Establishing Theoretical Minimal Sets of Mutants

Paul Ammann*, Marcio E. Delamaro†, and Jeff Offutt*
*Software Engineering, George Mason University, Fairfax, VA, USA
Emails: {pammann,offutt}@gmu.edu
†Instituto de Ciências Matemáticas e de Computaçao, Universidade de São Paulo, São Carlos, SP, Brazil
Email: delamaro@icmc.usp.br

***Abstract*—Mutation analysis generates tests that distinguish variations, or mutants, of an artifact from the original. Mutation analysis is widely considered to be a powerful approach to testing, and hence is often used to evaluate other test criteria in terms of mutation score, which is the fraction of mutants that are killed by a test set. But mutation analysis is also known to provide large numbers of redundant mutants, and these mutants can inflate the mutation score. While mutation approaches broadly characterized as reduced mutation try to eliminate redundant mutants, the literature lacks a theoretical result that articulates just how many mutants are needed in any given situation. Hence, there is, at present, no way to characterize the contribution of, for example, a particular approach to reduced mutation with respect to any theoretical minimal set of mutants. This paper's contribution is to provide such a theoretical foundation for mutant set minimization. The central theoretical result of the paper shows how to minimize efficiently mutant sets with respect to a set of test cases. We evaluate our method with a widely-used benchmark.**

***Keywords* - Mutation testing, minimal mutant sets, dynamic subsumption**

I. Introduction

Mutation analysis [5] is an approach to generating tests that distinguish all of a set of variants, or *mutants*, from some artifact. Mutation analysis is widely considered to be a powerful approach, so much so that other approaches to test generation are commonly evaluated on the basis of mutation score. One long-standing problem with using mutation score to evaluate other approaches is the presence of "redundant" mutants that do not contribute in any material way to the quality of a test set. For example, some mutants are killed by almost any test. Hence, eliminating such mutants from consideration does not affect which tests are chosen, but does result in a different mutation score. In other words, mutation scores can be inflated by redundant mutants, and this can make the mutation score harder to interpret.

The research area of reduced mutation has focused on achieving high quality test sets with fewer mutants [20], [23], [27], [29], [22], [21], [26], [6]. Selective mutation is a reduced mutation approach that limits the set of mutation operators to a subset of the available operators [20], [23], [27], [29], [22], [21], [26], [6]. Some approaches to reduced mutation limit the number of mutants considered to a random subset of mutants generated [19], [24]. Other approaches analyze relationships between specific mutants and remove redundant mutants [13], [11], [14]. Still others engineer higher-order-mutants (HOMs) that subsume one or more first-order mutants (FOMs)[1]. While these approaches clearly reduce the number of mutants under consideration, there is still a significant research gap. Specifically, there is no way to measure how close reduction techniques get to the goal of minimizing the number of mutants created while maintaining the quality of the corresponding test set.

This paper addresses exactly that research gap. We develop a theoretical framework for determining minimal sets of mutants. In particular, we show that, given a test set, a particular type of subsumption, called dynamic subsumption, enables efficient computation of minimal sets of mutants. We evaluate our approach against a benchmark set of programs and tests.

It is important to appreciate the role of the test set in our approach. Computing minimal mutant sets for *all* possible test sets is clearly undecidable; it is the fact that we limit attention to a particular test set that makes our approach computable. One way to think of our approach is that it approximates a limit: If one were able to run every possible test, then determining minimal sets of mutants with dynamic subsumption would, in fact, be both sound and complete. That is, any computed minimal mutant set would be, in fact, a "real" minimal mutant set. A corollary of this observation is that the more comprehensive the test set used in the analysis, the more accurate the resulting computation of minimal mutant sets.

Existing approaches to reduced mutation that use subsumption, such as the HOM approach, rely on detailed white-box analysis of the artifact under consideration. If a HOM is engineered to subsume several other mutants, then a test that kills that HOM will, of course, kill the subsumed mutants. However, equivalent mutants, that is, mutants that computes the same function as the original artifact, complicate the situation. If a HOM happens to be equivalent, or if the test engineer simply fails to find a test that kills the HOM, then the subsumption relationship does not help, since there may be tests that kill one or more of the subsumed mutants.

In contrast to the HOM approach to subsumption, our model takes a black-box perspective. We consider only the behavior of some fixed artifact in the context of a specific set of mutants and a specific set of test cases. In particular, our notion of subsumption is only assumed to hold with respect to the specific set of test cases under consideration, and it is possible

[1]In the development of Jia et al. [10], one mutant *subsumes* a second mutant if every test that kills the first mutant is guaranteed also to kill the second. The same notion of subsumption is used to reduce the number of logic mutants generated for DNF predicates [12].

that the subsumption relation would not hold for a different set of tests. Essentially, we replace the risk of equivalent mutants, which affects the HOM approach to subsumption, with the risk of incomplete test sets[2].

Our approach to modeling has two advantages. First, it frees us from the details of any particular programming language or artifact and lets us model the problem in a very general way. Second, it allows us to provide a precise definition for what constitutes a minimal set of mutants. While the definition itself is not constructive; the main result of the paper shows that a different notion of subsumption, called dynamic subsumption, completely characterizes mutant set minimality.

We used the Siemens suite [9], [7], to show the impact of our model. The Siemens suite includes a large number of tests. The evaluation shows that the size of the minimal mutant sets is much smaller than current approaches to reduced mutation achieve. The evaluation further shows that high mutation scores from different approaches to reduced mutation on a given test set are potentially misleading; once redundant mutants are removed, the scores are lower, sometimes much lower. In other words, there is substantial room for improvement in choosing mutants. Correspondingly, users of mutation scores should be cautious; large numbers of redundant mutants may make such scores misleading.

Again, it is important to appreciate the role of the chosen test set in the analysis of minimal mutants: generating a different test set might result in a different set of minimal mutants. That being said, most applications of mutation analysis end up with exactly one test set–namely the first set that kills enough mutants. From a practical perspective, an important question in this context is simply, "How many mutants (and which ones) are really needed to end up with this test set?" It is only in the context of the chosen test set that we determine which mutants are relevant.

The paper is structured as follows. Section II introduces a *score function* model for describing the relationship between mutants and test cases, and then develops the main theoretical results about minimal mutant sets. Section III applies the Proteum mutation tool to the Siemens suite of programs and computes minimal test and mutant sets for a specific initial set of tests. Section IV discusses related work. Section V puts the results into context and concludes the paper.

