Hyperheuristic Observation Based Slicing of Guava

Seongmin Lee[1] and Shin Yoo[1]

Korea Advanced Institute of Science and Technology
Republic of Korea

Abstract. Observation Based Slicing is a program slicing technique that depends purely on the observation of dynamic program behaviours. It iteratively applies a deletion operator to the source code, and accepts the deletion (i.e. slices the program) if the program is observed to behave in the same was as the original with respect to the slicing criterion. While the original observation based slicing only used a single deletion operator based on deletion window, the catalogue of applicable deletion opeartors grew recently with the addition of deletion operators based on lexical similarity. We apply a hyperheuristic approach to the problem of selecting the best deletion operator to each program line. Empirical evaluation using four slicing criteria from `Guava` shows that the Hyperheuristic Observation Based Slicing (HOBBES) can significantly improve the effeciency of observation based slicing.

1 Introduction

Program slicing aims to delete parts of the source code that does not affect the value of a specific variable at a point of interest [8]. While many applications, including testing [4], debugging [1]. maintenance [5], and program comprehension [6], have been proposed, program slicing suffered from limitations in scalability and lack of support for multi-lingual systems: both due to the fact that traditional slicing techniques rely heavily on static dependency analysis.

Observation Based Slicing (ORBS) [2, 3] is a new slicing technique that is purely dynamic and language independent. The intuition behind ORBS is that program slicing can be simply conceived as a series of deletions that preserves the behaviour of the program. The original ORBS iteratively considered deletions of consecutive lines. Recently, new deletion operators, based on lexical similairity, have also been introduced, increasing the pool of deletion operators for ORBS.

This paper evaluates a Hyperheuristic Observation Based Slicing (HOBBES). HOBBES applies deletion operators iteratively at each program line, but it uses a hyperheuristic approach to choose the next deletion operator. We formulate an online selective hyperheuristic approach using all available deletion operators as the lower level heuristic. The results of the empirical study using four slicing criteria from `Guava` project suggest that HOBBES can bring the best of both worlds: HOBBES can finish the given number of iterations sigficantly faster than Window-ORBS, while being able to delete comparable numbers of lines.

2 HOBBES: Hyperheuristic Observation Based Slicing

2.1 Observation Based Slicing

ORBS is not only language independent [2] but also can slice programs [3] or even graphics generated by Picture Description Languages [9] that traditional slicers cannot handle. This is because it decides whether to delete certain lines or not based on dynamic observation rather than static dependency analysis.

The deletion operator used by the original ORBS is called a Window-deletion (we hereby call the original ORBS with Window-deletion as W-ORBS): if deleting a single line results in failure (in either compilation or preservation of execution trajectories), it incrementally attempts to delete up to n consecutive lines, n being a parameter to the operator. This way, W-ORBS can delete lines that can only be deleted together (such as openning and closing curly brackets in `C`.). While the W-ORBS can successfully slice various programs, one of its major limitations is the time efficiency. For each line, W-ORBS may attempt up to n compilations and executions before accepting its deletion.

2.2 Deletion Operators Based on Lexical Similarity

Recently, a new group of deletion operators, based on lexical similarity in the source code, have been introduced [7]. Vector Space Model (VSM) deletion operator (hereby called VSM-deletion) represents all source code lines in VSM, and attempts to delete the current line under consideration as well as all other lines that are within the distance δ from the current line. Latent Dirichlet Analysis (LDA) deletion operator (hereby called LDA-deletion) works similarly, but uses LDA-based topic modeling to measure distances between source code lines. With both operators, the intuition is that the lines, that are lexically similar with each other, are likely to have a dependency, so they should be deleted together. Both operators have been shown to provide an attractive cost-benefit trade-off: while they produce larger slices, they are also significantly faster than W-ORBS.

While ORBS using VSM- or LDA-deletion provide better time efficiency compared to W-ORBS, the new operators only delete about 25% of the lines deleted by W-ORBS. This is because neither VSM- nor LDA-deletion can delete lines that are not related by lexical similarity together.

