Constructing Subtle Higher Order Mutants from Java and AspectJ Programs

Elmahdi Omar, Sudipto Ghosh and Darrell Whitley
Department of Computer Science
Colorado State University

Problems with Mutation Testing

A fault-based testing technique that help testers measure and improve the ability of test suites to detect faults

Majority of First Order Mutants (FOMs) represent trivial faults that are often easily detected [Jia and Harman 2008]

Real faults are complex

- A large majority of real faults cannot be simulated with FOMs [Purush. and Perry 2005]
- A typical real fault involves about three to four tokens [Gopinath et al 2014]

Higher Order Mutants (HOMs) can be used to simulate real and complex faults

Subtle Higher Order Mutants

HOMs that are not killed by an existing test suite that kills all the FOMs of a given program

Can help researchers and practitioners gain a better understanding of the nature of faults and their interactions

Subtle Higher Order Mutants…

Can be costly to find:

- The search space of HOMs is (exponentially?) large
- *Coupling effect* makes subtle HOMs rare
- High computational cost of evaluating mutants
  - Involves compilation and execution

Contributions

Developed search techniques for finding subtle HOMs

- Search-based software engineering techniques
- Random search technique
- Enumeration search technique

Automated the process of finding subtle HOMs

- Developed a Higher Order Mutation Testing tool for AspectJ and Java programs (HOMAJ)

Performed a set of empirical studies

- Evaluated the relative effectiveness of the developed search techniques
- Investigated different factors that impact the creation of subtle HOMs

Objective Function

Provides a metric to measure the quality of HOMs

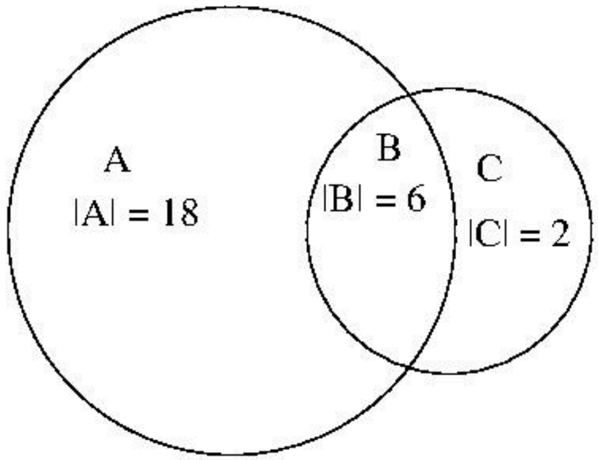
- *fitness* (*HOM*) = $\alpha *$ difficulty of killing (HOM) + $(1-\alpha) *$ fault detection difference (*HOM*)

Classifies HOMs Based on their fitness value as follow:

fitness (*HOM*) =

- 0 -> Entirely Coupled HOMs
- 0 < & < 1 -> Promising HOMs
- 1 -> Subtle HOMs (optimal solutions)

Objective Function

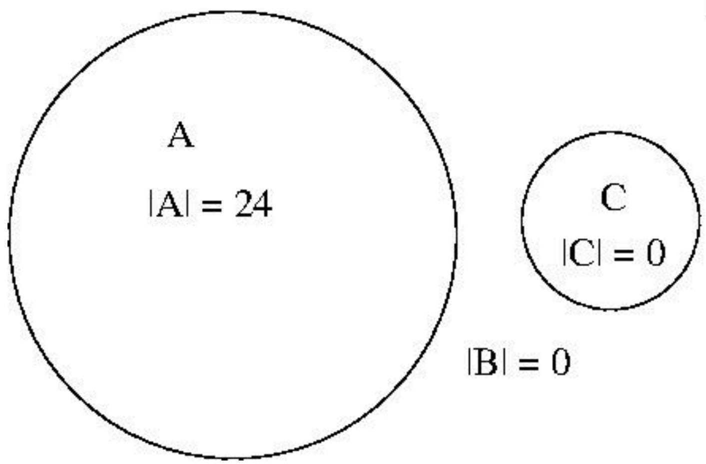


DFF = 20/26

DOK = 18/26

$$\mathrm{DOK} = \frac{|A|}{|A| + |B| + |C|}$$

$$\mathrm{DFF} = \frac{|A| + |C|}{|A| + |B| + |C|}$$



DFF = 24/24

DOK = 24/24

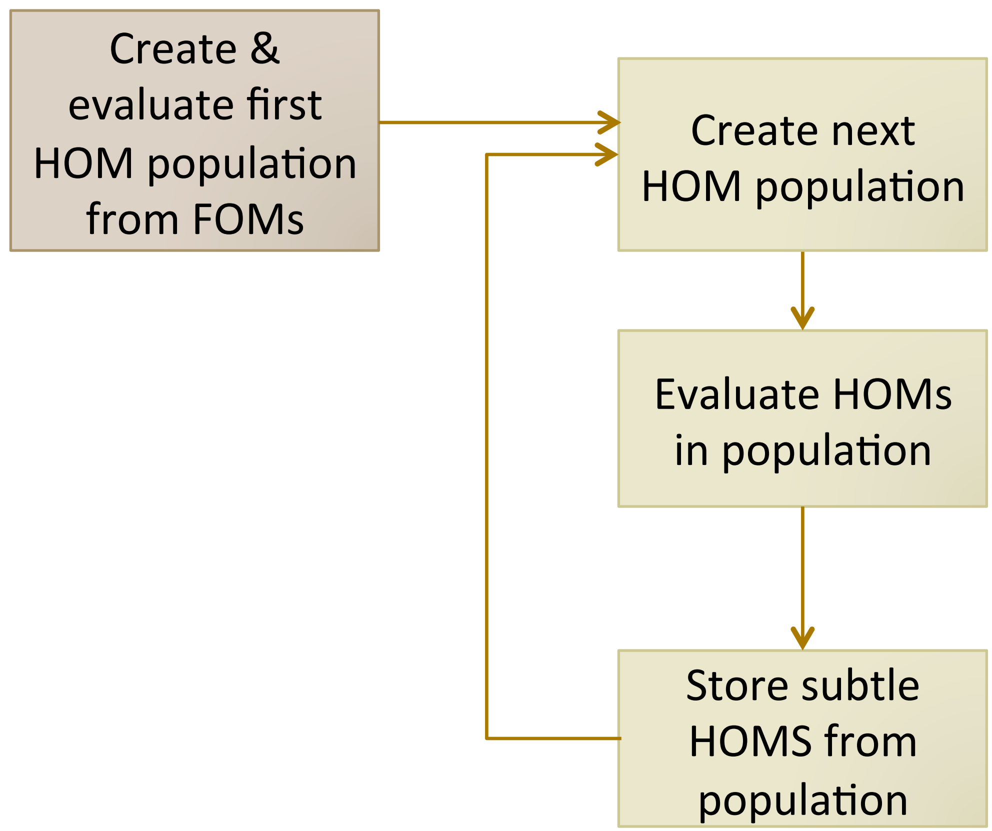
Objective Function

Note that every HOM not killed by the test set is a globally optimal solution.

