TREE CLUSTERING SCHEMES FOR CONSTRAINT PROCESSING

Rina Dechter
Judea Pearl

September 1987
CSD-870054

TECHNICAL REPORT

R-92

August 1987

TREE-CLUSTERING SCHEMES FOR CONSTRAINT-PROCESSING *

Rina Dechter

Artificial Intelligence Center
Hughes Aircraft Company, Calabasas, CA 91302
and
Cognitive System Laboratory, Computer Science Department
University of California, Los Angeles, CA 90024

Judea Pearl

Cognitive Systems Laboratory
Computer Science Department
University of California, Los Angeles, Ca 90024

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 to be amortized and maintained over many problem instances. Once found, the database permits a large variety of queries and local perturbations to be process swiftly, either by sequential backtrack-free procedures, or by distributed constraint-propagation processes.

* This work was supported in part by the National Science Foundtation, Grant #DCR 85-01234

TREE-CLUSTERING SCHEMES FOR CONSTRAINT-PROCESSING

Rina Dechter & Judea Pearl

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, Freuder, 1985, Dechter, 1985]. 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. This property is also applicable to processing CSPs of arbitrary topologies; tree-structured simplifications can be used as heuristics to guide choices in backtracking [Dechter, 1987a], and tree-solving algorithms can be invoked when subproblems are recognized to be tree-

structured [Dechter, 1987b].

Another important feature of tree topology lies in facilitating unsupervised, constraint-propagation mechanisms. Parallel 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. This means that any value combination chosen from the variables in the final network constitutes a global solution to the CSP.

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 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, 1985]. Related methods were also used for structuring statistical data bases [Malvestuto, 1987], Bayesian inferences [Lauritzen, 1987], and the analysis of belief functions [Tung, 1986].

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, $R_1, \ldots, R_n$ 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** (called "Intersection-graph" in database theory) [Maier, 1983] represents each constraint by a node (called a **c-variable**) and associates a labeled arc with any two nodes that share 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).

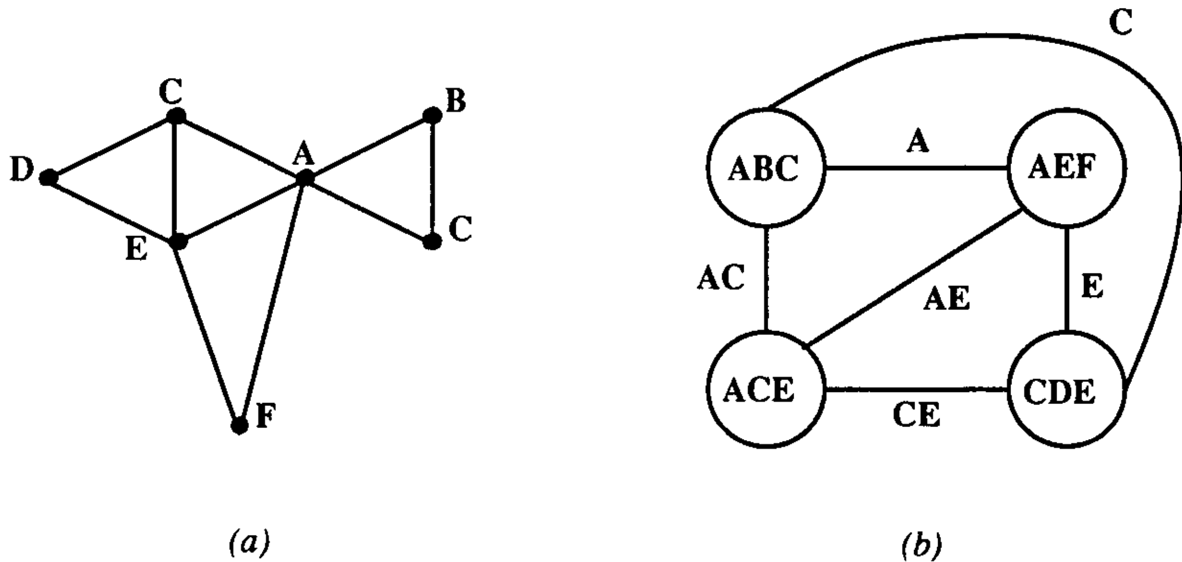

Figure 1: A primal and dual constraint graphs of a CSP

The dual-constraint-graph transforms any non-binary CSP to a special type of binary CSP: The domain of the c-variables ranges over all possible value-combinations permitted by the corresponding constraints, and any two adjacent c-variables must obey the restriction that their shared variables should have the same values, (i.e., the c-variables are bounded by "equality" constraints). Using this representation and exploiting the structure of the dual-constraint-graph, we can solve a non-binary CSP using methods developed for binary CSPs. In particular, if the dual constraint-graph is a tree, the problem can be solved in linear time using the tree-algorithm described in [Dechter, 1985].

