August 1992

Search Through Systematic Set Enumeration

Ron Rymon
University of Pennsylvania

Search Through Systematic Set Enumeration

Abstract

In many problem domains, solutions take the form of *unordered* sets. We present the *Set-Enumerations* (SE)-tree - a vehicle for representing sets and/or enumerating them in a best-first fashion. We demonstrate its usefulness as the basis for a unifying search-based framework for domains where minimal (maximal) elements of a power set are targeted, where minimal (maximal) partial instantiations of a set of variables are sought, or where a composite decision is not dependent on the *order* in which its primitive component-decisions are taken. Particular instantiations of SE-tree-based algorithms for some AI problem domains are used to demonstrate the general features of the approach. These algorithms are compared theoretically and empirically with current algorithms.

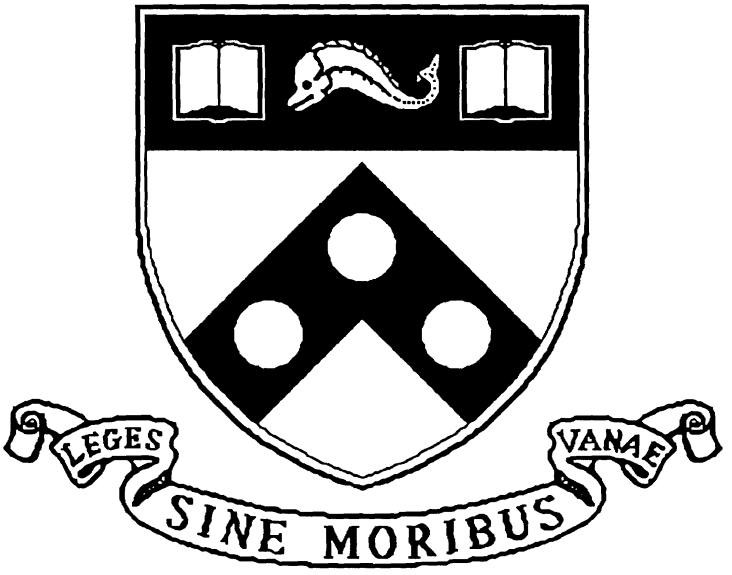
Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-66.

Search Through Systematic Set Enumeration

MS-CIS-92-66
LINC LAB 234

Ron Rymon



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

August 1992

Search through Systematic Set Enumeration

Ron Rymon*
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

In Proceedings KR-92, Cambridge MA, October 1992

Abstract

In many problem domains, solutions take the form of *unordered* sets. We present the *Set-Enumeration* (SE)-tree – a vehicle for representing sets and/or enumerating them in a best-first fashion. We demonstrate its usefulness as the basis for a unifying search-based framework for domains where minimal (maximal) elements of a power set are targeted, where minimal (maximal) partial instantiations of a set of variables are sought, or where a composite decision is not dependent on the *order* in which its primitive component-decisions are taken. Particular instantiations of SE-tree-based algorithms for some AI problem domains are used to demonstrate the general features of the approach. These algorithms are compared theoretically and empirically with current algorithms.

1 INTRODUCTION

Many computer science problems admit solutions which are elements of a given power-set. Typically, such sets are required to satisfy some problem-specific criterion which designates them as solutions. In many cases, such criteria either include, or are augmented with, some minimality/maximality requirement. Consider, for example, the Hitting-Set (HS) problem [Karp 72]. Given a collection of sets, solutions are required to have a non-empty intersection with each member of the collection. In applications of the HS problem, *interesting* solutions are typically minimal with respect to set inclusion. In a more general class of problems, solutions are partial instantiations of a set of variables. A hitting-set, for example, can also be described as a membership-based mapping from the underlying set of primitive elements to $\{0, 1\}$. More generally, variables can be instantiated from an arbitrary domain.

Researchers in Artificial Intelligence (AI) have also made use of such abstract problems in their models. The HS problem, for example, was used by [Reiter 87] in his formalization of diagnosis. In a newer characterization, diagnoses are viewed as partial assignments of state to components [de Kleer et al. 90]. Many other AI problems are, or could be, formulated so as to admit sets as solutions.

Our goal in introducing the *Set-Enumeration* (SE)-tree is to provide a unified search-based framework for solving such problems, albeit their problem-independent solution criteria. SE-tree-based algorithms for different problems will share their skeletal structure, but will each use additional domain-specific *tactics*. Furthermore, at a certain level of abstraction, even those tactics are general and can be shared across domains. General tactics identified here include pruning rules which exploit the SE-tree structure, exploration policies, and problem decomposition methods. Incremental versions of SE-tree-based algorithms can be constructed for some problem domains. In what follows, we use particular instantiations of SE-tree-based algorithms to demonstrate the general features of the approach.

Consistency-based formulations of diagnosis will serve as a working example for most of this paper. Section 2 begins with a formal description of the basic SE-tree. Section 3 presents an SE-tree-based hitting-sets algorithm (SE-HS) for Reiter's original formulation. Being best-first, it can use any of a number of exploration policies. This algorithm is first contrasted, as is, with the original algorithm. The SE-tree structure is then used to improve SE-HS by pruning away unpromising parts of the search space. We conclude with an empirical comparison of the two algorithms.

In Section 4, we extend the SE-tree to fit the more general class of problems, where solutions take the form of partially instantiated sets of variables. Section 5 presents an extended version of SE-HS for a newer

*Address for correspondence: Ron Rymon, Computer and Information Science, Room 423C, 3401 Walnut Street, Philadelphia PA 19104, e-mail: rymon@linc.cis.upenn.edu.

characterization of diagnoses [de Kleer et al. 90]. Although derived from a very general search framework, this algorithm corresponds to a prime implicate generation algorithm proposed by [Slagle et al. 70], and is empirically shown to perform quite well compared to a recent algorithm [Ngair 92]. Unlike Slagle et al.'s algorithm, the extended SE-HS can work under diagnostic theories with *multiple* fault modes, can use a variety of exploration policies for focusing purposes, and has an incremental version. Furthermore, we subsequently augment it with a problem decomposition tactic, thereby obtaining an improved version of Slagle et al.'s algorithm. Finally, we briefly review potential use of the SE-tree in abductive diagnostic frameworks.

In Section 6, we contrast features of the SE-tree with decision trees in the context of learning classification rules from examples. For lack of space, the scope of this study is very limited and the reader is referred to [Rymon 92b] for a more detailed analysis and empirical evaluation.

2 THE BASIC SE-TREE

The *Set-Enumeration* (SE)-tree is a vehicle for representing and/or enumerating sets in a best-first fashion. The *complete* SE-tree systematically enumerates elements of a power-set using a pre-imposed order on the underlying set of elements. In problems where the search space is a subset of that power-set that is (or can be) closed under set-inclusion, the SE-tree induces a complete irredundant search technique. Let E be the underlying set of elements. We first index E's elements using a one-to-one function $ind : E \rightarrow \mathbb{N}$. Then, given any subset $S \subseteq E$, we define its SE-tree view:

Definition 2.1 A Node's View

$$View(ind,S) \stackrel{\text{def}}{=} \{\ e \in E \mid ind(e) > \max_{e' \in S} ind(e')\ \}$$

Definition 2.2 A Basic Set Enumeration Tree

Let F be a collection of sets that is closed under $\subseteq$ (i.e. for every $S \in F$, if $S' \subseteq S$ then $S' \in F$). T is a Set Enumeration tree for F iff:

1. *The root of T is labeled by the empty set;*
2. *The children of a node labeled S in T are* $\{\ S \cup \{e\} \in F \mid e \in View(ind,S)\ \}$.