II. Model

This section presents a formal model for minimizing sets of mutants with respect to a test set. The model does not address any details of the artifact from which mutants are generated. Rather, it captures the "black-box" relationship of precisely which test cases kill which mutants.

A. Definitions

Let M be a finite set of mutants on some artifact P. P may be any testable artifact amenable to mutation analysis– a program, a specification, a design, etc. Let m, possibly subscripted, denote an element of M. Denote the cardinality of M as $|M|$.

Let T be a finite test set for P. Let t, possibly subscripted, denote an element of T. Denote the cardinality of T as $|T|$.

The boolean *score function* S specifies which mutants each test kills. Specifically, $S(i,j)$, $i = 1,..,|T|$, $j = 1,..,|M|$, is *true* iff test t_i kills mutant m_j. So S can be considered to be a binary matrix with $|T|$ rows and $|M|$ columns.

T is *mutation-adequate* for M if for each mutant $m_j \in M$, there is some test $t_i \in T$ such that $S(i,j)$ is true. The development in this paper does not require that the test set in the score function be mutation-adequate. From a practical perspective, our algorithms can be applied at any stage of testing. The richer the test set T is, the more mutants a minimal mutant set requires to capture the behavior exhibited by the artifact with respect to that test set.

In terms of the score function, if T is not mutation adequate, then there will be at least one mutant m in M that is *live*, which means that no test $t \in T$ kills m. A live mutant m may be equivalent, or T may rather be missing a suitable test that kills m. Each live mutant has a column in the score function without any *true* entries. Instead of insisting on mutation adequacy, we constrain our minimization procedures to maintain the effectiveness of mutation, evaluated by which mutants are killed by a given test set. Formally, a subset of T, denoted $T_{maintain}$, *maintains the mutation score* with respect to M (and T) if for every mutant m in M, if T kills m then $T_{maintain}$ kills m.

The score function captures all of the information about the mutants and tests of interest in this paper. If two tests kill precisely the same set of mutants, we consider the tests to be *indistinguished*, even though, in terms of the domain of P, the tests may have different input values. Similarly, if two mutants are killed by precisely the same set of tests, we consider the mutants to be *indistinguished* (thus far), even though the mutants may involve different syntactic changes to the underlying artifact. Indeed, indistinguished mutants may well cause different semantic changes to the underlying artifact, but these semantics are simply not captured by the test set T, and hence are not reflected in the score function S. Put another way, if T were augmented with additional tests, these additional tests might distinguish previously indistinguished mutants.

Below we show a score function for an example with five tests and four mutants: $T = \{t_1, t_2, t_3, t_4, t_5\}$ and $M = \{m_1, m_2, m_3, m_4\}$. T is mutation-adequate, all tests in T are distinguished, and all mutants in M are also distinguished. We use this score function as a running example through the rest of this section.

	m_1	m_2	m_3	m_4
t_1	t	t		t
t_2		t	t	
t_3	t		t	
t_4	t	t	t	t
t_5	t	t		

[2]Both the problem of determining whether a mutant is equivalent and the problem of finding a test case that kills a mutant are, of course, undecidable.

Observation 1: Score Function Boundedness.
The score function has, at most, $2^{|M|}$ distinguished rows. The reason is that each distinguished test kills some specific subset of M and there are exactly $2^{|M|}$ such subsets.

Observation 1 is important because it makes clear that although the domain of P may be large or unbounded, the number of distinguished rows in the score function is bounded. Put another way, the score function can identify every possible input in the domain of P with one of $2^{|M|}$ equivalence classes, depending on which mutants that input kills.

B. Minimal Sets of Tests

The key theoretical contribution of this paper is describing sensible minimizations to the score function. The motivation for minimizing tests is straightforward: if killing mutants is the goal, why run tests that do not increase the mutation score? Minimal test sets directly help the practicing test engineer.

The motivation for minimizing mutants has less to do with the practicing test engineer than with mutation testing researchers. The motivation for minimizing mutants is identifying the theoretical boundary of just how many mutants are required, and comparing existing mutation analysis methods against this boundary to see whether they can be improved, and, if so, potentially how much. While this theoretical lower bound may never be reached, it gives testing researchers an important tool. By knowing what's possible, we can objectively evaluate the effectiveness of our current engineering techniques to reduce the number of mutants. That is, this analysis gives us a firm bound against which to measure.

First we address test set minimization, a well-understood process that we include here for completeness.

Definition 1: Minimal test sets.
A test set $\hat{T}$ is *minimal* iff for any test $t_i \in \hat{T}$, $\hat{T} - \{t_i\}$ does not maintain the mutation score with respect to M and T.

Note that $\hat{T}$ depends on exactly which mutants are used.

There may be multiple minimal test sets, possibly of varying cardinalities, for any given test set T. Let $\bar{T}_M = \{\hat{T}_1, \hat{T}_2, ...\}$ denote the set of *all* possible minimal test sets with respect to mutant set M. Any element of $\bar{T}_M$ with the smallest cardinality is not only minimal, but also *minimum*.

In the example, $\bar{T}_M$ contains three minimal test sets:

$$\bar{T}_M = \{\{t_4\}, \{t_1,\ t_2\}, \{t_1,\ t_3\}\}$$

Note that a given test need not be part of any minimal test set. In the example, t_5 is not in any minimal test set. Of the three minimal test sets, one, namely $\{t_4\}$, has least cardinality (equal to 1), and hence is minimum.

Although finding a *minimum* test set is, like many optimization problems, computationally hard[3], generating a minimal test set is straightforward. Algorithm 1 generates a minimal test set with time complexity $|T| * |M|$. Note that Algorithm 1 selects tests for removal in an arbitrary order. If Algorithm 1 is applied to all possible permutations of tests in T, then it will generate all possible minimal test sets.

[3]Finding a *minimum* test set is an NP-complete problem. Finding a minimum test set is an instance of the Set Covering Problem (SCP) [16], where the universe is the set of mutants M, and the family of subsets of M is given by the rows of the score function, S.

