COMPUTER ARCHITECTURE FOR

SOLVING CONSISTENT LABELING PROBLEMS

J. R. Ullmann *
R.M. Haralick **
L.G. Shapiro **

* University of Sheffield

**Virginia Polytechnic Institute

Abstract

Consistent labeling problems are a family of NP-complete constraint satisfaction problems such as school timetabling, for which a conventional computer may be too slow. There are a variety of techniques for reducing the elapsed time to find one or all solutions to a consistent labeling problem. In this paper we discuss and illustrate solutions consisting of special hardware to accomplish the required constraint propagation and an asynchronous network of intercommunicating computers to accomplish the tree search in parallel.

Index Terms

Consistent labeling.
NP complete.
Backtrack search.
Network computer.
Asynchronous sequential circuits.
Programmable logic arrays.
Array processor.

Special Architecture for Consistent Labeling Problems

Introduction

School-timetabling, subgraph isomorphism, graph coloring, propositional theorem proving, and scene labeling problems can be formulated as special cases of the consistent labeling problem [1]. The consistent labeling problem is NP-complete, which means that in the worst case we may have to resort to exhaustive enumeration in order to find a solution, and the time needed for this enumeration may increase exponentially with the number of variables. Despite the possibility of this combinatorial explosion, problems such as school time tabling have to be solved in practice. Smart searches using look-ahead operators to perform tree pruning have been devised to mitigate the combinatorial explosion. This helps, but look-ahead operators cannot be guaranteed to keep program execution time within modest bounds.

In the present paper we explore several possibilities for special architecture using parallelism to reduce the elapsed time for solving consistent labeling problems. To avoid brute force enumeration, we use a constraint propagation tree-pruning technique in which there is an inevitable sequential part that can make the quest for parallelism non-trivial.

El-Dessouki and Huen [2] have considered computer network architecture for NP-complete problems of determining extremal values of objective functions. Consistent labeling problems are more primitive in that their solutions have to satisfy relational constraints that constitute a boolean objective function. Restriction to the binary case broadens the possiblities for special architecture for solving consistent labeling problems. However, in this paper we consider the general n-ary relational constraint case.

Earlier papers on special architecture for solving the consistent labeling problem include Cherry and Vaswani (1961) who had actually built special architecture for a boolean satisfiability problem [3] (which is a consistent labeling problem [1]). We believe, however, that the general possibilities of using special architecture to soften the practical effects of the combinatorial explosion have not previously been explored adequately. Schmidt and Strohlein [4] remark that "recent developments in computer technology and software engineering have not yet reached the area of time-table programming."

A Formulation of the Consistent Labeling Problem

The consistent labeling problem was formulated in [1] as an N-ary constraint satisfaction problem for fixed integer N. In this paper, we generalize the formulation to allow for multi-dimensional constraints. Our new formulation will also aid in the development of the hardware designs to be presented.

Let U be a set of objects called units, and L be a set of possible labels for those units. Let $T \subseteq \{f \mid f \subseteq U\}$ be the collection of those subsets of units from U that mutually constrain one another. That is, if $f = \{u_1, u_2, \ldots, u_K\}$ is an element of T, then not all possible labelings of $u_1,\ldots,u_K$ are legal labelings. Thus there is at least one label assignment $l_1, l_2, \ldots, l_K$ so that u_1 having label l_1, u_2 having label l_2, $\ldots$, u_K having label l_K is a forbidden labeling. T is called the unit constraint set.

Finally, let $R \subseteq \{g \mid g \subseteq U \times L,$ g single-valued, and $Dom(g) \in T\}$ be the set of unit-label mappings in which constrained subsets of units are mapped to their allowable subsets of labels. If $g = \{(u_1,l_1),(u_2,l_2), \ldots, (u_k,l_k)\}$ is an element of R, then $u_1, u_2, \ldots, u_k$ are distinct units, $\{u_1,u_2, \ldots, u_k\}$ is an element of T meaning $u_1, u_2, \ldots, u_k$ mutually constrain one another, and u_1 having label l_1, u_2 having label l_2, ..., and u_k having label l_k are all simultaneously allowed.

