ON THE COMPLEXITY OF ACHIEVING K-CONSISTENCY [1]

by

Raimund Seidel

Technical Report 83-4

Department of Computer Science
Cornell University
Ithaca NY. 14853

Abstract:

A number of combinatorial search problems can be formulated as constraint satisfaction problems. Typically backtrack search is used to solve these problems. To counteract the frequent thrashing behaviour of backtrack search, methods have been proposed to precondition constraint satisfaction problems. These methods remove inconsistencies involving only a small number of variables from the problem. In this note we analyze the time complexity of the most general of these methods, Freuder's k-consistency algorithm. We show that it takes worst case time $O(n^k)$, where n is the number of variables in the problem.

---

[1] This research was done while the author was a student at the Department of Computer Science of the University of British Columbia.

I. Introduction

This note deals with the complexity analysis of Freuder's [Fr1] algorithm for establishing k-consistency in a constraint satisfaction problem (CSP). Freuder defines a CSP as follows: We are given a set of n variables $X_1,...,X_n$ and constraints on subsets of these variables limiting the values they can take on. These constraints taken together constitute a global constraint which specifies which sets of values $a_1,...,a_n$ for $X_1,...,X_n$ can simultaneously satisfy all given constraints. In other words, the constraints define an n-ary relation. Our problem is to synthesize this relation, i.e. to determine those sets of values which simultaneously satisfy the set of constraints.

A number of combinatorial search problems can be modelled as CSPs. Among them are graph colouring, graph isomorphism, the eight queens problem, labelling problems in scene analysis, and others (see [H-S]). Typically CSPs are solved by backtrack search. However, several authors ([Wa],[Mon],[Ma]) have pointed out the frequent thrashing behaviour of backtrack search. To remedy the situation, notions of local consistency were introduced.

One of the simplest notions is so-called arc consistency. It applies to all CSPs in which all constraints involve at most two variables. Arc consistency is violated if for some value a_i of some variable X_i there is a variable X_j for which no instantiation a_j is allowed by the binary constraint between X_i and X_j. Clearly a_i cannot appear in any solution to the CSP. Thus the purpose of arc consistency algorithms is to remove such values a_i from the domain of X_i in order to prevent that such an inconsistency will be discovered by a backtrack search repeatedly.

Waltz [Wa] and Mackworth [Ma] gave algorithms to achieve arc consistency. The worst case running time of these algorithms was originally unknown and subject of controversy (see [Ga]). However, recently Mackworth and Freuder [M-F] settled the question and showed that the fastest of these algorithms (AC-3 in [Ma]) has a worst case running time linear in the number of constraints.

Montanari [Mon] developed a more general notion of consistency called path consistency. Mackworth [Ma] gave algorithms for achieving path consistency. Later he and Freuder [M-F] analyzed the complexity of these algorithms and found that the fastest one has a worst case running time of $O(n^2)$.

The notions of arc and path consistency are subsumed by the notion of k-consistency invented by Freuder [Fr1]. Informally, k-consistency of a CSP means that any valid instantiation of k-1 variables can be extended for any one of the remaining variables to a valid instantiation of k variables. Freuder presented a short and highly recursive algorithm for achieving k-consistency but did not prove any complexity results about it. In [Fr2] he gave conditions when k-consistency alone is sufficient to find a global solution of a CSP quickly.

In the following we will show that, for fixed k, the running time of Freuder's k-consistency algorithm is bounded by a polynomial in the number of variables which has degree k.

II. Analysis of Freuder's Algorithms

Let us at first present several definitions which will enable us to state Freuder's algorithm precisely. These definitions follow almost verbatim the ones given in [Fr1].

We are dealing with CSPs involving n variables $X_1,...,X_n$ which may take on values from the finite domains $D_1,...,D_n$ respectively. It is assumed that the size of each D_i is not greater than some integer constant d. Let I=$\{1,...,n\}$. Many of the definitions will be made for any non-empty subset $J \subset I$. We denote by X_J the indexed set of variables $\{X_j\}_{j \in J}$. A value a_i in D_i will be called an *instantiation of* X_i. An *instantiation of a set of variables* X_J, denoted by a_J, is an indexed set of values $\{a_j\}_{j \in J}$. The indexed set notation implies that there is a function, a, from J onto the instantiation a_J, which serves to indicate which member of a_J instantiates which variable: the value of a at j, denoted by a_j, is the instantiation of X_j.

