AN EFFICIENT RELEVANCE CRITERION FOR MECHANICAL THEOREM PROVING*

David A. Plaisted
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

ABSTRACT

To solve problems in the presence of large knowledge bases, it is important to be able to decide which knowledge is relevant to the problem at hand. This issue is discussed in [11]. We present efficient algorithms for selecting a relevant subset of knowledge. These algorithms are presented in terms of resolution theorem proving in the first-order predicate calculus, but the concepts are sufficiently general to apply to other logics and other inference rules as well. These ideas should be particularly important when there are tens or hundreds of thousands of input clauses. We also present a complete theorem proving strategy which selects at each step the resolvents that appear most relevant. This strategy is compatible with arbitrary conventional strategies such as Pl1-deduction, locking resolution, etc. Also, this strategy uses nontrivial semantic information and "associations" between facts in a way similar to human problem-solving processes.

I RELEVANCE FUNCTIONS

Definition. A support set for a set S of clauses is a subset Ti of S such that S-Ti is consistent. A support class for S is a set {Ti, ..., Tn} of support sets for S.

Definition. A (resolution) proof of C from S is a sequence C1,C2,...,Cn of clauses in which Cn is C and each clause Ci is either an element of S (an input clause) or a resolvent of two preceding clauses in S. (Possibly both parents of Ci are identical.) The length of such a refutation is n. A refutation from S is a proof of NIL (the empty clause) from S.

Definition. A relevance function is a function R which, given a set S of clauses, a support class T for S, and an integer n, maps onto a subset R(S, T) of support sets for S. In fact, the derivation from R(S, T) will be a subderivation of the derivation from S, for all relevance functions considered here. Thus if there is a length n Pl1-deduction from S, there will be a Pl1-deduction of length n or less from Rn(S, T), and similarly for other complete strategies.

Definition. Suppose S is a set of clauses. The connection graph of S, denoted G(S), is the graph whose nodes are the clauses of S, and which has a directed arc from C1 to C2 labeled (L1, L2) if there are literals L1 of C1 and L2 of C2 such that L1 and L2 are unifiable. Such graphs have been introduced and discussed in [2]. Note that there will also be an arc labeled (L2, L1) from C2 to C1 in the above case.

Definition. Suppose C1 to Cn in G(S) is a sequence C1,C2,...,Cn of clauses of S such that there is an arc from C1 to Cn in G(S), for 1≤i≤n. Also, the length of the path is n.

Definition. A path from C1 to Cn in G(S) is a sequence C1,C2,...,Cn of clauses of S such that there is an arc from C1 to Cn in G(S), for 1≤i≤n. Also, the length of the path is n.

Definition. The distance d(C1, C2) between C1 and C2 in G(S) is the length of the shortest path from C1 to C2 in G(S), and ∞ if no such path exists.

Definition. If S is a set of clauses, T is a support class for S, and n is a nonnegative integer, then Qn(S, T) is {C ∈ S: d(C, Ti)≤n in G(S) for all Ti in T}, where d(C, Ti) is min{d(C, D):D ∈ Ti}.

Intuitively, if d(C1, C2) is small, C1 and C2 are "closely related." Also, Qn(S, T) is the clauses that are closely related to all the support sets. Typically we will know that several clauses are essential to prove a theorem, and such clause by itself can be made into a support set.

Definition. A set S of clauses is fully matched if for all C ∈ S, for all literals L1, L2, there exists C2 ∈ S and L2′ ∈ S such that L1 and L2′ are unifiable.

* This research was partially supported by the National Science Foundation under grant MCS-79-04897.
Definition. \( R(S,T) \) is the largest fully matched subset of \( Q^n(S, T) \). Thus we obtain \( R(S, T) \) from \( Q^n(S, T) \) by repeatedly deleting clauses containing "unmatched" literals. This definition is not ambiguous, since if \( Q^n(S, T) \) contains more than one nonempty fully matched subset, then \( R(S, T) \) is the union of all such subsets.

