following clustering procedure for CSPs:

1. Given a CSP and its primal graph, use the triangulation algorithm to generate a chordal primal graph.

2. Identify all the maximal cliques in the primal-chordal graph. Let $C_1,...,C_t$ be all such cliques indexed by the rank of its highest nodes.

3. Form the dual-graph corresponding to the new clusters and identify one of its join-trees by connecting each C_i to an ancestor C_j $(j < i)$ with whom it shares the largest set of variables [Maier, 1983].

4. Solve the subproblems defined by the clusters $C_1, \ldots, C_t$, (this amounts to generating and listing the consistent subtuples for each cluster).

5. Solve the tree problem with treating the clusters as singleton variables: a. perform directional arc-consistency (DAC) on the join-tree [Dechter, 1987a]. b. solve the join-tree in a backtrack-free manner.

For example, consider a CSP defined by the constraint-sets: (AC), (AD), (BD), (CE), (DE). The primal graph is given in figure 2a.

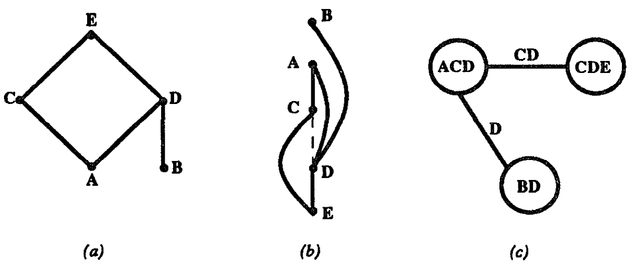


Figure 2.

The ordering $d = E, D, C, A, B$ is one possible m-ordering (Figure 2b). The fill-in required by this ordering adds the arc (C, D) and results in the chordal graph of figure 2b. The maximal cliques associated with this graph are: (ADC), (DCE), and (DB). A join-tree of these constraints is shown in figure 2c. The three subproblems associated with the sets of variables (ADC), (DCE) and (DB), are solved and then, using these local solutions as domains for the c-variables, the tree is solved in the usual manner (step 5).

The first three steps of the algorithm, i.e., triangulation and fill-in, cluster-identification and join-tree generation, all manipulate the topology of the primal graph and all are bounded by $O(n^2)$, the size of the resultant chordal graph. Cliques identification is easy since in the filled-in cordal graph, any vertex V and its parent set $C(V)$ form a clique, and thus all maximal cliques (there are at most n) can be determined in decreasing order of V, discarding a newly generated clique that is contained in a previous clique. Determining the join-tree, is linear in the size of the triangulated primal graph and can be performed greedily (see step 3).

The fourth step which requires solving the subproblems defined by each clique may dominate the overall computation since it takes $O(k^r)$ when k is the number of values and r is the size of the maximal clique. Finally, the last step of solving the join-tree is $O(n \cdot t \log t)$ when t is the maximum number of solutions in each clique. Considering all the above steps the overall complexity of the clustering scheme is roughly bounded by $O(k^r)$. The space complexity is also $O(k^r)$ since the solution set explicated for each clique at step 4 may be exponential in the size of the clique. For more details see [Dechter, 1987b]

We will see next that some computation can be saved in steps 4 and 5, by executing the clustering steps in a coordinated way, by consulting the solutions found in one clique for pruning the set of solutions assembled in adjacent cliques.

4. Adaptive-consistency

Studies on the level of local consistency required to guarantee that solutions can be retrieved in a "backtrack-free" manner, show [Freuder, 1982, Dechter, 1987a] that an ordered constraint-graph is backtrack-free if the level of directional strong-consistency along this order is greater then the width of the ordered graph. We show how this theory leads to a clustering scheme similar to that of section 3.

