REMOVING REDUNDANCIES IN CONSTRAINT NETWORKS

Avi Dechter

February 1987
CSD-870006

REMOVING REDUNDANCIES IN CONSTRAINT NETWORKS

Avi Dechter

Department of Management Science
California State University, Northridge, CA 91330
and
Cognitive Systems Laboratory, Computer Science Department
University of California, Los Angeles, CA 90024

Rina Dechter

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

Topic area: Automated reasoning (constraint satisfaction)
Track: science track
Keywords: constraint satisfaction problems, path-consistency, path-redundancy, backtracking

ABSTRACT

The removal of inconsistencies from the problem's representation, which has been emphasized as a means of improving the performance of backtracking algorithms in solving constraint satisfaction problems, increases the amount of redundancy in the problem. In this paper we argue that some solution methods might actually benefit from using an opposing strategy, namely, the removal of redundancies from the representation. We present various ways in which redundancies may be identified. In particular, we show how the path-consistency method, developed for removing inconsistencies can be reversed for the purpose of identifying redundancies, and discuss the ways in which redundancy removal can be beneficial in solving constraint satisfaction problems.

REMOVING REDUNDANCIES IN CONSTRAINT NETWORKS

Avi Dechter and Rina Dechter

1. Introduction

A binary *Constraint Satisfaction Problem (CSP)* is concerned with the task of finding either one or all of the n-tuples allowed by a given *network of binary constraints* (or finding that no such n-tuple exists).

A network R of binary constraints defined on a set of variables $\{X_1, \ldots, X_n\}$ is a set of relations R_{ij} from every variable X_i to every variable X_j. A network of constraints R represents a unique (possibly empty) n-ary relation ρ (i.e., a subset of the space $X = dom(X_1) \times \cdots \times dom(X_n)$, where $dom(X_i)$ denotes the domain of X_i) such that an n-tuple t is allowed by ρ if and only if its projections on all the uni-dimensional and two-dimensional subspaces of X simultaneously satisfy the binary constraints of the network R.

A *constraint graph* corresponding to a network of constraints consists of a vertex for each variable and an edge for each binary constraint which is not the *universal constraint* (i.e., comprising the entire subspace).

CSPs are inherently difficult problems to solve and typically are solved using some sort of a backtracking search algorithm. The issue of improving the performance of these algorithms has been on the agenda of researchers in Artificial Intelligence for quite some time (e.g., [Gaschnig1979, Haralick1980, Bruynooghe1981] , as many AI tasks can be formulated as CSPs (e.g., line-drawing analysis [Waltz1975] and reasoning about temporal intervals [Allen1985]). Observing that there are possibly many equivalent network representations of a given n-ary relation ρ, attempts were made at finding ways for moving from some initial representation to one which is better suited to be solved by backtracking. A central theme in the literature on this sub-

ject is that of the benefit of removing local inconsistencies from the problem's representation. Such inconsistencies may be discovered either prior to, or during, backtracking [Mackworth1977, Dechter1986c].

An inconsistency, in general, is a state of affairs where a certain action (i.e., instantiating a variable to a certain value during backtracking) is permitted by one piece of data (considered in isolation), and prohibited by another. If the data permitting the action is consulted first, then the algorithm might expand much work based on the assumption that the particular action is globally permitted only to discover later that this assumption if false. An inconsistency, once discovered, is eliminated by recording the fact that the action is not permitted.

The removal of an inconsistency results in a redundancy in the database, namely, a situation whereby the fact that an action is prohibited is expressed by more than one piece of data. Excessive redundancy in the data has its own potential adverse effects on our ability to solve the problem efficiently. First, it often increases the amount of data that has to be stored, and second, it tends to obscure any special structure the problem might have, and which may be exploited by the solution procedure. Of particular importance is the *connective structure* of the constraint graph, which strongly affects to the tractability of the problem. The strength of the relationships between the connective structure of the constraint network and the complexity of it solution is most vividly demonstrated by Freuder's [Freuder1982] conditions for backtrack-free search. in particular, he shows that is the constraint graph forms a tree, then backtrack-free search is guaranteed.

Thus, if the problem already has some desirable structure, then it might be beneficial to modify the process of obtaining consistency so that the structure is preserved. Furthermore, by eliminating certain types of redundancies, while adding inconsistencies, it may be possible to bring about an improvement in the representation of the problem.