Theorem 1. The function \( R \) is a relevance function. That is, if there is a length \( n \) refutation from \( S \), and \( T \) is support class, then there is a refutation from \( R(S, T) \) of length \( n \) or less. In fact, there is such a refutation from \( R(S, T) \).

Proof. Assume without loss of generality that \( \text{NIL} \) appears only once in the refutation and that every clause in the refutation contributes to the derivation of \( \text{NIL} \). Let \( S_1 \) be the set of input clauses appearing in the refutation. Then \( S_1 \) is connected, intersects all the support sets, and has at most \( n \) elements. Using properties of binary trees we can show that \( S_1 \) has at most \( p(n) \) elements. Note that \( R(S, T) \) is a "global" relevance criterion. That is, it depends in a non-trivial way on all the input clauses and on interactions between all the support sets in \( T \).

Example 1

Let \( S \) be the following set of clauses:

\[
P_1 P_2 P_3 P_4 P_5 P_6 P_7\]

Here \( P_1 \) indicates \( \{P_1, P_1'\} \), i.e., \( P_1 \lor P_1' \) etc. (at the end.

Let \( T_1 \) be \( \{\{P_1\}, \{P_2\}\} \), and \( T_2 = \{\{P_5\}\} \). Then \( R_1(S, T_1) = R_2(S, T_2) = R_3(S, T) = \{\{P_1, P_2\}, \{P_3, P_4\}\} \) which is in fact a minimal inconsistent subset of \( S \).

Example 2

For a second example, let \( S \) be the following:

\[
\begin{align*}
\text{IN}(a, \text{box}) \\
\text{IN}(x, \text{box}) &\supset \text{IN}(x, \text{room}) \\
\text{IN}(x, \text{room}) &\supset \text{IN}(x, \text{house}) \\
\text{IN}(x, \text{house}) \\
\text{ON}(x, \text{box}) &\supset \text{IN}(x, \text{box}) \\
\text{ON}(x, \text{street}) &\supset \text{IN}(x, \text{house}) \\
\text{IN}(x, \text{house}) &\supset \text{IN}(x, \text{village}) \\
\text{IN}(\text{house, box}) \\
\text{AT}(\text{house, street}) \\
\text{ON}(x, \text{box}) \\
\text{ON}(c, \text{street}) \\
\text{IN}(d, \text{village})
\end{align*}
\]

Let \( T_1 \) be \( \{\text{IN}(a, \text{box})\} \) and let \( T_2 \) be \( \{\text{IN}(x, \text{house})\} \). Also, \( T = \{T_1, T_2\} \). Then \( R_1(S, T) = R_2(S, T) = \emptyset \) but \( R_3(S, T) = \{\text{IN}(a, \text{box})\}, \{\text{IN}(x, \text{box})\}, \{\text{IN}(x, \text{room})\}, \{\text{IN}(x, \text{house})\}\). This is a minimal inconsistent subset of \( S \). Here "box", "room", "house", "a" et cetera are constants and \( x \) is a variable.

Note that we cannot always guarantee that this relevance criterion will yield minimal inconsistent sets as in these examples.

III ALGORITHMS

Suppose \( S \) is a set of clauses. Let \( ||S|| \) be the length of \( S \) in characters when written in the usual way. Let \( \text{Lits}(S) \) be the sum over all clauses \( C \) in \( S \), of the number of literals in \( C \).

If \( S \) is a set of propositional clauses and \( ||S|| = n \), then \( G(S) \) may have \( O(n^2) \) arcs. However, we can construct a modified version \( G_1(S) \) of \( G(S) \) which has the same distances between clauses as \( G(S) \) does but which has only \( O(n) \) arcs. The idea is to reduce the number of arcs as follows: Suppose \( C_1, \ldots, C_k \) are the clauses containing \( L \) and \( D_1, \ldots, D_k \) are the clauses containing \( L \). Then we add a node \( N_L \) and arcs as follows in \( C_k(S) \):

The numbers indicate the lengths of the arcs. Similarly, there are arcs of the form \( \sigma, 0, -1, 1, \), and \( 0 \).

