TR-333

Pruning Search Trees in Assumption-based Reasoning

by
K. Inoue

January, 1988

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Pruning Search Trees in Assumption-based Reasoning

Katsumi Inoue

ICOT Research Center,
1–4–28, Mita, Minato-ku, Tokyo 108, Japan
phone: +81-3-456-3192
telex: ICOT J 32964
csnet: inoue%icot.jp@relay.cs.net
uucp: {cnea,inria,kddlab,mit-eddie,ukc}!icot!inoue

ICOT Technical Report TR-333
December 1987

Abstract

This paper describes a general search algorithm for multiple contexts that should be considered for reasoning during problem solving. The idea is based on AND/OR tree search with underlying assumption-based reasoning. In assumption-based reasoning, a context can be represented by a combination of assumptions, while in AND/OR tree search procedures, it can be represented by a solution tree. A context deriving a contradiction is checked in a mechanism of truth maintenance, making the search efficient. The proposed search procedure is more powerful than those of various truth maintenance systems. The mechanism is also shown to be very suitable for the constraint satisfaction problem.

Keywords: Assumption-based Reasoning, Dependency-Directed Search, Truth Maintenance, Multiple Contexts, Constraint Satisfaction Problem.

1. Introduction

It is necessary for all AI systems to search in problem solving. For example, synthesis problems (such as design and planning) and conjectural reasoning (such as nonmonotonic reasoning, inductive reasoning and analogy) are basically combinatorial problems. A major problem of such AI systems is that of the combinatorial explosion of search space. During problem solving, search space can be modelled incrementally as construction of AND/OR graphs. One way to tackle the combinatorial explosion in searching these graphs is a deterministic choice of OR parts using a production rule as strong heuristics in rule-based systems. In this case, heuristics are treated as complete knowledge, otherwise they are not helpful for determinicity. In the real world, however, this way is too rigid, because we are often forced to make some decision even if complete information is not available. When we have alternatives for solving problems, we often select a choice nondeterministically and backtracking is needed. For this purpose, *dependency-directed backtracking* (DDB) [Stallman & Sussman 77] is a good way to avoid redundant computing and rediscovering failures involved by chronological backtracking, and it is very common for recent AI systems to control reasoning with *dependency-directed search* (DDS), which is a search mechanism based on DDB. Although DDS itself is a powerful way to treat OR parts intelligently, previously proposed methods have some problems because of their algorithmic difficulties.

The proposal in this paper follows a logical analysis for DDS in terms of the assumption-based reasoning introduced in section 2. General algorithms for DDS, which overcome some problems of the previously proposed methods described in section 3, are shown in section 4. There are two classes of algorithms; one is for searching logically consistent contexts in multiple contexts, and the other, which is for solving a plausible or preferred context in some sense, can be constructed by changing the first algorithm slightly. This technique can be thought of as a hierarchical search for hypothesis space, and may therefore be applied to the constraint satisfaction problem in section 5.

2. Assumption-based Reasoning

DDS plays an important role in *truth maintenance systems* (TMSs) such as [Doyle 79], [McDermott 83] and [de Kleer 86a]. The main task of TMSs is to maintain consistency of dynamic knowledge bases, while DDS can give a guide for problem solving. TMSs perform *assumption-based reasoning*. Assumption-based reasoning is desirable when dealing with alternative knowledge or incomplete knowledge in problem solving, and it makes assumptions for these types of knowledge, with which we can infer what kind of data holds based on what kind of assumptions. In general, a problem solver using a TMS depends on a problem domain, while a TMS is generic, or domain-independent.

A TMS is necessary to maintain dynamic knowledge base, but the style a TMS is implemented with is not important, because our proposal in this paper is not based on a mechanism for truth maintenance but rather on a mechanism of how assumption-based reasoning can be used for problem solving. However, we should know and must specify how a TMS can perform its task. Here, we consider the logical framework for assumption-based reasoning based on the first order predicate calculus. We assume that the TMS can answer the following question correctly:

Input:

Σ : a set of closed formulae, which are submitted by the problem solver to be *premises*, i.e., they hold universally.

Δ : a set of formulae, which are submitted by the problem solver to be *assumptions*, i.e., they are incomplete knowledge, or not guaranteed to be true.

C : a ground formula.

Query:

Compute all *logical supports* D for C with respect to Σ, where D is a set of conjunctions of ground instances of elements of Δ and for all $d \in D$,

1. (*justifiability*): $\Sigma \models d \supset C$,
2. (*satisfiability*): There exists a model M of Σ such that $\models_M d$, and
3. (*minimality*): No proper conjunct subset of d has properties 1 and 2. $\square$

The above query is denoted as $SUPPORT(\Sigma, C)$. Here, $d \in D$ is called the *supporting assumption* (or *logical support*) for C with respect to Σ, and D is called the *set of supporting assumptions* (or *all logical supports*) for C with respect to Σ. A combination of ground instances of elements of Δ is called a *context*. Therefore, any supporting assumption for any formula constructs a context, and a set of supporting assumptions for any formula may represent *multiple contexts*. If E represents a context, a maximal set of logical consequences of a set $\Sigma \cup E$ such that $\Sigma \cup E$ is *consistent*, that is, property 2 (satisfiability) is satisfied, is called an *extension* of Σ and E. Of course, the TMS may perform other tasks such as determining what data is believed or not in a particular context, but it is the $SUPPORT(\Sigma, C)$ procedure that can characterize its following logical framework. First, the TMS deals with *assumptions*, which are specialized data for incomplete knowledge, so that all logical supports for any formula are constructed by the set of conjunctions of ground instances of elements of Δ. These computation of logical supports, which are sometimes called *hypotheses*, is called *hypothetical reasoning* [Poole 87] and can be applied directly to explain an observation in diagnostic reasoning. The separation between assumptions and other formulae makes knowledge bases comprehensive and helps the system to compute clearly and efficiently. This is in contrast to computing all or some 'prime implicants' of Σ of propositional clauses as in the *Clause Management System* [Reiter & de Kleer 87]. Second, the TMS maintains multiple contexts simultaneously to compute all logical supports for C. This is similar to de Kleer's *Assumption-based Truth Maintenance System* (ATMS) [de Kleer 86a]. If any assumption and any supported formula C are atomic formulae and Σ is a set of grounded Horn clauses, the above three properties are a formalism for the ATMS, so that D, $d \in D$, a context, and an extension correspond to a 'label', an 'environment', a 'characterizing environment for a context', and a 'context' of the ATMS. In problem solving, for example, the selection of models in design problems, there may exist multiple worlds corresponding to various items of alternative knowledge and we often want to compare those worlds, so that a mechanism of maintaining multiple contexts is necessary. Third, the above framework gives the semantics for assumption-based reasoning. It subsumes various TMSs. Although an implementation of the TMS is beyond the scope of this paper, it is desirable to implement it by a logic-based approach, an outline of which was proposed in [Inoue 87]. In the following discussion, however, it is assumed for simplicity that any element of Δ is atomic like the ATMS, but the method of controlling reasoning is different. This restriction for the TMS is not essential for searching multiple contexts.

