

Inferring Test Models from Kate's Bug Reports using Multi-objective Search

Yuanyuan Zhang Mark Harman Yue Jia and Federica Sarro

University College London



SSBSE 2015 Challenge Track Paper

Inferring Test Models from Kate's Bug Reports using Multi-objective Search

Yves-Denis Flipo, Mark Brown, Yan Yu, and Fabrice Heuss
UMR5175, Inria Rennes - Bretagne Atlantique,
35042 Rennes Cedex, France, {flipo, brown, yu, heuss}@irisa.fr

Abstract. Models abstract from concrete test cases the way to use a test model to generate test cases. In this paper, we infer test models from test logs. We assume that an oracle in the target language, the abstract models can be used to derive test cases which better cover the target programs. We use the state-of-the-art MOEA and the multi-objective search algorithm to infer test models. We compare our results with the state-of-the-art MOEA.

Keywords: MOEA, Test Models, Multi-objective Search

1 Introduction and Background

Many systems allow users to submit bug reports when they experience unexpected behavior. Developers need to address and fix these issues, based on these bug reports. Unfortunately, not all of the bug reports are reported. It means that many reports that go up to 20% of the total bug reports are not reported. It means that many reports that go up to 20% of the total bug reports are not reported. It means that many reports that go up to 20% of the total bug reports are not reported.

Consequently, abstract test cases that cover the reported bugs features could improve software development conditions in these bug fixes. In this paper, we infer test models from abstract test cases. We assume that an oracle in the target language, the abstract models can be used to derive test cases which better cover the target programs. We use the state-of-the-art MOEA and the multi-objective search algorithm to infer test models. We compare our results with the state-of-the-art MOEA.

In this work, our approach aims to infer models from test logs. We assume that an oracle in the target language, the abstract models can be used to derive test cases which better cover the target programs. We use the state-of-the-art MOEA and the multi-objective search algorithm to infer test models. We compare our results with the state-of-the-art MOEA.

test logs. An abstract model from these logs reports a generalization of the set of test cases which best represent the logs. The model can be used to generate new test cases targeting the same reported bugs. Before the oracle, traditional single objective inferring approaches tend to suffer from two important problems. The abstract model often covers more behaviors specified in bug reports (under consideration) and includes more irrelevant behaviors (over generalization). To overcome this limitation, we adapted the multi-objective search proposed by Trauch et al. [23] to address these two conflicting objectives.

We apply our approach to the MOEA 2013 Challenge program that is a popular multi-objective test oracle. The general reported behaviors that the model generated from our approach can only generate good results (better quality and cost approximation) but also provide a good level of both objective quality.

2 Model Inference Framework
The approach to infer model abstract test cases is divided into four phases. The first phase consists in test case reports from the test log tracking system. Then, the second phase aims to infer the abstract test cases from the test logs. The third phase aims to infer the abstract test cases from the test logs. The fourth phase aims to infer the abstract test cases from the test logs.

Phase 1 - Bug Report Extraction. Kate is a multiplatform test oracle system in C/C++. The MOEA inferring system [2] is used by the test oracle to generate test logs from the reported software bugs. A test oracle can implement to collect test logs. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

Phase 2 - Test Case Parsing. An abstract bug description for each bug case by parsing the test data according to a set of search rules. We manually designed the search rules based on MOEA. We compare abstract test cases with the test logs to infer test models. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

Phase 3 - Data Mining From Results. Our test oracle in Phase 2 is designed to generate test logs. It generates test logs in a structured way. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

test logs. An abstract model from these logs reports a generalization of the set of test cases which best represent the logs. The model can be used to generate new test cases targeting the same reported bugs. Before the oracle, traditional single objective inferring approaches tend to suffer from two important problems. The abstract model often covers more behaviors specified in bug reports (under consideration) and includes more irrelevant behaviors (over generalization). To overcome this limitation, we adapted the multi-objective search proposed by Trauch et al. [23] to address these two conflicting objectives.

We apply our approach to the MOEA 2013 Challenge program that is a popular multi-objective test oracle. The general reported behaviors that the model generated from our approach can only generate good results (better quality and cost approximation) but also provide a good level of both objective quality.

2 Model Inference Framework
The approach to infer model abstract test cases is divided into four phases. The first phase consists in test case reports from the test log tracking system. Then, the second phase aims to infer the abstract test cases from the test logs. The third phase aims to infer the abstract test cases from the test logs. The fourth phase aims to infer the abstract test cases from the test logs.

Phase 1 - Bug Report Extraction. Kate is a multiplatform test oracle system in C/C++. The MOEA inferring system [2] is used by the test oracle to generate test logs from the reported software bugs. A test oracle can implement to collect test logs. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

Phase 2 - Test Case Parsing. An abstract bug description for each bug case by parsing the test data according to a set of search rules. We manually designed the search rules based on MOEA. We compare abstract test cases with the test logs to infer test models. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

Phase 3 - Data Mining From Results. Our test oracle in Phase 2 is designed to generate test logs. It generates test logs in a structured way. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

Table 1. Example of Test Cases Generated

Test Case	Test Case	Test Case	Test Case
Test Case 1	Test Case 2	Test Case 3	Test Case 4
Test Case 5	Test Case 6	Test Case 7	Test Case 8
Test Case 9	Test Case 10	Test Case 11	Test Case 12
Test Case 13	Test Case 14	Test Case 15	Test Case 16
Test Case 17	Test Case 18	Test Case 19	Test Case 20

In order to infer test models, we use the state-of-the-art MOEA. We compare abstract test cases with the test logs to infer test models. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

Phase 4 - Model Inference. We use the multi-objective search algorithm to infer test models. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

Table 2. Objective and performance results for GA, MOEA/D and MOEA/D-DE

Algorithm	Quality	Performance
GA	0.1	0.2
MOEA/D	0.3	0.4
MOEA/D-DE	0.5	0.6

Table 2 shows the mean, best and highest values of three objectives and average running time and the number of iterations generated by GA and MOEA/D. In the MOEA/D, the MOEA/D generates abstract test cases which better cover the target programs. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

The abstract test cases generated from our approach can only generate good results (better quality and cost approximation) but also provide a good level of both objective quality.



Fig. 3. Test plots of the test objectives, test objectives and test cases from the model abstract test cases. The figure shows three plots: (a) Test Case 1, (b) Test Case 2, and (c) Test Case 3. Each plot shows the distribution of test cases for a specific objective. The x-axis represents the test case number, and the y-axis represents the objective value. The plots show that the abstract test cases generated by the MOEA/D algorithm are more diverse and cover a wider range of test cases compared to the state-of-the-art MOEA.

Table 3. Objective and performance results for GA, MOEA/D and MOEA/D-DE

Algorithm	Quality	Performance
GA	0.1	0.2
MOEA/D	0.3	0.4
MOEA/D-DE	0.5	0.6

In order to infer test models, we use the state-of-the-art MOEA. We compare abstract test cases with the test logs to infer test models. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

Phase 4 - Model Inference. We use the multi-objective search algorithm to infer test models. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

The abstract test cases generated from our approach can only generate good results (better quality and cost approximation) but also provide a good level of both objective quality.

Table 4. Objective and performance results for GA, MOEA/D and MOEA/D-DE

Algorithm	Quality	Performance
GA	0.1	0.2
MOEA/D	0.3	0.4
MOEA/D-DE	0.5	0.6

In order to infer test models, we use the state-of-the-art MOEA. We compare abstract test cases with the test logs to infer test models. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

Phase 4 - Model Inference. We use the multi-objective search algorithm to infer test models. We assume that the test logs are reported. There have been 1,000 test logs since reported including their details (reported, tested and closed since January 2008). Our model uses the extracted test logs to infer test models.