A *constraint on* X_J, denoted C_J, is a set of instantiations of X_J. It is also possible to represent a constraint C_J as a set of m-tuples, where m=|J|, and thus as an m-ary relation or predicate. We have found it useful, however, to use set notation rather than to refer to cross products of predicates in the presentation which follows.

A *constraint expression of order k* is the set of constraints $C = \{C_J \mid 1 \leq |J| \leq k\}$. This represents the logical conjunction of the relations expressed by the C_J. Normally we will not be given constraints for all $J \subset I$, $|J| \leq k$ explicitly. However, we can assume that they exist, with no loss of generality, as the "non-constraint" for X_J can always be specified: the set of all combinations of elements from the domains of the variables in X_J.

We say an instantiation a_J *satisfies* a constraint C_J if $a_J \in C_J$. The instantiation a_J *satisfies* a constraint C_H, $H \subset J$, if the set $\{a_j \in a_J\}_{j \in H}$ is a member of C_H, and we say a_J *satisfies* C_H, $J \subset H$, if there is an a_H in C_H for which $\{a_j \in a_H\}_{j \in J} = a_J$. An instantiation a_J with $|J| = k$, *k-satisfies* a constraint expression C, if a_J satisfies all the constraints $C_H \in C$ *with* $H \subset J$. Observe that an instantiation a_I which n-satisfies a constraint expression is a solution of the CSP corresponding to the expression.

Consider two constraint expressions $B = \{B_J\}$ and $C = \{C_J\}$ on the same variable set. A constraint C_J *with* $|J| = k$ is *k-compatible with B*, if all members of C_J k-satisfy B. C_J is said to be *k-*

complete for B, if any instantiation of a_J which k-satisfies B is a member of C_J. The constraint expression C is *k-compatible with B (k-complete for B),* if every C_J with $|J| = k$ is k-compatible with B (k-complete for B).

A constraint expression of order at least k is *k-consistent,* if, for any set X_H of k-1 variables, any instantiation a_H of X_H which (k-1)-satisfies C_H, and for any choice of k-th variable X_i, there exists an instantiation of X_i which combines with a_H to k-satisfy C_J, where $J = H \bigcup \{i\}$. A constraint expression of order at least k is *strongly k-consistent,* if it is j-consistent for all j, $1 \leq j \leq k$.

In the following we state Freuder's k-consistency algorithm. Given a constraint expression C in n variables and some integer k, $1 \leq k \leq n$, this algorithm yields a constraint expression B which is k-compatible with and k-complete for C and which is also strongly k-consistent. However, at first we must explain the notion of propagation between constraints.

If C_J and C_H are two constraints in a constraint expression C, then C_J and C_H are called *neighbours,* if $J \subset H$ and $|J| = |H| + 1$ (or equivalently $H \subset J$ and $|H| = |J| + 1$). To *locally propagate* the constraint C_J to a neighbouring constraint C_H, remove from C_H all a_H which do not satisfy C_J. To *globally propagate* a constraint C_J through a neighbouring constraint C_H first locally propagate C_J to C_H, and then, if anything was removed from C_H by the local propagation, globally propagate C_H through all its neighbours except C_J. To *propagate* a constraint C_J, globally propagate C_J through all its neighbours.

Algorithm:

Given a constraint expression C in n variables of order m and an integer $k \geq m$, the algorithm constructs a constraint expression B of order k which is k-compatible with C, k-complete for C, and which is strongly k-consistent.
(For proof of correctness of the algorithm see [Fr1]).

I. Let B be the empty set.

II. For all $J \subset I$ with |J|=1 let $B_J := C_J$ and $B := B \bigcup \{B_J\}$.

III. For i=2 to k do
For each $J \subset I$ with |J|=i do
$B_J := C_J$
insert B_J in B
locally propagate to B_J from all its neighbours in B
propagate B_J

Theorem:

Let d and k be integer constants.

For any order k constraint expression C in $n \geq k$ variables each of which can take on at most d values the above algorithm has worst case time complexity $O(n^k)$.