By exploiting the above TMS to DDS, we can characterize the intelligent search procedure for solving non-deterministic problems in problem solving. DDB has an intelligent cache for the problem solver to avoid much of redundant computation. The search procedure realizing DDS will be shown in section 4. Here, we describe how the logical framework of the TMS is concerned with search strategies for problem solving. When we encounter alternatives or draw some conclusion from incomplete knowledge, we can make assumptions for that decision. A combination of such knowledge constructs a context, and this context must satisfy *consistency*, that is, satisfiability of logical supports. Note that logical supports must be computed for all necessary data which should hold in some context, and a context itself is a minimal combination of such individual logical support such that it satisfies all properties of the logical support for conjunction of data, and such that it satisfies properties 1 and 2, but does not satisfy property 3 for individual formulae. As stated above, there may be many consistent combinations of assumptions, i.e., multiple contexts. When there are multiple contexts, which context should be selected is a problem. We believe that the *multiple contexts problem* should be solved dependently on problem domains, because at present it seems to be too weak for a general framework of commonsense reasoning like nonmonotonic logics to select one context [Hanks & McDermott 87]. Therefore, this paper focusses on the search for multiple contexts, and the TMS is generic for maintaining multiple contexts while the problem solver reasons dependently on the problem domain. The search procedure based on DDS is considered to be the generic interface between the TMS and the problem solver, but to be domain-dependent in the sense of utilizing domain heuristics.

3. Searching Multiple Contexts with TMSs

Several techniques have been proposed to handle multiple contexts. These techniques manage search in knowledge-based systems. They can be broadly classified as CONNIVER-style, justification-based or assumption-based, which roughly correspond to the spatial, serial and conditional methods in the classification proposed by [Post 87]. Hybrid approaches have also been developed to avoid the pitfalls of these techniques.

The *CONNIVER-style* approach sprouts alternative contexts that coexist in the database. It was originally developed by the programming language *CONNIVER* [Sussman & McDermott 72]. This method models contexts using named tags with assumptions corresponding to assertion and deletion of facts. Commercial expert system tools *ART* and *KEE* use a similar mechanism. It is easy to switch or compare contexts with each other during search. Its disadvantages are duplication of facts in different contexts and the difficulty of multiple disjunctive dependencies.

The *justification-based* approach keeps only one current consistent context and DDB is provided if a failure occurs. This mechanism was introduced in the nonmonotonic *Justification-based TMS* (JTMS) [Doyle 79], and is the most popular method to provide default reasoning. The main difficulty of this method is that of comparing and constructing alternatives, so that the costs of context switching are high. The recent AI language *SCHEMER* [Zabih *et al.* 87], which has a DDB mechanism in a simple binary OR tree as search space and so realizes McCarthy's AMB operator, has the same problem. However, the scheme proposed by McDermott, insisting on a single context, is somewhat different from other JTMSs. It retains the essence of the CONNIVER-style approach with underlying combinations of assumptions as multiple contexts [McDermott 83]. This point is clarified further by the ATMS.

The *assumption-based* approach maintains a global, concurrent representation of all contexts by labelling each item of data with dependent assumptions. This is represented in the ATMS. Many advantages of the ATMS over the JTMS are given in [de Kleer 86a]. It allows multiple contexts to be compared, switched, or synthesized as needed. In the ATMS, only the *environment*, i.e., a combination of assumptions, identifies a context. This point provides an advantage over the CONNIVER-style approach, because the ATMS avoids redundant computations and duplication of conclusions in different contexts. Therefore, the method proposed in this paper follows the assumption-based approach described in the previous section.

With the ATMS, however, there is still a major problem of controlling the inference during search for multiple contexts. Search in the ATMS is based on *interpretation construction*, which is regarded as searching the *environment lattice*, i.e., the Hasse diagram of the power set of assumptions. During interpretation construction, the earlier the most general environment of a node's label is found, the more label updating is avoided, and the earlier the most general *nogoods*, that is, inconsistent assumption sets, are found, the more steps concerning ultimately inconsistent environments are reduced. For those purposes, a problem solver focusses breadth-first on contexts with fewer assumptions first through a specialized interface, or *consumer architecture* [de Kleer 86c]. The scheduler needs backtracking when only part of the search space should be explored for the purpose of the characteristics of tasks, such that not all solutions are required at once, or requirement of efficiency. Therefore, a simple method that combines depth-first search of DDB and breadth-first search of consumer architecture is proposed in [de Kleer & Williams 86]. The algorithm, called *assumption-based DDB* (ADDB), however, still has the following problems:

First, their method that the ATMS is guaranteed to explicitly identify all nogoods which follow from the current set of justifications does not appear to be very effective in ADDB. ADDB checks (by the breadth-first search) the consistency of the current environment consisting of assumptions, one from each disjunctive *control assumption*, so that consumers of ultimately unnecessary environments may be executed by the scheduler. Even if not all of the nogoods involved in the current environment are discovered, they will be identified if necessary. The advantage appears only in avoiding checking the consistency of other contexts including already identified nogoods. The disadvantage that more consumers of intermediate environments ultimately independent of solutions are executed seems to be more expensive. Second, in ADDB, the current environment must be flatly constructed from all control assumptions. Therefore, the task that some choices are dependent on other contexts and some are not cannot be dealt with by ADDB directly. It is a problem of the ATMS itself that all assumptions are treated in the same flat structure, without hierarchy. Third, the interpretation construction of the ATMS is isomorphic to the minimum set covering problem which is NP-complete. Since no algorithm, including ADDB, tries to reduce the inherent combinatorial explosion of search space, high efficiency cannot be expected of these TMS algorithms.

4. The Context Search Algorithm

Given the problems analyzed in the previous section, our main goal is to formalize a general search algorithm for multiple contexts so as to overcome them. The approach follows from the assumption-based style which functions as the TMS described in section 2. The underlying logic represented by $SUPPORT(\Sigma, C)$ can make the TMS prune the search tree by resolution as described later.

In assumption-based reasoning, it is absurd to use a stack control regime in order to realize context switching easily. Our method for controlling search is via an *AND/OR tree*, where the *root node* represents an overall problem to be solved, and *arcs* in it indicate logical dependencies between *nodes* representing decomposition processes or relations of assumptions. Nodes with successors are called *nonterminal*, and those with no successor are called *terminal*. Each nonterminal node has immediate successors either of type AND or of type OR. Nonterminal nodes with successors of type AND are called *AND nodes*, and their successors correspond to conjunctive partial problems. Nonterminal nodes with successors of type OR are called *OR nodes*, and their successors correspond to disjunctive assumptions. As already described in section 1, multiple contexts can be basically represented by an AND/OR tree or an AND/OR graph. Hence, it is possible for us to apply directly several algorithms for AND/OR graph search, such as AO^* or *GBF* [Pearl 84], to search control in assumption-based reasoning. Although the search space for nondeterministic problems has been dealt with by OR trees such as a binary tree for SCHEMER, an OR tree can be thought as a special case of AND/OR trees. By comparing an AND/OR tree with an equivalent OR tree, the former has several advantages over the latter, such that with an AND/OR tree it is more natural to describe a problem and more compact to represent a problem avoiding duplication.

