Partial Constraint Satisfaction

Eugene C. Freuder*
Computer Science Department
University of New Hampshire
Durham, New Hampshire 03824

Abstract

A constraint satisfaction problem involves finding values for variables subject to constraints on which combinations of values are allowed. In some cases it may be impossible or impractical to solve these problems completely. We may seek to partially solve the problem in an "optimal" or "sufficient" sense. A formal model is presented for defining and studying such partial constraint satisfaction problems. The basic components of this model are a constraint satisfaction problem, a problem space, and a metric on that space. Algorithms for solving partial constraint satisfaction problems are discussed. A specific branch and bound algorithm is described. Some initial experimental experience with this algorithm is presented.

1 Introduction

A constraint satisfaction problem (CSP) involves finding values for variables subject to constraints on which combinations of values are allowed. These problems are widely studied in artificial intelligence [Winston, 1984, Mackworth, 1977, Haralick and Shapiro, 1979]. The n-queens problem, place n queens on an n by n chessboard such that no two attack one another, is often used as an example and for experimental comparison of algorithms [Gaschnig, 1978, Haralick and Elliott, 1980, Nadel, 1988].

If a CSP is impossible or impractical to solve, we may be willing to settle for a solution to a "weaker" version of the problem. 1 call this the partial constraint satisfaction problem (PCSP). It may be characterized loosely as "do the best you can (or at least as well as...)" or "find me the closest problem that you can solve (or one at least as close as...)". This paper presents a formal model for PCSPs, discusses algorithms for solving them and presents some initial experimental results with one specific algorithm.

We will use the 3-queens problem in this paper as one source of simple illustrations. The 3-queens problem is unsolvable. However, suppose we only require placing two queens, or allow one pair of queens to attack each other, or allow queens to attack along diagonals, or expand the board to a 4 by 3 grid. In each case the problem becomes solvable. Notice also that the latter two partial problems are non-optimal in a set of similar problems: it would be enough to allow one pair of queens to attack diagonally or add a single additional square to the 3 by 3 board

As AI increasingly confronts real world problems, in expert systems and robotics, for example, we are increasingly likely to encounter situations where, rather than searching for a solution to a problem, we must, in a sense, search for a problem we can solve. Partial constraint satisfaction problems have arisen in several contexts. Boming uses "constraint hierarchies" to deal with situations in which a set of requirements and preferences for the graphical display of a physical simulation can not all be satisfied. [Borning, *et al,* 1987]. Descotte and Latombe make compromises among antagonist constraints in a planner for machining problems [Descotte and Latombe, 1985]. Fox introduces "relaxations" (alternative values) and "importance" to constraint representations to cope with conflicting constraints in job-shop scheduling [Fox, 1986]. A related problem is the expression of preferences in database queries [Lacroix and Lavency, 1987].

This paper introduces an abstract model for the study of PCSPs and their algorithms. Section 2 of the paper presents a model of PCSPs. Section 3 discusses algorithms for solving problems which fit this model.

2 Model

This section formalizes the notion of "partial constraint satisfaction problem." A partial constraint satisfaction problem (PCSP) consists of three components

<(P,U), (PS <), (M,(N,S))>

where:

P is a constraint satisfaction problem
U is a set of "universes", potential values for each variable in P
(PS,<) is a problem space; PS a set of problems, < a partial order
M is a metric on the problem space
(N,S) are necessary and sufficient bounds on the metric distance between a solvable problem in the space PS and the given problem P.

Each of these components will be discussed in turn. A solution to a PCSP is a problem P' from the problem space PS along with a solution to that problem where the metric

*This paper is based upon work supported by the National Science Foundation under Grant DCR-8601209.

distance of F from P is less than N. An optimal solution is one where the metric distance of F from P is minimal over the problem space. The optimal solution is maximal if there is no optimal solution which involves a problem Q such that F < Q.