This paper is concerned with issue manipulating the representation of a given CSP by identifying redundant constraints, namely, constraints whose removal from the network, while changing the connective structure of the problem, does not affect the set of solutions ρ.

2. Consistency and Redundancy Re-defined

The definition of both consistency and redundancy in networks of constraints relies on a distinction that can be made between the *direct constraint* R_{ij} between two variables X_i and X_j, and constraints *induced* on them by the rest of the network. An induced constraint can be thought of as a binary relation consisting of all the instantiations of the variable-pair (X_i, X_j) which are consistent with some subset of the other constraints. The intersection of all the constraints induced on (X_i, X_j), i.e., the constraint induced by all the constraints except the direct constraint, is called the *network-induced* constraint, denoted $\bar{R}_{ij}$. The *global* constraint between X_i and X_j is the intersection of the direct and the network-induced constraints.

An inconsistency in a constraint occurs when some pair of values is permitted by (i.e., is part of) the direct constraint between two variables but prohibited by their network-induced constraint. An inconsistency can be eliminated by simply erasing the value-pair from the direct constraint between the variables. Clearly, such a change does not alter the set of all solutions in any way. When none of its pairs are inconsistent with the network-induced constraint, a direct constraint is said to be *explicit*.

Definition A direct constraint R_{ij} is said to be explicit if the remainder of the network does not add any further restrictions on the global constraint between X_i and X_j, i.e., if $R_{ij} \subseteq \bar{R}_{ij}$.

A pair of values is permitted by an explicit constraint if, and only if, it is part of at least one solution. Montanary [Montanari1974] has shown that if a relation can be represented by a binary network of constraints then there is a unique representation where all the constraints are explicit, called the *minimal network of constraints*.

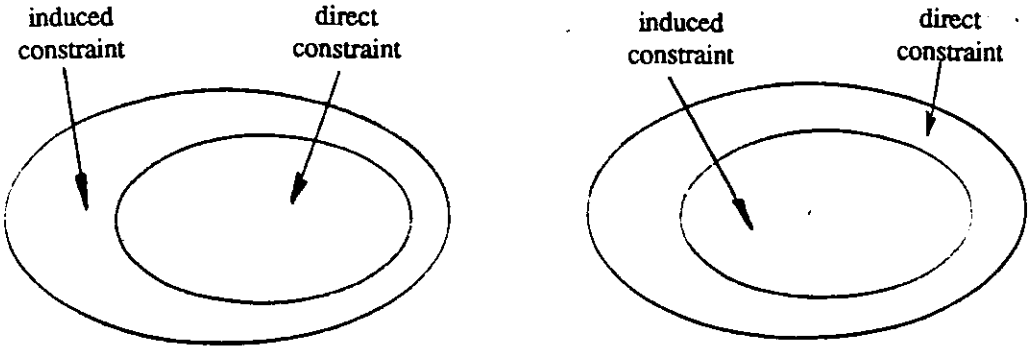
In contrast, redundancy in a constraint occurs when a pair of values which is prohibited by the network-induced constraint is also prohibited by (i.e., absent from) the direct constraint between two variables. A redundancy is eliminated by *adding* the pair to the direct constraint. This change, too, cannot have any effect on the set of all solutions.

Of special interest is the case where *all* the pairs that are prohibited by direct constraint are already prohibited by the induced constraint. In this case the entire constraint is said to be redundant.

Definition: A direct constraint R_{ij} is said to be redundant if it does not add any further restrictions on the global constraint between X_i and X_j, i.e., if $\overline{R}_{ij} \subseteq R_{ij}$.

All redundancies associated with a redundant constraint are eliminated by simply removing the entire constraint from the network.

The definitions of explicit and redundant constraints are given graphical representation in the Venn diagrams of Figure 1. Observe that a constraint can be explicit and redundant at the same time. In this case the direct constraint and the network-induced constraints coincide.



(a) Explicit Constraint (b) Redundant Constraint

Figure 1: Explicit and Redundant Constraints

Improving the representation of a CSP involves two types of operations: (1) Eliminating inconsistencies and making each constraint as explicit as possible, and (2) Eliminating redundancies by identifying and removing redundant constraints. Although these operations represent two opposing objectives they are, in large part, orthogonal to each other. First, a constraint which is redundant will never become non-redundant as a result of making any other constraint more explicit, because this can possibly only tighten the network-induced constraint and thus make the constraint more redundant. Second, a constraint can never become less explicit as a result of the removal of another, redundant, constraint since this can possibly only loosen the induced constraint and make the constraint even more explicit.