Given an AND/OR tree representation of assumptions, we can identify its different solutions, each one representing a possible context, by a *solution tree*. A solution tree, T, of an AND/OR tree, G, is a subtree of G with the following two properties: (i) the root node of G is the root node of T, and (ii) if an AND node of G is in T, then all of its successors are in T, and if an OR node of G is in T, then exactly one of its successors is in T. AND/OR tree search is used to explore contexts. We assume that the search tree G should be *incrementally* constructed, expanded, and traversed during problem solving, so that it is not necessary to explore all assumptions as a whole search tree. This point is very important for practical problem solving because not all of assumptions or their relations are represented explicitly before any inference starts. We shall use a representation for a set of solution trees, that is, a *partial solution tree*. A partial solution tree, T', of an AND/OR tree, G, is a subtree of G with the following three properties: (i) the root node of G is the root node of T', (ii) if any node other than the root of G is in T', then its ancestors are also in T', and (iii) if an OR node of G is in T', then at most one of its successors is in T'. It is possible to say that T' is a set of T and that it is an incomplete solution tree which may be extended. A partial solution tree is called *active* when it represents a consistent context. Therefore, an active partial solution tree corresponds to a set of assumptions satisfying consistency described in section 2, with their logical dependencies.

We now present two classes of search algorithms for multiple contexts. One of them, given in section 4.1, is for searching all (or one) logically consistent contexts corresponding to complete solution trees. This is available when all assumptions immediately following any context are treated equally, or not ordered, or equivalently simply ordered. The first and the second problems of ADDB described in section 3 will be solved with this algorithm. The other, given in section 4.2, is for searching one (or all) optimal context(s) by some estimation, and can be obtained by changing another algorithm slightly. This is available when a *preference* relation, such as some estimation function or partial ordering, can be obtained among assumptions. This algorithm could solve the third problem of ADDB described in section 3.

4.1 Search for Consistent Contexts

Let Σ be a set of data maintained by the TMS, as defined in section 2. The context search algorithm called *GSEARCH* maintains another four sets, that is, $OPEN$, D, $NOGOOD$ and $SOLUTION$. $OPEN$ is a set of active partial solution trees, each of which represents a state of traversal corresponding to a consistent context. D is a set of maximal consistent contexts that have been found to be consistently combined assumptions during search. Each element $d \in D$ corresponds to an *interpretation* of the ATMS [de Kleer 86b]. $NOGOOD$ is a set of minimal inconsistent contexts that have been found, i.e., nogoods. $SOLUTION$ is a set of solutions, each of which represent a complete consistent solution tree.

In the $GSEARCH$ algorithm, the basic loop consists of picking one active partial solution tree from $OPEN$, checking its consistency by the *CHECK* procedure, executing the problem solving procedures attached to its context if the test is all right, and then decomposing it or traversing search by the *EXPAND* procedure. An active partial solution tree in $OPEN$ can be represented by a set of tip nodes. If $CHECK$ succeeds, a context corresponding to the partial solution tree is added to D, otherwise, a contradiction occurs in the context, so that it is added to $NOGOOD$. To perform $CHECK$, the TMS described in section 2 is used with it arranged especially to find inconsistencies of contexts. Logical dependencies of assumption and justifications given by invoking *consumers*, i.e., attached problem solving procedures, are added to Σ by the *ADDCLAUSE* procedure. In *GSEARCH*, at the beginning, a problem, P, to be solved is assigned to *OPEN*, and all consumers attached to the overall context $\{\}$ which consists of no assumptions are executed.

```
procedure GSEARCH:
  1. Let $OPEN := \{\{P\}\}$, $D := \phi$, $NOGOOD := \phi$ and $SOLUTION := \phi$.
  2. Halt if a termination condition is satisfied; consistent contexts are given by
     $SOLUTION$. A termination condition is either of the following: (a) when all
     consistent contexts are desired, $OPEN = \phi$, or (b) when only one solution is
     desired, $SOLUTION \neq \phi$.
  3. Select an element, $L$, from $OPEN$ and delete it from $OPEN$. Suppose $L = \{C, C_0, C_1, \dots, C_k\}$.
  4. Let $E := SUPPORT(\Sigma, C)$. If $E = \{\}$ holds, then go to 8.
  5. Let
       $S := \{\{C\}, \{C, C_0\}, \{C, C_1\}, \dots, \{C, C_k\},$
            $\{C, C_0, C_1\}, \dots, \{C, C_0, C_k\}, \dots\dots, \{C, C_0, C_1, \dots, C_k\}\}.$
  6. If $S = \phi$ holds, then go to 8.
  7. Select the first element, $s$, from $S$. If $CHECK(s, E) =$ true, then return to 6.
     Otherwise, return to 2.
  8. $EXPAND(L)$ and return to 2. □
```

```
procedure CHECK(context,E):
  1. Let $C$ be the first element of context, and $F$ be the set of the remaining elements
     of context.
  2. If there exists $d \in D$ such that $E \cup F \subseteq d$, then return(true).
  3. If there exists $n \in NOGOOD$ such that $E \cup F \supseteq n$, then return(false).
  4. Execute consumers attached to $E \cup F$, then $ADDCLAUSE(\Sigma, p)$ for each jus-
     tification $p$ given by invoking it. If a justification derives $\perp$ (or falsity), then
     return(false).
  5. Add $E \cup F$ to $D$ and delete all elements which are a subset of $E \cup F$ from $D$, then
     return(true). □
```

```
procedure EXPAND(L):
Remark. This procedure is depth-first.
  1. Let $C$ be the first element of $L$, and $F$ be the set of the remaining elements of $L$.
     Suppose $F = \{C_0, C_1, \dots, C_k\}$.
  2. If $C$ is nonterminal, generate all the sons, $C_{S_i} (i = 1, \dots, m)$, of $C$, where $m$ is the
     number of sons, and then $ADDCLAUSE(\Sigma, R)$ for the logical dependencies, $R$,
     of assumptions. If each $C_{S_i}$ is of type AND, then add $\{C_{S_1}, C_1, \dots, C_k\}$ to the
     head of $OPEN$, otherwise, add $\{C_{S_i}, C_1, \dots, C_k\}$ for all $i = 1, \dots, m$ to the head
     of $OPEN$. Return.
  3. If $C$ is terminal, backtrack to the sibling node $A_S$ of the ancestor node of $C$,
     where $A_S$ is the nearest unexpanded node of type AND from $C$, and then add
     $\{A_S, C, C_1, \dots, C_k\}$ to the head of $OPEN$, and return. If there is no node $A_S$, add
     $L$ to $SOLUTION$, then return. □
```

