Search–Based Amorphous Slicing

Deji Fatiregun
King's College London
Strand, London
WC2R 2LS, UK.
fatir@dcs.kcl.ac.uk

Mark Harman
King's College London
Strand, London
WC2R 2LS, UK.
Mark@dcs.kcl.ac.uk

Robert M. Hierons
Brunel University
Uxbridge, middlesex
UB8 3PH, UK.
Rob.Hierons@brunel.ac.uk

Keywords: slicing, transformation, search based software engineering

Abstract

Amorphous slicing is an automated source code extraction technique with applications in many areas of software engineering, including comprehension, reuse, testing and reverse engineering. Algorithms for syntax–preserving slicing are well established, but amorphous slicing is harder because it requires arbitrary transformation; finding good general purpose amorphous slicing algorithms therefore remains as hard as general program transformation.

In this paper we show how amorphous slices can be computed using search techniques. The paper presents results from a set of experiments designed to explore the application of genetic algorithms, hill climbing, random search and systematic search to a set of six subject programs. As a benchmark, the results are compared to those from an existing analytical algorithm for amorphous slicing, which was written specifically to perform well with the sorts of program under consideration.

The results, while tentative at this stage, do give grounds for optimism. The search techniques proved able to reduce the size of the programs under consideration in all cases, sometimes equaling the performance of the specifically-tailored analytic algorithm. In one case, the search techniques performed better, highlighting a fault in the existing algorithm.

1 Introduction

Program slicing is a technique for extracting parts of a program which affect a chosen set of variables of interest. By focusing on the computation of only a few variables the slicing process can be used to eliminate parts of the program which cannot affect these variables, thereby reducing the size of the program. The reduced program is called a slice. The traditional, syntax–preserving, paradigm of computing slices by a process of statement deletion is well studied, and there are well established algorithms for automated syntax–preserving slice construction [20, 34].

Amorphous slicing is a form of slicing, in which *any* transformation can be applied (not merely statement deletion). As a result the slice produced can be a lot smaller, but the algorithms for computing such slices are less well developed than those which use the statement deletion transformation alone. Furthermore, unlike traditional, syntax–preserving slices, the amorphous slices do not preserve the syntactic structure of the original program. All applications of slicing require slices to be as small as possible and many (such as reverse engineering) do not require syntax-preservation. For those applications where syntax preservation is unimportant, amorphous slicing is clearly attractive.

Slicing is useful in Reverse and Re- Engineering because it assists in many associated activities, for example Comprehension[12], Salvaging [8] and Regression Testing [1]. Amorphous Slices produces smaller slices but loses the syntactic connection to the original. However for Re-engineering, this loss of reduction is often inevitable and smaller slices are highly desirable. This makes amorphous slicing ideally suited to re-engineering.

Since amorphous slicing involves general program transformation, finding good general algorithms is as hard as finding good general transformation algorithms which is known to be a very challenging problem [6, 26]. Although there exist several analytic algorithms for amorphous slicing (using the program dependence graph [2] and the abstract syntax tree [18]), these are not general amorphous slicing algorithms. Rather, they are constructed to apply a specific set of transformation rules in a strategy which is optimised to producing amorphous slices for a certain class of programs.

This paper proposes a new method for generating amorphous slices for programs using search–based techniques. It is our hypothesis that searching for a series of meaning preserving transformations based upon an appropriate objective measure may result in the generation of good amorphous slices. We tackle the problem using four different search heuristics which will be described further - a Genetic Algorithm (GA), a Hill Climb Algorithm (HCA), a Systematic Search Algorithm (SSA) and a pure Random Search Algorithm (RSA).

The problem of computing amorphous slices is reformulated as a problem of removing redundant computations by transformation. The search explores the space of possible transformation sequences. The fitness function rewards those transformations which can reduce the redundancies in the program. The advantage of using search is that the search techniques are highly adaptive, producing a sequence of transformations tailored to the particular program to be sliced. In this way, the overall approach is, at once, both general and specific. It is a general approach because it can be deployed to slice arbitrary programs. However, application of the approach to each program to be sliced is, nonetheless highly specific; the transformation sequence which results is tailored to the problem of slicing that particular program.

The paper presents results from the application of the four search techniques and compares these with results obtained from the syntax-directed analytic slicing algorithm LinIAS [18]. The results presented here come from the application of these techniques without a significant degree of 'tuning'. Nonetheless, the results are very encouraging. The search techniques were always able to reduce the size of the original program. Since minimal slicing is not decidable (even in the restricted, syntax–preserving, paradigm [34]) *any* reduction in program size is useful.

