Computer Science Department Technical Report
Cognitive Systems Laboratory
University of California
Los Angeles, CA 90024-1596

OPTIMIZATION IN CONSTRAINT NETWORKS

Rina Dechter
Avi Dechter
Judea Pearl

July 1988
CSD-880059

TECHNICAL REPORT
CSD-8700XX
R-111
April 7, 1988

OPTIMIZATION IN CONSTRAINT NETWORKS *

Rina Dechter
Avi Dechter
Judea Pearl

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

ABSTRACT

This paper deals with the task of finding an optimal solution to a set of variables constrained by a network of compatibility relationships. The paper shows that the optimization task does not require an exhaustive search among all consistent solutions but rather can be incorporated naturally into the process of finding consistent solutions. In many problem instances the interaction between the objective function and the constraints does not add any computational complexity to the task of finding a consistent solution, and when it does, the additional complexity can be estimated before the solution is attempted and be used to decide between an exact or a heuristic approach to optimization.

* This work was supported in part by the NSF Grant, IRI-851234, and by the U.S. Army Center for Signals Warfare under contract #DAAB10-86-C-0603.

1. Introduction

Traditionally, decision problems have been formulated as optimization tasks on probabilistic knowledge-bases such as statistical tables or influence diagrams. In several AI applications it is useful to express knowledge in terms of categorical constraints among facts and events rather then conditional probabilities. For example, resource limitations, temporal and spatial relationships, class hierarchies and concept definitions are stated as black-and-white constraints which allow a limited set of feasible solutions. This abstraction enjoys the advantage of descriptive simplicity because it requires fewer parameters; a model can be specified naturally and qualitatively, and the representation often leads to somewhat simpler inference procedures. For example such categorical abstraction, called compatibility relation, is the backbone of Shafer-Dempster formalism, which aims at processing evidence when a complete probability model is unavailable.

To handle inferences subject to qualitative constraints, a representation scheme called "constraint network" have been developed, which facilitates the control of solution strategies by the logical properties of the constraints. By recognizing substructures for which effective algorithms are available, the network representation enables one to determine how problems should be decomposed. Works on constraint processing has so far focused on the task of finding feasible solutions [Mackworth, 1984, Haralick, 1980, Dechter, 1987a], because in some applications of AI preferences play only a minor role and finding one feasible solution suffices.

However there are applications where preferences cannot be ignored. In vision we seek the most likely interpretation of a scene given visual clues and semantic constraints on objects and their relationships. In non-monotonic logic, we seek a minimal theory that is compatible with evidential sentences. In planning, one normally faces options leading to conflicting goals and seeks a strategy that maximizes the overall utility of consequences, subject to resource and feasibility constraints. It is, therefore, essential to develop algorithms that find an optimal solution, not merely a satisficing one.

This study extends the applications of constraint networks to include optimization task, especially distributed optimization. The paper treats the optimization of linearly decomposed criteria functions over a set of constrained variables. We provide an efficient algorithm that exploit the structure of each problem instance and give simple criteria for assessing its worst case complexity.

A **constraint network** involves a set of n variables, $X = \{X_1, \ldots, X_n\}$, and a set of constraints. A constraint $C_i(X_{i_1}, \cdots, X_{i_j})$ is a subset of the Cartesian product $R_{i_1} \times \cdots \times R_{i_j}$, where $X_{i_1}, \ldots, X_{i_j}$ is an arbitrary subset of variables and R_k represents the domain of variable X_k. The tuples of a constraint specify all the simultaneous assignments of values to the variables of the constraint which are, as far as this constraint is concerned, legal. An assignment of values to all the variables of the network such that all the constraints are satisfied is called a solution. The term **constraint satisfaction problem (CSP)** describes the computational task of finding either one solution or the set of all solutions of a given constraint network.