The **width of a node** in an ordered graph is the number of links connecting it to nodes lower in the ordering. The **width of an ordering** is the maximum width of nodes in that ordering, and the **width of a graph** is the minimal width of all its orderings. A CSP is **i-consistent** if for any consistent value-assignment for $i-1$ variables, there exists a value for any i^{th} variable, such that the i values together are consistent. d-i-consistency requires only that the $i-1$ values can be consistently extended by any variable that succeed all instantiated variables in the ordering d. **Strong-i-consistency** holds when the problem is **j-consistent** for $j \le i$. **strong-d-i-consistency** can be defined accordingly.

If the width of the graph is $i-1$ but the problem is not i-consistent, algorithms enforcing i-consistency can be applied to it, e.g., the algorithms known as **arc-consistency** and **path-consistency** enforce 2-consistency and 3-consistency respectively [Montanari, 1974, Mackworth, 1984, Dechter, 1987a]. However, since i-consistency may add arcs to the graph and thus change its width, there is a need to adapt the level of consistency imposed during this process in order to guarantee backtrack-free search. The following procedure, we first presented in [Dechter, 1987a] takes this issue into consideration. A similar algorithm, suggested by Seidel [Seidel, 1981] accomplished, essentially, the same idea.

Given an ordering, d, we establish d-i-consistency recursively, letting i change dynamically from node to node to match its width at the time of processing. Nodes are processed in decreasing order, so that by the time a node is processed, its final width is determined and the required level of consistency can be achieved. For each variable, X, let PARENTS(X) be the set of all variables connected to it and preceding it in the graph.

```
Adaptive-consistency($X_1, \ldots, X_n$)
 Begin
  1. for i=n to 1 by -1 do
  2. Compute PARENTS($X_i$)
  3. connect all elements in PARENTS($X_i$)
     (if not yet connected)
  4. perform consistency($X_i$, PARENTS($X_i$))
  5. find solution using backtrack($X_1, \ldots, X_n$)
 End
```

The procedure **consistency**(V,SET) generates and records those tuples of variables in SET that can be consistent with at least one value of V. The procedure may impose new constraints over clusters of variables as well as tighten existing constraints. The topology of the **induced graph** can be found prior to executing the procedure, by recursively connecting any two parents sharing a common successor.

Consider our example of figure 2 in an ordering (E,D,C,A,B) shown in figure 3a. Adaptive-Consistency proceeds from B to E and imposes consistency constraints on the parents of each processed variable. B is chosen first and the algorithm enforces a 2-consistency on D (namely an arc-consistency on (D,B)), since the width of B is 1. A is selected next and, having width 2, the algorithm enforces a 3-consistency on its parents $\{C,D\}$. This operation may require that a constraint between C and D be added, and in that case an arc (C,D) is added. When the algorithm reaches node C its width is 2 and, therefore, a 3-consistency is enforced on C's parents $\{E,D\}$. The arc (E,D) already exists so this operation may merely tighten the corresponding constraint. The resulting graph is given in Figure 3b.

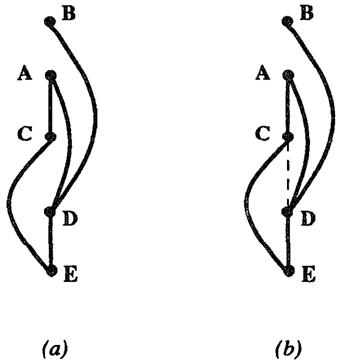


Figure 3.

Let $W(d)$ be the width of the ordering d and $W^*(d)$ the width of the induced graph. The complexity of solving a problem using Adaptive-Consistency preprocessing phase (steps 1-4) and then backtracking (freely) along the order d (step 5) is dominated by the former. The worst-case complexity of the "consistency(V, PARENT(V)) step" is exponential in the cardinality of variable V and its parents. Since the maximal size of the parent-sets is equal to the width of the induced graph we see that solving the CSP along the ordering d is $O(\exp(W^*(d)+1))$.