It is clear that in the "ideal" representation of a CSP the constraint between any two variables should be either explicit or universal (i.e., non-existing). Such networks of constraints are called U-minimal.

Definition: A network of constraints for which a subset U of constraints are universal and all other constraints are explicit is said to be U-minimal.

The properties of U-minimal networks are discussed in [Dechter1986a]. The task of obtaining a U-minimal representation of a given network of constraints is as illusive as that of finding the minimal network of constraints because it requires the knowledge of the network-induced constraint for all pairs of variables. For the same reason, the task of deciding whether a given constraint is redundant or not is a difficult one. An approximate method, based on the notion of path-consistency is presented in the following section.

3. Path-Redundancy

Montanary suggested that the minimal network of constraints may be approximated by replacing the requirement that each constraint be explicit by a weaker condition, called *path consistency*.

Definition: A pair of values (x_i, x_j) is said to be *allowed* by a path of length m through nodes $(V_i=V_{k_0}, V_{k_1}, \ldots, V_{k_{m-1}}, V_{k_m}=V_j)$ if there is a sequence of values $(z_1, z_2, \ldots, z_{m-1})$ such that

$$R_{k_0k_1}(x_i, z_1) \text{ and } R_{k_1k_2}(z_1, z_2) \text{ and } \cdots \text{ and } R_{k_{m-1}k_m}(z_{m-1}, x_j) .$$

Definition: A pair of values (x_i, x_j) which is allowed by every path from node V_i to node V_j in the complete network R is called *path-induced.* Otherwise, it is called *path-illegal*

Definition: A binary constraint R_{ij} is said to be path-consistent if all of its pairs are path-induced. A network of constraints R is path-consistent if all of its constraints are path consistent.

The requirement of a constraint being path-consistent is weaker than that of being explicit because every explicit constraint must be path-consistent, but not every path-consistent constraint is necessarily explicit.

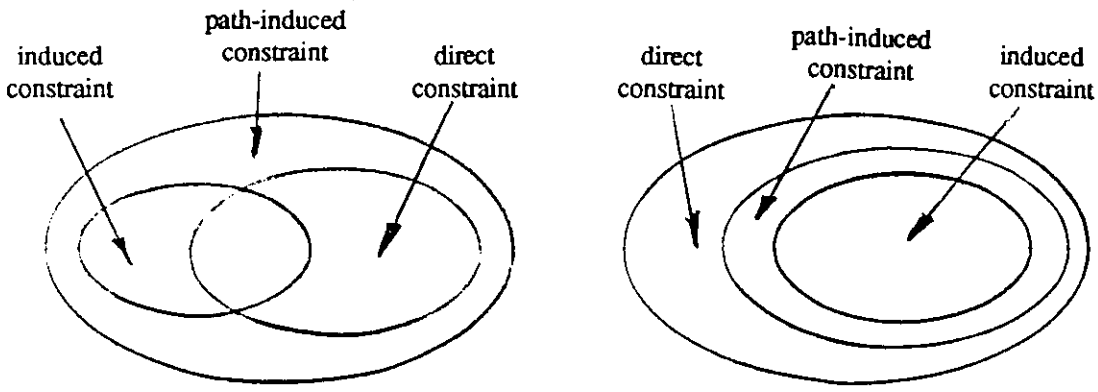
Montanary showed that a pair of values is path-induced if, and only if, it is allowed by all paths of length $m=2$. Path consistency algorithms repeatedly check all paths of length $m=2$ and remove all path-illegal pairs until no such pairs remain [Montanari1974, Mackworth1977].

The task of recognizing all the redundant constraints in a network can be approximated by replacing redundancy with a stronger requirement called *path-redundancy.*

Definition: A constraint R_{ij} is path-redundant if every path-induced pair is already permitted by it.

The condition of a constraint being path-redundant is stronger than that of being redundant, since every path-redundant constraint must be redundant, but not every redundant constraint is necessarily path-redundant.

The definitions of path-consistency and path-redundancy, and their relationships to those of consistency and redundancy are shown graphically in Figure 2. As the diagrams show, a path-consistent constraint is not necessarily explicit, but a path-redundant constraint must be redundant.