The results also show that random search performs well. This indicates that there is 'more mileage' yet to be gained from the more sophisticated search techniques, such as Hill Climbing and Systematic Search. The systematic search also performs well, indicating that a hybrid approach which combines elements of systematic search with genetic algorithms and hill climbing may be effective.

Finally, a search-based approach was, at least on one occasion able to perform *better* than the analytic algorithm. This was highly surprising, because the analytic algorithm is known to perform well for this class of programs. In this case, the use of search revealed a bug in the algorithm used to construct syntax–preserving slices, which was a component of the amorphous slicing algorithm used in an implementation of the LinIAS system [18].

The primary contributions of this paper are to:

- Introduce the idea of using search heuristics to construct amorphous slices.
- Present initial results indicating the feasibility of the search-based approach to slice construction.
- Compare different search approaches for this problem, including a systematic search technique designed specifically for search-based transformation.

The rest of the paper is organised as follows: Section 2 describes traditional syntax–preserving slicing and amorphous slicing. Section 3 informs of the motivation behind using search techniques in generating amorphous slices and descirbes how search is applied. Section 4 describe the algorithms analysed in this work. Sections 5 and 6 detail the experiments carried out and provide a discussion of the results respectively highlighting interesting outcomes. Section 7 describes related work in this area while Section 8 describes future areas in this research. Section 9 presents the conclusions.

2 Slicing and Amorphous Slicing

Program Slicing is an automated source code extraction technique which produces a version of a program that preserves a projection of the original program's semantics [3, 11, 30, 34]. Traditionally, this projection is defined in terms of a subset of variables of interest and is constructed using the sole transformation of statement deletion [34]. The slice is therefore a subprogram which preserves a subcomputation.

Amorphous slicing can be thought of as a generalisation of slicing: the semantic requirement is retained, but the syntactic requirement is relaxed. This makes amorphous slicing more like transformation than traditional slicing. Similarly, amorphous slicing can be thought of as a generalization of program transformation: the semantic requirement to preserve the meaning of the program is relaxed to the requirement that the (projected) semantics *with respect to the slicing criterion* is preserved.

Transformations which tend to reduce program size are like slices in that they preserve the (projected) semantics of the program, but are unlike slicing because they do not preserve syntax. On the other hand, slicing is like transformation in the way it preserves the semantics of the program for some projection of the original semantics. Thus, *amorphous slicing* is the result of combining slicing and transformation.

It is widely believed that amorphous slices tend to be smaller than static slices because of the relaxed syntactic requirements placed on them and the fact statements and expressions may be transformed further using rules from a pool available to the amorphous slicer. Figure 1 shows an example of an amorphous slice and the possible further reduction in slice size when compared against syntax–preserving slices. Amorphous Slicers currently work using analytical algorithm that reduce dependencies, transforming program constructs. However it is also known that the problem of achieving minimal slices is undecidable. Therefore, it may still be possible to find even smaller slices than those being achieved by existing amorphous slicers. It is this reasoning that has fed the idea of a search-based approach.

That given a similar set of domain specific transformation rules as is available to an existing amorphous slicing system and an appropriate objective metric (slice size reduction), that the search heuristics may find sequences and combinations of the rules to return smaller slice sizes.

Researchers such as Tip [30], and Binkley and Gallagher [3] and De Lucia [11] have all provided surveys of Program Slicing: techniques and applications and slicing paradigms. Binkley and Harman [4] present a survey of empirical results on program slicing.

3 Amorphous Slicing as Search-Based Transformation

Inherent in the idea of amorphous slicing, is the application of transformation rules such as constant propagation, statement deletion, etc., in unfolding expressions. These specific transformation rules are selected from a pool using an appropriate selection strategy. Typically if the pool of transformation rules is sufficiently large, then the number of combinations of rules which may be applied at any node in the program is exponential.

The search space is thus wide enough for any number of heuristic search methods applied to explore the possibilities for finding sequences of good meaning–preserving transformation rules that would generate smaller program slices.

3.1 Reformulating Amorphous Slicing as a Redundancy Removal Problem

In order to compute valid amorphous slices that preserve the computation for a single variable of interest using a search–based approach, we re–formulate the problem as a redundancy removal problem by appending killing assignment statements to the program and then optimise by removing (transforming out) these redundant statements.

Given a variable x captured by our slicing criterion and a set of defined variables V in a program, such that, $V' = V - x$, to obtain a slice S which represents a projection of the semantics of a program P with respect to x in the slicing criterion, we need to ensure that our algorithm preserves the existence of x in S.

