Neutral Networks Enable Distributed Search in Evolution

Joseph Renzullo, Stephanie Forrest, Melanie Moses

Department of Computer Science, University of New Mexico

1. Introduction

How can we efficiently search for repairs to bugs in computer programs?

■ Automated tools exist, but the search space is enormous

■ Evolving populations of biological organisms face a simiar search space problem

■ Exploiting characteristics of the search space is a promising direction

■ We investigate the structural relationship of randomly-generated, functionally-similar variants of computer programs

1.1 Terminology

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.

We focus on evolution as a distributed search process and how it uses ***neutral networks*** to produce complexity.

1.2 How We Mutate Programs

Our experiments use two different kinds of mutations to the **Abstract Syntax Tree** (AST) of a partially-compiled program:

■ Delete a randomly selected node (and its subtree if one exists) from the AST.

■ Copy a random node (and its subtree if one exists) in the AST to another random location.

2. Theoretical Biology

Evolution favors high robustness and high innovation networks.

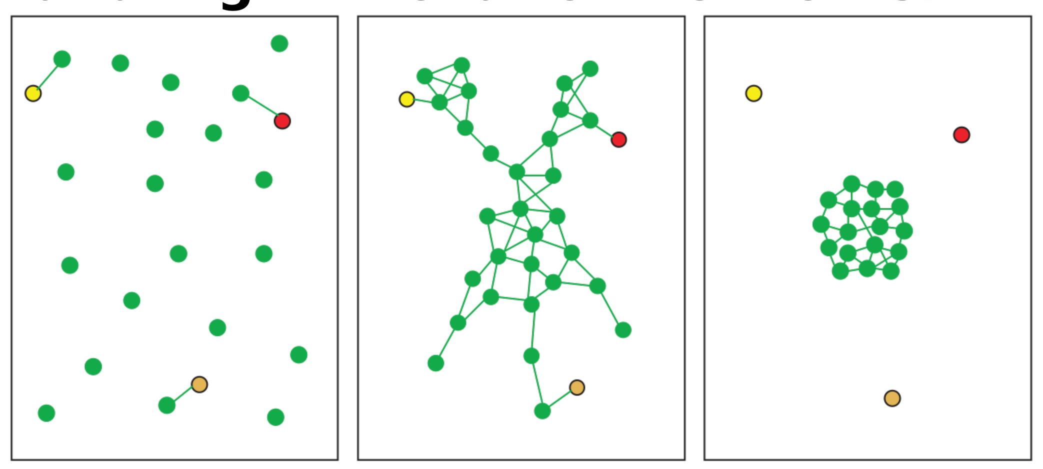

Figure 1 Reproduced from Ciliberti, Stefano, Olivier C. Martin, and Andreas Wagner. "Innovation and robustness in complex regulatory gene networks." *Proceedings of the National Academy of Sciences* 104.34 (2007): 13591-13596.

A neutral network can enable a **safe** search for innovations:

■ If not robust, single mutations often do not maintain fitness (**left**)

■ If robust enough, exploration by single mutations is possible (**center**) while maintaining existing fitness

■ If too robust, mutations cannot change fitness (**right**)

3. Mutating UNIX **`look`**

Each node in the graph to the right is a mutated variant of UNIX **look**, a dictionary lookup program with ~1000 lines of code.

Each variant passes all **test cases** (input/output pairs with known answers) the original program passed.

We call variant programs with identical test case behavior **neutral** variants.

We investigated the local neighborhood around the original program in order to determine the viability of moving around the neutral network of a computer program.