(a) Path-Consistent Constraint (b) Path-Redundant Constraint

Figure 2: Path-Consistent and Path-Redundant Constraints

A convenient way to check whether a given constraint is path-redundant or not is to consider a set of value-pairs which is guaranteed to contain the path-induced constraint (for example, the Cartesian product of the domains of the two variables involved), and to check the path-legality status of each pair of this set which *is not* in the direct constraint. The direct constraint is path-redundant if, and only if, all such pairs are path-illegal. This process is, essentially, the reverse of achieving path-consistency, since it can be thought of as the process of adding path-illegal pairs to the direct constraint in an attempt to make it universal. If this attempt is successful, then the constraint is redundant.

An algorithm for determining whether a constraint R_{ij} is path-redundant is given below.

The algorithm returns *true* if the constraint is path-redundant and *false* otherwise.

```
procedure PATH-REDUNDANT((i, j))
begin
  let U_ij = dom(X_i) × dom(X_j)
  for all k ≠ i,j
    let Q = U_ij − R_ij
    for each p ∈ Q
      if not PERMIT (p, (i,j), k) let R_ij = R_ij ∪ {p}
    end
    if R_ij = U_ij return true
  end
  return false
end
```

The procedure **PERMIT**$(p, (i,j), k)$ (given below) returns **true** if the the pair p is permitted for the variable pair (X_i, X_j) by the path $(i-k-j)$, and **false** (i.e., the pair p is path-illegal) otherwise. Thus, the algorithm examines all the paths of length 2 anchored at nodes i and j, and augments the relation R_{ij} by pairs that are found to be path-illegal. If this augmentation process results with the constraint becoming the universal constraint, then the original constraint is redundant.

```
procedure PERMIT(p, (i,j), k)
begin
  let D_k be the domain of X_k
  let p be (p_i, p_j)
  for all v ∈ D_k
    if (p_i, v) ∈ R_ik and (v, p_j) ∈ R_kj return true
  end
  return false
end
```

The **PERMIT** procedure considers all legal values of X_k, and returns **true** as soon as one value is found consistent with the pair p. The complexity of **PERMIT** is $O(k)$, where k is the number of possible values of each variable. The complexity of **PATH-REDUNDANT** is,

therefore, $O(nk^3)$ where n is the number of variables.

The status of being redundant, for a given constraint, is dependent on the other constraints in the network, and any change in the network may affect this status. In particular, the removal of one redundant constraint, while not affecting the set of solutions, may cause other redundant constraints to become non-redundant. Therefore, a set of constraints which are found to be path-redundant in a given network, may not be removed simultaneously, but rather in sequence, where the path-redundancy of the constraints in the set needs to be re-examined after the removal of each one them. Constraints that were non-redundant to start with, or that became non-redundant in the process, need not be checked again as they cannot become redundant again. The number of constraints that can be removed in this way is dependent on the sequence in which they are considered.

When the removal of any constraint in some set of redundant constraints does not diminish the redundancy of any of the other constraints in the set, we say that they are all independently redundant. For example, if the process of path-consistency results in adding to the network constraints that were universal in the original network, then all the added constraints are independently redundant since their simultaneous removal will not affect the solution set. Notice, however, that while these constraints are redundant, they are not necessarily path-redundant, and thus the algorithm **PATH-REDUNDANT** is not guaranteed to recognize their redundancy.

On the other hand it is easy to see that a set of path-redundant constraints which are not adjacent to one another, i.e., no two of them share a common variable, are independently path-redundant. This is so because only paths of length $m = 2$ are used to determine the path redundancy of a constraint.

The property of a set of constraints being independently redundant is desirable because it alleviates the need to search among all possible ordering of the constraints for some subset of them that may be removed.

4. The Use of Redundancy Elimination in Problem Solving

Elimination of redundant constraints is expected to be beneficial for solution methods that rely on the connective structure of the problem as depicted by its constraint graph.

Backtracking algorithms benefit from consulting the constraint graph in two ways. First, it provides a simple way of *backjumping* [Gaschnig1979, Dechter1986c]. Backjumping is an improvement of standard backtracking whereby, at a deadend, the algorithm goes back to the first variable which could be the "reason" for the deadend (rather than to previous variable in the stack, as called for by standard backtracking). A variable which is not connected (directly or indirectly) to the deadend variable cannot possibly be the source of the deadend, and thus it is always safe to jump back to the first available variable which is connected to the deadend variable while pruning the search tree.