Although \( G_1(S) \) is not a connection graph, and has arcs of length 0 and 1, it preserves distances between clauses as in \( G(S) \). Using this modified connection graph, we have linear time algorithms to do the following, if \( S \) is a set of propositional clauses, \( T = \{T_1, \ldots, T_k\} \) is a support set, and \( n \) is a positive integer:

1. Construct \( G_1(S) \) from \( S \).
2. Find \( \{C \in S : d(C, T_i) \leq n\} \).
3. Given \( Q(S, T) \) for support class \( T \), to find \( R(S, T) \).

Since step 2 must be performed \( |T| \) times to obtain \( Q(S, T) \), the total algorithm to obtain \( R(S, T) \) requires \( O(|T| 

The algorithm to find \( \{C \in S : d(C, T_i) \leq n\} \) is a simple modification of standard shortest-path algorithms. For a presentation of these standard algorithms see [3]. This can be done in linear time because the edge lengths are all 0 and 1. We compute \( R_n(S, T) \) as follows:

Definition. If \( S \) is a set of clauses, let \( M(S) \) be the largest fully matched subset of \( S \). Note that \( R_n(S, T) = M(Q_n(S, T)) \).

The following algorithm \( M_1 \) computes \( M(S) \) for set \( S \) of propositional clauses in linear time. This algorithm can therefore be used to compute \( R_n(S, T) \) if \( S \) is a set of propositional clauses. Note that \( t \) is a push-down stack.
procedure M₁(S);
  t ← empty stack;
  for all L such that L ∈ C or L ∈ C for some C ∈ S do
    clauses(L) ← (C ∈ C) \ L;
    count(L) ← |clauses(L)| od;
  for all C ∈ S do
    member(C) ← TRUE;
    for all L ∈ C do
      if count(L) = 0 then push C on t; member(C) ← FALSE fi od;
  while t not empty do
    pop C off t;
    count(L) ← count(L) − 1;
    if count(L) = 0 then
      for all C ∈ L do
        if member(C) then push C on t; member(C) ← FALSE fi
      fi od;
  return((C ∈ C: member(C) = TRUE));
end M₁;

If S is a set of first-order clauses, then G(S) can be constructed in O(Lits(S) * |S|) time using a linear unification method [4]. This bound results because a unification must be attempted between all pairs of literals of S. The number of edges in G(S) is at most Lits(S)². Given G(S), we can find \{C ∈ S: d(C, T) ≤ n\} in time proportional to the number of edges in G(S) (since all edge lengths are one). Also, given Q(S, T), we can find R(S, T) in time proportional to the number of edges in G(S) by a procedure similar to M₁ above. The total time to find R(S, T) is therefore 0(|T|²Lits(S)² + |S|²Lits(S)). By considering only the predicate symbols of literals, the propositional calculus algorithms can be used as a linear time preprocessing step to eliminate some clauses from S. An interesting problem is to compute R(S, T) efficiently for many values of n at the same time. There are methods for doing this, but we do not discuss them here.

IV REFINEMENTS

A. Connected Components

Proposition 1. If there is a length n refutation from S and T is a support class for S, then there is a length n refutation from one of the connected components of R₁(S, T). Also, the connected components can be found in linear time [3].

B. Iteration

Definition. If T = \{T₁, T₂, ..., Tₖ\} is a sup-
port class for S and S₁ is a subset of S then Tₛ₁(T restricted to S₁) is {T₁ ∩ S₁, T₂ ∩ S₁, ..., Tₖ ∩ S₁}. It may be that Tₛ₁(S, T) ≠ Tₛ₁(S, T) for some Tₛ₁(S, T). Also, Rₛ₁ is the limit of the sequence R₁(S, T), R₂(S, T), ..., Rₙ(S, T).