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. A join-graph is an equivalent representation to the original dual-graph though it may contain fewer arcs.

For example, in figure 1a, 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 that has a join-tree can be solved efficiently. If there are p constraints in the join-tree, each with at most l subtuples, the CSP can be solved in $O(pl^2)$. The set of CSPs that possess a join-tree is called **acyclic-databases** (called **Acyclic-CSPs** here), and their desirable properties (only part of which are mentioned here) are discussed at length in [Beeri, 1983]. Efficient procedures for identifying a join tree of an acyclic database are described in [Maier, 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 reduce 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 primal graph is **conformal** if each of its maximal clique corresponds to a constraint in the original CSP.

The clustering scheme described in this paper is based on an efficient triangulation algorithm [Tarjan, 1984] which transforms any graph into a chordal graph by adding edges to it. The maximal cliques of the resulting chordal graph are the clusters necessary for forming an acyclic CSP.

The triangulation algorithm 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 **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. Tarjan et. al. give a maximum cardinality search algorithm that can be implemented in $O(n+deg)$ where n is the number of variables and deg is the maximum degree. The fill-in step of the algorithm runs in $O(n+m')$ when m' is the number of arcs in

* the order here is the reverse of that used in Tarjan et. al. and was changed to simplify the presentation. Such ordering will be called **m-ordering**.

the resultant graph. There is no guarantee that the number of edges added by this process is minimal, however, since for chordal graphs the m-ordering requires no fill-in, the fill-in required for non-chordal graphs, is usually small.

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 (if the primal graph is chordal no arc will be added).

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 $(j < i)$ that contains all variables that C_i shares with its ancestors [Maier, 1983].

4. Solve the subproblems defined by the clusters $C_1, \ldots, C_t$, (this amounts to generating higher-order constraints from the lower-order constraints internal to each cluster, i.e., listing the consistent subtuples for the variables in each cluster).

5. Solve the tree problem with the clusters serving as variables.

   a. perform directional arc-consistency (DAC) on the join-tree [Dechter, 1985].

   b. solve the join-tree in a backtrack-free manner.

For example, consider a CSP on variables $\{A, B, C, D, E\}$, defined by the constraints: (A, C),(A, D),(B, D), (C, E), (D, E). The primal graph is given in figure 2a.