For all variables v in V', we append killing assignments such that: $v = C$ (where C is some arbitrary constant value). Figure 2 illustrates the reformulation as redundancy removal by the addition of killing assignments. [13] describes the idea of search-based transformations and the mapping between a sequence of transformation rules and the encoding system used in this work.

3.2 Fitness Measure

Subsequent to the reformulation of the problem as one of redundancy removal through the inclusion of the killing assignment statements to the statements, we select an appropriate fitness measure which our search procedure would optimise. The optimisation problem is to minimise the size of the amorphous slice computed, that is, the smaller the slice size obtained, the better the solution.

The fitness measure for our algorithms minimises the size of the source program (LoC) and is computed by subtracting the size of the slice computed from the size of the original program:

fitness of individual = length of original program - length of slice produced by individual

4 Search Algorithms

4.1 Genetic Algorithm

A Genetic Algorithm (GA) is a population-based search procedure, which starts with an initial random solutions (represented as chromosomes) called the population and evolves over several iterations or generations, such that the individuals (solutions) in successive generations have better or at least of no worse fitness values than those in preceding generations. An optimised individual is one which presents a more desirable solution to the given problem.

GAs imitate the natural process of evolution and use evolutionary operators such as crossover and mutation to alter the population across several generations toward optimality.

We employ a GA to evolve the sequences of transforms that would result in the best possible program. Our approach is to use the transformation sequence to be applied to the program as the individual to be optimised. Using the transformation sequence as the individual makes it possible to define crossover relatively easily. In our implementation, we combine two sequences of transformations using a single point crossover, selecting a point at random along the chromosome and swapping adjacent sides. The result is a valid transformation sequence and since all transformation rules are meaning preserving, so are all sequences of transformation rules.

`D := 2*r;`
`FaceArea := pi*r*r;`
`C := pi*D;`
`TempArea := pi * FaceArea;`
`SArea := 2*FaceArea+h*C;`
`slice := SArea;`
	`D := 2*r;`
	`FaceArea := pi*r*r;`
	`C := pi*D;`
	`SArea := 2*FaceArea+h*C;`
	`slice := SArea;`
		`slice:=2*pi*r*r+h*pi*2*r;`

Original	Syntax-Preserving slice	Amorphous slice

Figure 1. Illustrative example showing Amorphous Slicing Producing Thinner Slices by Removing Syntactic Restrictions. Slice computed w.r.t. variable `slice` at end.

`D := 2*r;`
`FaceArea := pi*r*r;`
`C := pi*D;`
`SArea := 2*FaceArea+h*C;`
`slice := SArea;`
	`D := 2*r;`
	`FaceArea := pi*r*r;`
	`C := pi*D;`
	`SArea := 2*FaceArea+h*C;`
	`C := 1;`
	`FaceArea := 1;`
	`D := 1;`
	`slice := SArea;`

Source Program: slicing criterion is variable `slice` at end-of-program	Re–Constructed Program with killing assignments appended to end-of-program

Figure 2. Re–structuring Source Program for Search–Based Transformations

4.2 Hill Climb Algorithm

In a local search method such as the Hill–Climbing algorithm (HC), one guesses a solution within the solution space and then moves toward a better solution closer to the goal.

We implemented a HC algorithm to find good transformation sequences. In our implementation, an initial sequence is generated randomly and serves as our starting point. The fitness of this individual is computed. The algorithm iterates through each neighbour to the current position and when a better individual is found, this individual replaces the old one as the current best individual. This process is repeated and if no better neighbour is found, we assume we have arrived at the top of the hill and the current solution remains our best.

The algorithm is restarted several times using a random sequence as the starting individual each time. The aim is that this would divert the algorithm from any local optima and increase the chances of finding a global solution.

4.3 Systematic Search Algorithm

In this section, we describe a Systematic Search Algorithm (SSA), which we implemented to optimize the search for an amorphous slice. In the SSA, we do not have any notion of an individual but each transformation rule (gene) is applied iteratively to a node in the program. If a particular rule is successful at that point, then it is repeated at the next node, otherwise a new rule is tried out at the specified node. This process is repeated until all the rules are examined per node and the program cursor reaches the end of source–code. The entire process may be restarted again over several iterations.

5 Experiment

The objective of the experiment was to determine whether search–based techniques could be used to compute valid amorphous slices. The algorithms presented were implemented in the Wide Spectrum Language (WSL) using transformation rules from the fermaT transformation workbench [31]. [13] lists some of the transformations rules implemented in the fermaT system and used in this work.