2.1 The Constraint Satisfaction Problem

A constraint satisfaction problem can in turn be specified as a pair <V,C> where:

V is a set of variables

C is a set of constraints, i.e. relations on the variables.

Often attention is restricted to problems with discrete finite domains and binary constraints (involving only pairs of variables). We shall take these restrictions here for simplicity, though the approach should generalize to non binary constraints, and to a degree at least, to infinite domains. A specific pair of variables (a,b) permitted by a constraint C_{ij}; will be called an element of the constraint C_{ij}.

Domains of variable values can be specified as unary constraints, but to avoid handling variable domains as special cases in the exposition below we will treat them as binary constraints also, between a variable and itself. The value, a, is in the domain of variable v if the constraint C between v and v holds for the pair (a,a).

The 3-queens problem could be represented as follows:

$V = \{Q_1, Q_2, Q_3\}$

$C = \{C_{11}, C_{22}, C_{33}, C_{12}, C_{13}, C_{23}\}$

(i,j) specifies the chessboard square in row i, column j.

$C_{ij} = \{((i\ 1)\ (i\ 1))\ ((i\ 2)\ (i\ 2))\ ((i\ 3)\ (i\ 3))\}$

These constraints specify the variable domains. They indicate that for each row i, a queen can be placed in any of the three columns (if permitted by the constraints imposed by the placement of the other queens). For example, $((2\ 3)\ (2\ 3)) \in C_{22}$ means the second queen can be placed in the second row, third column. Note that we have taken the customary initial step of reducing the variable domains to single rows, rather than the entire chessboard.

$C_{12} = \{((1,1)\ (2,3))\ ((1,3)\ (2,1))\}$

$C_{13} = \{((1,1)\ (3,2))\ ((1,2)\ (3,1))\ ((1,2)\ (3,3))\ ((1,3)\ (3,2))\}$

$C_{23} = \{((2,1)\ (3,2))\ ((2,3)\ (3,1))\}$

These constraints specify combinations of values for the variables which are permitted by the constraint. For example, ((1,1) (3,2)) says that it is all right to put the first queen in the upper left hand corner and the third in the middle of the bottom row.

Solving a CSP means finding a set of values one from the domain of each variable which simultaneously satisfy all the constraints, i.e. the various combinations of values are contained in the domains of the corresponding relations.

U is a set of universes Uj one for each variable. For the 3-queens problem each U_i might be {(i,l) (i,2) (i,3) (i.4)| allowing the possibility of weakening the problem by adding additional squares in a fourth column.

As noted, the 3-queens problems has no solution. As a practical matter, even when a CSP is solvable the effort required to obtain a solution may be unacceptable. This brings us to consider a space of alternative problems, some of which may be both solvable, and "close enough" to the original problem for our purpose.

2.2 The Problem Space

A problem space is a partially ordered set, (PS,<), where:

PS is a set of CSPs

< is a partial order on PS defined as follows:

$P_1 < P_2$ if f the set of solutions to $\supseteq$ the set of solutions to P_2

We will say that P_1 is equivalent to P_2 and write $P| = P_2$ if the set of solutions to P_1 is the same as the set of solutions to P_2. If $P_1 \leq P_2$ and $P_1 \neq P_2$, we will write $P_1 < P_2$ and say that P_1 is weaker than P_2. The problem space for a PCSP must contain P, the original problem.

One natural problem space for a PCSP with problem P coasists of all problems Q such that Q < P. This set can be obtained by considering all the ways of weakening the constraints, i.e. all combinations of added values.

For example, in the 3-queens problem, the addition of another square at the end of row 2 corresponds to enlarging the C_{22} constraint to include ((2,4) (2,4)), and presumably adding elements to the C_{12} and C_{23} constraints, e.g. ((2,4) (3,1)). (We could, alternatively, have included, for example, ((2,4) (3,1)) in the original problem P, so that adding the square in row 2 would not, perhaps, involve moving as far from P.) The relaxation of the restriction that the first two queens cannot attack each other vertically corresponds to adding to C_{12} the elements ((1,1) (2,1)), ((1,2) (2,2)) and ((13) (2,3)).

