DECOMPOSING AN N-ARY RELATION INTO A TREE OF BINARY RELATIONS

Rina Dechter

February 1987
CSD-870011

TECHNICAL REPORT
R-76
CSD-8700##
January 1987

DECOMPOSING AN N-ARY RELATION INTO A TREE OF BINARY RELATIONS* †

Rina Dechter
Cognitive Systems Laboratory
UCLA Computer Science Department
Los Angeles, California 90024

*This work was supported in part by the National Science Foundation, Grant #DCR 85-01234;
† accepted for publication in *Proceedings* of the 6th ACM SIGACT-SIGMOD-DIGART Symposium, March 1987.

DECOMPOSING AN N-ARY RELATION INTO A TREE OF BINARY RELATIONS

Rina Dechter

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

ABSTRACT

We present an efficient algorithm for decomposing an n-ary relation into a tree of binary relations, and provide an efficient test for checking whether or not the tree formed represents the relation. If there exists a tree-decomposition, the algorithm is guaranteed to find one, otherwise, the tree generated will fail the test, then indicating that no tree decomposition exist. The unique features of the algorithm presented in this paper, is that it does not apriori assume any dependencies in the initial relation, rather it derives such dependencies from the bare relation instance.

1. Introduction

The primary use of functional dependencies and multivalued dependencies in relational databases is to guide the decomposition of a relation scheme into a database scheme consisting of smaller relations that satisfy the join-dependency property, i.e., the reconstruction of the whole relation from its components by the natural join operation is lossless [6]. The goal in decomposing a relation is to save storage by avoiding redundancy. Query processing, on the other hand, which may sometimes require the reconstruction of the whole relation, becomes more expensive as the result of decomposition.

Some of the shortcomings of decomposition are avoided if it is decomposed into a **tree of binary relations**. Such decomposition will improve storage representation since binary relations require considerably less space then an n-ary relation. At the same time it also posses desired properties for query processing. The fact that the relations do not contain cycles guarantees that when the natural-join operations are performed in an order prescribed by the tree-structure, the size of the intermediate relations will grow monotonically. The same goal is achieved by constructing join-trees for given acyclic data-bases [6].

In this paper we present a greedy algorithm for lossless decomposing a relation into a tree of binary relations provided that such decomposition exists. We also provide a simple test for determining whether a lossless tree decomposition exists. Furthermore, we show that if the relation is not **tree-decomposable** then the tree of binary relations generated by the algorithm is the best approximation of the given relation in a certain sense (to be expounded later). This work was motivated by a method proposed in [1] for approximating discrete probability distribution with a **dependence tree**.

The results reported here have their origins in area of solving Constraint Satisfaction Problems (CSPs), which have many applications in Artificial Intelligence [5]. Constraint satisfaction problems involve the assignments of values to variables subject to a set of constraints, where each constraint is an i-ary relation on a subset of i variables ($i \leq n$), and the task is to find one or all consistent solutions. The task of finding all consistent assignments is equivalent to that of reconstructing the whole relation from a given lossless decomposition. Thus, a CSP is a data-base instance, variables in a CSP correspond to attributes in a database and the constraints of the CSP correspond to the relations in the database.

The general CSP is NP-Complete since some NP-complete problems (e.g., the graph coloring problem) fall into this class. However, various subclasses can be solved efficiently and their solutions were found to be useful in solving more general problems [4, 2].

One way for characterizing easy CSPs is by identifying dependencies and independencies in the relation. If we associate an hypergraph representation with a given database scheme, where the nodes of the graph are the attributes and the hyperarcs are subsets of attributes which appear in the same relation, the graph structure explicitly represents independencies that can be used by algorithms that process the constraints. A special graph representation arises for binary CSPs, i.e., where all the constraints are binary. In that case the hypergraph is a regular graph (called a constraint graph), and each arc corresponds to a binary constraint.

The best known and most useful result in this area is that binary CSPs whose constraint graph is a tree can be optimally solved in time bounded by $O(nk^2)$, where n is the number of variables and k is the number of values for each variable [2]. Therefore, having a tree-representation of a binary-CSP is valuable. Consider, for example, the relation on **FLIGHT, DAY-OF-WEEK** and **PLANE-TYPE** presented in figure 1. This relation can be decomposed losslessly into the pairs (FLIGHT, DAY-OF-WEEK) and (FLIGHT PLANE-TYPE). The associated graph is given in figure 2.

