On the Inherent Level of Local Consistency in Constraint Networks

Peter van Beek
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
vanbeek@cs.ualberta.ca

Abstract

We present a new property called *constraint looseness* and show how it can be used to estimate the level of local consistency of a binary constraint network. Specifically, we present a relationship between the looseness of the constraints, the size of the domains, and the inherent level of local consistency of a constraint network. The results we present are useful in two ways. First, a common method for finding solutions to a constraint network is to first preprocess the network by enforcing local consistency conditions, and then perform a backtracking search. Here, our results can be used in deciding which low-order local consistency techniques will *not* change a given constraint network and thus are not useful for preprocessing the network. Second, much previous work has identified conditions for when a certain level of local consistency is sufficient to guarantee a network is backtrack-free. Here, our results can be used in deciding which local consistency conditions, if any, still need to be enforced to achieve the specified level of local consistency. As well, we use the looseness property to develop an algorithm that can sometimes find an ordering of the variables such that a network is backtrack-free.

Introduction

Constraint networks are a simple representation and reasoning framework. A problem is represented as a set of variables, a domain of values for each variable, and a set of constraints between the variables. A central reasoning task is then to find an instantiation of the variables that satisfies the constraints. Examples of tasks that can be formulated as constraint networks include graph coloring (Montanari 1974), scene labeling (Waltz 1975), natural language parsing (Maruyama 1990), temporal reasoning (Allen 1983), and answering conjunctive queries in relational databases.

In general, what makes constraint networks hard to solve is that they can contain many local inconsistencies. A local inconsistency is a consistent instantiation of $k-1$ of the variables that cannot be extended to a *kth* variable and so cannot be part of any global solution. If we are using a backtracking search to find a solution, such an inconsistency can lead to a dead end in the search. This insight has led to the definition of conditions that characterize the level of local consistency of a network (Freuder 1985; Mackworth 1977; Montanari 1974) and to the development of algorithms for enforcing local consistency conditions by removing local inconsistencies (e.g., (Cooper 1989; Dechter & Pearl 1988; Freuder 1978; Mackworth 1977; Montanari 1974; Waltz 1975)). However, the definitions, or necessary and sufficient conditions, for all but low-order local consistency are expensive to verify or enforce as the optimal algorithms are $O(n^k)$, where k is the level of local consistency (Cooper 1989; Seidel 1983).

In this paper, we present a simple, sufficient condition, based on the size of the domains of the variables and on a new property called *constraint looseness*, that gives a lower bound on the the inherent level of local consistency of a binary constraint network. The bound is tight for some constraint networks but not for others. Specifically, in any constraint network where the domains are of size d or less, and the constraints have looseness of m or greater, the network is strongly $(\lceil d/(d-m) \rceil)$-consistent[1]. Informally, a constraint network is strongly k-consistent if a solution can always be found for any subnetwork of size k in a backtrack-free manner. The parameter m can be viewed as a lower bound on the number of instantiations of a variable that satisfy the constraints. We also use the looseness property to develop an algorithm that can sometimes find an ordering of the variables such that all solutions of a network can be found in a backtrack-free manner.

The condition we present is useful in two ways. First, a common method for finding solutions to a constraint network is to first preprocess the network by enforcing local consistency conditions, and then perform a backtracking search. The preprocessing step can reduce the number of dead ends reached by the backtracking algorithm in the search for a solution. With a similar aim, local consistency techniques can be interleaved with backtracking search. The effectiveness of using

[1] $\lceil x \rceil$, the ceiling of x, is the smallest integer greater than or equal to x.

local consistency techniques in these two ways has been studied empirically (e.g., (Dechter & Meiri 1989; Gaschnig 1978; Ginsberg *et al.* 1990; Haralick & Elliott 1980; Prosser 1993)). In this setting, our results can be used in deciding which low-order local consistency techniques will *not* change the network and thus are not useful for processing a given constraint network. For example, we use our results to show that the n-queens problem, a widely used test-bed for comparing backtracking algorithms, has a high level of inherent local consistency. As a consequence, it is generally fruitless to preprocess such a network.

