Comparing Search Techniques for Finding Subtle Higher
Order Mutants

Elmahdi Omar, Sudipto Ghosh, and Darrell Whitley
Dept. of Computer Science, Colorado State University
Fort Collins, Colorado, USA
Elmahdi.omar@colostate.edu, ghosh@cs.colostate.edu, whitley@cs.colostate.edu

ABSTRACT
Subtle Higher Order Mutants (HOMs) are those HOMs that
cannot be killed by existing test suites that kill all First
Order Mutants (FOMs) for the program under test. Subtle
HOMs simulate complex, real faults, whose behavior cannot
be simulated using FOMs. However, due to the coupling
effect, subtle HOMs are rare in the exponentially large space
of candidate HOMs and they can be costly to find even for
small programs.

In this paper we propose new search techniques for finding
subtle HOMs and extend our prior work with new heuristics
and search strategies. We compare the effectiveness of six
search techniques applied to Java and AspectJ programs.

Our study shows that more subtle HOMs were found when
the new heuristics and search strategies were used. The
programming language (Java or AspectJ) did not affect the
effectiveness of any search technique.

Categories and Subject Descriptors
D.2.5 [Software Engineering]: Testing and Debugging

General Terms
Algorithms, Experimentation

Keywords
Higher order mutation testing, search-based software engi-
nineering, software testing, genetic algorithm, local search

1. INTRODUCTION

The majority of traditional First Order Mutants (FOMs)
represent trivial faults that are often easily detected. Thus,
researchers use higher order mutation testing to identify the
combinations of single faults that represent complex faults.
Jia and Harman [4] introduced the notion of Strongly Sub-
suming HOMs, which represent the HOMs that can replace
their constituent FOMs without loss of test effectiveness.

Permission to make digital or hard copies of all or part of this work
for personal or classroom use is granted without fee provided that copies are not
made or distributed for profit or commercial advantage and that copies bear
this notice and the full citation on the first page. Copyrights for components
of this work owned by others than ACM must be honored. Abstracting with
credit is permitted. To copy otherwise, or republish, to post on servers or to
redistribute to lists, requires prior specific permission and/or a fee. Request
permissions from Permissions@acm.org.

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
Copyright 2014 ACM 978-1-4503-2662-9/14/07 ...$15.00.
http://dx.doi.org/10.1145/2576768.2598286.

and thus reduce test effort. Omar et. al. [8] introduced the
notion of subtle HOMs, which represent the HOMs that are
not killed by test suites that kill all FOMs for the program
under test. Subtle HOMs denote cases where single faults
interact (e.g., by masking each other) to produce complex
faulty behavior that cannot be simulated using FOMs, and
thus they can be used to improve test effectiveness.

Purushothaman and Perry [10] showed that the majority
of real faults not detected during testing are complex faults
that cannot be simulated using FOMs. Their study reported
that fixing 90% of these complex faults required making mul-
tiple changes to the source code. Because HOMs are created
by making multiple syntactic changes to the source code,
they can simulate such complex faults.

Subtle HOMs can be used as the basis for generating new
test inputs to improve the fault detection effectiveness of
existing test suites. However, the cost of finding subtle
HOMs can be high even for small programs. This is be-
cause the number of candidate HOMs is exponentially large
and a large majority of them are killed by test suites that
kill all FOMs (coupling effect) [4, 6, 12], making the subtle
ones relatively rare.

In previous work [8], we presented three search techniques,
Genetic Algorithm, Local Search, and Random Search, for
finding subtle HOMs. We also conducted an initial evalu-
tion to demonstrate the ability of the search techniques to
find subtle HOMs. Although all the techniques were able to
find subtle HOMs, we were able to identify ways in which to
improve their effectiveness based on our analysis of the gen-
erated HOMs and how each technique explored the search
space of candidate HOMs.