5.1 Experimental Set-Up

We examined the application of search–based methods to find transformations which would optimize programs to generate amorphous slices with respect to a given slicing criterion. The research included comparing the results from four search methods: Genetic Algorithm, Hill–Climb Algorithm, Systematic Search Algorithm and Random Search.

To apply our search methods, there is the need to distinguish the slicing criterion from other asssigned variables within the source by restructuring the program by way of adding killing assignments to the source (as described in section 3). For the individual based search methods, we define arbitrarily a fixed length of 20 genes per individual for the transformation sequences.

We implemented a standard Genetic Algorithm using single point crossover, a mutation rate of 7% and the tournament selection strategy for choosing mating parents. Each combination of genetic material between the parents results in the creation of a single offspring which may replace the parent with a worse fitness value. We define a constant population size of 50 and run the algorithm over 100 generations. The individual with the best fitness score is replicated across successive generations.

The experiment with the hill–climb algorithm was set up with multiple restarts, keeping track of the best individual across the different restarts. The HC Algorithm uses a first ascent technique. A neighbouring individual is analysed and its fitness value computed. If the fitness of the neighbour is better than that of our current position, the neighbour becomes our new current best position. Neighbourhood in our hill–climb algorithm is described as the mutation of a single gene in an individual (all other genes remain unchanged). The algorithm is restarted 10 times with a new random individual each time.

With the Systematic Search Algorithm, each transformation rule is applied iteratively at each node in the program. Because the algorithm works with single genes rather than individuals, a collection of sequences is built up as the algorithm progresses through each program node.

Lastly, we implemented a random search for transformation sequences, also using a fixed length sequence of 20 transformation rules per individual. The experiment is terminated after a fixed number of iterations and the individual with the best fitness score is recorded.

The results presented are averaged over 10 runs for each algorithm.

5.2 Test Programs

During the experiment, each of the algorithms described was tested using 6 subject programs including 2 from a large industrial automobile company: a simple odd and evens programs, a simplified UK tax program, a program computing student marks on a given course, a calendar program, a rear-end window defroster and a braking system controller.

An amorphous slice was computed for each test program w.r.t. every assigned variable in the source code. The number of slices generated per technique ranged from 4 slices to over 60 slices computed for the braking system controller.

		Search Technique %				
Program	Size (LoC)	GA	SS	HC	RS	LinIAS
OddEven	41	74	79	82	74	87
Tax	77	30	33	31	30	34
Calendar	87	26	51	27	26	53
Defroster	123	72	77	74	71	78
Marking	155	48	55	53	48	61
Braking Cllr.	326	60	70	59	59	72

Figure 3. Results of executing different search techniques to compute amorphous slice and the analytic amorphous slicing system showing average percentage reduction in slice size

We observe two outputs from each simulation of the algorithms: the amorphous slice and the number of lines of code for the slice. In the implementation, we keep the source program static. This means that each new application of an individual is applied to the same original source program. The effects on a program of previous applications of transformation seqeuences are discarded before the next sequence is applied.

6 Discussion

Figures 4–9 shows the results of applying different heuristic search approaches to amorphous slicing and are compared against the analytic-based system. On each box plot, each box represents the distribution of the resulting slice sizes when slicing the subject program w.r.t. all assigned variables in the source code. Each chart shows the different amorphous slicing techniques on the x-axis and the resulting slice size in number of lines of code (LoC) on the y-axis. Each result shows at least one of the search techniques perform quite comparable on the average with the results of the slices computed by LinIAS.

We observe that the search heuristic achieved success in slicing the program under test with respect to the slicing criterion, returning in some cases, slice sizes that matched those returned by LinIAS. However, there are variations in the results obtained from the various search techniques. The Hill–Climb Algorithm performed better with the OddEven program than it did with the Tax test program while the Systematic Search Algorithm performed more consistently across the various tests. This perhaps suggests that a hybrid