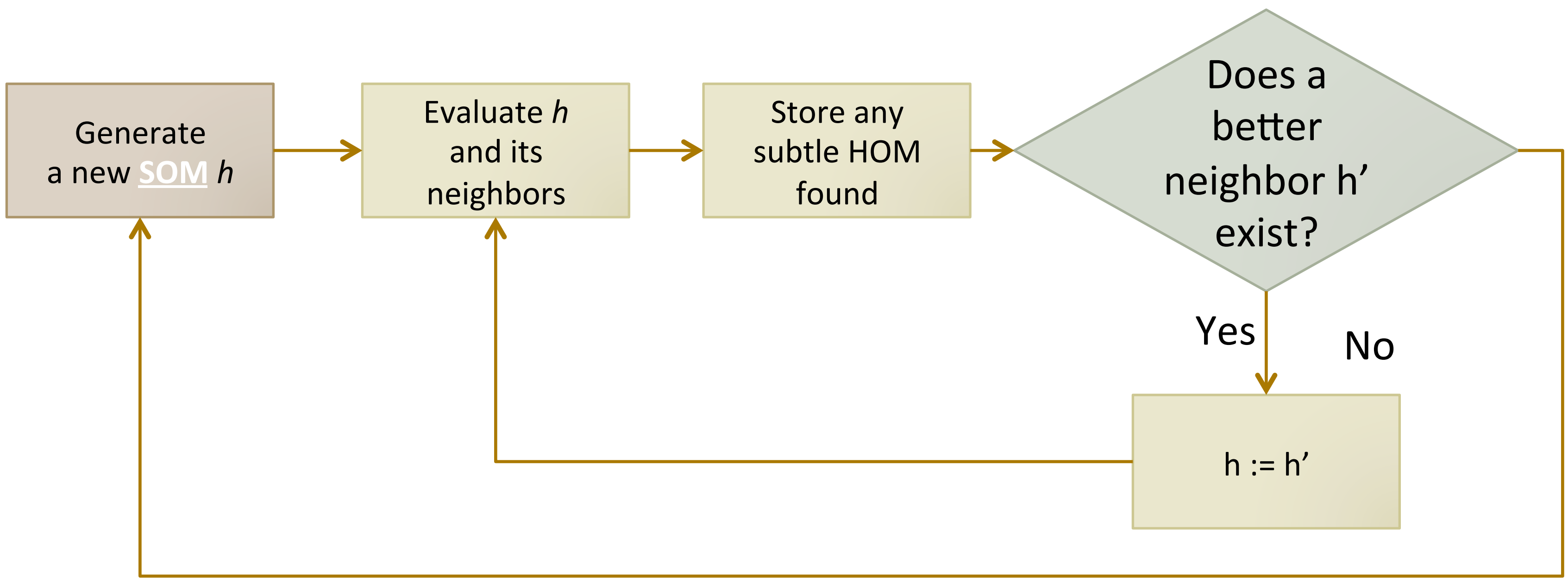
So we are looking for all (or many) globally optimal solutions.

This is different than many other types of objective functions.

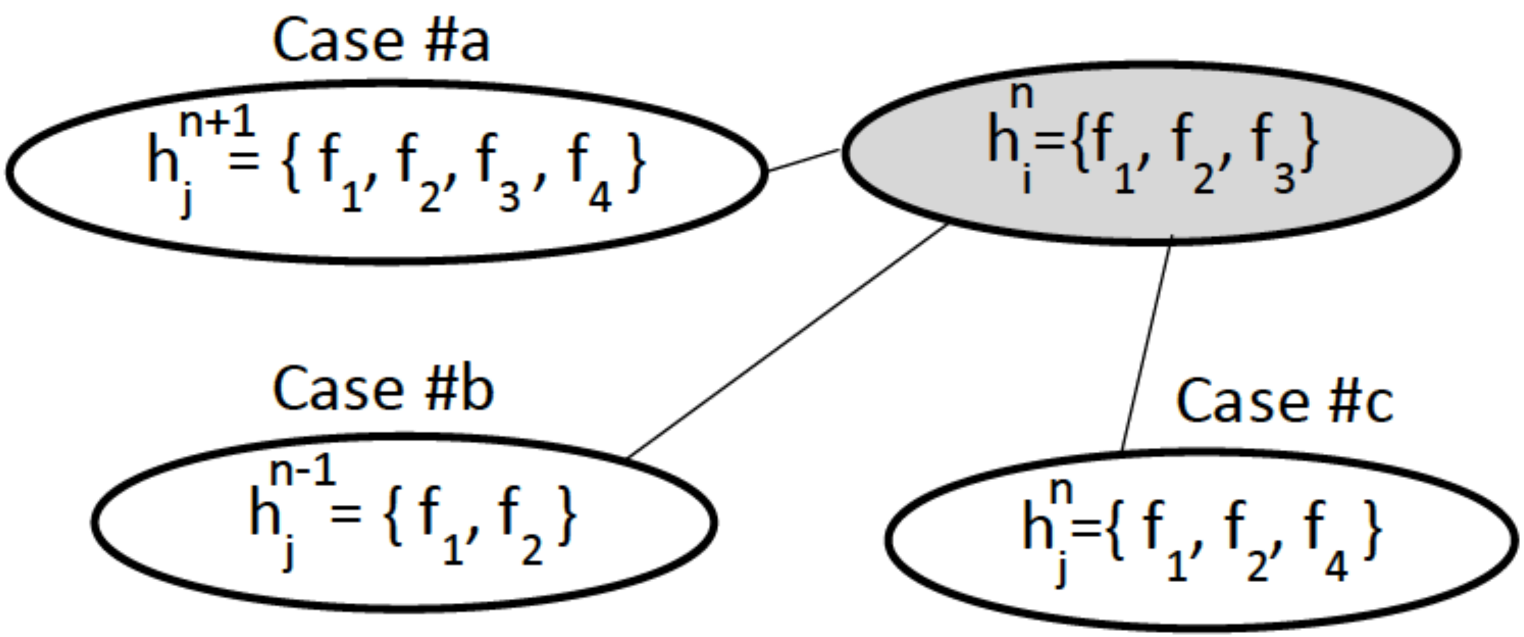
Genetic Algorithm



Local Search



Local Search



Data-Interaction Guided Local Search

Explores only neighboring HOMs that their mutated statements access the same variable(s)

Example:

Mutation (fom_1)= { return movieType; => return movieType**++**; }

Mutation (fom_2)= { if (movieType == “C” => if (movieType **!=** “C”) }

Mutation (fom_3)= { if (custName.equals(name) => if (**!** custName.equals(name) }

Considered HOMs = {(fom_1,fom_2), (fom_1,fom_2,fom_3))

Discarded HOMs = {(fom_1,fom_3), (fom_2,fom_3))