Proposition 2. Rₛ₁(S, T) ≤ Rᵱ(S, T) for i > 1. Therefore the limit R(S, T) exists. Also, Rₙ(S, T) can be computed in at most |S| iterations of Rₛ₁(S, T). Can it be computed more efficiently than this?

Theorem 2. If there is a length n refutation from S, and T is a support class for S, then there is a length n refutation from one of the connected components of R(S, T).

Proposition 3. For all i ≥ 0 there exist n, S and T such that Rᵱ⁺₁(S, T) = Rᵱ(S, T) ≠ Rᵱᵱ(S, T). Thus this computation can take arbitrarily long to converge.

Proof. Let n = 2, S = \{P₁, P₂, ..., Pₙ\} U \{Pᵢ₊₁ \ Pᵢ : 1 ≤ i < k\}. Let T = \{T₁, T₂, ..., Tₙ\} where Tᵢ = \{Pᵢ \ Pᵢ₊₁ : i = j (mod 3)\} \ {Pᵢ₊₁ \ Pᵢ : 1 ≤ i < k (mod 3)}. Then R₁(S, T) = \ ø if 2a < k but R₂(S, T) ≠ \ ø if 2a < k.

C. Selecting Support Clauses

We now give another approach.

Theorem 3. Suppose there is a length n refu-
tation from S and T = \{T₁, T₂, ..., Tₖ\} is a support class for S. Then there exist clauses C₁ ∈ T₁, 1 ≤ i ≤ k, such that

a) \{C₁, C₂, ..., Cₖ\} ⊆ Rᵱ⁺₁(S, \{C₁\}, \{C₂\}, ..., \{Cₖ\})

b) there is a length n refutation from Rᵱ(S, \{C₁\}, ..., \{Cₖ\}).

Thus it is possible to select particular clauses from the support sets and use them to define R. There may be many sets Rᵱ(S, \{C₁\}, ..., \{Cₖ\}) satisfying condition a) above, but they may be smaller than the connected components of Rᵱ(S, T). It is possible to construct examples having this property. Therefore it may help to use the above sets rather than R(S, T) when searching for refutations. Another advantage is that it is possible to examine the clauses \{Cᵢ \ i ≤ k\} in some heuristically determined order.
above approach is useful when the support sets \( T_i \) are not known in advance but are obtained one clause at a time.

We now give a recursive procedure "rel3" for generating all the sets as in Theorem 3. The idea is to order the \( T_i \) so as to reduce the branching factor as much as possible near the beginning of the search. Thus we use the principle of "least commitment." This procedure has as input sets \( S_l \) and \( S \) of clauses, integer \( n \), and support class \( T \) for \( S \). Let \( S_1 = \{ D_1, \ldots, D_i \} \) and \( T = \{ T_1, \ldots, T_k \} \).

The procedure rel3(\( S_1, n, S, T \)) outputs all sets \( R^n(S, \{\{D_1\}, \ldots, \{D_i\}, \{C_1\}, \ldots, \{C_k\}\}) \) having \( \{D_1, \ldots, D_i, C_1, \ldots, C_k\} \) as a subset, for \( C_i \in T_i \), \( 1 \leq i \leq k \). If there is a length \( n \) refutation from \( S_l \) and \( T \) is a support class for \( S \), then there will be a length \( n \) refutation from some set output by rel3(\( S, \frac{n+2}{4}, S, T \)).

Definition. If \( S = \{ D_1, \ldots, D_j \} \) then Single(\( S \)) = \{\{D_1\}, \ldots, \{D_j\}\}.

procedure rel3(\( S_1, n, S, T \))
\[ S_2 = R^n(S, T \cup \text{Single}(S_1)) \]
\[ \text{if } S_1 \subset S_2 \text{ then} \]
\[ \text{if } T = \emptyset \text{ then output } (S_2) \text{ else} \]
\[ T_1 = T \setminus S_1 \]
\[ \text{choose } T_2 \in T_1 \text{ minimizing } |T_2|; \]
\[ \text{for all } C \in T_2 \text{ do} \]
\[ \text{rel3}(S_1 \cup \{C\}, n, S_2, T_1 \setminus T_2) \]
\[ \text{od}; \]
\[ \text{fi}; \]
end rel3;

D. Center Clauses

By using the idea that graphs have "centers," we can reduce the distance needed to search for relevant clauses by another factor of 2.