In the consistent labeling problem, we are looking for functions that assign a label in L to each unit in U and satisfy the constraints imposed by T and R. That is, a consistent labeling is one which when restricted to any unit constraint subset in T yields a mapping in R. In order to state this more precisely, we first define the restriction of a mapping. Let $h:U \rightarrow L$ be a function that maps each unit in U to a label in L. Let $f \subseteq U$ be a subset of the units. The restriction $h|_f$ (read " h restricted by f ") is defined by $h|_f = \{(u,l) \in h \mid u \in f\}$. With this notation, we define a consistent labeling as follows.

A function $h:U \rightarrow L$ is a consistent labeling if and only if for every $f \in T$, $h|_f$ is an element of R. (1)

A Simple Example

Suppose the inputs to the problem are as follows:

U = {1, 2, 3, 4, 5}
L = {a, b, c}
T = { {1}, — unnary constraint
{1, 2}, {2, 5}, — binary constraints
{1, 3, 4}} — ternary constraint
R = { {(1,a)},{(1,b)}, — unary constraint
{(1,a),(2,a)}, {(1,a),(2,b)}, {(1,b),(2,b)}, {(2,a),(5,a)}, {(2,b),(5,c)}, — binary constraints
{(1,a),(3,a),(4,c)}, {(1,b),(3,a),(4,a)}} — ternary constraints

Then h = {(1,a) (2,a) (3,a) (4,c) (5,a)} is a consistent labeling. To see this note that

$h|_{\{1\}} = \{(1,a)\}$, $h|_{\{1,2\}} = \{(1,a),(2,a)\}$, $h|_{\{2,5\}} = \{(2,a),(5,a)\}$, and $h|_{\{1,3,4\}} = \{(1,a),(3,a),(4,c)\}$ are all elements of R.

Some Examples of Real Consistent Labeling Problems

A School Time-Tabling Problem We consider the scheduling of lessons over a fixed period such as one week, assuming that instructors have already been assigned to lessons. The problem is to assign to each lesson, a time and classroom satisfying the constraints that 1) any pair of lessons attended by the same instructor or students must be at different times and 2) no distinct pair of lessons is assigned to the same time and room. This problem fits the consistent labeling model as follows.

U is the set of all lessons. If there are three history lessons, then there are three separate elements of U. L is a set of pairs of the form (time, classroom) which includes all possible lesson times and for each time, all possible classrooms available at that time. The unit constraint set T consists of unary constraints and binary constraints. The unary constraints are for those lessons that a priori cannot be scheduled in a particular (time, classroom) pair. The binary constraints consist of all pairs of distinct lessons since 1) these are constrained not to meet in the same classroom at the same time and 2) a subset of these are constrained not to meet at the same time. From this we get

$$T = T_1 \cup T_2$$

where $T_1 = \{\{u\} \mid u \in U$ and u cannot be scheduled at time t, classroom c for some pair $(t,c) \in L\}$

and $T_2 = \{\{u_1,u_2\} \mid u_1,u_2 \in U$ and $u_1 \neq u_2\}$. Furthermore, $R = R_1 \cup R_{21} \cup R_{22}$ where

$R_1 = \{\{(u,l)\} \mid \{u\} \in T_1$ and a priori knowledge says that label l can be assigned to u$\}$.

$R_{21} = \{\{(u_1,l_1),(u_2,l_2)\} \mid \{u_1,u_2\} \in T_2$ and $l_1 \neq l_2\}$,

and $R_{22} = \{\{(u_1,l_1),(u_2,l_2)\} \mid \{u_1,u_2\} \in T$, there exists a person who must attend both u_1 and u_2, and $time(l_1) \neq time(l_2)\}$.

Other constraints can be added to the model as required. For instance if there are pairs of lessons that must be given in consecutive hours, we can define

$T_3 = \{\{u_1,u_2\} \mid u_1,u_2 \in U$ and u_2 must be scheduled the hour after $u_1\}$

and $R_3 = \{\{(u_1,l_1),(u_2,l_2)\} \mid \{u_1,u_2\} \in T_3$ and $time(l_2) = time(l_1)$ plus one hour$\}$.

A Three-Dimensional Packing Problem The problem is to fit a given collection of solid objects into a box. Some of these objects may be placed on top of others. For simplicity, we partition the inside of the box into unit cubes, designated by the coordinates of their centers, and assume that all objects are made up of uniquely named unit cubes whose sides will always be parallel to the sides of the box.