Test-Case Guided Local Search

Explores only neighboring HOMs that their constituent FOMs are killed by similar/common test cases

Example:

KilledBy(fom_1)= { tc_1, tc_3, tc_{13}, ….. }

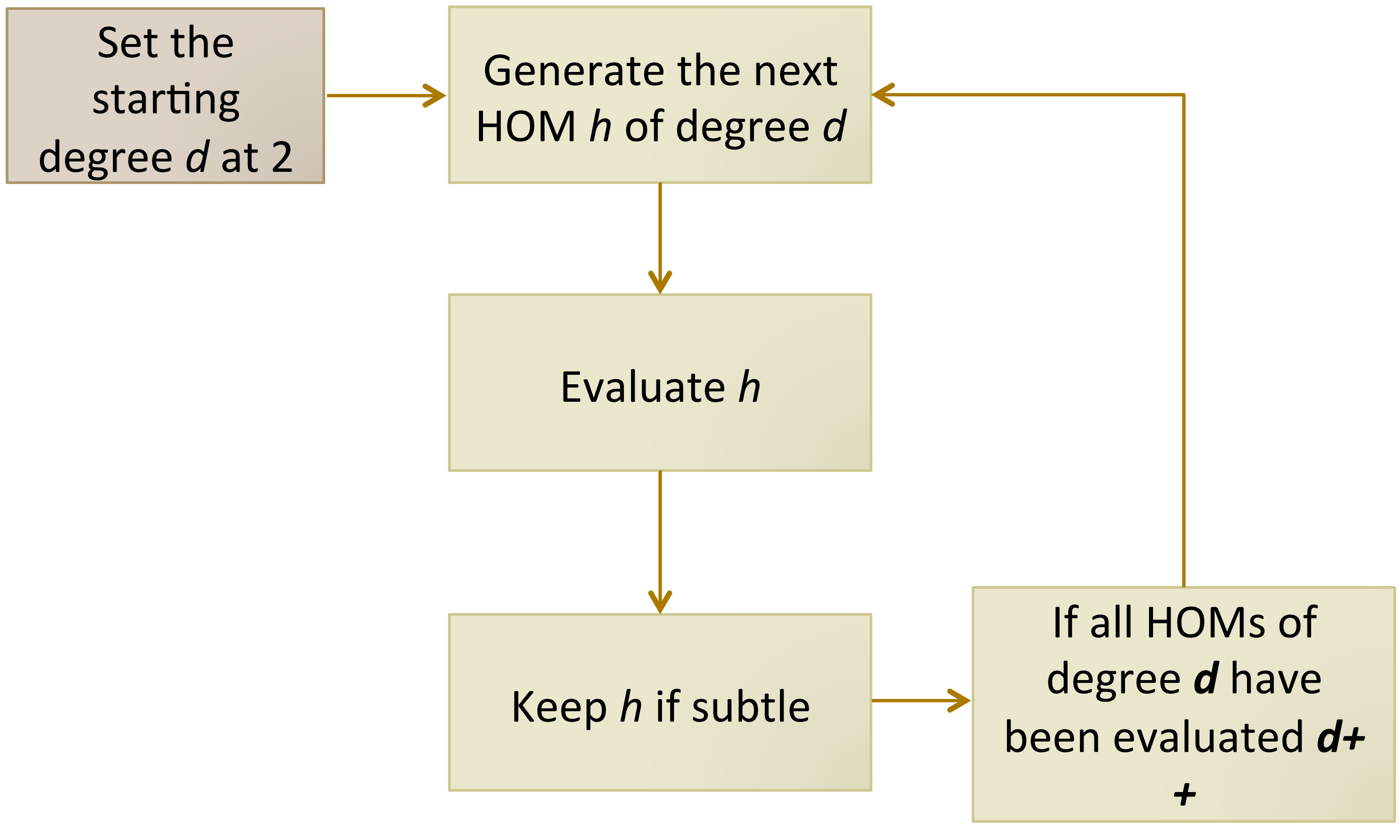
KilledBy(fom_2)= { tc_1, tc_5, tc_{11}, ….. }

KilledBy(fom_3)= {tc_5, tc_{11}, ….. }

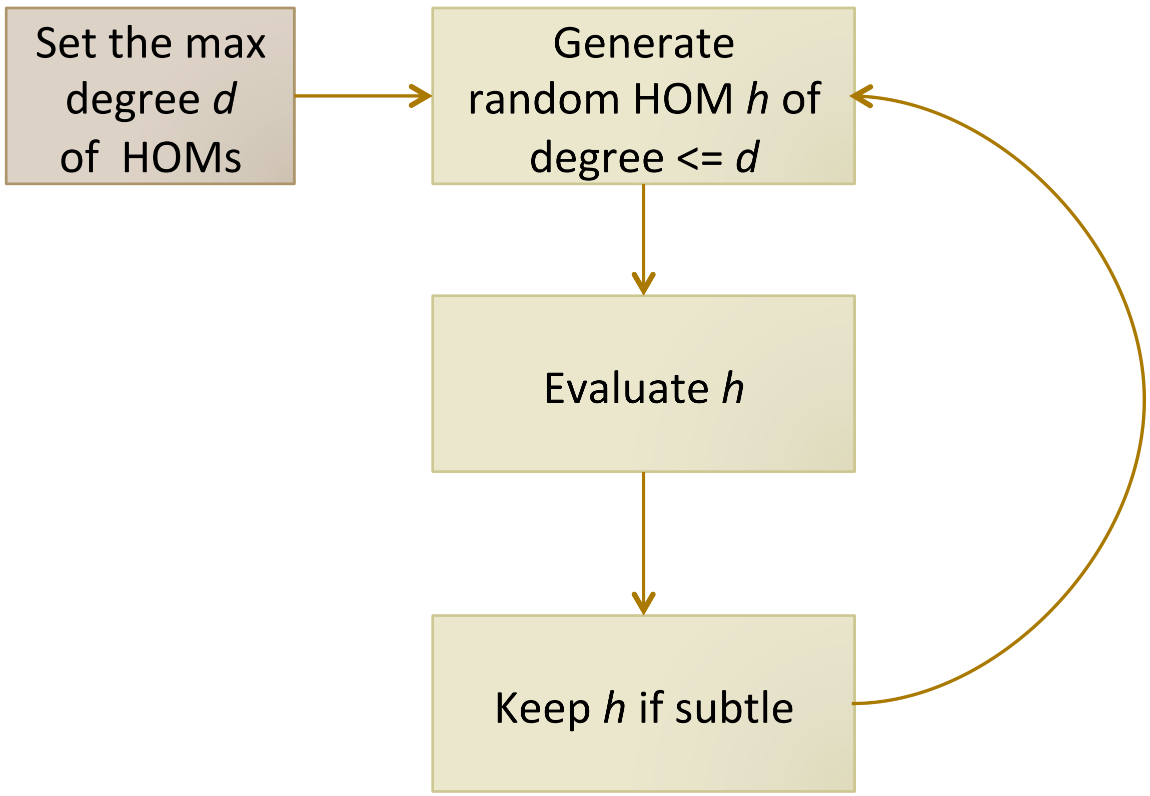
Considered HOMs= { (fom_1, fom_2), (fom_2,fom_3), (fom_1, fom_2,fom_3) }

