Dual Mutation: The "Save the Mutants" Approach

Márcio Eduardo Delamaro
Centro Universitário Eurípides de Marília (UNIVEM)
Av. Hygino Muzzi Fillho, 529
Marília - SP, Brazil
17525-901
delamaro@fundanet.br

Abstract

Mutation testing is a fault based testing criterion that has been widely used and studied. It has been shown an effective fault revealing criterion and its characteristics allow its use in a large range of entities like regular programs and behavioral specifications of real time systems. To evaluate the adequacy of a test set **T** in the test of a program **P**, mutation testing uses a set of alternative implementations of **P** called mutants. The adequacy of **T** is assessed by its ability on demonstrate that the mutants produce different results of **P**. In this paper we present the idea of Dual Mutation Testing (DMT). DMT uses the same mutants as mutation testing but requires test cases that show that the mutants can produce the same results of **P**. In a case study we apply Dual Mutation Testing and compare it to traditional mutation testing.
Keywords: software testing, mutation testing, dual mutation testing.

1 Introduction

Testing is a crucial activity in the software lifecycle. It is expensive and time consuming. For this reason much effort has been spent on developing techniques and tools to support the testing activity. An important result of those researches is the definition of techniques and criteria to drive the generation of test sets that can suitably exercise a program.

Mutation testing is a fault based test technique. It uses a set of rules called **mutant operators** to create programs slightly different from the program under test. These programs are called **mutants**. The goal of mutation testing is the generation of a test set that distinguishes the behavior of the mutants from the original program. The ratio between the number of distinguished mutants (also called **dead** or **killed** mutants) and the total number of mutants, measures the adequacy of the test set.

According to the **coupling effect** hypothesis [12], test cases that distinguish simple faults injected in the original program to create the mutants should also be able to reveal faults that can be obtained as a composition of simple faults. Thus, mutant operators can be seen as representative of common faults usually found in software.

In several empirical studies, mutation-based test adequacy criteria were found to be effective for the selection and evaluation of test cases [9, 18]. However, the cost of using such criteria, measured in terms of the number of mutants to be executed, is a barrier to their applicability. Some approaches can be taken to reduce the cost of mutation testing, for example, by applying constrained mutation criteria [14, 15, 19], without any significant loss in the effectiveness to reveal faults.

Besides the cost to create and execute mutants, there is also the cost to analyze mutants and decide their equivalence. In this case only a few studies have been conducted aiming at reducing such cost. Offutt and Craft [11] conduct a study to identify ways to automatically detect equivalent mutants. They proposed six techniques based on strategies of code optimization and data-flow analysis. Using those techniques in an experiment with FORTRAN programs, the authors concluded that a significative percentage of the equivalent mutants could be automatically detected. In some cases this number reaches 50% of the equivalent mutants. Jorge et al. [8] have proposed a technique to select mutant operators based not only on the relative strength of each operator but also on its tendency to create equivalent mutants.

In this paper we present an alternative way to use mutant to select test cases. This criterion, named **dual mutation testing** (DMT) selects test cases that makes mutants produce the same results of the original program. A case study has been conducted in order to evaluate test cases selected by DMT against traditional mutation criterion.

In the next section a discussion about mutation testing is presented. In Section 3 the criterion Dual Mutation is described. In Section 4 a case study applying DMT is discussed and Section 5 presents the conclusions.

2 Mutation testing

Mutation testing (MT) is used to evaluate the quality of a test set based on its ability to reveal simple faults injected in the program being tested. According to the *coupling effect* a test set capable of revealing simple faults is capable of revealing more complex faults. Some empirical support is available for the coupling effect [10, 12, 13, 16].

In terms of testing criteria, mutation testing can be viewed as a **partition**, or more precisely, a **sub-domain criterion**. Let us consider a program **P** and its input domain $\mathcal{D}$. A sub-domain criterion divides $\mathcal{D}$ in several sub-domains and then determines that test cases should be selected from each sub-domain. In general, the sub-domains are calculated based on some kind of testing requirement. In the case of mutation testing the requirement is that a sub-domain is composed by elements that distinguish a given mutant. More precisely, given the mutants $\{M_1, M_2, ..., M_n\}$ of **P** it is possible to define the sub-domains $\{d_1, d_2, ..., d_n\}$ such that

$$d_i = \{t \in \mathcal{D} | P(t) \neq M_i(t)\}$$

.