4. Locating repairs to a bug in UNIX **`look`**

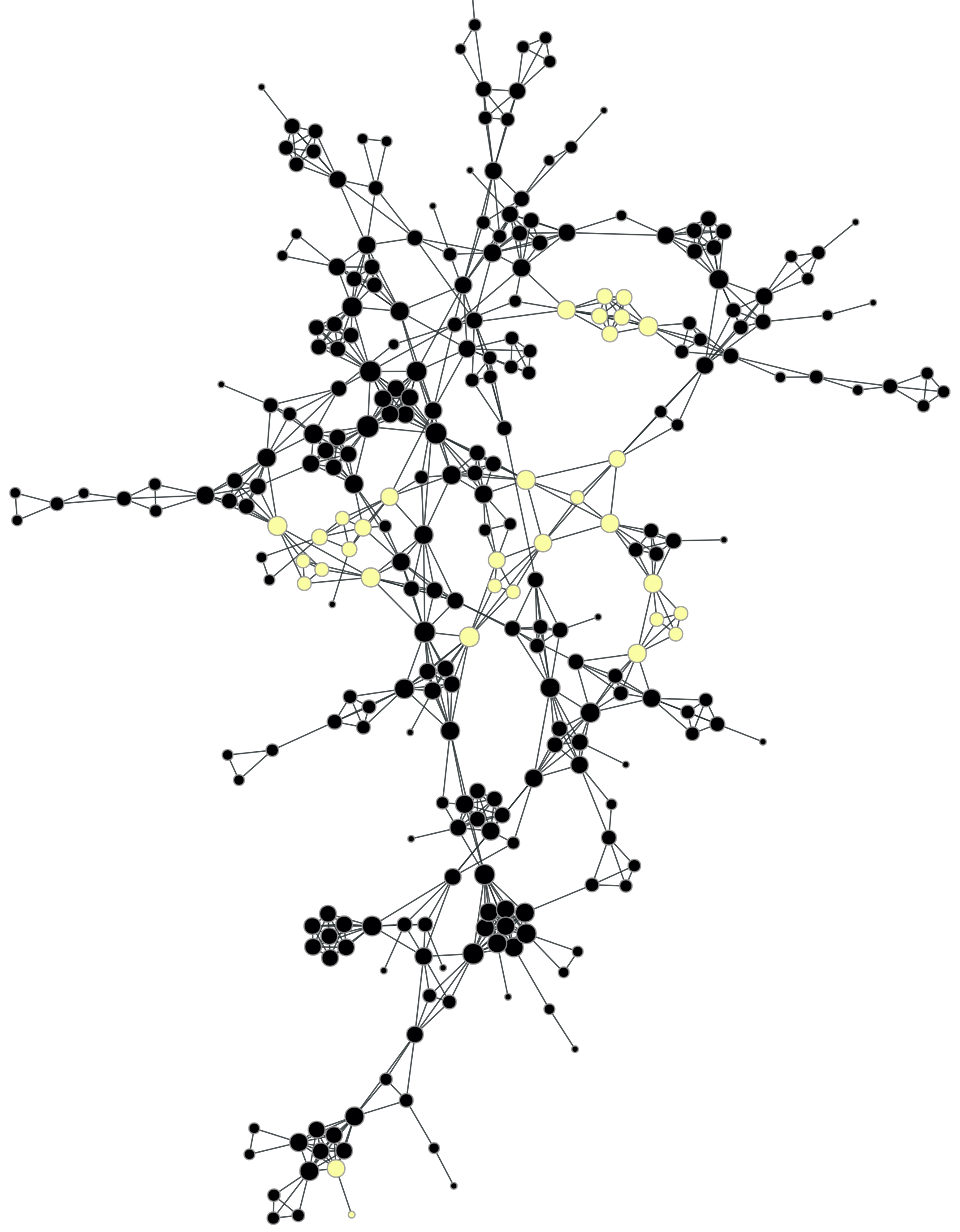

Figure 2 Clustering of program variants which repair a bug in UNIX look. The structure is potentially exploitable by search algorithms. It is *prima facie* similar to the theoretically-optimal structure.

Nodes in black retain the original program's behavior - they have the bug

Nodes in pale yellow repair the bug

Repaired programs often appear in clusters.

Implications:

This topology may inform strategies for automatic repair search.

It should be possible to sample less exhaustively and still hit clusters.

5. Differentiating sources of repair

We explored a broader region around the original program, and we varied the intensity of the search.

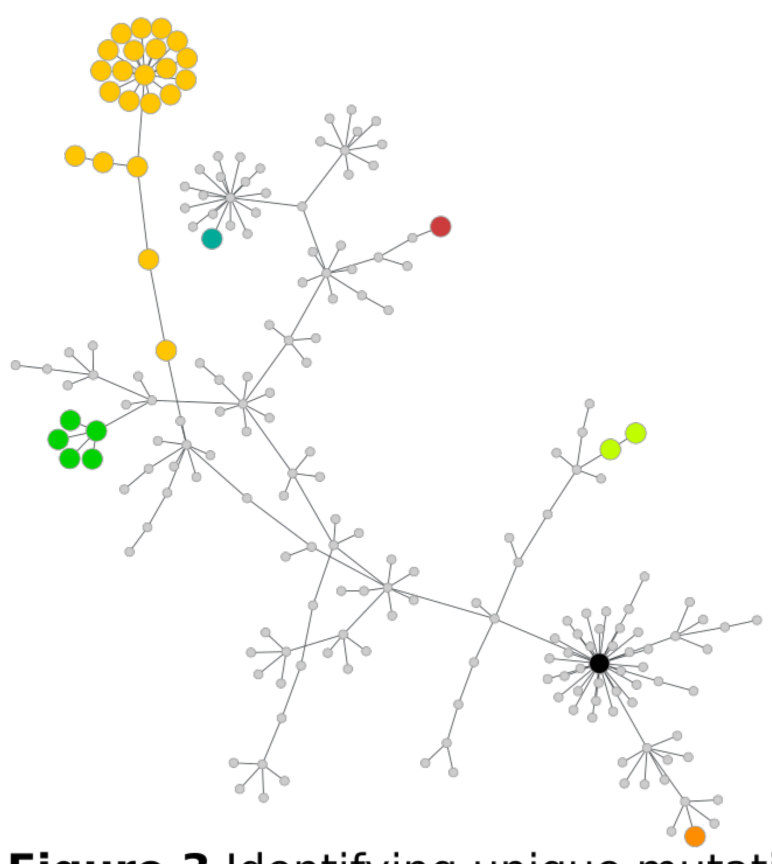

Figure 3 Identifying unique mutations which repair a bug in UNIX look.

■ The original program is black
■ Neutral variants are grey
■ Variants which repair the bug are various colors

Each color represents a different single edit responsible for the repair.

We find diverse mutations which repair

This work was supported by a James S. McDonnell Foundation Complex Systems Scholar Award. The authors gratefully acknowledge the partial support of NSF (1518878,1444871), DARPA (FA8750-15-C-0118), AFRL (FA8750-15-2-0075), the Sandia National Laboratories Academic Alliance, and the Santa Fe Institute.