Neutrality and Epistasis in Program Space

Joseph Renzullo
Arizona State University
renzullo@asu.edu

Westley Weimer
University of Michigan
weimerw@umich.edu

Melanie Moses
University of New Mexico
melaniem@cs.unm.edu

Stephanie Forrest
Arizona State University
stephanie.forrest@asu.edu

I4Soft
Intelligence for Software

CAPES

UFG

INSTITUTO DE INFORMÁTICA
UFG

Questions

1. *How should we scale up current methods in genetic improvement to tackle much larger codebases and more complex repairs?*
2. *What is the best way to navigate fitness plateaus (a.k.a.* ***Neutral Network****)?*
3. *Which are common when test cases are used as fitness functions?*
4. *What portion of a* ***fitness landscape*** *is made up of this plateau?*
5. *Does this portion change, along with mutational robustness, as we move away from the original program by iterated mutation?*
6. *Are there more or better repairs* ***near the original program?*** *Or* ***farther away*** *in program-space?*
7. *How do edits interact, as they accumulate?*

Questions

These motivating questions **are not answered here**.
In this paper we show **how they can be investigated** by adapting techniques from evolutionary biology.

Neutrality

- Genetic variants that have the **same fitness** are referred to as neutral;
- A **neutral network** is a set of **equal-fitness** individuals related by single mutations;
- The **topology** of neutral networks is key to find high-fitness innovations (exploration) while preserving already-discovered innovations (exploitation);
- Neutrality and **robustness** are key to evolution.

Mutational Robustness

- In biology, mutational robustness refers to an organism’s ability to preserve its phenotype in the face of internal genetic mutations;
- Mutational robustness is high (> 30%) in a corpus of open-source programs. This means that ~30% of single mutations are neutral;
- Repairs for bugs (innovations) are clustered in different regions of the network corresponding to different ways of repairing the same bug.

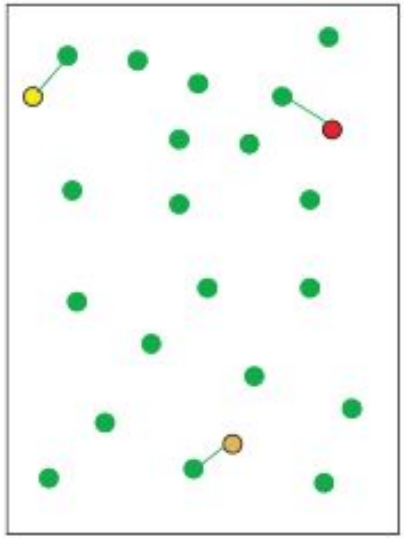
Mutation Operators

We begin with source-level C programs, translate them into the corresponding **abstract syntax tree** (AST) and apply mutations **at this level**.

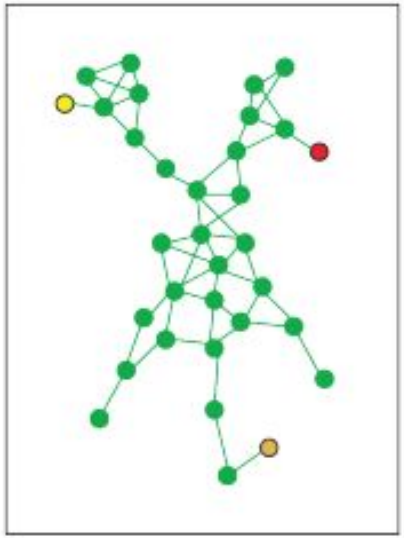
- **Delete** - deletes a randomly selected node (and its subtree if one exists) from the AST.
- **Copy** - selects a random node (and its subtree if one exists) in the AST and copies it to another random location.
- **Swap** - selects two random nodes (and their subtrees if they exist) in the AST and exchanges their positions

We say that the mutation is neutral if the mutation does not change the behavior of the program on its test suite.