When a network possesses more than one solution, a natural extension of the constraint satisfaction problem is to identify the solution, or solutions, which are "best" according to some criterion. In this paper we provide a procedure for finding a consistent solution which maximizes the value of a criterion function of the form:

$$f(x) = \sum_{i \in T} f_i(x^i) \tag{1}$$

Where $T = \{1,2,...,t\}$ is a set of indices, designating subsets of variables $X^1, X^2, \ldots, X^i, \ldots, X^t$, x is an instantiation of all considered variables, while x^i is the instantiation x restricted to variables in a subset X^i of X. The functions $f_i(x^i)$ are the **components** of the criterion function, also called an objective function, and are specified, in general, by means of stored tables. We focus first on the special case when the domain of values are real numbers and the objective function is a simple utility function of the form:

$$U(x_1, \ldots, x_n) = \sum_{i=1}^{n} w_i x_i \tag{2}$$

The algorithm, to be presented in section 3, is an adaptation of non-serial dynamic programming methods [Bertele, 1972] to the case of categorical constraints, and relies on the theory of databases [Beeri, 1983]. It is guided by the network structure of each problem instance, and its complexity, for the case of utility function (2), does not exceed the complexity of finding an arbitrary solution.

Section 2 discusses the structural properties of constraint-networks and defines the class of acyclic CSPs. In section 3 we present an efficient scheme for finding a maximum utility solution given that the problem is an acyclic CSP. In section 4 we extend the approach to objective functions whose components are **contained** within the CSP's constraints. Section 5 generalizes the scheme to any objective function and any CSP, section 6 provides a complexity analysis and section 7 provides concluding remarks.

2. Structure of constraint-networks

The structure of a constraint network, represented by a graph, has two forms, called **primal and dual.**. The **primal-constraint-graph** represents variables by nodes and associates an arc with any two nodes residing in the same constraint. The **dual-constraint-graph**, represents each constraint by a node and associates a labeled arc with any two nodes that share at least one variable. The dual constraint graph can be viewed as the primal graph of an equivalent constraint network, where each of the constraints of the original network is a variable (called a c-variable) and the constraints call for equality among the values assigned to the variables shared by any two c-variables.

Consider, for example, a CSP with variables A, B, C, D, E, F and constraints on the subsets (ABC),(AEF), (CDE), and (ACE), and assume that the the domain of each variable is the set $\{0,1,2\}$. The constraint for the variable subset (ABC) is given in Table 1; it imposes an **order** on the 3-tuples and can be expressed concisely as $C_{ABC} = \{(a,b,c) \mid a \leq b \leq c\}$. The same constraints apply also to the subsets (AEF) and (CDE). The constraint for the variable subset (ACE) is $C_{ACE} = \{(a,c,e) \mid a \leq c < e\}$, and is given explicitly in Table 2. Figures 1(a) and 1(b) depict the primal and dual constraint-graph of this problem, respectively.