The abstract test cases generated from our approach can only generate good results (better quality and cost approximation) but also provide a good level of both objective quality.

The abstract test cases generated from our approach can only generate good results (better quality and cost approximation) but also provide a good level of both objective quality.

The abstract test cases generated from our approach can only generate good results (better quality and cost approximation) but also provide a good level of both objective quality.

Inferring Test Models from Kate's Bug Reports using Multi-objective Search

Xiaohu Zhang, Mark Harman, Yu Yu, and Roberto Ruiz
UNIST, Department of Computer Science,
University College London, Gower Street, London, WC1E 6BT, UK

Abstract. Models abstract from concrete execution logs to be used to test ground systems before they are deployed. We infer test models from test logs that are written in the natural language. The abstract models can be used to derive test cases which further exercise the target programs required by users. Our search-based approach infers abstract models from concrete logs. To achieve this, we use a multi-objective search algorithm. It is a search algorithm that considers both the number of test cases and the number of test models. We apply our approach to Kate's bug reports to infer test models in terms of test cases and test models. We show that our approach can infer test models from test logs in a more efficient way than the state-of-the-art.

Keywords: SMT, SLP, Test Models, Multi-objective Search

1 Introduction and Background

Many systems allow users to submit bug reports when they encounter unexpected behavior. Developers need to address and fix these issues based on these bug reports. Unfortunately, not all of the bug reports are reported. It is more likely reports that go to 200, about 10% of the bug reports are reported. This means that many of the reported bug reports are not reported. This means that many of the reported bug reports are not reported. This means that many of the reported bug reports are not reported.

Concretely, abstract models that capture the required target features could support software development conditions in their logs. In this paper, we infer an abstract model from concrete logs for each test case. This model can be used to generate test cases for the target system. This model can be used to generate test cases for the target system. This model can be used to generate test cases for the target system.

In this work, our approach aims to infer models from test logs that are written in the natural language. Bug reports submitted by users are written in natural language. Many of these reports are not reported. This means that many of the reported bug reports are not reported.

the logs. An abstract model from these logs reports a generalization of the test cases that are reported. The model can be used to generate test cases for the target system. This model can be used to generate test cases for the target system. This model can be used to generate test cases for the target system.

2 Model Inference Framework
The approach to infer test models from logs is divided into three phases. The first phase extracts test logs from the bug reports. The second phase extracts test logs from the bug reports. The third phase extracts test logs from the bug reports.

Phase 1: Bug Report Extraction. Kate is a multiplatform test case system. It is used to generate test cases for the target system. It is used to generate test cases for the target system. It is used to generate test cases for the target system.

Phase 2: Raw Data Parsing. We extract test logs from the bug reports. We extract test logs from the bug reports. We extract test logs from the bug reports.

Phase 3: Data Mining. We extract test logs from the bug reports. We extract test logs from the bug reports. We extract test logs from the bug reports.

test models, conditions and generated models. From an area where a test case fails and then we have the generated models that will generate more test cases. This means that many of the reported bug reports are not reported.

In the second step, we extract model information from logs using a test case generator. We extract model information from logs using a test case generator. We extract model information from logs using a test case generator.

Phase 4: Model Refinement. We use a multi-objective search algorithm to refine the models. We use a multi-objective search algorithm to refine the models. We use a multi-objective search algorithm to refine the models.

3 Experimental Results

We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

Our approach can infer test models from logs in a more efficient way than the state-of-the-art. Our approach can infer test models from logs in a more efficient way than the state-of-the-art.

Table 1: Example of Test Cases Generated

Test Case	Test Case	Test Case	Test Case
Test Case 1	Test Case 2	Test Case 3	Test Case 4
Test Case 5	Test Case 6	Test Case 7	Test Case 8
Test Case 9	Test Case 10	Test Case 11	Test Case 12
Test Case 13	Test Case 14	Test Case 15	Test Case 16
Test Case 17	Test Case 18	Test Case 19	Test Case 20

In order to compare the performance of our approach, we use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

Table 2: Performance of our approach on test cases

Test Case	Test Case	Test Case	Test Case
Test Case 1	Test Case 2	Test Case 3	Test Case 4
Test Case 5	Test Case 6	Test Case 7	Test Case 8
Test Case 9	Test Case 10	Test Case 11	Test Case 12
Test Case 13	Test Case 14	Test Case 15	Test Case 16
Test Case 17	Test Case 18	Test Case 19	Test Case 20

Table 2 shows the mean, best and highest values of test cases and average running time and the number of test cases generated by our approach. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

The statistical analysis of performance results is reported in Table 3. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

level. The results of all algorithms are significantly different. The other use of the test case generator is to generate test cases for the target system. We use a multi-objective search algorithm to infer test models.



Fig. 3: Box plots of the test cases, test models and test models from the models generated by multi-objective search, SMT and SLP.

Table 3: Performance results of the statistical analysis for SMT, SLP and SLP.

Algorithm	Mean	Best	Highest	Average
SMT	1.5	1.0	2.0	1.5
SLP	1.5	1.0	2.0	1.5
SLP	1.5	1.0	2.0	1.5

In order to compare the performance of our approach, we use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

Our approach can infer test models from logs in a more efficient way than the state-of-the-art. Our approach can infer test models from logs in a more efficient way than the state-of-the-art.

The statistical analysis of performance results is reported in Table 3. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

4 Conclusion

We have presented the use of multi-objective search algorithms to infer models from software bug reports. The models abstract test cases from the natural

Table 4: Performance results of the statistical analysis for SMT, SLP and SLP.

Algorithm	Mean	Best	Highest	Average
SMT	1.5	1.0	2.0	1.5
SLP	1.5	1.0	2.0	1.5
SLP	1.5	1.0	2.0	1.5

language. The models abstract test cases from the natural language. The models abstract test cases from the natural language. The models abstract test cases from the natural language.

Our approach can infer test models from logs in a more efficient way than the state-of-the-art. Our approach can infer test models from logs in a more efficient way than the state-of-the-art.

The statistical analysis of performance results is reported in Table 3. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

Our approach can infer test models from logs in a more efficient way than the state-of-the-art. Our approach can infer test models from logs in a more efficient way than the state-of-the-art.

The statistical analysis of performance results is reported in Table 3. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

Our approach can infer test models from logs in a more efficient way than the state-of-the-art. Our approach can infer test models from logs in a more efficient way than the state-of-the-art.

The statistical analysis of performance results is reported in Table 3. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

Our approach can infer test models from logs in a more efficient way than the state-of-the-art. Our approach can infer test models from logs in a more efficient way than the state-of-the-art.

The statistical analysis of performance results is reported in Table 3. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

Our approach can infer test models from logs in a more efficient way than the state-of-the-art. Our approach can infer test models from logs in a more efficient way than the state-of-the-art.

The statistical analysis of performance results is reported in Table 3. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

Our approach can infer test models from logs in a more efficient way than the state-of-the-art. Our approach can infer test models from logs in a more efficient way than the state-of-the-art.