procedure ADDCLAUSE(Σ,s):

```
1. Add clause $s$ to $\Sigma$.
2. If $s$ or a resolvent of $s$ and a clause in $\Sigma$ derives $\bot$, compute an inconsistent
   assumption set $n$ by $SUPPORT(\Sigma, \bot)$. Add $n$ to $NOGOOD$ and, for each element
   $d \in D$, if $n$ subsumes $d$, substitute $d$ for $d - n$. Delete all elements subsumed by the
   other element in $NOGOOD$ from $NOGOOD$, and delete all elements subsumed
   by any element in $NOGOOD$ from $OPEN$.
3. Return. $\Box$
```

Note that $GSEARCH$ is not only a single search procedure, but can be extended to a class of search procedures that result by specifying the selection rule in step 3 and altering the expansion rule in $EXPAND$. Although we have proposed the depth-first procedure here, the subsequent section shows that informed search procedures with best-first search are constructed by changing these rules slightly.

Remarks. (1) In step 4 of $GSEARCH$, $SUPPORT(\Sigma, C)$ returns logical supports for C including C itself if C is an assumption. Therefore, when $SUPPORT(\Sigma, C) = \{\}$, $\Sigma \models C$ holds, that is, C holds universally as a fact. (2) $SUPPORT(\Sigma, C)$ is only necessary in step 4 of $GSEARCH$ and in step 2 of $ADDCLAUSE$. However, there may be many cases implicitly using it for executing consumers to record logical dependencies of data, so that the scheduler collects all consumers for a particular context. (3) $NOGOOD$ and $SOLUTION$ grow monotonically, while D and $OPEN$ grow nonmonotonically. (4) $CHECK$ is a procedure to test *whether an assumption contradicts the context deriving it.* For example, when a negative clause $\neg A \vee \neg B$ is in Σ, the context search algorithm prunes all partial solution trees including both A and B by recording $\{A, B\}$ in $NOGOOD$, which means that B cannot be added to the context $\{A\}$ because $A \supset \neg B$ holds. (5) Once a consumer is executed, it is never executed again. This is ensured by step 2 of $CHECK$, because contexts that have been already checked are returned immediately before the consumer is executed. $\Box$

Proposition 1. A context L picked from $OPEN$ (in step 3 of $GSEARCH$) satisfies the condition where the remainder F of L without the first element C is consistent. There exists $d \in D$ such that $F \subseteq d$.

Remark. The set F is like a *kernel environment* of the ATMS [de Kleer 86c], but it is used differently. F is a set of conditions of which each solution must be consistent and $GSEARCH$ only checks the consistency of combinations of such a condition and new assumption C, so that we need not check the consistency of the power set of F again. In the ATMS, a kernel environment is a set of conditions with which any solution must be consistent, while F depends on each context in $GSEARCH$.

Proof. No partial solution tree in $OPEN$ includes any element of $NOGOOD$ by step 2 of $ADDCLAUSE$. The first element of each partial solution tree in $OPEN$ must be constructed in front of F in step 2 or 3 of $EXPAND$, where F is either a subset of a consistent context (in step 2) or a consistent context (in step 3), whose consistency has already been checked in step 4 or steps 5 to 7 of $GSEARCH$, and has already been added if it has not been included by any element of D in step 5 of $CHECK$. $\Box$

Proposition 2. In $GSEARCH$, the following properties hold for any point during search:
(1) $D \supseteq SOLUTION$, and
(2) For all $d \in D$, there does not exist $n \in NOGOOD$ such that $d \supseteq n$.

Proof. The first property is obvious from the fact that any solution is added to $SOLUTION$ in step 3 of $EXPAND$ after its consistency was checked and added to D. The second property is immediate from step 2 of $ADDCLAUSE$. $\square$

Theorem 1. At the end of $GSEARCH$, any solution tree in $SOLUTION$ is consistent, that is, $GSEARCH$ is *sound*, and when all consistent contexts are desired, $SOLUTION$ holds all of them, that is, $GSEARCH$ is *complete*.

Proof. The soundness is obvious from Proposition 2. To prove the completeness we assume the contrary and obtain a contradiction. Suppose $GSEARCH$ misses the solution, S. Since every consistent partial solution tree is added to $OPEN$ and picked from it unless it becomes contradictory, there exists a context, S', which is a partial solution tree of S such that S' is selected from $OPEN$ and is not to be inserted in $OPEN$ again. Then, S' is either found to be inconsistent, or added to $SOLUTION$. Both cases contradict the supposition. $\square$

$GSEARCH$ has several advantages as it searches an AND/OR tree constructed with hierarchy incrementally. First, with $GSEARCH$, problem solving can proceed efficiently with *compiled knowledge* because a contradiction in an intermediate level can be found so that a kind of compilation of some condition on a set of low level knowledge can be represented. This is our solution for the second problem of ADDB. Second, $GSEARCH$ can prune a search tree by resolution. Pruning a partial solution tree corresponds to a resolution of a nogood in $NOGOOD$ and a logical dependency between nodes such as conjunction or exhaustivity of disjunction. For example, a nonterminal node becomes inconsistent if it has sons of type AND and one of them is found to be inconsistent, or if it has sons of type OR and all of them are found to be inconsistent. Resolution by the TMS can make $GSEARCH$ prune partial solution trees, which is ultimately inconsistent. A partial solution tree, T, is pruned when its expanded partial solution trees are inconsistent and they cover exhaustively the expanded sons of an OR node of T. This pruning corresponds to *negative hyperresolution* of the ATMS [de Kleer 86b] which is used to ensure completeness when the ATMS deals with positive clauses as well as Horn clauses as justifications. In $GSEARCH$, this can be implemented more simply because of the concept of the partial solution tree. Third, $GSEARCH$ *surpasses* ADDB, that is, $AREA(GSEARCH) \subseteq AREA(\text{ADDB})$ holds for any problem P, where $AREA(X)$ denotes the set of contexts whose consumers are executed using X. In ADDB, although the number of contexts tested for the consistency is reduced, the number of consumers executed is not reduced by the backtracking algorithm. If the costs of executing consumers are very high, the first problem of ADDB will arise. In $GSEARCH$, the problem is solved with the search order in steps 5 to 8 so that fewer consumers of ultimately unnecessary contexts are executed than ADDB. This property will be illustrated in examples in section 5.

