Traceback: A Fault Localization Technique for Molecular Programs

Michael C. Gerten
Iowa State University
Ames, USA
mcgerten@iastate.edu

James I. Lathrop
Iowa State University
Ames, USA
jil@iastate.edu

Myra B. Cohen
Iowa State University
Ames, USA
mcohen@iastate.edu

ABSTRACT

Fault localization is essential to software maintenance tasks such as testing and automated program repair. Many fault localization techniques have been developed, the most common of which are spectrum-based. Most techniques have been designed for traditional programming paradigms that map passing and failing test cases to lines or branches of code, hence specialized programming paradigms which utilize different code abstractions may fail to localize well. In this paper, we study fault localization in the context of a class of programs, molecular programs. Recent research has designed automated testing and repair frameworks for these programs but has ignored the importance of fault localization. As we demonstrate, using existing spectrum-based approaches may not provide much information. Instead we propose a novel approach, Traceback, that leverages temporal trace data. In an empirical study on a set of 89 faulty program variants, we demonstrate that Traceback provides between a 32-90% improvement in localization over reaction-based mapping, a direct translation of spectrum-based localization. We see little difference in parameter tuning of Traceback when all tests, or only code-based (invariant) tests are used, however the best depth and weight parameters vary when using specification based tests, which can be either functional or metamorphic. Overall, invariant-based tests provide the best localization results (either alone or in combination with others), followed by metamorphic and then functional tests.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

fault localization, molecular programs, chemical reaction networks, software debugging

ACM Reference Format:
Michael C. Gerten, James I. Lathrop, and Myra B. Cohen. 2024. Traceback: A Fault Localization Technique for Molecular Programs. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), September 16–20, 2024, Vienna, Austria.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3650212.3652138

ISSTA '24, September 16–20, 2024, Vienna, Austria

ACM ISBN 979-8-4007-0612-7/24/09
https://doi.org/10.1145/3650212.3652138

1 INTRODUCTION

Fault localization is a challenging yet essential task used continuously during the software evolution life cycle. Advanced testing techniques improve our ability to efficiently find and report faults. However, significant developer time may still be needed to determine their causes. While automated repair techniques aim to help the developer in this process by suggesting correct patches, good fault localization can be a major factor in their success [37, 38].

Designing efficient and effective automated fault localization techniques has been and still is an active area of research[1, 2, 5, 20, 23, 26, 30, 36, 50–52, 67, 69, 70, 73, 76]. One of the most common approaches uses spectrum-based fault localization (SBFL) techniques which assign a suspiciousness score to elements of a program (usually program statements) based on how often they are executed when running both passing and failing test cases. There are slight differences in the suspiciousness scores based on which metric (e.g. Tarantula[26], DStar [66], or Ochiai [2]); however, they all work under the same assumption. Elements executed more often by passing test cases are less likely to indicate where the fault lies than those elements executed more often in failing test cases. We almost never see a clear division (i.e. failing tests will cover correct elements and passing tests will cover failing elements). Hence we rank elements that are the most suspicious so developers can focus on those elements early in the debugging process.

A key strength is that the idea of a *program element* used to calculate these spectrum-based scores generalizes and can be applied to different types of elements (statements, model components, program states); therefore, SBFL techniques have been extended to many types of problems such as localization for Prolog programs [57], Alloy models [27, 61], and Simulink models [35, 36].

Armed with this knowledge, we turned to SBFL techniques as the most obvious approach to use while working on program repair for a non-traditional programming paradigm, molecular programming [46]. We attempted to naively apply SBFL to chemical reaction networks (one type of molecular program), but quickly realized it was not providing any benefit; in fact the localization was poor. Hence, we took a step back to examine this problem further.

Molecular programs model the interaction of molecules to algorithmically (or programmatically) manipulate matter at the nanoscale level and have been used to control nanodevices from molecular robots to targeted drug delivery [3, 11, 22]. This emerging programming paradigm is being leveraged more frequently than it was perhaps 10 years ago[49, 68, 72], and we expect it will eventually become mainstream. Molecular programs are often represented by chemical reaction networks (CRNs), an abstraction of the standard chemical equations found in most chemistry textbooks[14]. While these programs can be physically constructed and executed (e.g. in a

beaker) in practice, most of the programming and debugging occurs in simulation environments (such as in Matlab) where the developer can easily run and rerun programs until they have confidence the program is correct. Due to the cost of physical experiments, simulation in environments like MatLab are used to design, test and debug CRNs. Debugging involves watching variable changes across the simulation time.

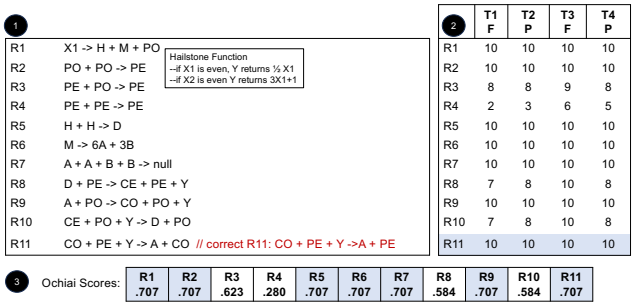

Figure 1: Example faulty molecular program ①, and the results of reaction coverage for 4 test cases ② and resulting Ochiai scores for each reaction ③.

In recent work, we have built a testing framework for finding faults in these simulated programs [17, 18], and have developed a prototype for automated program repair of CRNs[46]. While we assumed SBFL would work for localization based on observations that different elements of the programs (species and reactions) are covered by different test cases in some of our experiments [18], we were wrong.

We began by creating a mapping that directly uses one of the SBFL approaches for CRN programs, but we quickly realized this type of program could not easily benefit from SBFL. Given that Simulink models have some of the same characteristics as CRNs, we were surprised. We realized after our initial attempt that the use of SBFL in Simulink[36] relies on a deterministic, differential equation-based model, and our models are stochastic, hence the gap in the quality of Simulink localization and our results.

Figure 1 demonstrates some observations we made during this process. We show a chemical reaction network program ① for a function that computes $Y = \frac{X1}{2}$ when $X1$ is even and $Y = 3 \times X1 + 1$ when $X1$ is odd. This is one of our study subjects. We will describe the representation of CRN programs in the next section, but for now, the important thing to notice is that there are 11 *reactions* (R1-R11) and 11 *species* (or variables) across the reactions. The program has a fault in the last reaction (species CO should be species PE). Even for a seasoned CRN developer, this fault may be hard to spot.

A program simulation for a CRN starts with a *concentration*, or number of molecules in the input species ($X1$). Reactions fire in a stochastic distributed manner until the system stabilizes, and the output result can be found in species Y. Since this is a distributed system, the reactions can fire in different orders but will eventually reach the same result (if the program is correct). It is possible that a test harness checks the output too soon (false negative), or in a faulty CRN, it never stabilizes but instead cycles between the correct (false positive) and incorrect answer. Hence, when testing these programs, some tests may appear flaky even on the correct program[17] unless they are given enough time to settle. To handle this issue, we run all simulations a number of times and if any of the simulations fail, we consider it a failed test.

② shows the number of times each reaction is executed for four different test inputs (T1-T4). Two tests (T2 and T4) pass (one even and one odd input), and the other two fail (one even and one odd). We ran each ten times on this program and recorded the number of runs where an element (in this case, the reaction) is covered. Reaction 11 (the faulty reaction) is executed by all four test cases in all ten simulations.

Hence, we are left with no information indicating reaction 11 is faulty. In fact, reactions 1,2,5,6,7 and 9 and 11 have the exact same coverage. Only four reactions (3,4,8 and 10) provide any differentiation between tests and even then there is disagreement between passing and failing tests. ③ shows Ochiai SBFL suspiciousness scores [1] over these four tests. A higher value indicates higher suspiciousness (e.g. that reaction is more likely to be involved in the fault). There are seven reactions (of 11) that all have the same score, providing little guidance. However, clearly there is some information provided which is guiding the programmer away from at least 4 of these reactions. Hence, we ask if it is possible to extract fault localization information from this type of system.

In this paper, we examine this problem in more depth. We evaluate existing SBFL techniques using different types of elements in CRNs (species and reactions). We then propose a new technique, *Traceback*, which leverages simulation traces augmented with reaction instrumentation to identify inflection points in the trace where the failure manifests. Traceback (1) identifies a time of potential failure in the trace, (2) identifies the reaction that was closest to the time of the fault (a suspicious reaction) and then (3) optionally traces backward in time to locate other potential failure reactions.

We evaluate our work in an empirical study on six benchmark CRNs, each with multiple faulty versions (89 faulty programs in total). We find that Traceback has the best overall localization results, but that the different types of tests used can also play a role.

The contributions of this work are:

(1) A new fault localization technique (Traceback) that uses time-based traces to calculate suspiciousness.
(2) A study on a set of 89 faulty CRN programs that:
   - compares Traceback with the state-of-the-art SBFL;
   - evaluates the impact of the Traceback parameters; and
   - examines the impact of different test types.

The rest of this paper is laid out as follows. In the next section, we present some background and related work on CRNs and fault localization. We then present all of our fault localization techniques and Traceback in Section 3. In Sections 4 – 5 we present the study and results. We then provide some discussion (Section 5.4) and end with conclusions and future work.

2 BACKGROUND

In this section, we present some background on testing CRNs and on fault localization.

Reaction 1: $X + X + X \rightarrow V$
Reaction 2: $Y + Y + Y \rightarrow V$
Reaction 3: $X + Y \rightarrow V$
Reaction 4: $V + X \rightarrow X$
Reaction 5: $V + Y \rightarrow Y$

Reaction 6: $X \rightarrow V$

Figure 2: Reactions 1 through 5 represent a CRN, `mod`, that computes if the input species X and Y are congruent modulo 3. The output variable V is non zero if X is congruent to Y modulo 3, otherwise V is 0. Reaction 6 creates a faulty (or mutant) CRN.