5. Relationships between Adaptive-Consistency ($A-C$) and Tree-Clustering ($T-C$)

The two schemes presented, although unrelated at first glance, share many interesting features. First, for any given ordering d, the set of fill-in arcs added by triangulation, is equal to the set of arcs added by Adaptive-Consistency scheme. Both methods recursively connect sets of nodes that share a common successor in the ordering, (see figures 2b and 3b). In particular, $A-C$'s induced graph is always chordal and, if the original graph is chordal and ordered by a max-cardinality search, its width will not change (no arcs are added in this case).

In addition, a strong structural resemblance exists between the clusters chosen by $T-C$ and the constraints (new or old) recorded by $A-C$. In each clique C of size r (in the induced graph) $A-C$ will record or tighten one constraint of size $r-1$. Namely, every cluster in $T-C$ (i.e., a maximal clique) is represented in $A-C$ by the constraints originally contained in that cluster, and at most one additional constraint for each size less then the cluster's cardinality. See in figure 4a and 4b the clusters generated by $T-C$ and the constraints recorded by $A-C$.

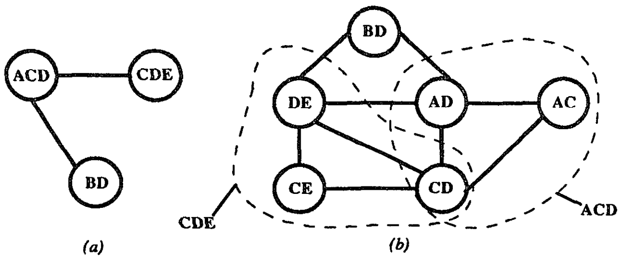


Figure 4. (a) clusters of T-C, (b) constraints of A-C.

Rough asymptotic bounds on the time and space-complexity of both schemes reveal that they are about the same. If $W^*(d)$ is the width of the induced graph, then $W^*(d)+1$ is the size of the largest clique and, therefore, both $A-C$ and T-C are space-bounded and time-bounded by $O(k^{W^*(d)})$, k being the number of values. These bounds can be further tightened to yield $O(exp(W^*+1))$ where $W^* = \min_d \{W^*(d)\}$. However, computing an optimal d was shown to be an NP-complete task [Arnborg, 1987], and among the various heuristic orderings studied in the literature [Bertele, 1972], the most popular are the minimal width and the m- orderings. The ease of finding these orderings enables us to calculate $W^*(d)$ under both orderings, and take the lowest value as a better upper bound estimate of W^*. Moreover, any minimum-width ordering, denoted d_{mw}, can be used for generating both a lower and an upper bound for W^* since $W(d_{mw}) \leq W^* \leq W^*(d_{mw})$.

In practice we may find cases favoring either one of the two schemes space-wise, because the explicit representation of $T-C$ may sometimes be more economical. Regarding actual time complexity we argue that A-C outperforms T-C, and in effect can be considered a more efficient approach to tree-clustering. The reason is that clusters are not assembled independently, but are pruned during construction. Algorithm $A-C$ constructs, in effect, a join-tree that is already directional-arc-consistent and, so, renders step 5a of $T-C$

unnecessary. The only difference is that $A-C$ does not explicitly enumerate the domains of the c-variables but, instead, represents them as local conjunctions of lower-arity constraints (see figure 4). This enumeration can be accomplished by step 5 of A-C using backtrack. In that case the resulting (implicit) join-tree would be fully arc-consistent. For more details see [Dechter, 1987b].

The question arises whether there is ever a need to fully explicate the domain of each clique in the join-tree. Obviously, if the ultimate task is merely finding one (or all) solution to the given CSP, then the representation constructed by the $A-C$ (steps 1-4) is sufficient. However, not all applications are suitable for a solution process committed to a fixed ordering. For example, to answer the query: "Is there a solution in which variable X_j attains the value x?" it is convenient to begin the search at X_j rather then at some other variable. In general, if the ultimate task is to maintain an effective database for answering a variety of queries, a balanced, undirectional representation is preferred, facilitating information retrieval in all orderings.