Several papers have addressed the problem of evaluating a test criterion or a testing technique based on the characteristics of its sub-domains. Just to mention some of them, the work of Hamlet and Taylor [6] and Duran and Ntafos [5] compared and analyzed the performance of partition testing against random testing and concluded that partition testing is not superior to random testing. In addition, if the cost to apply the partition criteria is high, then random testing is likely more cost effective than partition testing. Weyuker and Jeng [17] analyzed the results presented by Duran and Ntafos and by Hamlet and Taylor and determined the characteristics a partition should have in terms of fault distribution, sub-domain size and test case probability that would make a partition testing better or worse than random testing. Unfortunately, the findings in those papers have not been very useful because in practice it is hard to predict or to assess those characteristics, in particular the fail rate of each sub-domain.

The sub-domains d_i, in general, overlap. In the particular case of mutation testing it has been observed that there is a lot of intersection among them, i.e., it is common to find a test case $\mathbf{t}$ that distinguishes many mutants. Early studies showed that 80% of the mutants generated using FORTRAN operators die very easily, i.e., can be distinguished with any test case [1]. Similar results have been obtained for C operators [7]. On the other hand, a few mutants are hard to kill and require very specific test cases. In this way, it would not be wrong to associate larger sub-domains to those mutants that are easier to distinguish and smaller sub-domains to those mutants that are harder to distinguish.

If we take the sub-domains d_1,... d_n and order them according to its cardinality from the smallest to the largest, and start randomly selecting test data from $\mathcal{D}$, it is most probable that the sub-domains at the end will be "covered" first and the ones at the beginning will last "alive" longer. The quality of mutation testing-adequate test sets rely on those small domains, which represent particular situations and that lead to a more complete examination of $\mathbf{P}$.

3 Dual mutation testing

In this section we present another way to use mutants to select test cases. The idea is to take those mutants that are killed too easily and use them to construct useful test cases. We propose to select test cases that do not distinguish mutants, i.e., test cases $\mathbf{t}$ such that for a given mutant M_i, $M_i(t) = P(t)$. Intuitively, selecting test cases in such a way would lead to sub-domains that would neglect the good quality of sub-domain criteria in at least one way:

- The intersection between them would be high. If one takes two mutants M_i and M_j and selects a test case $\mathbf{t}$ that does not execute the mutated statement in M_i neither the mutated statement in M_j, then it is easy to see that $\mathbf{t}$ is in the intersection of the sub-domains corresponding to those mutants. In general, it would not be difficult to find such a test case.

In summary, killing the mutants would be just a matter of finding test cases that do not execute the mutated statement. But what we mean in the Dual Mutation Testing (DMT) criterion is to select test cases that execute the mutated statement and still produces the same results of the original program.

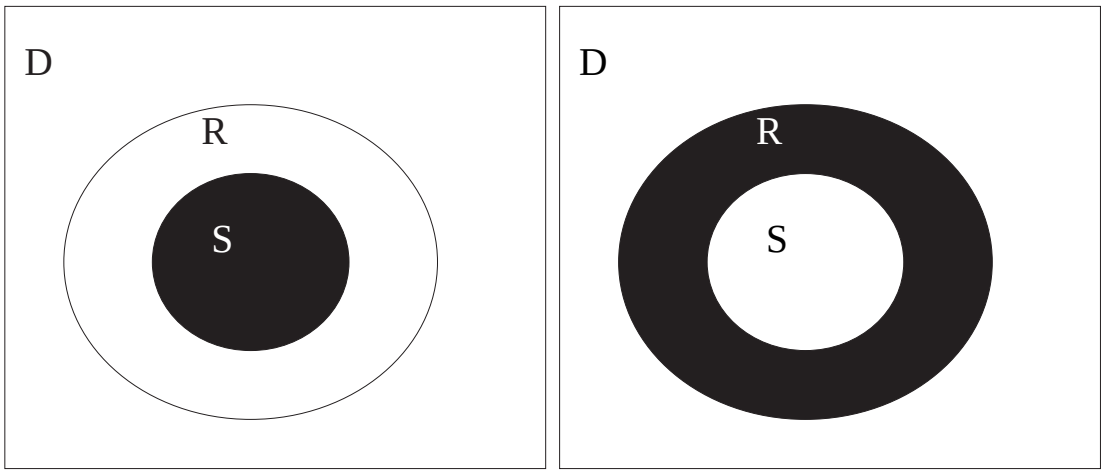
In order to clarify this concept, some definitions are necessary. Consider the program under test $\mathbf{P}$, a test set $\mathbf{T}$, a mutant set $\mathbf{M}$ and the input domain $\mathcal{D}$ of $\mathbf{P}$. Then, according to the DMT technique we have:

Definition 1 *A mutant $M_i \in \mathbf{M}$ is dead when executed with $\mathbf{T}$, iff $\exists \mathbf{t} \in \mathbf{T}$ sucha that two conditions hold:*

- *the execution of $\mathbf{P}$ with $\mathbf{t}$ reaches the statement that was mutated to create M_i;*
- *$M_i(t) = P_i(t)$;*

This definition establishes the necessary conditions to consider that a mutant is dead. Note that in this case, "dead" is the opposite of "distinguished", so the terms cannot be used interchangeably as in conventional mutation testing. We will use $t \succ M_i$ to indicate that test case $\mathbf{t}$ kills mutant M_i and $t \not\succ M_i$ to the opposite case.