2.1 Chemical Reaction Networks

The Chemical Reaction Network Model (CRN) has been used for over 50 years to describe natural phenomena relating to the interaction of molecules in a well-mixed solution [12–14, 28, 55]. There are two main variants of the CRN model in use, deterministic and stochastic, modeled by differential equations and Markov processes, respectively [4, 14]. In this paper, we use the stochastic CRN model, where the semantics describe the system in terms of molecule counts governed by a Markov process. These systems form large parallel distributed systems that can have complex molecular interactions. Using DNA nanotechnology, theoretically, any CRN can be translated to a strand displacement system (even those that do not model natural systems) and realized in vitro. In fact, the CRN model is Turing complete, meaning that any algorithm can be implemented in the CRN model [15, 54]. Hence, the CRN model is a programming language, and software engineering techniques are being utilized to verify and test these systems [40, 42, 44].

We present an example CRN, `mod`, (first five reactions of Figure 2) that computes a function to determine if two numbers are congruent modulo 3, i.e., $X \equiv Y \bmod 3$. In this example, there are three *species* (types of molecules) depicted, X, Y, and V, and five reactions depicted, with *reactants* left of the arrow, and *products* right of the arrow. We have two input species, X and Y which are initialized by supplying values that are represented by *molecular counts* or concentrations. The single output species V contains the answer when the computation is finished, i.e, when no more reactions can fire. When complete, the output species Y contains 0 molecules if X is not congruent to Y modulo 3, and it contains at least one molecule if it is.

We show a *trace* of simulation for this program using an input of 10 and 12 in Table 1. At time 0, (step 1), input species X and Y contain initial values of 10 and 12, respectively. *Reactions* are rules that transform *reactants* into *products*. For example, when *Reaction 1 fires*, three molecules of type X are removed and a single molecule V is produced (step 4 in the trace). As time progresses, reactions fire nondeterministically and stochastically governed by a Markov process. This trace is just one possible ordering of reactions given this input. Each simulation of the CRN can (and likely will) yield a different ordering of reactions. If the CRN always produces the same output given an input independent of the reaction order, the CRN is *stable*. However, not all CRNs are stable and for these we need to use probabilistic testing approaches. Furthermore, the time at which reactions fire is stochastic, resulting in variation in the time stable CRNs complete. This can introduce what appear to be flaky tests if the simulation is terminated too early[7, 17]

Table 1: Example trace of $V = X \equiv Y \bmod 3$. Inputs are X=10 and Y=12. Output is V=0 indicating these are not congruent.

Step	time	Reaction	X	Y	V
1.	0.0000	Init	10	12	0
2.	0.0007	Reaction 2	10	9	1
3.	0.0014	Reaction 3	9	8	2
4.	0.0018	Reaction 1	6	8	3
5.	0.0050	Reaction 2	6	5	4
6.	0.0080	Reaction 1	3	5	5
7.	0.0110	Reaction 4	3	5	4
8.	0.0193	Reaction 5	3	5	3
9.	0.0204	Reaction 2	3	2	4
10.	0.0300	Reaction 4	3	2	3
11.	0.0490	Reaction 4	3	2	2
12.	0.0933	Reaction 1	0	2	3
13.	0.4050	Reaction 5	0	2	2
14.	0.5445	Reaction 5	0	2	1
15.	1.0164	Reaction 5	0	2	0

2.2 Testing CRNs

We have been working on build a testing framework for CRNs.[17, 18] as well as on applying automated repair approaches[46]. In this line of work, we have created a framework built on top of the Matlab simulator. We utilize a property-driven test approach. *Abstract tests* are written using a temporal logic and these are then concretized over a set of values. The logic uses operators such as Future Globally (i.e. at all points in the future that property holds). In practice these properties are evaluated at a single point in time and the future global operator is inferred (without guarantees). In ChemTest we use the test specification language[47] to select values for the abstract tests and partition the input space based on equivalence classes. In this work we do not write tests, but utilize existing ones from ChemTest artifacts. A test case in ChemTest evaluates the program behavior by reading the outputs of a trace (all species and their values will be listed at each time step as shown in Table 1). The test chooses a time for evaluation, usually set at some number (N simulation seconds) which is internal to the simulator and differs from clock time. N is expected to be sufficient for the program to stabilize on a result. If the result matches the test passes. If it does not match it fails.

ChemTest uses different types of test cases. Functional and Metamorphic test cases are based on the program specifications and manually created. Given the difficulty of writing these, in follow-on work, [18], we designed Invariant tests which are structural and are extracted from the static model of a correct CRN. These are useful during regression testing or when we have an implementation, but no formal specifications.

If we return to our example trace (Table 1), and we have an abstract test for the mod function which requires the initial value of

the species X and Y do not satisfy $X \equiv Y mod\ 3$, we can concretize this with the input values, 10 and 12. Then we would expect V to be zero at the end. In this simulation, at step 15 (the end of the trace), the concrete test would pass. At step 14, however, it would fail (we have 1 instead of 0).

We may oscillate before converging; it is possible we have a 0 at some point (see Figure 4) and then return to a positive value before the reactions are stable. In testing we aim to give enough time for stabilization to avoid this type of time-dependent issue. In our localization algorithms, we utilize this oscillation, however, since incorrect CRNs often fail to converge and we try to identify points where the test changes from passing to failing as possible suspiciousness locations.

2.3 Spectrum-Based Fault Localization

Spectrum-based fault localization (SBFL) is one of the most popular approaches [1, 2, 20, 23, 26, 30, 51, 52, 67, 70, 73] used in traditional programs. It works by running the target program against a test suite partitioned by passing and failing tests. The target for localization (e.g. a statement, branch, method) is called an element. The use of SBFL is broad, being applied to types of program models such as Alloy [27] and Simulink[36].

SBFL tracks which elements were executed in both the passing and failing buckets and then assigns each element a suspiciousness score based on the number of times it appeared in each bucket. If the element was executed during the test case in a particular bucket, the test result is added to the element's count. If the element was not executed, it would be included as part of the non-element count. We collect four counts: element failing (E_f), element passing (E_p), non-element failing (nE_f), and non-element passing (nE_p). We then sum across all tests, getting values for each of the groups. These values are passed to the spectrum-based fault localization metric. Many available metrics utilize these counts; thirty-one are listed in a recent survey [67]. Some of the most popular ones are Tarantula [26], DStar [66], and Ochiai[2, 48]. We performed a small pre-study and determined the exact SBFL metric used did not impact our results, hence we focus on one, Ochiai[2, 48] and leave a deeper dive into the subtleties of these metrics in CRNs as future work.

Ochiai works by applying Equation 1 to the counts from the full test suite where E_f, E_p, etc. are the elements involved in failing, passing, etc.

$$Ochiai(j) = \left(\frac{E_f(j)}{\sqrt{(E_f(j) + nE_f(j)) * (E_f(j) + E_p(j))}} \right) \quad (1)$$

The metric is evaluated for each element j in the program and a list in descending order is created, with values from 1 to 0. The higher the value, the more suspicious the element is. In a perfect case, the fault is found at the first element returned. If there are no failures, Ochiai returns no localization.

3 FAULT LOCALIZATION TECHNIQUES

We utilize three different techniques for performing fault localization in this paper. The first two map elements of the CRN to Ochiai. These are the state of the art. We then present our new approach called Traceback.

Figure 3 shows the overall process and details how the different techniques vary. We split localization into three steps. The first step (①) is the program simulation which collects traces for all tests in the test suite. There are two ways to perform the simulation. The first is based on ChemTest [17]. It runs the test cases and collects trace information which consists of the species values at each point in time. We call this No Reaction Counting (NRC), and it is the simplest approach. In follow-on work [18], we added instrumentation to the simulations, called Reaction Counting, (or RC). We used this to evaluate inequality invariants. We implemented both approaches.

Step ② is the evaluation step where the oracles for tests are applied to traces at different points in time. For testing, this is usually performed one time at the end of the traces (1x). However, as we will see, we may want to do this at every step (Tx), where T is the total number of steps in the trace.

The last step (③) is localization itself. Our first and simplest approach, Species uses the native traces and is evaluated at the end (1x) Our second technique, Reaction, requires reaction counting (RC) but only evaluates once at the end of the trace (1x). Our new technique, Traceback, uses both reaction counting (RC) and evaluation at each step in the trace (Tx). This is our most expensive localization technique.

3.1 Species-based Fault Localization

Our first technique uses species of the CRN as the element measured in Ochiai. For each test we examine all runs of that test case, and if any fail, we mark them as a failing test case. We partition failing and passing tests and then count the elements. For a test T, a species U is *executed* by test T if it is used in the evaluation of T. For example, looking at the test T = "if input X is congruent mod Y, then output $V > 0$" used to test the mod function in [18], species V is executed by the test T. Note that species involved in the precondition (in this case it is "X and Y") does not count as being executed.

The species counts are summed over all inputs and all tests to determine a total failure count and passing count for each species. In the last step, the species counts are mapped to reactions by a *Species Mapping Function*, which sums the counts for all species involved in each reaction. Given a CRN $N = (S, R)$, a species $U \in S$ is *involved* with reaction $\rho \in R$ if the species U is a noncatalytic species in the reaction. For example, only V is involved in $V + X \rightarrow X$, since X is a catalyst (both a reactant and product). With this mapping, we can use Ochiai where E_f is the failing evaluation count, and E_p is the passing evaluation count.

