

**THE ANATOMY OF EASY PROBLEMS:
A CONSTRAINT-SATISFACTION FORMULATION**

**Rina Dechter
Judea Pearl**

**December 1984
Report No. CSD-840063**

MASTER COPY



The Anatomy of Easy Problems: A Constraint-Satisfaction Formulation

Rina Dechter and Judea Pearl

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024
telephone number:(213) 825-3243
netmail address: dechter@UCLA-LOCUS.ARPA
judea@UCLA-LOCUS.ARPA

Topic: problem solving (automatic reasoning)

Keywords: constraint-satisfaction, backtracking, heuristic-discovery, relaxation

Word count: 4900

Abstract

This work aims towards the automatic generation of advice to guide the solution of difficult constraint-satisfaction problems (CSPs). The advice is generated by consulting relaxed, easy models which are backtrack-free.

We identify a subset of CSPs whose syntactic and semantic properties make them easy to solve. The syntactic properties involve the structure of the constraint graph, while the semantic properties guarantee some local consistencies among the constraints. In particular, problems supported by tree-like constraint graphs, and some width-2 graphs, can be easily solved and are therefore chosen as the target model for the relaxation scheme. Optimal algorithms for solving easy problems are presented and analyzed. Finally, an efficient method is introduced for extracting advice from easy problems and using it to speedup the solution of hard problems.

1. Introduction

1.1 Why study easy problems?

An important component of human problem-solving expertise is the ability to use knowledge about solving easy problems to guide the solution of difficult ones. Only a few works in AI [Sacerdoti 1974]

[Carbonell 1983], , have attempted to equip machines with similar capabilities. Gaschnig [Gaschnig 1979] , Guida et.al [Guida 1979]. and Pearl [Pearl 1983] suggested that knowledge about easy problems could be instrumental in the mechanical discovery of heuristics. Accordingly, it should be possible to manipulate the representation of a difficult problem until it is transformed into an easy one, solve the easy problem, then use the solution to guide the search process in the original problem.

The implementation of this scheme requires three major steps: 1. simplification 2. solution 3. advice generation. Additionally, to perform the simplification step, we must have a simple, a-priori criterion for deciding when a problem lends itself to easy solution.

This paper uses the domain of constraint-satisfaction tasks to examine the feasibility of these three steps. It establishes criteria for recognizing classes of easy problems, it provides special procedures for solving them, and it introduces an efficient method of extracting advice from them.

Constraint-satisfaction problems (CSP) involve the assignment of values to variables subject to a set of constraints. Understanding three-dimensional drawings, graph coloring, electronic circuit analysis, and truth maintenance systems are examples of CSP problems. These are normally solved by some version of backtrack search which may require exponential search time (for example, the graph coloring problem is known to be NP-complete.)

In general, a problem is considered easy when its representation permits a solution in polynomial time. However, since we are dealing mainly with backtrack algorithms, we will consider a CSP *easy* if it can be solved by a *backtrack-free* procedure. Normally, a backtracking

algorithm instantiates variables in a predetermined order, and for each next variable it chooses one value that is consistent with all previous assignments. If it doesn't find one, it *backtracks* to the previous variable, tries a new assignment for it, and continues from there. The algorithm stops when all variables have been assigned values or when no new untried values are left for the first variable. A backtrack-free search is one in which the backtracking algorithm completes without backtracking, thus producing a solution in time linear with the number of variables.

Most of our discussion is based on the concept of constraint-graphs [Mackworth 1977] in which the nodes represent variables and the undirected arcs represent the existence of an explicit constraint between them. Freuder [Freuder 1982] has identified sufficient conditions for a constraint graph to yield a backtrack-free CSP, and has shown, for example, that tree-like constraint graphs can be made to satisfy these conditions, with a small amount of processing. Our main purpose here is to further study classes of constraint graphs leading themselves to backtrack-free solutions and to devise efficient algorithms for solving them. Once these classes are identified they can be chosen as targets for a problem simplification scheme; constraints can be selectively deleted from the original specification so as to transform the original problem into a backtrack-free one. The simplified problem can then provide advice on choices pending in the original problem. For example, we propose to use the "number of consistent solutions in the simplified problem" as a figure of merit to establish priority of value assignments in the backtracking search of the original problem. We show that this figure of merit can be computed in time comparable to that of finding a single solution to an easy problem. (For details regarding the process of constraint-deletion see [Dechter 1984])

