MINIMAL CONSTRAINT GRAPHS

Avi Dechter

February 1987
CSD-870007

TECHNICAL REPORT
R-74
December 1986

MINIMAL CONSTRAINT GRAPHS

by

Avi Dechter and Rina Dechter
Cognitive Systems Laboratory, Computer Science Department
University of California, Los Angeles, CA 90024
Tel: (213) 825-3243
Net address: avi%locus.ucla.edu

Paper type: full paper
Topic area: reasoning
Track: science track
Keywords: constraint satisfaction problems, path consistency, backtracking

ABSTRACT

Strong relationships exist between the topological properties of the constraint graph of given constraint satisfaction problem (CSP) and the computational tractability of that problem. Generally speaking, it is best for the constraint graph to have as few edges as possible. This paper deals with two main issues. First, we discuss the type of information exhibited by the constraint graph and the relevance of this information to the solution of the CSP. Second, we consider the possibility of changing the representation of a given CSP in order to obtain a more informative constraint graph. Observing that ordinary path-consistency operation usually adds edges to the network, we propose a modified path-consistency scheme preventing this from occurring. We then develop two approximation algorithms (which do, in some sense, the reverse of path-consistency) for reducing the connectivity in the constraint graph, and prove their properties.

MINIMAL CONSTRAINT GRAPHS

Avi Dechter and Rina Dechter

1. Introduction

Graphs offer useful representations for a variety of phenomena. The constraint graphs associated with networks of binary constraints are a prominent example.

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 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).

A *Constraint Satisfaction Problem (CSP)* is concerned with the task of finding either one or all of the n-tuples allowed by the relation ρ represented by a given network of constraints R (or finding that it is empty). CSPs have applications in many AI areas. Picture recognition, electronic circuit analysis, and truth maintenance systems are examples.

The constraint graph has an obvious role as a means of visualizing a network of constraints. More significantly, however, strong relationships exist between the topological properties of the constraint graph and the tractability of the constraint satisfaction problem. For example, it was shown that if the constraint graph is a tree then the problem can be solved in time linear in the number of variables.

The utility of the constraint graph stems from the fact that it carries important *independency information* about the relations under consideration. The fact that the constraint graph is a tree, for example, means that instantiating any variable to some specific value renders the subproblems associated with the subtrees adjacent to the instantiated variable completely independent on one another, in the sense that choices made in one subproblem do not affect in any way choices made in any other subproblem. This is precisely the property that makes the solution of tree-structured CSPs an easy task.

This capacity for highlighting independencies, makes any constraint graph an *Independency map* (I-map [Pearl1986]), of the relation it represents. A graph G is said to be an *I-map* of a *dependency model I* if there is a one-to-one correspondence between the variables of the model and the vertices of G and if each *separation in the graph* implies a corresponding *conditional independence* in the model (where the exact meaning of conditional independence depends on the specific model).

An I-map works only in one way: it guarantees that vertices found to be separated always correspond to independent variables, but does not guarantee that all those shown to be connected are in fact dependent. A complete graph is a trivial I-map, but it does not show any independencies at all. An I-map (and, as such, a constraint graph) would carry the greatest amount of information when depicting as many independencies as possible, i.e., when it is *edge-minimal*.

The theory of graphoids [Pearl1986] uncovers sufficient conditions for a set of independencies among a finite collection of elements to possess a *unique edge-minimal I-map*. (These conditions define a *graphoid*.) Unfortunately, independencies induced by binary networks of constraints do not satisfy all of these conditions and, in general, do not have unique minimal I-maps. Similarly, as we show later, binary networks of constraints do not have unique edge-minimal constraint graphs. In this paper we examine the issue of improving the representation of a constraint satisfaction problem by reducing the number of edges in its constraint graph.

The paper has the following plan. The next section discusses the relationships between the tractability of a CSP and the amount of independencies exhibited by its constraint graph, and motivates the need for reducing the connectivity of the constraint graph. Section 3 briefly explores the properties of constraint graphs as I-maps. Section 4 develops the notion of reducibility of networks of constraints. Section 5 suggests a procedure for obtaining an approximation of a network of constraint whose constraint graph is minimal. A modified version of the algorithm, called directional path-redundancy, is discussed in Section 6. Section 7 contains conclusions and suggestions for further study.

2. Why Reduce the Constraint Graph?

The information conveyed in the fact that two variables are not directly constrained, i.e., that they are independent given some subset of variables, can be valuable, if it is exploited by algorithms that solve CSPs. As mentioned above, if the constraint graph of a CSP has a tree structure, it is possible to solve the problem in $O(nk^2)$ when n is the number of variables and k is the number of values [Freuder1982, Dechter1985]. Therefore, if the constraint graph can be reduced into a tree, the special tree-algorithm can be used to solve the problem efficiently.

Most problems, however, may not be reducible into a tree structure. In these cases it is possible to exploit independencies in the constraint graph in several ways.