3.2 Reaction-based Fault Localization

This method determines if a reaction is fired during any failed trace of a test case and any passed trace of a test case. Since we have some nondeterminism it is possible that the reaction appears in both passing and failing buckets for a single test case (in different traces). The Ochiai program elements are reactions, hence this requires the reaction counting scaffolding to be enabled during simulation. If the test failed, a failed reaction count is incremented for each reaction; likewise, if the test passed, a similar pass reaction count is incremented. The passing and failing (E_f and E_p) are sums over all of the failing/passing traces and we feed this directly into Ochiai.

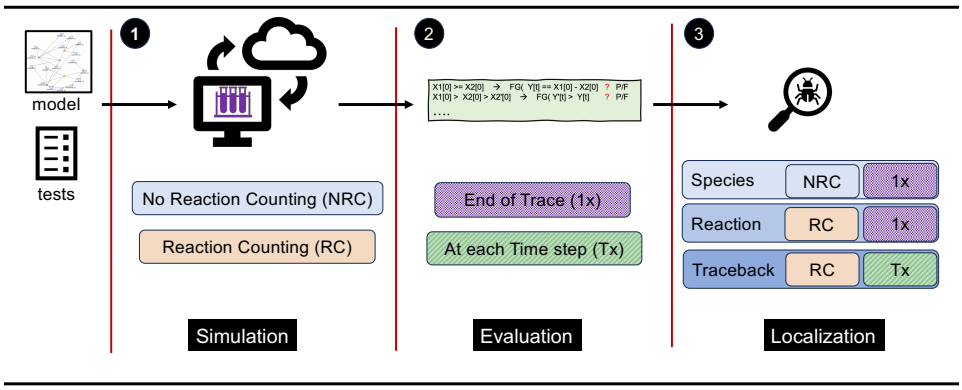

Figure 3: Overview of Localization Techniques and Process. Step 1 is the simulation technique, Step 2, the evaluation time and Step 3 is the localization approach.

3.3 Traceback Fault Localization

Our intuition for Traceback is that a CRN will stabilize over time (Gerten et al.[17]). We showed if a test is evaluated too soon, it will give either a false positive or false negative result. CRNs also compute numerically, meaning they slowly converge toward a correct numerical result. Furthermore, in Gerten et al.[18], we showed that different invariant tests covered different reactions and species. Together, this may mean we can look for points of change (or inflection) in a trace to gain information about which reactions occurred before that inflection point. This has some similarity with the work of Beszédes et al. where they enhance statistical fault localization with function call information[8]. It also is similar to work on using partial invariants for dynamic fault localization [5, 50, 69]. However, those studies use traditional program constructs and slicing techniques (which are code-based) or focus on changes in runtimes[69]. We use dynamic traces which represent the order of reaction firings, and we use other types of tests besides invariants.

Traceback works on individual traces of a test case. Traceback first identifies a single suspicious reaction in the trace. This is the last reaction that fired when inflection occurs, and then additional reactions are identified by examining prior reactions in the trace (*traceback*) to a depth k ($k \geq 0$). It uses a weight (parameter w) to assign importance to the k reactions. Since the algorithm produces localization data for each trace generated by the test suite, the localization data must be summarized in order to give a single suspiciousness value for each reaction. This summarization of the data is accomplished by averaging, summing, or maximizing results first at the abstract test case level (this gives equal weight in the test suite to each abstract test despite different numbers of inputs) and then at the concrete and evaluation levels.

We demonstrate Traceback through examples and begin by describing how to find an initial suspect reaction given a trace of a faulty program derived from our example program in our motivating example (Figure 1). This program includes Reaction 6: X → V. We use a concrete test with inputs X and Y as 10 and 12, respectively, and an abstract test (or property) $V = 0$ if X is not congruent to Y mod 3. Figure 4 shows the trace of V output values over time

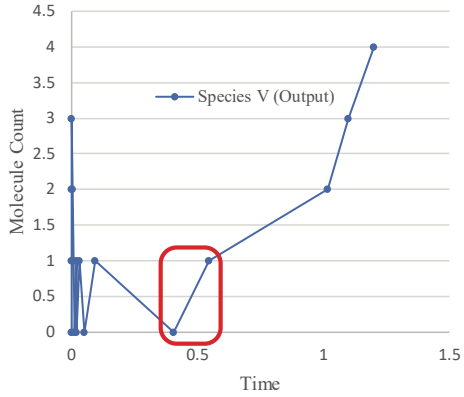

Figure 4: Trace of example mutant with inputs X=10 and Y=12, showing output V with respect to a passing value for property if $X \not\equiv V \bmod 3$ then $V = 0$.

during simulation. The correct output value for this function is $V = 0$; however, the CRN failed since the final value is $V = 4$.

To locate the initial suspicious reaction, we evaluate the property at the end of the simulation when $V = 4$ and continue evaluating the property on V until the property does not fail. The red box in Figure 4 depicts where this happens. While this may be a false positive, our assumption is that it is more suspicious. This occurs when the value of $V = 1$ (failing) transitions to the value of $V = 0$ (passing). Hence we mark the initial reaction that caused the species V to change from 0 to 1 as the initial suspicious reaction. Note that some properties contain equality relationships. In this case, the property is broken into two properties: one that tests a $\leq$ relationship and one that tests a $\geq$ relationship.

We now define the data structures used in our algorithm for locating additional suspicious reactions in a trace once an initial suspicious reaction is found in the trace. A *trace* of a CRN N is a finite execution sequence from some initial state $\mathbf{x}_0$ with parameter m that specifies the maximum length of the trace as follows.

$$\text{trace}_m^N = (\mathbf{x}_0, \mathbf{x}_1, \ldots \mathbf{x}_{<m}),$$

where $\mathbf{x}_0, \mathbf{x}_1, \ldots$ is the sequence of species states (molecular counts) at increasing time steps. We also introduce one additional sequence useful for describing our localization algorithm.

$$\text{Rtrace}_m^N = (\perp, \rho_1, \rho_2, \ldots)$$

is the sequence of reactions fired associated with a specific trace. Note that the initial state has no fired reaction and is not defined. For a trace T and associated Rtrace, the i^{th} element (step) of the sequence is denoted using square brackets. For example, for a trace T, $T[i]$ denotes the state of the CRN after the i^{th} reaction fires.

We describe a useful data structure based on a trace of the CRN, the *reaction dependency graph*, a key component of our technique. For a trace T, the reaction dependency graph for the i^{th} reaction of the trace is a linear graph of reactions in reverse order of reactions that fired previously to the i^{th} reaction defined by Algorithm 1.

Algorithm 1 Reaction Dependency Sequence

```
function ReactionDependency(i)
    g ← Rtrace_m^N[x_i]
    step ← i − 1
    (r, p, k) ← Rtrace_m^N[x_i)]
    dspecies ← {S | r(S) − p(S) ≠ 0}
    while step > 0 do
        (r, p, k) ← Rtrace_m^N[x_step]
        if {S | r(S) − p(S) ≠ 0} ∩ dspecies ≠ ∅ then
            dspecies ← dspecies ∪ {S | r(S) − p(S) ≠ 0
            g ← g · Rtrace_m^N[x_step]
        end if
        step ← step − 1
    end while
    return linear graph g
end function
```

Intuitively, the algorithm determines prior reactions that have fired that could have affected a specified reaction of interest. This is computed by scanning the CRN trace from the specified reaction backward through time, looking for reactions that may affect the reactants of the specified reaction. This creates a *dependency graph*. We show this in Figure 5 (left). A *proper dependency graph*, Figure 5 (right) is now computed in which a reaction may only appear once in the graph traversal when it is encountered for the first time in the backward scanning.

Traceback uses the proper dependency graph to assign a weight to each reaction in the CRN by utilizing tests to find a possible reaction that initiated the error. While tests can identify a single reaction per trace as suspicious, the proper dependency graph can be used to spread that suspiciousness to other reactions. The idea is that the identified faulty reaction may have been caused by the firing of a previous reaction in the trace, which propagated forward in time to the specified reaction. We could have also used the dependency graph instead to spread the suspiciousness to other reactions. However, when a series of duplicate reactions occur in succession, they may hide a latent reaction in the past that was responsible for the error. Such information might add to the spectral data and may be worth investigating in the future.

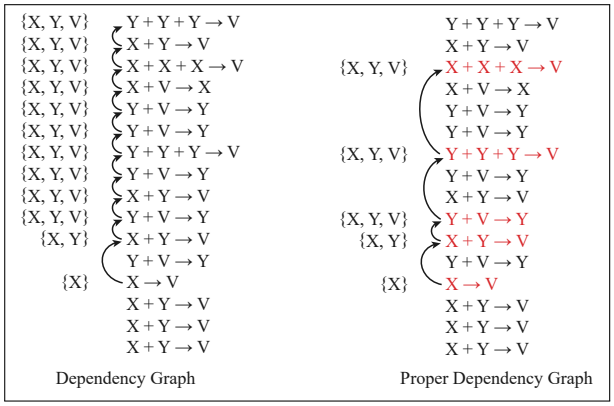

Figure 5: Dependency Graph and Proper Dependency Graph for a CRN trace.

There are many strategies for adding weights to additional reactions in the trace based on the proper dependency graph. Two methods, which we combine, are described below.

(1) The suspicious reaction is weighted 1, and subsequent reactions in the graph are weighted by some scheme (in this work we use an exponential decay) from the start node of the graph. For example, each subsequent reaction in the dependency graph traversal is 0.5 the *weight* of the previous reaction. If the weighting factor is 1, all reactions are equally weighted with a value of 1. If the weighting factor is 0, then no spreading is performed, and only the original error reaction is given any weight.
(2) Only the first k reactions (depth) are used from the dependency graph traversal for assigning suspiciousness.