Second, much previous work has identified conditions for when a certain level of local consistency is sufficient to guarantee a solution can be found in a backtrack-free manner (e.g., (Dechter 1992; Dechter & Pearl 1988; Freuder 1982; 1985; Montanari 1974; van Beek 1992)). These conditions are important in applications where constraint networks are used for knowledge base maintenance and there will be many queries against the knowledge base. Here, the cost of preprocessing will be amortized over the many queries. In this setting, our results can be used in deciding which local consistency conditions, if any, still need to be enforced to achieve the specified level of local consistency.

Background

We begin with some needed definitions.

Definition 1 (binary constraint networks; Montanari (1974)) *A* binary constraint network *consists of a set X of n variables $\{x_1, x_2, \ldots, x_n\}$, a domain D_i of possible values for each variable, and a set of binary constraints between variables. A* binary constraint *or relation, R_{ij}, between variables x_i and x_j, is any subset of the product of their domains (i.e., $R_{ij} \subseteq D_i \times D_j$). An* instantiation *of the variables in X is an n-tuple $(X_1, X_2, \ldots, X_n)$, representing an assignment of $X_i \in D_i$ to x_i. A* consistent instantiation *of a network is an instantiation of the variables such that the constraints between variables are satisfied. A consistent instantiation is also called a* solution.

Mackworth (1977; 1987) defines three properties of networks that characterize local consistency of networks: *node*, *arc*, and *path consistency*. Freuder (1978) generalizes this to k-consistency.

Definition 2 (strong k-consistency; Freuder (1978; 1982)) *A network is* k-consistent *if and only if given any instantiation of any $k-1$ variables satisfying all the direct relations among those variables, there exists an instantiation of any kth variable such that the k values taken together satisfy all the relations among the k variables. A network is* strongly k-consistent *if and only if it is j-consistent for all $j \leq k$.*

Node, arc, and path consistency correspond to strongly one-, two-, and three-consistent, respectively. A strongly n-consistent network is called *globally consistent*. Globally consistent networks have the property that any consistent instantiation of a subset of the variables can be extended to a consistent instantiation of all the variables without backtracking (Dechter 1992).

Following Montanari (1974), a binary relation R_{ij} between variables x_i and x_j is represented as a (0,1)-matrix with $|D_i|$ rows and $|D_j|$ columns by imposing an ordering on the domains of the variables. A zero entry at row a, column b means that the pair consisting of the ath element of D_i and the bth element of D_j is not permitted; a one entry means the pair is permitted. A concept central to this paper is the looseness of constraints.

Definition 3 (m-loose) *A binary constraint is* m-loose *if every row and every column of the (0,1)-matrix that defines the constraint has at least m ones, where $0 \leq m \leq |D| - 1$. A binary constraint network is m-loose if all its binary constraints are m-loose.*

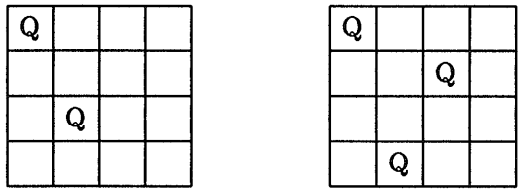


Figure 1: (a) not 3-consistent; (b) not 4-consistent

Example 1. We illustrate some of the definitions using the well-known n-queens problem. The problem is to find all ways to place n-queens on an $n \times n$ chess board, one queen per column, so that each pair of queens does not attack each other. One possible constraint network formulation of the problem is as follows: there is a variable for each column of the chess board, $x_1, \ldots, x_n$; the domains of the variables are the possible row positions, $D_i = \{1, \ldots, n\}$; and the binary constraints are that two queens should not attack each other. The (0,1)-matrix representation of the constraints between two variables x_i and x_j is given by,

$$R_{ij,ab} = \begin{cases} 1 & \text{if } a \neq b \wedge |a-b| \neq |i-j| \\ 0 & \text{otherwise,} \end{cases}$$

for $a, b = 1, \ldots, n$.