1.2 Definitions and Nomenclature

Definition 1 ([Freuder 1982]) An ordered constraint graph is a constraint graph in which the nodes are linearly ordered to reflect the sequence of variable assignments executed by the backtrack search algorithm. The *width* of a node is the number of arcs that leads from that node to previous nodes, the *Width of an ordering* is the maximum width of all nodes, and the *width of a graph* is the minimum width of all the orderings of that graph.

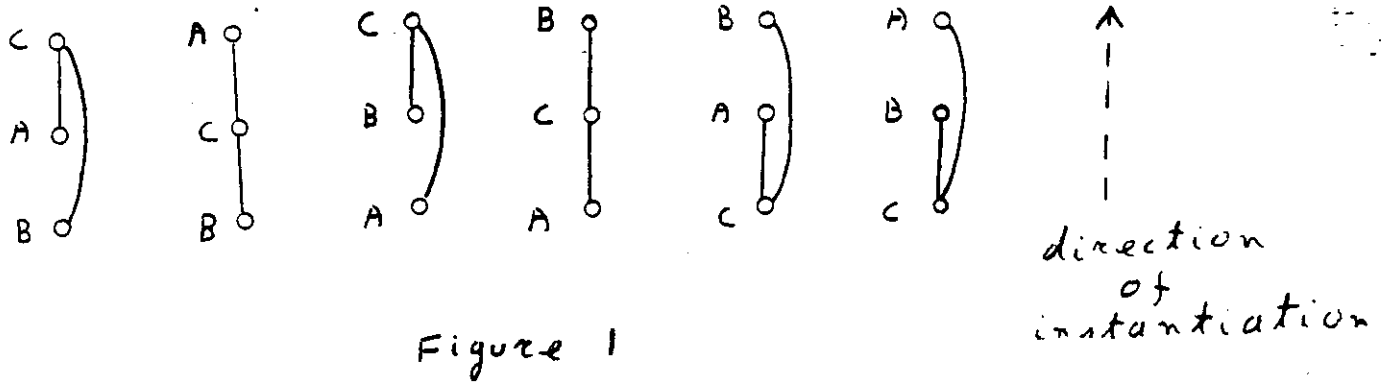


Figure 1 presents six possible orderings of a constraint graph. The width of node C in the first ordering is 2 while in the second ordering it is 1. The width of the first ordering is 2 while the second is 1. The width of the constraint graph is, therefore, 1. Freuder provided an efficient algorithm for finding both the width of a graph and the ordering corresponding to this width. He further showed that a constraint graph is a tree iff it is of width 1.

Montanari [Montanari 1974] and Mackworth [Mackworth 1977] have introduced two kinds of local consistencies among constraints named *arc consistency* and *path consistency*. Their definitions assume that the graph is directed, i.e., each symmetric constraint is represented by two directed arcs.

Let D_i stand for the domain of variable V_i ; $R_{ij}(x,y)$ stands for the assertion that (x,y) is permitted by the explicit constraint R_{ij} between V_i and V_j .

Definition 2 ([Mackworth 1977]): Directed arc (V_i, V_j) is *arc consistent* iff for any value $x \in D_i$, there is a value $y \in D_j$, such that $R_{ij}(x,y)$

Definition 3 ([Montanari 1974]): A path of length m through nodes (i_0, i_1, \dots, i_m) is *path consistent* iff for any value $x \in D_{i_0}$ and $y \in D_{i_m}$ such that $R_{i_0 i_m}(x,y)$, there is a sequence of values $z_1 \in D_{i_1}, \dots, z_{m-1} \in D_{i_{m-1}}$ such that

$$R_{i,j}(x, z_1) \text{ and } R_{i,j}(z_1, z_2) \text{ and } \dots \text{ and } R_{i,m}(z_{m-1}, y).$$

$R_{i,j}$ may also be the universal relation e.g., permitting all possible pairs.