The first condition above is also required in conventional mutation testing, as stated by DeMillo [4], but with a slightly different meaning. There, it is not a condition that must be checked in order to consider a mutant dead, but it is a pre-requisite for $\mathbf{t}$ to achieve $M_i(t) \neq P(t)$. Figure 1(a) shows how reachability relates to killability in mutation testing. R is the set of test cases that causes the mutated statement to execute and $S \subseteq R$ is the set of test cases that distinguish M_i. Figure 1(b) highlights the set $R - S$ that is the set of mutants that dual-kill M_i.



(a) (b)

Figure 1: Relation between reachability and killing test cases for (a) mutation testing; (b) Dual Mutation Testing

Definition 2 *A mutant* $M_i \in \mathbf{M}$ *is* **dual-equivalent** *to* $\mathbf{P}$ *iff*

$$\forall \mathbf{t} \in \mathcal{D}, \mathbf{t} \not\succ M_i$$

It means, a mutant is dual-equivalent if there is no test case that executes the mutated statement and makes the mutant behave the same as the original program.

Definition 3 *The (dual) mutation score of a test set* $\mathbf{T}$ *is given by*

$$ms(T, M, P) = \frac{\#\ of\ dead\ mutants}{|M| - \#\ of\ dual\text{-}equiv.\ mutants}$$

Note that this definition has not changed, since we changed the definition of dead mutants in Definition 1.

Lets take as an example the program in Figure 2. It takes as parameters two integers x and y greater than or equal to 0 and computes x^y.

In Table 1 the three first mutants, generated by real mutant operators implemented in the tool PROTEUM/IM [3] show how dual mutation can select very specific test cases out of mutants that are practically useless for regular mutation. The last column shows the conditions required by the input in order to dual-kill the mutant.

```
int pow(int x, int y)
{
int s = 1;
int i;

    for (i = 0; i < y; i++)
    {
        s *= x;
    }
    return s;
}
```

Figure 2: Program to compute x^y.

Table 1: Examples of mutants and test data to dual-kill them.

Original statement	Mutated statement	Mutant operator	Constraint to kill
`s *= x;`	`s *= 0;`	CLSR	$x = 0$ $y > 0$
`s *= x;`	`s *= 1;`	VLSR	$x = 1$ $y > 0$
`s *= x;`	`s *= -1;`	CRCR	$x = 1$ $y \geq 2$ $y\%2 = 0$
`return s;`	`return -1;`	CRCR	equivalent

The last mutant shows that also in dual mutation we are not free from the equivalents. There is no test case that makes the mutant return the same value of the original program. Such a mutant is useless for regular as well as for dual mutation.

In the next section a case study is presented comparing DMT and mutation testing. The objective is to compare the strength of test cases generated by Dual Mutation Testing against mutation testing and vice-versa.

4 A Case study

In this section we present a case study that compares MT-adequate test sets against DMT and vice versa. We used a very simple program (the Unix *cal*) and generated for it 40 test sets using mutation testing and then evaluated these sets using DMT. Then, generated 40 adequate test sets with DMT and evaluated them using mutation testing.

To generate an MT-adequate or a DMT-adequate test set the equivalent mutants are identified then test cases are generated at random using a simple test profile. If a test case kills a mutant the test case is kept in the test set. Otherwise the test case is discarded. This process continues until a mutation score of 1.0 is reached or until no improvement in the mutation score is obtained in a sequence of 1000 test cases, i.e., if 1000 test cases are generated and throw away in a row, then the process terminates. In this case, test

cases are manually inserted to achieve the 100% adequate test set. Figure 3 summarizes the generation of one adequate test set (from this point, called a *section*).