4.2 Informed Search for an Optimal Context

When some estimation for assumptions, contexts or solutions is available, we can expect to improve the search performance. We can order contexts by comparing them with some preference relation, and an optimal solution can be gained. For this purpose, we might change $GSEARCH$ in section 4.1 slightly. The concept of consistency in $CHECK$ might be altered to *feasibility*, or, possibility for optimality. The selection rule in step 3 of $GSEARCH$ or the expansion rule in step 2 or 3 of $EXPAND$ might be altered so that the most preferred context is selected and expanded. The termination condition, however, is left as in the case of the all solution search of $GSEARCH$ to exclude local optimization like hill-climbing. The resulting procedure performs *best-first search* like AO*, or it can support the *branch and bound* procedure (B&B) [Ibaraki 77]. There are several approaches and concepts, which are not independent, for searching the optimal context, as follows.

(1) Ordering
The *ordering* strategy can make the search procedure more efficient, because it finds many general nogoods early in $GSEARCH$. This strategy is useful even if we desire logically consistent solutions as described in section 4.1. A simple example where the arrangement of subproblems makes search efficient will be shown in the next section. This is one aspect where $GSEARCH$ surpasses ADDB.

(2) Best-first Search
In an assumption-based approach like the ATMS, multiple concurrent contexts are maintained. It seems to be rationally efficient for problem solving with this approach that best-first search is employed to elicit advantages of concurrency. In best-first search, context switching or backtracking happens not only when a context becomes contradictory, but also when there is a more preferred active context. When using some estimation function, when an estimation of some context is updated, context switching will happen still more. Generally the number of times to decompose the problem in best-first search is smaller than any other search such as depth-first, at the cost of some overhead of space.

(3) Pruning with B&B
In TMSs, *constraints* are used for detecting contradiction. With B&B it is possible to enhance efficiency for constraint solving when an object value should be computed. A constraint can be changed dynamically during search so that infeasible contexts are pruned by the bound. For example, in job-shop scheduling problems, derived plans must satisfy the constraint of the limit of machining time. However, this constraint can be improved by the object value of the best feasible solution currently available (i.e., the *incumbent*).

(4) Heuristic Search
Generally, to solve combinatorial problems, it is hard to keep all active contexts like breadth-first search. Some *heuristic information* helps problem solving to tackle the combinatorial explosion as humans do. If heuristic information is available, heuristic search proceeds along an plausible inference path. Note that heuristic search does not

always find an optimal solution. Whether optimality is ensured or not depends on the properties of the preference relation. If a sufficient condition that the solution of heuristic search is the best solution can be obtained, the heuristic information can, of course, be used. This depends very much on the problem to be solved.

(5) Dominance Relation
When an appropriate estimation function cannot be obtained, it may be possible to reduce the search with a *dominance relation*. A dominance relation $\mathcal{D}$ is a binary relation defined on the set of partial problems such that $P_1 \mathcal{D} P_2$ implies that P_2 can be excluded from consideration without loss of optimality of the given problem, if P_1 has been already generated when P_2 is selected for the check. This is used in B&B and efficiency is obtained with or without an estimation function. The dominance relation may be used when partial ordering is used.

5. Application to Constraint Satisfaction Problem

The context search algorithm described in the previous section must be used with a problem solver dependent on a problem domain. Here, the working of the $GSEARCH$ algorithm on an example of a *constraint satisfaction problem* (CSP) is illustrated. In CSP, consistent assignment of values for a set of variables which satisfies all constraints is found [Mackworth 77]. A constraint $C_i(X_{i_1}, \ldots, X_{i_j})$ specifies which values of the variables are compatible with each other. A CSP is a task to be solved in the design problem. To solve CSP, *generate and test* (G&T) can be used as a technique. The constraints are used to test consistency of the assignments made by a generator. When the constraints can be applied to partial assignments, partial solutions can be pruned by *hierarchical G&T*. Chronological backtracking is used to implement hierarchical G&T, but DDS can enhance performance more.

The $GSEARCH$ procedure can be applied to CSP as follows. An assumption is an assignment of a value to a variable. With the constraints folded into the generator by the consumer of the context, $GSEARCH$ can assign values for variables such as a *look-ahead* scheme or *constraint propagation*. This can be done by depth-first search of $EXPAND$. Hierarchical G&T with DDS such as a *look-back* scheme make the search more efficient. This can be supplied by $CHECK$. Therefore, $GSEARCH$ has the advantages of both the look-ahead and look-back schemes. The specification of the problem P can be given by any logical form of decomposition to subproblems. AND/OR tree search in $GSEARCH$ is then applied to the specification of P. In any level of the tree, when an assumption is newly considered, the TMS can check if it is consistent with a current environment by computing logical supports of it or its negation. $GSEARCH$ never generates a context that happens to occur in an impossible combination. Note that an assumption in an intermediate level corresponding to a nonterminal node is a state of partial problems or a subset of variables and it might not have an explicit value or a consumer. However, an assumption in an intermediate level can represent compiled knowledge as described in section 4.1, so that a partial solution tree can be pruned by the constraints. We can also represent 'components', that is, partial solutions to a set of constraints which can then be ignored because the constraints are already implicit in themselves. This way of representation is reported to be very useful for CSP in an other independent research of [Mittal & Frayman 87].

Next, some simple examples illustrate the application of our proposed approaches based on *GSEARCH* for CSP. Through these examples, DDB, ATMS, and ADDB are compared. Here, the main characteristics for these systems are justification-based approach with depth-first search such as SCHEMER for DDB [Zabih *et al.* 87], and assumption-based approach with breadth-first search for the ATMS [de Kleer 86b], and the integration of DDB into ATMS for ADDB [de Kleer & Williams 86]. *GSEARCH* surpasses any of these systems because ADDB searches only the common parts of DDB and ATMS, and *GSEARCH* searches no more contexts than ADDB. Note that in the following examples only part of the advantages of *GSEARCH* is displayed because the constraints work on only the lowest level, or terminal nodes, and never work on compiled knowledge hierarchically. These examples are used only to show an advantage of *GSEARCH* over other algorithms even when the problem has a flat structure that is handled by them. First, let us consider the CSP presented in Example 1, which is discussed in [de Kleer & Williams 86].

Example 1. The problem P is a conjunction of three subproblems P_1, P_2 and P_3. Each subproblem corresponds to a variable whose possibilities of values are:

$$P_1 \in \{a, b\},\ P_2 \in \{c, d\},\ P_3 \in \{e, f\}\,.$$

Assume that the following inconsistencies are ultimately detected:

$$\begin{gathered} a \wedge d \supset \bot\,, \\ b \supset \bot\,, \\ c \wedge e \supset \bot\ . \end{gathered}$$

Note that these justifications deriving inconsistencies can be thought of as *integrity constraints* and might not be given explicitly. Instead, the following dual constraints may be given to the original CSP like [Mackworth 77]:

$$C_1(P_1, P_2),\ C_2(P_2, P_3),\ C_3(P_1),\ \ldots\ ,$$

and each C_i is satisfied, for instance, in the following cases:

C_1: $(a, c), (b, c), (b, d)$.
C_2: $(c, f), (d, e), (d, f)$.
C_3: (a).