Discarded HOMs= { (fom_1,fom_3) }

Restricted Enumeration Search



Restricted Random Search



Experimental Setup

Used 5 AspectJ and 5 Java programs of different sizes

Generated random test cases for each program that achieved statement coverage and killed all non-equivalent FOMs

Experiment steps:

- Ran each search technique 30 times per subject program
- The termination condition for each run was the exploration of 50,000 distinct HOMs
- Calculated the number of distinct, subtle HOMs that were found by each run

Experimental Setup

Table 6.1: Subject Programs

Subject program	Type	LOC	# of FOMs	# of classes	# of aspects	# of advices	# of pointcuts	# of ITDs	# of test cases
Coordinate	Java	121	242	2	0	0	0	0	14
Roman Numbers	Java	179	208	2	0	0	0	0	11
Cruise Control	Java	917	129	6	0	0	0	0	18
Elevator	Java	1046	249	17	0	0	0	0	14
XStream	Java	14,388	1216	318	0	0	0	0	96
Kettle	AspectJ	125	125	1	2	4	3	2	12
Movie Rental	AspectJ	191	316	3	1	8	9	0	15
Banking	AspectJ	243	92	2	2	2	2	1	9
Telecom	AspectJ	928	152	10	3	9	12	9	10
Cruise Control	AspectJ	1008	215	9	3	18	19	15	26

Measuring the Relative Effectiveness of the Search Techniques

RQ1: What is the relative effectiveness of the search technique in terms of their ability to find subtle HOMs?

- Effectiveness is measured in terms of the average number of subtle HOMs that can be found
- Restricted Random Search was used as a base line measure for the other five techniques

Average Number of Subtle HOMs

Program	Genetic	Local	Data Inter. Guided	Test Case Guided	Restricted Enumeration	Restricted Random
Cruise Java	76.1	77.8	80.7	77.1	34.8	**25.8**
Banking	30.9	29.2	30.8	28.9	27.1	**23.3**
Cruise AspectJ	20.3	29	39.9	33.2	22.1	**7**
Movie Rental	39.1	59.8	93.3	15.3	22	**4.7**
Kettle	35.3	55.1	56.1	57.7	31.5	**19.7**
Coordinate	72.4	200.9	213.8	223.3	84.4	**27.5**
Elevator	13.7	26	24.1	20.6	19	**5.7**
Telecom	10.3	20.5	19	19.9	6.8	**4**
XStream	0.4	20	11.4	12	13.4	**0.3**
Roman	28.6	30.4	35.4	37.9	41	**16.7**

Cost of Killing Subtle HOMs

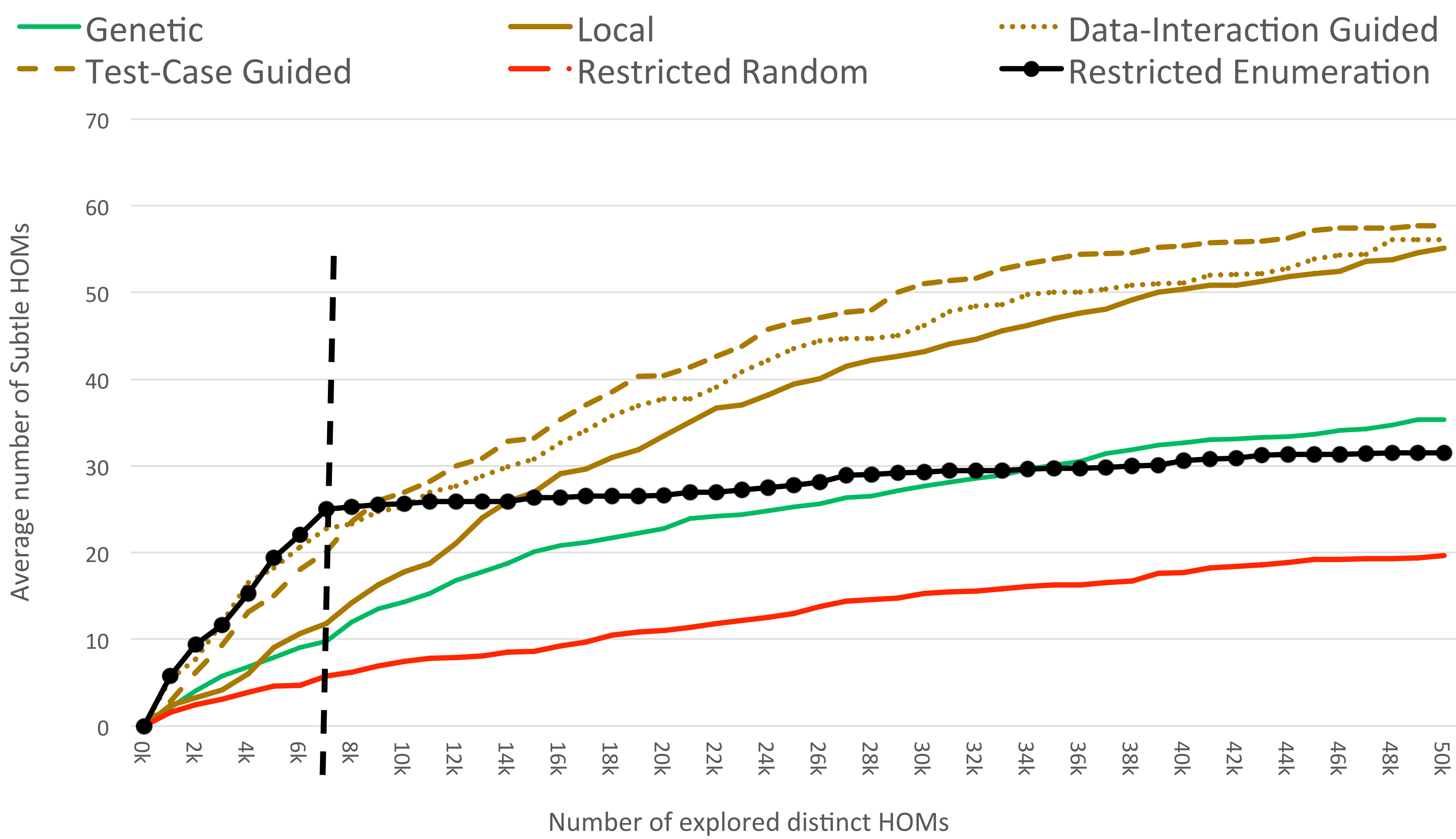
Subject program	# of test cases that killed all FOMs	# of test cases that were generated to kill subtle HOMs.
Coordinate	14	1290
Roman	11	876
Cruise (Java)	18	818
Elevator	14	1017
XStream	96	0
Kettle	12	912
Movie Rental	15	1015
Banking	9	1012
Telecom	10	1115
Cruise (AspectJ)	26	908

Measuring the Relative Effectiveness of the Search Techniques

RQ2: How does the relative effectiveness of the search techniques compare over time?