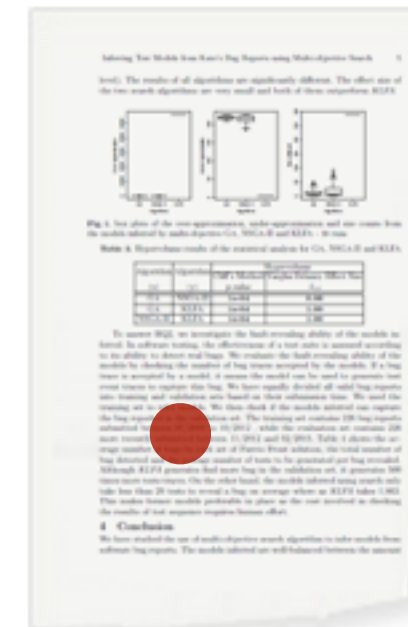
The statistical analysis of performance results is reported in Table 3. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

Our approach can infer test models from logs in a more efficient way than the state-of-the-art. Our approach can infer test models from logs in a more efficient way than the state-of-the-art.

The statistical analysis of performance results is reported in Table 3. We use a multi-objective search algorithm to infer test models. We use a multi-objective search algorithm to infer test models.

Our approach can infer test models from logs in a more efficient way than the state-of-the-art. Our approach can infer test models from logs in a more efficient way than the state-of-the-art.

Which page should we put more effort to proofread?



Motivation

ID ▲	Product	Comp	Assignee ▲	Status ▲	Resolution ▼	Summary	Changed ▼
349015	kate	general	kwrite-bugs-null	UNCO	---	Upgrading from kate4 does not provide any upgrading instructions, loses all data	07:48:33
349006	kate	kwrite	kwrite-bugs-null	UNCO	---	KWrite and Kate have the same configuration	23:35:03
348977	kate	general	kwrite-bugs-null	UNCO	---	Kate crashes trying to open file with <SPC><LF>	Wed 13:49
344341	kate	syntax	kwrite-bugs-null	UNCO	---	Kate ignores custom syntax highlighting xml files	Wed 12:29
348317	kate	syntax	kwrite-bugs-null	UNCO	---	[PATCH] Katepart syntax highlighting should recognize \u0123 style escapes for JavaScript	Tue 20:53
205447	kate	part	kwrite-bugs-null	UNCO	---	[BiDi/Unicode] Non-BMP characters are incorrectly handled	Tue 19:00
348934	kate	kwrite	kwrite-bugs-null	UNCO	---	Kwrite hangs while opening non-empty files via sftp	Tue 16:28
348921	kate	general	kwrite-bugs-null	UNCO	---	Cannot open files via sftp	Mon 16:11
347147	kate	syntax	kwrite-bugs-null	UNCO	---	keyword missing in csound	Mon 20:23
348846	kate	Vi Input	kwrite-bugs-null	UNCO	---	Cannot use the CTRL+W+Direction shortcut anymore	Sun 15:32
348845	kate	Vi Input	kwrite-bugs-null	UNCO	---	split and vsplit commands are reversed	Sun 15:25
348843	framewor	general	kwrite-bugs-null	UNCO	---	Editor highlights floating point number 1.039 in red in C++ code	Sun 15:15
348765	kate	syntax	kwrite-bugs-null	UNCO	---	Perl syntax highlighting is wrong when using scalar references (backslash quote)	Sat 00:12

Up to **24%** of post-release bug fixes of large software systems are **incorrect** and even **introduce additional faults**

Event-based Model Inference

Input: an abstraction of the observed sequences

- log files (contain a sequence of execution traces
function calls)

<println, formatter, close, println>

- bug reports (written in the natural language)

Output: an inferred model

- a FSM (accepts more traces than the observed ones and might not accept some of the observed traces)

Event-based Model Inference Challenges

- bug reports (written in the natural language)

a "profane" user upgrading from kate4 to kate5 gets is development settings completely wiped out as a result of the upgrade.

A description of a bug

Reproducible: Always

Reproducible or not?

Steps to Reproduce:

1. Install KDE4/kate4.x
2. Save your settings/data
3. Upgrade to KDE5/kate5.0.0

Steps to Reproduce

Actual Results:

- All the session data is lost
- All the custom syntax files are lost
- Probably more settings i didn't use are lost, everything saved into ~/.kde/* in general

Actual Results

Expected Results:

- Kate5 showing a BIG warning that the user data is lost and what is changed and instructing the user on how to recover it:
- ~/.kde is now ~/.local
 - copy ~/.kde/share/apps/kate as ~/.local/share/kate

Expected Results

Event-based Model Inference Challenges

the generalisation capability of a model

- a FSM (accepts more traces than the observed ones and might not accept some of the observed traces)

introduce infeasible behaviours

~~overgeneralisation~~
overgeneralisation

exclude possible behaviours

~~undergeneralisation~~
undergeneralisation

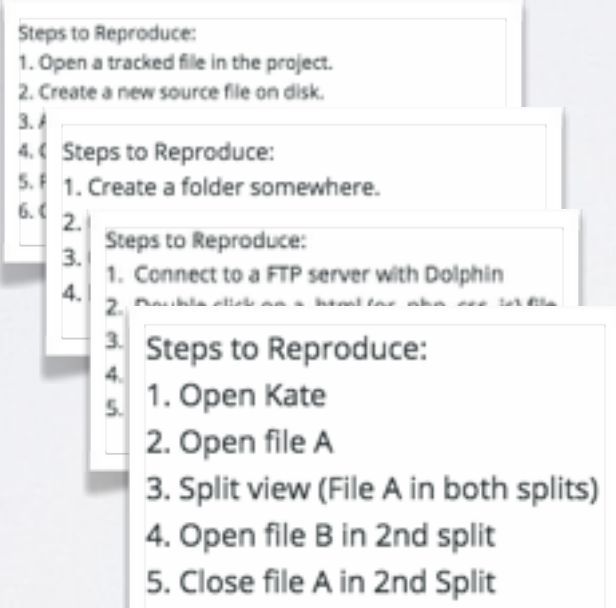
Model Inference Framework

Extraction



1

Parsing



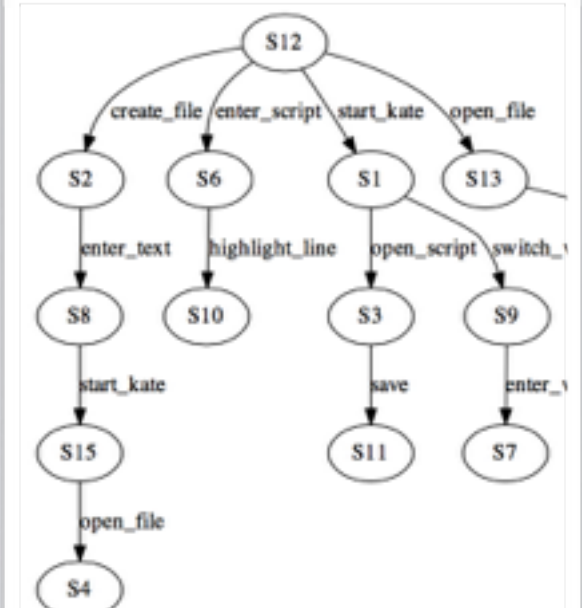
2

Data Mining

```
dictionary = corpora.  
corpus = [dictionary.  
tfidf = models.TfidfM  
corpus_tfidf = tfidf[  
lsi = models.LsiModel  
#lda = models.LdaMode  
index = similarities.
```

3

Search-Based Model Generation



4

Model Inference Framework

Extraction

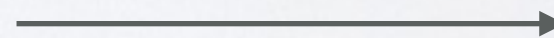


I. Filter Bug Reports

filter terms: product; bug status; bug severity

II. Extract All Valid Bug Report IDs

III. Crawl Bug Reports



Model Inference Framework

Parsing



Steps to Reproduce:
1. Open a tracked file in the project.
2. Create a new source file on disk.
3. ...