Figure 1 illustrates an SE-tree for the complete power-set of $\{1,2,3,4\}$. Note that restricting a node's expansion to its *View*, ensures that every set is *uniquely* explored within the tree. By itself, the idea of using an imposed order is not new; it is used for similar purposes in many specific algorithms. Our contribution is in identifying the SE-tree as a recurring search structure, thereby facilitating its use in a general framework and the sharing of particular tactics.

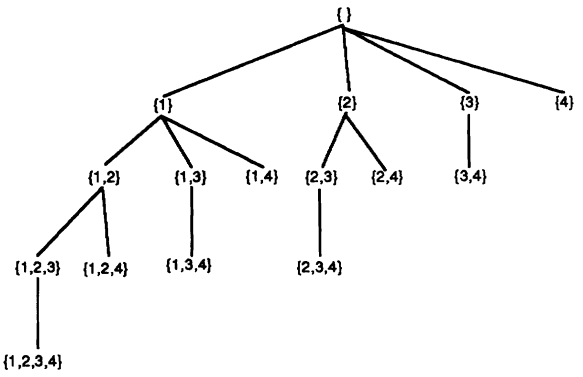


Figure 1: SE-tree for $\mathcal{P}(\{1,2,3,4\})$

Notice also that the SE-tree can be used as a data structure for *caching* unordered sets, and as an effective means of checking whether a new set is subsumed by any of those already cached. [de Kleer 92] has made such use of an SE-tree and reports significant improvements in run-time. As a caching device, the SE-tree is a special case of Knuth's *trie* data structure [Knuth 73], originally offered for *ordered* sets. While we too use the SE-tree for caching solutions and for subsumption checking, our main objective in this paper is its use in a search framework.

3 AN SE-TREE-BASED HITTING-SET ALGORITHM

In this section, we demonstrate the use of the basic SE-tree structure for a hitting-set algorithm in the context of Reiter's theory of diagnosis. We open with a brief introduction of Reiter's theory, to the point in which a hitting-set problem is formulated. An SE-tree-based algorithm (SE-HS) is then contrasted with the dag-based algorithm proposed by [Reiter 87, Greiner et al. 89] to show that a large number of calls to a subsumption checking procedure can be saved. Then, the SE-tree systematicity allows improving SE-HS via a domain-specific pruning rule. Empirical comparison of the improved SE-HS with the dag-based implementation of [Greiner et al. 89] supports our claims.

3.1 REITER'S THEORY OF DIAGNOSIS

Reiter's theory of diagnosis [Reiter 87] is among the the most widely referenced logic-based approaches to model-based diagnosis. For lack of space, we shall only present the concepts and theorem which Reiter uses to derive his hitting-set algorithm.

Definition 3.1 A Diagnostic Problem [Reiter 87]

A diagnostic problem is a triple (SD,COMPS,OBS)*:*

1. SD – *the system description, is a set of first order sentences;*

2. COMPS $\stackrel{\text{def}}{=} \{c_i\}_{i=1}^{n}$ – *the system's components, is a finite set of constants; and*

3. OBS – *the observations, is also a set of first order sentences.*

The language in which diagnostic problems are expressed is thus first order, and is augmented with an extra AB predicate (for abnormal).

Definition 3.2 Conflict Set

Given a diagnostic problem, a conflict is a set of components that cannot all *be functioning correctly. Let* CONFLICTS *denote the collection of conflict sets.*

Theorem 3.3 [Reiter 87] *Given a diagnostic problem, minimal diagnoses are precisely the minimal hitting sets for* CONFLICTS.

Reiter's algorithm is an implementation of Theorem 3.3. In two steps, it first discovers conflicts, and then runs an HS algorithm on the conflicts discovered. We shall concentrate on the latter phase.

3.2 DAG-BASED APPROACH

Given a collection of conflict sets, Reiter's algorithm grows an HS-tree in which nodes represent partial hitting sets and leaves represent complete ones. To avoid highly redundant exploration, Reiter augments this basic algorithm with a set of rules for *reusing* and *pruning* nodes. [Greiner et al. 89] present a correction to this algorithm which uses a directed acyclic graph (dag). It proceeds as follows:

1. Let D represent a growing HS-dag. Label its root with an arbitrary $C \in$ CONFLICTS;
2. Process nodes in D in a breadth-first order. To process a node n:
   (a) Let $H(n)$ be the set of edge labels on the path from the root to n. If $H(n)$ hits all sets in CONFLICTS, mark it as a minimal hitting set. Otherwise, label n with the first set of CONFLICTS which is not hit by $H(n)$.
   (b) If n is labeled by Σ, generate a downward arc labeled by any $\sigma \in \Sigma$.

This algorithm is augmented with three types of rules for expanding a node n:

1. *Reusing:* If there is another node m for which $H(m) = H(n) \cup \{\sigma\}$, do not expand n, but rather link it to m, labeling that link with σ.
2. *Closing:* If there is a node m which is marked as a hitting set, such that $H(m) \subseteq H(n)$, then close n, i.e. do not expand it at all.
3. *Pruning:* If a set Σ is to label a node n and it has not been used previously, then try to prune D:
   (a) If there is a node m which has been labeled with a set S' such that $\Sigma \subseteq S'$, then relabel m with Σ. Prune all edges from m with arcs labeled with σs from $S'-\Sigma$.
   (b) Interchange S' and Σ in CONFLICTS.

3.3 SE-TREE-BASED ALTERNATIVE

SE-HS (Algorithm 3.4) is an SE-tree-based hitting set algorithm. In a best-first fashion, it explores nodes in an order conforming to some predetermined priority function. For that purpose, nodes along the tree's expanding fringe are kept in a priority queue and the next node to be expanded is accessed via the *Next-Best* operation. Prioritization allows implementation of various exploration policies, to be discussed shortly. Let us first assume that nodes are explored by their cardinality; i.e. breadth-first.

Algorithm 3.4 Finding Minimal Hitting Sets

Program SE-HS (CONFLICTS)

1. *Let HS* $\leftarrow \{\}$; *OPEN-NODES* $\leftarrow \{\{\}\}$
2. *Until OPEN-NODES is empty do*
3. *Expand (Next-Best(OPEN-NODES))*

Procedure Expand(S)

1. *Let Window(S)* $\leftarrow \{\ c \mid c \in$ *View(ind,S)* $\}$
2. *For each* $c \in$ *Window(S) which is a member of some set from NYH(S) do*
3. *Unless there is* $S' \in HS$ *such that* $S' \subseteq S \cup \{c\}$
4. *If* $S \cup \{c\}$ *is a hitting set, add it to HS;*
5. *Otherwise, add it to OPEN-NODES.*

The main SE-HS program simply implements a best-first search. The algorithm's functionality is embodied in its *Expand* procedure, where the SE-tree structure is used, and where hitting sets are identified. In choosing viable expansions for a node labeled S, we restrict ourselves to components within S's *View*. Such components are also required to participate in conflicts not yet hit by S (denoted $NYH(S)$). Step 3 in *Expand* prunes away nodes subsumed by minimal hitting sets. It corresponds to the *closing* step in [Greiner et al. 89]. However, SE-HS *avoids* the redundancy for which *reusing* rules were devised, and does not require *pruning*.