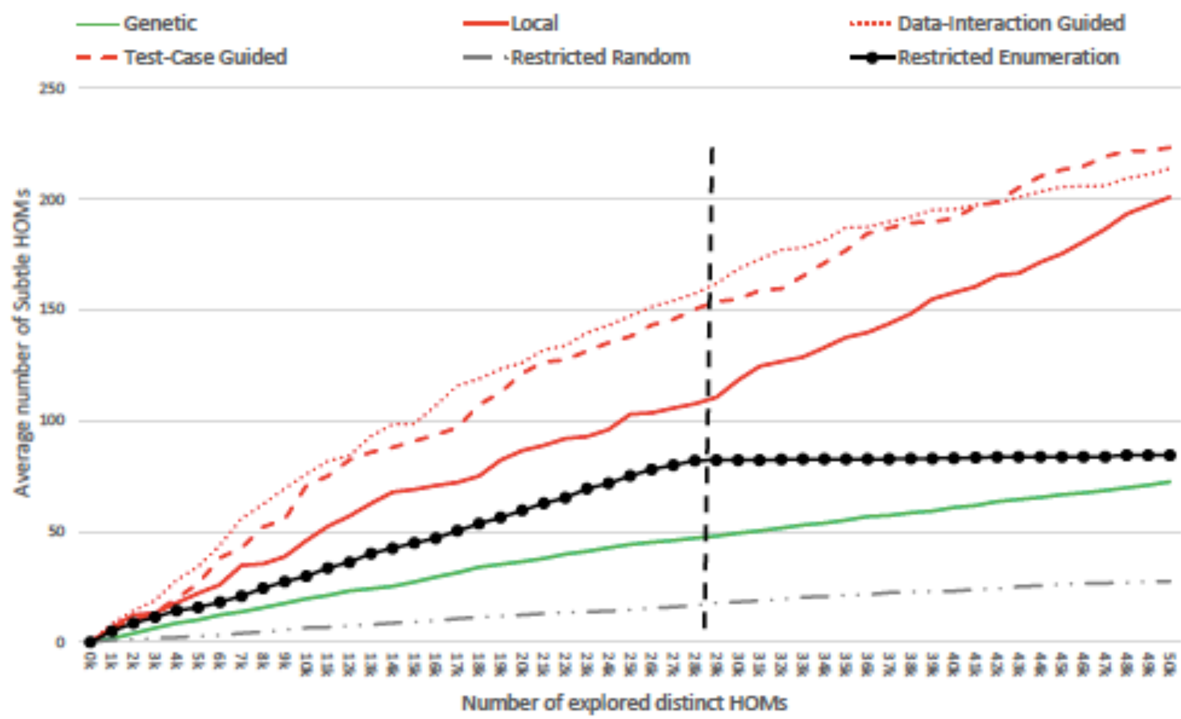
- Investigated the growth in the average number of distinct, subtle HOMs
- The number of explored, distinct HOMs is considered a quasi-representation of the time

Growth in the Average Number of Subtle HOMs Over Time



Data from the Kettle program

Growth in the Average Number of Subtle HOMs Over Time



Line Chart 7.17: Growth in the average number of subtle HOMs that were found over the number of explored HOMs for Coordinate

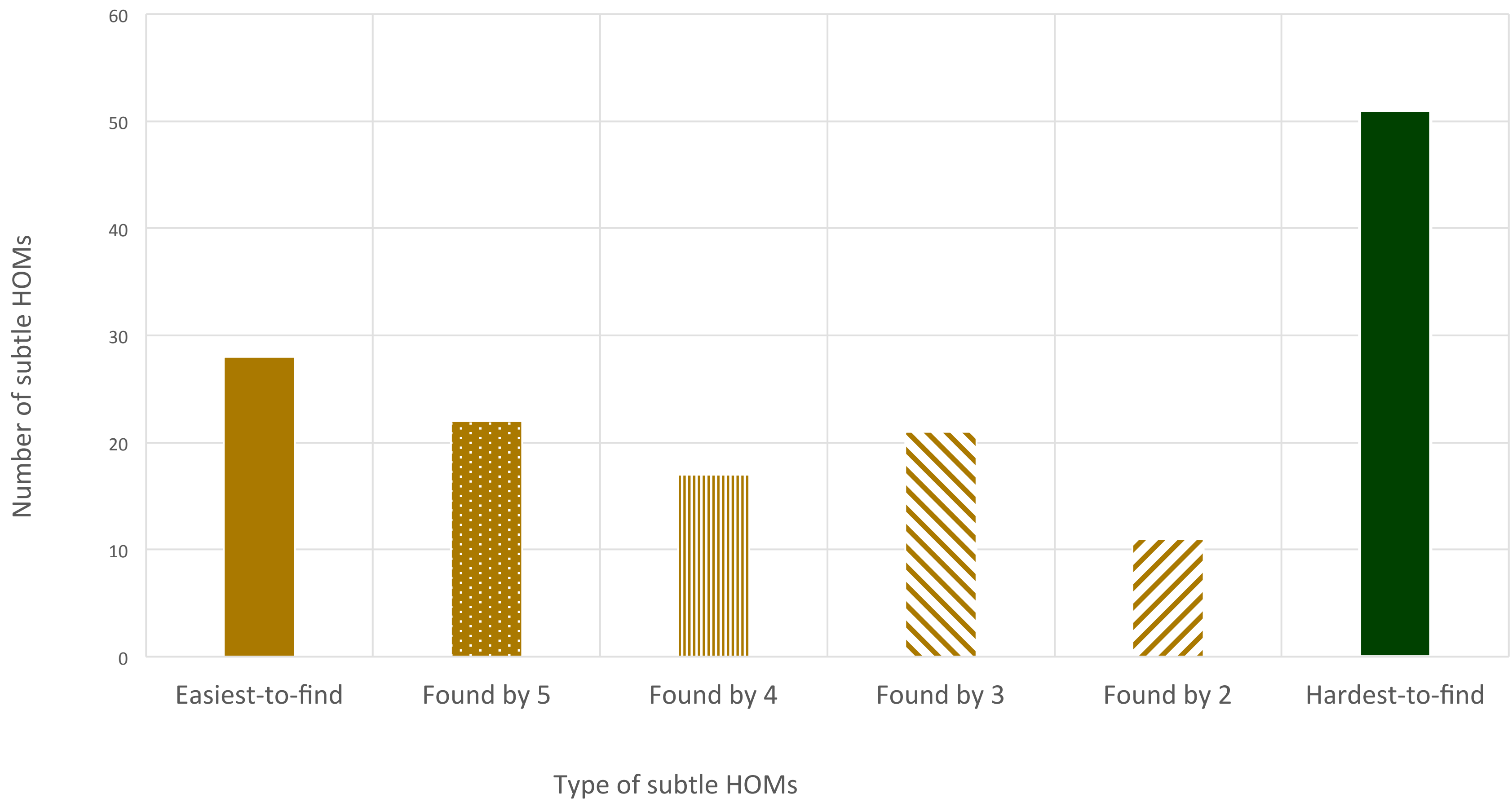
Comparing Sets of Subtle HOMs Found by Different Search Techniques

RQ1: What set of subtle HOMs is found by all techniques and what set of subtle HOMs is uniquely found by each technique?