In general CSPs the n-ary relation is, of course, not available explicitly. However, having a procedure for deriving a tree decomposition from the whole relation can benefit those applications where the CSP is being used several times, e.g., databases and Truth-maintenance systems [3].

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

FLIGHT	DAY-OF-WEEK	PLANE-TYPE
106	Monday	747
106	Thursday	747
106	Monday	1011
106	Thursday	1011
204	Wednesday	707
204	Wednesday	727

Figure 1: the relation (FLIGHT, DAY-OF-WEEK,PLANE-TYPE)

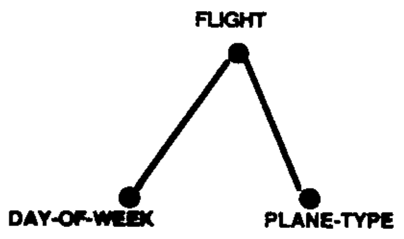


Figure 2: the graph associated with (FLIGHT,DAY-OF-WEEK),(FLIGHT,PLANE-TYPE)

2. Definitions and Preliminaries

Let ρ denote an n-ary relation over the set of attribute U = $\{X_1, \ldots, X_n\}$, which is a subset of the cartesian product $Dom(X_1) \times, \ldots, \times Dom(X_n)$ when Dom(X_i) is the set of values of attribute X_i. ρ_S denotes the projection of ρ on a subset S of attributes. The assignment of specific values to some subsets of attributes is called **instantiation.** An instantiation corresponds to a **restriction** of ρ to only those n-tuples that "match" the instantiation. The property of a relation that enables a lossless decomposition into two smaller relations is what we call **conditional independence** which is closely related to the notion of **Multi-valued-dependencies (MVDs).** Two subsets of attributes S_1 and S_2 are said to be **independent in** ρ if $\rho_{S_1S_2} = \rho_{S_1} \times \rho_{S_2}$ where S_1S_2 denotes the union of S_1 and S_2. If X_i and X_j are independent, then instantiating one of them to any legal value does not make any legal value of the other illegal.

Definition: S_1 and S_2 are **conditionally independent** given S_3, denoted by $<S_1 \mid S_3 \mid S_2>$, if they are independent in the restrictions corresponding to all possible instantiations of S_3.

The conditional independence $<S_1 \mid S_3 \mid S_2>$ means that knowing any particular instantiation of S_3 makes the instantiation of S_1 irrelevant for S_2 (and vice versa). Conditional independence parallels the notion of embedded MVDs [6]. That is, $<S_1 \mid S_3 \mid S_2>$ iff $S_3 \rightarrow\rightarrow S_2$ in the projection of ρ on $S_1S_2S_3$. If in a relation ρ, $<S_3 \mid S_1 \mid S_2>$, when $S_3 = U - S_1S_2$ then ρ can be decomposed losslessly into the database scheme S_1S_2 and S_1S_3. For instance, in the relation of figure 1, the attribute DAY-OF-WEEK is conditionally independent of PLANE-TYPE give FLIGHT.

The constraint-graph associated with a particular decomposition explicitly represents some of the conditional independencies embodied in the relation. Every **separation** in the graph corresponds to a conditional independence, i.e., if a subset S_1 separates (in the graph) subset S_2 from S_3 then $<S_2 \mid S_1 \mid S_3>$. The term **separation** will therefore, be also used to denote conditional independence. For a detailed discussion of conditional-independencies and their graphical representation see [7].

The following notations will be used throughout.
$n(x_i) \triangleq$ number of n-tuples in ρ for which $X_i = x_i$
$n(x_i, x_j) \triangleq$ number of n-tuples in ρ for which $X_i = x_i$ and $X_j = x_j$.
and in general, $n(x_{i1}, \ldots, x_{it}) \triangleq$ number of n-tuple in ρ for which $X_{i1} = x_{i1}, \ldots, X_{it} = x_{it}$
Let S be any subset of U. $n_S(x_i) \triangleq$ number of $|S|$-tuples in the projection ρ_S for which $X_i = x_i$
and in general, $n_S(x_{i1}, \ldots, x_{it}) \triangleq$ number of $|S|$-tuple in ρ_S for which $X_{i1} = x_{i1}, \ldots, X_{it} = x_{it}$

3. Tree-decomposition

The following is a demonstration of the algorithm which will be developed in the sequel. Consider the relation over the binary attributes $\{X,Y,Z,T,F\}$, given in figure 3.