Theorem 3.5 *If nodes are prioritized by their label's cardinality, then SE-HS is correct (produces all and only minimal hitting sets.)*

3.4 PROJECTED GAIN

The HS-dag algorithm uses three pruning rules, each of which is computationally expensive and requires numerous calls to a subsumption checking procedure. In examining the purpose of these rules, we note that (1) *Reusing* is aimed at avoiding redundancy in search, i.e. the phenomena that same part of the search space is repeatedly explored within the HS-tree. It requires comparing each new node to *every* previous node; (2) *Closing* is aimed at shutting nodes which are supersets of minimal diagnoses. For that purpose, if the HS-dag is explored breadth-first, each node will only have to be compared against previous minimal hitting sets; finally (3) *Pruning* is aimed at "correcting" the HS-dag from the effects of non-minimal conflict sets. The same effect could also be achieved *a priori*, by "sorting" CONFLICTS by cardinality.

As previously explained, while closing cannot be avoided, SE-HS requires neither reusing, nor pruning. Avoiding numerous calls to a subsumption checking procedure results in a tremendous improvement in run time (see Section 3.7).

3.5 EXPLORATION POLICIES

Due to its potentially exponential size, it may often be impossible to completely explore the space of sets. In such cases, it may be beneficial to characterize *partial* outputs of an SE-tree-based algorithm, given a variety of exploration policies

Definition 3.6 Correct Exploration Policy

An exploration policy is a priority function ψ, defined for each set. It is correct if whenever open nodes are so prioritized, the resulting algorithm is correct.

For the particular case of SE-HS, a variety of exploration policies are sensible.

Proposition 3.7 *Any monotonic function ψ (i.e. such that for every $S \subseteq S'$ we have $\psi(S) \leq \psi(S')$) is a correct exploration policy for SE-HS.*

We have already seen that exploration by cardinality is correct. Simpler diagnoses are explored first using this exploration policy. Other interesting policies include exploration by *probability* ($\psi(S) \stackrel{\text{def}}{=} Prob(S \text{ is a diagnosis})$), and by *utility* or some other monotonic external criterion imposed on sets.

3.6 PRUNING UNPROMISING PARTS OF THE SEARCH SPACE

So far, nodes were pruned only if subsumed by known hitting-sets, thereby using the minimality requirement and the monotonicity of the SE-tree with respect to set-inclusion. We have not used the systematic ordering of nodes in the SE-tree for that purpose. That ordering provides a *restriction* on node labels which can occur in a given node's sub-tree. More specifically, let S be a node's label, then the sub-tree rooted at that node will only have nodes whose labels are expansions of S with components from $View(ind, S)$. Thus, in choosing viable expansions for S, we can restrict ourselves to expansions such that every set that will *not* be hit by the expanded set will still contain components within its *View* (and thus stand the chance of being hit by any of that node's descendants).

This is, in fact, a general feature of an SE-tree-based search program: the systematic enumeration embedded in the SE-tree structure allows us to ignore parts of the space which do not have the *potential* to lead to a solution.

To incorporate this pruning rule into SE-HS, it is sufficient to modify the node expansion routine.

Algorithm 3.8 Node Expansion (version 2)

Procedure Expand(S)

1. *Let Window(S)*$\leftarrow$
   $\{ c \mid c \in View(ind,S) \} \cap$
   $\{ c \mid ind(c) \leq \min_{S' \in NYH(S)} \max_{c' \in S'} ind(c') \}$
2. *For each $c \in Window(S)$ which is a member of some set from NYH(S) do*
3. *Unless there is $S' \in HS$ such that $S' \subseteq S \cup \{c\}$*
4. *If $S \cup \{c\}$ is a hitting set, add it to HS;*
5. *Otherwise, add it to OPEN-NODES.*

This algorithm is identical to Algorithm 3.4, except for the additional restriction in line 1. This change is an example of a domain-specific SE-tree-based pruning rule. The algorithm remains correct, but fewer nodes need be explored. We demonstrate this in the next section by way of an example, and via empirical experiments.

3.7 DEMONSTRATED GAIN

To demonstrate the advantages of SE-HS over the dag-based algorithm, we will first work through a complete example (taken from [Reiter 87]), and will then present the results of extensive empirical experiments.

Example 3.9 Consider the following collection of conflicts $\{\{2,4,5\}, \{1,2,3\}, \{1,3,5\}, \{2,4,6\}, \{2,4\}, \{2,3,5\}, \{1,6\}\}$ [Reiter 87]. Figure 2 depicts the corresponding HS-dag, where O's mark hitting sets, and X's denote closed nodes. The rightmost branch from the root was pruned by the last node to be explored (itself a descendant of that branch).

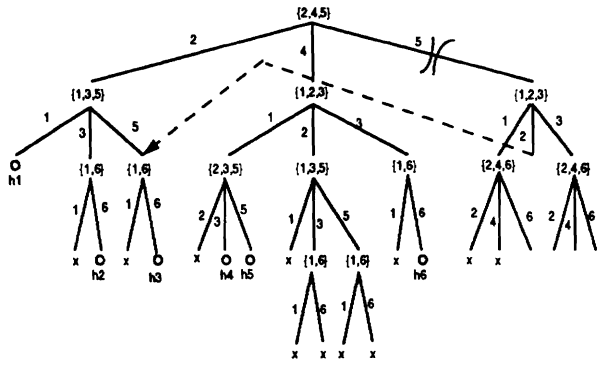


Figure 2: An HS-dag

In contrast, figure 3 shows an SE-tree for the same problem. In comparing, note the difference in notation: nodes in the HS-dag are labeled with the set that is chosen to be hit next whereas the SE-tree's labels are partial hitting sets. In the HS-dag the partial hitting sets label the path from the root to the particular node. The new pruning rule results in fewer nodes being explored: 16 in SE-HS versus 34 in HS-dag.

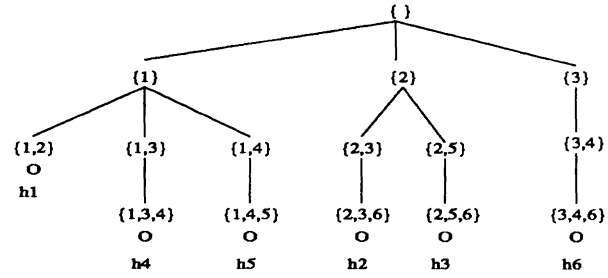


Figure 3: An SE-tree

In addition, we have empirically compared SE-HS's performance with that of an HS-dag implementation which was provided to us by Barbara Smith [Greiner et al. 89]. The run-time and number of nodes generated by each of the two implementations were tested on hundreds of randomly generated test cases. Test cases were generated using three parameters: number of conflicts (denoted #conf), number of components in each conflict (#lit), and overall number of components (#comp). Figures 4 and 5 use logarithmic scale to present one-way sensitivity analyses with respect to each of the parameters and with respect to both run-time (CPU seconds) and nodes generated. SE-HS's performance is indicated by the shadowed squares, and that of HS-dag by the open ones. Each data point was obtained by averaging each algorithm's performance on 10 random cases with same parameters.