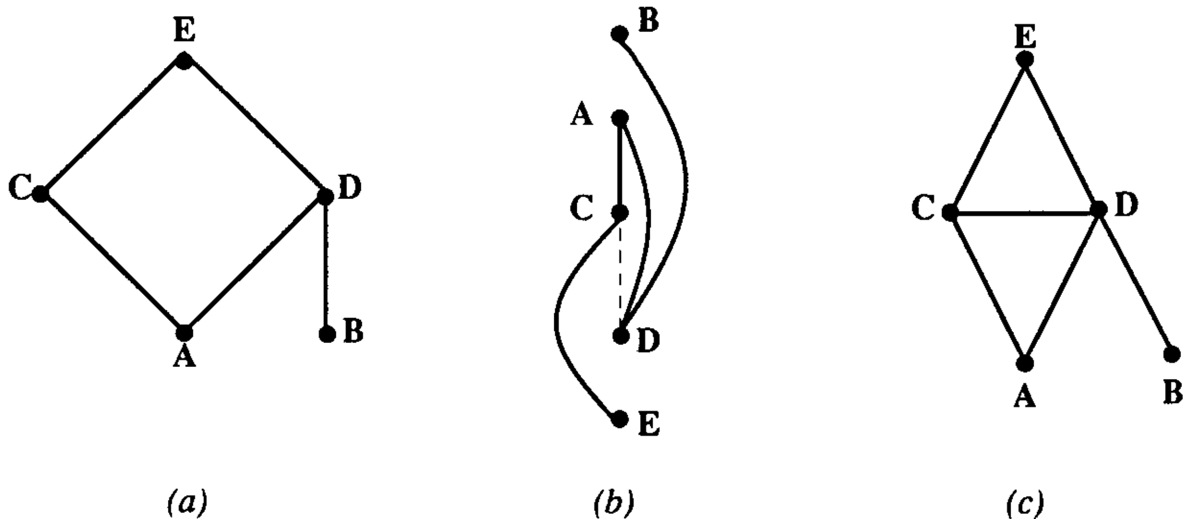

Figure 2

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 2c. The maximal cliques associated with this graph are: $(A, D, C,)$, (D, C, E), and (D, B) (see figure 3c). The dual graph associated with these constraints and one of its associated join-trees are shown in figure 3a, and 3b respectively. To solve the problem shown in figure 3b, we first solve the three subproblems associated with the sets of variables (A, D, C), (D, C, E) and (D, B), then, using these local solutions as domains for the c-variables, the tree is solved in the usual manner. For example, solving subproblem (A, D, C) means finding all assignments to A, D, C which are consistent with the input constraints (A, C) and (A, D).

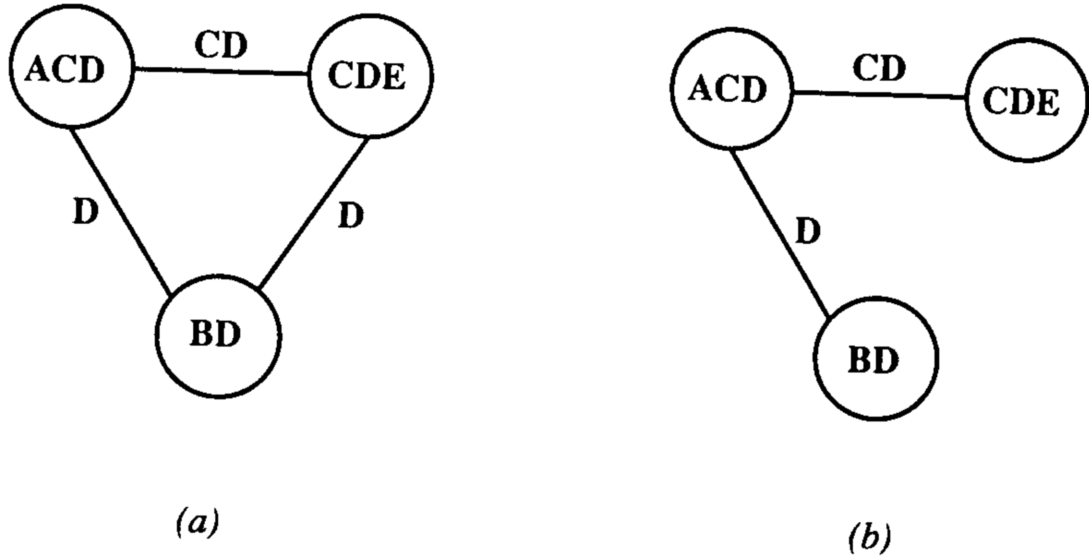

Figure 3

We can estimate the running time of the algorithm as follows. Given a CSP having n variables, its primal graph may be of $O(n^2)$ of the original problem size. Triangulating the primal graph is also bounded by $O(n^2)$ since both the fill-in and the maximal cardinality search are bounded by the size of the resultant graph. For the second step (i.e. identifying all maximal cliques) observe that in the filled-in graph, any vertex V and its parent set $C(V)$ (those which are connected to it and precede it in the m-ordering) form a clique. The reason being that any two parent vertices which were not connected in the original graph will be "filled" by the fill-in step of the chordality algorithm. Therefore, to enumerate all maximal cliques we can determine the cliques $C(V)$ in decreasing order of V, discarding newly generated clique that is contained in a previous clique. In this process each arc will be tested once to determine adjacency and, therefore, the complexity of this step is $O(|E'|)$ when E' is the set of edges in the filled graph. Notice that the maximum number of cliques is n.