In this paper, we introduce three improved search tech-
niques, Guided Local Search, Restricted Random Search,
and Restricted Enumeration Search for finding subtle HOMs.
Guided Local Search uses a heuristic to focus on the FOM
combinations that are more likely to produce subtle HOMs.
Restricted Random Search uses a configurable parameter to
limit the search to a region of the search space where subtle
HOMs are more likely to be found. Restricted Enumeration
Search enumerates all candidate HOMs in a predefined se-
dence starting in the region of the search space where more
subtle HOMs are expected to be found.

In our previous study [8], we only used AspectJ programs
to demonstrate the ability of the search techniques to find
subtle HOMs. In this paper, we also included Java to deter-
mine whether or not differences in programming language
constructs resulted in different levels of effectiveness.
2. SEARCH TECHNIQUES FOR FINDING SUBTLE HOMs

We first describe the objective function used by the search techniques to measure the fitness of HOMs. We summarize the previously introduced search techniques and describe the new ones. Omar et al. [8] provides a more detailed description of the previously proposed search techniques. We also briefly describe the prototype tool called HOMAJ [9] for higher order mutation testing for Java and AspectJ programs.

2.1 Objective Function

The objective function needs to identify the HOMs that have a better chance of becoming subtle HOMs if a mutated statement is added or removed. Below we provide a formal definition of the objective function.

- $F = \{f_1, \ldots, f_l\}$ is the set of all non-equivalent FOMs for the program under test.
- $H$ is the space of all candidate HOMs. $H = \mathcal{P}(F)$.
- $U$ is the universe of all possible test cases.
- $T = \{tc_1, \ldots, tc_l\}$ is the set of all test cases under consideration (the given test suite), $T \subset U$ and $T$ kills all the FOMs in $F$.
- $h_i^n \in H$ is an HOM constructed from $n$ FOMs, such that $h_i = \{f_{i_1}, \ldots, f_{i_n}\}$. The notation can be simplified to $h_i = h_i^1$ without confusion.
- Let $T_{h_i} \subseteq T$ denote the set of those test cases in $T$ that kill $h_i$. $T_{h_i} = \emptyset$ if none of the test cases in $T$ kill $h_i$.
- There are $n$ test sets $T_{i_1}, \ldots, T_{i_n}, \forall j \in [1, \ldots, n], T_{i_j} \subseteq T$ and $T_{i_j}$ contains all test cases that kill $f_{i_j}$ in $h_i$.
- $TU_i$ is a test set such that

$$TU_i = \bigcup_{j=1}^{n} T_{i_j}$$

Objective Function

$$fitness(h_i) = \frac{|(TU_i \cup T_{h_i})| - |(TU_i \cap T_{h_i})|}{|TU_i \cup T_{h_i}|} + sublety(h_i)$$  \hspace{1cm} (1)

Subtlety Function

$$sublety(h_i) = \left\{ \begin{array}{ll} 1 & \text{if} \quad T_{h_i} = \emptyset \\ 0 & \text{otherwise} \end{array} \right.$$  \hspace{1cm} (2)

To calculate the fitness of an HOM, the objective function first measures the fault detection difference between the HOM and its constituent FOMs (first term in Equation 1) and then measures its subtlety. The fault detection difference quantifies the variation between the set of test cases that kill the HOM and the union of all sets of test cases that kill the individual constituent FOMs. The subtlety value shows whether the HOM is killed by the given test set or not.

The fitness value of an HOM lies between 0 and 2. Note that a solution which has an evaluation of 2 represents a global optimum in the search space, and there are potentially many globally optimal solutions. HOMs with higher fitness values are favored in the selection process. FOMs have zero fitness value. HOMs are classified based on their fitness values as follows:

1. ** Entirely Coupled HOMs** have a fitness value of 0 because there is no difference between the set of test cases that kill the HOM and the union of all sets of test cases that kill the individual constituent FOMs.
2. **Partially Coupled HOMs** have a fitness value greater than 0 but less than 1 because the set of test cases that kill the HOM and the union of all sets of test cases that kill the individual constituent FOMs are not identical but they overlap.
3. **Decoupled HOMs** have a fitness value of 1 because they are killed by a set of test cases that is totally different from the union of all sets of test cases that kill the individual constituent FOMs.
4. **Subtle HOMs** have a fitness value of 2 because they are not killed by any test case in the given test set. They represent new faulty behavior that has not been tested.