4 THE EXTENDED SE-TREE

Sometimes the space being searched consists not of sets of components, but rather of sets of partially instantiated attributes (variables). We next extend the SE-tree accordingly.

Definition 4.1 Partial Descriptions

Let $ATTRS \stackrel{\text{def}}{=} \{A_i\}_{i=1}^{n}$ *be a set of attributes, with domains* $\{Dom(A_i)\}_{i=1}^{n}$. *A partial description is a subset of ATTRS, each of which is instantiated with one value from its domain. It is complete if all attributes are instantiated.*

Consider, for example, the space defined by 3 boolean attributes. The set $\{A_2{=}T\}$ is a partial description in that space. $\{A_1{=}T, A_2{=}F, A_3{=}F\}$ is a complete description.

As with its basic counterpart, to define the *extended* SE-tree we first impose an ordering (*ind*) on ATTRS, and define a node's *View* as all attributes ranked higher than the highest ranked attribute participating in that node. Then,

Definition 4.2 An Extended Set Enumeration Tree

Let F be a collection of sets of attribute instantiations such that each set contains at most one value for each attribute and such that F is closed under $\subseteq$, *then T is an extended SE-tree for F iff:*

1. *The root of T is labeled by the empty set;*
2. *The children of a node S in T are*

   $\{S \cup \{(A{=}v)\} \in F \mid A \in View(ind,S),\ v \in Dom(A)\}$.

Figure 6 depicts an extended SE-tree for the complete space defined by three boolean attributes. Note the use of reduced notation where i stands for $\{A_i = T\}$, and $-i$ represents $\{A_i = F\}$.

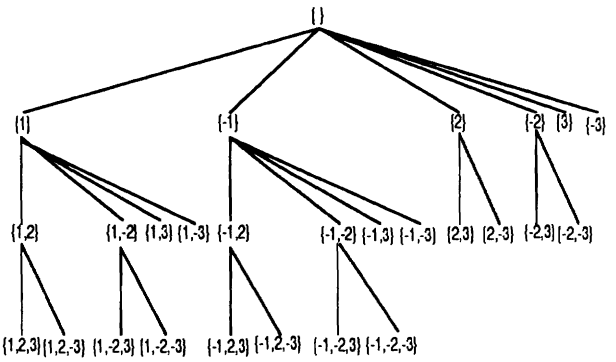


Figure 6: Complete SE-tree for 3 Boolean Attributes

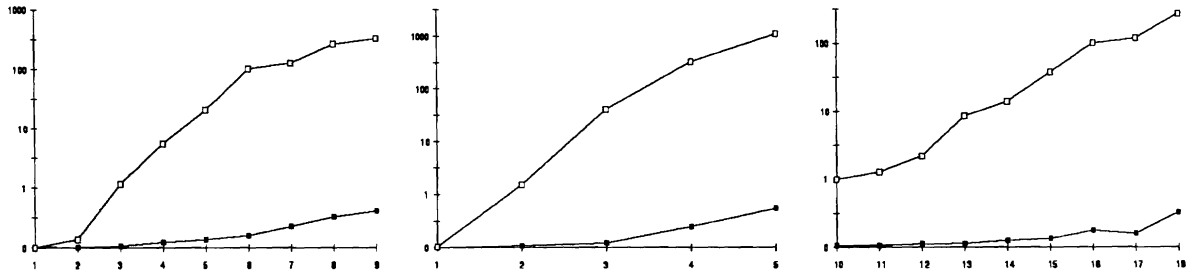


Figure 4: Run Time of SE-HS (■) versus HS-dag (□)

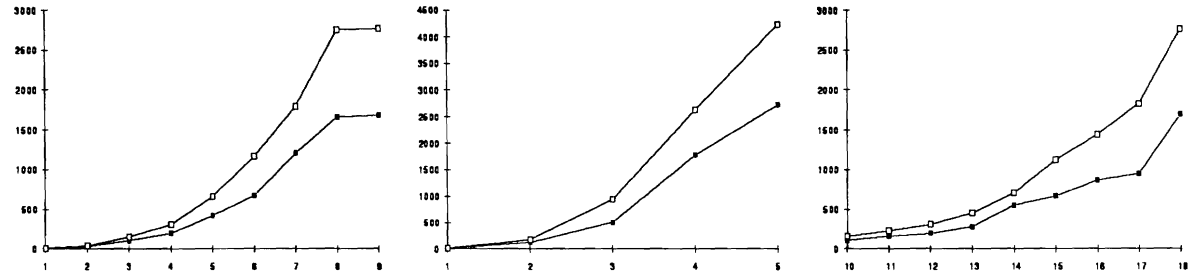


Figure 5: Number of Nodes Explored by SE-HS (■) versus HS-dag (□)

5 SE-TREE-BASED PRIME IMPLICATE ALGORITHM

In this section, we present an extension of SE-HS and demonstrate its use for the diagnostic framework of [de Kleer et al. 90]. We begin with a short description of the extended theory where kernel diagnoses are characterized as prime implicants of the (newly defined) set of conflicts. An extension of SE-HS, presented next, can be used to find those kernel diagnoses. The extended SE-HS has other useful properties: it can be flexibly focused; it can work with multiple behavioral modes; and it has an incremental version. A two-mode restriction of this algorithm corresponds to an old prime implicate generation algorithm [Slagle et al. 70]. We first demonstrate the empirical performance of this restricted version compared to a recent prime implicate generation algorithm. We then augment it with a new problem decomposition tactic, thereby obtaining an improved algorithm for prime implicate generation.

5.1 EXTENDED THEORY OF DIAGNOSIS

[de Kleer et al. 90] extended Reiter's theory with the notion of kernel diagnoses. Rather than having a diagnosis represent only faulty components (with the implicit assumption that all other components function properly), the new theory allows a diagnosis to explicitly specify working *and* non-working condition, without any presumption about other components' state.

Definition 5.1 AB-Clause [de Kleer et al. 90]

Let an AB*-literal be* AB*(c), or* ¬AB*(c) for some* c∈COMPS. *An* AB*-clause is a disjunction of* AB*-literals containing no complementary pair of* AB*-literals. An* AB*-clause is positive if all its* AB*-literals are positive.*

Definition 5.2 Conflict [de Kleer et al. 90]

A conflict is any AB*-clause entailed by SD∪OBS. A conflict set is its underlying set of* AB*-literals.*

Note that the new definition extends Reiter's original definition which, roughly speaking, allows only *positive* conflicts. We shall interchangeably speak about conflicts and their underlying sets.

Definition 5.3 Partial Diagnosis [de Kleer et al. 90]

A partial diagnosis is a conjunction of AB*-literals P such that P is satisfiable (does not contain complementary pairs), and for any other satisfiable conjunction ϕ covered by P, SD∪OBS∪ϕ is satisfiable.*

In other words, not only is P consistent with the system description and the observed behavior, but also any extension of P that assigns either AB, or ¬AB to components not mentioned in P, is also consistent.