For example, consider the constraint network for the 4-queens problem. The constraint R_{12} between x_1 and x_2 is given by,

$$R_{12} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}.$$

Entry $R_{12,43}$ is 0, which states that putting a queen in column 1, row 4 and a queen in column 2, row 3

is not allowed by the constraint since the queens attack each other. It can be seen that the network for the 4-queens problem is 2-consistent since, given that we have placed a single queen on the board, we can always place a second queen such that the queens do not attack each other. However, the network is not 3-consistent. For example, given the consistent placement of two queens shown in Figure 1a, there is no way to place a queen in the third column that is consistent with the previously placed queens. Similarly the network is not 4-consistent (see Figure 1b). Finally, every row and every column of the (0,1)-matrices that define the constraints has at least 1 one. Hence, the network is 1-loose.

A Sufficient Condition for Local Consistency

In this section, we present a simple condition that estimates the inherent level of strong k-consistency of a binary constraint network. The condition is a sufficient but not necessary condition for local consistency.

It is known that some classes of constraint networks already possess a certain level of local consistency and therefore algorithms that enforce this level of local consistency will have no effect on these networks. For example, Nadel (1989) observes that an arc consistency algorithm never changes a constraint network formulation of the n-queens problem, for $n > 3$. Dechter (1992) observes that constraint networks that arise from the graph k-coloring problem are inherently strongly k-consistent. The following theorem characterizes what it is about the structure of the constraints in these networks that makes these statements true.

Theorem 1 *If a binary constraint network, R, is m-loose and all domains are of size $|D|$ or less, then the network is strongly $\left(\left\lceil \frac{|D|}{|D|-m} \right\rceil\right)$-consistent.*

Proof. We show that the network is k-consistent for all $k \leq \lceil |D|/(|D|-m) \rceil$. Suppose that variables $x_1, \ldots, x_{k-1}$ can be consistently instantiated with values $X_1, \ldots, X_{k-1}$. To show that the network is k-consistent, we must show that there exists at least one instantiation X_k of variable x_k that satisfies all the constraints,

$$(X_i, X_k) \in R_{ik} \qquad i = 1, \ldots, k-1$$

simultaneously. We do so as follows. The instantiations $X_1, \ldots, X_{k-1}$ restrict the allowed instantiations of x_k. Let v_i be the (0,1)-vector given by row X_i of the (0,1)-matrix R_{ik}, $i = 1, \ldots, k-1$. Let $\text{pos}(v_i)$ be the positions of the zeros in vector v_i. The zero entries in the v_i are the forbidden instantiations of x_k, given the instantiations $X_1, \ldots, X_{k-1}$. No consistent instantiation of x_k exists if and only if $\text{pos}(v_1) \cup \cdots \cup \text{pos}(v_{k-1}) = \{1, \ldots, |D|\}$. Now, the key to the proof is that all the v_i contain at least m ones. In other words, each v_i contains at most $|D| - m$ zeros. Thus, if

$$(k-1)(|D|-m) < |D|,$$

it cannot be the case that $\text{pos}(v_1) \cup \cdots \cup \text{pos}(v_{k-1}) = \{1, \ldots, |D|\}$. (To see that this is true, consider the "worst case" where the positions of the zeros in any vector do not overlap with those of any other vector. That is, $\text{pos}(v_i) \cap \text{pos}(v_j) = \emptyset$, $i \neq j$.) Thus, if

$$k \leq \left\lceil \frac{|D|}{|D|-m} \right\rceil,$$

all the constraints must have a non-zero entry in common and there exists at least one instantiation of x_k that satisfies all the constraints simultaneously. Hence, the network is k-consistent. □

Theorem 1 always specifies a level of local consistency that is less than or equal to the actual level of inherent local consistency of a constraint network. That is, the theorem provides a lower bound. Graph coloring problems provide examples where the theorem is exact, whereas n-queens problems provide examples where the theorem underestimates the true level of local consistency.