Proof:

The basic idea of the analysis is to calculate how often a constraint B_H can be locally propagated to a neighbouring constraint B_J. We claim that this can happen at most d^h times, where h=|H|.

For a proof recall when local propagation from B_H to B_J is triggered.

(i) It is triggered by the algorithm exactly once when B_H is added to the constraint expression (if |H|=|J|+ 1) or when B_J is added to the expression (if |H|=|J|-1).

(ii) It is triggered by the removal of one (or several) instantiations from B_H which leaves at least one instantiation in B_H. (Observe that if B_H becomes empty, the CSP represented by the constraint expression must be unsatisfiable.) As B_H can contain at most d^h instantiations this can happen at most d^h-1 times.

From (i) and (ii) one can conclude that B_H can be locally propagated to B_J at most d^h times.

Now it just remains to determine

a) the cost of locally propagating B_H to a neighboring B_J, and

b) how many neighbour pairs B_H , B_J there are.

For a) one needs to consider two cases:

(i) |J| = |H|-1 By a straightforward brute force method B_H can be propagated to B_J in time $PROP_{-}(h) = O(d^{2h-1})$.

(ii) |J| = |H|+ 1 By the same brute force method propagation can be performed in time $PROP_{+}(h) = O(d^{2h+1})$.

b) There are clearly $CONSTR(h) = \binom{n}{h}$ subsets of I of cardinality h, and each of these sets has $NEI_{-}(h) = h$ subsets of cardinality h-1 and $NEI_{+}(h) = n-h$ supersets of cardinality h+ 1. Note that for the purposes of this algorithm there are no neighbour pairs B_H, B_J with |H|=k and |J|=k+ 1. Thus $NEI_{+}(k)=0$.

If we let NO_OF_LPS(h) denote the maximum number of local propagations from a constraint B_H with |H|=h to a neighbouring constraint, then we get the following expression for the worst case time complexity of the algorithm:

$$\sum_{h=1}^{k} CONSTR(h) * NO\_OF\_LPS(h) * (NEI_{-}(h)*PROP_{-}(h) + NEI_{+}(h)*PROP_{+}(h))$$

Using the values derived above we get

$$\sum_{h=1}^{k-1} \binom{n}{h} d^h * (h*O(d^{2h-1}) + (n-h)*O(d^{2h+1})) + \binom{n}{k} d^k * k * O(d^{2k-1}).$$

This is bounded from above by

$$n*\sum_{h=1}^{k-1} \binom{n}{h} O(d^{3h+1}) + \binom{n}{k} * k * O(d^{3k-1}) = O(d^{3k+1} n^k).$$

As d is assumed to be constant this is $O(n^k)$.

Q.E.D.

Acknowledgements

I would like to thank Alan Mackworth and Gene Freuder for useful discussions on this topic.

References

[Fr1] Freuder, E.C.,
Synthesizing Constraint Expressions.
CACM 21, 11 (Nov. 1978), pp.958-966.

[Fr2] Freuder, E.C.,
A Sufficient Condition for Backtrack-free Search.
JACM 29, 1 (Jan. 1982), pp.24-32.

[Gas] Gaschnig, J.,
Performance Measurement and Analysis of Certain Search Algorithms.
CMU-CS-69-124 Tech. Report, Carnegie-Mellon Univ., 1979.

[H-S] Haralick, R.M. and Shapiro, L.G.,
The Consistent Labeling Problem: Part I.
IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no.2 (1979), pp.173-184.

[Mac] Mackworth, A.K.,
Consistency in Networks of Relations.
Artificial Intelligence 8, 19771, pp.99-118.

[M-F] Mackworth, A.K. and Freuder, E.C.,
The Complexity of some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems.
Univ. of British Columbia, Comp.Sci.Dept. Tech. Report 82-6, 1982.

[Mon] Montanari, U.,
Networks of Constraints: Fundamental Properties and Applications in Picture Processing.
Information Science 7, 1974, pp.95-132.

[Wal] Waltz, D.E.,
Generating Semantic Descriptions of Scenes with Shadows.
Tech. Report MAC AI-TR-271, MIT, Cambridge, MA, 1972.