Second, consulting the constraint graph can reduce the amount of work required for the expansion of nodes in the search tree. This is so, because no consistency checks are required between the node being expanded and nodes with which it is not directly connected.

Utilizing these features has resulted in substantial improvements in backtracking performance. The amount of improvement was directly related to the sparseness of the constraint graph [Dechter1986c]. Thus, the removal of redundant constraints, and, consequently, their corresponding edges in the constraint graph has a potential for improving the performance of backjumping.

As an example, consider the CSP given in Figure 3(a) consisting of three variables, A, B, and C, all having the same domain {a, b, c}. The constraint between the variables B and C is redundant and may be removed, resulting in the graph of Figure 3(b).

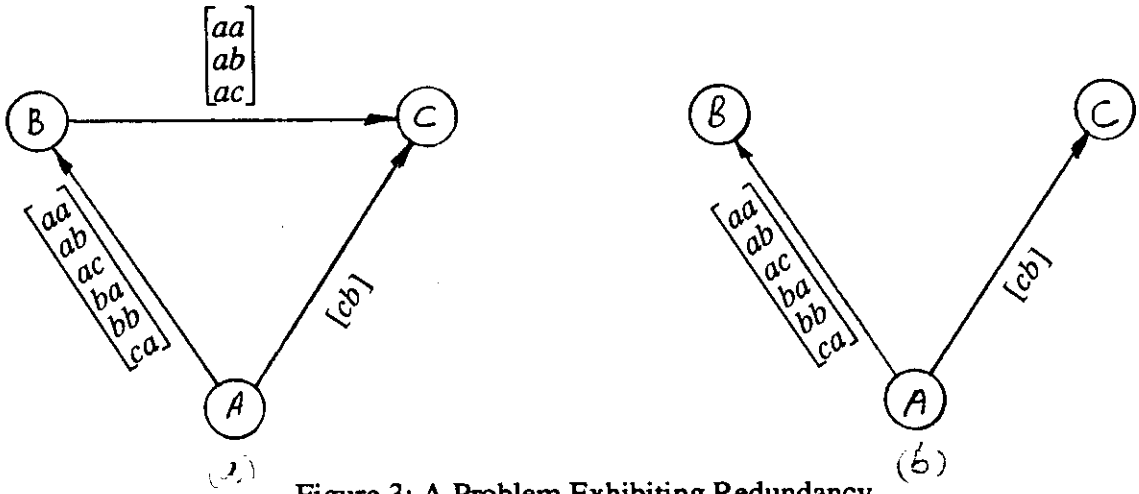


Figure 3: A Problem Exhibiting Redundancy

The search performed by a backjumping algorithm which considers the nodes in the order A, B, C is shown in Figure 4(a). If the redundant constraint is removed, then the search is reduced to that shown in Figure 4(b) because the first occurance of a deadend permits jumping back to variable A.

```
a
 aa  (deadend)
 ab  (deadend)
 ac  (deadend)
b
 ba  (deadend)
 bb  (deadend)
c
 ca
  cab  (solution)
```

(a)

```
a
 aa  (deadend)
b
 ba  (deadend)
c
 ca
  cab  (solution)
```

(b)

Figure 4: Backjumping Searches for the Problems in Figure 3

Removal of redundant constraints, while potentially beneficial, generally results in increasing the search space, which may wipe out its benefits. What is needed, therefore, is a way to identify redundant constraints whose removal will not cause the search space to increase. This can be accomplished by tying the notion of redundancy to the *order* in which the backtracking algorithm instantiates the variables. Let $d = X_{i_1}, \ldots, X_{i_n}$ be an ordering of the variables.

Definition: A constraint $R_{i_j i_k}$ is said to be *directional-path-redundant* with respect to d if its redundancy can be ascertained by considering only paths of length m=2 of the form $(X_{i_j} - X_{i_l} - X_{i_k})$, for $l > \max\{j, k\}$.

The removal of a constraint which is directional-path-redundant with respect to an ordering which is precisely the *reverse* of the order used by a backtracking algorithm, will not affect the search space explored by the algorithm. To see why this is so, consider a network whose constraint graph is given in Figure 5:

Figure 5 - An Ordered Constraint Graph

Suppose that backtrack algorithm considers the variables in the order X_1, X_2, X_3, X_4. If the constraint R_{23} is redundant with respect to an ordering $d = X_4, X_3, X_2, X_1$ (i.e., using X_1 alone), then its removal will not cause the algorithm to develop any more nodes because all the relevant information of this constraint must be contained in constraints R_{12} and R_{13}, which will be known to the algorithm by the time it gets to X_3. By contrast, if X_4 is also needed to establish the redundancy of constraint R_{23}, then the removal of the constraint may cause the algorithm to

consider values of X_3 that would not be considered had it remained in the network.