A	B	C
0	0	1
0	1	2
0	0	2
1	1	2
0	0	0
0	1	1
0	2	2
1	1	1
1	2	2
2	2	2

Table 1

A	C	E
0	0	1
0	1	2
0	0	2
1	1	2

Table 2

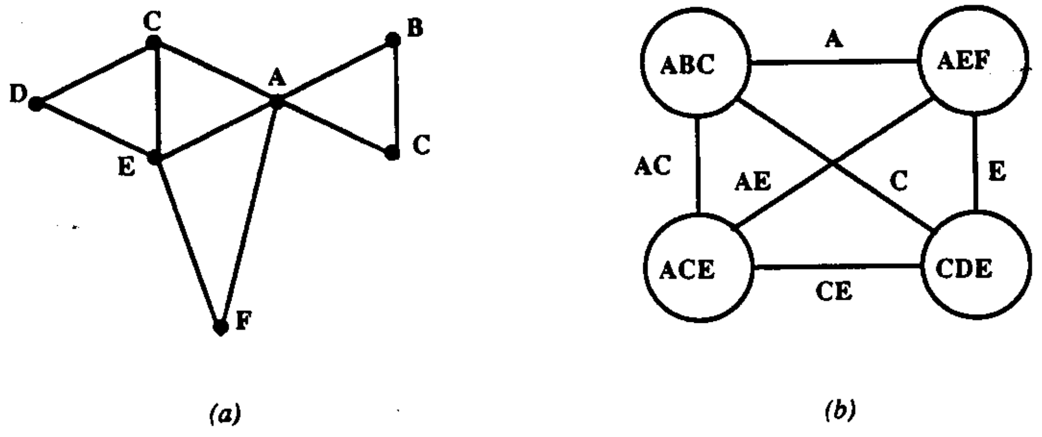

Figure 1. A primal and dual constraint graphs of a CSP

A constraint is considered **redundant** if its elimination from the network (and the possible removal of the corresponding **redundant arc** from the constraint graph) does not change the set of all solutions. In the dual constraint graph it is easy to identify redundant arcs and remove them from the graph, since all constraints are equalities. Any cycle for which all the arcs share a common variable contains redundancy, and any arc having all of the variables in its label common in some cycle, can be removed. The graph resulting from such removal of arcs is called a **join-graph**, and the corresponding network is an equivalent representation of the original network.

For example, in figure 1(b), the arc between (AFE) and (ABC) can be eliminated because the variable A is common along the cycle (AFE)—A—(ABC)—AC—(ACE)—AE—(AFE), so the consistency of the A variables is maintained by the remaining arcs. By a similar argument we can remove the arcs labeled C and E, thus turning the join-graph into a **join-tree** (figure 2).

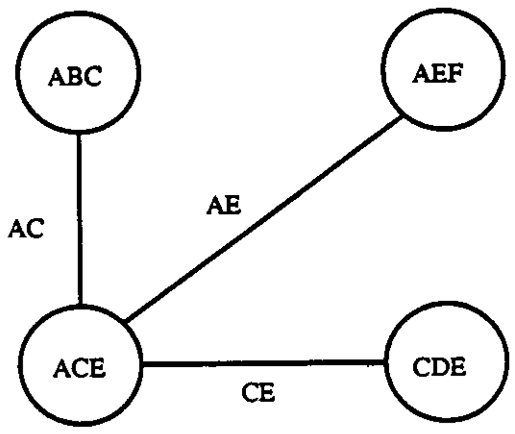

Figure 2. A Join-Tree

A CSP which possesses a join-tree, is said to be **acyclic**. Solutions for acyclic CSPs can be derived efficiently. If there are p constraints in the join-tree, each with at most l subtuples, the CSP can be solved in $O(pl \cdot \log l)$ [Dechter, 1987a].

3. Finding a best utility solution for join-trees

Let x_S denote a tuple in the constraint S (S, Q, R, etc., will stand for variable subsets as well as for the constraints provided for them.). If Q is a variable-subset of S then let $(x_S \mid Q)$ denotes the projection of x_S on Q. For example, the projection of $x_{AEF} = (0,1,0)$ on EF is the subtuple (1,0). The utility of a subtuple is the restriction of the utility function (2) to the tuples' variables.

The join-tree of an acyclic CSP can be made into a **directed** tree by designating one of the nodes as the root, orienting all edges from the root outward, and identifying the set of child nodes and one parent node for each of the non-root. Let $S_1, \ldots, S_l$ be the children of a constraint S, and let T_S denote the subset of variables in the subtree rooted at S (see figure 3).

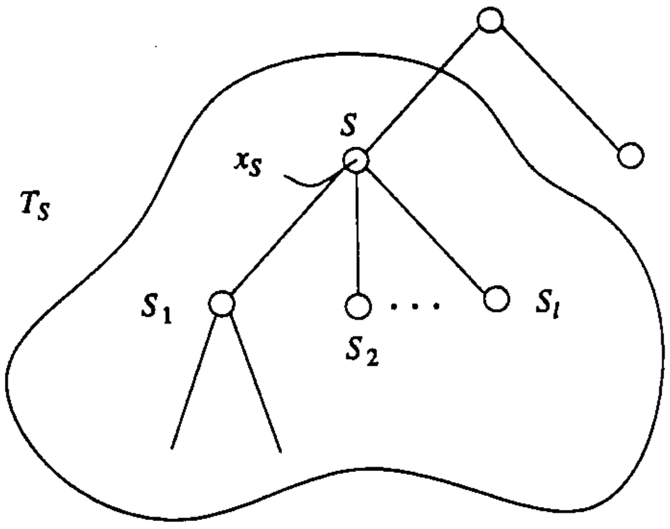

Figure 3

With each tuple x_S of C we associate a value, $v(x_S)$, which is equal to the utility of the best instantiation for variables in T_S compatible with x_S. Consistency in this case simply means that the projection of the partial solution on the variables in S is equal to x_S. Clearly, the max-v-value associated with the tuples of the root node is the utility of the optimal solution of the entire network. It is easy to see that for any parent node, S, and it children, $S_1, \ldots, S_l$, the v-values satisfy the following recurrence:

$$v(x_S) = \{u(x_{T_s}) \mid (x_{T_s} | S) = x_S \text{ and for every i, } v(x_{T_s} | S_i) = \max \{v(x_{S_i})\} . \quad (3)$$

Namely, the partial solution with the highest utility in T_S compatible with a tuple of S is composed of those sub-tuples of its child nodes having the highest value among those that are consistent with it. The reason that we can take the best in each child to find a globally optimal solution is that the join-tree property guarantees that all the variables which are shared by the children appear in their parent.

This recurrence suggests that the computation of the v-values can be performed from the leaves to the root, recursively. Values of leaf-nodes-tuples are their utility values. A node will compute its v-values only after all its child-nodes have computed their owns. For each tuple x_S the parent node chooses from each child node a consistent maximum-value tuple, finds a partial solution in its subtree which is composed of these sub-tuples, computes its utility, and associates this value with x_S. During this process the tree can also be made directional-arc-consistent [Dechter, 1987a], namely, tuples of a constraint which cannot be extended to a solution can be eliminated.

As an example consider the join-tree of figure 2. ACE is chosen to be the root of the tree, while ABC, AEF, and CDE are its child nodes. Suppose that the utility function is given by:

$$u(a,b,c,d,e,f) = 6a+5b+4c+3d+2e+f \tag{4}$$

Figure 4 displays the computation of the values associated with each tuple of the root node ACE (given in Table 2). Consider for instance the tuple (0,1,2). This tuple is consistent (w.r.t. equality constraints) with only two tuples of ABC, namely, (0,0,1) and (0,1,1), with one tuple ((0,2,2)) of AEF, and with two tuples ((1,1,2) and (1,2,2)) of CDE. The values of these leaf tuples are their utility values (indicated in parenthesis), and the highest value (starred in the figure) will be chosen. The utility of tuples of ABC ,derived by restricting (4) to the variables $\{A,B,C\}$, are given by $u(a,b,c) = 6a+5b+4c$ yielding utility of 4 to the tuple (0,0,1) and utility 9 to the tuple (0,1,1). Similarly the utility of a tuple of AEF is computed by $u(a,e,f) = 6a+2e+f$ etc. Thus, tuple (0,1,1) of ABC, tuple (0,2,2) of AEF, and tuple (1,2,2) of CDE are selected. The solution, composed of these subtuples, is ($ABCDEF$ = 011222) whose utility is 21. Similarly each tuple of ACE is associated with a value, and the highest value (27) is achieved for tuple (1,1,2) with a corresponding solution $ABCDEF$ = 111222.

