An Investigation into the Effect of Control and Data Dependence Paths on Predicate Testability

Dave Binkley
Loyola University Maryland

James Glenn
Yale University

Abdullah Alsharif
Saudi Electronic University

Phil McMinn
University of Sheffield

***Abstract*—The *squeeziness* of a sequence of program statements captures the loss of information (loss of entropy) caused by its execution. This information loss leads to problems such as failed error propagation. Intuitively, longer more complex statement sequences (more formally, longer paths of dependencies) bring greater squeeze. Using the cost of search-based test data generation as a measure of lost information, we investigate this intuition. Unexpectedly, we find virtually no correlation between dependence path length and information loss. Thus our study represents an (unexpected) negative result.**

Moreover, looking through the literature, this finding is in agreement with recent work of Masri and Podgurski. As such, our work replicates a negative result. More precisely, it provides a conceptual, generalization and extension replication. The replication falls into the category of a conceptual replication in that different methods are used to address a common problem, and into the category of generalization and extension in that we sample a different population of subjects and more rigorously consider the resulting data. Specifically, while Masri and Podgurski only informally observed the lack of a connection, we rigorously assess it using a range of statistical models.

I. Introduction

When a program "squeezes out" information, random and search-based test data generation work less effectively. For example, when a program reduces a large input domain down to a small number of state values, it becomes less likely that random search will find inputs that lead to a particularly rare program state. Moreover, when there is very little state information, the fitness landscape for a predicate may be reduced to a small number of plateaus, resulting in a lack of guidance for the search in search-based testing [?], [?].

The *squeeziness* of a sequence of program statements is the loss of information (loss of entropy) caused by its execution [?], [?]. Information loss occurs when these statements reduce the amount of information present in the current execution state. For instance, the statement `x = x % 2` (where `x` is an 8-bit unsigned integer) reduces `x`'s 256 values down to just two. A killing assignment, which overwrites the value of a variable, is the most extreme example of information loss, removing all information that a variable previously held.

The information loss that leads to a reduction in entropy is caused by the particular computations of the program. In this paper, we study the intuition that more complex computations, specifically those associated with longer sequences of statements, lead to greater loss. In conversations regarding this work, other researchers have ubiquitously agreed with this intuitive appeal. Yet, recent evidence in the literature suggests otherwise: Masri and Podgurski [?] studied entropy loss and dependence path length, but found no relation. This paper aims to shed light into this apparent paradox by presenting a conceptual, generalization and extension replication [?] of the work of Masri and Podgurski [?].

We formalize computation complexity using control and data dependencies and thus more formally we investigate the relationship between *dependence path length* and information loss. For example, a data dependence path of length two connects the definition `sum = 0` to `sum = sum + A[i]`, and finally to `average = sum / count`. Our study focuses on correlating dependence path length with test data generation expense, which is measured as the average number of test data (fitness) evaluations needed to find test data to execute the statement at the end of a given dependence path. We measure this expense using three different algorithms (one random and two search-based). Because it captures the effort required by the test data generation algorithm, it captures "difficulty," and thereby forms the metric that we use in our experiments to assess *testability*.

Note that we are not proposing an absolute difficulty scale. In fact, we *expect* each different algorithm to use different numbers of fitness evaluations. However, for a particular algorithm, the number of evaluations provides a relative measure of the effort required and hence is sufficient for our experiments. Our estimation of "testability" in this way is reminiscent of studies such as the early work of Voas and Miller [?].

Considering the testability and dependence paths for 416 predicates taken from 28 C functions found in four programs, this paper investigates the correlation between path length and testability. The core of the paper addresses this question using over 1500 statistical models including both linear and nonlinear models. For example, if long paths of data dependencies, rather than control dependencies, bring greater domain squashing and thereby increase test generation cost, then we have support for data dependence being the cause of squeeziness, and consequently, tool designers should pay greater attention to data dependence issues.

II. Background

Search-based software testing formulates the problem of test input generation as an *optimization problem* that can be attempted by a meta-heuristic search technique, such as a Genetic Algorithm. Meta-heuristic algorithms rely on a fitness function to guide the search towards a global optimum (which,

```
1  void fn_under_test(float x, float y, float z) {
2    float a = x + 1;
3    if (a == 0) {
4      float b = y / 2;
5      float c = z * 2;
6      if (b == c) {
7        if (a == c) {
8          /* target */
9    ...
```

Fig. 1. Example C function

for this problem, is the desired test inputs). In this paper, we are interested in estimating the computational effort, or the *cost* of generating test data, which for search-based approaches is equivalent to the number of fitness function evaluations needed to find particular inputs.

Because we are interested in the difficulty of generating inputs for specific predicates, we apply the "traditional" fitness function that focuses on generating test data that executes specific "targets" (e.g., a branch) in the source code, as opposed to approaches that seek to execute as many branches as possible with the same inputs (as with the whole test suite approach used by EvoSuite [**?**], and multi/many-objective approaches, e.g., MOSA [**?**]). To cover specific targets in a program, for instance a particular branch from a decision statement, the fitness function comprises two measures: an approach level and a branch distance [**?**]. The fitness function is minimized by the search, such that when both of these measures are zero, the desired test inputs have been found. The approach level records how many of a target's control predecessors were not executed by a particular input. The fewer control dependent nodes executed, the "further away" an input is from executing the target in terms of the program's control flow. Consider the example shown in Figure **??** and assume the target is Statement 8. If an input reaches Statement 3, but takes the false branch, the approach level is 2; if an input reaches Statement 6, but takes the false branch the approach level is 1, and finally 0 if the input reaches Statement 7 and takes the false branch.

Whenever an input misses the current target branch, the *branch distance* measures how "close" the input was to staying on a path leading to the target. The branch distance is computed using the condition of the last (Control Flow Graph) node in an input's execution trace that holds a transitive control dependence on the target, and where execution diverged from the target. Resuming the example from Figure **??**, if an input takes the false branch at Statement 3, the branch distance is calculated using $|a - 0|$, i.e., using the values the program computes to evaluate the predicate at runtime. The exact branch distance formula used depends on the predicate [**?**]. In this way, control dependencies factor into the approach level part of the fitness formula, while data dependencies influence branch distance calculations. For example, the branch distance computed at Statement 7 involves the values of the variables a and c, which are defined at Statements 2 and 5 respectively.