2.2 Genetic Algorithm

The Genetic Algorithm (GA) [8] takes as input the list of FOMs for the program under test along with number of crossover points, mutation rate, population size, degree of FOMs for the program under test, and the number of elite HOMs that get carried over at each iteration. Each chromosome is represented as a one dimensional array of strings such that each element in the array represents a line of code (one Java/AspectJ statement) of the program under test.

The Genetic Algorithm uses tournament selection to implement generational replacement of the population. A certain number of HOMs with the highest fitness values are automatically carried over (copied) to the next generation. The crossover happens between two selected HOMs (parents) that are recombined to produce two HOMs (offspring).

The Genetic Algorithm mutation operator applies mutation to an existing HOM by either adding or removing an FOM. The FOMs to be added to an HOM are randomly selected from the list of FOMs for the program under test and the FOMs to be removed from an HOM are randomly selected from the HOM’s constituent FOMs. All HOMs in the population are created, compiled, and executed, and then the objective function evaluates their fitness.

2.3 Local Search

The Local Search (LS) [8] technique takes as input the list of FOMs and the degree of the incumbent HOM. The incumbent HOM represents the search starting point. Local Search starts by generating an incumbent HOM using a number of randomly selected FOMs from the list of FOMs. The maximum degree of the incumbent HOM is set using a configurable parameter. After evaluating the incumbent HOM, Local Search generates all the HOMs neighboring the incumbent HOM. The neighboring HOMs are those that vary by one FOM (one step) from the incumbent HOM. A list of neighboring HOMs is maintained.
After creating and evaluating all HOMs in the neighborhood list, Local Search looks for the best neighboring HOM that has an equal or higher fitness value than the incumbent HOM. If such an HOM exists, it becomes the next incumbent HOM, and the process starts all over again. If no better HOM exists, Local Search restarts by selecting a new incumbent HOM.

2.4 Guided Local Search

The Guided Local Search (GLS) technique introduced in this paper uses the same steps as Local Search. However, Guided Local Search utilizes program structural information to help it focus on the FOM combinations that are more likely to produce subtle HOMs. When an incumbent HOM is selected and evaluated, Guided Local Search generates a smaller set of neighboring HOMs than Local Search. Guided Local Search aims to avoid creating and evaluating HOMs that are expected to be entirely coupled to their constituent FOMs.

During the early empirical studies that set the foundation for this work, we found that the majority of HOMs that have a fitness value greater than zero (partially coupled, decoupled, and subtle HOMs) are those where the mutated statements shared a common variable. For example, for the Kettle Program which is used in the empirical evaluation in this paper, we found that in 96% of HOMs with fitness value greater than zero, at least two of their mutated statements shared at least one common variable. On the other hand, 60% of the HOMs with a fitness value equal to zero, their mutated statements shared some common variables. Although the presence of a common variable among the mutated statements in the HOM cannot guarantee that the HOM is not entirely coupled, the initial data we obtained for most subject programs motivated us to utilize this knowledge in the search process.

Guided Local Search explores only neighboring HOMs where at least two of their mutated statements share at least one common variable. That is, there exists at least one pair of mutated statements where both statements read and/or write to at least one program variable. In addition, Guided Local Search maintains a list of all second order mutants that were found and evaluated during the search process. This list contains the fitness values for the evaluated second order mutants. Guided Local Search uses this list to further refine and minimize the list of neighboring HOMs by ignoring all HOMs that contain pairs of mutated statements that share a common variable when their combination was found to result in an entirely coupled second order mutant.