It may be natural to consider a space which does not include all Q < P. We may wish to specify how the problem can be weakened. Some weaker problems may make more "semantic sense".

For example, in the 3-queen problem, we might allow removal of the restriction that the first two queens be non-attacking vertically, by adding three elements to C_{12} as just described, and we might allow a similar weakening of C_{13} and C_{23}; however, we might not allow into the problem space the following possibilities: We might prohibit on the one hand the problems that removed the vertical attacking restrictions on more than one pair of queens at a time. We might on the other hand prohibit the problems that only allowed a partial weakening of the vertical attacking restriction between pairs, e.g. by adding ((1,1) (2,1)) and ((1,2) (2,2)) but not ((1,3) (2,3)).

The specification of the problem space PS can clearly affect the efficiency of the PCSP search process. One way to specify the problem space is to specify generators, or operators, that take us from one problem P to a permitted set of problems Q_i, $Q_i \leq P$. There may be "global"

restrictions on these generators, e.g. choose one constraint from column A, one from column B.

The process of weakening CSP's can be naturally viewed as involving four options: enlarging a variable domain, enlarging a constraint domain, removing a variable, removing a constraint. However, all of these can in turn be expressed in terms of the basic process of enlarging constraint domains. We have already viewed variable domains as constraints. Enlarging a constraint Cj, until it contains all pairs in $U_I X U_J$, all pairs allowed by the specified universes for the two variables, is tantamount to removing the coastraint. Indeed for constraints Cjj, inot=j, enlarging C_{ij}; to contain all pairs (a,b) such that C_{ii}(a,a) and Cjj(bb) effectively removes Cjj at least until such a time as Cjj or C_{jj} may be enlarged. Removing C_{ij} for all j has the effect of removing the variable V_I.

In general, PS could contain problems, Q, which are stronger than P, P<Q, or problems, Q, such that neither Q<P nor P<Q; < is only a partial order. However, if we collect all the constraints in all the problems in PS into a single problem M, then all the problems in PS can be regarded as weakenings of M

2.3 The Metric

An obvious metric derives from the partial order. M(P,P') = the number of solutions not shared by P and P' When P' < P, this metric measures the number of solutions we have added by weakening P. This is a natural measure of how "good" our partial solution is likely to be.

Computing such a metric, however, is not likely to be easy. However, after finding a set of optimal solutions with another metric we might wish to distinguish among these by computing their solutions and determining maximal, optimal solutions. We also may wish to consider how well an alternative metric does tend to reflect this natural metric.

Another natural metric is a count of the number of constraint elements not shared by P and P' To some extent, this metric does reflect the metric based on the partial order. If P' is obtained from P by adding elements to the constraints then P' < P, because of the monotonic nature of constraint satisfaction problems. I.e. if for each constraint Cjj associated with P and constraint Cjj associated with P'. C_{ij} 2 C_{ij}, then P' < P.

The metric may do more than measure differences in constraint size by counting added constraint elements. Preferences can be expressed by ordering constraints [Descotte and Latombe, 1985] or by representing their importance [Fox, 1986]. Preferences could be associated with individual constraint elements or sets of elements. The metric can combine constraint deviations in a local or global manner [Bottling *et al.,* 1987]. Given the set of constraints C_{ij} associated with P' we might base the difference between ij P and P' on the maximum difference between Cjj and C_{ij} for any i, j, rather than the sum of differences. We might consider some kind of average difference or least squares measurement. The initial constraints may be viewed as ideal points which we seek to approximate by some measure.