Example 2. Consider again the well-known n-queens problem discussed in Example 1. The problem is of historical interest but also of theoretical interest due to its importance as a test problem in empirical evaluations of backtracking algorithms and heuristic repair schemes for finding solutions to constraint networks (e.g., (Gaschnig 1978; Haralick & Elliott 1980; Minton *et al.* 1990; Nadel 1989)). For n-queens networks, each row and column of the constraints has $|D|-3 \leq m \leq |D|-1$ ones, where $|D| = n$. Hence, Theorem 1 predicts that n-queens networks are inherently strongly ($\lceil n/3 \rceil$)-consistent. Thus, an n-queens constraint network is inherently arc-consistent for $n \geq 4$, inherently path consistent for $n \geq 7$, and so on, and we can predict where it is fruitless to apply a low order consistency algorithm in an attempt to simplify the network (see Table 1). The actual level of inherent consistency is $\lfloor n/2 \rfloor$ for $n \geq 7$. Thus, for the n-queens problem, the theorem underestimates the true level of local consistency.

Table 1: Predicted ($\lceil n/3 \rceil$) and actual ($\lfloor n/2 \rfloor$, for $n \geq 7$) level of strong local consistency for n-queens networks

n	4	5	6	7	8	9	10	11	12
pred.	2	2	2	3	3	3	4	4	4
actual	2	2	2	3	4	4	5	5	6

The reason Theorem 1 is not exact in general and, in particular, for n-queens networks, is that the proof of the theorem considers the "worst case" where the positions of the zeros in any row of the constraints

$R_{ik}, i = 1, \ldots, k-1$, do not overlap with those of any other row. For n-queens networks, the positions of some of the zeros do overlap. However, given only the looseness of the constraints and the size of the domains, Theorem 1 gives as strong an estimation of the inherent level of local consistency as possible as examples can be given where the theorem is exact.

Example 3. Graph k-colorability provides examples where Theorem 1 is exact in its estimation of the inherent level of strong k-consistency. The constraint network formulation of graph coloring is straightforward: there is a variable for each node in the graph; the domains of the variables are the possible colors, $D = \{1, \ldots, k\}$; and the binary constraints are that two adjacent nodes must be assigned different colors. As Dechter (1992) states, graph coloring networks are inherently strongly k-consistent but are not guaranteed to be strongly $(k+1)$-consistent. Each row and column of the constraints has $m = |D| - 1$ ones, where $|D| = k$. Hence, Theorem 1 predicts that graph k-colorability networks are inherently strongly k-consistent.

Example 4. We can also construct examples, for all $m < |D| - 1$, where Theorem 1 is exact. For example, consider the network where, $n = 5$, the domains are $D = \{1, \ldots, 5\}$, and the binary constraints are given by,

$$R_{ij} = \begin{bmatrix} 0\,1\,1\,1\,1 \\ 0\,0\,1\,1\,1 \\ 1\,0\,0\,1\,1 \\ 1\,1\,0\,0\,1 \\ 1\,1\,1\,0\,0 \end{bmatrix}, \; 1 \leq i < j \leq n,$$

and $R_{ji} = R_{ij}^T$, for $j < i$. The network is 3-loose and therefore strongly 3-consistent by Theorem 1. This is exact, as the network is not 4-consistent.

We conclude this section with some discussion on what Theorem 1 contributes to our intuitions about hard classes of problems (in the spirit of, for example, (Cheeseman, Kanefsky, & Taylor 1991; Williams & Hogg 1992)). Hard constraint networks are instances which give rise to search spaces with many dead ends. The hardest networks are those where many dead ends occur deep in the search tree. Dead ends, of course, correspond to partial solutions that cannot be extended to full solutions. Thus, networks where the constraints are loose are good candidates to be hard problems since loose networks have a high level of inherent strong consistency and strong k-consistency means that all partial solutions are of at least size k.