For this problem, we let U be the set of unit cubes that make up the objects and let L be the set of possible (x,y,z) coordinates specifying those positions in the box that can coincide with unit cube centers. The unit constraint set T consists of two kinds of constraints. First, for each object, there is a set of unit cubes representing that object. For each such element of T, there are in R, mappings corresponding to each positional shift and each allowed rotation of the object represented by that element. If, for example, we wish to prevent a particular object from being packed upside down, then we allow in R no mapping of the unit cubes of this object to coordinates which place the object in an upside down position. The second kind of constraint in T is that all pairs of units are constrained not to have the same label. Thus we have

$T = T_1 \cup T_2$ where

$T_1 = \{\{u_1, u_2, \ldots, u_{n_i}\} \mid u_1, u_2, \ldots, u_{n_i}$ are the unit cubes of the i'th object$\}$ and

$T_2 = \{\{u_j, u_k\} \mid u_j, u_k \in U,\ j \neq k\}$

for unit constraints. For allowed mappings, we have

$R = R_1 \cup R_2$ where

$R_1 = \{\{(u_1,l_1),(u_2,l_2),\ldots,(u_{n_i},l_{n_i})\} \mid$ $\{u_1,u_2,\ldots,u_{n_i}\} \in T$ and positioning the center of cube u_1 at location l_1, the center of cube u_2 at location $l_2, \ldots,$ the center of cube u_{n_i} at location l_{n_i} represents an allowable placement of object i in the box $\}$ and

$R_2 = \{\{(u_j,l_j),(u_k,l_k)\} \mid \{u_j,u_k\} \in T$ and $l_j \neq l_k\}$.

We can determine how to pack the box by finding a consistent labeling. If no consistent labeling can be found, then it is not possible to pack all of the given objects in the box.

Further examples of consistent labeling problems have been formulated mathematically elsewhere [1].

A Constraint Propagation Algorithm for Consistent Labeling Problems

Various algorithms have been theoretically and experimentally explored in [1] and [5]. In the present paper, we will consider just one simple and computationally efficient approach.

In the standard backtracking tree search to find a consistent labeling, a mapping (initially empty) is extended by adding a new unit-label pair so that the partial mapping defined so far satisfies all the constraints. Such a search can investigate many useless branches of the tree when a partial labeling is consistent alone, but actually has no consistent extension to the remainder of the units. Many times such partial mappings having no extensions can be discovered with a small amount of constraint propagation work. In these cases, the subtree to be searched below the partial labeling can be eliminated as not containg any consistent labeling, and the tree search can continue by trying the next label for the current unit or, if there are no labels left, then backtracking.

The constraint propagation technique is the natural generalization to forward checking discussed in [5] and corresponds exactly to forward checking for the case of binary constraints. The constraint propagation is done immediately after a label is instantiated for the current unit. It is the constraint of this assignment combined with the older label assignments which is propagated forward further constraining label possiblities for future units. The unit constraints participating in the propagation are those unit constraints $f \in T$ containing the current unit and having at least one of their units be a future unit. We call the set of constraints participating in the propagation Q. The constraint propagation consists of restricting each unit's current or future possible label assignment to exactly those labels common to the set of labels permitted by each of the constraints participating in the propagation. If as a result of the propagation, there is no label for some unit, then the current partial labeling cannot be completed to a consistent labeling and the tree search must continue by instantiating the next label for the current unit or, if there is none, to backtrack.

We denote by UF (future units) the set of units not instantiated. These are the ones having no labels assigned so far. The set Q of constraints participating in the propagation is defined by $Q = \{f \in T \mid (f \cap UF) \neq \emptyset$ and f contains the current unit$\}$.

We denote by H, $H \subseteq U \times L$, the relation of possible label assignments currently permitted for each unit. If $u \in UF^c$ then

(1) $(u,l) \in H$ implies that l is the instantiated label for unit u

(2) $(u,l) \in H$ and $(u,n) \in H$ imply $n = l$ since only one label can be assigned to an instantiated unit

If $u \in UF$, then u is a unit yet to be instantiated and $H(u)$ is the set of labels still permitted for unit u. The way H and the constraint propagation work in the tree search, it is guaranteed that H is defined everywhere ($H(u) \neq \emptyset$ for every u).

The constraint propagation replaces H by a relation that is possibly smaller and certainly no larger. The new H is given by

$$H \longleftarrow \bigcap_{f \in Q} R(H,f)$$

where

$R(H,f) = \{(u,l) \in U \times L \mid$
either 1) $u \notin f$
or 2) $u \in f$, and there exists $g \in R$ with $(u,l) \in g$, satisfying $dom(g) = f$ and
$g \subseteq H \}$.