A constraint graph is arc (path) consistent if each of its directed arcs (paths) is arc (path) consistent. Achieving "arc-consistency" means deleting certain values from the domains of certain variables such that the resultant graph will be arc-consistent, while still representing the same overall set of solutions. To achieve path-consistency, certain pairs of values that were initially allowed by the local constraints should be disallowed. Montanari and Mackworth have proposed polynomial-time algorithms for achieving arc-consistency and path consistency. In [Mackworth 1984] it is shown that arc consistency can be achieved in $O(ek^3)$ while path consistency can be achieved in $O(n^3k^3)$. n is the number of variables, k is the number of possible values, and e is the number of edges.

The following theorem is due to Freuder.

Theorem 1

- (a) If the constraint graph has a width 1 (i.e. the constraint graph is a tree) and if it is arc consistent then it admits backtrack-free solutions.
- (b) If the width of the constraint graph is 2 and it is also path consistent then it admits backtrack-free solutions.

The above theorem suggests that tree-like CSP'S (CSP's whose constraint graph are trees) can be solved by first achieving arc consistency and then instantiating the variables in an order which makes the graph have width 1. Since this backtrack-free instantiation takes $O(nk^2)$ steps the whole problem can be solved in $O(nk^3)$ and, therefore, tree-like CSP's are easy. The test for this property is also easily verified; to check whether or not a given graph is a tree can be done by a regular $O(n^2)$ spanning tree algorithm.

It is important to note that a given CSP may have several equivalent representations, in the sense of admitting the same set of solutions. Yet each representation may have a different constraint-graph, one of which may be a tree. However, testing whether a CSP has an

equivalent tree representation and finding such a representation might be a very difficult task.

The second part of the theorem tempts us to conclude that a width-2 constraint graph should admit a Backtrack-free solution after passing through a path-consistency algorithm. In this case, however, the path consistency algorithm may add arcs to the graph and increase its width beyond 2. This often happens when the algorithm deletes value-pairs from a pair of variables that were initially related by the universal constraint (having no connecting arc between them), and it is often the case that passage through a path-consistency algorithm renders the constraint-graph complete. It may happen, therefore, that no advantage could be taken of the fact that a CSP possesses a width-2 constraint graph if it is not already path consistent. We are not even sure whether width-2 suffices to preclude NP-completeness.

In the following section we give weaker definitions of arc and path consistency which are also sufficient for guaranteeing backtrack-free solutions but have two advantages over those defined by [Montanari 1974] and [Mackworth 1977] :

1. They can be achieved more efficiently, and
2. They add fewer arcs to the constraint-graph, thus preserving the graph width in a larger classes of problems.

2. Algorithms for achieving directional consistency

2.1 Case of Width-1

In constraint-graphs which are trees, full arc-consistency is more than what is actually required for enabling backtrack-free solutions. For example, if the constraint graph in figure 2 is ordered by (V_1, V_2, V_3, V_4) , nothing is gained by making the directed arc (V_3, V_1) consistent. To ensure backtrack-free assignment, we need only make sure that any value assigned to variable V_1 will have at least one consistent value in D_3 . This can be achieved by making only the directed arc (V_1, V_3) consistent, regardless of whether (V_3, V_1) is consistent. We therefore see that arc-consistency is required only w.r.t a single direction, that specified by the order in which the



Figure 2

backtrack algorithm will later choose variables for instantiations. This motivates the following definitions.

Definition: Given an order d on the constraint graph R , we say that R is d -arc-consistent if all the directed edges which follow the order d are arc-consistent.

Theorem 2

Let d be a width-1 order of an ordered constraint-tree, T . If T is d -arc-consistent then the backtrack search along the order d is backtrack-free.

proof

Suppose V_1, V_2, \dots, V_t were already instantiated. The variable V_{t+1} is connected to at most one previous variable (follows from the width-1 property), say V_i , which was assigned the value v_i . Since the directed arc (V_i, V_{t+1}) is along the order d , its arc-consistency implies the existence of a value v_{t+1} such that the pair (v_i, v_{t+1}) is permitted by the constraint $R_{(t+1)}$. Thus, the assignment of v_{t+1} is consistent with all previous assignments.