If the dependency graph is not weighted as in item 1, $w = 0$, the localization results may not be diverse and yields a single suspicious reaction, and this may not identify the fault. We examine the impact of k and w in RQ2 (and RQ3). We see that for some types of test cases (but not all) their values matter.

Table 2: Suspiciousness values using the Traceback method of a set of traces for a single input for three summarization strategies, sum, maximum, and average.

		Sum	Max	Average
R1	X + X + X → V	14.1758	1.0	0.22256
R2	Y + Y + Y → V	10.8984	0.75	0.11162
R3	X + Y → V	14.2032	0.75	0.25225
R4	V + X → X	19.3593	0.75	0.24873
R5	V + Y → Y	13.1133	0.75	0.24873
R6	X → V	32.8125	1.0	0.80625

We now describe how traces are combined over abstract and concrete tests. First, as in [17], for each input, we simulate the CRN to produce N traces. These traces may yield different localizations since the reaction order will likely differ between simulations. There are many ways to combine the localization data. We explore three methods: sum, average, and maximum. These methods combine

all trace localization values with the function, giving one set of values for the set of traces. We can use the same summarization techniques on the concrete tests. Similarly, we can combine the results of several concrete tests partitioned by abstract tests to yield a final localization value for each reaction.

Table 2 shows the suspiciousness scores using different strategies. A higher score indicates the reaction is more likely to be faulty. In all cases, R6 appears to be in the top two reactions with respect to suspicious.

4 EMPIRICAL STUDY

We conducted an empirical study to evaluate the quality of the different fault localization techniques. Artifacts can be found on our supplementary website.[1]

We ask the following research questions.

RQ1. How effective is Traceback localization compared to the state of the art?
RQ2. How does tuning k and w impact Traceback?
RQ3. What is the impact of test type on localization?

4.1 Study Subjects

We use six benchmark CRN subjects covering a variety of behaviors with existing test cases and faulty mutants from prior work [17, 18]. These were archived for the community to use for replicability [19]. They have functional (specification-based, metamorphic) test cases which were manually written using a specialized temporal logic and instantiated (they are abstract tests) using concrete inputs modeled using the test specification language [47]. There are also invariant test cases which were automatically derived [18]to cover the structure of the programs. Each program has between 20 and 25 randomly created mutants (faulty) programs for evaluating fault detection. The mutants each consist of a single change to the original program, i.e. adding, removing or changing a species and adding removing or changing a reaction. While there is no formal set of benchmarks for CRNs, the data set we have used represents an aggregation of multiple papers on this topic and consists of different types of programs and test cases.

We dropped subjects with two or fewer reactions since localization would be trivial. We also kept only one version of the Hailstone program which implements the same function in different ways. We chose the first one (the one which was used in ChemTest (H1)) [17]. We removed the Molecular Watchdog Timer program since its inputs use a wave signal and were not straightforward to integrate into our testing process (we have rebuilt our test harness for this work). Last, we chose only one of the subjects with invariant only test cases as a representative subject. We selected the *E. Coli* Glucose Pathway. We describe each one next.

(1) **Approximate Majority (AM)**. Probabilistically computes which species $X1$ $X2$ has an initial majority by converting the smaller species into the majority. The probability of correctness depends on the difference between $X1$ and $X2$.
(2) **Modolus 3 (Mod)**. Computes if two input species X and Y are congruent modulo three. If $X \equiv Y mod\ 3$, then output species V is greater than 0.
(3) **Maximum (Max)**. Computes the maximum of two input species X and Y, and outputs the result in species Z.
(4) **At least 1 (AL1)**. Determines if there is at least one molecule in both of the species $A1$ and $A2$ and if true, the output species Y contains at least one molecule and otherwise contains no molecules.
(5) **Hailstone 1 (H1)**. Computes one iteration of the Collatz conjecture which is the function $f(n) = n/2$ if n is even, and $3n+1$ if n is odd. The CRN computes the function with input species $X1$ and output species Y.
(6) ***Escherichia coli* Glucose Pathway (ECG)**. Model of *E. coli* glycolysis pathways was extracted from the *E. coli* metabolic network. The species in the media of the biological model define the input species.

Table 3: Number of program mutants (Num Mut) species (Spec) and reactions in the (correct) CRN as well as average across all mutants, followed by the size of the test set: All, Functional(Func), Metamorphic(Meta), and Invariant(Invar). The test suite sizes are concrete test counts.

	Num		Reaction		Test Suite Size			
Subj	Mut	Spec	Base	Avg.	All	Func	Meta	Invar
AM	12	3	4	4.6	558	279	-	279
Mod	20	3	5	5.6	188	50	-	138
Max	19	5	3	3.4	390	38	122	230
AL1	17	7	5	5.6	471	196	51	224
H1	16	11	11	11.8	88	8	25	55
ECG	5	23	12	12.6	46	-	-	46

Table 3 summarizes key aspects of the CRNs in our study. First it shows the number of mutants. We only kept mutants with faults found by all runs of our testing step (#2 evaluation step in Figure 3). In total, we keep 89 of 138 faults from the benchmark for localization. Next is the number of species, followed by the number of reactions in the original program and the average number in the mutants (we can have at most 1 additional reaction beyond the correct program based on Gerten et al.[18]). Next are the total number of concrete test cases per subject (All) followed by a breakdown of functional (Func), metamorphic (Meta) and invariant (Invar) tests. All tests were obtained from the companion website for Gerten et al. [19].

4.2 Independent and Dependent Variables

Our independent variables for RQ1 are the three localization techniques, Species (SP), Reaction (R) and Traceback (TB). In RQ2 we vary the parameters k and w of the Traceback algorithm. In RQ2 and RQ3 we include the type of test case as another independent variable, Functional, Metamorphic, Invariant and All. For the depth of Traceback we have 6 values of k, {0, 1, 2, 3, 4, 5}, for w we there are 7 values, {0,0.1,0.25,0.5,0.75,0.9,1}.

For the dependent variables, we consider both effectiveness and efficiency. For efficiency, we measure the runtime of the different algorithms. For effectiveness, we use several metrics described next.

4.2.1 Metrics. We measure the effectiveness by ranking each reaction in a subject by its suspiciousness score with the highest suspiciousness score assigned a rank of 1. This is a common method for relating the suspiciousness scores to different program elements

[1] https://doi.org/10.5281/zenodo.10900920

in localization[48]. For reactions with the same score, all reactions are given the worst rank. For example, if the top three reactions all have a suspiciousness score of 0.43, then all three reactions would receive a rank of 3 rather than 1.

Some authors use the relative position of a rank within a program (e.g. 5th position of 10 elements is .50) to evaluate localization, but given that we have differing numbers of reactions in our programs, and the total numbers are relatively small (compared with a program of 1,000+ statements), we will see a large variation between scores if we try to compare between subjects with differing numbers of reactions. Furthermore, a rank of 1 should always be the same since it is the best we can do. Hence, we normalize the localization ranking metric with $z = (x - 1)/(r - 1)$, where z is the normalized value, x is the input rank to normalize, and r is the number of reactions. This gives us a value between 0 and 1, with 0 indicating a rank of 1 (perfect localization) and 1 indicating the worst possible rank (no information). For example, if we have a rank of 2 and 10 reactions, this gives us $1/9 = 0.11$, and 2 out of 3 reactions gives us a score of .50. Now, if we have a rank of 1, both the 10 reaction and the 3 reactions ranks will be zero.

4.3 Method

We follow the process in Figure 3. For simulation we run each test case 10 times (as in Gerten et al.)[18]. We use a cluster of Intel Xeon Gold 6244 processors, with 366 gigabytes in a 2-core environment. Since we need instrumentation for reaction counting we cannot use the existing traces from Gerten et al., but re-run all of the simulations and evaluations on their models (with their test cases) using Matlab R2022b[21]. We run each of our experiments 5 times and allow 8 hours for each subject to complete simulation and another 8 hours to complete evaluation. At the end of this process we selected those subjects where we found the fault in all five runs (some runs timed out during at least one of the 5 runs). We then complete the final step (localization) on all of the faulty subjects. This gives us 5 localization results for each mutant of each subject. We use k=5 and w=0.75 for RQ1. We experiment with these in RQ2 and RQ3. We use max for the evaluation summarization and average for the other two stages of summarization in all experiments. These were chosen heuristically in a pre-study.

4.4 Threats to Validity

With respect to generality, we used only 6 CRN subjects, however, these have been used in prior work and represent a range of types of CRN behavior and sizes and total 89 individual programs. With respect to internal validity we acknowledge that our programs may have faults. We did not find all of the faults from Gerten et al. in this work, because we required all 5 runs to fail to be eligible for localization. We did validate however, that all faults we found were also found by Gerten et al., confirming our fault detection is working correctly. We have also carefully checked our data and have unit tests for our programs. We also provide our data on the artifacts website for others to use and re-validate. With respect to construct validity, we could have used other metrics, however some metrics, such as rank and time are standard, and we believe our normalization metric makes sense for our research questions.

5 RESULTS

In this section, we present results from three research questions.

5.1 RQ1: Localization Effectiveness

To evaluate the effectiveness of our localization techniques, we examine Table 4. The top portion shows ranks by subject and the bottom half shows normalized ranks. Species and Reaction data represent the state-of-the-art SBFL applied to CRNs. Traceback identifies the top one or two reactions in almost all subjects. For instance AM and ECG always produce a rank of 1. It has an average of 0.10 overall (when normalized) where 0 would be a rank of 1 every time. Species localization on the other hand has an overall normalized rank close to 1.0 (0.92) suggesting it provides almost no localization information. Reaction localization sits in the middle. It performs slightly better than Species (overall of 0.66) but worse than Traceback. Figure 6 shows a boxplot of the normalized rank by technique. Reaction localization only provides marginal information, while Traceback appears to include the faulty reaction ranked highly, and often as the first element (normalized rank of 0.0).