For the 3-queens problem, we might decide that allowing diagonal attacks was preferable to allowing vertical attacks, and thus count the addition of ((1,1) (2,2)) as moving 2 units away from the original problem while adding ((1,1) (2,1)) only moves I unit away. If a problem P' differed from P only in (he constraint:

C_{12} = {((1,1) (2,3)) ((1,3) (2,1)) ((1,1) (2,2)) ((1,1) (2,1)))

M(P,P') might be measured as: 1+2=3, or max(l,2)=2 or ave (1,2)=1.5.

N and S provide bounds which can facilitate the search process. An acceptable solution must be closer to P than N. A search path can terminate when it is clear that it will not lead to such a solution. Any solution as close as S to P will suffice; all search can terminate when such a solution is found.

If we have no initial guidance of this nature to provide, N can be infinity and S can be O. We may know one "obvious" partial solution that provides an initial N. The larger we set S the "easier" we make the PCSP.

For the 3-queens problem we may know that no exact solution is possible and thus be willing to set S = 1. We may know that the 4-queens problem is solvable, thus one obvious partial solution of 3-queens would involve adding squares to make a 4 by 4 board.

There may be some "hidden agenda" embodied in the metric. For example, we may wish to drive a problem toward a weaker version that is easily solvable, e.g. by removing constraints to yield a problem with a tree-structured constraint graph [Mackworth and Freuder,1985, Dechter and Pearl, 1985].

3 Algorithms

The first part of this section discusses some general criteria for designing PCSP algorithms. The second part presents some initial experimental experience with a specific algorithm.

3.1 Design Criteria

A search paradigm must be chosen for seeking a PCSP solution. The one we will focus on here is branch and bound. This is a natural choice in seeking an optimal solution where partial solutions may be recognized as suboptimal. It is a natural extension of backtracking, which is the standard approach to CSPs. A further extension, which merits study, would be the A* paradigm. A* may be viewed as incorporating branch and bound, while allowing more heuristic flexibility in directing the search.

Branch and bound basically keeps track of the best solution found so far and abandons a line of search when it becomes clear that it cannot lead to a better solution. The N and S bounds provide further opportunity for discontinuing search.

The necessary bound N can be based on a priori knowledge of a possible solution, as suggested earlier. As branch and bound proceeds, if a solution is found at a distance D closer to P than N, N is, in effect, replaced by D. Though in general we would expect that changing a CSP problem to a PCSP problem by allowing partial solutions increases the complexity, if the sufficient bound S is large enough, the PCSP problem may be easier. In the 3-queens problem a

sufficient bound of just 1 can permit a solution to the 3-queens problem with less backtracking than would be required to discover that no exact solution exists.

Branch and bound may be integrated with CSP backtracking to produce a PCSP algorithm. The natural points at which to perform this integration are the failure points in a standard CSP backtracking algorithm.

The simplest choice is to add to backtracking upon top level failure, when no solution is found. A branch and bound loop can be added on the outside of the backtracking algorithm. This loop will run through the problems in the problem space, keeping track of the closest problem P' to P solved so far. Problems no closer to P' than P will be rejected immediately.

At the other extreme, branch and bound can be integrated at the lowest level of backtrack failure. As soon as a choice of a value, c, for a variable fails alternative problems P can be considered. For problems which allow c and which are still closer to the original problem P than the best solution problem so far, backtrack search can continue to the next variable, with a partial solution that includes the value c. A version of this approach has been implemented as algorithm PCSP1 and is described in more detail below.

Branch and bound can be integrated at failure points in between these extremes. The natural compromise would occur at the points where all the values for a given variable have been exhausted in standard backtrack search. At these times options for alternative problems may be explored.

There is a tradeoff involved in the choice of how to integrate CSP backtrack and PCSP branch and bound. By integrating at a lower level we take greater advantage of backtrack pruning to avoid unnecessary effort. On the other hand by integrating at a higher level we allow greater flexibility in heuristically guiding the search through the space of alternative problems.