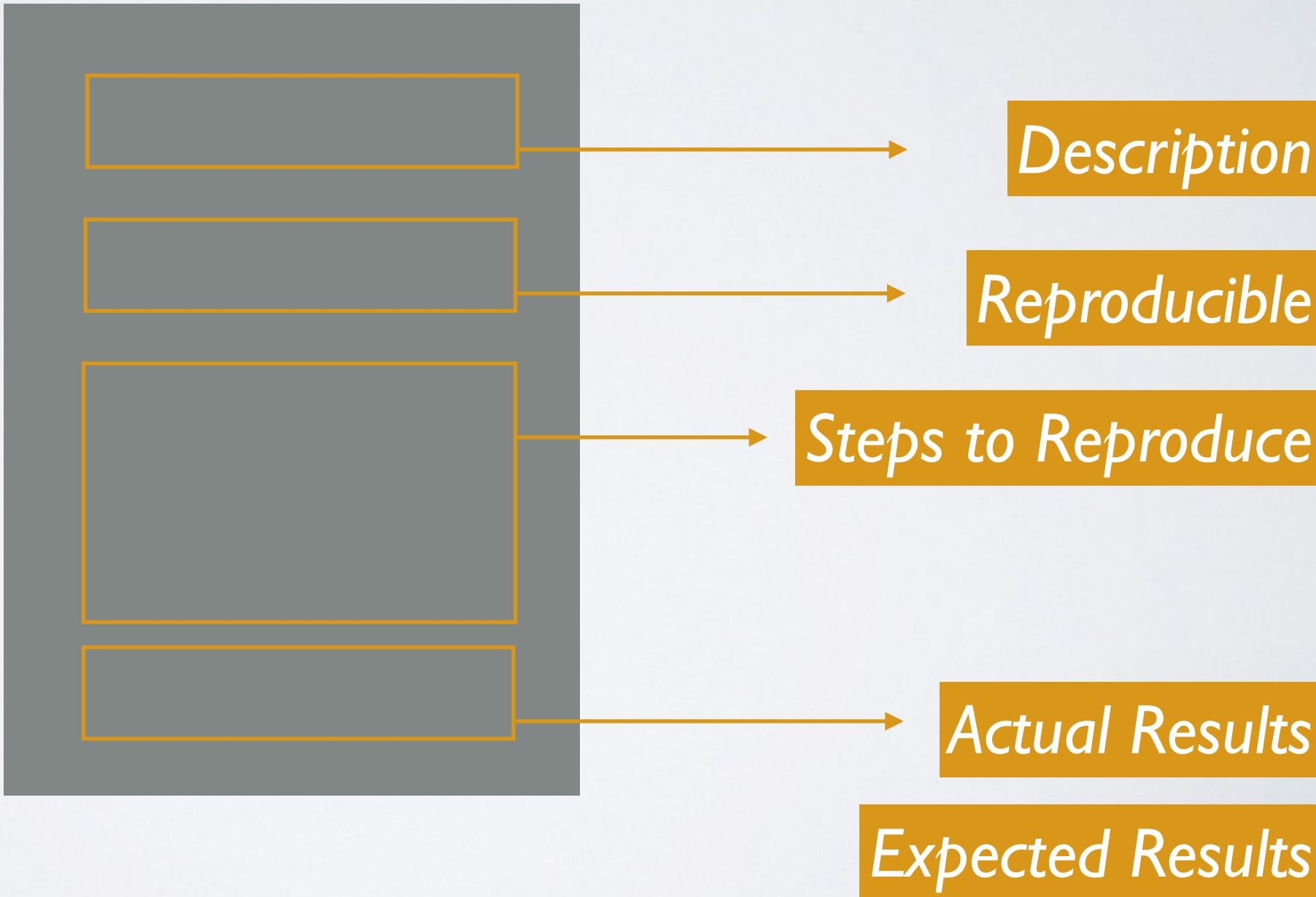
Steps to Reproduce:
1. Create a folder somewhere.
2. ...
3. ...

Steps to Reproduce:
1. Connect to a FTP server with Dolphin
2. ...
3. ...

Steps to Reproduce:
1. Open Kate
2. Open file A
3. Split view (File A in both splits)
4. Open file B in 2nd split
5. Close file A in 2nd Split

2

1. Filter Bug Reports



Model Inference Framework

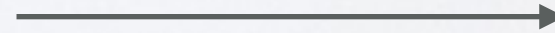
Parsing



2

I. Filter Bug Reports

II. OUTPUT .str Files



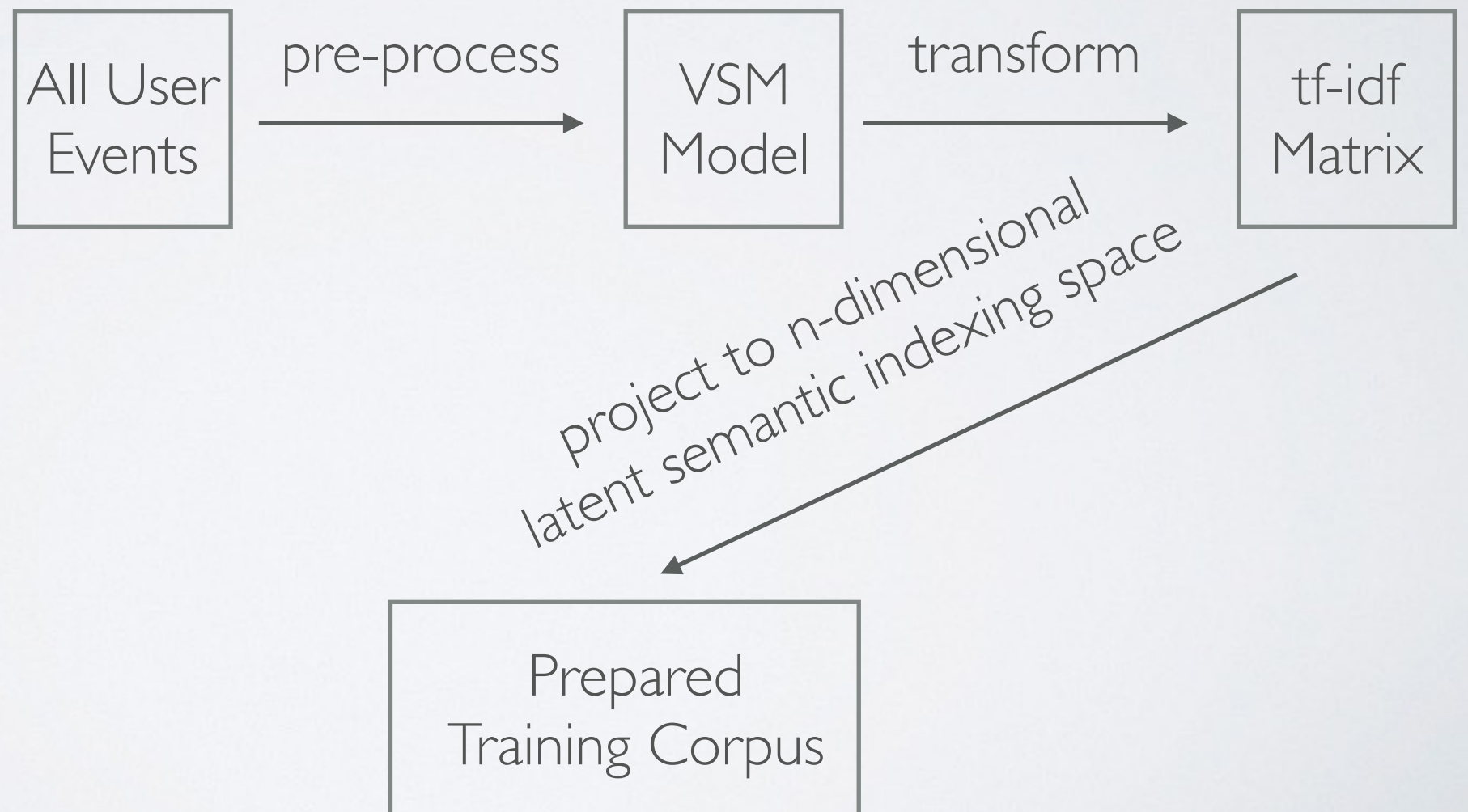
Model Inference Framework

I. Preparing Training Corpus

Data Mining

```
dictionary = corpora.  
corpus = [dictionary.  
tfidf = models.TfidfM  
corpus_tfidf = tfidf[  
  
lsi = models.LsiModel  
#lda = models.LdaMode  
  
index = similarities.
```