The search algorithm typically starts off by assigning random values to program inputs and then, using a meta-heuristic algorithm guided by the fitness function, updates these values. The number of fitness evaluations is a measure of how many modifications to the program inputs were required before the search found values that reach the desired target. The number of fitness evaluations required by a search therefore approximates the "difficulty" of finding program inputs that cover a given target. In this paper, we study two meta-heuristic search techniques, the Alternating Variable Method (AVM) and the Genetic Algorithm (GA), both of which are implemented in the publicly available IGUANA search-based test data generator for C functions [**?**], [**?**]. AVM was originally due to Korel [**?**]. It takes as input a test vector and makes local improvements, starting with small changes to each input and successively making increasingly larger changes so long as fitness continues to improve. The GA built into IGUANA models that of Wegener et al. [**?**]. It uses a population of 300 individuals (input vectors) split into six competing subpopulations, each with a different mutation rate. It uses discrete recombination as a crossover method, tournament selection, and an elitist reinsertion method, where the top 10% of the individuals from the last generation are automatically re-inserted into the next.

Additionally, we apply Random search as a baseline technique in this paper, which simply generates random values for each variable of a test input vector. Each algorithm iterates until it finds inputs that execute the target, or resources are exhausted (due to a timeout or the search reaching the fitness evaluation limit).

III. Reference Experiment

This section briefly describes the reference experiment conducted by Masri and Podgurski [**?**]. Because we conduct a conceptual replication we are not attempting to reproduce the setup and structure of the reference experiment; thus we focus this description on the relevant points of similarity and difference required to provide context for our experiments and the subsequent discussion and comparison of its results in Section **??**.

To begin with, we consider completely different subject programs (of comparable size) written in a different language (C versus Java). Consideration of a completely new data set serves to increase the external validity of the results. We also consider the *static* control and data dependencies provided by CodeSurfer [**?**], in contrast to the reference experiment, which makes use of *dynamic* control and data dependencies. Here again, similar findings will serve to increase external validity.

In addition, both studies consider paths of only data dependencies, only control dependencies, and both dependence kinds. Finally, both experiments include tools that apply to scalar variables only. However, because the reference experiment studies Java, it is able to exploit an object's hashCode method, which we are not able to mirror in C.

In addition to being a conceptual replication, we also provide a generalization and extension replication. For example, we incorporate test generation while Masri and Podgurski focus solely on the relation between dynamic data dependence and information flow. We also provide more elaborate empirical study primarily by applying a range of statistical models to our data. Masri and Podgurski are less formal in their study. For example, they provide primarily visual evidence such as "examination of the charts does not reveal any consistent pattern in the relationship between length and average strength" where we rely on statistical evidence.

Where the two studies share the most in common is in their primary research question. Both studies include some preliminary research questions dealing with the suitability of the data gathered. Masri and Podgurski introduce their primary research question, "is the length of an information flow indicative of its strength?", by saying that "it seems plausible that the strength of information flows tends to attenuate as their length increases." Attenuated flow is equivalent to greater loss and thus this research question mirrors our main research question, *can testability (our measure of information loss) be modeled using dependence path length?*

IV. Experimental Design

Path Analysis. We use Grammatech's CodeSurfer [?] to build the System Dependence Graph (SDG) for each program. This graph includes a Procedure Dependence Graph (PDG) for each of the program's functions [?]. To match the testability data, the path data is collected at the function level; and therefore the starting point of a path is either the PDG's entry vertex, its body vertex, or one of its formal-in vertices. The ending point of a path is the vertex representing the component under test (the target of the search for test data). Paths are composed of control and data dependence edges. A control dependence edge captures the controlling influence of a predicate. A data dependence edge captures the flow of values from a definition to each use reached by the definition.

Three different kinds of paths are considered: *Control Only*, CO, paths composed of only control dependence edges, *Data Only*, DO, paths of only data dependence edges, and finally, *Both*, BO, paths of intermixed control and data dependence edges. Thus each target has three sets of paths corresponding to the use of CO, DO, and BO. The lengths of these paths are summarized by computing the *minimum path length*, min, the *maximum simple path length*, max, and the *mean path length*, mean.

The problem of computing the maximum path length in a graph is *NP*-hard, and approximating the maximum in a directed graph [?] and computing the mean path length [?] are also difficult. However, for directed acyclic graphs (DAGs), both maximum and mean can be computed with a polynomial-time algorithm that avoids enumerating all paths, and many of the PDGs under study are DAGs. Furthermore, in many of the remaining cases, the PDGs can be decomposed into DAGs of strongly connected components that are small enough to compute the maximum and mean path length by enumerating all of the paths, or by removing edges one-by-one and proceeding recursively. For the few remaining cases we use importance sampling [?] to estimate the mean path length, where the sampled paths are generated by a naive random walk. We then take the length of the longest path generated as an estimate of the maximum path length. If such cases were more prevalent it would be worthwhile to improve the estimated mean using more sophisticated sampling, for example, Roberts and Kroese's length-distribution method [?].

Of the 416 predicates we used in our study, which we introduce fully below, only 33 required estimation of path length. Furthermore, for 31 of these the search fails to reach the given target. As such, the data used to build the statistical models require only two path-length estimates.

Statistical Analysis. To capture testability, the statistical analysis uses test data search cost measured as the number of fitness evaluations, Evals, as the response variable. It also uses the three path length measures (min, mean, and max) along with File, the name of the source-code file, as explanatory variables. Including File enables each model to have a different intercept for each file, enabling them to account for artifacts of a given file's coding style not related to the correlation of testability to dependence path length.