Proposition 4. Suppose \( S \) is a connected subset of \( G(S) \), and if \( C_1, C_2 \in S \) then \( d(C_1, C_2) \leq m \). Then there exists \( C \in S \) such that \( d(C, C_2) \leq \frac{|S|}{2} \).

Theorem 4. Suppose there is a length \( n \) refutation from set \( S \) of clauses, and \( T \) is a support class for \( S \). Then there exists a clause \( C \in S \) and a set \( S_1 \subset S \) having the following properties:

1. There is a length \( n \) refutation from \( S_1 \).
2. \( S_1 \) is fully matched.
3. \( S_1 \) intersects all the support sets in \( T \).
4. \( C \in S_1 \) and for all \( C_1 \in S_1 \), \( d(C, C_1) \leq \frac{|S_1|}{2} \).

Proof. Let \( S_1 \) be the input clauses actually used in some minimal refutation from \( S \). Then \( |S_1| \leq \frac{|S|}{2} \). Choose a "center" \( C \) of \( S_1 \), and note that \( \frac{|S_1|}{2} + \frac{1}{2} = \frac{|S|}{2} \).

By searching for such sets \( S_1 \), we can sometimes obtain much better relevance criteria than by previous methods. The use of centers insures that elements of \( S_1 \) will be closer together than in previous methods.

To implement this method, let \( S_2 \) be \( \{ R^n(S, T) \} \). For each \( C \in S_2 \), let \( S_3 \) be \( R^n(S, \{C\}) \).

If \( S \) intersects all support sets, then it is a candidate set of input clauses for a length \( n \) refutation. Here \( S_2 \) is a set of possible centers.

Note that two clauses of \( S_2 \) will have distance at most \( \frac{|S|}{2} + 1 \) in \( G(S) \). Note also that if \( n=6 \) then \( \frac{|S|}{2} = 2 \) and if \( n = 10 \) then \( \frac{|S|}{2} = 3 \). Thus we can get somewhat nontrivial refutations with quite small distance bounds.

E. Typing Variables

For these relevance criteria to be useful, there must exist clauses \( C_1 \) and \( C_2 \) of \( S \) such that \( d(C_1, C_2) \) is large. However, if the axiom \( x = y \supset \neg x \lor y \) is in \( S \) then two clauses of the form \( t_1 = t_2 \lor D_1 \) and \( t_3 = \neg t_4 \lor D_2 \) will have distance 3 of less. This may cause everything to be close to everything else. To reduce this problem, we propose that all variables be typed as integer, Boolean, list, string, etc. and unifications only succeed if the types match. Thus the above clauses would not necessarily be within distance 3 if \( t_1 \) and \( t_4 \) or \( t_2 \) and \( t_3 \) have different types. The use of types may increase the number of clauses, since more than one copy of some clauses may be needed. However, the overall effect may still be beneficial.

F. Logical Consequences

The preceding ideas can also be applied to derivations of clauses other than \( \text{NIL} \) from \( S \).

Definition. A support set for \( S \) relative to \( C \) is a subset \( V \) of \( S \) such that \( C \) is not a logical consequence of \( S-V \). A support class for \( S \) relative to \( C \) is a collection of support sets for \( S \) relative to \( C \). For example, if \( I \) is an interpretation of \( S \) in which \( C \) is false, and \( V \) is the set of clauses of \( S \) that are false in \( I \), then \( V \) is a support set for \( S \) relative to \( C \).

Definition. \( M(S, C) \) is the largest subset of \( S \) in which all literals are matched, except possibly those having literals of \( C \) as instances.

Definition. \( R^n(S, T, C) \) is \( M^{n-2}(S, T, C) \).