The third step, determining the join-tree, is linear in the size of the triangulated primal graph. Considering the maximal cliques in the reversed order dictated by the m-ordering, each will be connected to one parent clique that precedes it and which shares a largest set of variables with it. The fourth step requires solving the subproblems defined by each clique. If r is the size of the largest clique and k is the number of values for each variable, this step is bounded by $O(k^r)$ and may dominate the overall computation. Finally, the last step of solving the join-tree is $O(n \cdot t^2)$ when t is the maximum number of solutions in each clique. This step is performed by executing directional arc-consistency from leaves to root [Dechter, 1985] (step 5a) , and then finding a solution in a backtrack-free manner (step 5b). Summing over all steps, the overall complexity is bounded by:

$$O(n^2) + O(k^r) + O(t^2)$$

Since $t \le k^r$, the total computation is bound by $O(k^{2r})$ while the space complexity is bounded by $O(k^r)$.

The question is whether some computation can be saved in steps 4 and 5, by executing the clustering steps in a coordinated way. For example, it appears wasteful to independently solve two adjacent cliques, only to find out later that many of the solutions found are incompatible with each other. A more economical way would be to consult the solutions found in one clique for pruning the set of solutions assembled in adjacent cliques. Such possibilities are offered by enforcing local **consistency** as shown in the next subsection.

4. Adaptive-consistency

Freuder [Freuder, 1982] has studied the level of local consistency required to guarantee that solutions can be retrieved in a "backtrack-free" manner. We show how this theory, coupled with the notion of directional consistency [Dechter, 1985], 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 set of $i-1$ variables along with value for each that satisfy all the constraints among them, there exists a value for any i^{th} variable, such that the i values together satisfy all the constraints among the i variables. **Strong-i-consistency** holds when the problem is **j-consistent** for $j \leq i$. Given an ordering d, **directional-i-consistency** (d-i-consistency for short) requires only that any consistent instantiation of $i-1$ variables can be consistently extended by any variable that succeed all of them in the ordering d. **strong-d-i-consistency** can be defined accordingly. The following theorem summarizes the conditions for backtrack-free search:

Theorem:([Freuder, 1982, Dechter, 1987a])

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.

□

When a problem is not i-consistent, algorithms enforcing i-consistency can be applied to it [Freuder, 1978], e.g., the algorithms known as **arc-consistency** and **path-consistency** enforce 2-consistency and 3-consistency respectively [Montanari, 1974, Mackworth, 1984, Dechter, 1985, Mohr, 1986]. It may seem that the above theorem can be used as follows. Given a CSP, find the width of its graph (Freuder presents a linear-time algorithm for finding the width, W, of a graph), perform a (W+1)-consistency algorithm, then solve the problem in a backtrack-free manner. Unfortunately, achieving i-consistency (i>2) often requires the addition of constraints which amounts to adding arcs to the constraint-graph and increasing its width, thus violating the conditions for backtrack-free search. The following procedure, originally presented in [Dechter, 1987a] is a modification to the above idea that takes this issue into consideration. (1)

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 recursively 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. The parents of each variable are computed only when they need to be processed.

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 they are not yet connected)
4. perform consistency(X_i, PARENTS(X_i))

(1) We recently learned that a similar procedure was proposed by [Seidel, 1981]

5. find a solution using backtrack on the ordering $(X_1, \ldots, X_n)$
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. Note that the procedure can generate constraints that contain other constraints. When the procedure terminates backtrack can solve the problem, in the order prescribed without encountering any dead end. The topology of the **induced graph** (identical to the one generated by directional-path-consistency) 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 4a. The adaptive-consistency algorithm 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. 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 merely results in tightening the corresponding constraint. The resulting graph is given in Figure 4b, and its dual constraint graph consisting of all recorded constraints, is shown in figure 4c.