In this example, however, it is not essential for the constraints to be given by explicit satisfying conditions or by implicit integrity conditions.

We represent an assumption that is a decision to assign a value to a variable by the capital letter of the value. For example, when a is selected for a value of P_1, the corresponding assumption is represented by A, and $A \supset a$ is added to Σ. Then the problem P forms an AND/OR tree shown in Figure 1. The solution of P is the context

$\{A, C, F\}$ and it is shown by the solution tree (the bold line) in Figure 1. An example of the partial solution tree is illustrated in Figure 1 by the dotted line, representing $\{P_2, A\}$.

DDB, ATMS and ADDB execute at most the consumers of the following contexts. In total, there are 27 contexts in P. The order below is the order in which each context is examined for consistency by the procedure.

DDB: $\{\}$, $\{A\}$, $\{C\}$, $\{A,C\}$, $\{E\}$, $\{A,E\}$, $\{C,E\}$, $\{A,C,E\}$, $\{F\}$, $\{A,F\}$, $\{C,F\}$, $\{A,C,F\}$, $\{D\}$, $\{A,D\}$, $\{D,E\}$, $\{A,D,E\}$, $\{B\}$, $\{B,C\}$, $\{B,F\}$ and $\{B,C,F\}$ (20 contexts)

ATMS: $\{\}$, $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$, $\{F\}$, $\{A,C\}$, $\{A,D\}$, $\{A,E\}$, $\{A,F\}$, $\{C,E\}$, $\{C,F\}$, $\{D,E\}$, $\{D,F\}$ and $\{A,C,F\}$ (16 contexts)

ADDB: $\{\}$, $\{A\}$, $\{C\}$, $\{A,C\}$, $\{E\}$, $\{A,E\}$, $\{C,E\}$, $\{F\}$, $\{A,F\}$, $\{C,F\}$, $\{A,C,F\}$, $\{D\}$, $\{A,D\}$, $\{D,E\}$ and $\{B\}$ (15 contexts)

Figure 2 shows the process of $GSEARCH$ in this problem. Note that the logical dependencies of nodes, such as $P \equiv P_1 \wedge P_2 \wedge P_3$ or $P_1 \equiv A \vee B$, are added to Σ and they are indicated by '*'. However these formulae are irrelevant to computation of logical supports in this example, because they only represent the decomposition relations. Therefore, all logical implications in formulae are bidirectional, or equivalent. In $GSEARCH$, these relations can be treated as partial solution trees and pruning search trees are done automatically in the algorithm, so that it is not necessary to check resolution with these formulae. Here is the ordered list of contexts executed for their consumers by $CHECK$ in $GSEARCH$, i.e. $AREA(GSEARCH)$:

$\{\}$, $\{A\}$, $\{C\}$, $\{A,C\}$, $\{E\}$, $\{C,E\}$, $\{F\}$, $\{C,F\}$, $\{A,F\}$, $\{A,C,F\}$, $\{D\}$, $\{A,D\}$ and $\{B\}$ (13 contexts).

In this example, $GSEARCH$ searches for *the minimum number of contexts*, that is, it only searches {the power set of solution contexts} $\cup$ {the power set of all given nogood contexts} . □

Example 2. Suppose that problem P has the same structure as Example 1, but the detected inconsistencies are different, as follows:

$$a \wedge e \supset \perp ,$$
$$b \supset \perp ,$$
$$a \wedge f \supset \perp .$$

In this case, there is no consistent solution. Figure 3 shows the environment lattice [de Kleer 86a] for this example. In Figure 3, the nodes in the lowest level correspond to a solution tree and the upper nodes are their subset contexts, but $\{\}$ is not represented there. Nogoods derived by these justifications are represented by netted nodes.

DDB searches the contexts above the line partitioning the lattice in Figure 3, and executes the consumers of 16 contexts. ATMS does not search the crossed-out nodes which correspond to the supersets of detected nogoods, but executes the consumers of 15 contexts. $AREA$(ADDB) is indicated by the conjunction of $AREA$(DDB) and $AREA$(ATMS), and 11 contexts are examined by ADDB. $GSEARCH$ examines the contexts which are the same as $AREA$(ADDB), and the order in which $GSEARCH$ examines a context is indicated to the corresponding nodes. Note that the context $\{A, D\}$ is not examined by any procedures except the basic ATMS. In $GSEARCH$, it is pruned because the partial solution tree $\{A, P_3\}$ is found to be inconsistent from the pruning rule, that is, $nogood\{A, E\}$ and $nogood\{A, F\}$ derive $nogood\{A, P_3\}$ and then derive $nogood\{A\}$. DDB and ADDB do not execute the consumers of $\{A, D\}$ either, but DDB checks consistency for $\{A, D, E\}$ and $\{A, D, F\}$, and ADDB schedules them, while in $GSEARCH$, they never appear in $OPEN$. □

Example 3. This is the same as Example 2, except that inconsistencies are detected as follows:

$$\begin{gathered} a \wedge c \supset \bot\,, \\ b \supset \bot\,, \\ a \wedge d \supset \bot\;. \end{gathered}$$

In this case, there are no consistent solutions like Example 2. The structure is thought to be the same with respect to nogoods. This example can be obtained by changing the order of P_2 and P_3 in Example 2.

In this example, the numbers of examined contexts with DDB, ATMS and ADDB are 16, 15 and 11, respectively, and these are the same as Example 2. However, $GSEARCH$ examines the minimum number of contexts (7 contexts). This result shows that the ordering strategy can have high search efficiency as described in section 4.2. □

To summarize, $GSEARCH$ is more powerful for CSP than other procedures even if the problem P has a flat structure like that shown in the examples. We emphasize again that $GSEARCH$ can treat problems with a hierarchical structure different from the above examples. If inconsistencies are found in an intermediate level, partial solution trees for some compiled knowledge are pruned. It should be also noted that $GSEARCH$ could be applied to another type of design tasks rather than simple CSP, such that the problems need to be selected their models as well as the values of the variables for models. A *model* can be represented by a set of constraints relating the desire, intention, specification or necessity given by the user forming a context. For example, in [Bose & Rao Padala 87], the justification-based approach is used for the flight planning problem. In $GSEARCH$, such models can be represented by the hierarchical structure of AND/OR graphs, and the parameters of the models can be attached below them. The assumption-based approach is very helpful for selecting models as it maintains multiple contexts elegantly.