2.5 Random Search

The Random Search (RS) [8] technique explores the space of all candidate HOMs by randomly selecting HOMs, one at a time, seeking out the HOMs that are not killed by the given test suite. Random Search iterates the process of generating an HOM with a set of randomly selected FOMs from the list of FOM. The degree of a generated HOM is randomly selected to allow Random Search to examine any candidate HOM in the search space. Each generated HOM is compiled and executed, its execution result is recorded, and its fitness value is calculated.

2.6 Restricted Random Search

The Restricted Random Search (RRS) technique introduced in this paper performs the same steps as the Random Search. However, in our previous study we found most of the discovered subtle HOMs were of lower degrees (less than six). Therefore, Restricted Random Search uses a configurable parameter to allow it to control the maximum degree of generated HOMs. This parameter allows Restricted Random Search to limit the search to a smaller part of the search space where subtle HOMs are more likely to be found.

2.7 Restricted Enumeration Search

The Restricted Enumeration Search (RES) technique introduced in this paper examines candidate HOMs in the search space in a predefined sequence until a defined stopping condition is met. The Restricted Enumeration Search takes as input the list of FOMs for the program under test, and then it starts by creating and evaluating mutants one at a time. It creates and evaluates all candidate second order mutants, then third order mutants and so on until a stopping condition is met.

The Restricted Enumeration Search is designed to start the search in the space of lower degree HOMs because of two reasons. First, because the space of HOMs grows exponentially (at least to a certain degree), the space of lower degree HOMs is smaller and it is easier to find subtle HOMs. Second, more subtle HOMs exist in space of lower degree HOMs because adding more faults to an HOM in general makes it easier to kill.

2.8 Prototype Tool HOMAJ

We developed a prototype tool HOMAJ: Higher Order Mutation for AspectJ and Java [9]. HOMAJ automates the process of generating, compiling, and executing both first order and higher order mutants. It takes as an input the program under test along with the test suite. It then generates Java and AspectJ FOMs, and compiles and executes them against the given test suite. It starts the search process based on the selected search technique seeking out FOM combinations that are not killed by the given test suite.

The stopping condition for the search process is configured by the tester. Any of the search techniques can be stopped if it reaches the maximum number of HOMs it is allowed to generate, finds a predefined number of found subtle HOMs, or reaches a specified time interval. When the search process stops, HOMAJ presents a list of subtle HOMs that were found during the search process. A subtle HOM can possibly be an equivalent HOM. Testers need to manually inspect the subtle HOMs and identify equivalent HOMs.

The current implementation of HOMAJ allows one fault per statement. Therefore, the maximum degree of generated HOMs is restricted by the number of statements that can be mutated for the program under test.

3. EXPERIMENTAL EVALUATION

This section describes the goals of the experimental evaluation of the search techniques.

3.1 Goals

The goal was to evaluate and compare the ability of search techniques to find subtle HOMs. Therefore, we ran all the techniques with the stopping condition of generating and
exploring a fixed number of HOMs. At the end of the execution, we compared the total number of distinct subtle HOMs found by each technique.

We also investigated how the search techniques explored the search space and how that affected their effectiveness. We investigated the growth in the number of subtle HOMs found by each search technique as the number of generated HOMs increased. We also investigated the degrees and the number of distinct HOMs generated by each technique.

### 3.2 Subject Programs

We used four AspectJ programs and two Java programs of different sizes. We chose AspectJ [11] programs because of the unique concepts and constructs introduced by the Aspect-Oriented Programming (AOP) paradigm. The unique constructs introduced new types of interactions between the program elements which produced new testing challenges [1, 13]. The AspectJ programs varied in size and implemented various Java and AspectJ constructs. They contained before, after, and around advices, inter-type declarations, as well as primitive and composed pointcuts. We also used two Java programs to find out if the search techniques produce different results when programs do not implement AspectJ concepts and constructs.

For each subject program, we used a test suite that covered all the statements and killed all non-equivalent FOMs. Table 1 provides information about the subject programs, their sizes, size of their test suites, and number of equivalent and non-equivalent FOMs. The Java programs have only classes and no aspects. All equivalent FOMs were manually identified and they are not included in the search process for subtle HOMs.