Computational experiments we performed on random problems provide evidence that loose networks can be hard. Random problems were generated with $n = 50$, $|D| = 5, \ldots 10$, and $p, q = 1, \ldots, 100$, where $p/100$ is the probability that there is a binary constraint between two variables, and $q/100$ is the probability that a pair in the Cartesian product of the domains is in the constraint. The time to find one solution was measured. In the experiments we discovered that, given that the number of variables and the domain size were fixed, the hardest problems were found when the constraints were as loose as possible without degenerating into the trivial constraint where all tuples are allowed. That networks with loose constraints would turn out to be the hardest of these random problems is somewhat counter-intuitive, as individually the constraints are easy to satisfy. These experimental results run counter to Tsang's (1993, p.50) intuition that a single solution of a loosely constrained problem "can easily be found by simple backtracking, hence such problems are easy," and that tightly constrained problems are "harder compared with loose problems." As well, these hard loosely-constrained problems are not amenable to preprocessing by low-order local consistency algorithms, since, as Theorem 1 states, they possess a high level of inherent local consistency. This runs counter to Williams and Hogg's (1992, p.476) speculation that preprocessing will have the most dramatic effect in the region where the problems are the hardest.

Backtrack-free Networks

Given an ordering of the variables in a constraint network, backtracking search works by successively instantiating the next variable in the ordering, and backtracking to try different instantiations for previous variables when no consistent instantiation can be given to the current variable. Previous work has identified conditions for when a certain level of local consistency is sufficient to ensure a solution can be found in a backtrack-free manner (e.g., (Dechter 1992; Dechter & Pearl 1988; Freuder 1982; 1985; Montanari 1974; van Beek 1992)). Sometimes the level of inherent strong k-consistency guaranteed by Theorem 1 is sufficient, in conjunction with these previously derived conditions, to guarantee that the network is globally consistent and therefore a solution can be found in a backtrack-free manner. Otherwise, the estimate provided by the theorem gives a starting point for applying local consistency algorithms.

In this section, we use constraint looseness to identify new classes of backtrack-free networks. First, we give a condition for a network to be inherently globally consistent. Second, we give a condition, based on a directional version of the looseness property, for an ordering to be backtrack-free. We also give an efficient algorithm for finding an ordering that satisfies the condition, should it exist.

We begin with a corollary of Theorem 1.

Corollary 1 *If a binary constraint network, R, is m-loose, all domains are of size $|D|$ or less, and $m > \frac{n-2}{n-1}|D|$, the network is globally consistent.*

Proof. By Theorem 1, the network is strongly n-consistent if $\lceil |D|/(|D| - m) \rceil \geq n$. This is equivalent to, $|D|/(|D| - m) > n - 1$ and rearranging for m gives the result. □

As one example, consider a constraint network with $n = 5$ variables that has domains of at most size $|D| = 10$ and constraints that are 8-loose. The network is globally consistent and, as a consequence, a solution can be found in a backtrack-free manner. Another example is networks with $n = 5$, domain sizes of $|D| = 5$, and constraints that are 4-loose.

Global consistency implies that all orderings of the variables are backtrack-free orderings. Sometimes, however, there exists a backtrack-free ordering when only much weaker local consistency conditions hold. Freuder (1982) identifies a relationship between the *width* of an ordering of the variables and the level of local consistency sufficient to ensure an ordering is backtrack-free.

Definition 4 (width; Freuder (1982)) *Let* $o = (x_1, \ldots, x_n)$ *be an ordering of the variables in a binary constraint network. The* width of a variable, x_i, *is the number of binary constraints between* x_i *and variables previous to* x_i *in the ordering. The* width of an ordering *is the maximum width of all variables.*

Theorem 2 (Freuder (1982)) *An ordering of the variables in a binary constraint network is backtrack-free if the level of strong k-consistency of the network is greater than the width of the ordering.*

Dechter and Pearl (1988) define a weaker version of k-consistency, called directional k-consistency, and show that Theorem 2 still holds. Both versions of k-consistency are, in general, expensive to verify, however. Dechter and Pearl also give an algorithm, called adaptive consistency, that does not enforce a uniform level of local consistency throughout the network but, rather, enforces the needed level of local consistency as determined on a variable by variable basis. We adapt these two insights, directionality and not requiring uniform levels of local consistency, to a condition for an ordering to be backtrack-free.