3



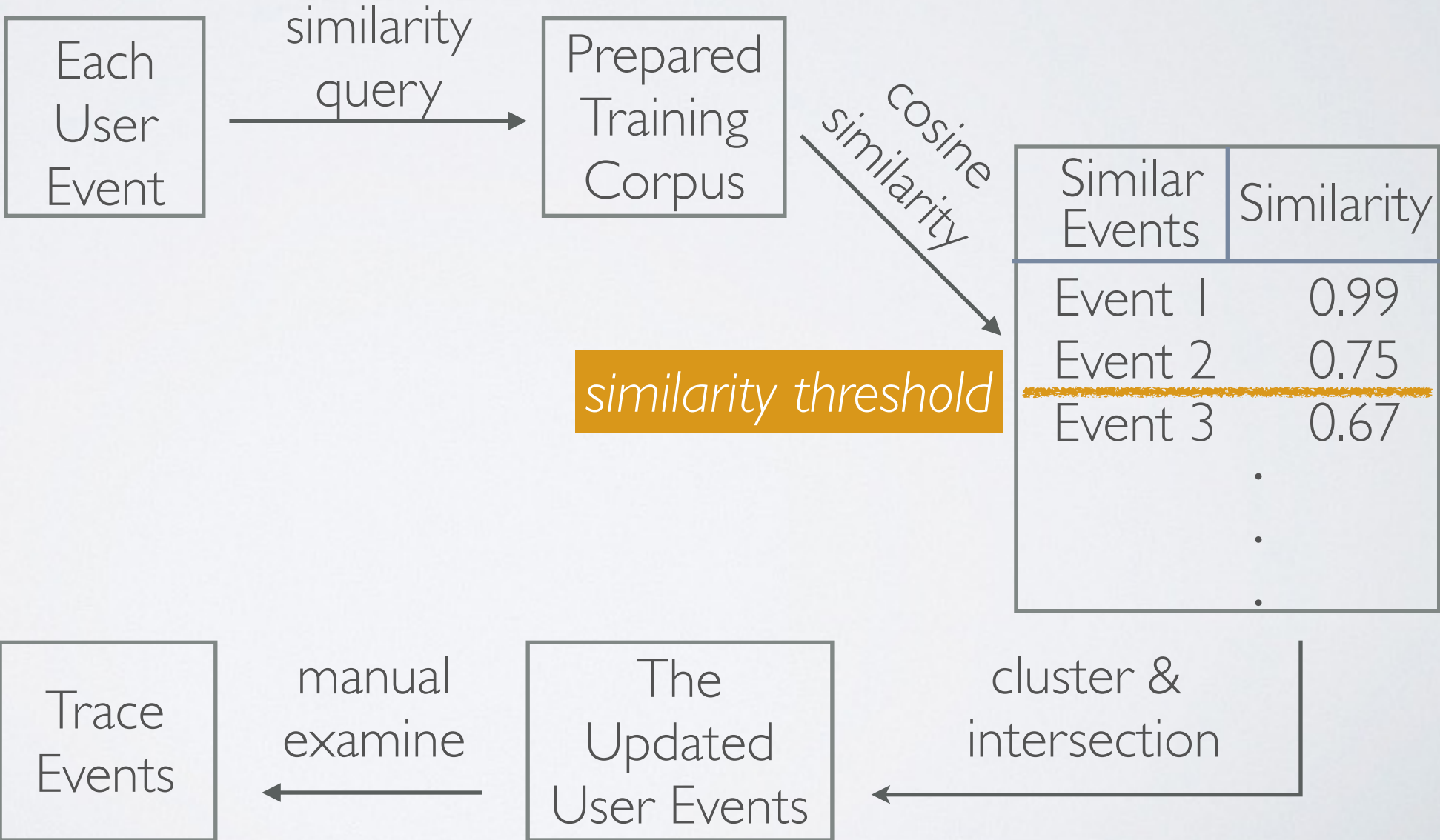
Model Inference Framework

Data Mining

```
dictionary = corpora.  
corpus = [dictionary.  
tfidf = models.TfidfM  
corpus_tfidf = tfidf[  
lsi = models.LsiModel  
#lda = models.LdaMode  
index = similarities.
```

3

II. Clustering Similar User Events



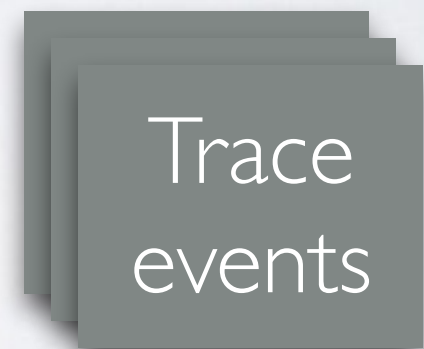
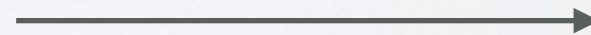
Model Inference Framework

Data Mining

```
dictionary = corpora.  
corpus = [dictionary.  
tfidf = models.TfidfM  
corpus_tfidf = tfidf[  
  
lsi = models.LsiModel  
#lda = models.LdaMode  
  
index = similarities.
```

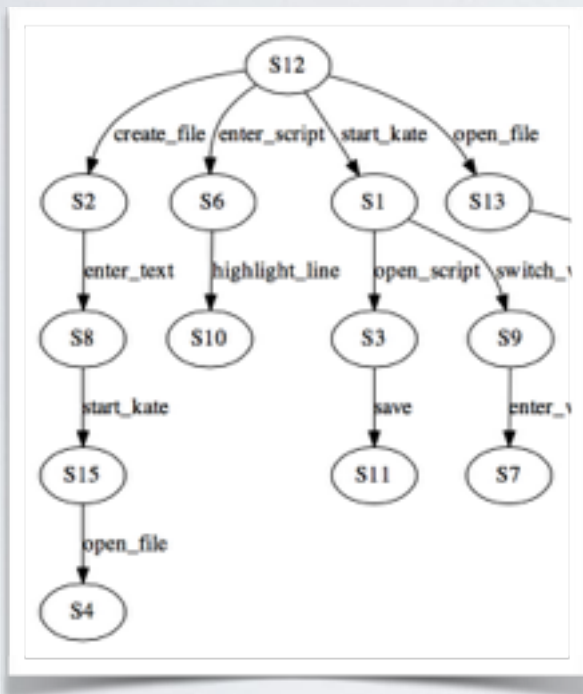
3

- I. Preparing Training Corpus
- II. Clustering Similar User Events
- III. Mapping Trace Events



Model Inference Framework

Search-Based
Model Generation



Multi-objective Fitness Functions

1. **Minimise over-approximation**

minimise the number of trace sequences generated from a model that do not correspond to any existing execution traces

2. **Minimise under-approximation**

minimise the number of trace sequences that are excluded from a model

3. **Minimise the number of states in a model**

Tonella, P., Marchetto, A., Nguyen, D.C., Jia, Y., Lakhota, K., Harman, M.: Finding the Optimal Balance between Over and Under Approximation of Models Inferred from Execution Logs. In: Proceedings of IEEE 5th International Conference on Software Testing, Verification and Validation (ICST), Montreal, QC, Canada, IEEE (17-21 April 2012) 21–30

Experimental Setting

Subject: Kate bug reports - KDE Bugtracking system

5583 bug issues reported since 2000 - 1/8 721 STR pattern

Search algorithms: multi-objective GA, NSGA-II

Benchmark tool: KLFA

Three Research Questions

RQ0: What are the prevalence and the characteristics of the trace events generated?