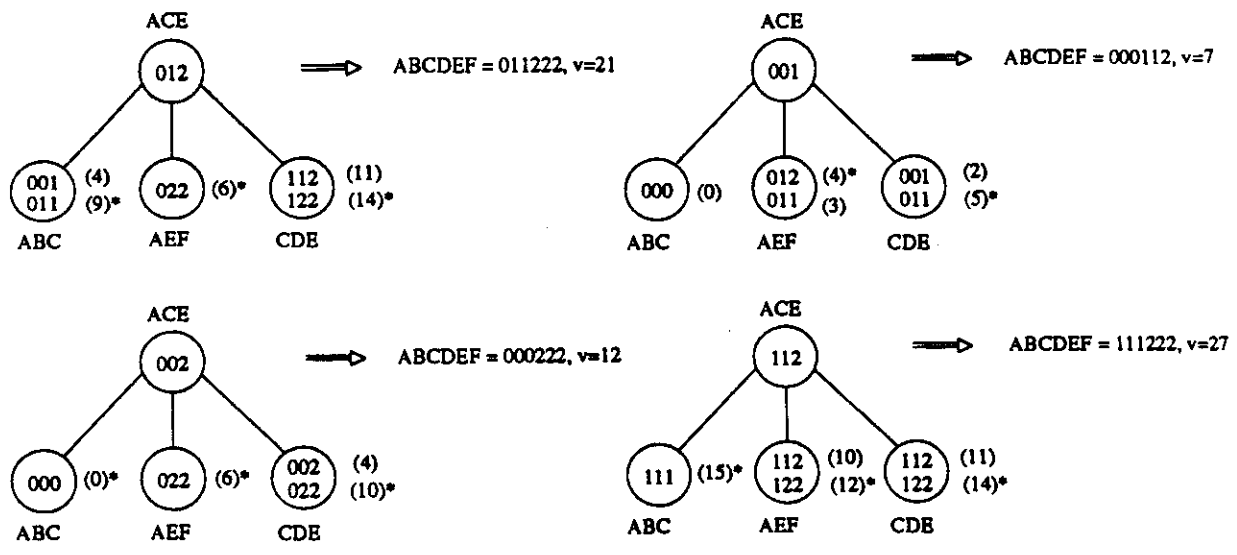

Figure 4

4. Extension to a general objective function within acyclic CSPs

When the objective function is generalized to be a function of the form (1) with the restriction that each function component is contained within at least one constraint in the acyclic CSP, we say that it obeys the **containment requirement** w.r.t. the CSP. The extension of the previous scheme to a general objective function that satisfies the containment requirement is straightforward.

Consider the constraint problem of figure 2, together with a new objective function given by: $f(a,b,c,d,e,f) = f_1(a,b) + f_2(c) + f_3(a,c,e) + f_4(a,f)$. The function's components are defined via tables and will not be specified here. Here, f_1 is defined for variables $\{A,B\}$ which are contained in constraint ABC, f_2 is defined over $\{C\}$ which is contained in ABC, ACE and CDE, f_3 is defined over $\{A,C,E\}$ contained in ACE and so on. Having this property, each function component can be associated with one of the constraints that contains it, and correspondingly, the functional value of each tuple of a constraint can be computed using the function components which were assigned to it. For instance, in our example, the components f_1 and f_2 can be assigned to constraint ABC and therefore each of its tuples will be associated with the function value: $f(a,b,c) = f_1(a,b) + f_2(c)$. Similarly, tuples from (ACE) and (AEF) are assigned: $f(a,c,e) = f_3(a,c,e)$, $f(a,e,f) = f_4(a,f)$ respectively. Since tuples from (CDE) were not assigned any function component they will be assigned a constant function value of "0".

Let us denote by $f_{/S}$ the objective function components which are assigned to constraint S (e.g., $f_{/(ABC)} = f_1(a,b) + f_2(c)$). As before, with each tuple x_S of a constraint S, $v(x_S)$ stands for the optimal value of $f_{/T_s}$ of a subtuple in T_S which is compatible with x_c (see figure 3). The computation of this values for a tuple of S given the values of all subtuples in his child constraints and given that it is consistent with at least one tuple in each of its child nodes is:

$$v(x_S) = f_{/S}(x_S) + \sum_{S_i} \max_{x_s | S \cap S_i = x_{s_i} | S \cap S_i} v(x_{S_i}) \qquad (5)$$

If x_c is not compatible with any tuple of one of its neighboring constraint it should either be eliminated or assigned a negative value which will not enable its inclusion in any solution.

The computation of the v-values will be performed from leaves to root recursively as in the case of the utility function. Values of leaf constraints are their assigned restricted objective function. To make the computation of a parent's values based on its child's values more efficient, each child can project its tuples on the variables common to itself and its parent, i.e. the variables that label the arc between them, and associate each projected tuple with the maximum values among the corresponding child tuples. Namely, a child node will go over its tuples, project each on the labeled variables and associate with the projection the highest value seen so far. If the projected relation is sorted, searching for a tuple can be reduced to logarithmic complexity rather then a linear one. Thus, if the number of tuples associated with each constraint is bounded by t, projecting the constraint of a child constraint on its outgoing arc and associating the projected tuples with their maximum values is bounded by $O(t \cdot logt)$. The parent constraint, for each of its own tuples will look for the projected tuples that match it. Since the projected constraint is sorted this search takes $O(logt)$, and since it is done for each tuple the overall complexity is $O(t \cdot logt)$ as well. Thus the whole communication between a child and its parent is bounded by $O(t \cdot \log t)$. Since in acyclic CSPs there are at most n constraints (n being the number of variables), the overall computation is bounded by $O(n \cdot t \cdot logt)$.