2.3 Algorithm of HOBBES

Algorithm 1 presents HOBBES. It takes the source program $\mathcal{P}$, a slicing criterion (v: variable, l: line index), a set of deletion operators $D = \{\mathrm{D1}, ..., \mathrm{D}N\}$. After instrumenting the slicing criterion and establishing the original trajectory V, it initializes the selection probability of each deletion operator.

HOBBES iteratively attempts to delete the source code, choosing a deleteion operator to apply at each line based on the corresponding selection probability. A chosen deletion operator creates a candidate slice: depending on the success of compilation and preservation of trajectories, HOBBES decides whether to

Algorithm 1: HOBBES

```
   input  : Source program, 𝒫 = {p_1, ..., p_n}, slicing criterion, (v, l, ℐ), Set of deletion
            operators, D = {D1, ..., Dn}
   output: A slice, S, of 𝒫 for (v, l, ℐ)
 1 O ← Setup(𝒫, v, l)
 2 V ← Execute(Build(O), ℐ)
 3 𝒟 ← InitOperator (D)                                   /* 𝒟 is set of (Dk, P(Dk)) */
 4 repeat
 5 |   deleted ← False
 6 |   i ← 1
 7 |   while i ≤ Length (O) do
 8 |   |   Dcurr ← SelectOperator(𝒟)
 9 |   |   O' ← Dcurr(O)
10 |   |   compile, execute, line_cnt ← DeleteAttempt(O', V)
11 |   |   𝒟 ← UpdateScore(𝒟, Dcurr, compile, execute, line_cnt)
12 |   |   if execute then
13 |   |   |   O ← O'
14 |   |   |   deleted ← True
15 |   |   end
16 |   |   i ← i + 1
17 |   end
18 until ¬deleted
19 return O
```

accept the candidate slice (i.e. the chosen deletion) and updates the selection probability of the chosen deletion operator.

2.4 Studied Deletion Operators

The library of deletions operator consists of 12 different deletion operators. We break down the original Window-deletion operator to four individual deletion operators with fixed size deletion windows: this results in four fixed Window-deletion operators that delete one, two, three, and four consecutive lines respectively. The remaining operators are VSM- and LDA-deletion operators with $\delta = \{0.9, 0.8, 0.7, 0.6\}$. We fixed the topic size of the LDA-deletion operators to 500 based on previous results; the LDA approach also generates the topic model only once at the beginning (see the previous work [7] for more details).

2.5 Selective Hyperheuristic

The selection probability of a deletion operator Dk, $P(\text{Dk})$ is initialized as $\frac{1}{|D|}$ by InitOperator. The function SelectOperator is a *roulette wheel selection* based on the probabilities.

The function UpdateScore updates $P(\text{Dk})$ as follows:

$$newP(\text{Dk}) = \begin{cases} \omega_{comp} \cdot P(\text{Dk}) & \text{when compile fails} \\ \omega_{exec} \cdot P(\text{Dk}) & \text{when compile suceeds and execution fails} \\ (1 + \log_{10} l) \cdot P(\text{Dk}) & \text{otherwise} \end{cases}$$

The penalty values, ω, for the compilation or execution failures, are set as $\omega_{comp}, \omega_{exec} \in [0, 1)$, $\omega_{comp} \leq \omega_{exec}$. Here, l denotes the number of lines deleted by the chosen operator (note that $\log_{10} l > 0$). In this study, we set ω_{comp} as 0.8 and ω_{exec} as 0.9. UpdateScore linearly normalizes the selection probabilities so that $\Sigma_{\text{K}} P(\text{Dk})$ is always 1.0.