RQ1: What are the performance of multi-objective optimisation compared to the benchmark model inference technique, *KLFA*, in terms of the hypervolume, running time and the number of solutions?

RQ2: What is the fault revealing ability of the models inferred?

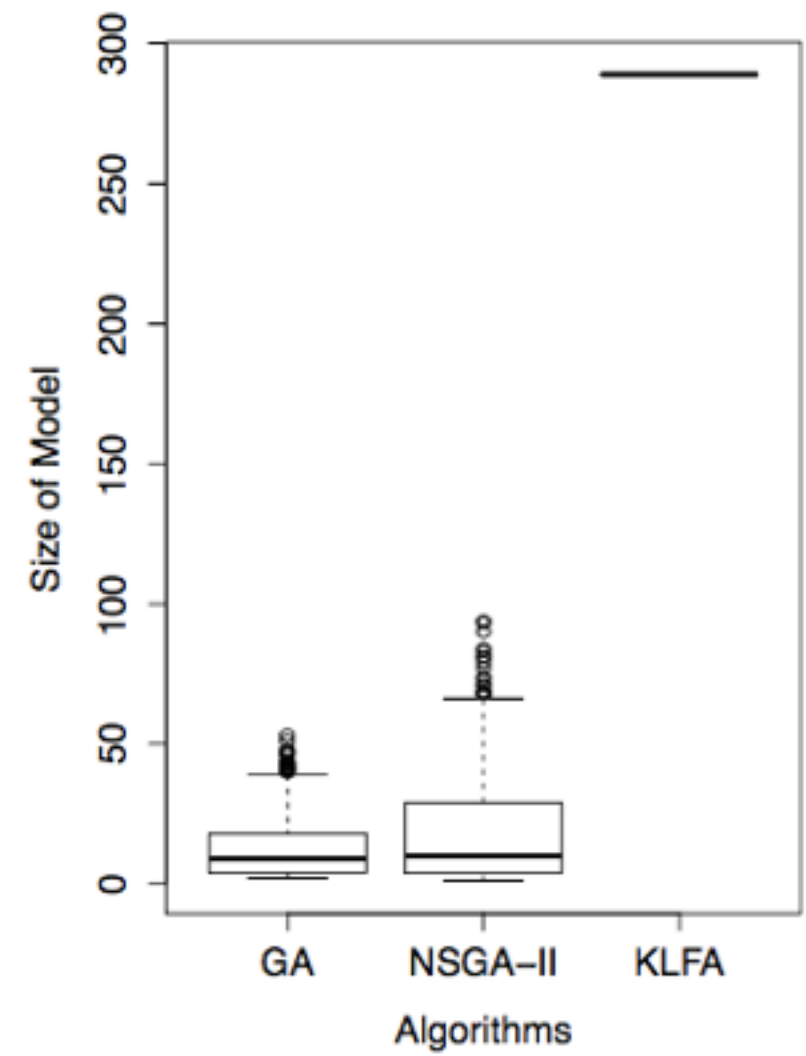
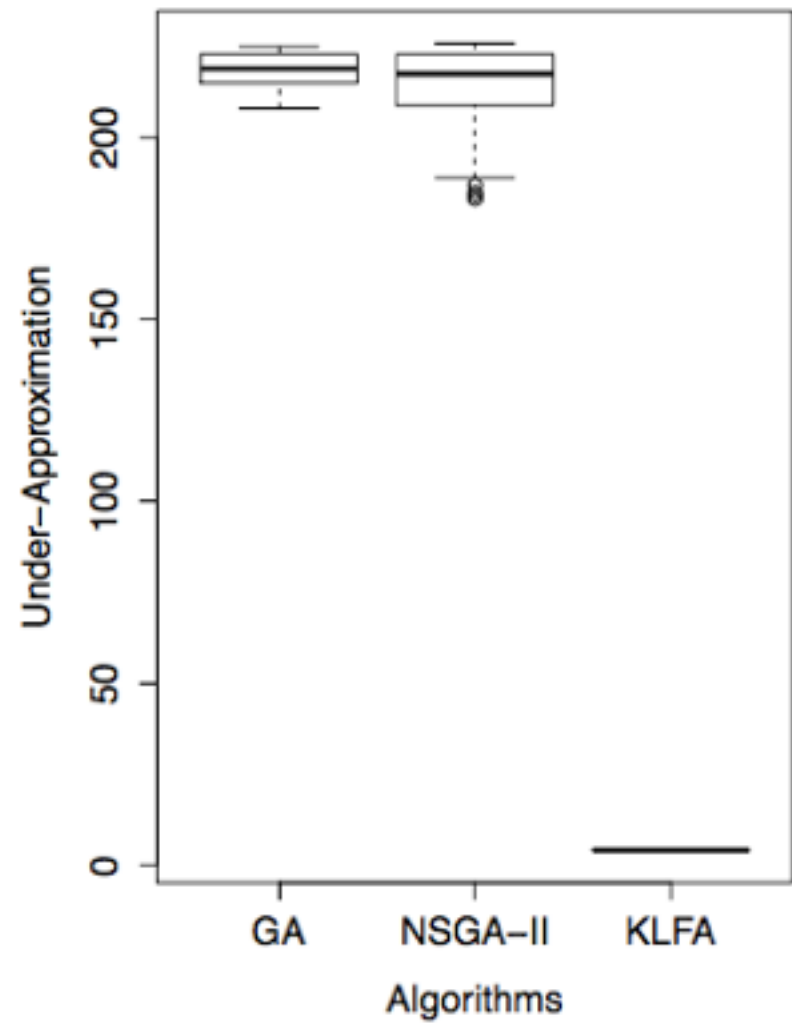
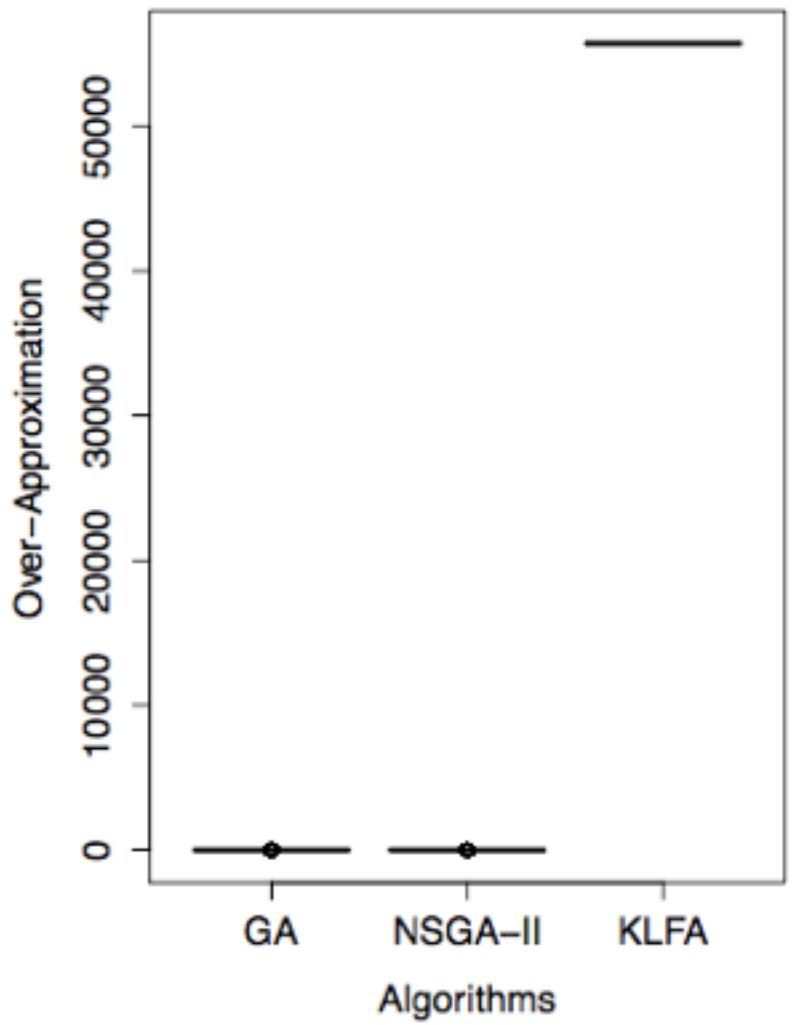
RQ0 Example of Trace Events Generated