Algorithm 1: Test set minimization

```
// Input:  Mutant set M and test set T
// Output: A minimal test set

  minSet = T
  for each t in minSet {
    // Note: t selected arbitrarily
    if (minSet-{t} maintains
        mutation score wrt M and T)
      minSet = minSet - {t}
  }
  return minSet
```

C. Minimal Sets of Mutants

We now turn to the problem of minimizing M, a topic that, to our knowledge, has not been previously addressed in the literature. We propose the following informal rationale for declaring mutants to be "unnecessary":

Testing P without considering unnecessary mutants should yield the exact same "results" as testing P with the full set of mutants M.

Building on this rationale, the *only* tests that a given set of mutants can "force" to be in a test set are those in some minimal test set. Hence, we define unnecessary mutants in terms of minimal test sets. We require that M generate precisely the same set of minimal test sets both *with* and *without* a *redundant* mutant. Recall that $\bar{T}_M$ denotes the set of minimal test sets of T with respect to some particular set of mutants M. The key part of the definition is the equality at the end:

Definition 2: Redundant mutants.
Let $M_j = M - \{m_j\}$ for some mutant $m_j \in M$. We say that m_j is *redundant* with respect to mutant set M and test set T iff $\bar{T}_M = \bar{T}_{M_j}$.

Again, note that this definition of redundant mutants is in the context of a particular test set T. Computing $\bar{T}$ for various mutant sets in the running example, first the full mutant set M, and then M with each mutant removed in turn, yields:

$$\bar{T}_M = \{\{t_4\}, \{t_1,\ t_2\}, \{t_1,\ t_3\}\}$$

$$\bar{T}_{M_1} = \{\{t_4\}, \{t_1,\ t_2\}, \{t_1,\ t_3\}\}$$

$$\bar{T}_{M_2} = \{\{t_4\}, \{t_1,\ t_2\}, \{t_1,\ t_3\}\}$$

$$\bar{T}_{M_3} = \{\{t_1\}, \{t_4\}\}$$

$$\bar{T}_{M_4} = \{\{t_4\}, \{t_1,\ t_2\}, \{t_1,\ t_3\}, \{t_2,\ t_5\}, \{t_3,\ t_5\}\}$$

Note that $\bar{T}_M$, $\bar{T}_{M_1}$, and $\bar{T}_{M_2}$ are identical. This means that both m_1 and m_2 are redundant with respect to M. If a pair of redundant mutants, m_1 and m_2, are indistinguished, it is possible that we might only be able to remove one of the mutants safely. Consider the case where mutant m_1 is *not* redundant with respect to M. If some additional mutant m_2 is indistinguished from m_1 and we form $M \cup \{m_2\}$ then only one of m_1 or m_2 can be removed from $M \cup \{m_2\}$ without altering the associated minimal test sets. Algorithm 2, based on the dynamic subsumption relation developed later in the paper, clarifies precisely which mutants can be removed safely. In particular, only one mutant from each set of indistinguished mutants is (possibly) needed; beyond that, all redundant mutants can be safely removed.

Since m_1 and m_2 are distinguished and redundant, both can safely be removed from M without altering the resulting minimal test sets, thereby yielding a minimal mutant set of $\{m_3,\ m_4\}$. In this example, there is only one minimal mutant set.

When a redundant mutant m is removed, it is possible that tests that were distinguished with respect to M are no longer distinguished with respect to $M - \{m\}$. From the practical perspective, this means that the test engineer has a choice about which test to use when constructing a minimal test set. In the example above, for the minimal set of mutants $\{m_3,\ m_4\}$, tests t_2 and t_3 are indistinguished.

Definition 3: Minimal mutant sets.
Mutant set M is *minimal* if it contains no redundant mutants.

We show the score function after minimization for our running example.

	m_3	m_4
t_1		t
t_2	t	
t_3	t	
t_4	t	t
t_5		

Although this example has only one minimal mutant set, there are potentially many minimal mutant sets.

Because there are a large number of minimal test sets for any given set of mutants, the definition of minimal *mutant* sets, which relies on comparing the associated minimal *test* sets, does not lend itself directly to an efficient algorithm. Hence, the next challenge is identify a way to compute efficiently which mutants are redundant.

D. Efficiently Computing Minimal Sets of Mutants

We turn to the notion of subsumption. Traditionally, one mutant is defined to subsume another for *all* possible executions based on internal reasoning about the artifact being mutated or the mutation operator in question. For example, mutants that negate a term in a Disjunctive Normal Form (DNF) predicate subsume mutants that negate the entire DNF formula. A variety of these relationships are shown in the fault hierarchy of Lau and Yu [18]. The proof of subsumption relies on properties of predicates expressed in DNF.

In this paper, we define a different notion of subsumption strictly in terms of black-box behavior of mutants M on a test set T as captured by the score function. Crucially, this new notion of subsumption does not necessary hold for all possible executions. Rather it is only guaranteed to hold for executions in the set T. Specifically, consider two mutants m_x and m_y where every test in T that kills m_x also kills m_y.

Definition 4: Dynamic subsumption.
If mutant x is not *live* and $S(i,x) \rightarrow S(i,y), i = 1..|T|$, we say that m_x *dynamically subsumes* m_y with respect to T.

Dynamic subsumption differs from the notion used in white-box mutation analysis in a crucial respect: Not only are tests that kill x also required to kill y, but T also has to have at least one test that kills x. In other words, dynamic subsumption disallows "vacuous" subsumption, which would be possible if we did not have a test that killed x. For example, it is possible, through white-box analysis, to design a HOM m that subsumes several other mutants, but it is (usually) not be possible to tell if m is equivalent. Since we work in the black-box context of a specific set of test cases T, the score function can distinguish among live mutants.

In any set M that contains both m_x and m_y, if m_x dynamically subsumes m_y, then m_y is redundant, and hence may be safely discarded, a fact we prove in the first part of Theorem 1 below.

Perhaps surprisingly, dynamic subsumption completely captures the notion of redundant mutants. That is, the *only* way in which a mutant becomes redundant is for it to be dynamically subsumed by some other mutant in M, a fact we prove in the second part of Theorem 1 below. The main result of this paper formalizes these two properties:

Theorem 1: Dynamic subsumption and minimal test sets. Mutant set M is minimal with respect to test set T iff there does not exist a distinct pair $m_x, m_y \in M$ such that m_x dynamically subsumes m_y.

Proof:
Step 1: If M is minimal, then there does not exist a distinct pair $m_x, m_y \in M$ such that m_x dynamically subsumes m_y.