Using these variables, we build linear regression models using the lm function implemented in R [?] to identify correlations between the response and explanatory variables. We used Stepwise Elimination [?], which removes the explanatory variable with the highest p-value provided that this value of greater than 0.05 and then rebuilds the model until only significant variables remain. Note that some non-significant variables ($p > 0.05$) are retained to preserve a hierarchical well-formulated model [?]. This occurs in the analysis, for example, when only some of a categoric variable's levels are significant.

We are aware that the statistics community now advocates a move away from the historic fixation on the use of a 0.05 significance threshold [?]. Unfortunately, they are yet to propose a replacement for its use in stepwise elimination. Based on the "World Beyond $p < 0.05$" proposals [?], the resulting elimination models include a greater number of explanatory variables than they might under more stringent criteria. We take this into account when describing the models and their implications.

Subjects. The source code studied was taken from the real-world systems *gimp*, *R*, *grelt*, and *spice*, which previously featured in studies of squeeziness [?] and search-based testing [?]. These programs contain many functions, from which we chose the 28 shown in Table **??**. In choosing these functions, we looked for functions directly called by a user or a program from outside of the project. Each function features boolean, integer, or floating point formal parameters, domains in which random and search-based approaches are easily applied [?]. Furthermore, our goal is to study predicates with a range of testability, it was necessary to select programs for which our chosen random and search-based techniques were likely to work and have a range of performance.

In addition to the program, file, and function, Table **??** provides descriptive statistics for the 28 functions studied. These include the number of Lines-of-Code (LoC) for each function as measured by the Unix utility wc and source lines of code (SLoC) as measured by sloc_count_c [?]. The final two columns show the number of predicates in each function

TABLE I
SOURCE CODE STUDIED

		Size		Predicates	
Program/File	Function	LoC	SLoC	Paths	Matched
gimp					
gimpdraw.c	gradient_calc_bilinear_factor	45	32	4	4
gimpdraw.c	gradient_calc_conical_asym_factor	62	47	6	5
gimpdraw.c	gradient_calc_conical_sym_factor	67	51	7	6
gimpdraw.c	gradient_calc_linear_factor	51	37	6	5
gimpdraw.c	gradient_calc_radial_factor	40	28	4	4
gimpdraw.c	gradient_calc_spiral_factor	59	43	5	4
gimpdraw.c	gradient_calc_square_factor	54	42	5	5
grelt					
i0.c	cephes_bessel_I0	13	11	2	2
i0.c	i0e	13	11	2	2
k0.c	cephes_bessel_K0	20	16	2	2
k0.c	k0e	19	15	2	2
unity.c	cephes_exp	22	17	7	3
unity.c	cephes_log	13	10	2	1
unity.c	cosm1	13	10	2	1
R					
gamma.c	gammafn	177	131	31	19
gamma_cody.c	gamma_cody	232	92	15	15
pnchisq.c	pnchisq	28	24	20	12
pnchisq.c	pnchisq_raw	182	145	31	19
pnorm.c	pnorm5	26	19	21	7
pnorm.c	pnorm_both	190	140	30	14
ptukey.c	ptukey	201	93	33	19
ptukey.c	wprob	162	86	10	10
rhyper.c	afc	30	28	2	2
rhyper.c	rhyper	259	214	56	43
toms708.c	bratio	334	243	73	41
toms708.c	psi	195	91	14	14
spice					
spice.c	clip_to_circle	244	158	24	21
Total		2751	1934	416	282

and the number after matching, as we describe in the next subsection. In the analysis, each predicate gives rise to three data points: one for each search method, GA, AVM, and Random.

Data Collection. IGUANA is designed to generate test data to cover individual branches. We can therefore use the number of fitness evaluations needed to execute a branch as a measure of the effort expended by a search algorithm in generating inputs for the branch's predicates. The more effort required, the more difficult the search problem, and the less *testable* it is.

Each conditional statement is associated with two branches: one that requires the conditional evaluating to true, and the other to false. Thus, for each predicate, IGUANA provides two fitness evaluation counts. Both capture the challenge in reaching the conditional statement. What remains is to capture the challenge represented by condition itself. Consider the condition a == b. Finding values that make a and b the same better captures the challenge in this predicate than finding values to make the two differ. Generalizing this observation, in our analysis we use the larger of the two fitness evaluation counts. In this way, by taking into account both the true and false outcomes of each conditional, our experiments account for both reaching a conditional and the computation of the conditional.

Consider again the code shown in Figure **??**, taking more simply the execution of the predicate at Statement 7 as the target (rather than the true branch from Statement 7, leading to Statement 8), and an input that causes the predicate at Statement 6 to be true. This same input very likely produces the value false for the predicate at Statement 7. However, further search effort (i.e., further fitness evaluations) is required to make the predicate at Statement 7 true (i.e., to make c equal to a as well as b). Here, it is the larger of the two search efforts that captures the prior computations of a and c.

Because of the stochastic nature of the search for test data, each search was run 30 times for each target, as is common in search-based testing experiments (e.g., [**?**]). This raises the question of how to best aggregate the multiple runs. Obviously the more data the better. We initially considered the mean of the 30 runs, but this necessitates the omission of predicates for which as few as one of runs fails to find viable test data since we treat such results as effectively ∞. The median is better able to include *hard* targets that skirt the resource limit. In the analysis we consider both the mean and the median so that we can assess the impact of this choice.

The success of a given search also depends on the size of the search space (i.e., the input domain used for each program). For example, for input variables of integer type, a tester will set a realistic range (e.g., -1,000 to 1,000) rather than use the whole range of the type. However, larger search spaces typically require more work on behalf of the search. To counter this potential threat, we used two input domain sizes. All inputs to the programs we studied were numeric, so with the *smaller* range, every integer has a range of -1,000 to 1,000, unless it was used as a Boolean by the program, in which case the range was 0 to 1; while we set every floating point variable to a range of of -100.0 to 100.0 (i.e., the same size as an integer variable, but accounting for one decimal place). For the *large* range, we followed the same rules, but use a range of -10,000 to 10,000 for integers and -1,000.0 to 1,000.0 for floating point types.

Using both the median and the mean and having two ranges gives rise to the four data sets used in the experiments: large-mean, large-median, small-mean, and small-median. Each of these provides a value of the response variable, Evals, which is our measure of testability.