Subtle HOMs were classified into:

- **Easiest-to-find subtle HOM**: can be found by all the search techniques
- **Hardest-to-find subtle HOM**: can be uniquely found by only one search technique

Easiest-to-find and Hardest-to-find Subtle HOM



Data from the Kettle program

Cost of Finding Subtle HOMs

RQ1: What is the computational cost of finding subtle HOMs using the search techniques?

- The cost is measured in terms of the time taken to find subtle HOMs

Answer:

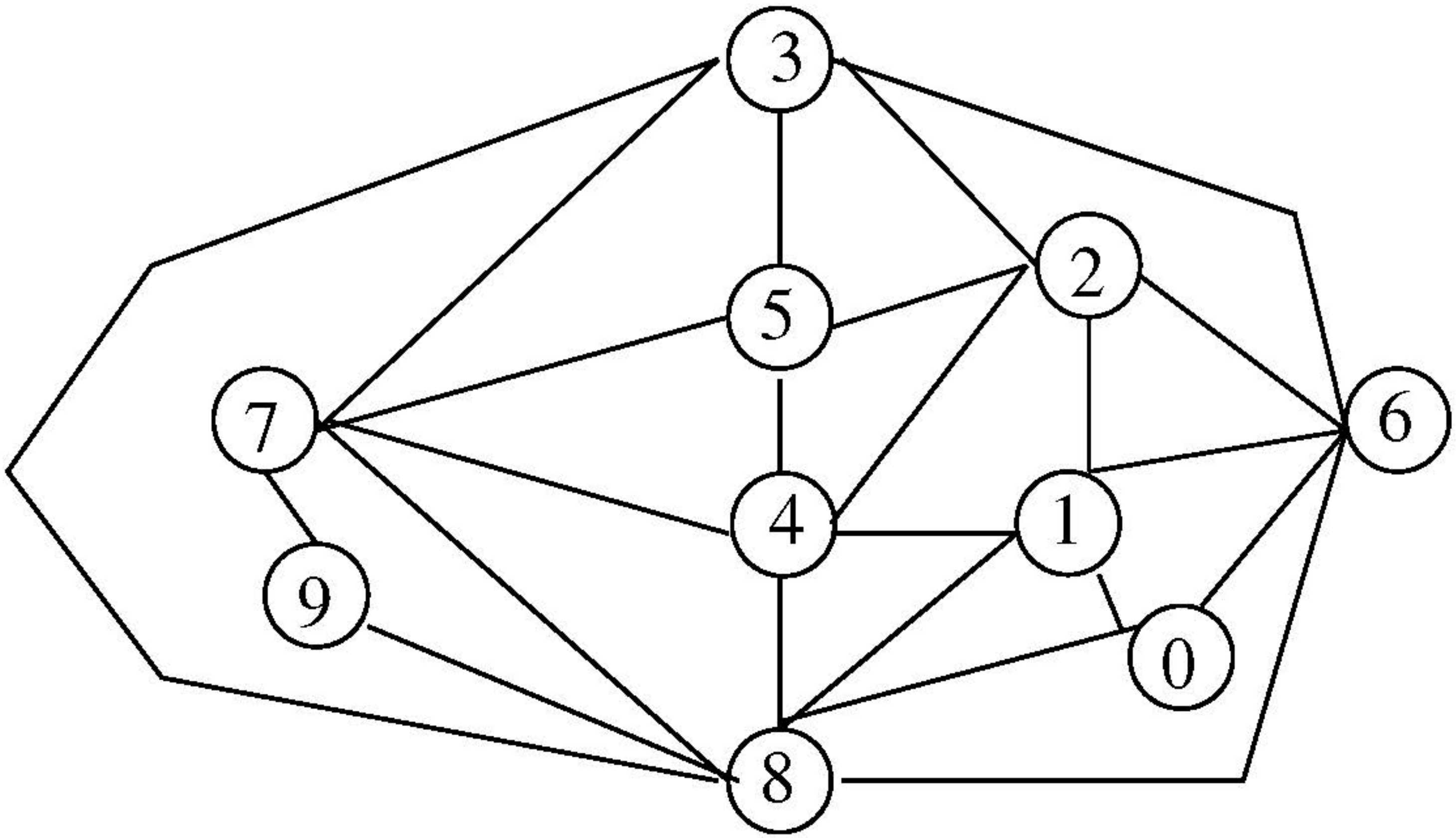
- On average, exploring and evaluating 50,000 HOMs requires around 19 hours
- The compilation and execution process of HOMs represented 98% of the computational cost of finding subtle HOMs
- Optimizing the compilation process of HOMs reduced the computational cost of finding subtle HOMs by 32%

Composition and Decomposition Relationships between Subtle HOMs

RQ1: Can subtle HOMs be composed to create new subtle HOMs of higher degrees?