We proceed by contradiction. Suppose there exist m_x and m_y such that m_x dynamically subsumes m_y. Consider the process of producing a minimal test set for either M or $M - \{m_y\}$ by applying Algorithm 1. If Algorithm 1 considers tests in the same order in each case, and the **if** test in Algorithm 1 always comes to the same conclusion, then Algorithm 1 produces the same minimal test set in for both M and $M - \{m_y\}$. Since this would happen for all possible orders of choosing tests, it means that $\bar{T}_M = \bar{T}_{M_y}$. But this would mean that M is not minimal–a contradiction.

Hence, the proof comes down to considering whether, at some stage of Algorithm 1, the **if** test evaluates differently for

some test t with respect to M and $M - \{m_y\}$. We proceed by case analysis:

- Case 1: t can be removed during the minimization with respect to M, but not the corresponding minimization with respect to $M - \{m_y\}$. Dynamic subsumption has nothing to do with this case. Rather, if a test is not needed for a particular set of mutants, it is clearly not needed for any subset either. Hence, Case 1 is impossible.
- Case 2: t can be removed during the minimization with respect to $M - \{m_y\}$, but not the corresponding minimization with respect to M. In algorithm 1, the variable $minSet$ must have some test that kills m_x, and thus, by dynamic subsumption, m_y as well. Hence, m_y cannot be the reason that t must be kept for set M. In other words, the **if** decision must be the same for both M and $M - \{m_y\}$. Hence, Case 2 is impossible.

Step 2: If there does not exist a distinct pair $m_x, m_y \in M$ such that m_x dynamically subsumes m_y, then M is minimal.

To show this part, for each m_x in M, we incrementally construct a test set T_x around m_x. We show that this test set is minimal with respect to $M - \{m_x\}$, but does not maintain the mutation score with respect to M. Hence m_x is not redundant, and cannot be removed from the mutant set. Since we show this for each mutant in the set, the set M must be minimal.

To construct T_x, consider each other mutant m_y in M. There must be some test in T that kills m_y but does not kill m_x, or else m_y would dynamically subsume m_x. Include this test in T_x. Note that T_x kills every mutant except for m_x. Choose some minimal set $\hat{T}_x$ subseteq T_x using Algorithm 1. Note that $\hat{T}_x$ is minimal with respect to $M - \{m_x\}$ but does not maintain the mutation score with respect to M. Hence no $m_x \in M$ is redundant, and so M is minimal with respect to T.

QED

Algorithm 2 uses Theorem 1 to efficiently compute minimal mutant sets. First, live mutants are removed. Next, indistinguished mutants are removed. Finally, dynamically subsumed mutants are removed.

Algorithm 2: Mutant set minimization

```
// Input:  Mutant set M; Score function S
// Output: A minimal mutant set

  remove live mutants from S
  remove duplicate columns from S
  minSet = remaining columns in S

  subsumed = dynamically subsumed
     mutants in minSet

  return (minSet - subsumed);
```

We now apply Algorithm 2 to our running example. There are no live mutants or duplicate columns in the score function, so the variable `minSet` in the algorithm starts with all four mutants, m_1, m_2, m_3, and m_4. Mutants m_1 and m_2 are dynamically subsumed by mutant m_4. Removing these two mutants from `minSet` yields exactly the same minimal set of mutants, namely $\{m_3, m_4\}$, identified in the previous section by considering minimal test sets.

E. Some Properties of Minimal Mutant Sets

Since a representative from each set of indistinguished mutants is chosen arbitrarily in the first step of Algorithm 2, where duplicate columns in S are removed, minimal mutant sets need not contain exactly the same mutants. However, somewhat surprisingly, minimal mutant sets *do* all have the same cardinality.

Theorem 2: Mutant set cardinality
Every minimal mutant set has the same cardinality.

Proof.
The key observation is that dynamic subsumption is just logical implication, and hence is transitive. This means that if one removes a dynamically subsumed mutant from a set of mutants, that removal does not affect which of the remaining mutants are dynamically subsumed. Hence, dynamically subsumed mutants may be removed in an arbitrary order, which is why the second part of Algorithm 2 is structured the way it is, as opposed to being an explicit loop that iteratively checks for dynamic subsumption. Put another way, a minimal mutant set is simply a mutant set with indistinguished mutants collapsed to single representatives and the remaining dynamically subsumed mutants removed–operations that always produce a result of the same cardinality.
QED

The appeal of Theorem 2 is that in states that, for a given test set T, a specific number of mutants (selected from M) are both necessary and sufficient to generate all possible minimal test sets (selected from T).

Observation 2. Minimal mutant sets for minimal test sets. If T happens to be a minimal test set, then every corresponding minimal set of mutants has exactly $|T|$ elements. The resulting score function is square. Every row has exactly one `true` value, and every column has exactly one `true` value.

In particular, if T has exactly one element, so does every minimal M. This extreme example illustrates the idea that M simply generates tests with respect to some underlying set of tests T. If that test set is already minimal, all M can do is generate exactly that set. If T is not minimal, then M can potentially generate more than one minimal test set.

III. Assessment

We now use Algorithm 2, to compute minimal sets of mutants with respect to a given test set. This section applies Algorithm 2 to a standard benchmark for testing research, namely the Siemens suite [9], [7], which consists of seven C programs and associated test sets. We have two goals:

1) Examine the relationship between total mutants generated by traditional approaches and minimal mutant sets.
2) Highlight the effect on mutation score of measuring against traditional mutant sets vs. minimal mutant sets.

This section is not a formal experiment. Hence, we do not enumerate research questions, results, threats, etc. Rather, we simply apply our definitions and report facts about test set minimization, mutant set minimization, and reduced mutation.

For each program in the Siemens suite, the Proteum tool [4] was used to generate mutants and the score function was collected for 512 tests randomly taken from the Siemens suite[4].

A. Minimal Test Sets

Table I presents characteristics about the test sets used in the study. The column labeled **Program** lists the programs. The column labeled **Total Tests** shows how many tests are available in the Siemens suite for each program. The column labeled **Used Tests** shows how many tests were used in this evaluation–512 for each program. The column labeled **Distinguished Tests** shows how many of the 512 tests are distinguished. Recall that two tests are indistinguished if they kill exactly the same subset of mutants. The table shows that for each of the seven programs, very few tests were indistinguished.