One approach, called the *cycle-cutset approach* [Dechter1986a] is based on the following observation: If a subset of instantiated variables cuts all cycles in the constraint graph, the remaining problem can be solved in linear time by the tree-algorithm. Thus, a general search algorithm can be used to find a partial consistent solution for the variables in the cutset, and then the tree-searching algorithm can be invoked to solve the rest of the problem.

If the cycle-cutset contains c variables, then for each consistent instantiation of these variables, a tree-problem with (n-c) variables needs to be solved. In the worst case the tree-

problem needs to be solved k^c times. Since the complexity of the tree algorithm is linear we get that the the worst-case performance of the cutset approach when the cutset-size is c is k^c, while the general known upper-bound for such problems is $O(k^n)$. As the size of the cutset becomes smaller the worst-case bound is smaller and the problem can be solved more efficiently. Problems that have sparser constraint graphs will generally tend to have smaller cutsets and can be more easily solved.

Another enhancement of backtrack algorithms, called *graph-based backjumping*, also benefits from the independencies represented by the constraint graph. It is based on the idea that, upon encountering a dead-end, the backtrack algorithm should "jump" back directly to a variable that could have caused that dead-end rather than simply backing to the most recently instantiated variable, as would a naive backtrack algorithm do. Pertinent information for finding the culprit is readily available in the constraint-graph: The algorithm should go back to the most recent variable which is *connected* to the dead-end variable [Dechter1986b]. It is clearly advantageous, for realizing the full benefits of this approach, that the constraint graph exhibits as many independencies as possible, since this will result in bigger backjumps which, in turn, will cause larger chunks to be pruned off the search space.

3. Constraint Graphs as I-maps

In Section 1 we argued intuitively that a constraint graph is an I-map of the relation it represents. To show this we must first provide a precise interpretation for the notion of conditional independence in the present context.

Let ρ denote an n-ary relation, i.e., a subset of the space $S = dom(X_1) \times \cdots \times dom(X_n)$, where $T = \{X_1, X_2, \ldots, X_n\}$ is a set variables and $dom(X_i)$ is the domain of variable X_i. Let $S_X = dom(X_{i_1}) \times \cdots \times dom(X_{i_m})$ denote a subspace of S, where $X = \{X_{i_1}, \ldots, X_{i_m}\}$ is a subset of the variables, and let $\Pi_X(\rho)$ denote the *projection* of ρ onto S_X. A set of values

$x = \{x_{i_1}, \ldots, x_{i_m}\}$ is said to be *legal* in ρ for X if $x \in \Pi_X(\rho)$. A specific set of values x is referred to as an *instantiation* of X and corresponds to a *selection* of ρ, denoted $\sigma_{X=x}(\rho)$, which is a relation consisting precisely of those tuples of ρ that satisfy the instantiation.

Definition: A subset of variables X is said to be *independent* of another subset Y in some relation ρ if $\Pi_X(\sigma_{Y=y}(\rho)) = \Pi_x(\rho)$ for any legal instantiation y of Y.

It is easy to see that when X is independent of Y, so is Y on X, and we simply say that X and Y are independent. A necessary and sufficient condition for X and Y to be independent is that $\sigma_{X=x,Y=y}(\rho) = \sigma_{X=x}(\rho) \times \sigma_{Y=y}(\rho)$ for any x and y.

Definition: Two non-intersecting sets of variables, X and Y, are said to be *conditionally independent* given a third subset of variables, Z, if they are independent in the selection corresponding to any legal instantiation of the variables in Z. The notation $(X \perp Z \perp Y)$ will be used to denote the independence of X and Y given Z.

For example, consider the following relation involving four variables A, B, C, and D:

A	B	C	D
a	a	a	a
a	b	b	a
a	b	a	b
b	b	b	b

A and B are not independent in ρ, but are conditionally independent given D because they are independent in selections corresponding to instantiating D to both a and b. C and D are independent in ρ, but are not conditionally independent given A.

The notion of conditional independence is equivalent to that of *embedded multivalued dependency (EMVD)* in the theory of relational databases [Maier1983]. An EMVD $Z \rightarrow\rightarrow X$ (Y) (read "Z multivalued determines X in the context of Y") implies that $(X \perp Z \perp Y)$, and vice versa. The notion of embedded multivalued dependency is, in turn, a generalization of that of *mul-*

tivalued dependency (MVD). An MVD $Z \rightarrow\rightarrow X$ is equivalent to $(X \perp Z \perp T - (Z \cup X))$. Thus, multivalued dependencies represent a special type of conditional independencies where one set of variables makes another set independent on the rest of the variables.

It is easy to show that if, in a constraint graph, the nodes corresponding to a subset X of variables are separated from those corresponding to a subset Y by the nodes corresponding to a subset Z, then X is independent of Y given Z. Thus, the constraint graph must be an I-map of the relation that it entails (more precisely, of the independencies embedded in the model represeted by the relation).