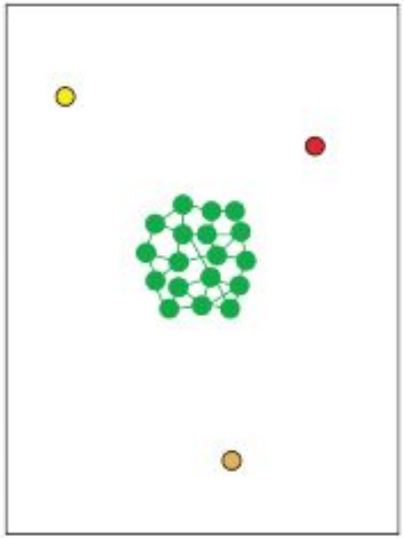
Mutational robustness and **evolvability**



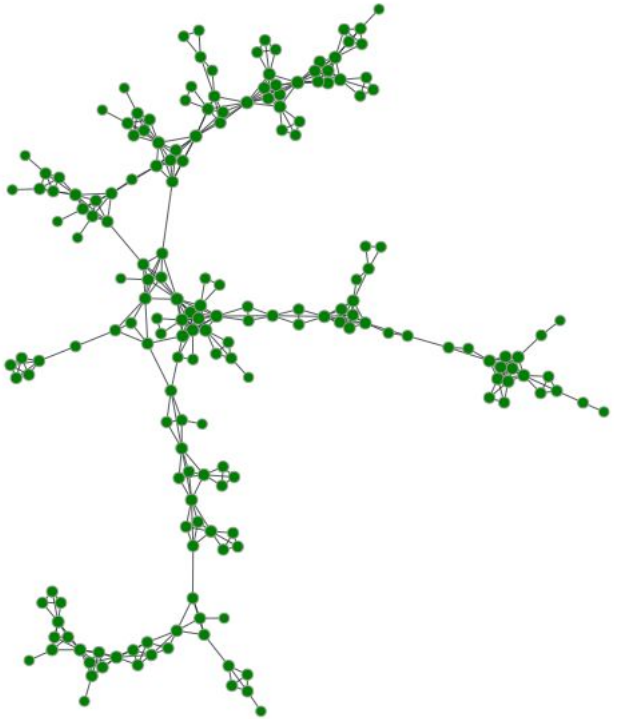
Low Robustness
Low Innovation



High Robustness
High Innovation



High Robustness
Low Innovation



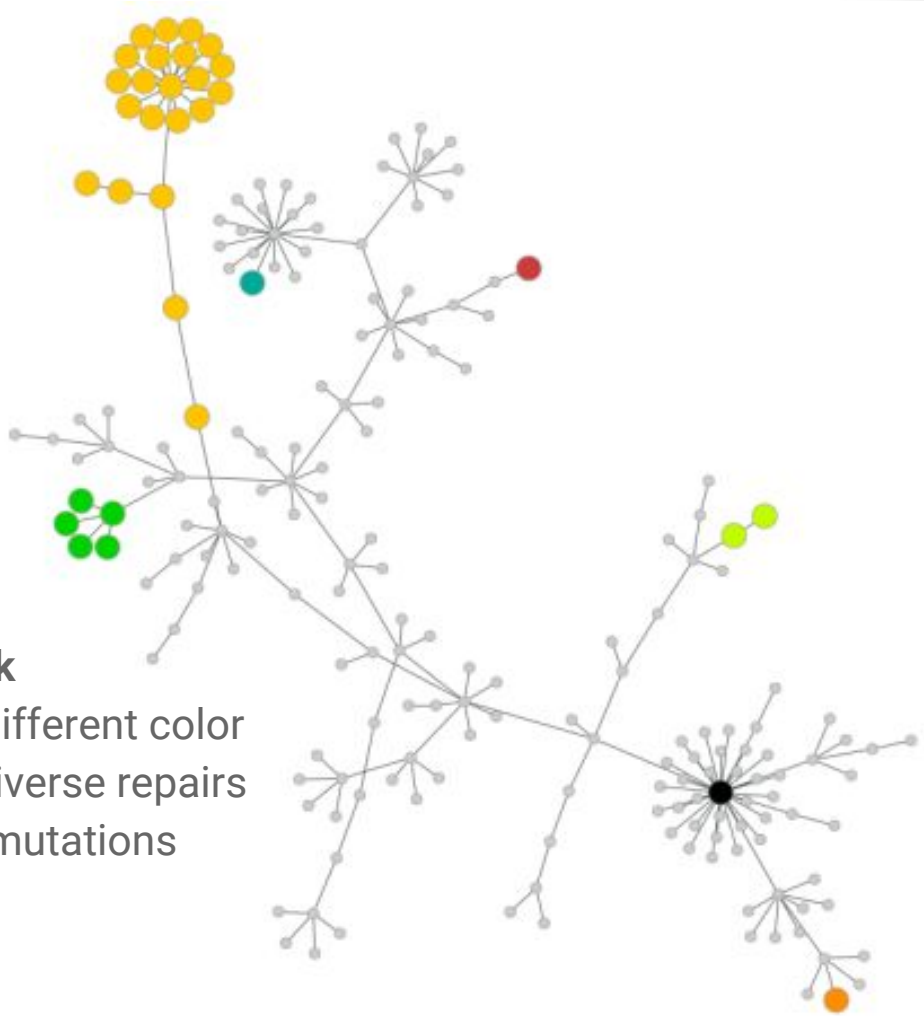
Traversability

To demonstrate the existence and traversability of neutral networks in software was applied single mutations iteratively on *1ook* code.

Algorithm 1 Generate Variants

Let k=5, g=10, $\alpha = \frac{2}{3}$, and $\beta = 1 - \alpha$.
for 1 **to** g **do**
    select k nodes at the edge of the graph
    for all selected nodes **do**