An algorithm for achieving directional arc-consistency for any ordered constraint graph is given next (The order $d = (V_1, V_2, \dots, V_n)$ is assumed)

DAC- d -arc-consistency

1. begin
2. For $i=n$ to 1 by -1 do

3. For each arc $(V_j, V_i); j < i$ do
4. REVISE(V_j, V_i)
5. end
6. end
7. end

The algorithm REVISE(V_j, V_i), given in [Mackworth 1977] , deletes values from the domain D_j until the directed arc (V_j, V_i) is arc-consistent.

REVISE(V_j, V_i)

1. begin
2. For each $x \in D_j$ do
3. if there is no value $y \in D_i$ s.t. $R_{j,i}(x, y)$ then
4. delete x from D_j
5. end
6. end

To prove that the algorithm achieves d-arc-consistency we have to show that upon termination, any arc (V_j, V_i) along d ($j < i$), is arc-consistent. The algorithm revises each d-directed arc once. It remains to be shown that the consistency of an already processed arc is not violated by the processing of coming arcs. Let arc (V_j, V_i) ($j < i$) be an arc just processed by REVISE(V_j, V_i). to destroy the consistency of (V_j, V_i) some values should be deleted from the domain of V_i during the continuation of the algorithm. However, according to the order by which REVISE is performed from this point on, only lower indexed variables may have their set of values updated. Therefore, once a directed arc is made arc-consistent its consistency will not be violated.

For comparison, the algorithm AC-3 that achieves full arc-consistency [Mackworth 1977] is given next:

AC-3

1. begin
2. $Q \leftarrow \{ (V_i, V_j) \mid (V_i, V_j) \in \text{arcs}, i \neq j \}$
3. while Q is not empty do
4. select and delete arc (V_i, V_m) from Q
5. REVISE(V_i, V_m)
6. if REVISE(V_i, V_m) caused any change then
7. $Q \leftarrow Q \cup \{ (V_i, V_k) \mid (V_i, V_k) \in \text{arcs}, i \neq k, m \}$
7. end
8. end

The directional arc-consistency algorithm takes ek^2 steps since the REVISE algorithm, taking k^2 tests, is applied to every arc exactly once. It is also optimal, because even to verify directional arc-consistency each arc should be inspected once, and that takes k^2 tests. By comparison, the complexity of AC-3, achieving full arc-consistency is, $O(ek^3)$.

In particular, when the constraint graph is a tree, the complexity of the directional arc-consistency algorithm is $O(nk^2)$.

Theorem 3:

A tree-like CSP can be solved in $O(nk^2)$ steps and this is optimal.

proof

Given that we know that the constraint graph is a tree, finding an order that will render it of width-1 takes $O(n)$ steps. A width-1 tree-CSP can be made d-arc-consistent in $n \cdot k^2$ steps, using the DAC algorithm. The backtrack-free solution on the resultant tree is found in nk^2 . Finding a solution to tree-like CSP's takes, therefore, $2nk^2 + O(n) = O(nk^2)$. This complexity is also optimal since any algorithm for solving a tree-like problem must examine each constraint at least once, and each such examination takes $O(k^2)$.

Interestingly, if we apply DAC w.r.t. order d and then DAC w.r.t. the reverse order we get a full arc-consistency for trees. We can, therefore, achieve full arc-consistency on trees in

$O(nk^2)$. Algorithm AC-3, on the other hand, can be shown to have a worst case performance on trees of $O(nk^2)$. On general graphs, however, the (full) arc-consistency algorithm cannot be improved, and the AC-3 algorithm is optimal (see [Dechter 1984]).