On the other hand, not every I-map for a relation can be associated with a constraint graph. Consider, for example, the following relation ρ:

A	B	C	D
a	a	a	a
a	b	b	b
b	b	a	c

The following independencies can be observed in ρ (among many others):

$$(A \perp D \perp B),\ (A \perp D \perp C),\ (B \perp D \perp C),\ (A \perp \{B,C\} \perp D),\ (C \perp \{A,B\} \perp D),\ (B \perp \{A,C\} \perp D)$$

The two graphs in Figure 1 are minimal I-maps of this relation, i.e., no edge can be removed without destroying its I-mapness. However, while the graph in 1(a) is also a constraint graph of ρ, the graph in 1(b) is not, i.e., there is no way to back the edges of 1(b) with constraints whose simultaneous solution yields ρ.

4. Reducibility in Network of Constraints

The absence of an edge between two nodes in a constraint graph indicates that the constraint between the corresponding variables is the universal constraint. Minimizing the number of edges in the constraint graph of some CSP corresponds, therefore, to representing the problem by a network of constraints with as many universal constraints as possible.

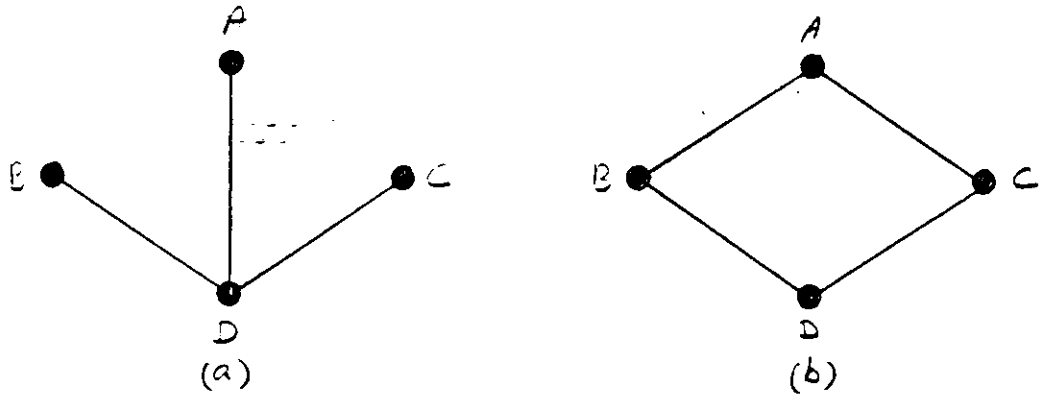

Figure 1 - Two minimal I-maps of ρ

Definition: A network of constraints for which a subset U of constraints are universal is called a *U-network*. U corresponds to the missing edges in the constraint graph.

Definition: Two networks of constraints R' and R'' which represent the same n-ary relation ρ are called *equivalent*.

Definition: A U-network of constraints R (or its constraint graph) is said to be *irreducible* if no equivalent U'-network R' exists such that $U \subset U'$.

For any network of binary constraints, a constraint between two variables may always be removed (i.e., replaced by the universal constraint), with the result being an equivalent network, if it is *redundant* in this network. This notion is given a precise definition next.

For any two variables X_i and X_j a distinction can be made between their *direct constraint* R_{ij}, and the constraint $\bar{R}_{ij}$ *induced* on them by the rest of the network. The 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 all other constraints in the network. The *global* constraint between X_i and X_j is the intersection of the direct and the induced constraints.

Definition: A 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 $\bar{R}_{ij} \subseteq R_{ij}$.

If at least one of the non-universal constraints of a network of constraints is redundant then, clearly, the network and its constraint graph are reducible. However, just the fact that none of the non-universal constraints is redundant does not necessarily imply that the network is irreducible. It might be possible to remove a non-redundant constraint at the expense of restricting one or more of the other non-universal constraints in the network.

As an example, consider the network of Figure 2(a), which possesses a single solution (a b a). None of its constraints is redundant and thus may not be unilaterally removed. However, the network is reducible, since, for instance, an equivalent network results from removing the constraint between variables B and C and, at the same time, deleting the pair (b b) from the constraint between A and C (see Figure 2(b)).

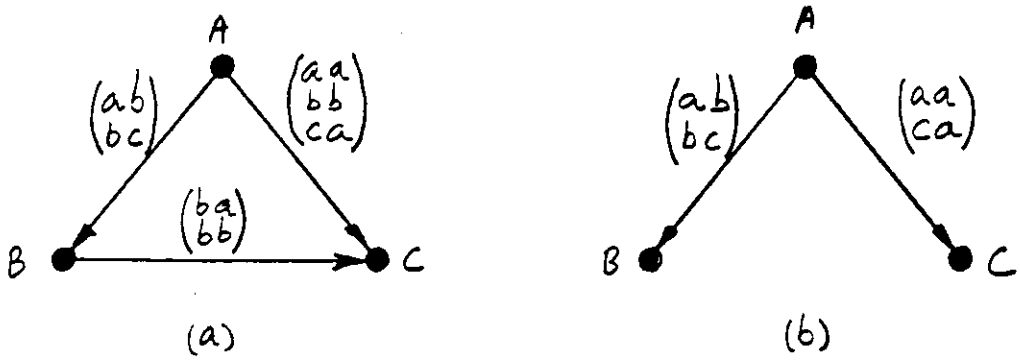

Figure 2 - A Reducible Network with No Redundant Constraints

This asymmetry in the relationships between reducibility and redundancy would be avoided if all non-universal constraints were *explicit*.