721 bug reports → 452 user events files → 265 unique trace events

Category	Basic Operation	Text Editing	Programming
Examples	Start_Kate open_multiple_files score_screen drag_cursor resize_window close_file	copy_paste_text change_input_method fold_section find_replace captialize_text set_bookmark_color	select_haskell_mode show_javascript_console check_regular_expression fold_function check_indentation enter_vi_command
Category	Configuration	Plugins	Shortcut
Examples	change_keyboard_setting change_background_color change_print_margin change_print_page_range enable_command_line enable_static_word_wrap	enable_plugin_quickswitcher enable_plug_xml enable_plugin_spellcheck enable_plugin_tabbar enable_plugin_terminal enable_plugin_treeview	ctrl_l ctrl_g ctrl_o ctrl_r alt_right alt_tab

RQ1 Performance of the Algorithms - Three Objectives

Performance Algorithm	Objectives - Mean (Min, Max)		
	Over Approximation	Under Approximation	Size of Model
GA	2 (0, 66)	219 (208, 225)	13 (2, 53)
NSGA-II	0.1 (0.0, 7)	215 (183, 226)	19 (1, 94)
KLFA	55707	4	289



KLFA

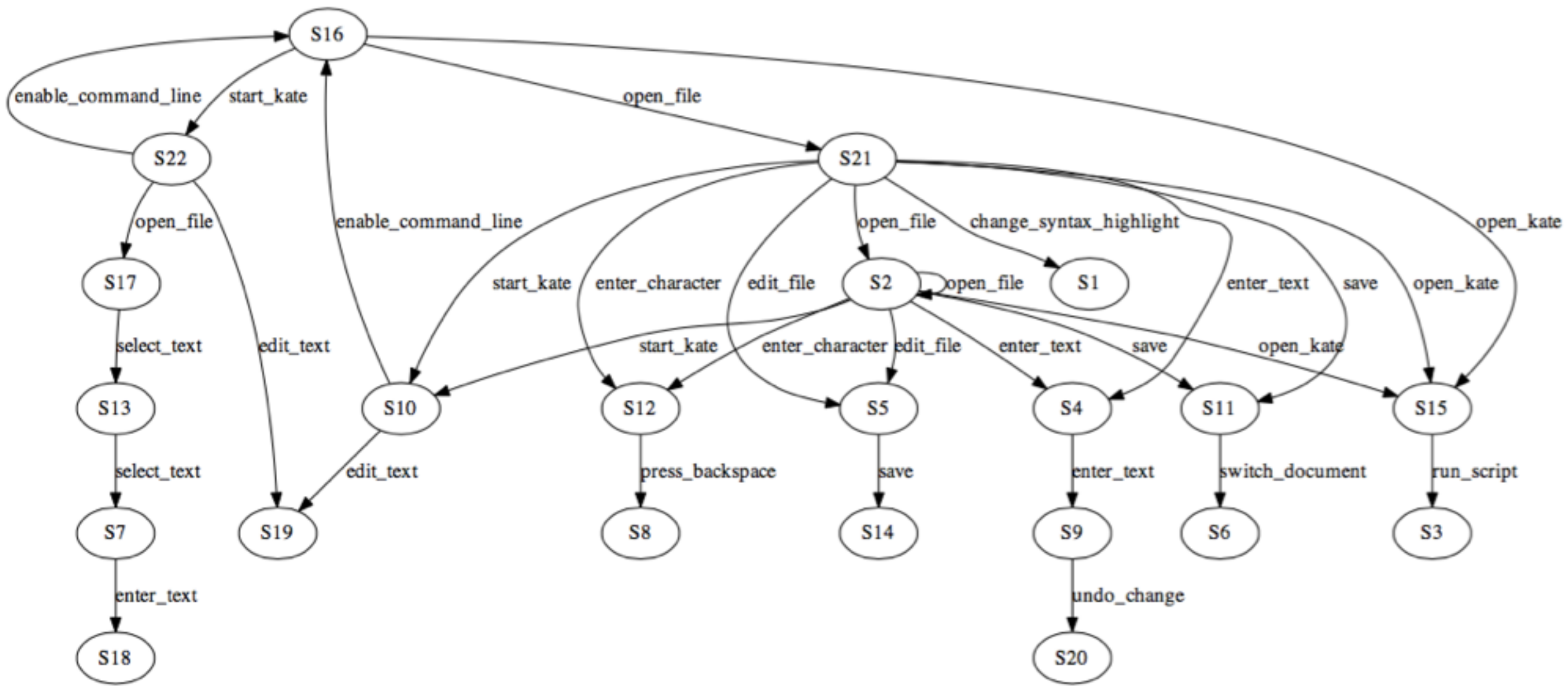


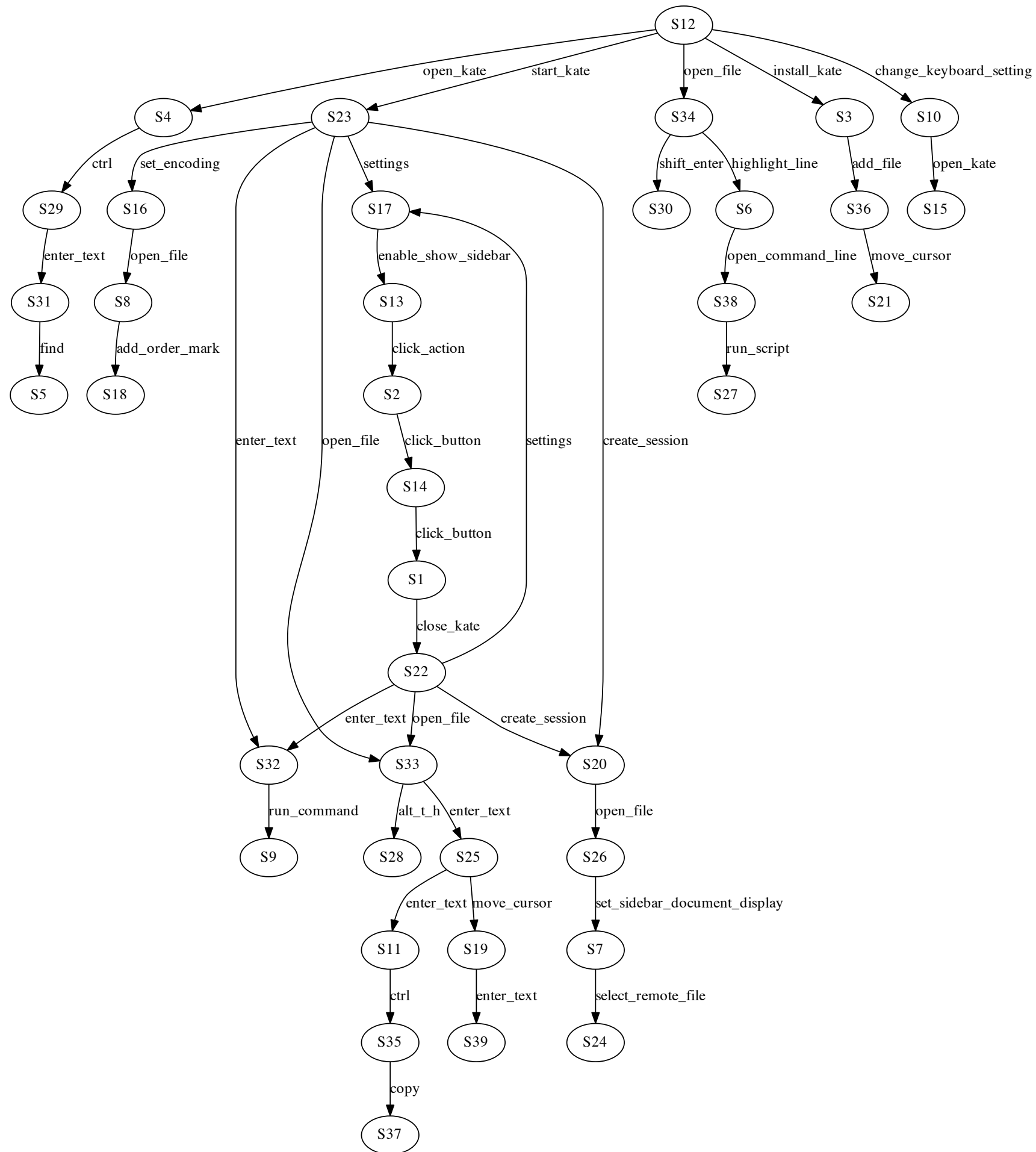
KLFA



KLFA







RQI Performance of the Algorithms - Quality Metrics

Performance Algorithm	Quality Metrics - Mean	
	Running Time	No. of Solutions
GA	3239.66s	25
NSGA-II	2341.14s	17
KLFA	556.30s	1

Algorithm (x)	Algorithm (y)	Hypervolume	
		Cliff's Method <i>p-value</i>	Vargha-Delaney Effect Size \hat{A}_{12}
GA	NSGA-II	1e-04	0.06
GA	KLFA	1e-04	1.00
NSGA-II	KLFA	1e-04	1.00

RQ2 Fault-Revealing Ability of the Model

Divided 452 user events files into training and test sets based on submission time.

Training set: 226 user events files from July 2009 to Oct. 2012

Test set: 226 user events files from Nov. 2012 to Feb. 2015

checking the number of trace events, which are in the test set, accepted by the models generated by the training set

If a bug trace event is accepted by a model, the model can be used to generate test trace sequence to capture this bug