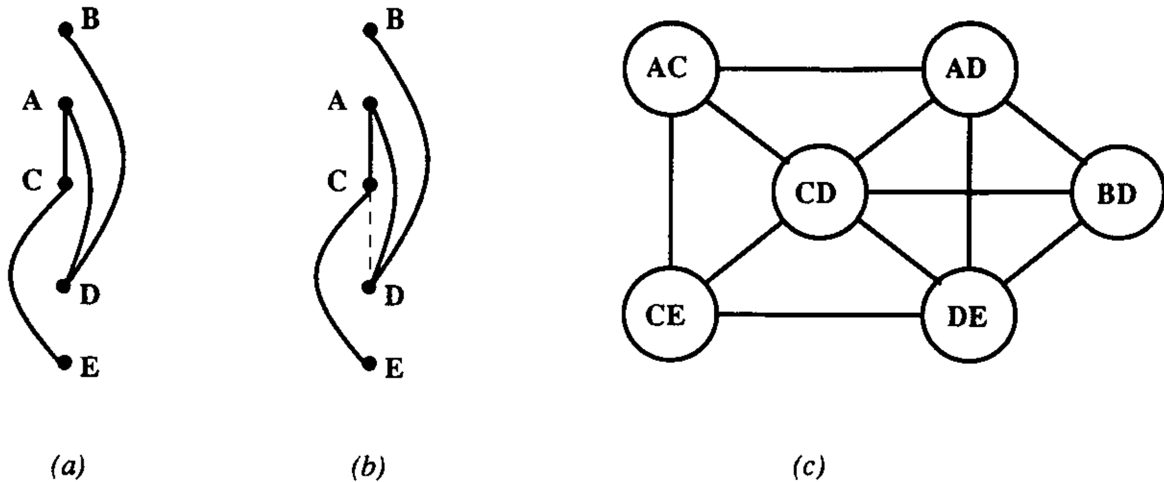

Figure 4

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 the 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 exp (W*(d)+1). The complexity bound can be further tightened to yield 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 good estimate of W^*.

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 triangulization, 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, so the two will induce the same final graph if initiated on the same ordered graph (see, for examples figures 2b and 4b). In particular, the 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).

Rough bounds on the **space-complexity** of both schemes reveals that they are about the same. If $W*$ is the width of the induced graph, then $W*+1$ is the size of the largest clique and, therefore, $A-C$ is space-bounded by $O(k^{W*})$ while $T-C$ is space bounded by $O(k^{W*+1})$, k being the number of values. In practice, however, we may find cases favoring either one of the two schemes, because the explicit representation of $T-C$ may sometimes be more economical.

In addition to the topological identity of the graphs induced by the two schemes, a strong structural resemblance exists between the clusters chosen by $T-C$ and the constraints (new or old) recorded by $A-C$. In each maximal clique C of size r (in the induced graph) $A-C$ will record or tighten at least one constraint of size $r-1$. If C contains another clique C' of size r' then this, too, is associated, with $A-C$ recording one constraint

of size $r'-1$. Namely, every cluster (i.e., a maximal clique) is represented in $A-C$ by the constraints originally contained in that cluster (some of which may be tightened), and at most one additional constraint for each size less then the cluster's cardinality.

In figure 5a and 5b, we present once again the clusters generated by $T-C$ and the constraints recorded by $A-C$. The $A-C$ scheme characterizes cluster (ACD) by three binary constraints (AD), (AC) and (CD), the latter is a newly recorded constraint. Cluster (CDE) is characterized by (CE), (DE) and (CD), while cluster (BD) corresponds to the original constraint. Thus, $A-C$ can be viewed as an assembler that efficiently constructs a join-tree of clusters, and represents them, somewhat implicitly, in a decomposed way. The reason for its greater efficiency ($O(k^{W^*})$, compared with $O(k^{2W^*})$ for $T-C$) lies, indeed, in the fact that clusters are not assembled independently, but are pruned during construction. This is illustrated in the following example.