To produce the final data sets, this testability data must be matched with the path-length data. Given our use of three kinds of paths, BO, CO, and DO, the matching process produces at most 416 × 3 values. In fact it produces fewer than this for several reasons. First, there are 65 predicates that are unreachable via paths of only data-dependence edges. This occurs, for example, when the predicate is a constant such as found in the oft-seen conventional C macro "#define printd if (DEBUG) printf", where DEBUG is either true or false and thus not data dependent on any variable.

The rest of the loss is caused by macro expansions, compound conditions, and timeouts. The first of these arises because the PDG is built after the C pre-processor has run, while IGUANA works with the original source code. For some more complex macro expansions it is not practically feasible to match the two. The second loss occurs because the matching is done per-condition, which is the level at which IGUANA works, and not per-predicate, which is the level at which CodeSurfer works. Thus for a compound condition such as a $>$ b || x $>$ y, which includes the two predicates a $>$ b and x $>$ y, we ignore the control-flow implications of C's short circuit evaluation.

These structural matching losses are independent of a model's ability to find test data and thus apply equally to all three models. For each function the number of predicates post matching is shown in the Table **??** column "Matched". In total, there are 282 matched predicates, which includes 43 of the 65 unreachable via DO paths. Thus we consider a total of 282 CO paths, 282 BO paths, and 239 DO paths for a total of 803.

The final loss category occurs when IGUANA fails to terminate and is therefore dependent on the data set. After ten minutes, an external monitoring script kills IGUANA causing no output to be produced for the current predicate and *also* for all subsequent predicates in the current file. We record an ∞ value for the current predicate just as if it had reached the fitness evaluation limit. However, we get no information about the subsequent predicates and thus record no data for them for the current run.

Obviously if all thirty runs omit a predicate we are forced to drop the predicate because it has no matching data. However we retain those that produce partial data. In no case did a predicate produce fewer than twelve (of 30) values, which we deemed sufficient for our analysis. The final number of matched predicates is shown in Table **??** where it is broken down based on method and data set size (see the rows labeled "pre-filter"). It is encouraging to note that for GA, and for AVM using the large range, the timeout monitor did not terminate the search, as the pre-filtered data includes all 803 predicates.

V. Empirical Analysis

This section considers three research questions. The first two take a look at the testability data and the path length data, and ask whether the data is sufficient to be amenable to further analysis. The core of our investigation is captured in the third research question, which investigates the correlation between path length and testability. After presenting our results for each of the three RQs, this section discusses threats to the validity of our experiments. The next section compares our results in light of the findings of the reference experiment introduced in Section **??**. It also discusses implications of both studies essentially yielding a negative result.

A. *RQ1: Does testability exhibit sufficient variation?*

RQ1 asks if the variation in the effort required to generate test data is sufficient to warrant study. To address this question, we consider the number of fitness evaluations used to find test data for each of the matched predicates. It should come as no surprise that there are predicates for which the search fails. As an extreme example, this is true for all unreachable predicates. The impact of these searches is evident in the upper graph of Figure **??**, which shows the effort required (regardless of method) using the small-median data set. The resulting flat region on the right at 10,000 (the y-axis uses a log_{10} scale) negatively impacts the creation of linear models because such models aim to fit the data with a sloped regression line. Therefore we filter out all predicates for which the search fails.

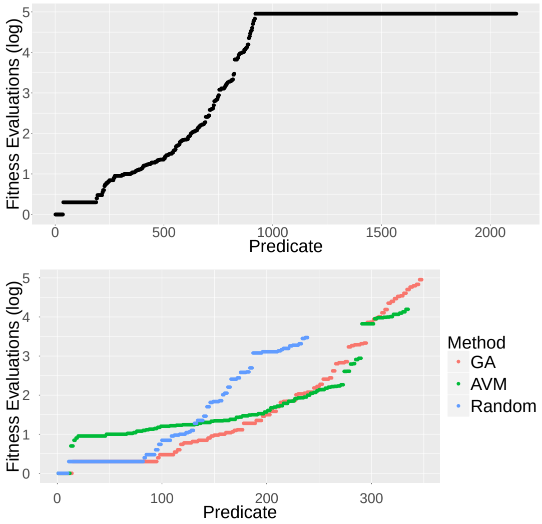

Fig. 2. Testability variation of the small-median data. The others are similar. The lower chart breaks the filtered data out by method, showing that each method includes ample variation. (Note that the y-axis uses a log_{10} scale.)

For the small-median data set, the lower graph of Figure **??** shows the filtered data, this time broken out by method. The other data sets produce virtually identical graphs. Table **??** shows the actual predicate counts pre- and post-filtering. The filtered values are shown for all four data sets and further broken down for each path type: BO, CO, and DO.

Numerically filtering removes about 63% of the predicates (Interestingly Masri and Podgurski at a similar stage in their experiments remove a comparable 61% of their "flows"). While more data would be preferable, the fact that so many predicates prove so difficult is an artifact of our choosing targets from production source code. With approximately one hundred predicates in each remaining category, there is sufficient data to support the statistical analysis. This observation is supported statistically by the bootstrapping considered in Section **??**. Combined with the range of values shown in Figure **??** where the y-axis uses a log scale and thus the data spans five orders of magnitude, we conclude that there is sufficient variation in the testability data. The filtered data sets are used in the remainder of the analysis.

B. *RQ2: Does path length exhibit sufficient variation?*

The second research question mirrors the first except that it looks at the dependence path data. Using the small-median data set, the three charts shown in Figure **??** graph the density of the three edge sets. (Informally, these graphs can be thought of as smoothed histograms where the area under the curve is normalized to one; thus, the area under the curve between two x values is the probability of a value being between those two x values.) The other three data sets produce only minor

TABLE II
Predicate counts

Data set	GA	AVM	Random	Total
large pre-filter	803	803	661	2267
mean filtered	284	321	206	818
BO	98	111	71	280
CO	98	111	71	280
DO	88	99	64	251
median filtered	322	375	239	936
BO	111	129	82	322
CO	111	129	82	322
DO	100	117	75	292
small pre-filter	803	656	661	2120
mean filtered	302	323	239	864
BO	104	111	82	297
CO	104	111	82	297
DO	94	101	75	270
median filtered	348	335	239	922
BO	120	115	82	317
CO	120	115	82	317
DO	108	105	75	288

variations, but retain the same general shape. Each graph has a line for each of the min, mean, and max path-length measures.