### 3.3 Experimental Configuration

We used multiple 64-bit Linux machines with Intel Core™-4x3.3G and 8Gb memory. The configuration used for each technique can affect its performance. In the early stages of this study we ran the search techniques with different configurations and finally selected the configurations that produced the highest number of subtle HOMs. The search techniques were configured the same for all subject programs as follows. For Genetic Algorithm, the number of crossover points was set at two, number of elite HOMs at 15, population size at 300, mutation rate at 0.1 of the number of mutable statements of the program, and first population degree at two. For both Local Search and Guided Local Search, the incumbent HOM degree was set at two. For Restricted Random Search, the maximum HOM degree allowed was set at six. Both Random Search and Restricted Enumeration Search did not have any configured parameters.

### 4. RESULTS AND ANALYSIS

To compare the ability of the search techniques to find subtle HOMs, we ran five techniques (Genetic Algorithm, Local Search, Restricted Random Search, Random Search, and Guided Local Search) 30 times per subject program with the stopping condition of generating 50,000 HOMs. Restricted Enumeration Search was run only once per subject program because it produces the same result each time. Table 2 shows the maximum, average, median, standard deviation, and minimum number of subtle HOMs found for the 30 runs per technique. Equivalent HOMs, which are treated as subtle HOMs by our objective function, are manually identified and removed. Omar et al. [8] provides more details about the treatment of equivalent HOMs.

For each subject program, all six techniques found subtle HOMs. However, Guided Local Search, Local Search, and Restricted Enumeration Search were more successful in finding the highest number of subtle HOMs than the other search techniques for all subject programs. Restricted Enumeration Search was more successful for Movie Rental and Roman programs, while Guided Local Search was more successful for all other programs.

Restricted Random Search produced better results than Random Search, which indicates that limiting the search to the space of lower degree HOMs is a good strategy because it produced a higher number of subtle HOMs. In fact, for most programs, Restricted Random Search found almost as many subtle HOMs as the Genetic Algorithm, which uses different sophisticated operators to find subtle HOMs. Although Random Search was the least successful technique because it considered every candidate HOM in the large search space at each iteration, it still managed to find subtle HOMs, which indicates subtle HOMs can be relatively easy to find.

To investigate these observations further, we analyzed the results produced by each technique. First we plotted the growth in the number of subtle HOMs over the number of generated HOMs. Figures 1 through 6 show how the average number of found subtle HOMs grew as each technique generates more HOMs. The number of generated HOMs is considered a quasi-representation of the time taken by each algorithm.

Second, we investigated the percentage of distinct HOMs generated by each technique as shown in Table 3. For example, only 33.1% of all HOMs generated by Genetic Algorithm for Banking program are found to be distinct while 91.5% of all HOMs generated by Random Search were distinct. Restricted Enumeration Search examines each HOM once, which explains why all the HOMs were distinct.

Last, we investigated the degree of HOMs generated by each technique. Table 4 shows for each technique, the percentage of HOMs for different degrees. For example, 29% of all HOMs for the Banking program generated by Genetic Al-
Figure 1: Growth in the average number of subtle HOMs over the number of generated HOMs for Banking

Figure 2: Growth in the average number of subtle HOMs over the number of generated HOMs for Coordinate

Figure 3: Growth in the average number of subtle HOMs over the number of generated HOMs for Kettle

tgorithm were second order mutants. The number of second order mutants reported for Restricted Enumeration Search reflects the actual number of all candidate second order mutants. For example, there are 48,000 candidate second order mutants for the Movie Rental program (96% of 50,000). Some of the techniques can possibly generate first order mutants. For example, the Genetic Algorithm can produce an FOM when crossing over two HOMs or when mutating an HOM by removing a mutated statement. FOMs have zero fitness values and their percentages are not included in Table 4.