TABLE I
TEST SET CHARACTERISTICS

Program	Total Tests	Used Tests	Distinguish-able Tests	Minimal Tests	Union : Intersection
print_tokens	4073	512	499	12.4	181 : 3
print_tokens2	4058	512	479	12.1	160 : 1
replace	5542	512	510	44.4	218 : 19
schedule	2650	512	482	14.5	158 : 2
schedule2	1052	512	479	17.1	131 : 4
tcas	1608	512	428	41.4	207 : 10
totinfo	4073	512	452	13.3	134 : 4

The column labeled **Minimal Tests** shows how many tests are in a minimal test set produced by Algorithm 1 applied to the 512 selected tests. Since there are many possible minimal test sets, this final number is the average of 100 minimal test sets generated by choosing tests to remove at random in the **if** statement of Algorithm 1. Note that the minimal test sets are relatively small compared to the number of distinguished tests. The column labeled **Union: Intersection** gives the number of tests (taken from 512) that appeared in the union and intersection of the 100 randomly selected minimal test sets. It is clear that even though minimal test sets are relatively small, many different tests can be used to construct a minimal test set. Conversely, there are very few tests that appeared in all 100 trials. This suggests that there are few, if any, "necessary" tests in the set of 512.

[4]The number 512 is an artifact of the Proteum tool, which processes tests in batches of size up to 512. Since minimal mutants are calculated with respect to a specific test set, as opposed to all possible inputs, it is sensible to carry out the analysis with *any* test set. For the same reason, we don't sample different test sets; instead we model a testing process where a particular test set is chosen, and then redundant mutants are identified.

B. Minimal Mutant Sets

Table II captures relevant facts about the mutants used in the study with respect to the test sets (of size 512) described above. Again, the column labeled **Program** lists the programs. The column labeled **Total Mutants** reports the total number of mutants. The column labeled **Live Mutants** reports live mutants. Specifically, for each entry of the form X:Y, X is the number of mutants live after execution of the complete Siemens test suite, and Y is the number of mutants live after execution of the chosen 512 tests. The column labeled **Difference (Ratio)** reports the difference between these two values in absolute form and also their ratio. By either measure, relatively few mutants are killed by the full suite, but not by the set of 512 tests. In terms of mutation score (not shown in the table), the 512-sized test sets exceeds 99% for all of the programs.

TABLE II
MUTANT CHARACTERISTICS

Program	Total Mutants	Live Mutants	Difference (Ratio)	Distinguished : Minimal
print_tokens	4336	597 : 625	28 (0.96)	437 : 28
print_tokens2	4746	692 : 704	12 (0.98)	439 : 30
replace	11101	2195 : 2318	77 (0.95)	2309 : 58
schedule	2109	267 : 271	4 (0.99)	520 : 42
schedule2	2627	488 : 495	7 (0.99)	461 : 46
tcas	2384	418 : 427	9 (0.98)	596 : 61
totinfo	6698	877 : 877	0 (1.00)	835 : 19

The first entry in the column labeled **Distinguished : Minimal** reports the number of distinguished mutants. Recall that two mutants are indistinguished if they are killed by exactly the same subset of tests. The number of mutants that are distinguished is much smaller than the total number of mutants. This suggests that many mutants are not only redundant, they also exhibit *identical* behavior with respect to the test set. Further, the fraction of mutants that are distinguished (17%) is much smaller than the fraction of tests that are distinguished (93%). In terms of distinguished entries, the score function exhibits different behavior when viewed from the row perspective than it does when viewed from the column perspective.

The second entry in the column labeled **Distinguished: Minimal** reports the number of minimal mutants in a minimal mutant set[5].

Not only is the number of minimal mutants much smaller than the total number of mutants (on average, only 1.2% of mutants are in a minimal set), it is also much smaller than the total number of distinguished mutants (on average, only 6.6% of distinguished mutants are in a minimal set). In other words, the dynamic subsumption relation eliminates a large fraction of the distinguished mutants.

For example, in the case of `totinfo` (last row in the tables), Proteum generated 6698 mutants, $(6698-877) = 5811$ of which were killed by both the full Siemens test suite and

[5]As Theorem 2 showed, there may be many minimal mutant sets for a given set T, but all are of the same size. Hence, there is no reason to run multiple trials and average the results, as was the case for minimal test sets.

also the set of 512 tests. Of these 6698 mutants, 835 were distinguished. Of the 5811 killed mutants only 19 mutants, or 0.3%, are needed for a minimal test set. Of the 834 distinguished killed mutants[6], only 19 mutants, or 2.3%, are needed for a minimal test set. By any measure, the number of generated mutants far exceeds the number necessary.

The two tables given so far give the dimensions of the score function for each program. For example, `print_tokens` has a score function with 512 rows, of which 499 are distinguished, and 4336 columns, of which 437 are distinguished.

C. Reduced and Selective Mutation

We turn next to analyzing reduced mutation, the idea that using fewer mutants is nearly as effective as the complete set of mutants. We consider five reduced mutation approaches, one random and four selective. The notion of selective mutation was first suggested by Mathur [20], developed by Offutt et al. [23], and studied extensively thereafter for both FORTRAN [22], [21] and C [1].

We use the Proteum mutation tool suite. We use generic labels for the approaches, and provide the Proteum names in parentheses.

1) STMT: Statement Deletion (Proteum SSDL)
2) ROR: Relational Operator Replacement (Proteum ORRN)
3) CON: Replace Scalars with Constants (Proteum CCSR)
4) 5RND: 5% random selection of all mutants
5) SELECT: An approximation of selective mutation (Proteum: OOAN+OLLN+ORRN+OLNG)

STMT has been studied as a stand-alone, cost-effective approach to mutation [6], [3], [26]. While ROR and CON have not been studied specifically as proposals for stand-alone operators, they are plausible candidates. A random percentage of all mutants has been widely used to reduce the number of mutants that need to be considered [19], [24]. We chose 5% of random mutants because the number of mutants selected approximated the mutants created by the SELECT strategy.