Despite being dominated by smaller values, the graphs shown in Figure **??** illustrate that the dependence path lengths show sufficient variation for use in the study. This is more true for the mean and max lengths, which is supported by the statistical analysis presented in the next section.

C. RQ3: Can testability be modeled using dependence path length?

To answer our main research question, we built statistical models for each pairing of a search method (GA, AVM, or Random) with a path edge type, (BO, CO, or DO). We built these nine models using each of the four data sets, resulting in the 36 models considered in this analysis. (The next section considers additional models including non-linear models.) Providing the details of all 36 models is overwhelming and space consuming, so where appropriate we focus the discussion on the nine models of the small-median data set. The others are comparable. Each model seeks to characterize testability, as captured by Evals, as a function of the explanatory variables, which include the three measures of path length, min, mean, and max, along with File, the file from which the predicate was taken.

As mentioned in Section **??**, the explanatory variable File enables the models to have different intercepts for each file and thus to account for differences related to local programming style. The estimates are relative to gimpdraw.c, which is randomly chosen by the analysis. In most cases the differences are not statistically significant. However, if for at least one file the difference is significant, then the explanatory variable File is retained in the model. An example is shown in Table **??** where spice.c (the second line) is one of several files that show a statistically significant difference from gimpdraw.c. In this case, its testability is less challenging, requiring, on average 777 fewer fitness evaluations.

TABLE III
Random: DO Statistical Model

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1245	164	7.62	< **0.0001**
spice.c	-777	235	-3.31	**0.0016**
gamma_cody.c	-784	372	-2.11	**0.0391**
i0.c	-1105	326	-3.38	**0.0012**
k0.c	-1074	326	-3.29	**0.0016**
pnchisq.c	-1107	613	-1.81	0.0756
pnorm.c	-618	208	-2.97	**0.0042**
ptukey.c	-952	222	-4.29	< **0.0001**
rhyper.c	-1028	443	-2.32	**0.0236**
toms708.c	-912	443	-2.06	**0.0436**
min	-136	57	-2.39	**0.0197**

Table **??** also provides an example where the stepwise elimination retains a term of dubious value [**?**]. While mean and max were eliminated, min's p-value of 0.0197 is below the technique's cutoff value of 0.05 and thus min is retained. However, its effect is minor. When describing a model, we tend to discount terms such as min that have dubious p-values. Furthermore, in the next section we consider a range of alternate models including models that initially omit such dubious explanatory variables. This evaluation enables us to assess the elimination's impact on our conclusions.

We begin the analysis by looking at the 36 model equations shown in Table **??**. Because it is a categoric variable, including File in a model equation requires a term for each level (i.e., for each file), making the resulting equations long and visually challenging to read. As a simplification, we use the term file to denote the inclusion of the nine File specific intercepts.

The 36 models exhibit several interesting patterns. Perhaps the most striking in the context of this research is the absence of min, mean, **and** max from 21 of the 36 models. The absence of influence from any of the path-length measures in over half of the models suggests (perhaps surprisingly) that squeeziness [**?**] is not related to longer dependence paths.

Of the fifteen remaining models, eight include only path-length measure with a *negative* coefficient indicating that longer paths are correlated with a *reduction* in testing cost. Two models include only measures with positive coefficients, and five include both mean and max, but with opposite signs (two of these five also include min).

A model that includes two related variables with opposite coefficient signs can indicate that the two act to counter balance each other, in which case they may be strongly correlated. We investigated the pairwise correlations between min, mean, and max in each data set and found that while min was not strongly correlated with the other two (R^2 values ranged from 0.08 to 0.31), mean and max were strongly correlated in all four data sets. The large-mean data set has the steepest slope with max $= 1.60\times$mean-1.26 with an R^2 value of 0.98. The large-median data set has the shallowest slope with max $= 1.50 \times$ mean $- 0.81$ with and R^2 value of 0.99. The two small data sets produce virtually identical models to their large counterparts. All four of these correlations are strong, as visually evident for the small-median data shown in

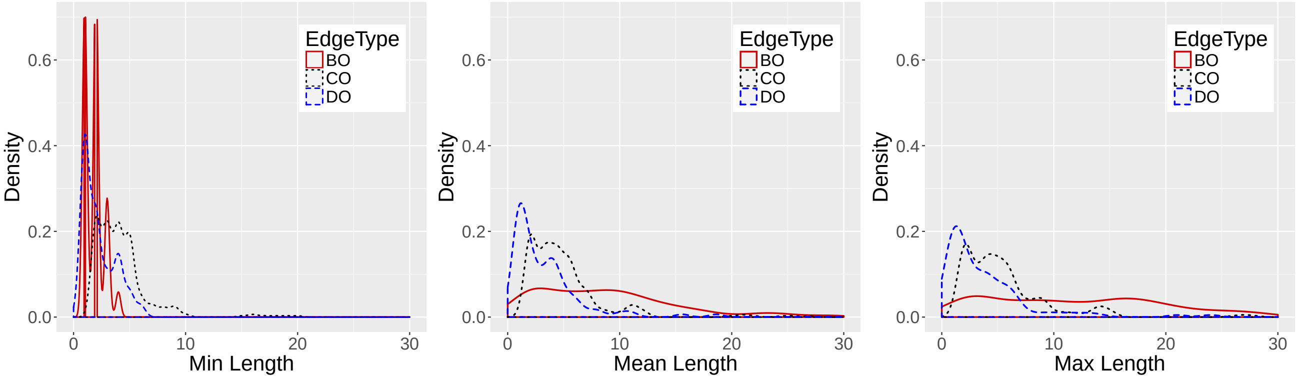

Fig. 3. Path length variation using the small-median data. All four data sets are similar.