For the Banking program, Guided Local Search, Restricted Enumeration Search and Local Search found most of the subtle HOMs before generating 15,000 HOMs. Because the number of second order mutants for the Banking program is considerably low compared to the other programs (only 8% of 50,000 HOMs) and because the Restricted Enumeration Search examines all second order mutants before it moves to order-3 and so on, Restricted Enumeration Search was able to find a high number of subtle HOMs in the early stages of the search process. A similar pattern was observed for the Kettle program. In our previous study, the majority of the found subtle HOMs were of low degrees.

Except for the Random Search, all other techniques eventually found a comparable number of subtle HOMs for the Banking program. This indicates that subtle HOMs in general were easier to find for the Banking program. This could be because the Banking program had the smallest test set.
Guided Local Search and Local Search found a higher number of subtle HOMS for the Coordinate program than all other programs even though it was the smallest in terms of size. Both Local Search and Guided Local search eventually found a higher number of subtle HOMs than Restricted Enumeration Search. The numbers of subtle HOMs found using Genetic Algorithm, Random Search, and Restricted Random Search were low compared to other search techniques.

For the Kettle program, Restricted Enumeration Search was more successful than Guided Local Search and Local Search in the early part of the search process because the Kettle program had a low number of second order mutants (only 15% of 50,000 HOMs).

For the Movie Rental program, Restricted Enumeration Search was more successful throughout the search process. Movie Rental has the highest number of second order mutants (96% of 50,000 HOMs) and Restricted Enumeration Search examined all of them and thus, it was more successful overall in finding more subtle HOMs. A similar trend can be seen for the Roman program where second order mutants represent 41% of evaluated 50,000 HOMs.

For the Roman program, Restricted Enumeration Search eventually found a higher number of subtle HOMs than Guided Local Search and Local Search. However, both Guided Local Search and Local Search were more successful than Restricted Enumeration Search in the early stages of the search process.
### Table 2: Number of Subtle HOMs Found by Search Techniques

<table>
<thead>
<tr>
<th>Programs</th>
<th>Technique</th>
<th># Subtle HOMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banking</td>
<td>GA</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>LS</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>RRS</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>GLS</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>RES</td>
<td>20</td>
</tr>
<tr>
<td>Coordinate</td>
<td>GA</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>LS</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>RRS</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>GLS</td>
<td>136</td>
</tr>
<tr>
<td></td>
<td>RES</td>
<td>52</td>
</tr>
<tr>
<td>Kettle</td>
<td>GA</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>LS</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>RRS</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>GLS</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>RES</td>
<td>33</td>
</tr>
<tr>
<td>Movie Rental</td>
<td>GA</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>LS</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>RRS</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>GLS</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>RES</td>
<td>13</td>
</tr>
<tr>
<td>Roman</td>
<td>GA</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>LS</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>RRS</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>GLS</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>RES</td>
<td>23</td>
</tr>
<tr>
<td>Telecom</td>
<td>GA</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>LS</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>RRS</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>RS</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>GLS</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>RES</td>
<td>8</td>
</tr>
</tbody>
</table>

### Table 3: Percentage of Distinct HOMs generated by Search Techniques

<table>
<thead>
<tr>
<th>Programs</th>
<th>Percentage of Distinct HOMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banking</td>
<td>GA 33.1</td>
</tr>
<tr>
<td>Coordinate</td>
<td>GA 43.2</td>
</tr>
<tr>
<td>Kettle</td>
<td>GA 33.8</td>
</tr>
<tr>
<td>Movie Rental</td>
<td>GA 45.1</td>
</tr>
<tr>
<td>Roman</td>
<td>GA 43.7</td>
</tr>
<tr>
<td>Telecom</td>
<td>GA 40.1</td>
</tr>
</tbody>
</table>

For the Telecom program, Guided Local Search found a higher number of subtle HOMs than Restricted Enumeration Search during all stages of the search process. Genetic Algorithm and Restricted Random Search found less than half the number of subtle HOMs found by Local Search and Guided Local Search.