Theorem 5. If there is a length \( n \) derivation of something subsuming \( C \) from \( S \), and \( T \) is a support class for \( S \) relative to \( C \), then there is a length \( n \) derivation of something subsuming \( C \) from \( R^n(S, T, C) \).

As before, we can introduce \( R^n(S, T, C) \) and other relevance criteria.
G. Procedures

To incorporate procedural and heuristic information, we may add clauses expressing the assertion $A(x) \supset (\exists y)B(x,y)$ where $A$ and $B$ are input and output assertions for the procedure and $x$ and $y$ are input and output variables. To account for the fact that heuristics may fail, we assign probabilities of truth to clauses. The task then is to find a set $S$ of clauses from which the desired consequence can possibly be derived, subject to the condition that the product of the probabilities of the clauses in $S$ is as large as possible. One way to do this is to run many trials, generating relevant subsets of $S$, where the clauses of $S$ are chosen to be present or absent with the appropriate probability. We then select a relevant set of clauses from among those clauses that have been found to be relevant in many of the trials. Note that if procedures are encoded as above, then a short proof may correspond to a solution using a few procedure calls, but each procedure may require much time to execute.

H. Subgoals

If procedures are encoded as above, then each procedure may call the whole theorem prover recursively. This provides a possible subgoal mechanism. By storing the clauses from all subgoals in a common knowledge base, we may get interesting interactions between the subgoals. By noticing when subgoals are simpler than the original problem in some well-founded ordering, we may be able to get mathematical induction in the system. The use of clauses, procedures, subgoals, and relevance criteria as indicated here provides a candidate for a general top-level control structure for an artificial intelligence system.

V A COMPLETE STRATEGY

The following procedure attempts to construct a refutation from set $S$ of first-order clauses:

```
procedure refute(s);
   for d = 1 step 1 until (NIL is derived) do
      for j = 1 step 1 until (j > d) do
         refl(s, j, d) od od;
   end refute;

procedure refl(S, i, d);
   let $T$ be a support class for $S$;
   $R \leftarrow R_i(S, i)$;
   if $R$ is empty then return $f$ fi;
   $V \leftarrow RU \{ \text{level 1 resolvents from } R \}$;
   if NIL $\in V$ or d = 1 then return $f$ fi;
   for j = 1 step 1 until (NIL is derived) do
      refl(V, i, d - 1) od;
   end refl;
```

This procedure selects at each step the clauses that seem most relevant and attempts to construct a refutation from them. Similar procedures can be given using other of the relevance functions described earlier.

A. Generating Support Sets

One way to generate support sets for the above procedure is to let each support set be the subset of $S$ in which specified predicate symbols occur with specified signs. This would yield $2^n$ support sets for a predicate symbol. Of course, it is not necessary to use all of these support sets. A more interesting possibility is to have a collection of interpretations of $S$ and to let $T_i$ be the set of clauses that are false in $I_i$. If $I_i$ has a finite domain then $T_i$ can be computed by exhaustive testing. Otherwise, special methods may be necessary to determine if a clause $C$ is true in $I_i$. If $I_i$ has an infinite domain, a possible heuristic is to let $T^i$ be the set of clauses that are false on some finite subset of the domain. If $f$ is an abstraction mapping or a weak abstraction mapping [5] and $I$ is an interpretation, then $\{C \in S: \text{some } C \in f(C) \text{ is false in } I \}$ is a support set for $S$. This approach may allow the use of nontrivial support sets which are easy to compute, especially if all elements of $f(C)$ are ground clauses for all $C$ in $S$. Note that $T$ may include support sets obtained both syntactically and semantically. Although it may require much work to test if $C$ is true in $I_i$, this kind of effort is of the kind that humans seem to do when searching for proofs. Also, this provides a meaningful way of incorporating nontrivial semantic information into the theorem prover. The arcs in the connection graph resemble "associations" between facts, providing another similarity with human problem solving methods.

REFERENCES