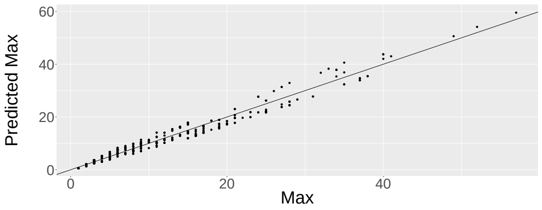

Fig. 4. Prediction of max using $1.54 \times \text{mean} - 0.97$ for the small-median data set.

Figure **??**. These strong correlations suggest the inclusion of interaction terms, which we investigate in the next section.

Because of the strong correlations, combining the mean and max terms is reasonable. For example, with the small-median data set we replace $7616 \times \text{mean}$ with $7616/1.54 \times \text{max}$ (and adjust the intercept) in GA's BO model. This replacement leads to an aggregate coefficient of $+316$ indicating that longer paths are correlated with greater testing challenge. The net effect is also positive in one other of the five models that include both mean and max, and negative in the remaining three.

In summary, over all 36 models increased path length has no effect on 21 of the models, it (surprisingly) has a negative effect on nine of the models, and a positive effect on just four. The remaining two models show a negative effect with respect to mean and max, but a positive impact with respect to min. With only 17% (6 of 36) models showing any positive correlation, it is hard to argue that longer paths are the cause of testability challenge.

We next separately consider the models for each edge type. To begin with, of the twelve DO models shown in Table **??**, five omit min, mean, and max. Six of the remaining seven include either min or max, but with a negative coefficient. In the remaining models that include both mean and max, the net effect of the two is again negative. Finally, the p-values for the length measures range from 0.016 to 0.038, indicating that none provide strong evidence. As such, for data dependence edges, longer paths are not correlated with test generation cost and even show the inverse more often than not.

We next consider the twelve CO models. Nine of these models exclude min, mean, and max. In two of the three that include path length measures, the net effect is negative. Only the large-median AVM-CO model shows a positive correlation between increased control dependence path length and increased testability cost. In addition, the two models showing a negative effect with respect to mean and max, but a positive impact with respect to min are both CO models. Looking again at the p-values, with the exception of mean's 0.034 in the large-median AVM model, all the others are reasonably strong. Compared to the DO models, the data suggests that control dependence is more important than data dependence when it comes to testability expense.

Finally, when considering paths of both edge kinds, seven models omit all three of min, mean, and max, and two show a negative correlation (the p-values of 0.041 and 0.017 indicate these influences are not strong). Of the remaining three, the large-mean AVM model includes a positive influence from min; however, its p-value of 0.0498 leads us to discount this influence. In contrast, the small-median model for GA and the large-median model for AVM include a net positive influence from mean and max both with p-values < 0.0001.

In summary, the dearth of positive correlations and the notable variation across the four data sets regarding which models do show the rare positive correlation, supports the absence of a noteworthy correlation between dependence path length and testability. Moreover, only two of the model's R^2 values exceed 0.50.

In summary for RQ3, there is an intuitive attraction to the observation that longer sequences of (dependent) statements should lead to greater "information squeeze" and thus to increased testability cost. Despite the appeal of this observation, the data suggests otherwise. Thus, in addition to the direct answer of "no" to RQ3's "Can testability be modeled using dependence path length?", this result raises some fascinating potential future work.

D. Threats to Validity

One threat to the validity of this study comes from the selection of C functions. Our results may not generalize to other programs or to code written in other languages. However, the 28 functions we selected represent a range of complexity, with 2 to 42 predicates (i.e., branching nodes) and from 13 to 334 lines of code — the latter representing a relatively large figure for a single function/method of an overall program.

A second external validity threat is that our results may not generalize to other test data generators that use different types of searches, fitness functions, or techniques to generate inputs.

TABLE IV
SIMPLIFIED MODEL EQUATIONS

Method	Edge Type	R^2	Equation				
Small Mean							
GA	BO	0.04	1062				-38 max
GA	CO	0.00	535				
GA	DO	0.00	591				
AVM	BO	0.27	7536	$+$ file	-1417 min		
AVM	CO	0.22	5246	$+$ file			
AVM	DO	0.25	6241	$+$ file			-281 max
Random	BO	0.32	1344	$+$ file			
Random	CO	0.32	1344	$+$ file			
Random	DO	0.37	1534	$+$ file			-58 max
Large Mean							
GA	BO	0.00	1029				
GA	CO	0.25	132	$+$ file	$+3949$ min	-8648 mean	$+4676$ max
GA	DO	0.06	1926				-217 max
AVM	BO	0.04	-342		$+437$ min		
AVM	CO	0.00	484				
AVM	DO	0.00	425				
Random	BO	0.00	2428				
Random	CO	0.00	2428				
Random	DO	0.31	9574	$+$ file	-1302 min		
Small Median							
GA	BO	0.39	6021	$+$ file		$+7616$ mean	-4665 max
GA	CO	0.45	9072	$+$ file	$+11036$ min	-27417 mean	$+16006$ max
GA	DO	0.21	8857	$+$ file			
AVM	BO	0.19	3704	$+$ file			
AVM	CO	0.19	3704	$+$ file			
AVM	DO	0.00	1490				
Random	BO	0.35	968	$+$ file			
Random	CO	0.35	968	$+$ file			
Random	DO	0.41	1245	$+$ file	-136 min		
Large Median							
GA	BO	0.00	2493				
GA	CO	0.00	2493				
GA	DO	0.00	2582				
AVM	BO	0.69	-3223	$+$ file		$+5729$ mean	-3362 max
AVM	CO	0.77	-8999	$+$ file	$+1909$ min	$+452$ mean	
AVM	DO	0.44	-1009	$+$ file		$+6230$ mean	-5054 max
Random	BO	0.32	9427	$+$ file			
Random	CO	0.32	9427	$+$ file			
Random	DO	0.36	10965	$+$ file			-447 max

For example, EvoSuite [?], takes a "whole test suite" approach, where the fitness function encourages the generation of tests that cover as many branches as possible. However, EvoSuite still bases its fitness function on the notion of branch distance, gradually peeling back levels of the control dependency graph during the search process. As such, we have reason to believe similar trends will be observed for example generating test suites for Java programs with EvoSuite.