Table 1. Result of Compile, Execute, Deletion per Time

		Iter1			Iter2			Iter3			Iter4			Iter5		
Subject	Strategy	C	E	D/T	C	E	D/T	C	E	D/T	C	E	D/T	C	E	D/T
escape1	HOBBES	502	66	0.20	926	104	0.13	1321	135	0.11	1699	165	0.09	2060	192	0.09
	W-ORBS	1711	183	0.10	3137	267	0.06	4523	342	0.04	5840	415	0.03	NA	NA	NA
escape2	HOBBES	1332	214	0.21	2424	309	0.15	3430	388	0.12	4384	455	0.11	5289	516	0.09
	W-ORBS	4179	655	0.13	7383	922	0.08	10436	1159	0.06	13460	1390	0.05	14116	1558	0.05
net1	HOBBES	513	70	0.17	955	114	0.11	1374	154	0.09	1771	189	0.08	2154	224	0.07
	W-ORBS	1759	189	0.09	3251	280	0.06	4707	364	0.04	6141	448	0.03	7174	517	0.03
net2	HOBBES	1341	222	0.20	2444	324	0.14	3460	402	0.11	4425	473	0.10	5346	536	0.09
	W-ORBS	4332	667	0.11	7781	963	0.07	11077	1237	0.05	14337	1504	0.04	14993	1672	0.04

3 Experimental Setup

3.1 Research Questions

We ask the following research questions:
RQ1: *How efficient is HOBBES compare to W-ORBS?* Previous work [7] showed that Window-deletion and the lexical similarity based deletion exhibit different cost-benefit trade-offs. RQ1 investigates whether using the selective hyperheuristic can improve the time efficiency of ORBS. We answer RQ1 by comparing the number of deleted lines, as well as the time the slicing took, between HOBBS and W-ORBS.
RQ2: *Does HOBBES actually use all deletion operators adaptively?* That is, does any single deletion operator exhibit dominant usage? We check whether HOBBES makes use of all operators adaptively by tracing the selection probabilities of each operator throughout the slicing operation.

3.2 Subjects, Configuration, and Environment

The slicing subjects have been chosen from the Guava library. We select two packages, `com.google.common.escape` and `com.google.common.net`, each with 590 and 1,569 LOCs. We choose 2 slicing criteria for each subjects. Since W-ORBS always produces the same slice for a deterministic program, we execute W-ORBS only once for each criterion. We repeat HOBBES 10 times to cater for the stochasticity in selection as well as in the LDA process.

The experiments have been performed on machines with Intel Core i7-6700K running Ubuntu 14.04.5 LTS. ORBS has been implemented in Python, whereas the subject programs have been built using Java version 1.8.

4 Results

RQ1. Efficiency of HOBBES: Figure 1 shows the results of W-ORBS and HOBBES for the four slicing criteria. The x-axis represent the slicing iterations, going up to the number of iterations W-ORBS requires to terminate. The barplots and lines represent the cumulative numbers of deleted lines and execution time, respectively. The result shows that, on average, HOBBES can delete about 71% of the number of lines that W-ORBS deletes. However, HOBBES only takes about 30% of the time spent by W-ORBS.