Definition 5.4 Kernel Diagnosis [de Kleer et al. 90]

A kernel diagnosis is a partial diagnosis such that the only partial diagnosis which covers it is itself.

[de Kleer et al. 90] use the notion of prime implicants to characterize kernel diagnoses:

Definition 5.5 Prime Implicant [de Kleer et al. 90]

A conjunction π of AB*-literals, containing no complementary pairs, is an implicant of SD∪OBS if it entails every formula in SD∪OBS. It is a prime implicant if it is not covered by any other implicant.*

Theorem 5.6 [de Kleer et al. 90] *The kernel diagnoses are precisely the prime implicants of SD∪OBS.*

There are several early algorithms for computing prime implicants (or prime implicates)[1], used primarily for Boolean minimization (e.g. [Tison 67, Slagle et al. 70]). Recent interest in the AI community, for tasks such as ATMS encoding and circumscription, has yielded new algorithms (e.g. [Ngair 92]) as well as improvements to old algorithms (e.g. [Kean & Tsiknis 90, de Kleer 92]). Next, an extension of SE-HS will be shown to find kernel diagnoses, and therefore to generate all prime implicants of a CNF formula.

5.2 SE-HS EXTENDED

[de Kleer et al. 90] characterize kernel diagnoses as the prime implicants of SD∪OBS. Alternatively, kernel diagnoses can be defined in terms of hitting sets.

Theorem 5.7 Kernel Diagnoses and Conflicts

Let CONFLICTS *be the collection of conflict sets. The kernel diagnoses are precisely those minimal hitting sets for* CONFLICTS *that do not contain complementary pairs of* AB*-literals.*

Two important implications are (a) that SE-HS can be modified to find kernel diagnoses, and (b) that the modified algorithm can also serve to find prime implicants (implicates) in other settings. The proof for an extended version of this theorem can be found in [Rymon 92a]. Algorithm 5.8 presents the extended version of SE-HS's *Expand* procedure; the main program remains as previously described.

[1] Prime implicates are the disjunctive counterparts of prime implicants. Although for the purpose of diagnosis, we will be interested in prime implicants, most algorithms can compute both.

Algorithm 5.8 Node Expansion (version 3)

Procedure Expand(S)

1. *Let Window(S)←*
   $\{ c \mid c \in View(ind,S) \} \cap$
   $\{ c \mid ind(c) \leq \min_{S' \in NYH(S)} \max_{c' \text{ appears in } S'} ind(c') \}$
2. *For each $c \in Window(S)$ for which there exists some $B \in \{$*AB*, ¬*AB*$\}$ such that $B(c)$ participates in some set from NYH(S) do*
3. *Unless there is $S' \in HS$ such that $S' \subseteq S \cup \{B(c)\}$*
4. *If $S \cup \{B(c)\}$ is a hitting set, add it to HS;*
5. *Otherwise, add it to OPEN-NODES.*

The new *Expand* procedure assigns state (AB or ¬AB) to a new component, not yet in the expanded set. The algorithm's correctness is easy to verify.

Besides its simplicity, being derived from a general SE-tree-based framework, SE-HS enjoys the following features:

1. *Focusing facility.* Due to the possibly overwhelming number of hypothetical diagnoses, much research on ATMS-based diagnostic programs has centered on methods for focusing on the most probable solutions (e.g. [Forbus & de Kleer 1988, de Kleer 91]). [Provan & Poole 91] advocate a preference criterion that is based on a diagnosis' *use*. Exploration policies, as in Section 3.5, can be used for that purpose.
2. *Fault models.* The importance of explicit models of faulty behavior has been recognized in the model-based diagnosis community (e.g. [Holzblatt 88, de Kleer & Williams 89]). In [Rymon 92a], we extend the diagnostic theory of [de Kleer et al. 90] to multiple behavioral modes and prove that kernel diagnosis in the new theory can still be characterized in terms of hitting sets. SE-HS can be easily extended to any number of behavioral modes.
3. *Incrementalism.* [Rymon 92a] outlines an incremental diagnostic framework that is based on a variation of SE-HS which can incrementally refine its hypothesis as conflicts arrive.

5.3 PERFORMANCE EVALUATION

We have implemented the extended SE-HS algorithm and have compared its performance to that of a PHI-based prime implicate generation algorithm [Ngair 92]. As before, the two algorithms were run on hundreds of examples that were randomly generated according to the three parameters (#conf, #lit, #comp). Due to the relative strength of both algorithms, we used larger examples in this experiment. As a side note,

the SE-HS implementation is general in that it can take any number of behavioral modes. This generality is not useful in the experiment, where examples are bi-modal. Figure 7 depicts two one-way sensitivity analyses (for #conf, #lit) and one three-way analysis. Again, shadowed squares correspond to SE-HS performance, open ones to that of the PHI-based algorithm.

5.4 PROBLEM DECOMPOSITION

As so far presented, we could draw a correspondence between nodes explored by the bi-modal version of the extended SE-HS algorithm and the operation of an old prime implicate generation algorithm proposed by [Slagle et al. 70]. This is important for two reasons: first, it reveals the general SE-tree-based features of Slagle et al.'s algorithm, but more importantly, our next improvement to SE-HS will result in an improved version of their algorithm.

Where feasible, problem decomposition (also referred to as divide-and-conquer) is a well known strategy to sharply reduce problem solving costs (time, space, etc.) In the context of diagnosis, such an opportunity may arise when a fault is composed of a number of unrelated, or partially related sub-faults. [Wu 90] shows tremendous gain in utilizing problem decomposition techniques in diagnosis.

In the context of multiple fault diagnosis, in addition to potential saving of time and space, decomposition may also lead to more compact *representation* of a solution. In many cases, a solution can be *written* more compactly if it is factored. For example, a solution of the form $\times_{i=1}^{n}$ { AB(c_{2i-1}), AB(c_{2i}) }, when expanded, consists of 2^n minimal diagnoses. Put differently, like formulae, some solutions can be represented compactly as CNF whereas others are more concise in their disjunctive form. This is, roughly, the intuition behind the following heuristic.

Theorem 5.9 Problem Decomposition

If CONFLICTS *can be partitioned into two disjoint subsets C' and C", such that no component appears in both subsets, then the minimal hitting sets (MHS) for* CONFLICTS *are given by:*

$$\text{MHS(CONFLICTS)} = \text{MHS(C')} \times \text{MHS(C")}$$

If a partition exists, it can clearly save significant work. Recursive application of SE-HS to each of the two partitions can cut the exponential search space into two smaller search spaces. The notion of partitioning can be extended to *any* number of partitions, making the latter equivalence classes and making the partitioning *unique*. The solution in such case is the *Cartesian product* of the sub-solutions.

Fortunately, if one exists, there is a simple, almost-linear, algorithm that finds a partitioning for a collection of sets (e.g., using a union-find strategy [Tarjan 83]). Moreover, even if there is no facilitating partitioning to begin with, it is possible that one exists when a node's particular view is considered. Given a node S, recall that any of S's descendants will only expand with respect to $View(ind, S)$. Thus, it is enough to look for a partition in the restriction of $NYH(S)$ to $View(ind, S)$.