A further threat arises from the tools used in our study. As we study C code, we used the publicly available IGUANA tool. It is possible that defects are present in its implementation, thereby affecting our results. However, IGUANA is a relatively mature tool, having featured in several previous studies [?], [?], [?], [?]. The same applies more or less to CodeSurfer and to the path-length counting code. In all three cases, we checked the results wherever possible for errors.

The statistical tests used are all well established and their implementations publicly available in R and thus well vetted. The linear models built by R's lm function do assume there is no autocorrelation of the residuals and that the residuals are normally distributed. An inspection of the acf plot reveals a varying degree of autocorrelation. To check its impact, bootstrapping was applied to the data. The resulting R^2 values are all similar to those found in Table **??**. Depending on the amount of autocorrelation, they range from 0–6 percentage points higher. However, they show the exact same trends as found in Table **??**. This indicates that non-normality of the residues is not impacting the interpretation of the data. Furthermore, the stability of the bootstrapping models indicates the sufficiency of each data set's size. By checking these assumptions and using well vetted techniques, we hope to mitigate threats to statistical validity; however, it is possible that more appropriate tests unknown to us might provide more appropriate evidence. We also endeavored to follow the most up-to-date information from the statistics community when interpreting the models [?].

Our IGUANA setup, subjects, experimental results and analysis are available for inspection in a replication package at *https://bitbucket.org/depchaintest/replication-package/*.

VI. DISCUSSION OF IMPLICATIONS

A. Replication Discussion

This section first discusses the implications of our replication and then of both results being negative. Despite being a conceptual replication and thus having a markedly different setup, such as the choice of explanatory variables, both studies (ours and the reference experiment) yield the same main finding: the absence of a correlation between information flow and program dependence. In addition, both experiments suggest that control dependence has the dominant effect in terms of information loss. Thus our replication reproduced all of the key results from the reference experiment.

In greater detail, based on visual graph inspection, Masri and Podgurski conclude that "the length of an information flow is not indicative of its strength." Following Voas and Miller, in our experiment we use the cost of search-based test data generation as a measure of information loss, which is the opposite of its *strength*. Substituting in the variables used in this paper, Masri and Podgurski find that "the length of a (dynamic) dependence path is not indicative of its testability." In other words, their chief finding matches ours for RQ3.

One of the more interesting differences in the two studies is our use of static dependences while Masri and Podgurski made use of dynamic dependences. One might expect dynamic dependence to more accurately mirror the flow of data during execution. On the other hand, because dynamic dependence analysis requests treating each instance of a dependence that occurs during runtime, one might expect patterns to be less pronounced. That both studies come to the same conclusion means that they mitigate the potential threat of the other. Here again, our use of a conceptual replication serves to increase external validity.

TABLE V
SPEARMAN CORRELATIONS

	Evals	min	mean	max
Evals	1.000	0.007	-0.038	-0.038
min		1.000	0.480	0.402
mean			1.000	0.990
max				1.000

Given the nature of search based algorithms, which adapt inputs to execute specific paths through a program to attain test coverage, one might expect that control dependence would dominate the cost of the search. Indeed, we find this to be the case. However, the explanation seems to run deeper than just our use of search-based test data generate techniques. For instance, in Table **??**, Random search actually gets easier with longer DO paths. Even more significant, Masri and Podgurski, who do not make use of search-based techniques, find that "flows due to data dependencies alone are stronger, on average, than flows due to control dependencies alone." Because a strong flow is the same as finding less loss, we can paraphrase their finding using our explanatory variables as "flows due to data dependencies alone show less loss, on average, than flows due to control dependencies alone." Or, as we have observed before, "control dependence brings greater information loss." Thus, both experiments support the notion that control dependence effects dominate those of data dependence.

B. Negative Result Discussion

One key difference in the two studies is our extensive use of statistical modeling rather the relying on visual graph inspection. To reinforce the validity of our modeling, we consider two possibilities. First, we examine the possibility of non-linear correlations between the variables, and, second, we go on a regression fishing expedition.

Non-linear Models. It is possible that there are strong *non-linear* correlations between Evals and the three length measures. To consider this possibility we applied Spearman's Rank Correlation. The results for the small-median data set are shown in Table **??**. From the top line of the table, it is clear that there are no strong rank correlations with Evals. The only strong correlation is between mean and max, which mirrors the result shown in Figure **??**.

Next, the charts shown in Figures **??** and **??** show evidence of skewed distributions. For example, Evals covers five orders of magnitude. Therefore, we applied a log-transformation, which is classically used to deal with skewed data. We do so independently to each explanatory variable. In short, this transformation does not improve any of the models.

Regression Fishing. Given the absence of strong correlations, we investigated the possibility that our initial statistical models or the third-party elimination software were inadequate. The investigation considers a range of simpler and more complex models. In one direction we started with fewer explanatory variables to see if elimination took the analysis down a "bad path." In the other direction, as suggested by the correlation between mean and max, we considered interaction terms between the three path-length measures. An interaction term between two variables A and B allows the effect of B to differ for different values of A.

For each data set we consider the following starting points (1) a model that includes all pairwise interactions between min, mean, and max, (2) the three models without one of the three interaction terms, (3) the three models without two of the three interaction terms, (4) the three models without interactions and without one of the three length measures, (5) the three models without interactions and without two of the three length measures, and (6) a model without any of the length measures. This yields fourteen additional models for each pairing of an edge type and a search method (126 total). These are on top of the 36 models shown in Table **??**. Additionally, we consider all these models with File, with Function, and with neither File nor Function. For example, the motivation for omitting File is that if the File specific intercepts are only marginally significant, then omitting them might cause other significant patterns to emerge. This yields 378 models for each data set, or a total of 1512 initial regression models used as input to the stepwise elimination.