6. Conclusions

Tree-Clustering offers a systematic way of regrouping elements into hierarchical structures capable of supporting information retrieval without backtracking. The basic Tree-Clustering scheme involves triangulating the constraint graph, identifying the maximal cliques of the triangular graph, solving the constraints associated with each clique and organizing the solutions obtained in a tree structure. A routine called Adaptive Consistency has been identified as an effective method of assembling the desired tree.

Once the clusters are formed and their join-tree established and processed, the resulting structure offers an effective database, to be amortized over many problem instances. A large variety of queries could be answered swiftly either by sequential backtrack-free procedures, or by distributed constraint propagation processes. In addition, when local new constraints (which do not alter the structure of the tree) are added, global consistency can still be maintained by unsupervised constraint-propagation processes.

The tree-clustering scheme can facilitate efficient computation of many functions which are easily solvable on a tree of binary constraints. Such application is shown for belief propagation in Bayesian-networks [Lauritzen, 1988], for belief-functions in Dempster-Shafer formalism [Shafer, 1988], and for constraint-optimization [Dechter, 1988].

Future experimental work is required to compare Tree-clustering schemes and backtrack algorithms in order to determine whether the advantages of these schemes, as manifested by their worse-case bounds, are translated into an actual improvement in performance.

References

[Arnborg 1987] Arnborg, S., D.G. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *Siam Journal of Algorithm and Discrete Math.*, Vol. 8, No. 3, 1987, pp. 277-284.

[Beeri 1983] Beeri, C., R. Fagin, D. Maier, and Nihalis Yanakakis, "On the desirability of acyclic database schemes, " *JACM*, Vol. 30, No. 3, 1983, pp. 479-513.

[Bertele 1972] Bertele, U. and F. Brioschi, *Nonserial dynamic programming*, New York: Academic press, 1972.

[Dechter 1987a] Dechter, R. and J. Pearl, "Network-based heuristics for constraint-satisfaction problems," *Artificial Intelligence Journal*, Vol. 34, No. 1, 1987, pp. 1-38.

[Dechter 1987b] Dechter, R. and J. Pearl, "Tree-clustering for constraint-networks," Technical Report #R-92, UCLA Cognitive Systems Laboratory, Los Angeles, CA., 1987. (*Artificial Intelligence*, forthcoming).

[Dechter 1988] Dechter, R., A. Dechter, and J. Pearl, "Optimization in constraint-networks." In *Proceedings*, Conference on Influence Diagrams for Decision Analysis, Inference, and Prediction, June 9-11, Berkeley, CA., 1988.

[Freuder 1982] Freuder, E.C., "A sufficient condition of backtrack-free search," *Journal of the ACM*, Vol. 29, No. 1, 1982, pp. 24-32.

Shafer, G. and P.P. Shenoy, "Bayesian and belief-function propagation," School of Business working paper No. 192, University of Kansas, Lawrence, Kansas, April 1988.

[Lauritzen 1988] Lauritzen, S.L. and D.J. Spiegelhalter, "Local computations with probabilities on graphical structures and their applications to expert systems." To appear in *J.R. Statist. Soc. B.* Vol. 50, 1988.

[Maier 1983] Maier, D., *The theory of relational databases*, Rockville, Maryland:Computer Science Press, 1983.

[Malvestuto 1987] Malvestuto, F.M., "Answering queries in categorical databases." In *Proceedings*, Sixth conference on the Principals of Database Systems, San Diego, CA., 1987, pp. 87-96.

[Montanari 1974] Montanari, U., "Networks of constraints, fundamental properties and applications to picture processing," *Information Science*, Vol. 7, 1974, 95-132.

[Seidel 1981] Seidel, R., "A new method for solving constraint-satisfaction problems." In *Proceedings*, IJCAI, 1981, pp. 338-342.

[Tarjan 1984] Tarjan, R.E. and M. Yannakakis, "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs," *SIAM Journal of Computing*, Vol. 13, No. 3, 1984, pp. 566-579.