Algorithm 5.10 An Amendment to Expand

1. *Let* Γ *be the restriction of NYH(S) to components in View(ind,S).*
2. *If there is a partitioning* $\Gamma = \times_{i=1}^{k} \Gamma_i$ *then*
3. *Run SE-HS on each of the* Γ_i *independently. Let Hitting*(Γ_i) *be the corresponding results, merge* $\{S\} \times (\times_{i=1}^{k}$ *Hitting*(Γ_i)) *into HS while checking for possible subsumption.*
4. *Otherwise, expand S as usual.*

Exact prioritization is a problem in the augmented algorithm since every node in a new tree represents only part of (possibly many) solutions. For similar reasons, subsumption has to be more aggressively monitored (although this is easily done when hitting sets are cached in an SE-tree-based data structure). Before, subsumption was avoided by the subsuming solution being discovered prior to the subsumed one. Now, it is possible that a solution node in the original SE-tree will be subsumed by some but not all of the solutions in which a given node in some new tree participates. Nevertheless, problem decomposition is still attractive since it is particularly effective in problems which admit highly disjunctive solutions. Those are hardest for the original SE-HS algorithm. The following example demonstrates the effectiveness of the problem decomposition heuristic.

Example 5.11 Consider the following collection of conflicts: { {AB(1),¬AB(3),AB(4)}, {AB(5),AB(6)}, {AB(3),AB(4)}, {AB(2),¬AB(6)}}. Figure 8 illustrates the SE-tree explored by SE-HS *without* decomposition. As before, O's denote hitting sets, X's mark closed nodes. Exploration for the same problem *with* decomposition is depicted in Figure 9. There, the first step involved partitioning the collection of conflicts into two disjoint sets. Thereafter, two sub-problems are solved, and the solution is the cross-product of the respective results, i.e. {{AB(1),AB(3)}, {AB(4)}} × {{AB(2),AB(5)}, {AB(2),AB(6)}, {AB(5),¬AB(6)}}. The reductions in time and space are obvious.

5.5 ABDUCTIVE DIAGNOSTIC MODELS

In [Reggia et al. 85], diagnosis is formulated as a generalized set covering (GSC) problem. In their basic

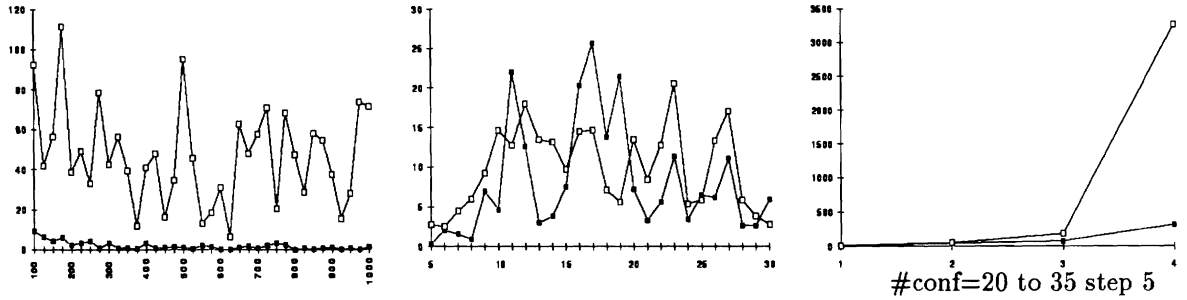


Figure 7: Run Time of extended SE-HS (■) versus PHI-based algorithm (□)

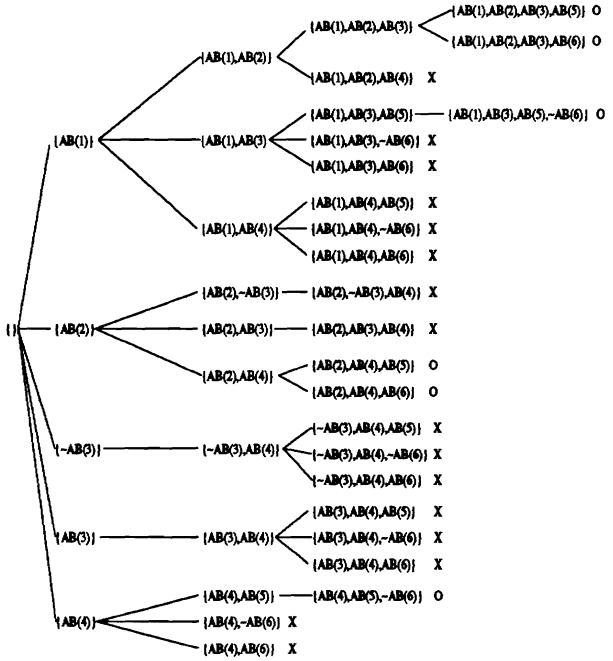


Figure 8: SE-tree without Decomposition

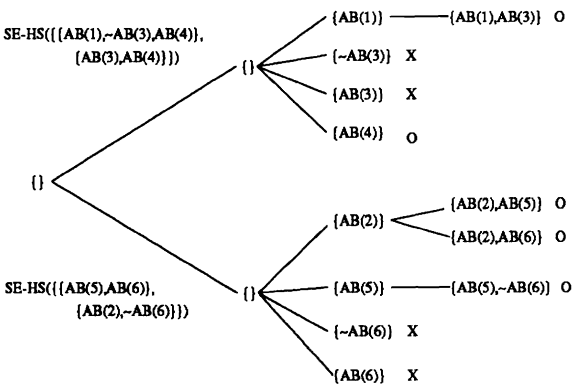


Figure 9: SE-tree Exploration with Decomposition

model, a diagnostic problem is represented in a bipartite graph in which symptoms and disorders form each of the respective partitions. Each disorder in the graph is linked to all of its symptoms via a *causes* relation. Given a set of observed symptoms, a diagnosis is defined as a minimal set of disorders which covers all symptoms.

A most-probable-first search algorithm for that problem is described in [Peng & Reggia 87]. It searches the space of sets of disorders for such sets which cover all symptoms. This algorithm, however, is redundant in that partial hypotheses may be discovered repeatedly during search. That redundancy could be avoided if an SE-tree framework were adopted.

Alternatively, the problem can be turned into a hitting set problem. [Reiter 87] presents a transformation of a GSC representation of a diagnostic problem into his own framework. There is, in fact, a better transformation which avoids the conflict generation part of Reiter's theory by mapping the GSC problem directly into an HS one. Then, we could simply use SE-HS. Given a set of symptoms s_i, we could define a "conflict set" for each symptom:

$$\text{conflict}(s_i) \stackrel{\text{def}}{=} \{d \mid d \text{ is a disease, } d \textit{ causes } s_i\}$$

Presented with s_i, the conflict asserts that it is impossible that none of its causing disorders are present. It is easy to prove that a set of disorders is a minimal set cover iff it is a minimal hitting set for such conflicts.