Thus $R(H,f)$ is the set of unit-label pairs that the constraint f does not rule out. By the definition of consistent labeling, a function $h:U \rightarrow L$ is a consistent labeling if and only if for every $f \in T$, $h|_f$ is an element of R. From this it immediately follows that a function h satisfies $h = \bigcap_{f \in T} R(h,f)$ if and only if h is a consistent labeling. This relationship for h is interesting in that it shows that one part of h does constrain another part. From this relationship it follows that the single-valued completion of any incomplete labeling H must lie in $\bigcap_{f \in T} R(H,f)$. Thus, if this intersection is not defined everywhere, then the partial labeling H so much constrains the labels of the future units (those not yet having a fixed label) that there is some as yet future unit that has no possibilities left for its label.

To illustrate the backtracking algorithm employing the look-ahead technique, we give a recursive procedure. The inputs to procedure TREESEARCH are UF: the set of units requiring labels (initially all the units), T: the unit constraint set, R: the allowable unit-label mappings. and H: the partial or incomplete labeling (initially $H = \{(u,l) \in U \times L \mid (u,l)$ is not ruled out by any constraint in $R\}$. Note that all unary constraints have been removed from T and used to produce the initial H.

The predicate ISEMPTY returns true if its argument is an empty set and false otherwise. The predicate DEFINED-EVERYWHERE returns true if its argument is a set of unit-label pairs including every unit and false otherwise. The procedure OUTPUT outputs a mapping. The function DELETEFIRST removes and returns the first element of its argument set. The function RESTRICT inputs a binary relation H, a unit u and a label l and returns a new binary relation consisting of all pairs (v,m) such that if $v = u$, then $m = l$. Procedure TREESEARCH is given below.

```
procedure TREESEARCH (UF,T,R,H)
u:= DELETEFIRST(UF);
Q:={f ∈ T | f  FU ≠ Ø and u ∈ f}
S:= {l | (u,l) ∈ H};
while not ISEMPTY(S) do
   begin
      l:= DELETEFIRST(S);
      H_u,l = RESTRICT(H,u,l)
      H' = ∩ (f ∈ Q) R(H_u,l ,f)
      if DEFINED EVERYWHERE(H')
      then if SINGLE VALUED(H')
         then OUTPUT(H')
         else call TREESEARCH(UF,T,R,H')endif
      endif;
   end;
return;
end TREESEARCH
```

For the simplified example of Section 2, we have initially

$UF = U = \{1,2,3,4,5\}$,

$T = \{\{1,2\},\{2,5\},\{1,3,4\}\}$
(the unary constraints have been removed, since they will be used to determine the initial H),

$R = \{\{(1,a),(2,a)\},\{(1,a),(2,b)\},\{(1,b),(2,b)\},\{(2,a),(5,a)\},\{(2,b),(5,c)\},\{(1,a),(3,a),(4,c)\},\{(1,b),(3,a),(4,a)\}\}$
(the unary unit-label pair sets have been removed here also), and

$H = \{(1,a),(1,b),(2,a),(2,b),(3,a),(4,a),(4,c),(5,a),(5,c)\}$
(the initial R was used to determine the legal labels for each unit.)

In the first call to TREESEARCH, u is set to 1, UF to $\{2,3,4,5\}$, and S to $\{a,b\}$. Next l is set to a, S reduced to $\{b\}$, and $H_{u,l}$ becomes

$\{(1,a),(2,a),(2,b),(3,a),(4,a),(4,c),(5,a),(5,c)\}$.
Now the constraint propagation calculates

$R(H_{u,l},\{1,2\}) = \{(1,a),(2,a),(2,b),(3,a),(4,a),(4,c),(5,a),(5,c)\}$

$R(H_{u,l},\{2,5\}) = \{(1,a),(2,a),(2,b),(3,a),(4,a),(4,c),(5,a),(5,c)\}$

$R(H_{u,l},\{1,3,4\} = \{(1,a),(2,a),(2,b),(3,a),(4,c),(5,a),(5,c)\}$.

Thus the intersection H' becomes

$H' = \{(1,a),(2,a),(2,b),(3,a),(4,c),(5,a),(5,c)\}$.
Since H' is defined everywhere but not single-valued, TREESEARCH is called again. This time we have

$UF = \{2,3,4,5\}$,

T and R remain the same, and

$H = \{(1,a),(2,a),(2,b),(3,a),(4,c),(5,a),(5,c)\}$.