        apply single mutations to generate n children of this node,
        s.t. $p(\textbf{copy}) = \alpha$, $p(\textbf{delete}) = \beta$,
        and $\forall_{i \in [0,4]} : p(n = 2^i) \propto \frac{1}{\log_2 2^{i+1}}$
    end for
end for

Traversability

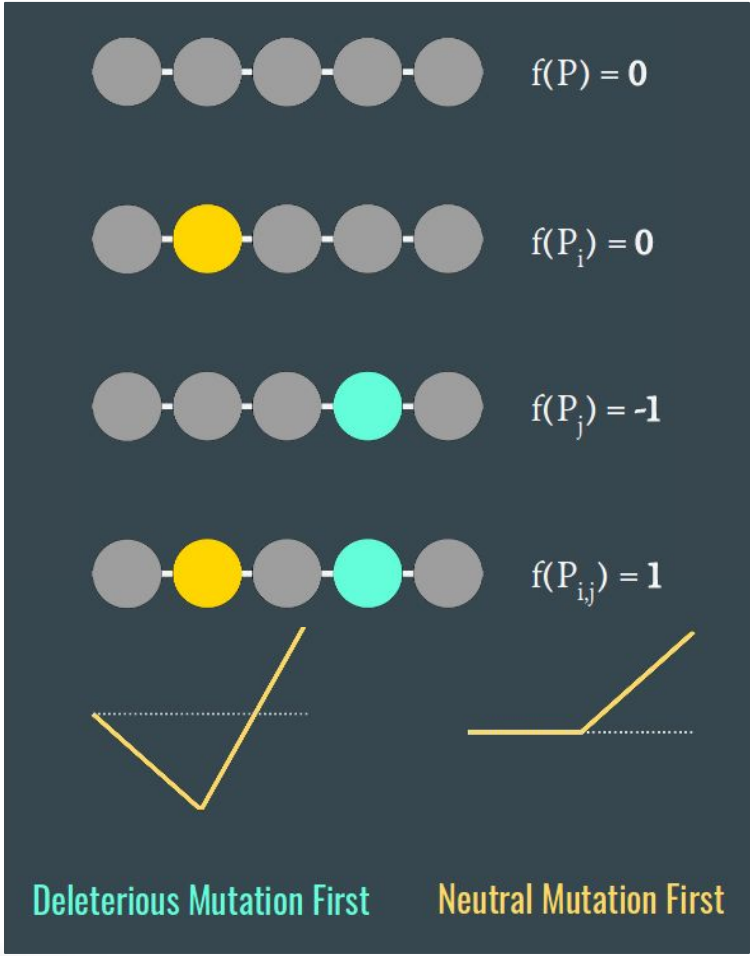


- Original program in **black**
- Each unique repair is a different color
- The network connects diverse repairs by neutral intermediate mutations

Epistasis

Epistasis in Software

- Interactions between genes or mutations;
- Genes commonly interact;
- The whole is seldom the sum of the parts;
- This affects evolutionary landscapes;
- Order Matters.