In [Peng & Reggia 87], hypotheses are explored by their likelihood. The SE-tree-based framework allows such exploration, as well as a variety of other exploration policies. In [Peng & Reggia 87], non-minimal hypotheses are also explored. This is easily done in SE-HS by removing the subsumption requirement (*Expand*, step 3). In addition, pruning rules (cf. Section 3.6) can be used to avoid unpromising parts of the search space. Problem decomposition (cf. Section 5.4) may also be helpful in reducing time and cost. Finally, it seems that other models of diagno-

sis in which solutions are defined in terms of sets, e.g. [Bylander et al. 91, Poole 91, Console & Torasso 91], can also use an SE-tree-based search framework in their implementations.

6 LEARNING MINIMAL CLASSIFICATION RULES

Decision trees are an important tool, and serve as an underlying representation in many problem solving tasks. Significant research in Machine Learning has used decision trees in architectures for induction of classification knowledge from examples. Best known are ID3 [Quinlan 86] and its descendants. In [Rymon 92b], we present an SE-tree-based characterization of the induction task, contrast it from classification and search perspectives with the decision-tree-based framework, and compare the two empirically. Here, we will only contrast features of the two representations, concentrating on search aspects.

Definition 6.1 Rules

A training set (TSET) is a collection of examples. Each example is a complete description for which a correct classification (denoted π) is known. A rule is a partial description R such that if $t,t' \in TSET$ are such that $R \subseteq t,t'$, then $\pi(t) = \pi(t')$. It is minimal if none of its subsets is a rule.

The objective of a learning system is to learn rules that can be expected to perform well not only on the training set, but also on new examples. While there is no consensus as to the precise composition of such a collection, it is fairly acceptable that *general* (minimal) rules are preferable to specific ones. We shall therefore concentrate on finding minimal classification rules[2].

6.1 PROPOSED SOLUTION

ID3 constructs a decision tree in which internal nodes are labeled with attributes, edges with instantiations of these attributes, and leaves with a class prediction. Briefly, the tree is constructed by successively partitioning the set of training examples until all remaining examples are equally classified. Such node becomes a leaf and is labeled with that class.

While construction of an arbitrary decision tree that correctly classifies the training data is straightforward, it is well known that the success of decision-tree-based algorithms on future data is crucially dependent on the particular order in which the attributes were chosen in the successive refinement steps [Fayyad & Irani 88, Goodman & Smyth 88]. As Quinlan notes, one can often not afford to generate all possible decision trees in order to choose the best one. Thus, ID3 (as do other algorithms) uses a heuristic to guide its choice of attributes. One prominent heuristic is based on entropy-minimization, using Shannon's information-theoretic measure.

6.2 SE-TREE-BASED ALTERNATIVE

Aimed at *all minimal* rules, SE-Learn (Algorithm 6.3) uses an SE-tree-based framework. As before, open nodes are prioritized, facilitating various exploration policies. In the context of learning, these will be used to represent *bias* and will be briefly discussed in the end of this section. As before, SE-Learn exploits the systematic ordering to prune away unpromising parts of the search space (i.e. nodes which cannot lead to minimal rules).

Definition 6.2 Candidate Expansions

Let S be a node, $TSET(S) \stackrel{\text{def}}{=} \{t \in TSET \mid S \subseteq t\}$. We say that $(A=v)$ is a candidate expansion of S if $A \in View(ind,S)$, $v \in Dom(A)$, and in addition $TSET(S \cup \{(A=v)\}) \neq TSET(S)$. A node S will be called impotent if either (1) TSET(S) is empty; or (2) there exist $t,t' \in TSET(S)$ disagreeing on their class, and only differing in their assignment to attributes not in View(ind,S).

Algorithm 6.3 Induction of Minimal Rules

Program SE-Learn (TSET)

1. *Let RULES $\leftarrow$ {}, OPEN-NODES $\leftarrow$ {{}}*
2. *Until OPEN-NODES is empty do*
3. *Expand(Next-Best(OPEN-NODES))*

Procedure Expand(S)

1. *For each candidate expansion $(A=v)$, let $R \stackrel{\text{def}}{=} S \cup \{(A=v)\}$, do*
2. *If R is not impotent, nor is it subsumed by any $R' \in RULES$ then*
3. *If R is a rule then add it to RULES;*
4. *Or else add it to OPEN-NODES.*

Theorem 6.4 *If open nodes are prioritized by their label's cardinality then SE-Learn is correct (produces all and only minimal rules.)*

Given the incompleteness of the examples with which they are presented, learning programs may often have to choose among a number of candidate classifiers, all of which are consistent with the training

[2] We simplify here. Motivation for learning *all* minimal rules, variations of an SE-tree-based algorithm that learn subsets of this collection, and empirical results are given in [Rymon 92b].

set. External preference criteria, also referred to as *bias* [Mitchell 80], may be necessary for that purpose. Within an SE-tree-based framework, exploration policies can serve in the implementation of such bias. In programs such as SE-Learn, where *all* rules are explored during the learning phase, an exploration policy will serve in the classification of new objects by guiding preference over possibly conflicting rules. In variants of SE-Learn in which only a subset of the rules are learned, an exploration policy will implement a preference among possible subsets. As was the case for SE-HS, any exploration policy that is monotonic will result in a correct algorithm. Important policies include (1) exploration by *cardinality*, where a preference is given to simpler rules; (2) by *probability* (using either a known distribution or frequency in the training set), resulting in preference to characterization of denser parts of the search space; (3) using *Shannon's information-theoretic measure*, preferring more discriminating rules; and (4) by *utility* or some other monotone preference criterion.

6.3 PROJECTED GAIN AND COST

Three related problems arise when a decision tree is used as a framework for search and representation of minimal rules:

1. *The minimality problem – rules will often not be discovered in their minimal form;*
2. *The multiplicity problem – a minimal rule may be discovered repeatedly, disguised in a number of its minimal subsets; and*
3. *The incompleteness problem – some minimal rules may not be discovered at all.*

The minimality problem is often addressed by subsequently pruning the rules extracted from the decision tree [Quinlan 87]. The replication problem, a special case of multiplicity in which sub-trees are replicated within a single decision tree, has been addressed by several researchers, e.g. [Rivest 87, Pagallo & Haussler 90]. The more general multiplicity problem, however, may take many other forms. Incompleteness is the result of the mutual exclusiveness property of decision-tree-based rules (see [Weiss & Indurkhya 91]).

In contrast, the SE-tree-based framework does not suffer from these problems:

1. *Rules are always discovered in minimal form;*
2. *Minimal rules are always discovered uniquely; and*
3. *All minimal rules are discovered.*

The fact that *any* given decision tree may suffer from those problems suggests that none is globally optimal. The SE-tree, however, can be shown to embed many decision trees. More specifically, *all* decision trees in which attributes are chosen monotonically with respect to some arbitrary indexing, are topologically and semantically equivalent to a tree formed from a subset of the SE-tree's edges.

Complexity-wise, the SE-tree's exhaustiveness and relatively large *initial* branching factor are deceiving. Its complexity is fairly close to that of a *single* decision tree:

Theorem 6.5 SE-Tree Size

If all attributes are b-valued, then the number of nodes in a complete decision tree is $\Sigma_{i=0}^{n} b^i > b^n$. *In sharp contrast, the size of a super-tree in which all decision trees are embedded is significantly larger:* $b^n \cdot n!$. *The size of a complete SE-tree is only* $(b+1)^n$.