5. The general case

In general, the CSP is not necessarily acyclic and the objective function's components do not obey the **containment** requirement. Consider the following constraint problem. The variables are denoted by A, B, C, D, E and the constraints are (AB), (BC), (BD), (DE), (AD), and (CE). The objective function is: $f(a,b,c,d,e) = f_1(a,b) + f_2(b,c) + f_3(b,d,e)$. The primal and the dual constraint graphs are depicted in figure 5a and 5b respectively.

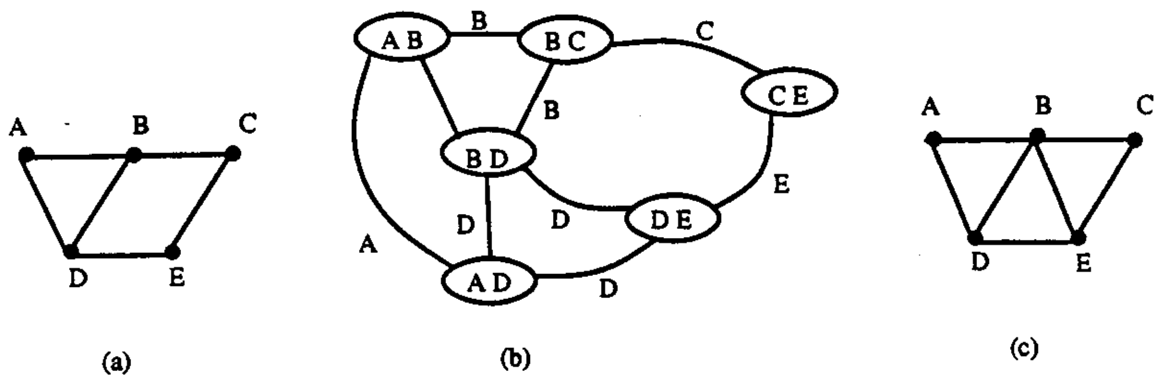

Figure 5

It can be easily verified the the problem is not acyclic (i.e., the graph of figure 5b does not possess a join-tree). Also, the function component f_3 is not contained in any given constraint. To deal with such cases we use the tree-clustering scheme presented in [Dechter, 1987b] while modifying it to account for the structure imposed by the objective function in addition to structure associated with the constraints themselves. We do that by augmenting in the primal graph of the constraint problem, arcs which are imposed by the components of the objective function. Namely, any two variables residing in the same function component will be connected in the primal graph, and the resulting graph will be called the **augmented primal graph**. In our example figure 5c depicts the augmented primal graph in which the arc (B, E) is added to account for component f_3 in the objective function.

The tree-clustering scheme will transform a general constraint optimization problem into an equivalent acyclic CSP with an objective function that satisfies the containment requirement. Once this transformation is applied, the rest of the problem can be solved using the scheme presented in sections 3 and 4. A summary of the tree-clustering scheme is given next.

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 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 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

* 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**.

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 constraint-satisfaction 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).

In order to transform a constraint optimization problem into an acyclic CSP that its objective function satisfies the containment requirement we will use the augmented primal graph rather then the primal graph as an input to the clustering scheme. Considering our example of figure 5, the augmented primal graph (see figure 5c) is already chordal, however, the maximum cliques do not correspond to the original constraints. The maximal cliques (ABC), (BDE) and (BEC) constitute an acyclic scheme whose join-tree is given in figure 6. Each cluster should be solved as a separate CSP and the components of the objective function, now satisfying the containment requirement, can be assigned to this new constraints.