The program *cal* takes zero, one or two arguments to execute. If no argument is provided the program should output the calendar of the current month. If a single argument is provided, it indicates the year whose calendar should be presented. If two arguments are provided, they represent the month and the year the user wants to see. Considering the domain as the set of all sequences of zero, one or two integer numbers in the interval [-MAXINT, MAXINT)[1], the following profile was used to generate the random test cases:

- 90% of the test cases are "valid" test cases, i.e., with the arguments, when provided, in the valid range: 1 to 12 for the month and 1 to 9999 for the year;
- from those valid test cases, 1% is generated with no argument, 49% with a single argument and 50% with two arguments;
- the non valid test cases, are evenly divided in four groups: 1) non-valid year only; 2) non-valid month and valid year; 3) valid month and non-valid year; and 4) non-valid month and non-valid year.

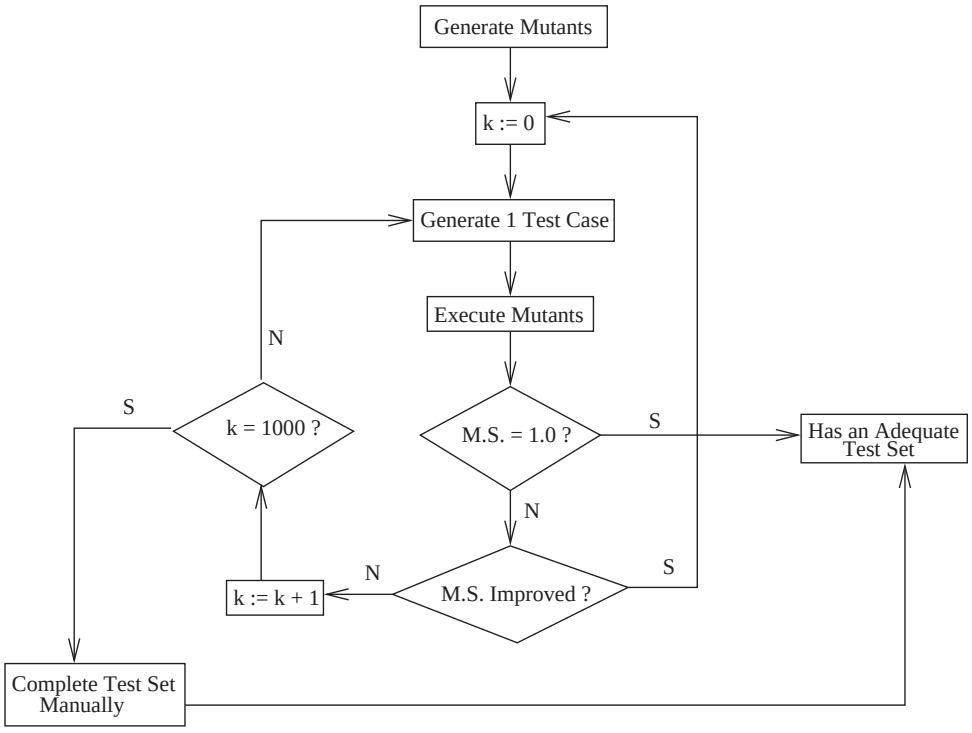


Figure 3: A session to generate one adequate test set

The processes to create mutation testing-adequate and DMT-adequate test sets are the same except for the differences related to the criteria application, as explained in Section 3.

[1]In our study MAXINT = 2^{31}

The program *cal* has four functions: *main*, *cal*, *jan1*and *pstr*. We tested them separately, so actually at the end there exists 20 sections for each function, 10 to generate MT-adequate test sets and 10 to generate the DMT-adequate test sets as shown in Figure 4. Each section in one side of the figure uses independent sequences of random generated test cases. Corresponding sections in both sides – indicated by the dashed arrows – use the same generation sequence.