Definition A 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 the explicit constraint R^*_{ij} if, and only if, it is part of at least one solution. Thus, $R^*_{ij} = \Pi_{(X_i,X_j)}(\rho)$, where ρ is the relation represented by the network. A network of constraints all of which are explicit is a unique *minimal network of constraints**

[Montanari1974]. It is minimal in the sense that none of its constraints can be tightened (by deleting one or more pairs of values) with the result being an equivalent network. We now generalize this notion to the case where some of the constraints in the network are forced to remain universal.

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

In the sequel we use the term U-minimal network as a generic name for networks of constraints which possess some set of universal constraints (while all other constraints are explicit), and the notation U'-minimal network, U''-minimal network, etc., to refer to networks with specific sets (U', U'', etc.) of universal constraints.

Theorem 1: A U-minimal network of constraints (and its constraint graph) is irreducible if, and only if, none of its constraints is redundant.

Proof: If at least one constraint is redundant then, clearly, the network is reducible. Now suppose that none of the constraints are redundant. The removal of an non-redundant constraint from a network of constraints representing an n-ary relation ρ results in a network representing a relation which is strictly greater than ρ. This can potentially be compensated by restricting one or more of the other constraints. However, the non-universal constraints cannot be restricted because they are explicit, while restricting any universal constraint will result in the addition of an edge to the network, thus reducing the set of universal constraints. □

By definition, the U'-minimal network representation of a relation ρ is unique and can be obtained by projecting ρ on all of its two-dimensional subspaces, except for the subset U' of

* The notion of a *minimal network of constraints* should not be confused with that of a *minimal constraint graph.* In fact, they represent two opposing tendencies: A minimal constraint graph is associated with reducing the redundancy in the constraints (by eliminating redundant edges), while a minimal network of constraints is likely to represent the relation in a redundant way and to increase the number of edges in the constraint graph.

constraints which are left universal. Clearly, not all relations are representable by a U'-network, with an arbitrary set U' of constraints. However, if a relation ρ cannot be represented by a U'-network for some set U' of constraints, the U'-network generated by the above construction is the best approximation of ρ by such a network. The proof of the following result is similar to that of Theorem 3.1 in [Montanari1974].

Theorem 2: The network of constraints R' obtained by projecting ρ on all of its two-dimensional subspaces, except for a subset U' of universal constraints, represents an n-ary relation ρ' such that

$$\rho \subseteq \rho' .$$

Furthermore, no network R'' with a set U' of universal constraints exists, which represents an n-ary relation ρ'' such that

$$\rho \subseteq \rho'' \subset \rho' .$$

Corollary: A relation ρ which is representable by a U'-network of constraints, has a unique U'-minimal network representing it. □

Given a U'-network of constraints, R, representing some relation ρ, the corresponding U'-minimal network representation of ρ is the best representation for determining whether R is reducible or not.

Theorem 3: A U'-network of constraints (and its constraint graph) is reducible if, and only if, at least one of the constraints in the corresponding U'-minimal network is redundant.

Proof: This statement follows from the previous corollary and from Theorem 1, that shows that redundancy of some constraint is necessary and sufficient condition for reducibility of a U-minimal network. □

There is no unique irreducible U-minimal network of constraints (and, therefore, no unique irreducible constraint graph) representing a given relation. Moreover, the sets of universal constraints of all equivalent irreducible U-minimal networks may not have equal cardinalities. For example, the two networks of Figure 3 are both irreducible U-minimal network representations of the following relation:

A	B	C	D
a	a	a	a
a	b	a	c
a	b	b	b
b	b	a	c

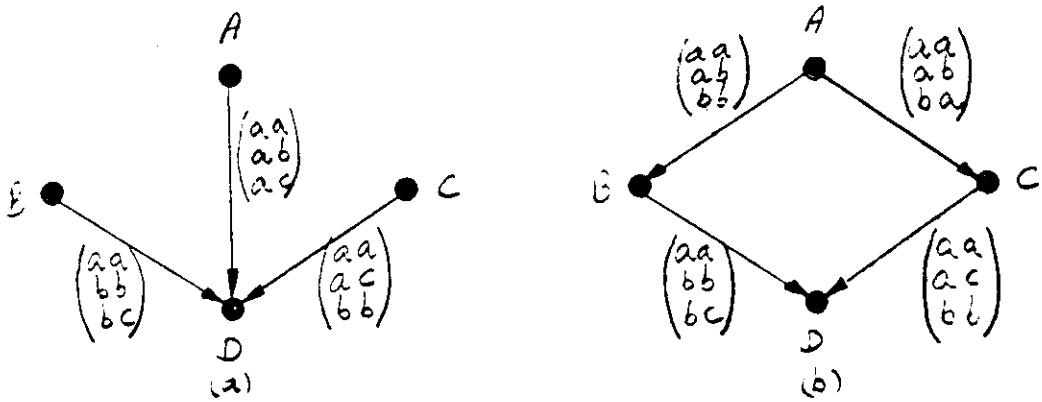

Figure 3 - Two Irreducible U-minimal Network Representations of the Same Relation.