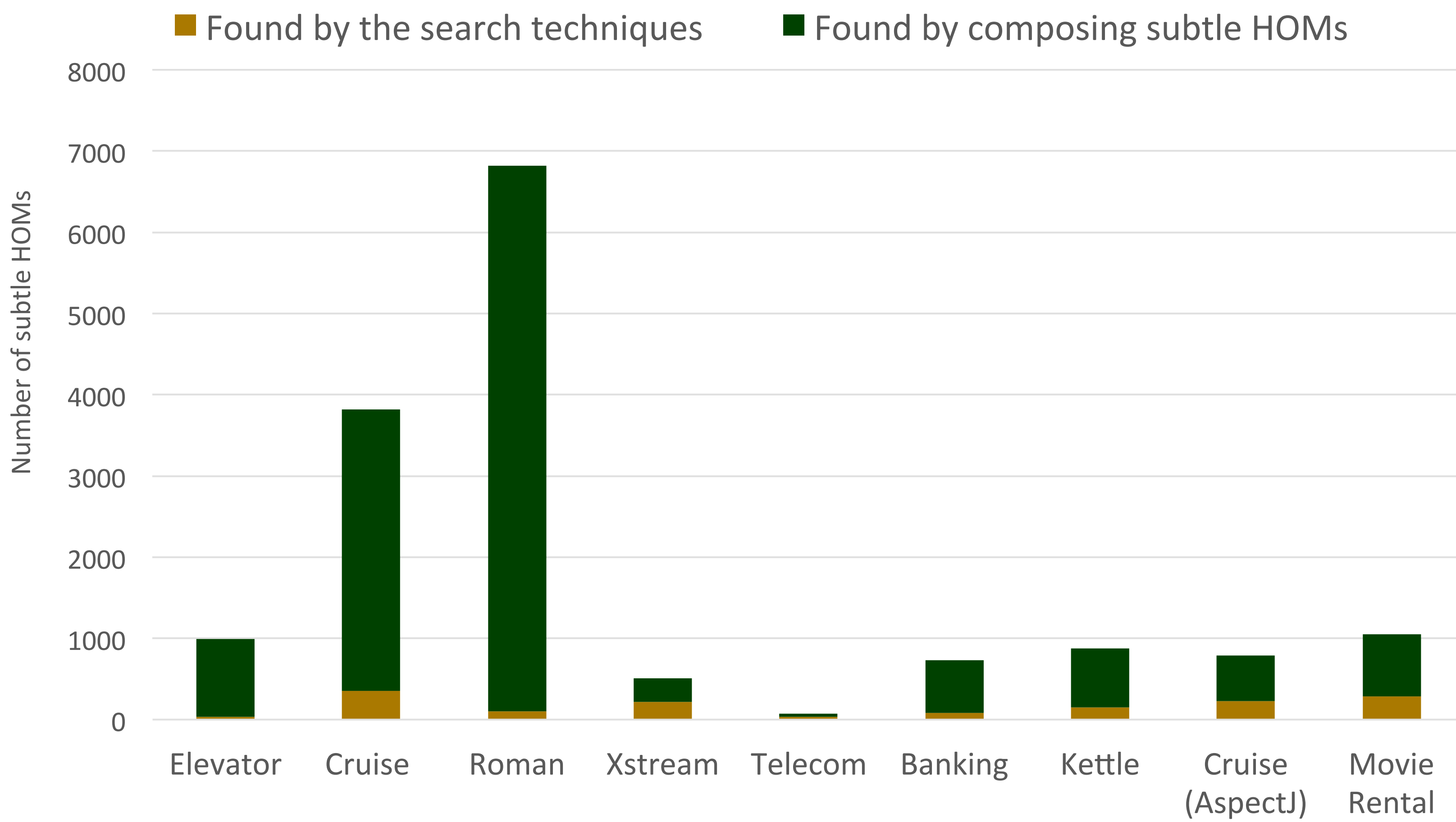
- Investigated composing subtle HOMs that were found by the Restricted Enumeration Search to create new subtle HOMs of higher degrees

Composing HOMs: Variable Interaction



The Variable Interaction Graph

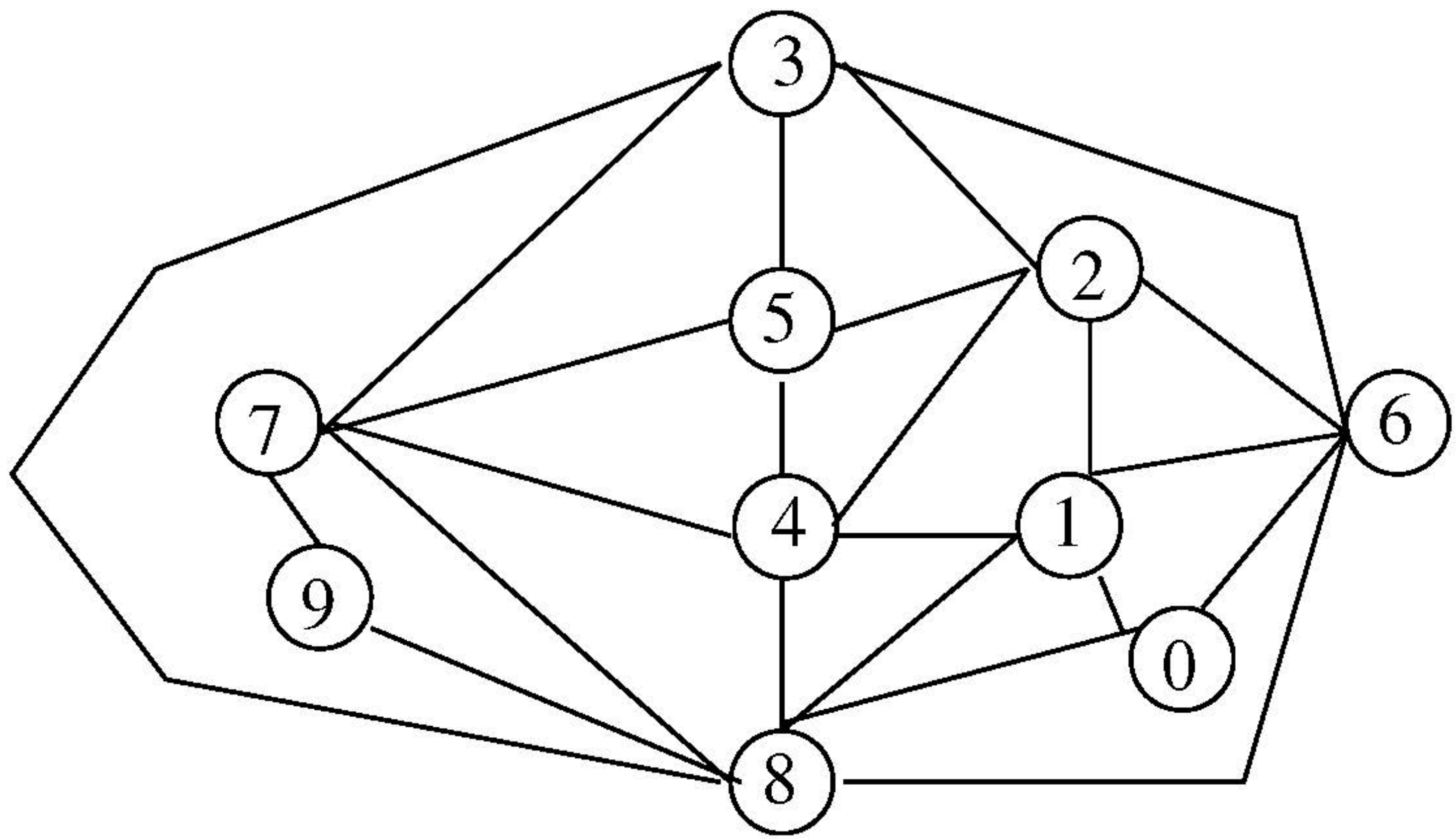
Composing Subtle HOMs



Average Number of Subtle HOMs

Program	Genetic	Local	Data Inter. Guided	Test Case Guided	Restricted Enumeration	Restricted Random
Cruise Java	76.1	77.8	80.7	77.1	34.8	**25.8**
Banking	30.9	29.2	30.8	28.9	27.1	**23.3**
Cruise AspectJ	20.3	29	39.9	33.2	22.1	**7**
Movie Rental	39.1	59.8	93.3	15.3	22	**4.7**
Kettle	35.3	55.1	56.1	57.7	31.5	**19.7**
Coordinate	72.4	200.9	213.8	223.3	84.4	**27.5**
Elevator	13.7	26	24.1	20.6	19	**5.7**
Telecom	10.3	20.5	19	19.9	6.8	**4**
XStream	0.4	20	11.4	12	13.4	**0.3**
Roman	28.6	30.4	35.4	37.9	41	**16.7**