6. Conclusion

This paper introduced a general problem solving technique to control reasoning with DDS. The resulting search procedure $GSEARCH$ works between the domain-dependent problem solver and the domain-independent TMS which maintain dynamic knowledge. The main characteristics of the proposed method are that reasoning is controlled by an AND/OR tree search mechanism, and that assumptions can be added to the TMS along their contexts incrementally rather than added to every possible world concurrently in a flat structure like the ATMS. Informed search can be corporated into this method to make searching more efficient. This mechanism can solve CSPs.

The proposed framework will be incorporated into an assumption-based reasoning system *APRICOT* [Inoue 87] to be developed. *APRICOT* will supply the basic architecture for maintaining multiple contexts based on logic-based TMS, and will be the core for controlling various reasoning in the knowledge system shell *PROTON-2* which will be implemented in the PSI-2 machine. PROTON-2 is an improved version of the expert system tool *PROTON* [Nagai *et al.* 87] in ICOT. We plan to apply this mechanism to constraint solving in mechanical design.

One of the future topics of research is the trade-off between efficiency with heuristic information and completeness or optimality of the algorithm, as described in section 4.2. We believe that our proposal will be a basis for the bridge between logical consistency such as TMSs in AI and algorithmic feasibility such as combinatorial optimization in OR.

Acknowledgments

I would like to thank Dr. Koichi Furukawa, Y. Nagai and K. Satoh for their useful comments. I would also like to thank Y. Fujii, the chief of the fifth laboratory at ICOT, for his kind encouragement. Finally, I wish to express my thanks to Dr. Kazuhiro Fuchi, Director of ICOT Research Center, who provided me with the opportunity to pursue this research in the Fifth Generation Computer Systems Project.

References

[Bose & Rao Padala 87] Bose, P. K. and Rao Padala, A. M., "Reasoning with Incomplete Knowledge in an Interactive Personal Flight Planning Assistant", *Proc. Avignon 87: Expert System & their Applications* (1987), pp. 1077–1092.

[de Kleer 86a] de Kleer, J., "An Assumption-based TMS", *Artificial Intelligence* **28** (1986), pp. 127–162.

[de Kleer 86b] de Kleer, J., "Extending the ATMS", *Artificial Intelligence* **28** (1986), pp. 163–196.

[de Kleer 86c] de Kleer, J., "Problem Solving with the ATMS", *Artificial Intelligence* **28** (1986), pp. 197–224.

[de Kleer & Williams 86] de Kleer, J. and Williams, B. C., "Back to Backtracking: Controlling the ATMS", *Proc. AAAI-86* (1986), pp. 910–917.

[Doyle 79] Doyle, J., "A Truth Maintenance System", *Artificial Intelligence* **12** (1986), pp. 231–272.

[Hanks & McDermott 87] Hanks, S. and McDermott, D., "Nonmonotonic Logic and Temporal Projection", *Artificial Intelligence* **33** (1987), pp. 379–412.

[Ibaraki 77] Ibaraki, T., "The Power of Dominance Relations in Branch and Bound

Algorithms", *J. ACM* **24** (1977), pp. 264–279.

[Inoue 87] Inoue, K., "Resolution Search Strategies for Assumption-based Problem Solving", *Proc. 35th Annual Convention IPS Japan* (1987) (in Japanese).

[Mackworth 77] Mackworth, A. K., "Consistency in Networks of Relations", *Artificial Intelligence* **8** (1977), pp. 99–118.

[McDermott 83] McDermott, D., "Contexts and Data Dependencies: A Synthesis", *IEEE Trans. Pattern Anal. Machine Intelligence* **5** (3) (1983), pp. 237–246.

[Mittal & Frayman 87] Mittal, S. and Frayman F., "Making Partial Choices in Constraint Reasoning Problems", *Proc. AAAI-87* (1987), pp. 631–636.

[Nagai *et al.* 87] Nagai, Y., Kubono, H. and Iwashita Y., "PROTON: Expert System Tool on the PSI Machine", *Proc. Avignon 87: Expert System & their Applications* (1987), pp. 79–98.

[Pearl 84] Pearl, J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, MA, 1984.

[Poole 87] Poole, D. L., *A Logical Framework for Default Reasoning*, Research Report CS-87-54, Department of Computer Science, University of Waterloo, 1987.

[Post 87] Post, S., "Nonmonotonic Reasoning by Minimizing Contradiction in a Hedged Predicate Calculus", *Proc. 3rd Annual Expert System Conference in Government* (1987), pp. 6–10.

[Reiter & de Kleer 87] Reiter, R. and de Kleer, J., "Foundations of Assumption-based Truth Maintenance Systems: Preliminary Report", *Proc. AAAI-87* (1987), pp. 183–188.

[Stallman & Sussman 77] Stallman, R. M. and Sussman, G. J., "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis", *Artificial Intelligence* **9** (1977), pp. 135–196.

[Sussman & McDermott 72] Sussman, G. J. and McDermott, D. V., "From PLANNER to CONNIVER—A Generic Approach", *Proc. AFIPS FJCC* (1972), pp. 1171–1179.

[Zabih *et al.* 87] Zabih, R., McAllester, D. and Chapman, D., "Non-Deterministic Lisp with Dependency-Directed Backtracking", *Proc. AAAI-87* (1987), pp. 59–64.

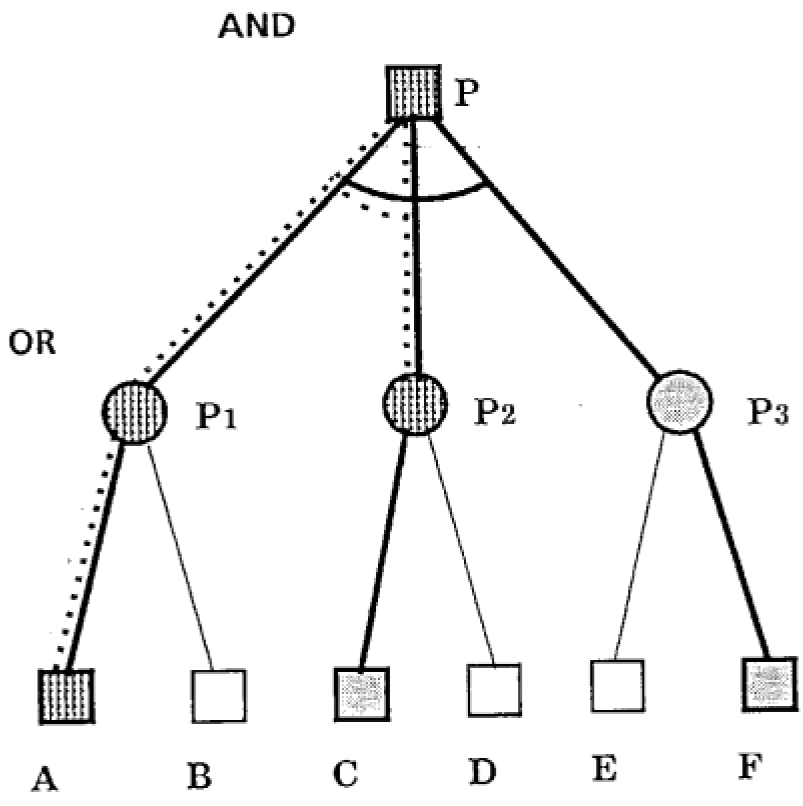

Figure 1: The AND/OR search tree for Example 1.