To demonstrate the amount of redundancy that might be present in a naive CSP representation of a problem we consider the 5-queen problem. The task in this problem is to place 5 queens on a 5×5 chessboard so that no queen is on the same row, column, or diagonal as any other. A standard formulation of this problem as a binary CSP associates a variable with each row (e.g., variables A, B, C, D, and E), each of which may be assigned one of five values (say, a, b, c, d, and e) corresponding to the columns. There is a constraint between every pair of variables (for a total of ten constraints), consisting of all pairs of values which are not in direct conflict. The constraints of this network are listed in the appendix to this paper. The constraint graph of this formulation is the complete graph shown in Figure 4(a).

To identify irreducible U-minimal network representation we first calculate the minimal network of constraints by finding all the solutions and projecting on all variable-pairs. (The constraints of the minimal network are listed in the appendix. Its constraint graph is, of course, also the complete graph of Figure 4(a)). Then, considering one constraint at a time, each constraint found to be redundant is removed. The specific constraints removed in this way, and their number, depend on the order in which the constraints are considered. In all orderings tried this number was either four or five. The constraint graph of one irreducible U-minimal network representation of this problem is shown in Figure 4(b). This representation is much easier to solve than the original one since all cycles are cut by instantiating a single variable.

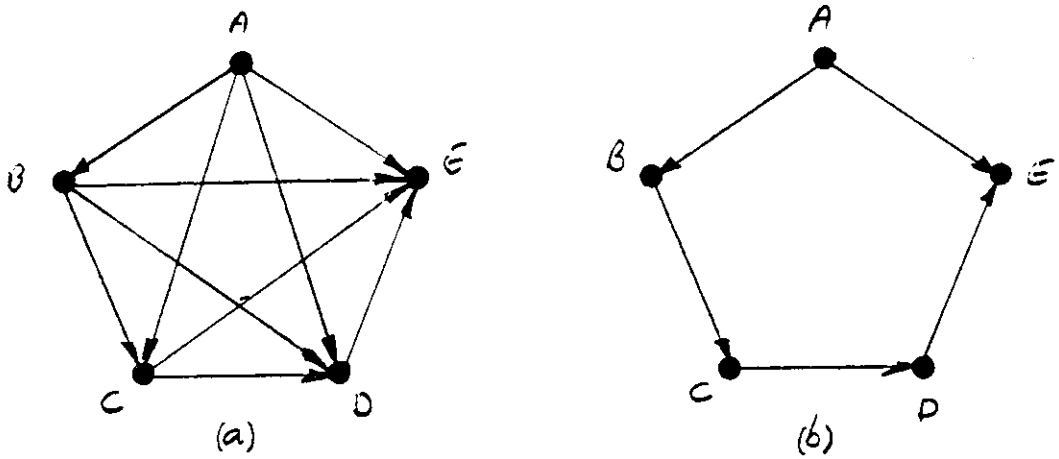

Figure 4 - The Constraint Graphs of Two Equivalent Representations of the 5-Queen Problem

The task of obtaining an irreducible U-minimal network of constraints equivalent to a given network, let alone finding one with a maximum number of universal constraints, is very difficult. It would require knowing the constraints induced on any pair of variables by the entire network (excluding the direct constraint between them), which is virtually equivalent to finding the set of all solutions. Therefore, we look for an approximation of an irreducible U-minimal network, i.e., a network which has as few constraints as possible and is still computable with local operations.

5. Approximating an Irreducible Network of Constraints

Montanary [Montanari1974] proposed to approximate the minimal networks of constraints of binary CSPs by what has become known as *path-consistent* networks.

Definition: The constraint R_{ij} between two variables X_i and X_j is said to be *path consistent* if for any pair of values (x_i, x_j) permitted by this constraint, and for any path of length m through nodes $(V_i=V_{k_0}, V_{k_1}, \ldots, V_{k_{m-1}}, V_{k_m}=V_j)$ there is a sequence of values $(z_1, z_2, \ldots, z_{m-1})$ such that

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

A network of constraint is path consistent if all of its constraints are path consistent.

Definition: A pair of values (x_i, x_j) for which the condition of path consistency is not satisfied by at least one path from node V_i to node V_j in the complete network R is called *path-illegal.*

A constraint which is not path consistent can be made one by simply removing all path-illegal pairs. The removal of these pairs does not change the problem because they are disallowed anyway by the *induced constraint* for the pair of variables in question. To determine whether a pair is path-illegal it is sufficient to consider only paths of length m=2 [Montanari1974]. Montanary and Mackworth [Mackworth1977] have proposed polynomial-time algorithms for achieving path consistency in a network of constraints.

Similar methods can be used for approximating irreducible networks of constraints. First, the notion of path consistency in networks can be generalized to cover U-minimal networks.

Definition: A network of constraints is said to be *U-path-consistent* if all of its constraints are path-consistent, except for a set U of universal constraints.