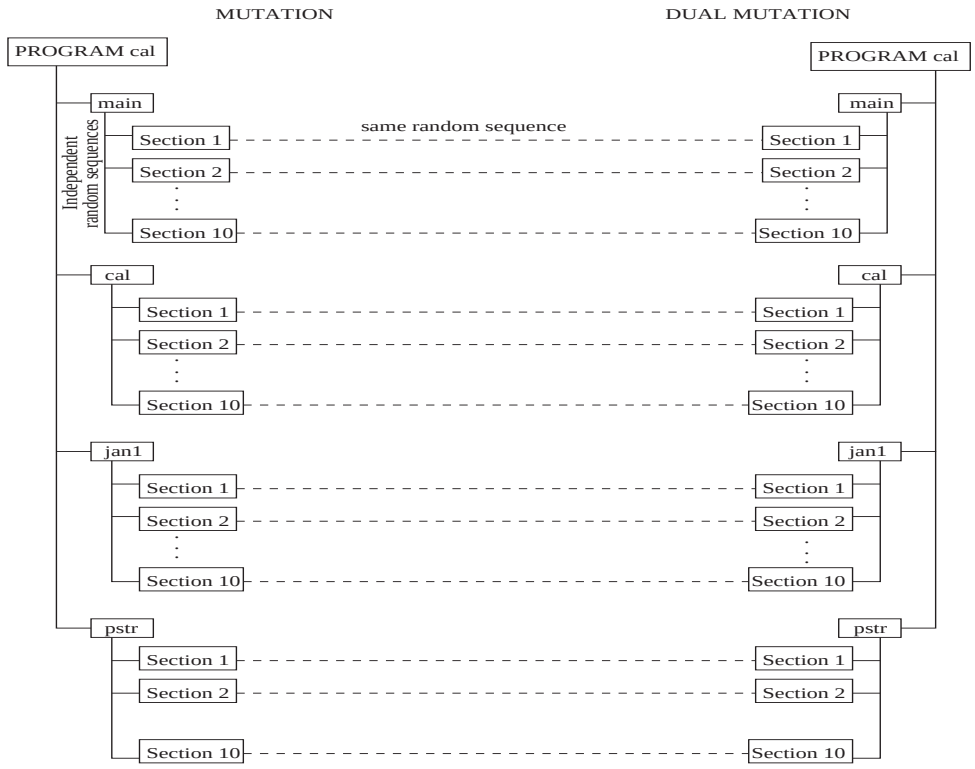


Figure 4: General view of the sections

The tool used in this case study (*PROTEUM/IM*) has two distinct sets of mutant operators: one for unit testing and one for the Interface Mutation criterion [2], aiming at interprocedural testing. We used a subset of the unit testing operators. The reasons we did not used the whole set are two:

- Some instrumented mutants do not make sense for dual mutation. For example the STRP operator replaces each statement by a trap function that distinguishes the mutant. So there is no way that the mutation point is executed and the mutant behaves as the original program; and
- The tool implements a mechanism to control the execution path of each test case in order to avoid the execution of those mutants that are not reached by some test cases. This mechanism is essential for dual mutation because it is necessary to know whether a mutation has not changed the behavior of the program for a test case or the mutation has not been reached by the test case. For a few operators that may change radically the shape of the control flow graph, the tool is not able to make such decision.

Table 2 shows the operators used in the sections and the number of mutants generated for each one.

Table 2: Number of mutants per mutant operator.

Operator	main	cal	pstr	jan1	Total	Operator	main	cal	pstr	jan1	Total
u-Cccr	612	312	14	88	1026	u-Ccsr	403	247	12	72	734
u-CRCR	155	95	20	40	310	u-OAAA	12	17	0	12	41
u-OAAN	48	37	4	36	125	u-OABA	9	12	0	9	30
u-OABN	33	27	3	27	90	u-OAEA	3	4	0	3	10
u-OALN	30	18	2	18	68	u-OARN	90	54	6	54	204
u-OASA	6	8	0	6	20	u-OASN	22	18	2	18	60
u-OBAA	0	0	0	0	0	u-OBAN	0	0	0	0	0
u-OBBA	0	0	0	0	0	u-OBBN	0	0	0	0	0
u-OBEA	0	0	0	0	0	u-OBLN	0	0	0	0	0
u-OBNG	0	0	0	0	0	u-OBRN	0	0	0	0	0
u-OBSA	0	0	0	0	0	u-OBSN	0	0	0	0	0
u-OCNG	9	5	4	2	20	u-OCOR	0	0	0	0	0
u-OEAA	55	50	20	10	135	u-OEBA	33	30	12	6	81
u-OESA	22	20	8	4	54	u-Oido	1	8	6	0	15
u-OIPM	0	0	2	0	2	u-OLAN	15	5	0	0	20
u-OLBN	9	3	0	0	12	u-OLLN	3	1	0	0	4
u-OLNG	9	3	0	0	12	u-OLRN	18	6	0	0	24
u-OLSN	6	2	0	0	8	u-ORAN	60	30	10	10	110
u-ORBN	36	18	6	6	66	u-ORLN	24	12	4	4	44
u-ORRN	60	30	10	10	110	u-ORSN	24	12	4	4	44
u-OSAA	0	0	0	0	0	u-OSAN	0	0	0	0	0
u-OSBA	0	0	0	0	0	u-OSBN	0	0	0	0	0
u-OSEA	0	0	0	0	0	u-OSLN	0	0	0	0	0
u-OSRN	0	0	0	0	0	u-OSSA	0	0	0	0	0
u-OSSN	0	0	0	0	0	u-SBRC	0	0	1	0	1
u-SCRB	0	0	0	0	0	u-SGLR	0	0	0	0	0
u-SRSR	49	31	12	8	100	u-STRI	10	6	4	4	24
u-VDTR	93	57	12	24	186	u-VGAR	28	24	0	0	52
u-VGPR	0	0	0	0	0	u-VGSR	0	0	0	0	0
u-VGTR	0	0	0	0	0	u-VLAR	4	0	0	0	4
u-VLPR	0	11	7	0	18	u-VLSR	197	154	14	26	391
u-VLTR	0	0	0	0	0	u-VSCR	18	0	0	0	18
u-VTWD	62	38	8	16	124	TOTAL	2268	1405	207	517	4397