As usual in branch and bound it is advantageous to order the search to heuristically increase the likelihood that a good, or ideally optimal, solution will be found early. The choice of values for a variable could be ordered, for example, so that those that requiring no further weakening of the problem were tried first, with the others tried in an order which reflected the amount by which the problem would have to be changed to accommodate these choices (how far the altered problem would be from the original P). For example, after placing the first queen in (1,1), the second queen can be placed in (2,1) or (2,2) by removing a constraint element, or (2,3) without changing the problem at all. It may prove desirable to take into consideration how many opportunities are opened up by altering a problem in a given way. For example, if a queen has been placed in (1,2), eliminating the requirement that queens not attack vertically adds one new possibility for row 2, while eliminating the requirement that queens not attack diagonally adds two new possibilities. If our metric treated both changes equally, the latter might prove preferable.

Whenever alternative problems are generated in an order which reflects their distance from the original problem, closest to furthest, generation can stop at that point when the necessary bound N is reached. If we have a top level integration of branch and bound, this point marks the termination of the PCSP algorithm.

In searching through the problem space for a solvable problem, it would be desirable to avoid changing the problem in ways that do not facilitate progress. For example, if two problems are equivalent, they both do not need to be considered. A nice feature of the PCSP1 algorithm is that for the type of PCSP for which it is designed it is able to use only the minimally different problem P required to proceed at each problem choice point.

We might naturally expect that the PCSP problem would be considerably harder than the CSP problem, and that its worst case complexity would be bounded by the size of the problem space PS. An analysis of the top level integration of branch and bound with backtracking seems to confirm that intuition. The top level loop could consider each problem in the problem space P. If we consider a problem space that allows all possible subsets of the constraint elements in the original problem P the size of this space could be roughly $2^{n^2} d^{n^2}$, where we assume d is the size of the universes of variable values. The bound on backtracking for a solution to each problem is exponential in n, where n is the number of variables. The resulting bound of $2^{n^2} d^{n^2} 2^n$ for the PCSP suggests that the PCSP problem is indeed potentially much worse than the CSP problem.

However, this is one of those interesting situations where our initial view of the problem could lead to false expectations that we might be too ready to see verified. The PCSP1 algorithm below clearly has worst case behavior no worse than CSP backtracking. (If the weakening of C_{ij}; constraints leads PCSP1 to consider more variable values than a CSP algorithm would, PCSP1 can have a higher bound; however, it will still be only exponential in the number of variables, as is the case for CSP backtracking.) How can this be? The answer lies in viewing the PCSP problem not as a search through a space of problems for a solvable one that is closest to the original problem, but as a search through the space of potential solutions for one that is closest to an exact solution.

While the former view can be useful, the latter clearly demonstrates that even a straightforward generate and test algorithm can solve the PCSP problem with no increase in the exponent in the worst case complexity bound over CSP backtracking.

On the other hand, the exponential CSP worst case bound is bad enough! Clearly we need to consider, as we have begun to in this section, how techniques like branch and bound and heuristic search may avoid achieving that bound.

3.2 A Branch and Bound Algorithm

The PCSP1I algorithm assumes that any weaker version of P that it wishes to use is in the problem space PS. It amends normal backtracking in a simple manner. Whenever a value c for a variable V would be rejected by normal backtracking, PCSP determines which constraints were violated, and weakens the problem by adding precisely those constraint elements needed to permit c. The metric used simply counts the number of constraint elements added. The distance of the weakened problem P' from the original problem P is calculated and used for branch and bound purposes. If the distance of P from P warrants, search continues to the next