Loop No.	*Context*	*ADDCLAUSE*(Σ, X)	*D*	*NOGOOD*	*OPEN*	*SOLUTION*
0					$\{P\}$	
1	$\{P\}$	$P \equiv P_1 \wedge P_2 \wedge P_3^*$			$\{P_1\}$	
2	$\{P_1\}$	$P_1 \equiv A \vee B^*$			$\{A\},\{B\}$	
3	$\{A\}$	$A \supset a$	$\{A\}$		$\{P_2, A\},\{B\}$	
4	$\{P_2, A\}$	$P_2 \equiv C \vee D^*$	$\{A\}$		$\{C,A\},\{D,A\},\{B\}$	
5	$\{C, A\}$	$C \supset c$	$\{A,C\}$		$\{P_3,C,A\},\{D,A\},\{B\}$	
6	$\{P_3, C, A\}$	$P_3 \equiv E \vee F^*$	$\{A,C\}$		$\{E,C,A\},\{F,C,A\},$ $\{D,A\},\{B\}$	
7	$\{E, C, A\}$	$E \supset e,\ \neg c \vee \neg e$	$\{A,C\},\{E\}$	$\{C,E\}$	$\{F,C,A\},\{D,A\},\{B\}$	
8	$\{F, C, A\}$	$F \supset f$	$\{A,C,F\},\{E\}$	$\{C,E\}$	$\{D,A\},\{B\}$	$\{A,C,F\}$
9	$\{D, A\}$	$D \supset d,\ \neg a \vee \neg d$	$\{A,C,F\},\{D\},\{E\}$	$\{C,E\},\{A,D\}$	$\{B\}$	$\{A,C,F\}$
10	$\{B\}$	$B \supset b,\ \neg b$	$\{A,C,F\}$	$\{E\},\{D\},\{B\}$	ϕ	$\{A,C,F\}$

Figure 2: The computation process by *GSEARCH* in Example 1.

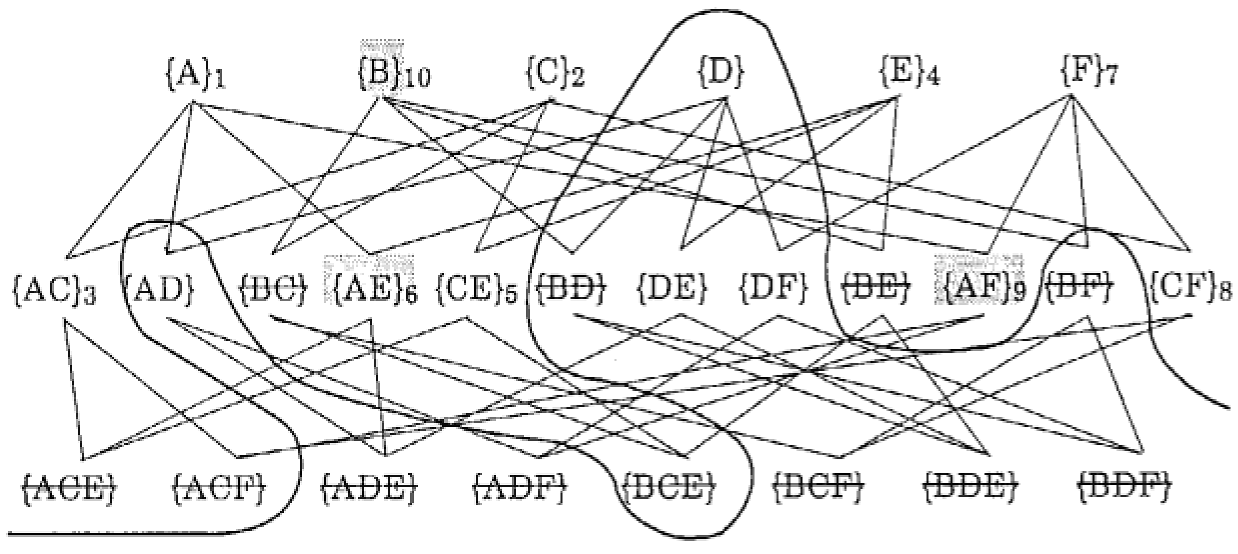

Figure 3: The search space for Example 2.

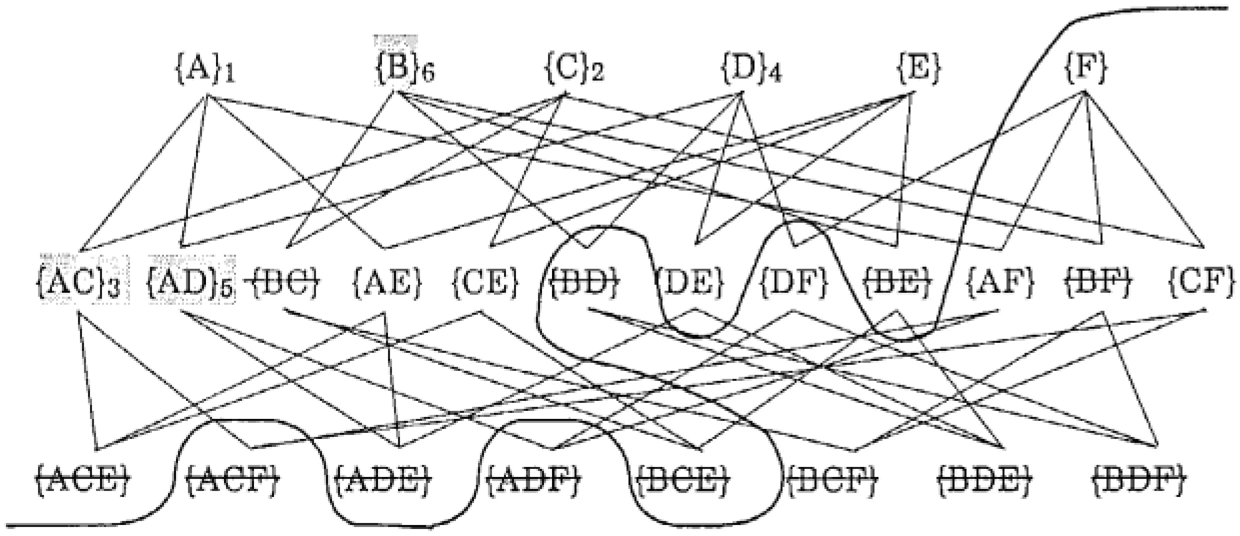

Figure 4: The search space for Example 3.