	Avg. # Traces (L = 4)	Avg. # Bugs Pareto Front	Total # Bugs	Avg. Test per bug revealed
GA	147	8	16	18
NSGA-II	116	6	22	19
KLFA	55,906	30	30	1863

Inferring Test Models from Kate's Bug Reports using Multi-objective Search

Model Inference Framework



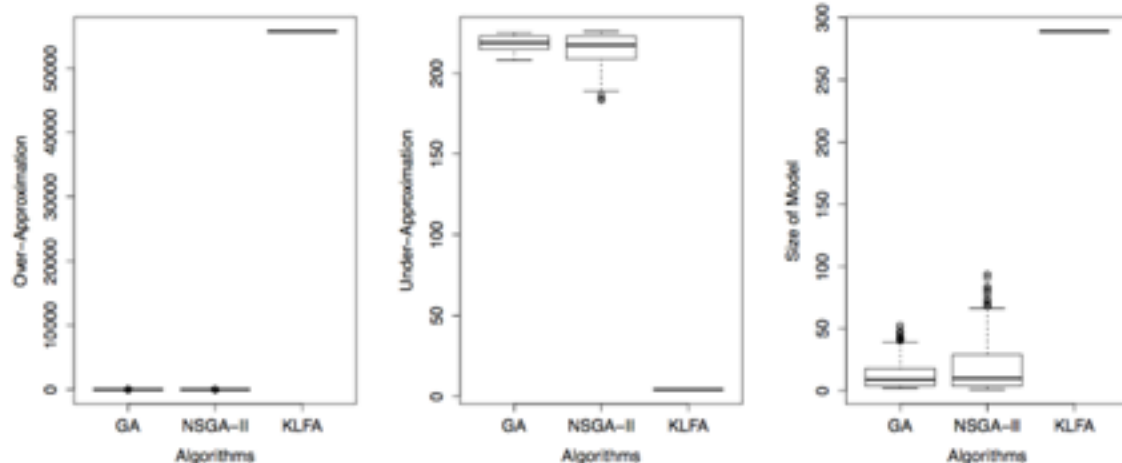
RQ0 Example of Trace Events Generated

721 bug reports → 452 user events files → 265 unique trace events

Category	Basic Operation	Text Editing	Programming
Examples	Start_Kate	copy_paste_text	select_haskell_mode
	open_multiple_files	change_input_method	show_javascript_console
	score_screen	fold_section	check_regular_expression
	drag_cursor	find_replace	fold_function
	resize_window	capitalize_text	check_indentation
	close_file	set_bookmark_color	enter_vi_command
Category	Configuration	Plugins	Shortcut
Examples	change_keyboard_setting	enable_plugin_quickswitcher	ctrl_l
	change_background_color	enable_plugin_xml	ctrl_g
	change_print_margin	enable_plugin_spellcheck	ctrl_o
	change_print_page_range	enable_plugin_tabbar	ctrl_r
	enable_command_line	enable_plugin_terminal	alt_right
	enable_static_word_wrap	enable_plugin_treeview	alt_tab

RQ1 Performance of the Algorithms - Three Objectives

Performance Algorithm	Objectives - Mean (Min, Max)		
	Over Approximation	Under Approximation	Size of Model
GA	2 (0, 66)	219 (208, 225)	13 (2, 53)
NSGA-II	0.1 (0.0, 7)	215 (183, 226)	19 (1, 94)
KLFA	55707	4	289



RQ2 Fault-Revealing Ability of the Model

Divided 452 user events files into training and test sets based on submission time.

Training set: 226 user events files from July 2009 to Oct. 2012

Test set: 226 user events files from Nov. 2012 to Feb. 2015

checking the number of trace events, which are in the test set, accepted by the models generated by the training set

If a bug trace event is accepted by a model, the model can be used to generate test trace sequence to capture this bug

	Avg. # Traces (L = 4)	Avg. # Bugs Pareto Front	Total # Bugs	Avg. Test per bug revealed
GA	147	8	16	18
NSGA-II	116	6	22	19
KLFA	55,906	30	30	1863