Producing 1512 models and then picking "the one" that shows a desired correlation is what gives "regression fishing" its name. Here we are doing the opposite. We are pointing out how wide-spread the *absence* of a correlation is even over a large collection of possible models.

We start by considering the 504 models that initially include the explanatory variable File. Over all four data sets and nine combinations of method and edge-type, 17 of the 36 models are never better than the one presented in Table **??**. For 18 of the remaining 19, including interaction terms improves the R^2 value. The largest improvements are for models whose R^2 value is 0.00 in Table **??** (the largest of these is to $R^2 = 0.42$). For the models that did not originally degenerate, the average improvement is only 0.07. In the final model, using the non-interaction initial models for the small-mean data set Random-DO shows an improvement of 0.01 when min or max are excluded from the initial model. Such a small difference is ignorable.

We next compare the 504 models that include the explanatory variable File with the corresponding models that do not. Overall, 154 are identical, while in the other 350, omitting File leads to a lowering of the R^2 value.

Finally, the replacement of File with Function produces no consistent pattern. Considering groups of models that share a method and an edge type, replacing File with Function improves six groups, worsens eleven, and leaves nine unaffected. Random is the "big winner" with nine improvements and three worsenings. The ratio is three to two for GA and four to six for AVM, making it the "big loser." Finally, as a function of edge-type, BO has five groups up and three down, CO has five up and four down, and DO has six up and four down, with the remaining groups being unaffected.

To summarize the results of this regression fishing expedition, none of the alternatives proves a clear improvement over the models presented in Table **??** nor did the log-transformation provide any improvement. This absence of any groups with strong correlations provides further support for the negative finding that dependence path length is not correlated with test generation difficulty.

Overall, having two independent studies that are only conceptually related in terms of their structure adds to the external validity of both results. Studying the question in the context of random and search-based test data generation provides us with some key lessons, however. The primary lesson is that dependence path length does not, on its own, predict testability for any of the search algorithms that we evaluated. Instead, we need to look at the nature of these dependencies. For example, the use of boolean flags in programs is well-known to introduce plateaus in fitness landscapes when used in predicates [?], providing no guidance to search-based techniques. This is one type of problematic data dependency that can be tackled using a special type of program transformation called a testability transformation [?]. Our results therefore call for researchers to orient themselves away from coarse-grained metrics in the search for good predictors of program testability, towards fine-grained features particular to the test methods concerned. Although we concentrate on C programs and particular search algorithms, we believe our results will be more widely applicable to search-based testing of programs written in other languages (e.g., Java), since the fitness functions involved are built on similar principles of monitoring control flow through approach-level-like metrics and/or branch distance calculations.

Despite our two studies providing essentially the same conclusion, the issue may be more complex than either study is capable of uncovering. For example, Androutsopoulos et al. [?] find a clear connection between squeeziness (information loss) and failed error propagation that seems at odds with our finding and thus deserving of future investigation.

Finally, Masri and Podgurski include the following caution in their conclusion "we caution, however, that definitively confirming our results and understanding their full implications for specific applications will require substantial further study." While not its primary goal, our study my be viewed as considering their results through the lens of the test data generation. However, despite there now being two studies supporting this negative result regarding dependence path length and information loss, we echo their words of caution.

VII. Related Work

Voas and Miller [?], [?] were some of the first to study sources of poor software testability. They defined a metric, known as the *domain/range ratio* (DRR) for program specifications. DRR is the ratio of the cardinality of the domain of a subfunction to the cardinality of its range. They posited that a high DRR, and thus high "narrowing" of the input space with respect to outputs, leads to a higher probability of faults "hiding" from testing.

Clark and Hierons take an information-theoretic view of domain narrowing, measuring information loss through the paths of a program through the definition of a metric called "squeeziness" [?]. Squeeziness occurs when correct and incorrect states coincide. So-called *collisions* of correct and incorrect program states allow faults to be masked in testing.

Masri and Podgurski explored variable dependence within a program and how this relates to (an estimate of) information flow [?]. They found that while there were many cases where there is a (data) dependence, there was negligible information flow. Our findings are similar and reinforce this previous work. 'The "squashing" of a number of inputs to relatively small number of intermediate or output variables during program execution has been shown to be a problem for search-based testing. The classical and most extreme case is that of the *flag problem* [?], where an entire input domain is squashed down to a true or false value through computations resulting in an assignment to a boolean variable. Harman et al. proposed a code transformation known as a *testability transformation* [?] to remove boolean flags from programs, for greater effectiveness of search-based techniques. As discussed in this paper, nested predicates are responsible for causing control dependence paths where information about each condition is revealed only incrementally as the levels of nesting are penetrated. This can cause the search for test data to fail in extreme cases. McMinn et al. [?] evaluated a nesting-flattening testability transformation, for greater testability, noting the problem to be at its most severe when the control paths also involve data dependencies. Testability transformations also have other uses, e.g., to create pseudo oracles [?] for search-based testing.

VIII. Conclusions and Future Work

This paper explores the relationship between testability and dependence path length. The paper is a conceptual, generalization and extension replication of a negative result first presented by Masri and Podgurski [?] who observed no relationship between the length of an information flow and its strength. However, they provide only visual evidence in the from of several charts. Thus our work formalizes their informal observation and provides it with formal statistical backing. This includes over 1500 statistical models including both linear and non-linear models. Thus it adds considerable weight to the counterintuitive result of the reference study.

The replication considers a different set of real-world programs as subjects and thus improves the external validity of both studies. It also makes use of static dependencies in contrast to the reference experiments use of dynamic dependencies. Finally, both studies find that control dependence is more important than data dependence for testability.

Looking forward, we are interested in experiments that help us better understand the relation between testability, dependence, and information flow, e.g., the connection between squeeziness (information loss) and failed error propagation found by Androutsopoulos et al. [?], which seems at odds with our finding and thus deserving of future investigation.

IX. Acknowledgments

Our thanks to Mark Harman and Kiran Lakhotia for their insightful conversations.