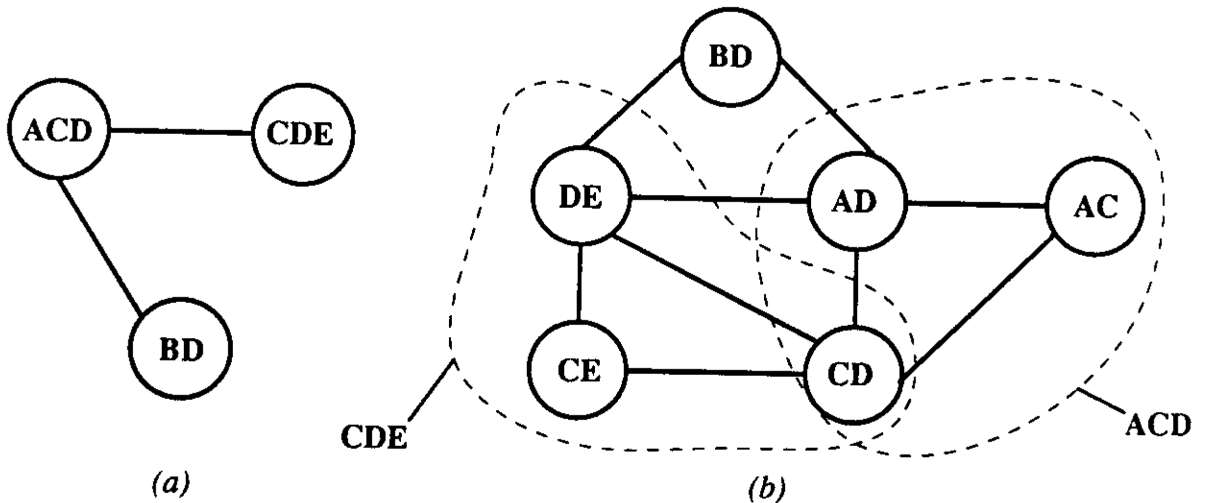

Figure 5

Consider the binary CSP represented by the constraint graph of figure 6a; the graph is chordal and doesn't change by either scheme. Assume the m-ordering of figure 6b and the join tree of figure 6c.

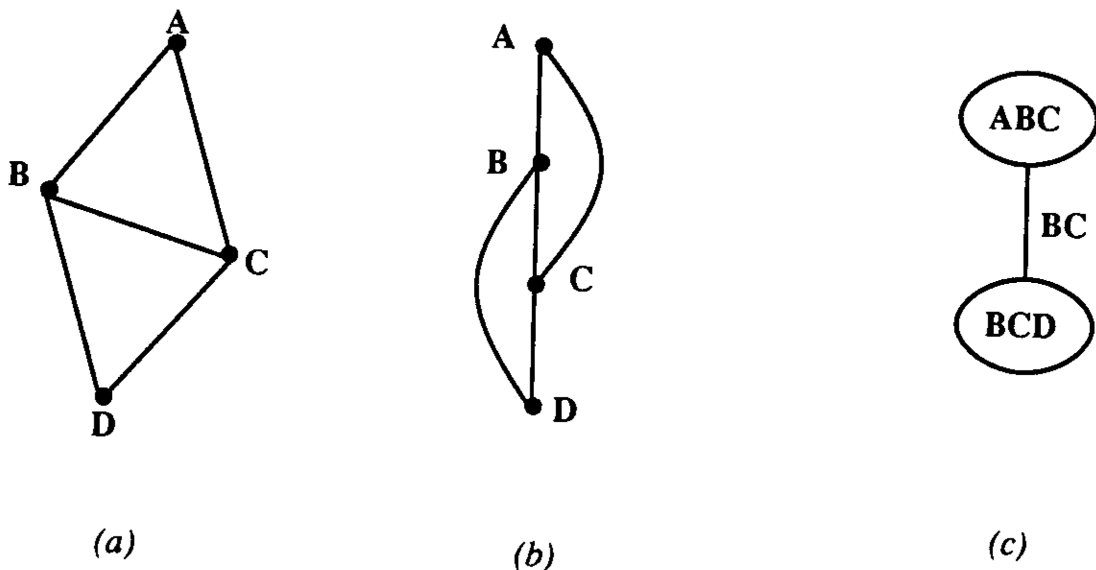

Figure 6

When Tree-Clustering solves the problem, the two subproblems (ABC) and (BCD) must first be solved **independently**. (ABC) may be solved by computing the constraint that A induces on (BC), then listing all consistent triplets. The same can be done for (BCD) but, since the two subproblems are solved independently, the solution found for (BCD) conforms only to the original constraint on (BC) not the one tightened by solving subproblem (ABC). Thus, several triplets in (BCD) would be generated and listed redundantly, only to be discarded, once the solutions of the two subproblems interact via the directional arc-consistency of step 5a.

Adaptive-Consistency essentially eliminates this redundancy. Proceeding along the order D,C,B,A, variable A tightens the constraint on (B,C), then B tightens the constraint on (C,D) and, finally, C induces a unary constraint on D. The problem can now be solved in a backtrack-free manner along the original ordering. Thus, the clusters assembled by $A\text{-}C$ are constructed and solved incrementally, in an order prescribed by the join-tree; the construction of each new cluster involves only value combinations consistent with previously established clusters.