Definition 5 (directionally m-loose) *A binary constraint is* directionally m-loose *if every row of the (0,1)-matrix that defines the constraint has at least m ones, where* $0 \leq m \leq |D| - 1$.

Theorem 3 *An ordering of the variables,* $o = (x_1, \ldots, x_n)$, *in a binary constraint network, R, is backtrack-free if* $\left\lceil \frac{|D|}{|D|-m_j} \right\rceil > w_j$, $1 \leq j \leq n$, *where* w_j *is the width of variable* x_j *in the ordering, and* m_j *is the minimum of the directional looseness of the (non-trivial) constraints* $R_{ij}, 1 \leq i < j$.

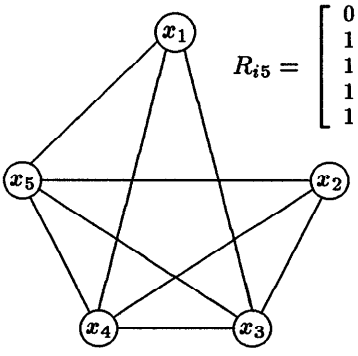
Proof. Similar to the proof of Theorem 1. □

A straightforward algorithm for finding such a backtrack-free ordering of the variables, should it exist, is given below.

FINDORDER(R, n)
1. $I \leftarrow \{1, 2, \ldots, n\}$
2. **for** $p \leftarrow n$ **downto** 1 **do**
3. find a $j \in I$ such that, for each R_{ij}, $i \in I$ and $i \neq j$, $\lceil |D|/(|D| - m_{ij}) \rceil > w_j$, where w_j is the number of constraints R_{ij}, $i \in I$ and $i \neq j$, and m_{ij} is the directional m-looseness of R_{ij} (if no such j exists, report failure and halt)
4. put variable x_j at position p in the ordering
5. $I \leftarrow I - \{j\}$

Example 5. Consider the network in Figure 2. The network is 2-consistent, but not 3-consistent and not 4-consistent. Freuder (1982), in connection with Theorem 2, gives an algorithm for finding an ordering which has the minimum width of all orderings of the network. Assuming that the algorithms break ties by choosing the variable with the lowest index, the minimal width ordering found is $(x_5, x_4, x_3, x_2, x_1)$, which has width 3. Thus, the condition of Theorem 2 does not hold. In fact, this ordering is not backtrack-free. For example, the partial solution $x_5 \leftarrow 1$, $x_4 \leftarrow 3$, and $x_3 \leftarrow 5$ is a dead end, as there is no instantiation for x_2. The ordering found by procedure FINDORDER is $(x_4, x_3, x_2, x_1, x_5)$, which has width 4. It can be verified that the condition of Theorem 3 holds. For example, w_1, the width at variable x_1, is 2, and the constraints R_{41} and R_{31} are both 3-loose, so $\left\lceil \frac{|D|}{|D|-m_1} \right\rceil = 3 > w_1 = 2$. Therefore all solutions of the network can be found with no backtracking along this ordering.

$$R_{ij} = \begin{bmatrix} 0&0&1&1&1 \\ 1&0&0&1&1 \\ 1&1&0&0&1 \\ 1&1&1&0&0 \\ 1&1&1&1&0 \end{bmatrix}, \; i = 1, 2; \; j = 3, 4$$



$$R_{i5} = \begin{bmatrix} 0&1&1&1&1 \\ 1&0&1&1&1 \\ 1&1&0&1&1 \\ 1&1&1&0&1 \\ 1&1&1&1&0 \end{bmatrix}, \; i = 1, \ldots, 4$$

$$R_{34} = \begin{bmatrix} 0&0&1&1&0 \\ 0&1&1&0&1 \\ 1&1&0&1&1 \\ 1&0&1&1&0 \\ 0&1&1&0&0 \end{bmatrix}$$