Returning to our primary aim of studying easy problems, we now show how advice can be generated for solving a difficult CSP using a relaxed tree-like approximation. Suppose that we want to solve an n variables CSP using a backtracking procedure with V_1, V_2, \dots, V_n as the order of instantiation. Let V_i be the variable to instantiate next, with $v_{i1}, v_{i2}, \dots, v_{ik}$ the possible candidate values. To minimize backtracking we should first try values which are likely to lead to a consistent solution but, since this likelihood is not known in advance, we may estimate it, instead, by counting the number of consistent solutions that each candidate admits in some relaxed problem. We generate a relaxed tree-like problem by deleting some of the explicit constraints given, then count the number of consistent solutions containing each of the possible k assignments, and finally use these counts as a figure of merit for scheduling the various assignments. In the following we show how the counting of consistent solutions can be imbedded within the d -arc-consistency algorithm, DAC, on trees.

Any width-1 order, d , on a constraint tree determines a directed tree in which a parent always precedes its children in d (arcs are directed from the parent to its children). Let $N(v_{jt})$ stands for the number of solutions in the subtree rooted at V_j , consistent with the assignment of v_{jt} to V_j . It can be shown that $N(\cdot)$ satisfy the following recurrence:

$$N(v_{jt}) = \prod_{\{c | V_c \text{ is a child of } V_j\}} \sum_{\{v_{ct} \in D_c | R_{j,c}(v_{jt}, v_{ct})\}} N(v_{ct})$$

From this recurrence it is clear that the computation of $N(v_{jt})$ may follow the exact same steps as in DAC; Simultaneously with testing that a given value v_{jt} is consistent with each of its children nodes, we simply transfer from each child of V_j to v_{jt} the sum total of the counts computed for the child's values that are consistent with v_{jt} . The overall value of $N(v_{jt})$ will be computed later on by multiplying together the summations obtained from each of the children. Thus, counting the number of solutions in a tree with n variables takes $O(nk^2)$, the same as establishing directional arc-consistency.

2.2 Case of Width-2

Order information can also facilitate backtrack-free search on width-2 problems by making path-consistency algorithms directional.

Montanari had shown that if a network of constraints is consistent w.r.t all paths of length 2 (in the complete network) then it is path-consistent. Similarly we will show that directional path-consistency w.r.t. length-2 paths is sufficient to obtain a backtrack-free search on a width-2 problems.

Definition: A constraint graph, R , ordered w.r.t. order $d = (V_1, V_2, \dots, V_n)$, is d -path-consistent if for every pair of values (x, y) , $x \in V_i$ and $y \in V_j$, s.t. $R_{ij}(x, y)$ and $i < j$, there exist a value $z \in V_k$, $k > j$ s.t. $R_{ik}(x, z)$ and $R_{kj}(z, y)$ for every $k > j$

Theorem 4

Let d be a width-2 order of an ordered constraint graph. If R is directional arc and path-consistent w.r.t d then it is backtrack-free.

proof

To ensure that a width-2 ordered constraint graph will be backtrack-free it is required that the next variable to be instantiated will have values that are consistent with previous chosen values. Suppose V_1, V_2, \dots, V_k were already instantiated. The variable V_{k+1} is connected to at most two previous variables (follows from the width-2 property). If it is connected to V_i and V_j , $i, j < k$ then directional path consistency implies that for any assignment of values to V_i, V_j , there exist a consistent assignment for V_{k+1} . If V_{k+1} is connected to one previous variable, then directional arc-consistency ensure the existence of a consistent assignment.

An algorithm for achieving directional path-consistency on any ordered graph will have to manage not only the changes made to the constraints but also the changes made to the graph, i.e. the arcs which are added to it. To describe the algorithm we use a representation in which a constraint R_{ij} is given by a matrix whose rows and columns correspond to the values of the two variables, and the entries are 0, and 1 for disallowed and allowed pairs, respectively. The matrix R_{ii} whose off-diagonal values are 0, represents the set of values permitted for variable V_i . Two operation on relations are needed: Intersection and Composition.

Intersection: If two constraints, R'_{ij} and R''_{ij} should hold simultaneously, then their intersection R_{ij} is written: $R_{ij} = R'_{ij} \& R''_{ij}$, and the entries in the corresponding matrices combine, term by term, by a logical \wedge .

Composition: Suppose relation R_{12} holds between V_1 and V_2 and R_{23} between V_2 and V_3 then the induced relation transmitted by V_2 is the composite relation R_{13} and it is defined by the matrix multiplication

$$R_{13} = R_{12} \cdot R_{23}$$

Given a network of constraints $R = (V, E)$ and an order $d = (V_1, V_2, \dots, V_n)$, we next describe an algorithm which achieves path-consistency w.r.t. this order.