Moreover, $A\text{-}C$ constructs, in effect, a compact version of the join-tree produced by $T\text{-}C$ that is already directional-arc-consistent and, so, renders step 4 of $T\text{-}C$ unnecessary. The only difference between the join-tree produced by $A\text{-}C$ and that resulting from step 4 of $T\text{-}C$ is that $A\text{-}C$ does not explicitly enumerates the domains of the c-variables but, instead, represents them as local conjunctions of lower-arity constraints. One of these constraints (corresponding to the largest fill-in recorded) has arity one-below the size of the clique.

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 $A\text{-}C$ algorithm is sufficient; solutions can be produced without backtracking in the ordering prescribed by $A\text{-}C$. 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. In particular, if the join-tree is fully arc-consistent, then conjunctive queries of the form: "What values of X_{n+1} are compatible with $X_1 = x_1, \cdots, X_n = x_n$?" can be answered in a single pass by initiating parallel constraint-propagation processes simultaneously, from the cliques containing $X_1, \cdots, X_n$. To enjoy this feature, it is useful to further process the join-tree so as to establish full, undirectional arc-consistency, and this requires explicating the domains of the cliques.

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 triangularizing the constraint graph, identifying the maximal cliques of the triangularized 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; it sidesteps most of the aforementioned steps and also guarantees that the resulting join-tree is directional-arc-consistent. Further processing of the join-tree, to establish full arc-consistency, is also advantagous for certain applications.

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 such as the

Waltz algorithm. In addition, when local new facts are added, global consistency can still be maintained by unsupervised constraint-propagation processes.

References

[Arnborg, 1987.] S. Arnborg, D. G. Corneil, and A. Proskurowski, ''Complexity of finding embeddings in a k-tree,'' *Siam Journal of algorithm and Discrete Math.,* Vol. 8, No. 2, 1987., pp. 277-184.

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

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

[Dechter, 1985] R. Dechter and J. Pearl, ''The anatomy of easy problems: a constraint-satisfaction formulation,'' in *Proceedings Ninth International Conference on Artificial Intelligence,* Los Angeles, Cal: 1985, pp. 1066-1072.

[Dechter, 1987a] R. Dechter and J. Pearl, ''Network-based heuristics for constraint-satisfaction problems.,'' UCLA, L.A. Cal., 1987. To be published at the AI-Journal.

[Dechter, 1987b] R. Dechter and J. Pearl, ''The cycle-cutset method for improving search performance in AI applications,'' in *Proceeding of the 3rd IEEE on AI Applications,* Orlando, Florida: 1987.

[Freuder, 1978] E.C. Freuder, ''Synthesizing constraint expression,'' *Communication of the ACM,* Vol. 21, No. 11, 1978, pp. 958-965.

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

[Freuder, 1985] E.C. Freuder, ''A sufficient condition for backtrack-bounded search,'' *Journal of the Association of Computing Machinery,* Vol. 32, No. 4, 1985, pp. 755-761.

[Lauritzen, 1987] S.L. Lauritzen and D.J. Spiegelhalter, ''Fast Manipulation of Probabilities with Local Representations - With Applictions to Expert Systems,'' Aalborg Univ. Institute of Electronic Systems, Aalborg, Denmark, Tech. Rep. R-87-7, 1987.

[Mackworth, 1984] A.K. Mackworth and E.C. Freuder, "The complexity of some polynomial network consistancy algorithms for constraint satisfaction problems," *Artificial Intelligence* , Vol. 25, No. 1, 1984.

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

[Malvestuto, 1987] F.M. Malvestuto, "Answering queries in categrical databases," in *Proceedings Sixth conference on the Principles of Database Systems,* San-Diego, Cal.: 1987, pp. 87-96.

[Mohr, 1986] R. Mohr and T.C. Henderson, "Arc and Path consistency revisited," *Artificial Intelligence,* Vol. 28, No. 2, 1986, pp. 225-233.

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

[Seidel, 1981] R. Seidel, "A New Method of Solving Constraint Satisfaction Problems," in *Proceedings IJCAI,* 1981, pp. 338-342.

[Tarjan, 1984] R. Tarjan and M. Yannakakis, "Simple Linear-Time Alsorithms 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.

[Tung, 1986] C. Tung and A. Kong, "Multivariate Belief Functions and Graphical Models," Harvard University, Cambridge, MA, Tech. Rep. S-107, 1986.