The SELECT strategy approximates the original selective mutation definition from the Mothra system [22]. The Mothra approach to selective mutation had five operators:

1) ABS: Absolute Value
2) AOR: Arithmetic Operator Replacement (Proteum: OAAN)
3) LCR: Logical Connector Replacement (AND and OR) (Proteum: OLLN)
4) ROR: Relational Operator Replacement (Proteum: ORRN, but this does not include using the constants true and false)
5) UOI: Unary Operator Insertion (Proteum, logic only: OLNG)

Of these five operators, Proteum has corresponding match for two and a partial match for two more. These matches are indicated in parentheses in the list above.

[6]Of the 835 distinguished mutants, 834 are killed by the test set, and one is live.

TABLE III
REDUCED MUTATION SCORES: TRADITIONAL VS. MINIMAL MUTANT SETS

Program	STMT	ROR	CON	5RND	SELECT
print_tokens	99 : 78	98 : 77	99 : 78	99 : 82	99 : 81
print_tokens2	99 : 47	99 : 56	99 : 49	99 : 48	99 : 57
replace	97 : 31	97 : 38	99 : 57	99 : 56	98 : 48
schedule	97 : 68	94 : 53	98 : 65	98 : 67	97 : 65
schedule2	97 : 72	92 : 56	98 : 77	98 : 72	97 : 72
tcas	88 : 27	90 : 38	88 : 33	94 : 45	93 : 44
totinfo	97 : 38	99 : 59	99 : 39	99 : 54	99 : 60

Table III shows the results of analyzing these five approaches to reduced mutation in the context of the chosen 512 test cases. The rows in the tables are again the programs from the Siemens suite. Each column of data represents one of the five approaches to reduced mutation. Table entries are designed to show the difference between traditional mutation scores and a mutation score measured against the minimal mutant set.

Each entry in the table is of the form X:Y. X is the mutation score, as a percentage, obtained by a test set adequate to the corresponding reduction strategy, against all mutants that are killed by the chosen 512 test cases. Y is the mutation score, again as a percentage, obtained by the same test set against a *minimal* set of mutants, again in the context of the 512 test cases.

The noteworthy aspect of this table is that although the traditional mutation scores generally seem excellent, the mutation scores against the minimal mutant set are not nearly as good, ranging from a low 27% to a high of 82%. One lesson from this evaluation, consistent with other recent studies [11], is that a mutation score measured over a large number of redundant mutants is inflated–possibly to the point of being meaningless.

Figure 1 shows the data from the STMT column of Table III in chart form. For each program, the left bar shows the mutation score with respect to all mutants, and the right bar shows the mutation score with respect to a minimal set of mutants. The basic observation from the chart is that the redundancy in the full set of mutants makes it difficult to interpret mutations scores computed using the full set of mutants.

To take a specific case, consider `tcas`. The STMT approach appears to achieve a respectable score of 88% mutation coverage. However, in terms of a minimal set of mutants, statement deletion mutation only kills about one in four.

Next, we present some data about tests in the minimal test sets. Table IV continues the analysis of reduced mutation. This time the table shows how many mutants killed by the 512 test cases are generated by the technique, along with the corresponding test size. Each value should be compared to the reference value in the column labeled **Minimal**, which (again) shows the number of mutants in the minimal set, along with the corresponding test size. The average number of tests required for the minimal mutant set is often larger than the number of tests required by a reduced approach. The reason is that the test sets for the reduced approaches are missing key tests. Specifically, they are missing tests that kill mutants in

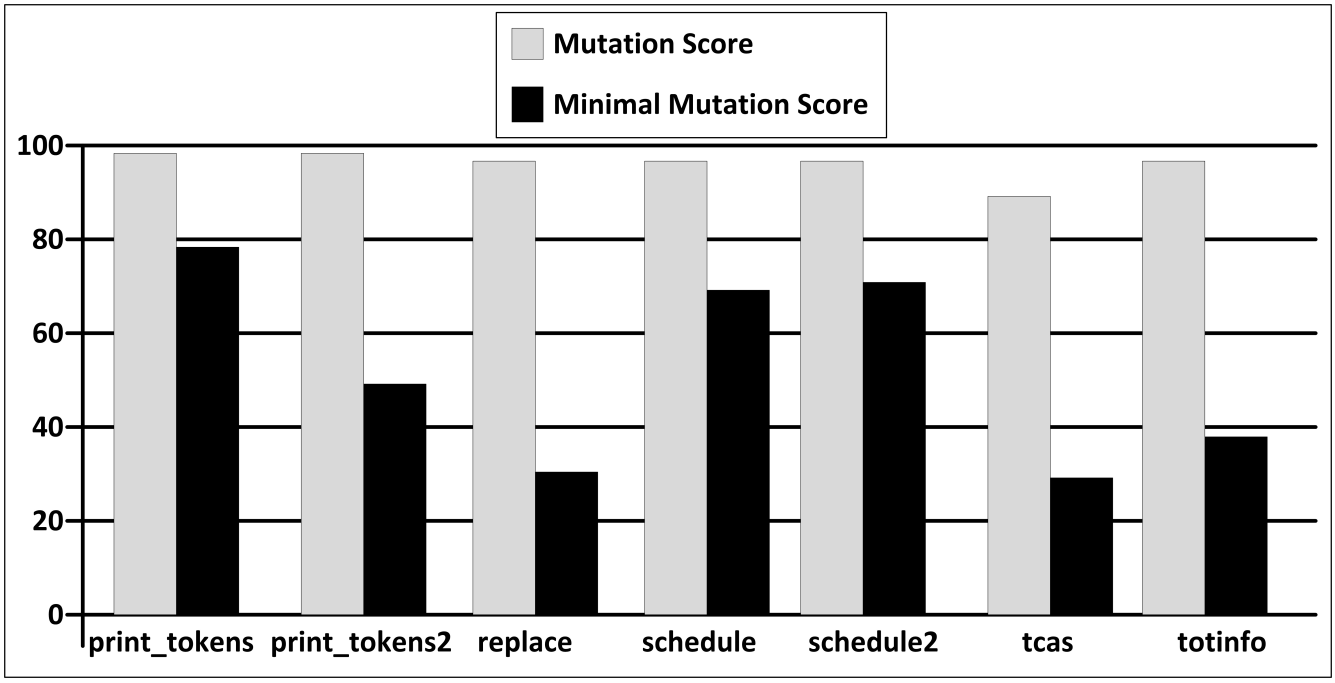

Fig. 1. Mutation Score vs. Minimal Mutation Score for STMT

a minimal mutant set. Put another way, the reduced mutation approaches set omits key mutants; mutants that could lead to very good tests.