Epistasis in Software

i and j are individual mutations and P is a program. f(P) return the fitness of program P less the fitness of the original program.

if $f(P_{ij}) > f(P_i)$ and $f(P_{ij}) > f(P_j)$:
  Epistasis(P_{ij}) = Positive
if $f(P_{ij}) < f(P_i)$ and $f(P_{ij}) < f(P_j)$:
  Epistasis(P_{ij}) = Negative

Listing 1: Example Repair

```
594     bot = 0L;
595 --  fseek(dfile, 0L, 2);
596     top = ftell(dfile);
597     while (1) { ... }
...
...
...
710     else {
...
717     tmp___0 = tmp;
718 ++  return (tmp___1);
719     }
```

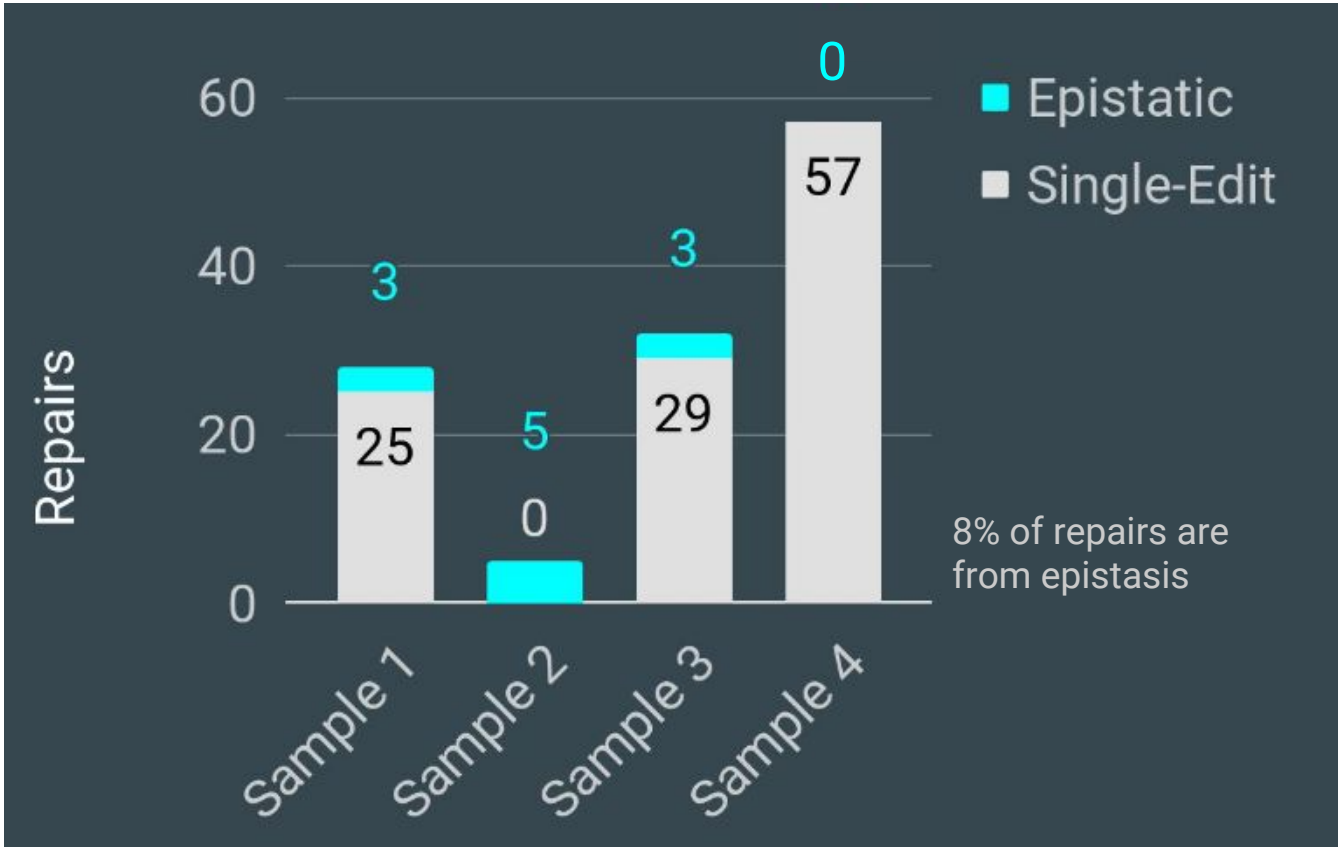
The experiment

Epistasis in Software, The Experiment

1. Begin with a buggy program with a latent bug (original program)
2. Sample 10 AST nodes (from the execution paths)
3. Generate all possible single - edit mutations affecting those nodes (10 delete, 45 append, and 45 swap)
4. Construct the entire set of pairs of edits (4950)
5. Compute pairwise epistatic interaction of edits & other metrics of interest
6. Repeat Step 2 - Step 5, four times per program