Fig. 1. Comparison Between W-ORBS, HOBBES

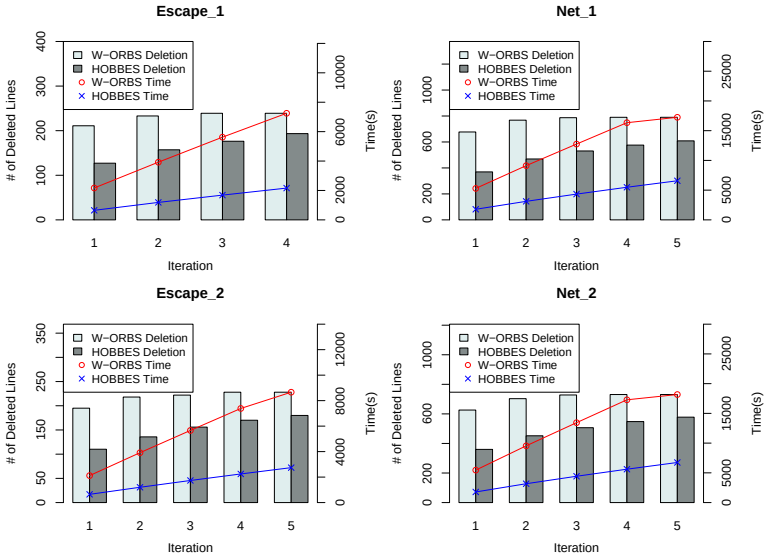


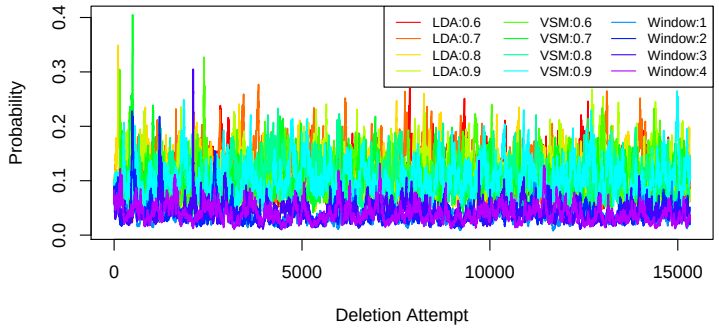
Table 1 shows the detailed results of W-ORBS and HOBBES until their fifth iteration. HOBBES performs fewer compilations and executions than W-ORBS, while showing higher time efficiency (i.e. more deletions per time). Answering **RQ1**, we report that HOBBES can improve the time efficiency of W-ORBS.

RQ2. Participation of Deletion Operators: Figure 2 shows how the selection probabilities of deletion operators change throughout the slicing of `net_1`. No deletion operator dominates the selection; also, there exist several peaks of different colours. We interpret this as each deletion operator being used at different stages by HOBBES. Note that the probabilities for both VSM- and LDA-deletion operators increased early in the slicing because these operators are not limited in the number of lines they can delete. However, we also observe that Window-deletion operators are also selected at different times.

5 Conclusion

We introduce a hyperheuristic version of ORBS, called HOBBES. HOBBES applies a selective hyperheuristic to choose a deletion operator iteratively at each source code line. A case study of HOBBES on two packages in the `Guava` library, using 12 deletion operators, shows that HOBBES can delete 71% of the number of lines deleted by W-ORBS, using only 30% of the time. Future work will investigate more diverse deletion operators as well as more sophisticated selective hyperheuristic algorithm.

Fig. 2. Change of Probability of Deletion Operators



References

1. Agrawal, H., DeMillo, R.A., Spafford, E.H.: Debugging with dynamic slicing and backtracking. Software Practice and Experience 23(6), 589–616 (Jun 1993)
2. Binkley, D., Gold, N., Harman, M., Islam, S., Krinke, J., Yoo, S.: ORBS: Language-independent program slicing. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering. pp. 109–120. FSE 2014 (2014)
3. Binkley, D., Gold, N., Harman, M., Islam, S., Krinke, J., Yoo, S.: ORBS and the limits of static slicing. In: Proceedings of the 15th IEEE International Working Conference on Source Code Analysis and Manipulation (2015)
4. Binkley, D.W.: The application of program slicing to regression testing. Information and Software Technology Special Issue on Program Slicing 40(11 and 12), 583–594 (1998)
5. Gallagher, K.B., Lyle, J.R.: Using program slicing in software maintenance. IEEE Transactions on Software Engineering 17(8), 751–761 (Aug 1991)
6. Korel, B., Rilling, J.: Program slicing in understanding of large programs. In: 6^{th} IEEE International Workshop on Program Comprenhesion (IWPC'98). pp. 145–152. IEEE Computer Society Press, Los Alamitos, California, USA (1998)
7. Lee, S., Yoo, S.: Using source code lexical similarity to improve efficiency of observation based slicing. Tech. Rep. CS-TR-2017-412, School of Computing, Korean Advanced Institute of Science and Technology (January 2017)
8. Weiser, M.: Program slicing. In: 5^{th} International Conference on Software Engineering. pp. 439–449. San Diego, CA (Mar 1981)
9. Yoo, S., Binkley, D., Eastman, R.: Observational slicing based on visual semantics. Journal of Systems and Software 129, 60–78 (2016)