We start our analysis by comparing the number of equivalent mutants and dual-equivalent mutants. Table 3, as well as many previous experiments, shows that the set of mutant operators we have been using is not too bad in the number of equivalent mutants they produce. It is true that for a toy program like *cal* it is painful to have to analyze 347 equivalent mutants. For dual-mutation the scene is even worse. Table 3 shows that the number of dual-equivalent mutants is much higher than for traditional mutation. In this case study, more than 50% of the mutants are dual-equivalent.

It is interesting to note – and we believe this analysis has not been done before – how bad the mutant operators we use are to create meaningful mutants, i.e., mutants that corroborate to the construction of good test cases. The dual-equivalent mutants show exactly this: the number of mutants in traditional mutation that are killed by any test case that executes the mutation point. We knew that the percentage of mutants that die easily is high but how many die with any test case, only the analysis of dual-equivalence

can show.

In summary, traditional mutation creates "few" equivalents and a large number of useless-always-die mutants. Dual mutation creates a large number of equivalents and few useless mutants. Finding equivalent mutants is much harder than finding the mutants that are always killed so in this matter, DMT is much more expensive than mutation testing. In this case study we felt that identifying dual-equivalent mutants is easier than equivalent mutants. This should not be taken in consideration so far because it lacks scientific bases, but this is a point we plan to investigate in the future.

Table 3: Number of equivalent and dual-equivalent mutants.

Function	Equivalents	Dual Equiv.	Mutants
main	199 (8.8%)	1424 (62.8%)	2268
cal	96 (6.8%)	675 (48%)	1405
pstr	18 (8.7%)	167 (80,7%)	207
jan1	34 (6.6%)	60 (11.6%)	517
TOTAL	347 (7.89%)	2326 (52.90%)	4397

Also in terms of required test cases, DMT has been found more expensive than mutation. Table 4 shows the size of adequate test sets for mutation and for DMT. Except for function *pstr* where the number of test cases is always 1 or 2 for both criteria, the number of test cases required by DMT is always larger than the corresponding set for mutation testing. In some cases like for the *cal* function, the DMT set can be as large as four times the set for mutation testing.

Besides the cost, we can try to analyze what the fact that DMT sets are larger suggests in terms of domain partition. For DMT we are considering a smaller number of mutants – since the number of equivalents is higher – and still the number of test cases is larger. This suggests that the sub-domains overlap less for DMT than for mutation testing. For mutation testing one test case might kill many mutants so the number of required test cases that kill all of them is lower than for DMT. This point must be further explored but it may be an interesting feature and increase the effectiveness of DMT in comparison to mutation testing.

According to Wong [20], two criteria C_1 and C_2 can be compared by their relative strength, i.e., by evaluating how a C_1-adequate test set behaves in relation to C_2 and vice-versa. Mutation testing has been shown an effective testing criterion [20] so a criterion with a high relative strength in relation to it is expected to have similar fault revealing effectiveness.

In this case study each of the 40 DMT-adequate test sets were evaluate in relation to mutation testing and each MT-adequate test set were evaluated in relation to DMT. Table 5 shows the result. The fourth column displays the average mutation score obtained by DMT-adequate test sets when evaluated by mutation testing. The second column shows the average mutation score obtained by MT-adequate test sets when evaluated by DMT.

It can be observed that in this matter, the study indicates that DMT is better than MT. The average mutation scores obtained by DMT-adequate test sets is significantly larger than those obtained by MT-adequate sets. In addition, except for function *pstr*, the

Table 4: Number of test cases in the adequate sets.