TABLE IV
REDUCED MUTATION: MUTANTS GENERATED VS. TESTS

Program	STMT	ROR	CON	5RND	SELECT	Minimal
print_tokens	196:11	98: 9	358:10	190:11	138:10	28:12.4
print_tokens2	203: 5	192: 8	445: 8	198: 9	244: 9	30:12.1
replace	219:23	264:27	1053:44	443:39	499:35	58:44.4
schedule	127:10	49: 7	78:13	95:12	84:10	42:14.5
schedule2	117: 9	75: 6	119:13	110:13	121:12	46:17.1
tcas	42:12	45:14	66:14	99:24	113:18	61:41.4
totinfo	110: 6	167:13	469:12	294:15	332:15	19:13.3

For example, consider `tcas` again. The STMT approach generated 42 mutants that were killed by the 512 test cases, which is in the neighborhood of the 61 mutants in the minimal mutant set. Unfortunately, the choice of these 42 mutants is far from optimal. A test set that kills these 42 mutants has only 12 tests, compared to the average of 41.4 tests needed to kill the minimal set of mutants. In other words, STMT is generating about 1/3 the number of required tests, a fact that was reflected in Table III in the poor STMT mutation score of 27% against the minimal mutant set.

What is striking about Table IV is that in many cases, significantly more mutants are generated than in the minimal mutant set, but, in terms of achieving the best coverage, they are not the optimal mutants, and significantly fewer tests than needed for full coverage are generated. This table highlights a research gap: it is clear that a small number of mutants can force the generation of a very high quality test set, but it is not known how to choose these mutants. The best techniques in practice today, selective mutation and SDL-mutation, are a very long way from generating mutant sets that both include the desirable mutants and exclude unnecessary (and, of course, equivalent) mutants. A complete solution is, of course, theoretically impossible. But even modest partial solutions have room to improve matters significantly. A key point is that minimal mutant sets are not a *replacement* for strategies such as reduced mutation–it is still necessary to execute each mutant to create the set of minimal mutants. Rather, minimal mutant sets give a bound against which to evaluate techniques such as reduced mutation.

IV. RELATED WORK

The subsumption relation has been studied in a variety of contexts for many years. Chusho observed that measuring branch coverage over all branches in a program led to an overestimation of quality, and defined the notion of *essential branches* as a way of removing redundant branches from coverage measures [2]. In this paper, dynamically subsumed mutants play exactly the same role as non-essential branches do in the Chusho analysis. The difference is that this paper is "black-box," whereas the Chusho paper considers the actual structure of the code. Hence, the Chusho results hold for all test sets; our results are specific to a particular test set T.

Harman and Jia defined the notion of subsuming Higher Order Mutants (HOMs) [10]. The idea was that a single HOM could stand in for several mutants. Langdon et al. applied subsuming HOMs to relational operators [17]. Lau and Yu identified subsumption relations between faults in Disjunctive Normal Form (DNF) predicates and presented this subsumption relation in a fault hierarchy [18]. Kaminski et al.[12] extended this work by defining special HOMs, which,

though relatively few in number, still subsumed all of the Lau and Yu hierarchy. In terms of the relationship to this work, subsuming HOMs are defined by internal analysis of the artifact under consideration; in contrast, we observe dynamic subsumption with respect to a specific test set.

Kaminski et al. [15] observed that the four of the seven mutants generated by Mothra's Relational Operator Replacement (ROR) were always subsumed by other mutants. The special treatment here was that the subsumed ROR operators depended on which operator appeared in the original code. Just et al. raised exactly the point that raw mutation scores led to overly optimistic evaluations of quality and defined subsuming mutants in the context of the Conditional Operator Replacement (COR) operator [11]. Again, in terms of the relationship to this work, eliminating mutants in these papers is done at the operator level before test cases are generated. Our approach to subsumption is based on the artifact's behavior after a specific test set is chosen.

Given that test set minimization is NP-complete, various researchers have developed test set minimization heuristics. Harrold et al. gave an authoritative treatment [8]. Studies have investigated whether minimizing test sets with respect to various coverage criteria has an effect on fault detection of the remaining tests. A positive result [28] reported on a case study in which minimizing test sets with respect to the dataflow "all-uses" coverage did not significantly reduce fault detection ability. A subsequent study [25] on the Siemens suite came to a contradictory conclusion: minimizing test sets with respect to edge (or branch) coverage severely compromized fault detection. The relevance of test set minimization to mutant minimization is that minimal mutant sets are defined in terms of minimal test sets; hence fault-detection bias introduced by minimal test sets potentially affects minimal mutant sets as well. Further research is needed to evaluate this issue.

V. Discussion and Conclusion

This paper has presented a way to identify precisely how many mutants are needed in the context of a given test set. The size of this set is much smaller than delivered by current best-practice approaches to mutation. We conclude that there is considerable scope for new approaches to mutation analysis that consider only relatively few mutants while at the same time thoroughly testing the underlying artifact.

Mutation score is widely used in the literature to evaluate the quality of an approach to generating test cases. As noted in Section IV, this approach has caused some disquiet in the research community due to the presence of redundant mutants. The results of this paper suggest a different methodology for evaluating testing approaches. Rather than evaluating a given approach against all mutants generated by some set of operators, we propose that, in addition, the approach should be evaluated against a minimal set of mutants. Any approach as strong as the chosen mutation operators will achieve 100% in either case. Weaker approaches can still be compared against criteria such as random selection, but using a minimal set of mutants for comparison removes the problem of redundant mutants from the evaluation.

The minimization approach developed in this paper focused on mutation analysis specifically to address the problem of redundant mutants. However, since the approach uses only the black-box score function, the model can also be applied to test requirements from any other coverage criterion, e.g., statement coverage, branch coverage, dataflow coverage, and so on.

The eventual goal of this line of research is to make mutation testing cost-effective enough to use in practice. The dynamic subsumption approach to minimizing the number of mutants demonstrates that it is, indeed, possible to reduce the number of mutants needed to a very small number. We hope the theoretical structure presented in this paper will lead to practical applications to dramatically reduce the number of mutants generated by actual mutation systems.

Acknowledgment

Prof. Marcio Delamaro's research is supported by FAPESP (Fundação de Amparo a Pesquisa do Estado de São Paulo), process number 2012/16950-5.