Table 4: The results of the three methods of fault localization. The top is the average rank of the faulty reaction for each subject. The bottom half presents the normalized ranks normalized based on the size of the subject.

Ranks						
	Species		Reaction		Traceback	
	Avg.	Std.	Avg.	Std.	Avg.	Std.
AM	4.58	0.50	3.00	1.43	**1.00**	0.00
Mod	5.60	0.49	3.39	1.92	**1.04**	0.40
Max	3.47	0.50	3.00	0.98	**1.37**	0.74
AL1	4.12	1.79	3.47	1.66	**2.35**	1.69
H1	11.78	0.50	10.68	2.71	**1.08**	0.27
ECG	8.00	3.42	4.40	1.66	**1.00**	0.00
Overall	5.97	3.16	4.64	3.37	**1.36**	0.98
Normalized Ranks						
	Species		Reaction		Traceback	
	Avg.	Std.	Avg.	Std.	Avg.	Std
AM	1.00	0.00	0.58	0.43	**0.00**	0.00
Mod	1.00	0.00	0.52	0.41	**0.01**	0.10
Max	1.00	0.00	0.84	0.37	**0.16**	0.33
AL1	0.69	0.39	0.54	0.37	**0.30**	0.38
H1	1.00	0.02	0.90	0.25	**0.01**	0.03
ECG	0.59	0.27	0.29	0.13	**0.00**	0.00
Overall	0.92	0.23	0.66	0.40	**0.10**	0.26

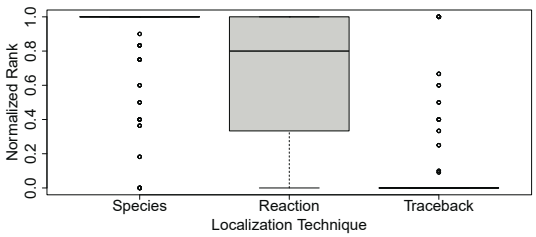

Figure 6: Normalized ranks for each localization technique.

Table 5: Average time in minutes for each localization step over all mutants by subject. Reaction counting (RC), no reaction counting (NRC) are the simulation methods. Evaluation methods are once (1x) or at all timesteps (Tx). Species (S), Reactions (R), and Traceback (T) are localization methods. Median Tot. Time is the total median time across all steps for each localization method.

	Simulation		Evaluation		Localization			Median Tot. Time		
Subj.	RC	NRC	1x	Tx	S	R	T	S	R	T
AM	227.7	17.0	29.6	32.5	11.0	11.2	10.9	5.4	5.4	5.5
Mod	42.1	11.8	20.6	22.0	11.2	11.4	11.8	2.2	2.2	2.3
Max	317.4	15.4	88.2	88.0	11.7	11.9	12.0	4.9	4.8	4.9
AL1	72.8	15.8	76.5	76.6	12.2	11.7	11.6	4.2	3.6	3.4
H1	967.0	13.9	283.8	267.9	16.0	14.8	16.3	14.0	46.0	45.6
ECG	10.3	9.6	18.5	16.8	12.5	13.0	13.8	1.5	1.5	1.6

Table 5 shows the runtimes of the different steps (Figure 3) used in our study. The numbers are averages over 5 runs in minutes. Each subject includes the time to complete each step for all of its mutants. The last 3 columns are the total median times for each localization method. In the simulation step reaction counting (RC) adds significant time in some subjects. For instance, H1 using reaction counting has an average simulation time of almost 16 hours to complete all mutants (about an hour per mutant) compared to 14 minutes overall (or less than 1 minute per mutant). In the evaluation step, we do not see much differentiation between the evaluation at one point in time versus at each step (used only in Traceback). H1 is again an outlier. Last, all of the localization techniques are relatively even time-wise, taking between 11 to 16 minutes to complete. The performance of the localization process seems to be even across all techniques. Overall, the simulation and evaluation are the most expensive steps, with reaction counting as the bottleneck.

Table 6: Average normalized rank per subject and all subjects for parameters k and w. Bold entries indicate the best results.

	Subject						
	AM	Mod	Max	AL1	H1	ECG	ALL
k							
0	**0.00**	**0.01**	**0.13**	0.32	**0.02**	**0.00**	**0.08**
1	0.02	0.08	0.19	**0.26**	0.05	**0.00**	0.12
2	0.08	0.12	0.24	0.31	0.08	**0.00**	0.16
3	0.08	0.12	0.25	0.33	0.10	**0.00**	0.17
4	0.10	0.12	0.25	0.34	0.11	**0.00**	0.18
5	0.10	0.14	0.25	0.35	0.11	**0.00**	0.19
w							
0	**0.00**	**0.01**	**0.13**	0.32	**0.02**	**0.00**	**0.08**
0.1	0.02	0.07	0.19	**0.29**	0.06	**0.00**	0.12
0.25	0.02	0.07	0.19	**0.29**	0.06	**0.00**	0.13
0.5	0.02	0.08	0.20	0.30	0.06	**0.00**	0.13
0.75	0.03	0.10	0.21	0.31	0.08	**0.00**	0.14
0.9	0.05	0.11	0.21	0.32	0.09	**0.00**	0.15
1	0.31	0.26	0.41	0.40	0.19	**0.00**	0.3

Summary of RQ1. Traceback is the most effective localization method outperforming the two SBFL techniques. Reaction based localization provides some differentiation, while Species based localization provides none. The most computationally expensive part of the process is the reaction counting based simulation which is required both by Reaction and Traceback localization.

5.2 RQ2: Traceback Parameter Tuning

We conducted a parameter tuning study to evaluate the effect of parameters on Traceback localization. Table 6 shows the ranks for the depth parameter (k) and weight parameter (w) for 6 values of k and 7 values of w. We see that a k value of 0 performs slightly better (bold is the best rank) in almost all subjects, except AL1 where $k = 1$ is the best result. For w we see a similar result with a weight of 0 giving the best result except in AL1. We note, that when either k or w is set to zero, the other one automatically becomes zero as well (w is a multiplier, and if k is zero there is no weight.)

We then evaluated the effect of k and w broken out by the different types of test suites to see if this has an impact on the results. Figure 7 shows the Hailstone subject using the three different test suite types for k (top row) and w (bottom row). Figure 8, shows similar data for the Mod subject for k (we do not have metamorphic tests for this program). The data for w is on our website. We can see that increasing k and w do not impact the overall localization, however, when using some types of tests, increasing these values may help. With the functional and metamorphic tests, higher k and w was needed to achieve the best ranks (0 performs worst).

Summary of RQ2. The parameter choice for Traceback has a small impact on the resulting ranks when we use all tests. However, we see differences when using the functional or metamorphic tests.

5.3 RQ3: Test Type Effectiveness

We now drill down a bit more and examine the difference in test case type for all of the approaches. We show this data in the boxplots in Figures 9 and 10. Figure 9 shows the rank by test suite type for the Hailstone (H1) subject and Figure 10 shows the normalized ranks across all subjects. In these graph Sp stands for Species, R for Reaction and TB for Traceback. All indicates the full test suite, F are functional tests, M are metamorphic tests and I are invariant tests. Based on the results from RQ2 we included data for both $k = 0$ and $k = 5$ (TB-F-k0, TB-F-k5, TB-M-k0, TB-M-k5) for the functional and metamorphic tests. We show the $k = 5$ plots in light blue; this is the value of k used in RQ1. In Species and Reaction there is no significant difference between test types. In Reaction localization the ALL test suite shows the best localization results (slightly) suggesting that all different types of tests together give the most information. In Traceback, the type of test appears to have impact on the quality of localization. The metamorphic tests provide slightly better localization than the functional tests for both

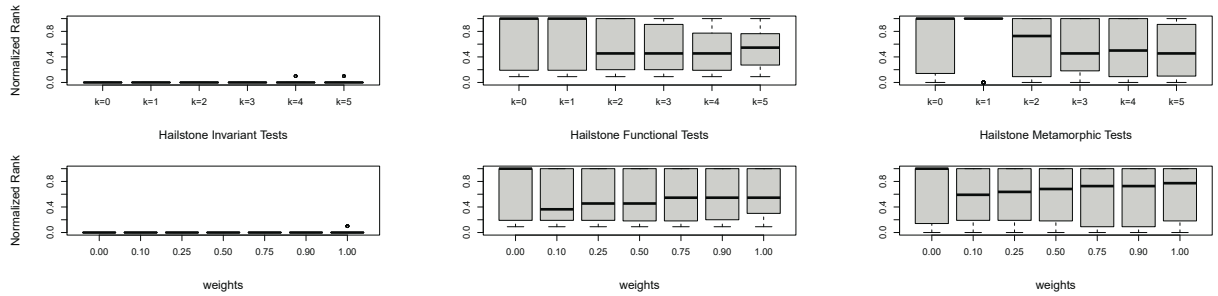

Figure 7: Normalized localization ranks for Traceback by test suite type for the Hailstone subject. Shows the different values of k (top) and w (bottom). A lower score is better.

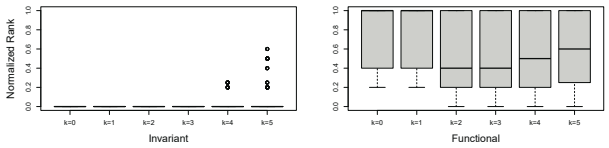

Figure 8: Normalized localization ranks with different values of k for Traceback by test suite type for the Mod subject. A lower score is better.

values of k. However, $k = 5$ gives the best result for both functional and metamorphic tests.