	Mutation Testing			Dual Mutation		
Function	Test cases	Avg.	Std. Dev.	Test cases	Avg.	Stdv Dev.
main	40 32 30 27 30 32 29 29 29 31	30.9	3.5418	54 57 54 56 55 59 55 55 55 55	55.5	1.5092
cal	7 8 5 16 7 9 10 9 8 6	8.5	3.0277	35 32 36 31 33 34 38 36 38 32	34.5	2.5055
pstr	2 2 1 2 1 1 2 2 2 1	1.6	0.5164	1 2 2 1 2 2 1 2 2 2	1.7	0.4830
jan1	16 12 17 14 12 16 14 14 16 15	14.6	1.7127	35 37 38 39 37 31 35 40 38 37	36.7	2.5408

largest score obtained by the MT-adequate sets is always bellow the lower score obtained by the DMT-adequate sets. For *pstr* the small number of test cases in the adequate sets produces very different scores between the sets. Some DMT-adequate sets achieved scores of 1.0 and some scores around 0.88. The MT-adequate sets concentrate their scores on 1.0 and 0.45.

Table 5: Comparison of DMT-adequate and MT-adequate test sets.

	MT-adequate to DMT		DMT-adequate to MT	
Function	Average MS	Std. Dev.	Average MS	Std Dev.
main	0.9364	0.0069	0.9886	0.0008
cal	0.7168	0.0797	1.0	0.0
pstr	0.6150	0.2657	0.9646	0.0571
jan1	0.7473	0.0720	0.9733	0.0137

A last characteristic we tried to evaluate in this case study is the intersection between the correspondent MT-adequate and DMT-adequate sets. As indicated in Figure 4, one MT-adequate and one DMT-adequate test sets use the same random generated sequence of test cases to try to kill their mutants. What we would like to know is whether they select the same test cases or different test cases.

Table 6 shows in the fourth column the size of the test sets generated using DMT criterion when used to evaluate MT-adequacy (only the test cases that kill at least one mutant). The second column shows the analog case where MT-adequate sets are used on DMT. These two columns give an idea of how the test sets behave in relation to its dual criterion. For example, we can see for function jan1 that from the DMT-adequate test sets which average 36.7 test cases, only 7.5 test cases are required to obtain the mutation score of 0.9733 as shown in Table 5. On the other hand, 14.1 out of the average 14.6 test cases of the MT-adequate sets are required to obtain the DMT score of 0.7473.

The sixth column shows the average size of the intersection between the effective sets used with their dual criteria. This number gives the idea of how many test cases are common in the MT and DMT-adequate sets. For example, for function *main* we can say that 19.6 out of the 30.9 test cases that are MT-adequate are also in the DMT-adequate set (in average 55.5 test cases large).

Table 6: Size of effective test sets.

	MT-adequate to DMT		DMT-adequate to MT		Intersection	
Function	Set length	Std. Dev.	Set length	Std Dev.	Set length	Std Dev.
main	22.4	1.5776	22.4	1.5055	19.60	1.5055
cal	7.4	1.8379	7.0	1.4142	6.4	1.0750
pstr	1.0	0.0	1.3	0.4830	1.0	0.0
jan1	14.1	1.8529	7.5	1.5092	6.70	1.4944

5 Conclusions

This paper presented the idea of Dual Mutation Testing (DMT). It uses mutants to create test cases for a given program, similar to what is done in traditional mutation testing. The difference is that it considers dead those mutants for which the tester has provided a test case that reaches the mutation point and does not distinguishes the mutant, i.e., that makes the mutant behave as the original program. The motivation behind this idea is trying to use those mutants that in traditional mutation testing are easily killed and do not contribute to create good test sets.

In a case study using a very simple program a few first results could be drawn. The first is that the number of equivalent mutants is much higher for DMT than it is for mutation testing. This is a consequence of the fact that a large number of mutants are useless for mutation testing. They are not only easy to kill, but any test case that reach the mutation point would distinguish them. They are the dual-equivalent mutants. This is a negative point to the use of DMT because analyzing equivalence or dual-equivalence is certainly the most expensive task in mutation based testing.

The cost of DMT is high also in terms of test cases needed to fulfill its requirements. The data show that the number of test cases necessary to build an DMT-adequate test set can be as large as four times the size of MT-adequate test sets. On the other hand, this may indicate that the sub-domains determined by each mutant in DMT overlap less than in traditional mutation. In traditional mutation a single test case can kill a large number of mutants and it seems that in DMT this number might be smaller. If the other problems with DMT can be overcome, this may be a good feature to improve fault detection effectiveness.

Another positive point in favor of DMT is the fact that DMT-adequate test sets are much closer to MT-adequacy than MT-adequate test sets are to DMT-adequacy. In addition, using DMT-adequate test sets to evaluate MT-adequacy produces test sets smaller than those produced by generating random test sets and then completing them by hand (the way the adequate test sets were produced in this case study). This suggests a way to use DMT and MT in conjunction.