Given a U'-network of constraints, an equivalent U'-path-consistent network can be obtained in one of two primary ways:

1. By executing a complete path-consistency algorithm and then removing the constraints of the set U' from the resulting path-consistent network. This procedure is valid because it simply amounts to augmenting the constraints of the set U' by all the pairs that were removed from them by the algorithm for being path-illegal. All these pairs remain path-illegal after performing path-consistency and therefore may be added back to these constraints.

2. By using a modified path-consistency algorithm which does not touch the constraints of the set U' at all. Any of the path-consistency algorithms proposed in the literature can be trivially modified in this way.

The two procedures will not result, in general, in the same U'-path-consistent network. The non-universal constraints of the network obtained by the first procedure are either equal or contained by the corresponding constraints of the network obtained by the second procedure. Thus, the first procedure is likely to provide a better approximation to the U'-minimal network. The second procedure, on the other hand, has an obvious computational advantage. We will refer to a generic algorithm that takes a U-network and generates a U-path-consistent network (employing any of these methods) as the ***U*-PATH-CONSISTENCY** algorithm.

Now consider the task of identifying additional redundant constraints for the purpose of their removal. The same consistency checks which are involved in attaining path-consistency can be used in this task. This is based on the following observation: A pair of values (x_i, x_j) which is path-illegal may be *added* to the constraint R_{ij} (if it is not already part of that

constraint) without changing the problem. If, as a consequence of adding path-illegal pairs, the constraint becomes the universal constraint, we may then conclude that it is redundant and remove it. Let U_{ij} be the universal constraint for the pair of variables (X_i, X_j), and let L_{ij} the set of all pairs of U_{ij} which are path-illegal. The above observation is summarized in the following definition.

Definition: A constraint R_{ij} is said to be *path-redundant* if $L_{ij} \supseteq U_{ij} - R_{ij}$.

Clearly, if a constraint is path-redundant then it is redundant and may be removed. The converse, however, is not true, i.e., it possible for a constraint to be redundant but not path-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 be the universal constraint for (X_i, 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.

Definition: A U-network of constraints is said to be *path-irreducible* if none of its non-universal constraints is path-redundant.

A path-irreducible U-network equivalent to a given network of constraints may be obtained by considering all non-universal constraints in some order and removing from the network each constraint which is found to be path-redundant. One pass through the network is sufficient because a constraint which is not path-redundant in a network will never become one as a consequence of removing other constraints. This process is likely to discover more redundancies if the non-universal constraints are as explicit as possible. this is accomplished by per-

forming U-path-consistency first. An outline of such a procedure is given below:

```
procedure REDUCE
begin
  perform U-PATH-CONSISTENCY on R
  let Q = {(i,j) | R_ij ∈ R, i ≠ j}
  for all (i,j) ∈ Q
    if PATH-REDUNDANT((i,j)) let R = R − (i,j)
  end
end
```

The procedure **REDUCE** returns a network which is both path-irreducible and U-path consistent. Its complexity is $O(cnk^3)$, where c is the number of constraints in R. The particular network produced by this procedure depends on the order in which the constraints are considered (i.e., their order in Q) and, of course, is not guaranteed to be of a maximal cardinality set U.

REDUCE was applied to the 5-queen problem CSP representation discussed in Section 4, using several random orderings. The number of constraint that were removed from the network was either 3 or 4 in all cases (compared with 4 to 5 when true irreducible U-minimal networks are found). Constraint graphs of two irreducible U-path consistent networks for this problem are shown in Figure 5 (the constraints themselves are the same as those of the minimal network appearing in the appendix). Notice that in order to cut all the cycles in the network of Figure 5(a), a minimum of two variables must be instantiated, but it take the instantiation of but a single variable (A or B) to cut all the cycles in the network of Figure 5(b).

The procedure **REDUCE** can be thought of as a greedy algorithm which operates on a search space defined as follows: Each state is a network of constraints equivalent to the original network, and a legal operator is the removal of one path-redundant constraint. Each leaf node in this space is a path-irreducible U-network of constraints. Our objective is to find a leaf node having a maximal set U, i.e., associated with the longest solution path.