$$R_{ji} = R_{ij}^T, \; j < i$$

Figure 2: Constraint network for which a backtrack-free ordering exists

Conclusions and Future Work

Local consistency has proven to be an important concept in the theory and practice of constraint networks. However, the definitions, or necessary and sufficient

conditions, for all but low-order local consistency are expensive to verify or enforce. We presented a sufficient condition for local consistency, based on a new property called constraint looseness, that is straightforward and inexpensive to determine. The condition can be used to estimate the level of strong local consistency of a network. This in turn can be used in (i) deciding whether it would be useful to preprocess the network before a backtracking search, and (ii) deciding which local consistency conditions, if any, still need to be enforced if we want to ensure that a solution can be found in a backtrack-free manner. Finally, the looseness property was used to identify new classes of "easy" constraint networks.

A property of constraints proposed by Nudel (1983) which is related to constraint looseness counts the number of ones in the entire constraint. Nudel uses this count, called a compatibility count, in an effective variable ordering heuristic for backtracking search. We plan to examine whether m-looseness can be used to develop even more effective domain and variable ordering heuristics. We also plan to examine how the looseness property can be used to improve the average case efficiency of local consistency algorithms. The idea is to predict whether small subnetworks already possess some specified level of local consistency, thus potentially avoiding the computations needed to enforce local consistency on those parts of the network.

Acknowledgements. Financial assistance was received from the Natural Sciences and Engineering Research Council of Canada.

References

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Comm. ACM* 26:832–843.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, 331–337.

Cooper, M. C. 1989. An optimal k-consistency algorithm. *Artificial Intelligence* 41:89–95.

Dechter, R., and Meiri, I. 1989. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *Proc. of the Eleventh International Joint Conference on Artificial Intelligence*, 271–277.

Dechter, R., and Pearl, J. 1988. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence* 34:1–38.

Dechter, R. 1992. From local to global consistency. *Artificial Intelligence* 55:87–107.

Freuder, E. C. 1978. Synthesizing constraint expressions. *Comm. ACM* 21:958–966.

Freuder, E. C. 1982. A sufficient condition for backtrack-free search. *J. ACM* 29:24–32.

Freuder, E. C. 1985. A sufficient condition for backtrack-bounded search. *J. ACM* 32:755–761.

Gaschnig, J. 1978. Experimental case studies of backtrack vs. waltz-type vs. new algorithms for satisficing assignment problems. In *Proceedings of the Second Canadian Conference on Artificial Intelligence*, 268–277.

Ginsberg, M. L.; Frank, M.; Halpin, M. P.; and Torrance, M. C. 1990. Search lessons learned from crossword puzzles. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 210–215.

Haralick, R. M., and Elliott, G. L. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14:263–313.

Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence* 8:99–118.

Mackworth, A. K. 1987. Constraint satisfaction. In Shapiro, S. C., ed., *Encyclopedia of Artificial Intelligence*. John Wiley & Sons.

Maruyama, H. 1990. Structural disambiguation with constraint propagation. In *Proceedings of the 28th Conference of the Association for Computational Linguistics*, 31–38.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1990. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 17–24.

Montanari, U. 1974. Networks of constraints: Fundamental properties and applications to picture processing. *Inform. Sci.* 7:95–132.

Nadel, B. A. 1989. Constraint satisfaction algorithms. *Computational Intelligence* 5:188–224.

Nudel, B. 1983. Consistent-labeling problems and their algorithms: Expected-complexities and theory-based heuristics. *Artificial Intelligence* 21:135–178.

Prosser, P. 1993. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence* 9:268–299.

Seidel, R. 1983. On the complexity of achieving k-consistency. Department of Computer Science Technical Report 83-4, University of British Columbia. Cited in: A. K. Mackworth 1987.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press.

van Beek, P. 1992. On the minimality and decomposability of constraint networks. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 447–452.

Waltz, D. 1975. Understanding line drawings of scenes with shadows. In Winston, P. H., ed., *The Psychology of Computer Vision*. McGraw-Hill. 19–91.

Williams, C. P., and Hogg, T. 1992. Using deep structure to locate hard problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 472–477.