A Fast Algorithm to Locate Concepts in Execution Traces

Soumaya Medini, Philippe Galinier, Massimiliano Di Penta,
Yann-Gaël Guéhéneuc, Giuliano Antoniol

SOCCER Lab. & Ptidej Team, Ecole Polytechnique de Montréal, Québec, Canada
University of Sannio, Italy

September 12, 2011

ÉCOLE POLYTECHNIQUE MONTRÉAL

Outline

Introduction

Concept location aims at identifying user-observable features and locating them within code regions

A typical scenario in which concept location takes part:

- A failure has been observed in a software system under certain execution conditions;
- Such conditions are hard to reproduce;
- An execution trace was saved during failure.

Result: An execution trace containing a failure is saved but we can not re-execute the same scenario.

Motivation and Problem Statement

- Concept location process as trace segmentation problem.
- Use textual content of methods to split execution trace into segments that implement concepts.

We attempt to automatically identify concepts in a single execution trace.

Motivation and Problem Statement

Asadi *et al.* [Asadi et al., 2010]: identify concepts in execution trace by finding cohesive and decoupled fragments of the trace.

Limitations

- Not scalable (thousands of methods).
- Stability problems: each run may produce a different concept assignment.

We propose a novel approach to overcome these limitations.

Background

Steps:

- Step 1: System instrumentation and trace collection;
- Step 2: Pruning and compressing traces;
- Step 3: Textual analysis of Method source code;
- Step 4: Search-based concept location.

Soumaya Medini, Philippe Galinier, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, Giuliano Antoniol

Background

Step 1: System Instrumentation and Trace Collection

- System instrumented using MODEC: tool to extract and model sequence diagrams.
- Trace is an ordered list of invoked methods.
- Trace includes labels manually set around parts of the code during the execution of the instrumented system.

Soumaya Medini, Philippe Galinier, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, Giuliano Antoniol

Background

Step 2: Pruning and Compressing Traces

- Pruning: Remove too frequent methods having invocations greater than Q3 + 2 x IQR.
- Compression: Remove repetitions of method invocations using Run Length Encoding (RLE) algorithm.

m1 m1 m1 m1 m1 → m1

m1 m2 m1 m2 m1 m2 → m1 m2

Background

Step 3: Textual Analysis of Method Source Code

- Extract terms from source code: identifiers and comments.
- Remove programming language keywords and english stop words.
- Split terms using Camel-Case.
- Perform stemming.
- Index terms and documents using the *tf-idf* indexing mechanisms.
- Apply Latent Semantic Indexing (LSI) to reduce the term-document matrix.

Our approach

Approach built upon a dynamic programming algorithm to:

- Compute the exact split of an execution trace into segments.
- Improve scalability.

Dynamic programming: method to solve a problem by dividing the problem into sub-problems that are recursively solved.

Our approach

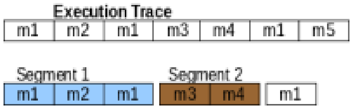
The solution of the problem is obtained by combining the solutions of the sub-problems.
We compute the quality of the segmentation of a trace split into K segments using the fitness function.

$$COH_l = \frac{\sum_{i=begin(l)}^{end(l)-1} \sum_{j=i+1}^{end(l)} \sigma(m_i,m_j)}{(end(l)-begin(l)+1)\cdot(end(l)-begin(l))/2} \quad (1)$$

$$COU_l = \frac{\sum_{i=begin(l)}^{end(l)} \sum_{j=1,j<begin(l)\ or\ j>end(l)}^{l} \sigma(m_i,m_j)}{(N-(end(l)-begin(l)+1))\cdot(end(l)-begin(l)+1)} \quad (2)$$

$$fitness = \frac{1}{K} \cdot \sum_{i=1}^{K} \frac{COH_i}{COU_i + 1} \quad (3)$$

Our approach



Possibilies

- New segment is added;
- The method is attached to the last solution segment.

Empirical Study

- RQ1: How do the performances of the GA and DP approaches compare in terms of fitness values, convergence times, and number of segments?
- RQ2: How do the GA and DP approaches perform in terms of overlaps between the automatic segmentation and the manually-built oracle, i.e., recall?
- RQ3: How do the precision values of the GA and DP approaches compare when splitting execution traces?

Empirical Study

Systems	**Scenarios**	**Original Size**	**Cleaned Sizes**	**Compressed Sizes**
ArgoUML v0.18.1	(1)Start, Create note, Stop	34,746	821	588
	(2) Start, Create class, Create note, Stop	64,947	1,066	764
JHotDraw v5.1	(1) Start, Draw rectangle, Stop	6,668	447	240
	(2) Start, Add text, Draw rectangle, Stop	13,841	753	361
	(3) Start, Draw rectangle, Cut rectangle, Stop	11,215	1,206	414
	(4) Start, Spawn window, Draw circle, Stop	16,366	670	433

Empirical Study

RQ1 Results

Compare the GA approach proposed by Asadi *et al.* [Asadi et al., 2010] with our approach.

System	Scenario	# of Segments		Fitness		Time (s)	
		GA	DP	GA	DP	GA	DP
ArgoUML	(1)	24	13	0.54	0.58	7,080	2.13
	(2)	73	19	0.52	0.60	10,800	4.33
JHotDraw	(1)	17	21	0.39	0.67	2,040	0.13
	(2)	21	21	0.38	0.69	1,260	0.64
	(3)	56	20	0.46	0.72	1,200	0.86
	(4)	63	26	0.34	0.69	240	1.00

The DP approach performs significantly better than the GA approach.

Empirical Study

RQ2 and RQ3 Results

Compare DP splitting results and manually splitted trace.

System	Scenario	Feature	Jaccard		Precision	
			GA	DP	GA	DP
ArgoUML	(1)	Create Note	0.33	0.87	1.00	0.99
	(2)	Create Class	0.26	0.53	1.00	1.00
	(2)	Create Note	0.34	0.56	1.00	1.00
JHotDraw	(1)	Draw Rectangle	0.90	0.75	0.90	1.00
	(2)	Add Text	0.31	0.33	0.36	0.39
	(2)	Draw Rectangle	0.62	0.52	0.62	1.00
	(3)	Draw Rectangle	0.74	0.24	0.79	0.24
	(3)	Cut Rectangle	0.22	0.31	1.00	1.00
	(4)	Draw Circle	0.82	0.82	0.82	1.00
	(4)	Spawn window	0.42	0.44	1.00	1.00

Conclusion

- We reformulated the trace segmentation problem as a dynamic programming (DP) problem:
  - We showed that we can benefit from the overlapping sub-problems
  - We obtained a dramatic gain in performance reusing computed scores of intervals and segmentation scores.
- The DP approach reuses pre-computed cohesion and coupling values among subsequent segments of an execution trace, which is not possible using GA.
- Results showed that the DP approach significantly out-performed the GA approach in terms of:
  - The times required to produce the segmentations;
  - The scalability.

Future Work

- Validate the scalability of the DP trace segmentation approach using larger traces.
- Use more traces obtained from different systems, to verify the generality of our findings.
- Complement the approach with segment labeling to make the produced segments more suitable for program-comprehension activities.

Asadi, F., Penta, M. D., Antoniol, G., and Guéhéneuc, Y.-G. (2010).
A heuristic-based approach to identify concepts in execution traces.
In *Proceedings of the European Conference on Software Maintenance and Reengineering*, pages 31–40. IEEE Computer Society Press.