7 Summary

Many problems in which partial sets or partially instantiated set of variables are targeted share a common structure when viewed as search problems. We presented the *Set-Enumeration* (SE)-tree as a simple, complete and irredundant vehicle for representing and/or enumerating sets in a best-first fashion. As such, it can serve as the basis for a search-based framework for many such problems.

To demonstrate its usefulness and effectiveness, we presented SE-tree-based algorithms for the hitting-set problem, in the context of consistency-based diagnosis. We used the particular instantiations of these algorithms to demonstrate general features of the paradigm, and compare it with current algorithms. Throughout this process, we developed several add-on tactics including SE-tree-based pruning rules, exploration policies, and problem decomposition methods. Besides their particular incarnations in the SE-HS algorithms, those methods are general and can be shared across many problem domains. In the last part of this paper, in the context of rule induction, we compared features of an SE-tree-based representation with one that is based on decision trees.

Acknowledgements

This research was supported in part by a graduate fellowship ARO Grant DAAL03-89-C0031PRI. I thank Teow-Hin Ngair, Greg Provan, Russ Greiner, Alex Kean, and Ron Rivest for useful discussions and suggestions. I also thank Kevin Atteson, Michael Niv, Philip Resnik, Jeff Siskind, and Bonnie Webber for comments on previous drafts. Finally, I am grateful to Barbara Smith for providing the HS-dag implementation and Teow-Hin Ngair for providing the code of his prime implicate generation algorithm.

References

[Bylander et al. 91] Bylander, T., Allemang, D., Tanner, M. C., and Josephson, J., The Computational Complexity of Abduction. *Artificial Intelligence* 49, pp. 25-60, 1991.

[Console & Torasso 91] Console, L., and Torasso, P., A Spectrum of Definitions of Model-Based Diagnosis. *Computational Intelligence*, 7, pp. 133-141, 1991.

[de Kleer & Williams 89] de Kleer, J. and Williams, B., C., Diagnosis with Behavioral Modes. *Proc. IJCAI-89*, Detroit MI, pp. 1324-1330, 1989.

[de Kleer et al. 90] de Kleer, J., Mackworth, A. K., and Reiter, R., Characterizing Diagnoses. *Proc. AAAI-90*, pp. 324-330, Boston MA, 1990.

[de Kleer 91] de Kleer, J., Focusing on Probable Diagnoses. *Proc. AAAI-91*, pp. 842-848, Anaheim CA, 1991.

[de Kleer 92] de Kleer, J., An Improved Incremental Algorithm for Generating Prime Implicates. *Proc. AAAI-92*, pp. 780-785, San Jose CA, 1992.

[Fayyad & Irani 88] Fayyad, U. M., and Irani, K. B., What Should be Minimized in a Decision Tree? *Proc. AAAI-90*, pp. 749-754, Boston MA, 1990.

[Forbus & de Kleer 1988] Forbus, K. D., and de Kleer, J., Focusing the ATMS. *Proc. AAAI-88*, pp. 193-198, Saint Paul MN, 1988.

[Goodman & Smyth 88] Goodman, R., and Smyth, P., Decision Tree Design from a Communication Theory Standpoint. *IEEE Trans. on Information Theory*, 34(5), 1988.

[Greiner et al. 89] Greiner, R., Smith, B. A., and Wilkerson R. W., A Correction to the Algorithm in Reiter's Theory of Diagnosis. *Artificial Intelligence*, 41, pp. 79-88, 1989.

[Holzblatt 88] Holzblatt, L. J., Diagnosing Multiple Failures Using Knowledge of Component States. *Proc. IEEE AI Applications*, pp. 139-143, 1988.

[Karp 72] Karp, R. M., Reducibility Among Combinatorial Problems. In Miller and Thatcher eds., *Complexity of Computer Computations*, Plenum Press, New York, pp. 85-103, 1972.

[Kean & Tsiknis 90] Kean, A., and Tsiknis, G., An Incremental Method for Generating Prime implicants/Implicates. *Journal of Symbolic Computation*, 9:185-206, 1990.

[Knuth 73] Knuth, D. E., The Art of Computer Programming, Vol. 3: Sorting and Searching. *Addison Wesley*, 1973.

[Mitchell 80] Mitchell, T. M., The Need for Biases in Learning Generalizations. *TR 5-110*, Rutgers University, 1980.

[Ngair 92] Ngair, T., *Convex Spaces as an Order-Theoretic Basis for Problem Solving*, Ph. D. Thesis, Department of Computer and Information Science, University of Pennsylvania, 1992.

[Pagallo & Haussler 90] Pagallo, G., and Haussler, D., Boolean Feature Discovery in Empirical Learning. *Machine Learning*, 5, pp. 71-99, 1990.

[Peng & Reggia 87] Peng, Y., and Reggia, J. A., Being Comfortable with Plausible Diagnostic Hypotheses. *TR 1753*, University of Maryland, 1987.

[Poole 91] Poole, D., Representing Diagnostic Knowledge for Probabilistic Horn Abduction. *Proc. IJCAI-91*, Sydney, Australia, 1991.

[Provan & Poole 91] Provan, G. M., and Poole, D., The Utility of Consistency-Based Diagnostic Techniques. *Proc. KR-91*, Cambridge MA, pp. 461-472, 1991.

[Quinlan 86] Quinlan, J. R., Induction of Decision Trees. *Machine Learning*, 1(1):81-106, 1986.

[Quinlan 87] Quinlan, J. R., Generating Production Rules from Decision Trees. *Proc. IJCAI-87*, pp. 304-307, 1987.

[Reggia et al. 85] Reggia, J. A., Nau, D. S. and Wang, P. Y., A Formal Model of Diagnostic Inference. I. Problem Formulation and Decomposition. *Information Sciences*, 37, pp. 227-285, 1985.

[Reiter 87] Reiter, R., A Theory of Diagnosis From First Principles. *Artificial Intelligence*, 32, pp. 57-95, 1987.

[Rivest 87] Rivest, R., Learning Decision Lists. *Machine Learning*, 2, pp. 229-246, 1987.

[Rymon 92a] Rymon, R., SE-tree-based Candidate Generation: Systematic Exploration in Model-Based Diagnosis. *In preparation*, 1992.

[Rymon 92b] Rymon, R., An SE-tree-based Characterization of the Induction Problem. *In preparation*, 1992.

[Tison 67] Tison, P., Generalized Consensus Theory and Application to the Minimization of Boolean Functions. *IEEE Trans. on Computers*, 16(4):446-456, 1967.

[Slagle et al. 70] Slagle, J. R., Chang, C, and Lee R. C., A New Algorithm for Generating Prime Implicants. *IEEE Trans. on Computers*, 19(4), 1970.

[Tarjan 83] Tarjan, R., Data Structures and Network Algorithms. *Society for Industrial and Applied Mathematics*, Philadelphia PA, 1983.

[Weiss & Indurkhya 91] Weiss, S. M., and Indurkhya, N., Reduced Complexity Rule Induction. *Proc. IJCAI-91*, pp. 678-684, Sydney, Australia, 1991.

[Wu 90] Wu, T. D., A Problem Decomposition Method for Efficient Diagnosis and Interpretation of Multiple Disorders. *Proc. SCAMC-90*, pp. 86-92, Washington DC, 1990.