In this activation, u becomes 2, UF becomes {3,4,5}, S becomes {a,b}, l becomes a, S is reduced to {b}, and $H_{u,l}$ becomes

$\{(1,a),(2,a),(3,a),(4,c),(5,a),(5,c)\}$. Now the constraint propagation calculates

$$R(H_{u,l},\{2,5\}) = \{(1,a),(2,a),(3,a),(4,c),(5,a)\}.$$

$R(H_{u,l},\{1,3,4\})$ is not calculated since unit 2 is not an element of {1,3,4}. The intersection H' becomes {(1,a),(2,a),(3,a),(4,c),(5,a)}. It is defined everywhere and single-valued, so a consistent labeling has been found. The procedure will go on to find a second consistent labeling also.

In the next section we discuss the way in which special hardware can implement the constraint propagation as part of the tree search extended by a single CPU and the way in which multiple CPU's can be connected in a network to execute the tree search in parallel.

Hardware for Solving the Consistent Labeling Program

Hardware can be designed to speed up the tree search by using multiple CPU's and to speed up the constraint propagation by using special purpose memories. Section 5.1 discusses how the constraint propagation can be implemented in hardware, and Section 5.2 discusses how multiple CPU's can be used to execute the tree search in parallel.

Constraint Propogation Hardware

In the hardware implementation of the constraint propagation discussed here and shown in Figure 1, sets and relations will be represented by bit vectors and arrays. Suppose that there are N units in U. The set of future units will be represented by a bit vector UF of length N. Formally, UF: U —> {0,1} is defined by UF(u) = 1 if u is not instantiated and UF(u) = 0 if u is instantiated. Likewise, a unit constraint element f is a bit vector, f(u) being 1 for all units u which participate in the unit constraint. The relation H storing currently allowed labels for units is represented by a bit array of N rows by K columns, where K is the number of possible labels. H: U x L —> {0,1} is defined by H(u,l) = 1 if label l is still permitted for unit u and otherwise.

The unit constraint set T is a bit array whose rows are the bit vectors f, each row representing some particular unit constraint. The number of rows in T is the number of unit constraints. T is stored in a special memory called the unit constraint memory, whose words are the rows of T and which has associative memory operations. The associative memory operations consist of a load operation and a pop operation. Load inputs the bit vector UF which has ones in those positions corresponding to uninstaniated units. After a load, a pop operation outputs into the buffer register the next unit constraint bit vector in memory having some bit position with value 1 and not covered by a 1 in the corresponding bit position of the bit vector UF which was last loaded. Associated with the pop operation is an empty status signal to indicate whether there was in fact a word which was popped or whether there were no words left to be popped. This status signal is like a stack empty status signal.

If the empty status signal indicates that a constraint vector was popped and loaded into a buffer register, the operation continues in a special memory called the unit label constraint memory, which accepts for its inputs the unit constraint vector f just popped and the current labeling array H. The memory returns R(H,f), a bit array of unit label pairs like H, and ANDS this into the register in which H resides. If a combinational logic check succeeds indicating that every unit in the modified H has some label not yet ruled out, then the cycle continues retrieving the next constraint from the unit constraint memory. If the combinational logic check fails, then the tree search must continue by instantiating the next label for the current unit or, if there are none, by backtracking.

The combinational logic check, which produces this signal called the propagation status indication, is itself simple. Since each row of H holds the labels not yet ruled out for the unit associated with the row, the combinational logic check must OR the values across the row and then AND these ORed values down its column. If the resulting bit is a 1, the propagation status indicator is a 1. If the status is a 0, then there is some unit having all labels ruled out and the tree search must continue in the next subtree.

Network Computer Implementation

Even after setting the problem up with constraint propagation, the execution time may be intolerably slow for man-machine interactive school timetabling or for real-time control of an automatic packing machine. To reduce elapsed time for finding consistent labelings, we can subdivide the search tree into N subtrees and use N separate processors to search these subtrees simultaneously, with no need for any synchronization between these fully independent processors. Each processor could have its own constraint propagation hardware as discussed in the previous section as well.

A specific method for this is to partition the label set L into N subsets, $L_1,\ldots,L_N$. The first processor would try to solve the consistent labeling problem, restricting the label assigned to the first unit to come from L_1. The second proces-

sor would try to solve the consistent labeling problem restricting the label of the first unit to come from L_2, and so on. Each processor would, to avoid memory access delays, have its own memory containing copies of all required data and code, and would execute the backtrack algorithm of section IV, thus searching a disjoint subtree.