Restricted Enumeration Search by construction, can only generate distinct HOMs. Among all the other search techniques, Random Search generated the highest percentage of distinct HOMs. Although Restricted Random Search limits the search to a smaller part of the search space, it generated a higher percentage of distinct HOMs than the other search techniques. Although Guided Local Search generated the lowest percentage of distinct HOMs for most programs, it found the highest number of distinct subtle HOMs for all programs. This suggests that focusing the search on a specific set of FOM combinations can be a rewarding strategy. This also demonstrates the effectiveness of the heuristic implemented in Guided Local Search.

### Table 4: Percentage By Degree for HOMs Generated By Search Techniques

<table>
<thead>
<tr>
<th>Programs</th>
<th>Percentage of HOMs by Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banking</td>
<td>GA 29.22</td>
</tr>
<tr>
<td>Coordinate</td>
<td>GA 48.36</td>
</tr>
<tr>
<td>Kettle</td>
<td>GA 26.11</td>
</tr>
<tr>
<td>Movie Rental</td>
<td>GA 65.7</td>
</tr>
<tr>
<td>Roman</td>
<td>GA 23.21</td>
</tr>
<tr>
<td>Telecom</td>
<td>GA 16.7</td>
</tr>
</tbody>
</table>

### 5. RELATED WORK

The majority of the proposed Search-Based Software Engineering testing techniques targeted test data generation for fault discovery [3]. However, other than the work presented by Harman et al. [4, 5], which is briefly described below, none of the proposed Search-Based Software Engineering testing techniques tackled similar testing goals as the techniques presented in this paper.
Jia and Harman [4] classified HOMs in terms of their coupling and subsumption relations with FOMs and introduced three search-based optimization approaches (Genetic Algorithm, Greedy Algorithm, and Hill Climbing Algorithm) for finding Strongly Subsuming HOMs in C programs. Unlike subtle HOMs, strongly subsuming HOMs aim to reduce testing effort and cost by reducing the total number of FOMs that need to be compiled and executed without loss of test effectiveness. Jia and Harman [4] conducted an experimental evaluation using 10 C programs. The results showed that subsuming HOMs existed in all studied programs. Although the proportions of strongly subsuming HOMs were low, the actual numbers were large. The results indicated that the Genetic Algorithm performed best in finding strongly subsuming HOMs because the subsuming HOMs were easier to generate from existing subsuming HOMs.

Langdon et al. [5] used the multi-objective Pareto optimal approach and Monte Carlo sampling Genetic Algorithms to search for HOMs that are hard-to-kill and are syntactically similar to the original program. The search algorithm produced HOMs that were harder-to-kill than their constituent FOMs and also HOMs that were constructed from single faults masking each other.

DiGiuseppe and Jones [2] presented a study of the effects of the interaction of different faults within a program. The results showed that multiple real faults could hide the impact of each other leading to fewer failures than expected. They also reported that fault obfuscation was the most prominent of all the fault interaction types.

Wedyan and Ghosh [13] presented a study of mutation testing for AspectJ programs. In the subject programs they studied, the pointcut mutants that match some of the intended join points and some of the unintended join points at the same time could only be obtained with the help of HOMs.

Omar and Ghosh [7] presented an exploratory study of higher order mutation testing in the context of AspectJ programs. They proposed four approaches to constructing higher order mutants in AspectJ programs and evaluated the approaches in terms of their ability to create higher order mutants that result in higher test effectiveness and lower test effort compared to first order mutants.

Although Guided Local Search produced the lowest percentage of distinct HOMs for most programs, it was more effective than Local Search and found the highest numbers of subtle HOMs for all the programs that were used. This demonstrates the effectiveness of the proposed heuristic and motivates us to investigate the observations further.

In the future, we will investigate the impact of different control and data flow criteria on the process of finding subtle HOMs. We will also investigate whether information about the test cases that kill each FOM can be used to help the search techniques focus on the FOM combinations that are more likely to produce subtle HOMs. We will develop new heuristics and investigate their impact on increasing the effectiveness of the search techniques. Further research is needed to develop test input generation techniques that can target subtle HOMs.

7. REFERENCES