> **Summary of RQ3.** The test type plays a role in the localization performance of Traceback, however, individual test types perform the same or worse in Species and Reaction. Within functional and metamorphic tests, the parameter k also impacts the results.

5.4 Discussion

The invariant tests provide the best localization overall, and these do not require k to be greater than zero. When we have invariants present they seem to provide the best localization, even in the presence of other types of tests. This is not too surprising since invariants are extracted directly from the CRN reactions, and have the most structural information. However, invariant tests may not always exist. They can only be generated during regression testing, when a valid program exists. And as shown in Gerten et al. [18], some types of faults cannot be found by invariant tests; they require metamorphic or functional tests to find. As in most types of testing, we recommend using both types of test cases (structural vs. specification) when possible. However, when performing localization it is worthwhile noting which type of tests are being run since we can avoid setting parameter weights if we only have invariant tests.

In both the metamorphic and functional tests, parameter weighting seems to be important for localization, hence this connection should be evaluated by additional studies. We intend to explore this along with the idea of automated parameter tuning in future work.

6 RELATED WORK

Molecular programming is an interdisciplinary field encompassing computer science, mathematics, chemistry, and biology and the CRN model is a commonly used representation [4, 14, 16, 24, 28, 29, 55, 64, 65]. Researchers have been exploring ways to apply requirements engineering [40, 41, 43, 45],verification[6, 25, 31, 33, 34, 42, 44, 56, 59] and software testing [17, 18, 59] to CRNs, as well as building new programming languages [60] and automating program repair[46].

Many spectrum-based methods for fault localization have been developed (see [52, 67, 73] for surveys detailing more than 30 different techniques). Well-known SBFL techniques include Tarantula[26], DStar[66] and Ochiai[2]. We don't attempt to expand on all here, but comment that Ochiai has shown to have consistently good results[48]; hence we use it as our exemplar. These techniques have been heavily evaluated, extended and improved [10, 20, 23, 39, 53, 63, 70, 71, 75, 76]. Examples of fault localization in other domains include machine learning [9, 62], model transformations[58], and Simulink automotive models[35, 36]. Alternative (earlier) approaches include statistical debugging[10] and delta debugging[74].

There has also been a line of research that uses dynamic program slicing [77] and partial invariants (dynamically inferred invariants) [5, 50]. The invariants we use are statically inferred (always hold on the original program), that can only be obtained from an existing, correct program (i.e. during regression). Traceback does not require invariants, but can use any type of test. We also use specification based functional and metamorphic tests in this work. Traceback has a similar flavor to the dynamic slicing work of Sahoo et al.[50], however, they repeatedly regenerate and rerun invariants to narrow the failing input space which is too expensive in CRNs and unnecessary for Traceback. Last they leverage control flow and dependency graphs during slicing to remove candidate program elements, however, there is no notion of control flow in a CRN; our dependence graph is independent of flow. In CRNs almost all reactions will occur in all traces (passing and failing) and in different orders, even on the same input, hence we can't use a single trace. Instead we use a statistical summarization to first aggregate over multiple runs of a single test, and then over all failing tests.

Petri nets have long been used to model physical processes from industrial automation to medical devices. It is not surprising that numerous variants of Petri nets have emerged in order to address

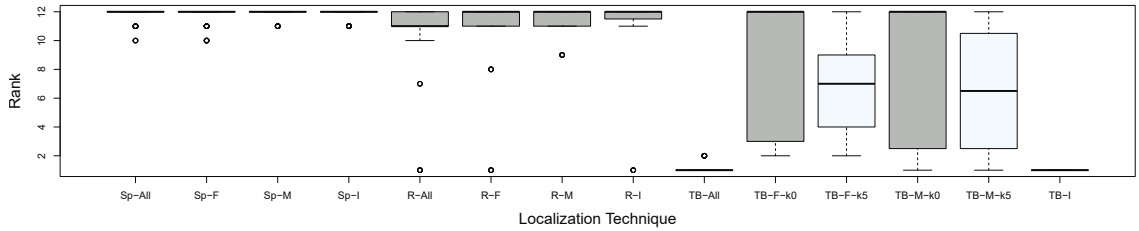

Figure 9: Ranks by technique for each test suite for the H1 subject. Sp stands for species, R stands for reaction, and TB for Traceback. All represent all tests, F are functional tests, M are metamorphic tests, and I are invariant tests. Lower is better.

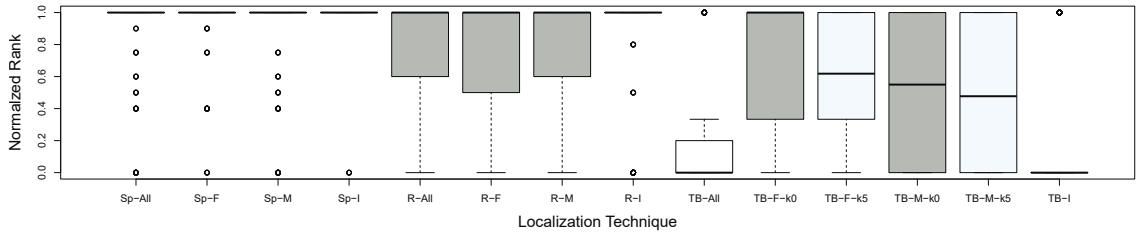

Figure 10: Normalized localization ranks by technique for by test suite for all subjects. Sp stands for species, R for reaction, and TB for Traceback. All represents all tests, F are functional tests, M are metamorphic tests and I are invariant tests. Lower is better.

specific issues and weaknesses in the model. The Place/Transition (P/T) Petri net is one of the most commonly used version, but other varieties such as Colored Petri nets, Timed Petri nets, Stochastic Petri nets, Real-valued Petri nets, Dynamic Petri nets, Object-oriented Petri nets, and many others are commonly used to model physical systems as well[32]. As a physical system, the CRN model is equivalent to a hybrid Petri model that combines Stochastic Petri (not Generalized Stochastic) nets and Colored Petri nets. Within this model (and to some degree more generally) methods for fault localization utilize model checking techniques, which in general do not scale well (see Gerten et al. [17]), and would be difficult if not impossible to apply to even small or moderate size CRN systems.

7 CONCLUSIONS AND FUTURE WORK

We explored the use of fault localization in chemical reaction networks. We applied two SBFL techniques directly to elements of the CRN (species and reactions) and proposed a new approach called Traceback. We found that Traceback provides the best localization results ranging from a 32-90% improvement over the reaction based method. Traceback is relatively stable with respect to its parameters k and w, however we learned that the test type has an impact both on localization and on the parameter settings. In general, invariant tests provide the best localization results. When we use functional or metamorphic tests, we may benefit from using non-zero values of k and w.

In future work, we plan to explore other types of CRNs such as those that use waveforms and those with multiple faults. We also plan to apply this localization to stochastic Simulink models which have some similarities with CRNs. We will also perform larger parameter tuning experiments and explore automated turning approaches.

ACKNOWLEDGMENTS

This work was funded in part by National Science Foundation Grants CCF-1909688 and FET-1900716. We thank the anonymous reviewers for useful comments.

REFERENCES

[1] Rui Abreu, Alberto González, Peter Zoeteweij, and Arjan J. C. van Gemund. 2008. Automatic Software Fault Localization Using Generic Program Invariants. In *Proceedings of the 2008 ACM Symposium on Applied Computing*. 712–717. https://doi.org/10.1145/1363686.1363855

[2] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. 2007. On the Accuracy of Spectrum-Based Fault Localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION 2007)*. IEEE, 89–98. https://doi.org/10.1109/TAIC.PART.2007.13

[3] Ebbe S. Andersen, Mingdong Dong, Morten M. Nielsen, Kasper Jahn, Ramesh Subramani, Wael Mamdouh, Monika M. Golas, Bjoern Sander, Holger Stark, Cristiano L. P. Oliveira, Jan Skov Pedersen, Victoria Birkedal, Flemming Besenbacher, Kurt V. Gothelf, and Jørgen Kjems. 2009. Self-assembly of a nanoscale DNA box with a controllable lid. *Nature* 459, 7243 (01 May 2009), 73–76. https://doi.org/10.1038/nature07971

[4] Rutherford Aris. 1965. Prolegomena to the rational analysis of systems of chemical reactions. *Archive for Rational Mechanics and Analysis* 19, 2 (1965), 81–99. https://doi.org/10.1007/BF00282276

[5] Tien-Duy B. Le, David Lo, Claire Le Goues, and Lars Grunske. 2016. A Learning-to-Rank Based Fault Localization Approach Using Likely Invariants. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. 177–188. https://doi.org/10.1145/2931037.2931049

[6] Benoît Barbot and Marta Kwiatkowska. 2015. On Quantitative Modelling and Verification of DNA Walker Circuits Using Stochastic Petri Nets. In *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, 1–32. https://doi.org/10.1007/978-3-319-19488-2_1

[7] Jonathan Bell, Owolabi Legunsen, Michael Hilton, Lamyaa Eloussi, Tifany Yung, and Darko Marinov. 2018. DeFlaker: Automatically Detecting Flaky Tests. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 433–444. https://doi.org/10.1145/3180155.3180164

[8] Árpád Beszédes, Ferenc Horváth, Massimiliano Di Penta, and Tibor Gyimóthy. 2020. Leveraging Contextual Information from Function Call Chains to Improve Fault Localization. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 468–479. https://doi.org/10.1109/SANER48275.2020.9054820

[9] Jialun Cao, Meiziniu Li, Xiao Chen, Ming Wen, Yongqiang Tian, Bo Wu, and Shing-Chi Cheung. 2022. DeepFD: Automated Fault Diagnosis and Localization for Deep Learning Programs. In *Proceedings of the 44th International Conference on Software Engineering*. 573–585. https://doi.org/10.1145/3510003.3510099