X	Z	Y	T	F
0	0	1	1	1
0	0	1	1	0
0	1	1	1	1
0	1	1	1	0
0	1	1	0	1
0	1	0	1	1
0	1	0	1	0
0	1	0	0	1

Figure 3:

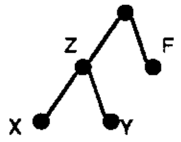
The first step is to compute the n-quantities $\{n(x_i)\}$ and $\{n(x_i,x_j)\}$ for all attributes and their values. Obtaining:

$n(X{=}0) = 8$, $n(Z{=}1){=}6$, $n(Z{=}0){=}2$,
$n(Z{=}0,Y{=}1){=}2$, $n(Z{=}1,Y{=}1){=}3$ etc.

Next, for each pair of attribute (X_i,X_j) we compute the weights $m(X_i,X_j)$ according to the formula given in equation (3) that follows:

$m(X,Z) = m(X,Y) = m(X,T) = m(X,F) = -16.63.$
$m(Z,Y) = -13.97$, $m(Z,T) = -15.95$, $m(Z,F) = -16.55$,
$m(Y,T) = -16.55$, $m(Y,F) = -17.13$, $m(T,F) = -15.50.$

Finally, using the maximum-weight spanning-tree algorithm on these weighted arcs, the tree shown in figure 4 is produced. This tree, and its associated data-base (see figure 4) is a lossless decomposition of the relation.



T	Z
1	0
1	1
0	1

T	F
1	1
1	0
0	1

X	Z
0	0
0	1
1	0

Z	Y
0	1
1	1

Figure 4

The following paragraphs contain the justification to this algorithm.

Let T be a directed tree on $X_1 \cdots X_n$, T can be represented by (parent, son) pairs of directed arcs i.e., $T = \{(X_i X_{j(i)})\}$, where $X_{j(i)}$ is the parent of X_i in T. X_0 denotes the root of the tree. With each tree T we associate a mapping (denoted also by T) $T: \rho \to R^+$ where R^+ are real numbers, defined by:

$$\forall \bar{x} \in \rho \qquad \bar{x} = x_1, \ldots, x_n$$

$$T(\bar{x}) = n(x_0) \cdot \prod_{i=1}^{n-1} \frac{n(x_i,\ x_{j(i)})}{n(x_{j(i)})} \tag{1}$$

For each tree T we define a measure $F_\rho(T)$

$$F_\rho(T) = \sum_{\bar{x} \in \rho} \log T(\bar{x})$$

$$= \sum_{\bar{x}} \log n(x_0) \cdot \prod_{(x_i x_{j(i)}) \in T} \frac{n(x_i\, x_{j(i)})}{n(x_{j(i)})}$$

$$F_\rho(T) = \sum_{\bar{x} \in \rho} \left[\sum_{\substack{i=1 \\ (x_i x_{j(i)}) \in T}}^{n-1} \log \frac{n(x_i x_{j(i)})}{n(x_{j(i)})} + \log n(x_0) \right]$$

$$= \sum_{\bar{x} \in \rho} \left[\sum_{\substack{i=1 \\ (x_i,\, x_{j(i)}) \in T}}^{n-1} \log \frac{n(x_i, x_{j(i)})}{n(x_i) n(x_{j(i)})} + \sum_{i=0}^{n-1} \log n(x_i) \right]$$

$$= \sum_{i=1}^{n-1} \sum_{\bar{x} \in \rho} \log \frac{n(x_i, x_{j(i)})}{n(x_i) n(x_{j(i)})} + |\rho| \sum_{i=0}^{n-1} \log n(x_i)$$

We get that:

$$F_\rho(T) = \sum_{i=1}^{n-1} \sum_{(x_i x_{j(i)}) \in T} n(x_i x_{j(i)}) \log \frac{n(x_i,\, x_{j(i)})}{n(x_i) n(x_{j(i)})} \tag{2}$$

$$+ |\rho| \sum_{i=0}^{n-1} \log n(x_i)$$

and therefore,

$$F_\rho(T) = \sum_{\substack{i=1 \\ (X_i,\, X_{j(i)}) \in T}}^{n-1} m(X_i, X_{j(i)}) + |\rho| \sum_{i=0}^{n-1} \log n(x_i)$$

where

$$m(X_i X_{j(i)}) = \sum_{(x_i x_{j(i)}) \in \rho_{X_i X_j}} n(x_i x_{j(i)}) \log \frac{n(x_i x_{j(i)})}{n(x_i) n(x_{j(i)})} \tag{3}$$

Since the second element in the sum is independent of the tree structure, $F_\rho(T)$ can be maximize by the MST algorithm when m is the weight of the arcs.