References

[1] Ellen Francine Barbosa, Jose Carlos Maldonado, and Auri Marcelo Rizzo Vincenzi. Toward the determination of sufficient mutant operators for C. *Software Testing, Verification, and Reliability, Wiley*, 11:113–136, 2001.

[2] T. Chusho. Test data selection and quality estimation based on the concept of essential branches for path testing. *IEEE Transactions on Software Engineering*, 13(5), May 1987.

[3] Marcio E. Delamaro, Lin Deng, Vinicius H. S. Durelli, Nan Li, and Jeff Offutt. Experimental evaluation of SDL and one-op mutation for c. In *7th IEEE International Conference on Software Testing, Verification and Validation (ICST 2014)*, Cleveland, Ohio, March 2014. To appear.

[4] Márcio E. Delamaro and José C. Maldonado. Proteum-A tool for the assessment of test adequacy for C programs. In *Proceedings of the Conference on Performability in Computing Systems (PCS 96)*, pages 79–95, New Brunswick, NJ, July 1996.

[5] Richard A. DeMillo, Richard J. Lipton, and Fred G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–41, April 1978.

[6] Lin Deng, Jeff Offutt, and Nan Li. Empirical evaluation of the statement deletion mutation operator. In *6th IEEE International Conference on Software Testing, Verification and Validation (ICST 2013)*, Luxembourg, March 2013.

[7] Hyunsook Do, Sebastian Elbaum, and Gregg Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 10(4):405–435, October 2005.

[8] Mary Jean Harrold, R. Gupta, and Mary Lou Soffa. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology*, 2(3):270–285, July 1993.

[9] Marlie Hutchins, H. Foster, Thomas Goradia, and Thomas Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of the Sixteenth International Conference on Software Engineering*, pages 191–200, Sorrento, Italy, May 1994. IEEE Computer Society Press.

[10] Yue Jia and Mark Harman. Constructing subtle faults using higher order mutation testing. In *2008 Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 249–258, Beijing, September 2008.

[11] Ren Just, Gregory M. Kapfhammer, and Franz Schweiggert. Do redundant mutants affect the effectiveness and efficiency of mutation analysis? In *Eighth Workshop on Mutation Analysis (IEEE Mutation 2012)*, Montreal, Canada, April 2012.

[12] Garrett Kaminski and Paul Ammann. Using a fault hierarchy to improve the efficiency of DNF logic mutation testing. In *2nd IEEE International Conference on Software Testing, Verification and Validation (ICST 2009)*, pages 386–395, Denver, CO, April 2009.

[13] Garrett Kaminski, Paul Ammann, and Jeff Offutt. Better predicate testing. In *Sixth Workshop on Automation of Software Test (AST 2011)*, pages 57–63, Honolulu HI, USA, May 2011.

[14] Garrett Kaminski, Paul Ammann, and Jeff Offutt. Improving logic-based testing. *Journal of Systems and Software, Elsevier*, 86:2002–2012, August 2013.

[15] Garrett Kaminski, Paul Ammann, and Jeff Offutt. Improving logic-based testing. *Journal of Systems and Software, Elsevier*, 86(8):2002–2012, 2013.

[16] Richard M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, New York, 1972.

[17] William B. Langdon, Mark Harman, and Yue Jia. Efficient multi objective higher order mutation testing with genetic programming. *Journal of Systems and Software, Elsevier*, 83(12):2416–2430, 2010.

[18] M. F. Lau and Y. T. Yu. An extended fault class hierarchy for specification-based testing. *ACM Transactions on Software Engineering Methodology*, 14(3):247–276, July 2005.

[19] Adidya Mathur and W. Eric Wong. Evaluation of the cost of alternative mutation testing strategies. In *Proceedings of the Seventh Brazilian Symposium on Software Engineering*, Rio de Janiero, Brazil, September 1993.

[20] Aditya Mathur. Performance, effectiveness, and reliability issues in software testing. In *Proceedings of the Fifteenth Annual International Computer Software and Applications Conference*, pages 604–605, Tokyo, Japan, September 1991.

[21] Elfurjani S. Mresa and Leonardo Bottaci. Efficiency of mutation operators and selective mutation strategies: An empirical study. *Software Testing, Verification, and Reliability, Wiley*, 9(4):205–232, 1999. December.

[22] Jeff Offutt, Ammei Lee, Gregg Rothermel, Roland Untch, and Christian Zapf. An experimental determination of sufficient mutation operators. *ACM Transactions on Software Engineering Methodology*, 5(2):99–118, April 1996.

[23] Jeff Offutt, Gregg Rothermel, and Christian Zapf. An experimental evaluation of selective mutation. In *Proceedings of the Fifteenth International Conference on Software Engineering*, pages 100–107, Baltimore, MD, May 1993. IEEE Computer Society Press.

[24] Mike Papadikis and Nicos Malevris. An empirical evaluation of the first and second order mutation testing strategies. In *Sixth Workshop on Mutation Analysis (IEEE Mutation 2010)*, pages 90–99, Paris, France, April 2010.

[25] Gregg Rothermel, Mary Jean Harrold, Jeffery Ostrin, and Christie Hong. An empirical study of the effects of test set minimization on fault detection capabilities of test suites. In *14th IEEE International Conference on Software Maintenance (ICSM 1998)*, pages 34–43, Bethesda, Maryland, 1998. IEEE.

[26] Roland Untch. On reduced neighborhood mutation analysis using a single mutagenic operator. In *ACM Southeast Regional Conference*, pages 19–21, Clemson SC, March 2009.

[27] W. Eric Wong, M. E. Delamaro, J. C. Maldonado, and Aditya P. Mathur. Constrained mutation in C programs. In *Proceedings of the 8th Brazilian Symposium on Software Engineering*, pages 439–452, Curitiba, Brazil, October 1994.

[28] W. Eric Wong, Joseph R. Horgan, Saul London, and Aditya P. Mathur. Effect of test set minimization on fault detection effectiveness. In *Proceedings of the 17th International Conference on Software Engineering (ICSE 1995)*, pages 41–50, Seattle, Washington, 1995. ACM.

[29] W. Eric Wong and Aditya P. Mathur. Reducing the cost of mutation testing: An empirical study. *Journal of Systems and Software, Elsevier*, 31(3):185–196, December 1995.