Unfortunately this simple idea may not make optimal use of the N processors to find all consistent labelings in minimal elapsed time. For it is the case that even if each of the N subsets contains the same number of possible labels, the N processors may not all take an equal amount of time to complete their subtree search, exactly because of differences in the effectiveness of tree pruning. Practical experience with algorithms of this type suggests that the elapsed time for one processor may turn out to be many times greater than that for another. Processors that have finished their work may wait idly for others to finish. Thus by using N parallel processors we may not succeed in reducing the overall elapsed time by a factor of N.

Overall elapsed time could be further reduced by interconnecting the N processors in a computer network as in [2]. One of the many possible operating policies is that when a processor completes its subtree search it interrogates all other processors that are still searching and then takes over half of the remaining search of the processor whose search is furthest from completion, leaving this processor to complete only the other half of its subtree search. When this network starts operating, with all processors searching (hopefully) equal-sized subtrees, there are at first no delays due to exchange of messages between processors. When more and more processors finish searching subtrees, more and more messages are exchanged, and this eventually constitutes a significant overhead. To prevent this overhead from exploding, we impose a restriction that no processor ever starts searching a subtree of less than a threshold size: if no subtree greater than or equal to this size is available for a processor then this processor becomes idle and is in effect deleted from the network. The threshold size should of course be chosen so as to minimise overall elapsed time.

This networking policy depends on splitting the subtree whose search is furthest from completion. How far a processor is from completion of a subtree search is easily determined by the number of not yet instantiated units.

Simulation of a variety of network architectures uniformly indicates that all other things being equal, (1) processors should execute the search in a depth first rather than breadth first manner so that there are as large subtrees as possible which the busy processors can give to a free processor, (2) busy processors should first hand off subproblems to the free processors least centrally located in the network wherever there is a choice.

REFERENCES

[1] R. M. Haralick and L. G. Shapiro, "The Consistent Labelling Problem, Part 1", IEEE Trans. Pattern Anal. Machine Intell. Vol. PAMI-1, pp. 173-184, April 1979, Part II, IEEE Trans. Pattern Anal. Machine Intell., Vol. PAMI-2, No. 3, pp. 193-203, May 1980.

[2] O. I. El-Dessouki and W. H. Huen, "Distributed Enumeration on Network Computers", IEEE Trans. Comput., Vol. C-29, pp. 818-825, September 1980.

[3] C. Cherry and P. K. T. Vaswani, "A New Type of Computer for Problems in Propositional Logic, with Greatly Reduced Scanning Procedures", Information and Control, Vol. 4, pp. 155-168, September 1961.

[4] G. Schmidt and T. Strohlein, "Timetable Construction - An Annotated Bibliography", The computer Journal (British Computer Society), Volume 23, pp. 307-316, Novermber 1980.

[5] R. M. Haralick & G. L. Elliott, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", To be published in Artificial Intelligence.

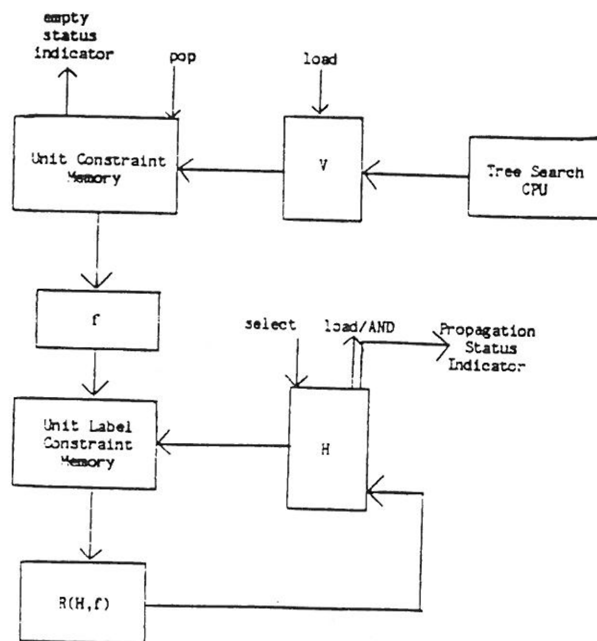

Figure 1 illustrates a block diagram of the constraint propagation hardware.