And this is the direction we plan to follow in this research. Initially it is necessary to expand the knowledge about DMT with more complete experiments. Comparing it with other criteria is also a strategy to follow. In particular, comparison with mutation testing can suggest how they relate to each other.

Then we plan to explore ways to use MT and DMT together. Experiments might suggest sets of mutant operators that are more adequate to DMT than to MT or more adequate to MT than to DMT. Other ways of using both might be explored as well, for example using one of them to pre-select test cases to the other.

References

[1] T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Theoretical and empirical studies on using program mutation to test the functional correcteness of programs. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, pages 220–233, New York, NY, 1980.

[2] M. E. Delamaro, J. C. Maldonado, and A. P. Mathur. Interface Mutation: An Approach for Integration Testing. *IEEE Transactions on Software Engineering*, 27(3):228–247, March 2001.

[3] M. E. Delamaro, J. C. Maldonado, and A. M. R. Vincenzi. Proteum/IM 2.0: An integrated mutation testing environment. In *Mutation 2000 Symposium*, pages 91–101, San Jose, CA, October 2000. Kluwer Academic Publishers.

[4] R. A. DeMillo and A. J. Offutt. Constraint Based Automatic Test Data Generation. *IEEE Transactions on Software Engineering*, 17(9):900–910, September 1991.

[5] J. Duran and S. Ntafos. An evaluation of random testing. *IEEE Transactions on Software Engineering*, SE-10:438–444, July 1984.

[6] D. Hamlet and R. Taylor. Partition Testing Does Not Inspire Confidence. *IEEE Transactions on Software Engineering*, 16(12):1402–1411, December 1990.

[7] R. F. Jorge. Teste de mutação: Subsídios para a redução do custo de aplicação. Master's thesis, ICMC-USP, São Carlos – SP, February 2002.

[8] R. F. Jorge, M. E. Delamaro A. M. R. Vincenzi, and J. C. Maldonado. Teste de Mutação: Estratégias Baseadas em Equivalência de Mutantes para Redução do Custo de Aplicação (Mutation Testing: Equivalency Based Strategies for Cost Reduction - in Portuguese). In *XXVII Latin-American Conference on de Informatics (CLEI)*, Meridas, Venezuela, June 2001.

[9] A. P. Mathur. Performance, Effectiveness and Reliability Issues in Software Testing. In *Proceeding of the 15th Annual International Computer Software and Applications Conference*, pages 604–605, Tokio, Japan, September 1991.

[10] L.J. Morell. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, 16(8):844–857, August 1990.

[11] A. J. Offut and W. M. Craft. Using compiler optimization techniques to detect equivalent mutants. *Journal of Software Testing Validation and Reliability*, 4(3):131–154, 1994.

[12] A. J. Offutt. Coupling Effect: Fact or Fiction. In *Proceedings of the 3rd Symposium on Software Testing, Analysis, and Verification (ISSTA'89)*, pages 131–140, Key West, FL, December 1989.

[13] A. J. Offutt. Investigations of the software testing coupling effect. *ACM Transactions on Software Engineering Methodology*, 1(1):3–18, January 1992.

[14] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. An Experimental Determination of Sufficient Mutant Operators. *ACM Transactions on Software Engineering Methodology*, 5(2):99–118, 1996.

[15] A. J. Offutt, G. Rothermel, and C. Zapf. An Experimental Evaluation of Selective Mutation. In *Proceedings of the 15th International Conference on Software Engineering*, pages 100–107, Baltimore, MD, May 1993.

[16] K. S. H. T. Wah. Fault coupling in finite bijective functions. *Journal of Software Testing Verification and Reliability*, 5(1):3–47, March 1995.

[17] E. J. Weyuker and B. Jeng. Analyzing Partition Testing Strategies. *IEEE Transactions on Software Engineering*, 17(7):703–711, July 1991.

[18] W. E. Wong. *On Mutation and Data Flow*. PhD dissertation, Department of Computer Science, Purdue University, W. Lafayette, IN, December 1993.

[19] W. E. Wong and A. P. Mathur. Reducing the Cost of Mutation Testing: An Empirical Study. *The Journal of Systems and Software*, 31(3):185–196, December 1995.

[20] W. E. Wong, A. P. Mathur, and J. C. Maldonado. Mutation Versus All-uses: An Empirical Evaluation of Cost, Strength, and Effectiveness. In *Proceedings of the International Conference on Software Quality and Productivity*, pages 258–265, Hong Kong, December 1994.