We will show next that if ρ is representable by a tree and if F gets its maximum value in T_0 then T_0 is a tree representing ρ. Otherwise the relation represented by the data-base T_0, is the "closest" to ρ, in some sense.

Theorem 1: If ρ has a tree representation then any T_0 which maximizes F provides such a representation, when each arc of T_0 is associated with the projection of ρ on the pair of attributes connected by the arc.

The validity of theorem 1 follows from:

Theorem 2: T is a tree representing relation ρ_T if and only if $\forall \bar{x} \in \rho_T$

$$n(x_0) \prod_{(i,j(i)) \in T} \frac{n(x_i x_{j_{(i)}})}{n(x_{j_{(i)}})} = 1 \tag{4}$$

To show that theorem 1 follows from (4) we reason as follows: Since F is a concave and symmetric function on $V = \{(T(\bar{x}_1), \ldots, T(\bar{x}_l)) \mid T \in TREES\}$ and it is bounded by a symmetric constraint $\sum_{\bar{x}} T(\bar{x}) \le l$ [1] where l is the size of the relation ρ_T, F's extremum (maximum) is achieved when $T(\bar{x})$ are all equal. Moreover, since F is monotone w.r.t. each of its components, if there exist T s.t $T(\bar{x}) = 1$ $\forall \bar{x}$, F will get its maximum in this T.

If ρ is not tree-decomposable, property (4) is not satisfied and therefore the point (1,1,...,1) is not in the domain, V, of the optimization function. The optimal T, in that case, will not constitute a lossless representation. However, the tree found will still represent a relation "closest" to ρ in the proximity measure defined by F. In that sense, the tree associated with this maximum can be regarded as the best approximation of ρ. We cannot conclude, however, that the relation represented by the maximum-weight-tree is also the closest to ρ in terms of the **number of extra tuples** it contains, although we believe that there is a strong correlation between the two measures.

Although, the proof could be derived from probability theory using Chaw's result and a mapping between relations and probability, we preferred to derive it in terms of relations and thus to have the advantage of additional insight into the properties of tree-decomposable relations.

Proof of theorem 2:

If an attribute, X_i, separates two subsets of attributes, S_1 and S_2, each containing X_i, then, in the joined relation each value of X_i that appear in ρ_{S_1} will be duplicated as many time as it appears in ρ_{S_2}. The following lemma states this property.

Lemma 1: Let S_1, S_2 and S_3 be subsets of U, s.t $U = S_1 S_2 S_3$ then S_3 separates S_1 from S_2 iff

$$\forall x \in \rho_{S_3} \quad n_\rho(x) = n_{S_3 S_1}(x) \cdot n_{S_3 S_2}(x)$$

□

A similar product form holds for a tuple in ρ:

Lemma 2: If X_j separates $S_1 = X_1 \cdots X_{j-1}$ from $S_2 = X_{j+1} \cdots X_n$ then $\forall \bar{x} \in \rho$

$$n(x_j) = n(x_1 \cdots x_{j-1} x_j) \cdot n(x_j x_{j+1} \cdots x_n) \tag{5}$$

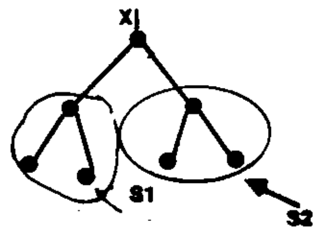


Figure 5: $<S_1 | X_j | S_2>$

Proof:

Let $S'_i = X_j S_i$, $i = 1,2$. Since $<X_1 \cdots X_{j-1} | X_j | X_{j+1} \cdots X_n>$ then from lemma 1

$$\forall x_j \in Dom(X_j) \quad n(x_j) = n_{S'_1}(x_j) \cdot n_{S'_2}(x_j) \qquad (6)$$

however

a. $n_{S'_1}(x_j) = n(x_j x_{j+1} \cdots x_n)$

b. $n_{S'_2}(x_j) = n(x_1 \cdots x_j)$.

