Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 50 (2008) 1189–1209
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Locating dependence structures using search-based slicing

Tao Jiang *, Nicolas Gold, Mark Harman, Zheng Li

King's College London, Strand, London WC2R 2LS, UK

Received 6 December 2006; received in revised form 23 October 2007; accepted 1 November 2007
Available online 17 November 2007

Abstract

This paper introduces an approach to locating dependence structures in a program by searching the space of the powerset of the set of all possible program slices. The paper formulates this problem as a search-based software engineering problem. To evaluate the approach, the paper introduces an instance of a search-based slicing problem concerned with locating sets of slices that decompose a program into a set of covering slices that minimize inter-slice overlap. The paper reports the result of an empirical study of algorithm performance and result-similarity for Hill Climbing, Genetic, Random Search and Greedy Algorithms applied to a set of 12 C programs.

Keywords: Program slicing; Search-Based Software Engineering

1. Introduction

Dependence analysis has been applied to several stages of the software engineering process, such as program restructuring [21,53], program comprehension [26], regression testing [12] and program integration [42]. It can also be an effective way of understanding the dependence structure of a program [13,52] and a measurement of dependence-related attributes such as cohesion and coupling [10,60]. For these applications, sets of slices are used to reveal interesting properties of the program under analysis, such as the presence of dependence clusters and the cohesive (and less cohesive) parts of the program.

The advent of commercial, scalable and robust tools for slicing such as Grammatech's CodeSurfer [36] makes it possible to construct all possible slices for large programs in reasonable time. By constructing the set of all slices of a program, it is possible to analyse the dependence structure of the program. This allows slicing to be used to capture the dependence of every point in the program, allowing analysis of the whole program dependence structure. This raises an interesting research question:

> "How can useful interesting dependence structures be formed in amongst the mass of dependence information available?"

In this paper, dependence is analysed using program slicing, and so this question is reformulated as:

> "Of the set of all possible slices of a program, which subsets reveal interesting dependence structures?"

Of course, for a program consisting of n program points, there will be n possible slices and, therefore, 2^n subsets of slices. Since the number of program points is always at least as large as the number of statements in the program, the powerset of all possible slices will be extremely large; too large to enumerate for any realistically sized program. This is merely a reflection of the mass of dependence information available and would need to be considered by any whole program dependence analysis. The overwhelming quantity of information motivates the search-based approach introduced in this paper.

The paper introduces an approach to location of dependence structures, founded on the principles of Search-Based Software Engineering (SBSE) [23,40]. Using this formulation, the problem becomes one of a search for a set of slices that exhibit interesting

* Corresponding author.
E-mail address: tao.jiang@kcl.ac.uk (T. Jiang).

0950-5849/$ - see front matter
doi:10.1016/j.infsof.2007.11.001

dependence structures. The choice of what constitutes an 'interesting dependence structure' is a parameter to the overall approach, making it highly flexible. In search-based software engineering, a fitness function is defined to capture such a property of interest. In the case of search-based slicing, it captures the properties of a dependence structure that make it interesting to a particular analysis.

The search process is realized by an algorithm that uses the fitness function to guide a search that seeks to find optimal or near optimal solutions with respect to the fitness function. In order to experiment with the search-based slicing approach, the paper presents the results of an implementation and associated empirical study into the search for slice sets that decompose a program into a set of slices that cover the program with minimal overlap. The fitness function used in the empirical study is motivated by work on slicing as a decomposition technique [34,73].

This instantiation of the search-based slicing approach formulates the decomposition problem as a set cover problem [31]. However, it must be stressed that this represents merely the *instantiation* of a parameter to the approach (the fitness function). The search-based slicing approach derives a great deal of flexibility from the fact that the fitness function (and therefore the property of interest) is merely a parameter; in order to search for a different kind of dependence structure, only the fitness function needs to be changed.

The paper reports the results of experiments with four different search algorithms for search-based slicing and presents the results of an empirical study involving 12 C programs. The empirical study aims to answer four related research questions:

(1) How well does each algorithm perform?
(2) How similar are the results produced by each algorithm?
(3) How can the results be visualized and what do they reveal?
(4) How efficiently can the best algorithm perform with large practical programs and for all the functions in programs?

The paper makes the following primary contributions:

(1) An approach that identifies dependence structures is introduced as a search problem over the powerset of the set of all possible program slices, allowing search-based algorithms to be used to search for interesting dependence structures.
(2) A fitness function is introduced that seeks to optimise the search towards solutions that decompose the program into a set of slices that collectively cover the whole program with minimal overlap. Four search algorithms are implemented in order to experiment with this fitness function.
(3) The results of an empirical study are reported, showing that the Greedy Algorithm performs better than Random, Hill Climbing and Genetic Algorithm approaches to the problem. This is an attractive finding, since Greedy Algorithms are extremely simple and efficient.
(4) A simple visualization is introduced to explore the results and their similarity. This shows a higher degree of similarity for the intelligent techniques over random search. This visual impression is augmented by computational analysis of results. The similarity of results for intelligent search provides that the results are consistent and meaningful.
(5) The visualization also has an interesting side effect, which may be a useful spin off: the presence of code clones becomes visually striking in some of the examples. However, clone detection is not the focus of this paper.
(6) The paper also reports results on redundancy. That is how often a slice is completely included by another one. The results suggest that redundancy phenomena are universal in 12 programs. However, it is shown that this redundancy does not affect the Greedy Algorithm advocated in the paper.
(7) Based upon the performance comparison with four search algorithms, the Greedy Algorithm is further applied to six larger programs to decompose each function of each program. The results show that majority of functions can be decomposed into sets of slices efficiently.

The data used in this paper are made available to the research community to facilitate replication at http://www.dcs.kcl.ac.uk/pg/jiangtao/.

The rest of the paper is organised as follows: Section 2 presents the problem description in more detail, while Section 3 introduces the search-based algorithms and their application to the problem. Sections 4 and 5 present the results of the empirical study. Sections 6 and 7 present related work and conclusions.

2. Problem description

The goal is to identify dependence structures by searching the space of all subsets of program slices. In this paper, static backward slicing is used, but the approach is not confined merely to static backward slicing; it can be used with any analysis that returns a set of program points (thereby including all forms of program slicing).

As an illustrative example, consider a program that has only 8 program points. Table 1 gives all the slices of this hypothetical example in terms of each program point as slicing criteria.

The table represents the value of each slice. In this table, a 1 represents a program point that is included in the slice, while a 0 represents a program point that is not included in the slice. In this situation, a good decomposition would be the set $\{1,5,7\}$, rather than $\{1,2,7\}$, $\{6\}$ or any other sub-