DPC-d-path-consistency

begin

(1) $Y^0 = R$

(2) for $k = n$ to 1 by -1 do

(a) $\forall i \leq k$ connected to k do

$$Y'_{ii} = Y^0_{ii} \& Y_{ik} \cdot Y_{kk} \cdot Y_{ki} / * \text{ this is REVISE}(i,k)$$

(b) $\forall i, j \leq k$ s.t. $(V_i, V_k), (V_j, V_k) \in E$ do

$$Y^{\pm}_{ij} = Y^{\pm-1}_{ij} \& Y^{\pm-1}_{ik} \cdot Y^{\pm-1}_{kj} \cdot Y^{\pm-1}_{ij}$$

$$E \leftarrow E \cup (V_i, V_j)$$

end

end

Step 2a is the equivalent of the REVISE(i,k) procedure, and it performs the directional arc-consistency. Step 2b updates the constraints between pairs of variables transmitted by a third variable which is higher in the order d . If $V_i, V_j, i, j < k$ are not connected to V_k then the relation between the first two variables is not effected by V_k at all. If only one variable, V_i , is connected to V_k , the effect of V_k on the constraint (V_i, V_j) will be computed by step (2a) of the algorithm. The only time a variable V_k effects the constraints between pairs of earlier variables is if it is connected to both. It is in this case only that a new arc may be added to the graph.

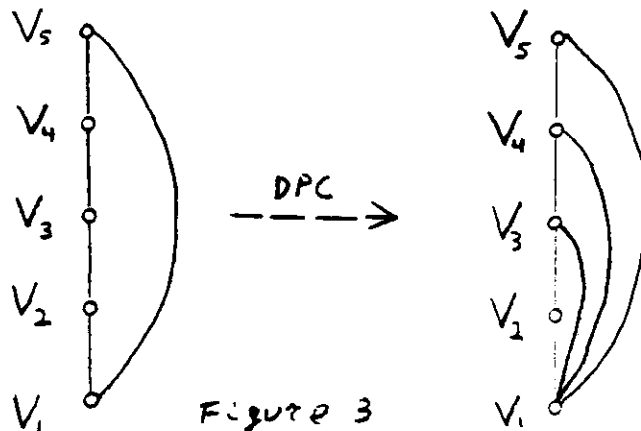
The complexity of the directional-path-consistency algorithm is $O(n^3k^3)$. For variable V_i , the number of times the inner loop, 2b, is executed is at most $O((i-1)^2)$ (the number of different pairs less than i), and each step is of order k^3 . The computation of loop 2a is completely dominated by the computation of 2b, and can be ignored. Therefore, the overall complexity is

$$\sum_{i=2}^n (i-1)^2 k^3 = O(n^3 k^3)$$

Applying directional-path-consistency to a width-2 graph may increase its width and therefore, does not guarantee backtrack-free solutions. Consequently it is useful to define the following subclass of width-2 CSP problems.

Definition: A constraint graph is *regular width-2* if there exist a width-2 ordering of the graph which remains width-2 after applying d-path-consistency, DPC.

A ring constitutes an example of a regular-width-2. Figure 3 shows an ordering of a ring's nodes and the graph resulting from applying the DPC algorithm to the ring. Both graphs are of width-2.



Theorem 5:

A regular width-2 CSP can be solved in $O(n^3k^3)$

Proof

Regular width-2 problem can be solved by first applying the DPC algorithm and then performing a backtrack-free search on the resulting graph. The first takes $O(n^3k^3)$ steps the second $O(n^2k^2)$ steps.

The main problem with the preceding approach is whether a regular width-2 CSP can be recognized from the properties of its constraint graph. One promising approach is to identify nonseparable components of the graph and all its separation vertices [Even 1979].

definition: A connected graph $G(V,E)$ is said to have a *separation vertex* v if there exist vertices a and b , such that all the paths connecting a and b pass through v . A graph which has a separation vertex is called *separable*, and one which has none is called *nonseparable*.