(a) is true since $(x_j x_{j+1} \cdots x_n) \in \rho_{S'_2}$ will appear in ρ as many times as $X_j = x_j$ appears in $\rho_{S'_1}$ i.e., as many as $n_{S'_1}(x_j)$. The same argument works for (b). Substituting (a) and (b) in (6) yields (5).

□

Lemma 3: Let ρ be a relation on $X_1 \cdots X_n$ represented by tree T. Let S be a subtree of T rooted at X_i with $X_{j(i)}$ as its parent nodes (see figure 6) then

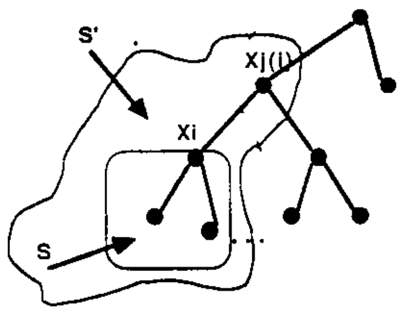


Figure 6

$$\forall \bar{x} = (x_{i_1}, \ldots, x_{i_l}) \in \rho_{S'}$$

$$n(x_{i_1}, x_{i_2}, \ldots, x_{i_l}) = \frac{n(x_i x_{j(i)}) \cdot n(\bar{x} - x_{j(i)})}{n(x_i)}$$

where S' is the union of S with $X_{j(i)}$, and $\bar{X} - x_{j(i)}$ is the tuple in $\rho_{S'}$ without its $j(i)$ element.

Proof:

Let S'' be the complement of S i.e. $S'' = U - S$, $\bar{x}$ be a member of $\rho_{S'}$ and $\bar{x} - x_{j(i)}$ is the tuple in $\rho_{S'}$ without its $j(i)$ element.

From lemma 2 when applied to the relation $\rho_{S'}$, and from the fact that X_i separates $X_{j(i)}$ from S, we get:

$$1 = \frac{n_{S'}(x_{j(i)} x_i) \cdot n_{S'}(\bar{x} - x_{j(i)})}{n_{S'}(x_i)} \qquad (7)$$

From the tree-structure we can also infer the following:

$$n(x_i) = n_{S'}(x_i) \cdot K \qquad (8)$$

where K is defined by

$$K = \sum_{x'_{j(i)} \text{ compatible with } x_i} n_{S''}(x'_{j(i)})$$

K denotes the number of times all values of $X_{j(i)}$ which are legal with the value x_i of X_i appears in the complement relation of S.

also

$$n(x_i, x_{j(i)}) = n_{S'}(x_i, x_{j(i)}) \cdot n_{S''}(x_{j(i)}) \qquad (9)$$

and

$$n(\bar{x} - x_{j(i)}) = n_{S'}(\bar{x} - x_{j(i)}) \cdot K \qquad (10)$$

Substituting (8), (9) and (10) in (7) we get:

$$n_{S''}(x_{j(i)}) = \frac{n(x_i, x_{j(i)}) \cdot n(\bar{x} - x_{j(i)})}{n(x_i)} \qquad (11)$$

From the tree-structure we can also infer that

$$n_{S''}(x_{j(i)}) = n(\bar{x}) \qquad (12)$$

which yields the desired result

□

Lemma 4: if S is a subtree of T rooted at X_i and if X_i separates S_1, S_2 (see figure 7)

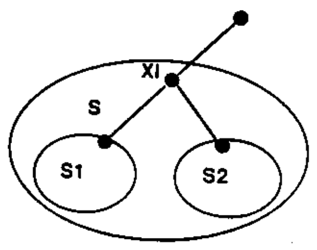


Figure 7

then $\forall \bar{x} \in S$

$$n(\bar{x}) = \frac{n(\bar{x} | S_1 X_i) \cdot n(\bar{x} | S_2 X_i)}{n(x_i)} \qquad (13)$$

where $\bar{x} | S$ denotes the restriction of a tuple $\bar{x}$ to attributes appearing in S.