The notion of directional-path-redundancy has the additional advantage that all the constraints found to be directional-path-redundant with respect to some direction d, are independently redundant and thus may be removed simultaneously. To show this it enough to demonstrate that there is an order by which the constraints can be removed so that the removal of each constraint cannot possibly interfere with the directional-path-redundancy of the remaining constraints. For $d = X_{i_1}, \ldots, X_{i_n}$, any order such that a constraint $R_{i_j i_k}$, $k > j$ is checked before any constraint $R_{i_l i_m}$ $m > l$, if $k < m$, has this property. To see this, refer back to the network of Figure 5, and assume again that $d = X_4, X_3, X_2, X_1$. Further assume that both constraints R_{34} and R_{23} are directional path-redundant with respect to d. The removal of constraint R_{34} cannot possibly interfere with the redundancy of R_{23} with respect to X_1.

Directional-path-redundancy can be found by slightly modifying algorithm **PATH-REDUNDANT**. Applying it before backjumping may improve its performance and is guaranteed not to cause it to deteriorate.

The *cycle-cutset approach* [Dechter1986b] is using the fact that tree-structured CSPs can be solved in linear time by switching to a specialized tree-algorithm whenever the set of variables instantiated by a backtracking (or a backjumping) algorithm forms a cutset of the constraint graph. The efficiency of this approach depends of the sparseness of the constraint graph, and therefore this method too should perform better if redundant constraints are removed.

5. Conclusion

Researchers in the area of solving constraint satisfaction problems have emphasized the advantages of increasing the amount of redundancy in the network representation of problems. In this paper we point out that some benefits can be obtained by removing redundancies. We extend the idea of path-consistency to enable identifying redundancies and present some ways in which

redundancy elimination is useful.

References

[Allen1985] Allen, J. F., "Maintaining Knowledge about Temporal Intervals," in *Readings in Knowledge Representation,* R. J. Brachman H.J. Levesque, Ed. Los Altos, CA: Morgan Kaufman Publishers, Inc., 1985, pp. 509-521.

[Bruynooghe1981] Bruynooghe, Maurice, "Solving combinatorial search problems by intelligent backtracking," *Information Processing Letters,* Vol. 12, No. 1, February 1981.

[Dechter1986a] Dechter, A. and R. Dechter, "Minimal Constraint Graphs," UCLA, Computer Science Department, Cognitive Systams Laboratory, Los Angeles, CA, Tech. Rep. R-74, December, 1986.

[Dechter1986b] Dechter, R. and J. Pearl, "The cycle-cutset method for improving search performance in AI applications," UCLA, Cognitive systems, Computer Science, Los Angeles, Cal., Tech. Rep. R-58, September, 1986.

[Dechter1986c] Dechter, R., "Learning while searching in constraint-satisfaction-problems," in *Proceedings AAAI-86,* Philadelphia, Pensilvenia: August, 1986.

[Freuder1982] Freuder, E.C., "A sufficient condition of backtrack-free search.," *Journal of the ACM,* Vol. 29, No. 1, January 1982, pp. 24-32.

[Gaschnig1979] Gaschnig, J., "Performance Measurement and Analysis of Certain Search Algorithms," Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, Tech. Rep. CMU-CS-79-124, 1979.

[Haralick1980] Haralick, R. M. and G.L. Elliot, "Increasing tree search efficiency for cconstraint satisfaction problems," *AI Journal,* Vol. 14, 1980, pp. 263-313.

[Mackworth1977] Mackworth, A.K., "Consistency in networks of relations," *Artifficial intelligence,* Vol. 8, No. 1, 1977, pp. 99-118.

[Montanari1974] Montanari, U., "Networks of constraints :fundamental properties and applications to picture processing," *Information Science,* Vol. 7, 1974, pp. 95-132.

[Waltz1975] Waltz, D., "Understanding Line Drawings of Scenes with Shadows," in *The Psychology of Computer Vision,* P. H. Winston, Ed. New York, NY: McGraw-Hill Book Company, 1975.