An $O(|E|)$ algorithm for finding all the nonseparable components and the separation vertices is given in [Even 1979] It is also shown that the connectivity structure between the nonseparable components and the separation vertices, has a tree structure.

The following points can be made:

1. Given any ordered constraint graph in which the separation vertices and the nonseparable components are identified, the directional path-consistency algorithm adds arcs only within each component.
2. Let R be a graph and SR be the tree in which the nonseparable components C_1, C_2, \dots, C_r and the separating vertices V_1, V_2, \dots, V_t are represented by nodes. A width-1 ordering of SR dictates a partial order on R, d^s , in which each separating vertex precede all the vertices in its children components of SR . It can be shown that if there exist a d^s ordering on R such that each nonseparable component is regular-width-2 then the total ordering is regular width-2.

As a corollary of these two points we conclude that a tree of simple rings is regular width-2.

3. Summary and conclusions

This paper examines the process of harnessing easy problems to help in the solution of complex constraint-satisfaction problems. Of the three main steps involved in this process – simplification, solution, and advice generation – we concentrated on the following:

1. The simplification part; we have devised criteria for recognizing easy problems based on their underlying constraint graphs. The characteristics that meet these criteria can be used as

goals for simplifying complex problems by deleting some of their constraints. The introduction of directionality into the notions of arc and path consistency enable us to extend the class of recognizable easy problems beyond trees, to include regular width-2 problems.

2. The solution part; using directionality we were able to devise improved algorithms for solving simplified problems and to demonstrate their optimality. In particular, it is shown that tree-structured problems can be solved in $O(nk^2)$ steps, and regular width-2 problems in $O(n^3k^2)$ steps.
3. The advice generation part; we have demonstrated a simple method of extracting advice from easy problems to help a backtracking algorithm decide between pending options of value assignments. The method involves approximating the remaining part of a constraint-satisfaction task by a tree-structured problem, and counting the number of solutions consistent with each pending assignment. These counts can be obtained efficiently and can be used as figures of merit to rate the promise offered by each option.

References

- [Carbonell 1983] Carbonell, J.G., "Learning by analogy: Formulation and generating plan from past experience," in *Machine Learning*, Michalski, Carbonell and Mitchell, Ed. Palo Alto, California: Tioga Press, 1983.
- [Dechter 1984] Dechter, R. and J.Pearl, "A problem simplification approach that generates heuristics for constraint satisfaction problems.," UCLA-Eng-rep.8497, Los Angeles, California, 1984.
- [Even 1979] Even, S., *Graph Algorithms*, Maryland, USA: Computer Science Press, 1979.
- [Freuder 1982] Freuder, E.C., "A sufficient condition of backtrack-free search.," *Journal of the ACM*, Vol. 29, No. 1, 1982, pp. 24-32.
- [Gaschnig 1979] Gaschnig, J., "A problem similarity approach to devising heuristics: first results," in *Proceedings 6th international joint conf. on Artificial Intelligence.*, Tokyo, Jappan: 1979, pp. 301-307.
- [Guida 1979] Guida, G. and M. Somalvico, "A method for computing heuristics in problem solving," *Information Sciences*, Vol. 19, 1979, pp. 251-259.
- [Mackworth 1977] Mackworth, A.K., "Consistency in networks of relations," *Artificial intelligence*, Vol. 8, No. 1, 1977, pp. 99-118.
- [Mackworth 1984] Mackworth, A.K. and E.C. Freuder, "The complexity of some polynomial consistency algorithms for constraint satisfaction problems," *To appear in AI Journal*, 1984.
- [Montanari 1974] Montanari, U., "Networks of constraints :fundamental properties and applications to picture processing," *information science*, Vol. 7, 1974, pp. 95-132.
- [Pearl 1983] Pearl, J., "On the discovery and generation of certain heuristics," *AI Magazine*, No. 22-23, 1983.
- [Sacerdoti 1974] Sacerdoti, E. D., "Planning in a hierarchy of abstraction spaces," *Artificial Intelligence*, Vol. 5, No. 2, 1974, pp. 115-135.