Proof: (exactly as the proof of lemma 3)

From lemma 2:

$$\forall \bar{x} \in S \quad 1 = \frac{n_S(\bar{x} | S_1 X_i) \cdot n_S(\bar{x} | S_2 X_i)}{n_S(x_i)} \qquad (14)$$

Let $S' = (U - S)X_i$, from the tree-structure we get:

$$n(x_i) = n_{S'}(x_i) \cdot n_S(x_i) \tag{15}$$

also for both S_1 and S_2 it holds that:

$$n(\bar{x} \mid S_i X_i) = n_S(\bar{x} \mid S_i X_i) \cdot n_{S'}(x_i) \tag{16}$$

Since also

$$n(\bar{x}) = n_{S'}(x_i) \tag{17}$$

we get that

$$n(\bar{x}) = \frac{n(\bar{x} \mid S_1 X_i) \cdot n(\bar{x} \mid S_2 X_i)}{n(x_i)} \tag{18}$$

□

Theorem 2 can be proved by recursively applying lemmas 2,3, and 4. Given an n-tuple $x \in \rho$, then for x_0, the root value we first apply lemma 2 to yield:

$$1 = n(x_1 \cdots x_0 \cdots x_n) = \frac{n(x_1 \cdots x_0) \cdot n(x_0 \cdots x_n)}{n(x_0)} \tag{19}$$

where the indices in the first tuple and in the second tuple correspond to two subtrees separated by X_0. We continue to decompose each part of the nominator by applying lemma 3 and lemma 4 as necessary until we have only single and pairs of n-quantities. In this decomposition, each parent node appears $b-1$ times in the denominator when b is its degree in the tree. For all parents accept X_0, $b-1$ is also the number of children; we therefore get:

$$1 = n(x_0) \, \Pi \frac{n(x_i x_{j_{(i)}})}{n(x_{j_{(i)}})} \tag{20}$$

which completes the proof of theorem 2.

4. An algorithm for tree-decomposition

The tree-decomposition algorithm obtain as input an arbitrary relation ρ and returns a set of tree-structured binary relations.

Tree-generation-algorithm

a. Compute basic quantities: $n(x_i)$ and $n(x_i, x_j)$

b. For every two attributes X_i , X_j compute the weights $m(X_i, X_j)$ given in (3).

c. Find the **Maximum weight spanning tree** algorithm on the complete graph w.r.t. the above arc-weights.

d. For each two attributes that correspond to an arc in the selected tree find the associated relation by projecting the overall relation on them.

The complexity of the algorithm is $O((l + k)n^2)$ where n is the number of attributes, and k bounds the domain sizes, which can be shown by following its individual steps. The computation of part (a) can be completed in $O(ln^2)$ steps when l is the size of the relation. Part (b) is bounded by $O(n^2k^2)$ since $O(n^2)$ is the number of weights needed to be computed, and each computation can take $O(k^2)$ steps. Since part (c), i.e., the MST algorithm, takes just $O(n^2)$ steps, the total complexity is $O((l+k^2)n^2)$.

To verify that the generated tree represent the input relation, we can compute the number of n-tuples represented by the tree and compare it to the size of the given relation. If the two numbers are equal, the data-base losslessly represents the relation, otherwise, we know that no tree-representation exist. The size of a relation represented by the tree can be computed in linear time. For details see [2].

5. Conclusions

We have presented an efficient algorithm for decomposing an n-ary relation into a tree of binary relations, and have provided an efficient test for checking whether or not the tree formed represents the relation. If there exists a tree-decomposition, the algorithm is guaranteed to find one, otherwise, the tree generated will fail the test indicating that no tree decomposition exist. Moreover, the decomposition generated represents the best possible tree-approximation along the proximity measure defined by F. In that case, it may be advisable to regroup some attributes, and reiterate the process, until a lossless tree among the compound variables is achieved. The initial tree generated by the algorithm may serve as a good starting point for selecting candidates for regrouping.

The unique features of the algorithm presented in this paper, compared to other work on Multi-valued-dependencies, is that it does not assume any dependencies or decomposition in the initial relation, it actually derives such dependencies from the bare relation instance.

References

[1] Chow, C.K. and C.N. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transaction on Information Theory,* 1968, pp. 462-467.

[2] Dechter, R. 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.

[3] Doyle, Jon, "A truth maintenance system," *Artificial Intelligence,* Vol. 12, 1979, pp. 231-272.

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

[5] Mackworth, A.K., "Consistency in networks of relations," *Artificial intelligence,* Vol. 8, No. 1, 1977, pp. 99-118.

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

[7] Pearl, J. and A. Paz, "On the logic of representing dependencies by graphs," in *Proceedings AI-86, Canadian Conference,,* Montreal Canada: 1986.