Data Generated: **20,200 mutated** variants per program (101,000 total)

Epistasis in Software, The Experiment



Epistatic Repairs: a closer look at *look*

Individual Edit Fitness

Neutral + Deleterious	7 repairs
Neutral + Neutral	4 repairs

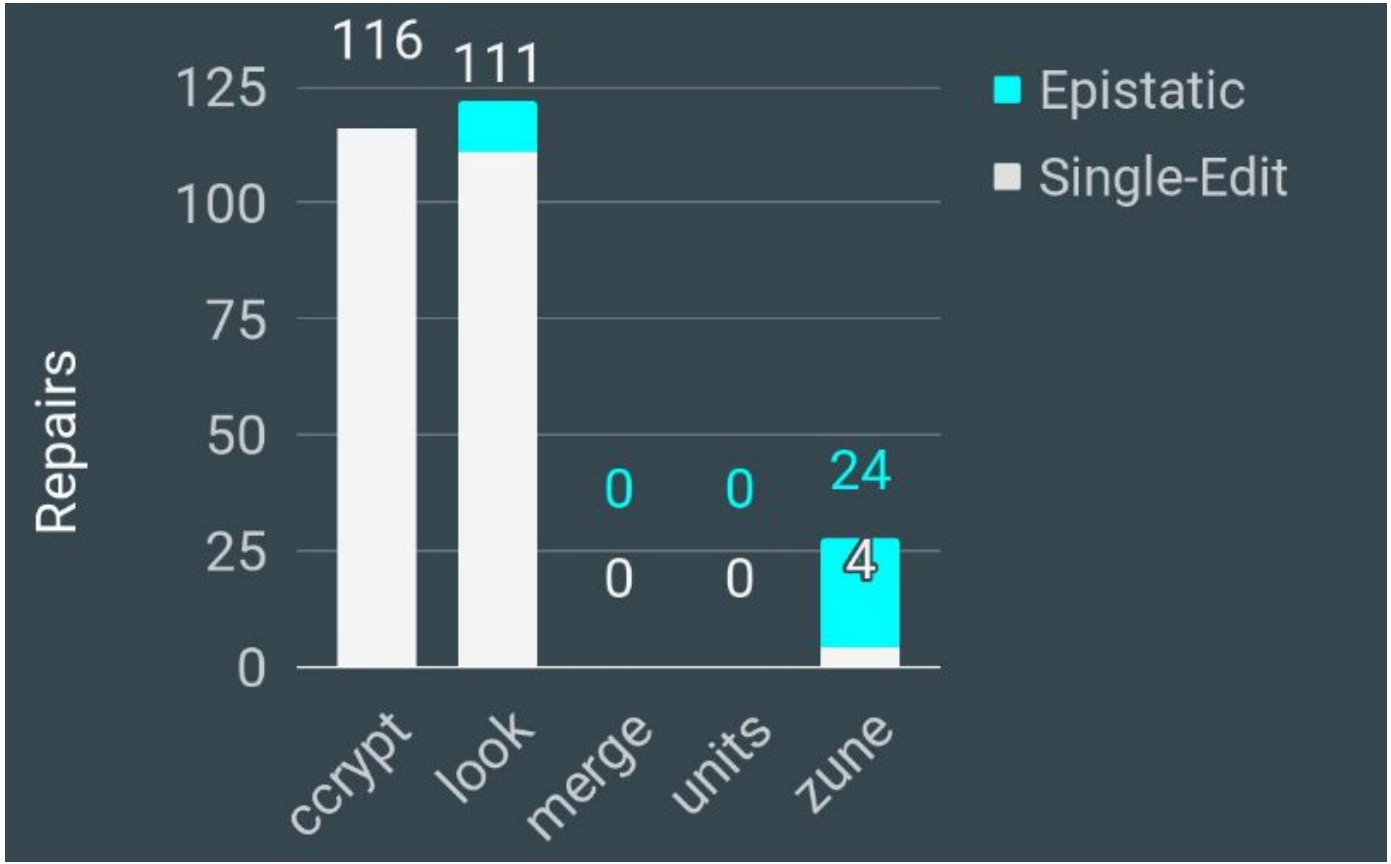
Effect of Mutation

Add a Return Statement + Change Control Flow	8 repairs
Change Control Flow + Change Control Flow	3 repairs

Epistasis in Software, The Experiment

	Sample 1	Sample 2	Sample 3	Sample 4	Average Percent
Does not compile	3901	3547	4304	2794	72.01%
Not Neutral	530	511	375	581	9.89%
Neutral	591	987	339	1618	17.50%
Repair	28	5	32	57	0.60%
Positive Epistasis	3	5	3	0	0.05%
Negative Epistasis	1	3	1	6	0.05%

Repairs in Five Small C Programs



Repairs in Five Small C Programs

	ccrypt	look	merge	units	zune
Does not compile	81.91% (16545)	72.01% (14546)	76.35% (15422)	86.58% (17489)	6.75% (1363)
Not Neutral	8.69% (1756)	9.89% (1997)	20.69% (4180)	9.90% (1999)	85.97% (17365)
Neutral	8.83% (1783)	17.50% (3535)	2.96% (598)	3.52% (712)	7.15% (1444)
Repair	0.57% (116)	0.60% (122)	0.00% (0)	0.00% (0)	0.14% (28)
Positive Epistasis	0.00% (0)	0.05% (11)	0.01% (2)	0.00% (0)	0.12% (24)
Negative Epistasis	0.00% (0)	0.05% (11)	0.00% (0)	0.00% (0)	0.00% (0)