level of the backtrack tree. Otherwise, c is rejected. Search continues even when a full solution is found unless the sufficient bound S has been reached or no further choices remain. PCSP1 does not permit adding new values to the variable domains (adding elements to the C_{ij}), though this could easily be included. The algorithm could be generalized to work with different metrics or problem spaces.

```
Algorithm PCSP1
Free variables:  initial-problem, N,S, best-solution,
                 best-distance
PCSP1 (variables, values, solution, distance,
       current-problem)
If variables = nil
    then
        best-solution <- solution
        N <- distance
        if N < S then exit at top level else return
    else
        if values    nil
        then
            solution' <- solution plus the first value,
                 c
            current-problem' <- current problem with
                 constraint elements added as needed to
                 permit c
            distance' <— the distance between the
                 initial-problem and the current-
                 problem
            if distance' <N
                 then
                      variables <- rest of the variables
                           minus the first
                      values <- domain of values for
                           the First of the rest of the
                           variables
                      PCSP1 (variables, values,
                           solution', distance', current-
                           problem)
                 else
                      values <- rest of the values
                           minus the first
                      PCSP1 (variables, values,
                           solution, distance, current-
                           problem)
        else
            return
```

A version of this algorithm was implemented in LISP and some tests performed to obtain some initial sense of the practical cost incurred by allowing partial solutions. The results were encouraging.

The implementation was tested on the n-queens problem and another variation of the n-queens problem devised to have no complete solution. The (n+l,n)-queens problem is defined here to be the problem of placing n+1 non-attacking queens on an n by n board.

In Figure 1, for the n-queens problem, n=4,5 and 6, the effort expended by PCSP1 is shown for different values of S. N is set to infinity to provide no assistance. PCSP1 requires more effort than backtracking for small S, but the work decreases as S increases, eventually becoming less than that required by backtracking for a complete solution. The work done is measured by counting the number of "pair-tests", i.e. checks to see if a pair (x,y) is an element of a constraint. (C_{ii} constraints were not used.) The solutions and the number of constraint elements added to permit the solutions (the distance from the original problem) are also shown. Solution a b c d stands for queens on squares (l,a), (2,b),(3,c)and(4,d).

S	CHECKS	SOLUTION	DISTANCE
		n=4	
0	73	2 4 1 3	0
1	38	1 1 4 2	1
2	38	1 1 4 2	1
3	19	1 1 2 2	3
4	9	1 1 1 2	4
5	9	1 1 1 2	4
6	6	1 1 1 1	6
		n=5	
0	240	1 3 5 2 4	0
1	234	1 3 5 2 2	1
2	86	1 1 2 5 3	2
3	66	1 1 1 5 2	3
4	58	1 1 1 4 2	4
5	30	1 1 1 2 2	5
6	30	1 1 1 2 2	5
7	14	1 1 1 1 2	7
8	14	1 1 1 1 2	7
9	14	1 1 1 1 2	7
10	10	1 1 1 1 1	10
		n=6	
0	820	2 4 6 1 3 5	0
1	466	1 1 5 2 6 3	1
2	376	1 1 3 5 2 4	2
3	274	1 1 2 2 6 3	3
4	171	1 1 1 2 6 3	4
5	157	1 1 1 2 5 3	5
6	94	1 1 1 1 6 2	6
7	83	1 1 1 1 4 2	7
8	43	1 1 1 1 2 2	8
9	43	1 1 1 1 2 2	8
10	43	1 1 1 1 2 2	8
11	20	1 1 1 1 1 2	11
12	20	1 1 1 1 1 2	11
13	20	1 1 1 1 1 2	11
14	20	1 1 1 1 1 2	11
15	15	1 1 1 1 1 1	15

Figure 1. PCSP1 on n-queens, N=infmity, varying S.

Figure 2 shows PCSPI data for different values of N, the necessary bound, with S=0, on the (n+l,n)-queens problem,

for n=4 and n=5. An initial value of N=5 can easily be arrived at for these problems, by considering the four "lines of attack" bearing upon the n+1 st queen. Solutions are available which add only two elements; if we knew that in advance we could benefit from setting N to 3. Solution a b c d & (e,f) stands for queens in squares (l,a), (2,b), (3,c), (4,d)and(e,f).