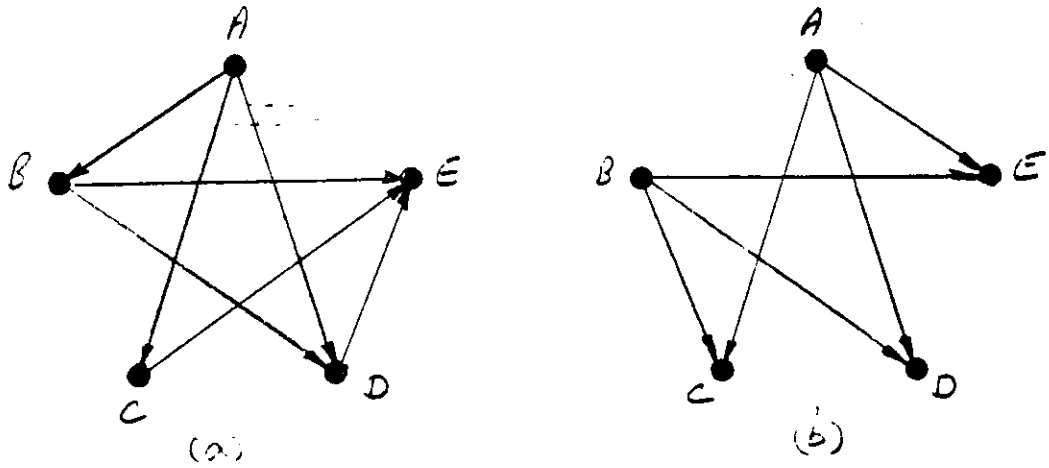

Figure 5 - Two Irreducible *U*-Path-Consistent Networks for the 5-Queen Problem

At each step the algorithm elects one out of the set of all path-redundant constraints for removal from the network. This may cause some (or all) of the other candidates to become non-path-redundant. therefore, heuristically, we would like to accord priority to those constraints whose removal leaves as many path-redundant constraints as possible. Several types of heuristic information is available for guiding the choice in each step.

First, we note that the removal of a constraint can affect the path-redundancy status of only constraints which are adjacent to it, i.e., share a variable with it. (This is so because only paths of length $m = 2$ are used to determine the path-redundancy of a constraint.) Two constraints which do not share a common variable are said to be *independent* of one another. Thus, a reasonable heuristic evaluation function would be the number of constraints in the candidate set which are independent of the constraint. A more precise approach would be to find the exact number of constraints that will remain path-redundant after the removal of each constraint.

Observe that there is no guarantee that all the constraints which remain path-redundant will eventually be removed by **REDUCE**. However, such would the case be if these constraints were independent on each other. Thus, another possibility is to elect for removal a constraint that has the largest set of path-redundant constraints which are both independent on it and

independent on each other. Unfortunately, this is equivalent to finding a maximal independent set of constraints, which is an NP-complete problem. An additional advantage of finding independent sets of path-redundant constraints is that they all may be removed simultaneously, thus saving considerable amount of calculation.

If an optimal solution is desired, i.e., a network with a maximal set U, it can be found by performing less then an exhaustive search. Algorithm A* can be used to find the longest solution path on the search space described above. The evaluation function of each node will be $f = g+h$ where g is the length of the path from the initial state to the current state, and h is the number of constraints which are still path-redundant. This heuristic function is admissible, since it is an upper bound on the longest path to the goal, and is even monotone. Therefore, A* with this evaluation function, will find the best solution and will never expand a node more then once. The number of independent constraints among the removable constraints could be used to update a lower bound of the longest solution path which, in turn, can be used to prune the search tree.

6. Directional Path-Redundancy

In Section 2 we explained that reducing the constraint graph will benefit backtrack algorithms that utilize the constraint graph for *backjumping*, because of the increased number of backjumping opportunities. At the same time, reducing the constraint graph involves the relaxation of certain constraints (those that are found to be redundant), which has the effect of increasing the search-space the algorithm must explore. This is to be expected, since the reduction process does exactly the opposite of what path-consistency operation, designed to reduces the search-space, does. Therefore, no categorical claims can be made on the overall effect the reduction operation might have on the performance of backjump algorithms, and these must be assessed experimentally.

It is possible, however, to guarantee that the search-space explored by the algorithm will not change at all, by using a restricted form of the reduction operation. The basic idea is to tie the definition of path-redundancy to the specific ordering of the variables that will be used later by the backjumping algorithm. Let $d = X_1, \ldots, X_n$ be an ordering of the variables.

Definition: A constraint R_{ij}, $i < j$, 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 $(X_i - X_k - X_j)$, for $k > j$.

The procedure **PATH-REDUNDANT** will be made directional by simply replacing line 3 with

for all k > i, j .

If none of the non-universal constraints of a U-network is directional-path-redundant with respect to some order d, the network is said to be *directional-path-irreducible* with respect to that order. The procedure **REDUCE**, using directional-path-redundancy, will turn any U-network into a directional-path-irreducible network.

If the reduction of the network uses only directional-path-redundancy with respect to an order which is precisely the *reverse* of the order used by a backtracking algorithm, the search space explored by the algorithm will not be affected. This is so because, in this case, by the time the backtrack algorithm is in a position to use a constraint which has been removed by the reduction procedure, it has already available to it all the information contained in that constraint. To clarify this point, consider a network whose constraint graph is given in Figure 6: 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 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. On the

Figure 6 - An Ordered Constraint Graph

other hand, 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 place.

Any ordering of the constraints used by **REDUCE** to remove directional-path-redundancy with respect to some ordering d of the variables will lead to a directional path-irreducible network. However, it is clearly best to tie the ordering of the constraints to d so that a constraint R_{ij}, $j>i$ will always be checked (and removed if directional-path-redundant) before any constraint R_{kl} $l>k$, if $j<l$. This order guarantees that removal of one constraint will not interfere with the removal of any other. To see this, refer back to the network of Figure 7, and assume 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. However, if R_{23} is removed first then this might cause R_{34} to be no longer path-redundant with respect to X_2.