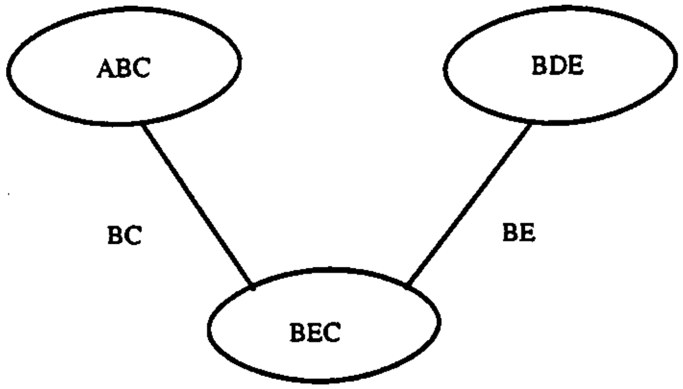

Figure 6

The complexity of the clustering scheme is dominated by step 4 which requires the solution of the clustered CSPs. If r is the size of the maximum cluster then its solution is bounded by $O(k^r)$ when k bounds the number of values in the domain of each variable. The space complexity is also bounded by $O(k^r)$ which is the size of the new constraint associated with the new cluster. The maximum size of clusters generated can be easily determined by computing the **width** of the ordered augmented primal graph resulting from the triangulation algorithm. The width of an ordered graph is the maximum width of each of its nodes, and the width of a node is the number of nodes connected to it which precedes it in the ordering. Let d denotes any ordering of the variables, and $W^*(d)$ denotes the width of the graph resulting from the fill-in procedure. It can be shown that the size of the maximum clique, which is also the size of the maximum cluster, is $W^*(d)$. Thus, the overall complexity of the clustering scheme is bounded by $O(k^{W^*(d)})$. Finding the ordering d for which $W^*(d)$ is minimal is an NP-complete task [Arnborg, 1987.] and the m-ordering suggested in this paper is one possible heuristic which provides good results.

The overall computation of our algorithm which is a composed of the clustering scheme and the optimal solution of acyclic CSPs with an objective function that satisfies the containment requirement, is given by:

$$O(k^{W^*(d)}) + O(t' \cdot \log t')$$

where t' is the maximum number of tuples in each constraint of the clustered CSP. Since $t \le k^{W^*(d)}$ we get that the complexity is bounded by $O(k^{W^*(d)})$.

The nice feature of associating the complexity analysis with the structure of the problem is that we can isolate the additional computation imposed by the optimization task. Comparing the width resulting from processing the primal graph and the augmented primal graph give us an upper bound on this additional complexity. Let W^*_p be the width of the processed primal graph, while W^*_a be the width of the processed **augmented primal graph** (note that different orderings may be used in the two graphs). Since

$$k^{W^*_a} = k^{W^*_p}\{k^{W^*_a - W^*_p}\}$$

we see that the optimization multiplies the power of the exponent by $\frac{W^*_a}{W^*_p}$. The above ratio can be consulted to decide whether or not to look for real optimal solution or to be satisfied with heuristic algorithms which will find good but not necessarily optimal solutions.

7. Conclusions

This paper shows that finding an optimal solution for a CSP need not necessarily involve an exhaustive search among all consistent solutions but can be incorporated naturally into the process of finding a consistent solution. In many problem instances the interaction between the objective function and the CSP do not add any complexity burden on the task of finding a consistent solution, and when it does, the additional computation complexity can be estimated ahead of time and be used to decide between exact or a heuristic approach to optimization.

References

[Arnborg, 1987.] S. Arnborg, D. G. Corneil, and A. Proskurowski, ''Complexity of finding embeddings in a k-tree,'' *Siam Journal of Algebraic and Discrete Methods.*, Vol. 8, No. 2, 1987., pp. 277-284.

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

[Dechter, 1987b] Rina Dechter, ''Tree-Clustering Schemes for Constraint-Processing,'' Cognitive Systems Laboratory, Tech. Rep. (R-92), 1987.

[Haralick, 1980] R. M. Haralick and G.L. Elliot, ''Increasing tree search efficiency for constraint satisfaction problems,'' *AI Journal,* Vol. 14, 1980, pp. 263-313.

[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.

[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.