N	CHECKS	SOLUTION	DISTANCE
		n=4	
0	116	none	-
1	116	none	-
2	658	none	-
3	679	1 1 4 2 & (1,3)	2
4	769	1 1 4 2 & (1,3)	2
5	835	1 1 4 2 & (1,3)	2
6	842	1 1 4 2 & (1,3)	2
7	865	1 1 4 2 & (1,3)	2
8	876	1 1 4 2 & (1,3)	2
9	878	1 1 4 2 & (1,3)	2
10	878	1 1 4 2 & (1,3)	2
		n=5	
0	711	none	-
1	711	none	-
2	3692	none	-
3	4782	1 3 5 2 2 & (5,4)	2
4	4915	1 3 5 2 2 & (5,4)	2
5	4997	1 3 5 2 2 & (5,4)	2
6	5208	1 3 5 2 2 & (5,4)	2
7	5227	1 3 5 2 2 & (5,4)	2
8	5393	1 3 5 2 2 & (5,4)	2
9	5404	1 3 5 2 2 & (5,4)	2
10	5410	1 3 5 2 2 & (5,4)	2
11	5450	1 3 5 2 2 & (5,4)	2
12	5469	1 3 5 2 2 & (5,4)	2
13	5472	1 3 5 2 2 & (5,4)	2
14	5472	1 3 5 2 2 & (5,4)	2
15	5472	1 3 5 2 2 & (5,4)	2

Figure 2. PCSP1 on (n+1,n)-queens, S=0, varying N.

These results suggest that the effort required to search for even optimal partial constraint satisfaction need not be prohibitive. In the right circumstances we may actually save effort by settling for partial constraint satisfaction, while still setting limits on how far we are willing to depart from the original problem.

Acknowledgments

I wish to thank Ram Kulkarni who implemented the algorithm and carried out and otherwise assisted with the experiments.

References

[Borning, *et al.,* 1987] Borning, A, Duisberg, R., Freeman-Benson, B., Kramer, A., and Woolf, M., Constraint hierarchies, *Proceedings of OOPSLA '87,* 1987.

[Dechter and Pearl, 1985] Dechter, R. and Pearl J., The anatomy of easy problems: a constraint-satisfaction formulation, *Proceedings ofIJCAI-85,* 1985.

[Descotte and Latombe, 1985] Descorte, Y. and Latombe, J.C., Making compromises among antagonistic constraints in a planner, *Artificial Intelligence 27,* 183-217, 1985.

[Fox, 1986] Fox, M, Observation on the role of constraints in problem solving, *Proceedings Sixth Canadian Conference on Artificial Intelligence,* 1986.

[Gaschnig, 1978] Gaschnig, J., Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for satisficing assignment problems, *Proceedings of the Second National Conference of the Canadian Society for Computational Studies of Intelligence,* 1978.

[Haralick and Elliott, 1980] Haralick R. and Elliott, G., Increasing tree search efficiency for constraint satisfaction problems, *Artificial Intelligence 14,* 263-313, 1980.

[Haralick and Shapiro, 1979] Haralick, R. and Shapiro, L., The consistent labeling problem: part I, *IEEE Trans. Pattern Anal. Machine Intel!.* 7, 173-184, 1979.

[Lacroix and Lavency, 1987] Lacroix, M. and Lavency, P., Preferences: putting more knowledge into queries, *Proceedings of the 15th International Conference on Very Large Data Bases,* 1987.

[Mackworth, 1977] Mack worth, A., Consistency in networks of relations, *Artificial Intelligence 8,* 99-118, 1977.

[Mackworth and Freuder, 1985] Mackworth, A. and Freuder, E., The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, *Artificial Intelligence 25,* 65-74, 1985.

[Nadel, 1988] Nadel, B., Tree search and arc consistency in constraint satisfaction algorithms, in L. Kanal and V. Kumar (eds.) *Search in Artificial Intelligence,* New York: Springer- Verlag, 1988.

[Winston, 1984] Winston, P., *Artificial Intelligence,* second edition, Reading, MA: Addison-Wesley, 1984.