Composing HOMs: Variable Interaction



The Variable Interaction Graph

Composing HOMs: Variable Interaction

Table 11.2: Comparing the number of subtle HOMs that were found by the search techniques and by composing subtle HOMs that were found by Restricted Enumeration Search

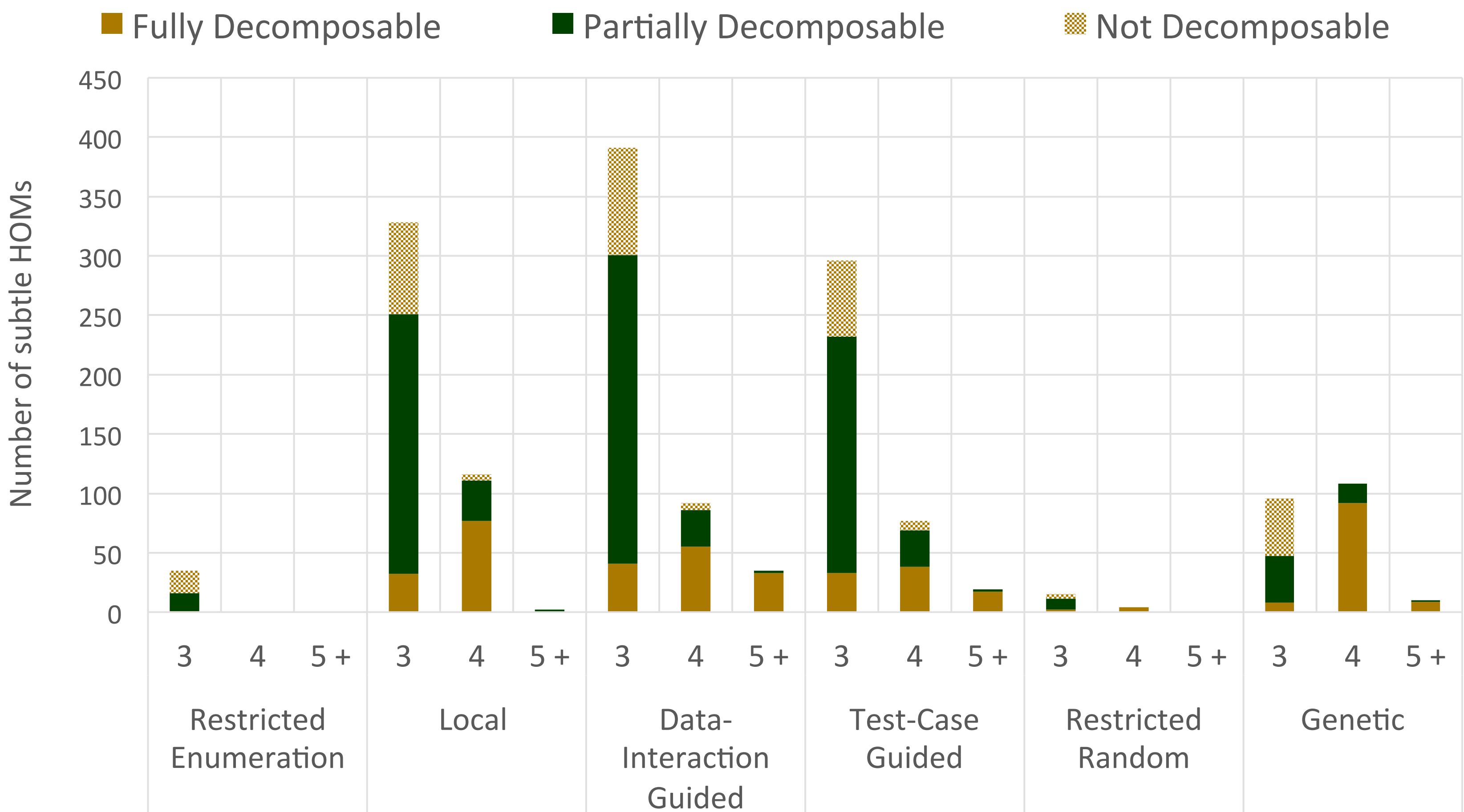
Program	# of all subtle HOMs that were found by the search techniques	# of all subtle HOMs that were found by composing subtle HOMs
Elevator	31	962
Cruise (Java)	355	3464
Roman	105	6717
XStream	216	291
Telecom	30	44
Banking	79	650
Kettle	150	724
Cruise (AspectJ)	227	558
Movie Rental	283	764

Composition and Decomposition Relationships between Subtle HOMs

RQ2: To what extent do subtle HOMs of higher degrees represent a composition of subtle HOMs of lower degrees?

- Investigated the number of subtle HOMs that were found by each search technique with respect to their decomposition type
  - Fully decomposable into other subtle HOMs
  - Partially decomposable into other subtle HOMs
  - Not decomposable into other subtle HOMs

Decomposing Subtle HOMs



Conclusions

The search-based software engineering techniques can produce a large number of distinct, subtle HOMs

Local Search and both the Guided Local Search techniques were more effective than the other techniques in terms of their ability to find subtle HOMs

Combining FOMs that are closer to each other in terms of their location is more likely to create subtle HOMs

Conclusions…

Subtle HOMs of higher degrees are likely to exist as compositions of multiple subtle HOMs of lower degrees

Subtle HOMs of higher degrees can be effectively found by composing subtle HOMs of lower degrees

The search-based software engineering techniques were able to find subtle HOMs of higher degrees that could not be found by composing subtle HOMs of lower degrees

Publications

1. Higher Order Mutation Testing Tool For Java and AspectJ Programs, ICST, proceed-ings of the 7th IEEE International Conference on Software Testing, Verification and Validation, Mutation, 2014

2. Comparing Search Techniques for Finding Subtle Higher Order Mutants, proceedings of the 23rd Conference on Genetic and Evolutionary Computation, 2014

3. Constructing Subtle Higher Order Mutants for Java and AspectJ Programs, In International Symposium on Software Reliability Engineering, 2013

4. An Exploratory Study Of Higher Order Mutation Testing In Aspect-Oriented Programming, In International Symposium on Software Reliability Engineering, 2012

5. Using a Genetic Algorithm Optimizer Tool to Generate Good Quality Timetables, In IEEE International Conference on Electronics, Circuits and Systems, 2003