A consequence of this is that all the constraints that are directional-path-redundant with respect to some direction d may be removed simultaneously, with the result being the smallest possible network (i.e., has a minimal number of arcs in its constraint graph) which is directional-path-irreducible with respect to that direction.

The notion of directional-path-redundancy is related to that of *directional-path-consistency* developed in [Dechter1985]. A network is said to be directional-path-consistent with respect to some ordering d of the variables if any constraint R_{ij}, $i<j$, is consistent with respect to all paths $(i-k-j)$ for $k>j$. An algorithm for achieving directional-path-consistency of a given network is a straightforward variation of any path-consistency algorithm. It is easy to see that all constraints *added* to the network by a directional-path-consistency algorithm are directional-path-redundant with respect to the same direction. Thus, all such constraints are guaranteed to be removed by the optimal reduction procedure discussed above.

7. Conclusions

This paper argues in favor of manipulating the representation of constraint satisfaction problems in order for their constraint graphs to show as many independencies among the variables as possible. This is accomplished by identifying redundancies in the constraint network which allow the removal of constraints. We develop a theoretical basis for this operation by defining the notion of U-minimal network of constraints, and propose an approximation procedure for discovering network that are irreducible.

Our conjecture (which is partially supported by the 5-queen example) is that many "naive" formulations of CSPs are not optimal from the point of view of the density of the constraint graph and thus can be improved by the methods suggested in this paper. Other classes of problems need to be investigated in order to draw firmer conclusions.

The practical utility of the ideas presented has yet to be tested. An investigation is currently under way to study the best ways these ideas can be incorporated into existing backtracking algorithms and the expected improvements in their performance.

APPENDIX

The original network of constraints for the 5-queen problem:

((A B) (a c) (a d) (a e) (b d) (b e) (c a) (c e) (d a) (d b) (e a) (e b) (e c))
((A C) (a b) (a d) (a e) (b a) (b c) (b e) (c b) (c d) (d a) (d c) (d e) (e a) (e b) (e d))
((A D) (a b) (a c) (a e) (b a) (b c) (b d) (c a) (c b) (c d) (c e) (d b) (d c) (d e) (e a) (e c) (e d))
((A E) (a b) (a c) (a d) (b a) (b c) (b d) (b e) (c a) (c b) (c d) (c e) (d a) (d b) (d c) (d e) (e b) (e c) (e d))
((B C) (a c) (a d) (a e) (b d) (b e) (c a) (c e) (d a) (d b) (e a) (e b) (e c))
((B D) (a b) (a d) (a e) (b a) (b c) (b e) (c b) (c d) (d a) (d c) (d e) (e a) (e b) (e d))
((B E) (a b) (a c) (a e) (b a) (b c) (b d) (c a) (c b) (c d) (c e) (d b) (d c) (d e) (e a) (e c) (e d))
((C D) (a c) (a d) (a e) (b d) (b e) (c a) (c e) (d a) (d b) (e a) (e b) (e c))
((C E) (a b) (a d) (a e) (b a) (b c) (b e) (c b) (c d) (d a) (d c) (d e) (e a) (e b) (e d))
((D E) (a c) (a d) (a e) (b d) (b e) (c a) (c e) (d a) (d b) (e a) (e b) (e c))

The minimal network of constraints for the 5-queen problem:

((A B) (a c) (a d) (b d) (b e) (c a) (c e) (d a) (d b) (e b) (e c))
((A C) (a b) (a e) (b a) (b c) (c b) (c d) (d c) (d e) (e a) (e d))
((A D) (a b) (a e) (b a) (b c) (c b) (c d) (d c) (d e) (e a) (e d))
((A E) (a c) (a d) (b d) (b e) (c a) (c e) (d a) (d b) (e b) (e c))
((B C) (a c) (a d) (b d) (b e) (c a) (c e) (d a) (d b) (e b) (e c))
((B D) (a b) (a e) (b a) (b c) (c b) (c d) (d c) (d e) (e a) (e d))
((B E) (a b) (a e) (b a) (b c) (c b) (c d) (d c) (d e) (e a) (e d))
((C D) (a c) (a d) (b d) (b e) (c a) (c e) (d a) (d b) (e b) (e c))
((C E) (a b) (a e) (b a) (b c) (c b) (c d) (d c) (d e) (e a) (e d))
((D E) (a c) (a d) (b d) (b e) (c a) (c e) (d a) (d b) (e b) (e c))

References

[Dechter1985] 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: August 1985, pp. 1066-1072.

[Dechter1986a] 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.

[Dechter1986b] 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.

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

[Maier1983] Maier, D., *The theory of relational databases*, Rockville, Maryland: Computer science press, 1983.

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

[Pearl1986] Pearl, J. and A. Paz, "On the Logic of Representing Dependencies by Graphs," in *Proceedings Sixth Canadian Conference on Artificial Intelligence,* Montreal, Quebec, Canada: May 1986, pp. 94-98.