[10] Trishul M. Chilimbi, Ben Liblit, Krishna Mehra, Aditya V. Nori, and Kapil Vaswani. 2009. HOLMES: Effective Statistical Debugging via Efficient Path Profiling. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09)*. 34–44. https://doi.org/10.1109/ICSE.2009.5070506

[11] Shawn M. Douglas, Ido Bachelet, and George M. Church. 2012. A Logic-Gated Nanorobot for Targeted Transport of Molecular Payloads. *Science* 335, 6070 (2012), 831–834. https://doi.org/10.1126/science.1214081

[12] Samuel J. Ellis, Titus H. Klinge, James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, Andrew S. Miner, and Hugh D. Potter. 2019. Runtime Fault Detection in Programmed Molecular Systems. *ACM Transactions on Software Engineering Methodology* 28, 2, Article 6 (March 2019), 20 pages. https://doi.org/10.1145/3295740

[13] Samuel J. Ellis, James I. Lathrop, and Robyn R. Lutz. 2017. State Logging in Chemical Reaction Networks. In *Proceedings of the 4th ACM International Conference on Nanoscale Computing and Communication (NanoCom '17)*. Article 23, 6 pages. https://doi.org/10.1145/3109453.3109456

[14] Péter Érdi and János Tóth. 1989. *Mathematical Models of Chemical Reactions: Theory and Applications of Deterministic and Stochastic Models*. Manchester University Press. https://doi.org/10.1002/bbpc.19890931228

[15] François Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. [n. d.]. Strong Turing Completeness of Continuous Chemical Reaction Networks and Compilation of Mixed Analog-Digital Programs. In *Computational Methods in Systems Biology* (2017), Jérôme Feret and Heinz Koeppl (Eds.). Springer International Publishing, 108–127. https://doi.org/10.1007/978-3-319-67471-1_7

[16] Martin Feinberg. 2019. *Foundations of Chemical Reaction Network Theory*. Springer International Publishing. https://doi.org/10.1007/978-3-030-03858-8

[17] Michael C. Gerten, James I. Lathrop, Myra B. Cohen, and Titus H. Klinge. 2020. ChemTest: An Automated Software Testing Framework for an Emerging Paradigm. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20)*. 548–560. https://doi.org/10.1145/3324884.3416638

[18] Michael C. Gerten, Alexis L. Marsh, James I. Lathrop, Myra B. Cohen, Andrew S. Miner, and Titus H. Klinge. 2022. Inference and Test Generation Using Program Invariants in Chemical Reaction Networks. In *International Conference on Software Engineering*. 1193–1205. https://doi.org/10.1145/3510003.3510176

[19] Michael C. Gerten, Alexis L. Marsh, James I. Lathrop, Myra B. Cohen, Andrew S. Miner, and Titus H. Klinge. 2022. *Supplementary Data: Inference and Test Generation Using Program Invariants in Chemical Reaction Networks Artifacts*. https://doi.org/10.5281/zenodo.6981488

[20] Liang Gong, David Lo, Lingxiao Jiang, and Hongyu Zhang. 2012. Diversity Maximization Speedup for Fault Localization. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE '12)*. 30–39. https://doi.org/10.1145/2351676.2351682

[21] The MathWorks Inc. 2022. MATLAB Version: 9.13.0.2049777 (R2022b). The MathWorks Inc..

[22] Shuoxing Jiang, Zhilei Ge, Shan Mou, Hao Yan, and Chunhai Fan. 2021. Designer DNA nanostructures for therapeutics. *Chem* 7, 5 (2021), 1156–1179. https://doi.org/10.1016/j.chempr.2020.10.025

[23] Wei Jin and Alessandro Orso. 2013. F3: Fault Localization for Field Failures. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA 2013)*. Association for Computing Machinery, 213–223. https://doi.org/10.1145/2483760.2483763

[24] Robert Francis Johnson. 2020. *Formal Design and Analysis for DNA Implementations of Chemical Reaction Networks*. Ph. D. Dissertation. California Institute of Technology. https://doi.org/10.7907/a74v-kv80

[25] Robert F Johnson, Qing Dong, and Erik Winfree. 2016. Verifying Chemical Reaction Network Implementations: A Bisimulation Approach. In *DNA Computing and Molecular Programming: International Conference, DNA*. Springer, 114–134. https://doi.org/10.1016/j.tcs.2018.01.002

[26] James A. Jones and Mary Jean Harrold. 2005. Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE '05)*. 273–282. https://doi.org/10.1145/1101908.1101949

[27] Tanvir Ahmed Khan, Allison Sullivan, and Kaiyuan Wang. 2021. AlloyFL: A Fault Localization Framework for Alloy. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021)*. 1535–1539. https://doi.org/10.1145/3468264.3473116

[28] Titus H. Klinge, James I. Lathrop, and Jack H. Lutz. 2020. Robust Biomolecular Finite Automata. *Theorecitcal Compututer Science* 816, C (May 2020), 114–143. https://doi.org/10.1016/j.tcs.2020.01.008

[29] Titus H. Klinge, James I. Lathrop, Peter-Michael Osera, and Allison Rogers. 2021. Reactamole: Functional Reactive Molecular Programming. In *27th International Conference on DNA Computing and Molecular Programming (DNA 27)*, Vol. 205. 10:1–10:20. https://doi.org/10.4230/LIPIcs.DNA.27.10

[30] Pavneet Singh Kochhar, Xin Xia, David Lo, and Shanping Li. 2016. Practitioners' Expectations on Automated Fault Localization. In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016)*. 165–176. https://doi.org/10.1145/2931037.2931051

[31] Chase Koehler, Divita Mathur, Eric Henderson, and Robyn Lutz. 2018. Probing the Security of DNA Origami. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 138–139. https://doi.org/10.1109/ISSREW.2018.00-14

[32] Vinod B. Kumbhar and Mahesh S. Chavan. 2023. A Review of Petri Net Tools and Recommendations. In *Proceedings of the International Conference on Applications of Machine Intelligence and Data Analytics (ICAMIDA 2022)*. Atlantis Press, 710–721. https://doi.org/10.2991/978-94-6463-136-4_61

[33] Marta Kwiatkowska. 2014. Challenges in Automated Verification and Synthesis for Molecular Programming. *Essays for the Luca Cardelli Fest, volume MSR-TR-2014-104 of Technical Report* (2014), 155–170.

[34] Matthew R Lakin, Andrew Phillips, and Darko Stefanovic. 2013. Modular Verification of DNA Strand Displacement Networks via Serializability Analysis. In *DNA Computing and Molecular Programming: 19th International Conference, DNA 19, Tempe, AZ, USA, September 22-27, 2013. Proceedings 19*. Springer, 133–146. https://doi.org/10.1007/978-3-319-01928-4_10

[35] Bing Liu, Lucia, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. 2016. Simulink fault localization: an iterative statistical debugging approach. *Software Testing Verification and Reliability* 26, 6 (2016), 431–459. https://doi.org/10.1002/stvr.1605

[36] Bing Liu, Shiva Nejati, Lucia, and Lionel C. Briand. 2019. Effective fault localization of automotive Simulink models: achieving the trade-off between test oracle effort and fault localization accuracy. *Empirical Software Engineering* 24, 1 (2019), 444–490. https://doi.org/10.1007/s10664-018-9611-z

[37] Kui Liu, Anil Koyuncu, Tegawendé F. Bissyandé, Dongsun Kim, Jacques Klein, and Yves Le Traon. 2019. You Cannot Fix What You Cannot Find! An Investigation of Fault Localization Bias in Benchmarking Automated Program Repair Systems. In *12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019*. 102–113. https://doi.org/10.1109/ICST.2019.00020

[38] Kui Liu, Shangwen Wang, Anil Koyuncu, Kisub Kim, Tegawendé F. Bissyandé, Dongsun Kim, Peng Wu, Jacques Klein, Xiaoguang Mao, and Yves Le Traon. 2020. On the efficiency of test suite based program repair: A Systematic Assessment of 16 Automated Repair Systems for Java Programs. In *ICSE '20: 42nd International Conference on Software Engineering*. 615–627. https://doi.org/10.1145/3377811.3380338

[39] Yiling Lou, Ali Ghanbari, Xia Li, Lingming Zhang, Haotian Zhang, Dan Hao, and Lu Zhang. 2020. Can Automated Program Repair Refine Fault Localization? A Unified Debugging Approach. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2020)*. 75–87. https://doi.org/10.1145/3395363.3397351

[40] Jack Lutz and Robyn Lutz. 2018. Writing Requirements for Molecular Programs. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*. IEEE Computer Society, 512–512. https://doi.org/10.1109/RE.2018.00011

[41] Robyn Lutz, Jack Lutz, James Lathrop, Titus Klinge, Eric Henderson, Divita Mathur, and Dalia Abo Sheasha. 2012. Engineering and Verifying Requirements for Programmable Self-Assembling Nanomachines. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE, 1361–1364. https://doi.org/10.1109/ICSE.2012.6227079

[42] Robyn R. Lutz. 2016. Requirements for Molecular Programmed Nanosystems (Keynote). In *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 2–2. https://doi.org/10.1109/RE.2016.23

[43] Robyn R. Lutz. 2022. Requirements Engineering for Safety-Critical Molecular Programs. In *2022 IEEE 30th International Requirements Engineering Conference (RE)*. 302–308. https://doi.org/10.1109/RE54965.2022.00045

[44] Robyn R. Lutz and Jack H. Lutz. 2016. Software Engineering for Molecular Programming. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16)*. 888–889. https://doi.org/10.1145/2889160.2891048