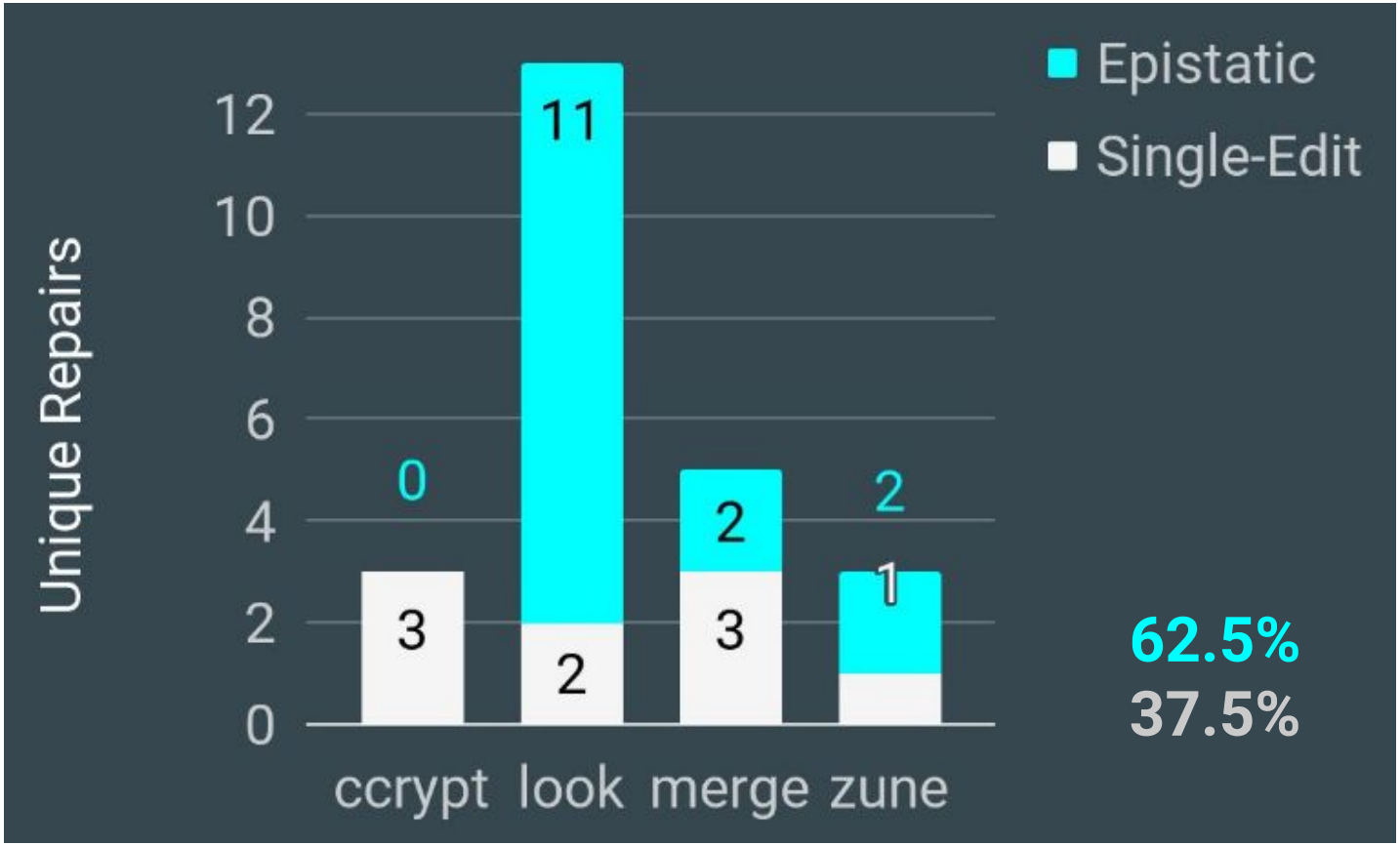
1 Million Mutations to *merge*

1 **Million** Mutations to *merge*

	single edits	double edits
Does not compile	60.04% (2991)	74.36% (739988)
Not Neutral	26.53% (1300)	23.44% (233290)
Neutral	12.37% (606)	2.18% (21719)
Repair	0.06% (3)	0.01% (103)
Positive Epistasis	N/A	0.00% (30)
Negative Epistasis	N/A	0.00% (13)

Table 3: We report results for 1,000,000 mutants of `merge`. All of the possible single-edit mutations (4,900) and approximately 4% (995,100) of the possible double-edit mutations were evaluated. Data are reported as the percentage of all mutations to that program which fall into the category described by the row labels (with raw counts in parentheses).

Unique repairs



Unique repairs

	ccrypt	look	merge	zune	total
Single Edit	3	2	3	1	37.50% (9)
Epistasis	0	11	2	2	62.50% (15)

Table 4: Unique repairs to `ccrypt`, `look`, `merge`, and `zune`. Data for `merge` represents 1,000,000 mutants, and `units` was omitted since no repairs were found. Data for the remaining programs represents 20,200 mutants. The rightmost column reports the percent of total unique mutations that were a result of single edits and epistasis.

Summary of results

- Most of the discovered single-edit repairs are functional duplicates. They contribute little to the diversity of solutions discovered;
- 62.50% of the unique repairs we found were the result of epistasis, and most of these (11 of 15) contained at least one non-neutral edit;
- Epistasis are:
  - rare in sample space;
  - common for repairs;
  - **majority of unique repairs**.
- Neutral Networks are iteratively traversable and exposes a diverse set of solutions

Future Work

- How many edits can we make and still be a neutral network?
- Are there optimal strategies to traverse neutral networks?
  - We have not yet formalized the mutation operators in a way that allow us take advantage of all this theory. Reversibility is a common assumption.
- Can we relate distance in neutral space to uniqueness of repair?
  - Challenge: how to categorize repair “types”
- How to search neutral space efficiently?

Future Work

Our results suggest that some properties of evolved biological systems (neutral networks, mutational robustness, epistasis) are present in software.

We conjecture that these same properties are a key reason why tools like GenProg can search effectively for repairs using evolutionary computation.

We don’t yet understand how or if these properties relate to human evolvability of software.

Future Work

Method:

1. **Scrape** - open source repos (or use ManyBugs)
2. **Generate** - large (sampled) networks
3. **Compute** - how close human-generated repair is to network
4. **Compare** - human-generated patches to machine-generated patches

Or get patches to small problems from students

OBRIGADO!

Dúvidas ou sugestões?

deuslirio.junior@gmail.com