[45] Robyn R. Lutz, Jack H. Lutz, James I. Lathrop, Titus H. Klinge, Divita Mathur, Donald M. Stull, Taylor G. Bergquist, and Eric R. Henderson. 2012. Requirements Analysis for a Product Family of DNA Nanodevices. In *Proceedings of the 20th International Conference on Requirements Engineering*. IEEE, 211–220. https:

//doi.org/10.1109/RE.2012.6345806

[46] Ibrahim Mesecan, Michael C. Gerten, James I. Lathrop, Myra B. Cohen, and Tomas Haddad Caldas. 2021. CRNRepair: Automated Program Repair of Chemical Reaction Networks. In *2021 IEEE/ACM International Workshop on Genetic Improvement (GI)*. 23–30. https://doi.org/10.1109/GI52543.2021.00014

[47] T. J. Ostrand and M. J. Balcer. 1988. The Category-partition Method for Specifying and Generating Fuctional Tests. *Commun. ACM* 31, 6 (June 1988), 676–686. https://doi.org/10.1145/62959.62964

[48] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D. Ernst, Deric Pang, and Benjamin Keller. 2017. Evaluating and Improving Fault Localization. In *Proceedings of the 39th International Conference on Software Engineering (ICSE '17)*. IEEE Press, 609–620. https://doi.org/10.1109/ICSE.2017.62

[49] William Poole, Ayush Pandey, Andrey Shur, Zoltan A. Tuza, and Richard M. Murray. 2022. BioCRNpyler: Compiling chemical reaction networks from biomolecular parts in diverse contexts. *PLOS Computational Biology* 18, 4 (April 2022), 1–19. https://doi.org/10.1371/journal.pcbi.1009987

[50] Swarup Kumar Sahoo, John Criswell, Chase Geigle, and Vikram Adve. 2013. Using Likely Invariants for Automated Software Fault Localization. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13)*. 139–152. https://doi.org/10.1145/2451116.2451131

[51] Raul Santelices, James A. Jones, Yanbing Yu, and Mary Jean Harrold. 2009. Lightweight Fault-Localization Using Multiple Coverage Types. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09)*. IEEE Computer Society, 56–66. https://doi.org/10.1109/ICSE.2009.5070508

[52] Qusay Idrees Sarhan and Árpád Beszédes. 2022. A Survey of Challenges in Spectrum-Based Software Fault Localization. *IEEE Access* 10 (2022), 10618–10639. https://doi.org/10.1109/ACCESS.2022.3144079

[53] Marius Smytzek and Andreas Zeller. 2022. SFLKit: A Workbench for Statistical Fault Localization. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022)*. 1701–1705. https://doi.org/10.1145/3540250.3558915

[54] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. [n. d.]. Computation with Finite Stochastic Chemical Reaction Networks. 7, 4 ([n. d.]), 615–633. https://doi.org/10.1007/s11047-008-9067-y

[55] David Soloveichik, Georg Seelig, and Erik Winfree. 2010. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences* 107, 12 (2010), 5393–5398. https://doi.org/10.1073/pnas.0909380107

[56] Kouichi Takahashi, Katsuyuki Yugi, Kenta Hashimoto, Yohei Yamada, Christopher JF Pickett, and Masaru Tomita. 2002. Computational Challenges in Cell Simulation: A Software Engineering Approach. *IEEE Intelligent Systems* 17, 5 (2002), 64–71. https://doi.org/10.1109/MIS.2002.1039834

[57] George Thompson and Allison K. Sullivan. 2020. ProFL: A Fault Localization Framework for Prolog. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2020)*. 561–564. https://doi.org/10.1145/3395363.3404367

[58] Javier Troya, Sergio Segura, Jose Antonio Parejo, and Antonio Ruiz-Cortés. 2018. Spectrum-Based Fault Localization in Model Transformations. *ACM Transactions on Software Engineering and Methodology* 27, 3, Article 13 (Sept 2018), 50 pages. https://doi.org/10.1145/3241744

[59] Marko Vasic, David Soloveichik, and Sarfraz Khurshid. 2020. CRNs Exposed: A Method for the Systematic Exploration of Chemical Reaction Networks. In *International Conference on DNA Computing and Molecular Programming (LIPIcs, Vol. 174)*. 4:1–4:25. https://doi.org/10.4230/LIPIcs.DNA.2020.4

[60] Marko Vasić, David Soloveichik, and Sarfraz Khurshid. 2020. CRN++: Molecular programming language. *Natural Computing* 19, 2 (June 2020), 391–407. https://doi.org/10.1007/s11047-019-09775-1

[61] Kaiyuan Wang, Allison Sullivan, Darko Marinov, and Sarfraz Khurshid. 2020. Fault Localization for Declarative Models in Alloy. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. 391–402. https://doi.org/10.1109/ISSRE5003.2020.00044

[62] Mohammad Wardat, Wei Le, and Hridesh Rajan. 2021. DeepLocalize: Fault Localization for Deep Neural Networks. In *Proceedings of the 43rd International Conference on Software Engineering (ICSE '21)*. 251–262. https://doi.org/10.1109/ICSE43902.2021.00034

[63] Ming Wen, Junjie Chen, Yongqiang Tian, Rongxin Wu, Dan Hao, Shi Han, and Shing-Chi Cheung. 2021. Historical Spectrum Based Fault Localization. *IEEE Transactions on Software Engineering* 47, 11 (2021), 2348–2368. https://doi.org/10.1109/TSE.2019.2948158

[64] Mingjian Wen, Evan Walter Clark Spotte-Smith, Samuel M Blau, Matthew J McDermott, Aditi S Krishnapriyan, and Kristin A Persson. 2023. Chemical Reaction Networks and Opportunities for Machine Learning. *Nature Computational Science* 3, 1 (2023), 12–24. https://doi.org/10.1038/s43588-022-00369-z

[65] Erik Winfree. 2019. Chemical Reaction Networks and Stochastic Local Search. In *Proceedings of the Twenty-Fifth International Conference on DNA Computing and Molecular Programming (Lecture Notes in Computer Science)*, Chris Thachuk and Yan Liu (Eds.). Springer, 1–20. https://doi.org/10.1007/978-3-030-26807-7_1

[66] W. Eric Wong, Vidroha Debroy, Yihao Li, and Ruizhi Gao. 2012. Software Fault Localization Using DStar (D*). In *Proceedings of the 2012 IEEE Sixth International Conference on Software Security and Reliability (SERE '12)*. 21–30. https://doi.org/10.1109/SERE.2012.12

[67] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740. https://doi.org/10.1109/TSE.2016.2521368

[68] Nuli Xie, Mingqiang Li, Yue Wang, Hui Lv, Jiye Shi, Jiang Li, Qian Li, Fei Wang, and Chunhai Fan. 2022. Scaling Up Multi-bit DNA Full Adder Circuits with Minimal Strand Displacement Reactions. *Journal of the American Chemical Society* 144, 21 (2022), 9479–9488. https://doi.org/10.1021/jacs.2c03258

[69] Cemal Yilmaz, Amit Paradkar, and Clay Williams. 2008. Time Will Tell: Fault Localization Using Time Spectra. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*. Association for Computing Machinery, 81–90. https://doi.org/10.1145/1368088.1368100

[70] Shin Yoo, Mark Harman, and David Clark. 2013. Fault Localization Prioritization: Comparing Information-Theoretic and Coverage-Based Approaches. *ACM Transactions on Software Engineering and Methodology* 22, 3, Article 19 (July 2013), 29 pages. https://doi.org/10.1145/2491509.2491513

[71] Junji Yu, Yan Lei, Huan Xie, Lingfeng Fu, and Chunyan Liu. 2022. Context-Based Cluster Fault Localization. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension (ICPC '22)*. 482–493. https://doi.org/10.1145/3524610.3527891

[72] Youyang Yuan, Hui Lv, and Qiang Zhang. 2022. Molecular device design based on chemical reaction networks: state feedback controller, static pre-filter, addition gate control system and full-dimensional state observer. *Journal of Mathematical Chemistry* 60, 5 (01 May 2022), 915–935. https://doi.org/10.1007/s10910-022-01340-z

[73] Abubakar Zakari, Sai Peck Lee, Rui Abreu, Babiker Hussien Ahmed, and Rasheed Abubakar Rasheed. 2020. Multiple Fault Localization of Software Programs: A Systematic Literature Review. *Information and Software Technology* 124 (Aug. 2020), 106312. https://doi.org/10.1016/j.infsof.2020.106312

[74] Andreas Zeller. 2002. Isolating Cause-Effect Chains from Computer Programs. In *Proceedings of the ACM Symposium on Foundations of Software Engineering*. 1–10. https://doi.org/10.1145/587051.587053

[75] Mengshi Zhang, Xia Li, Lingming Zhang, and Sarfraz Khurshid. 2017. Boosting Spectrum-Based Fault Localization Using PageRank. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017)*. Association for Computing Machinery, 261–272. https://doi.org/10.1145/3092703.3092731

[76] Mengshi Zhang, Yaoxian Li, Xia Li, Lingchao Chen, Yuqun Zhang, Lingming Zhang, and Sarfraz Khurshid. 2021. An Empirical Study of Boosting Spectrum-Based Fault Localization via PageRank. *IEEE Transactions on Software Engineering* 47, 6 (June 2021), 1089–1113. https://doi.org/10.1109/TSE.2019.2911283

[77] Xiangyu Zhang, R. Gupta, and Youtao Zhang. 2003. Precise dynamic slicing algorithms. In *25th International Conference on Software Engineering, 2003. Proceedings.* 319–329. https://doi.org/10.1109/ICSE.2003.1201211

Received 16-DEC-2023; accepted 2024-03-02