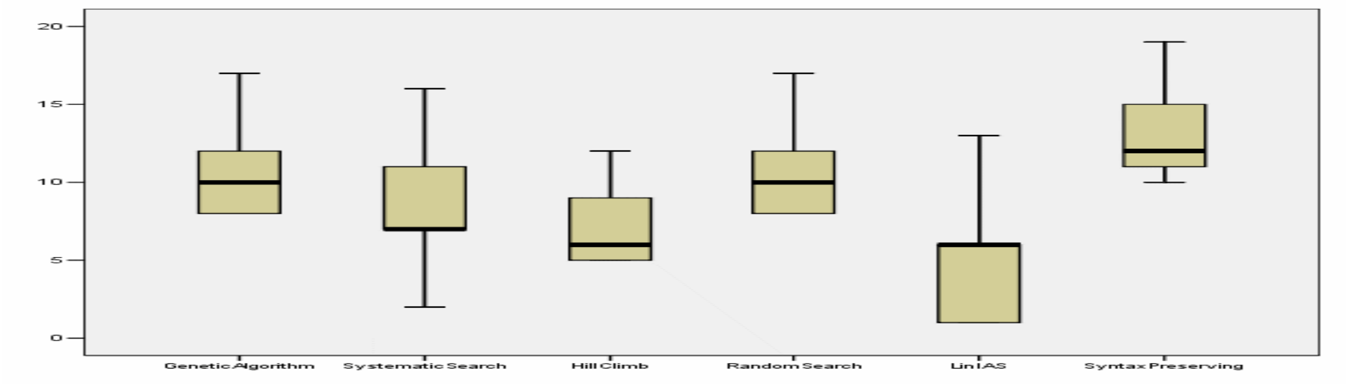

Figure 4. Results of executing different search techniques to compute amorphous slices and the analytic slicing system on the OddsEven subject program

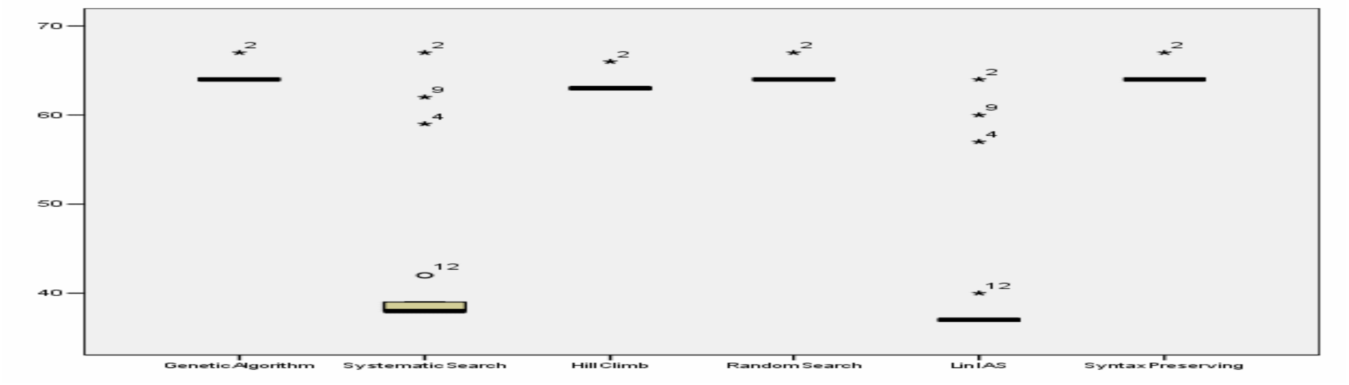

Figure 5. Results of executing different search techniques to compute amorphous slices and the analytic slicing system on the Calendar subject program

algorithm may combine the best of each of these individual approaches to produces even better results.

Figure 8, which illustrates the results from the test on the rear-end window controller, shows the Systematic Search technique considerably reducing the program size to a level almost comparable to LinIAS. Encouragingly, the average size for slices produced by search techniques are better than those returned by Random Search. Figures 4, 5, 7 and 9 all show the Systematic Search and \ or the Hill-Climb Algorithm perform significantly better than Random. In Figure 3, we observe the average reductions in slice sizes and notice in one case an over 80% reduction in the original program size. Statistically we observe positive correlation between the results achieved by the various search approaches and those from LinIAS for each test program with the search techniques performing well for test cases where LinIAS performs well and vice versa.

Furthermore, we also observe that in the simplified UK Tax test case results in Figure 6 and in the OddsEven program in Figure 4, examining specifically four slicing criteria, that the Hill–Climb computed smaller amorphous slices than LinIAS w.r.t these four variables.

Disappointingly however, we notice across the results that whilst the Genetic Algorithm computed valid amorphous slices that were smaller in size than the original program, the size of the slices tended to be comparable to those returned by Random Search. This might either be due to the GA simply being naïve or the problem better suited to local and more directed search heuristics.

Significantly, although the LinIAS system on average returns smaller slices than the search–based approaches, it has a seemingly unfair advantage. It uses as one of its transformations, a traditional syntax-preserving static slicer, which the search methods did not have available to within

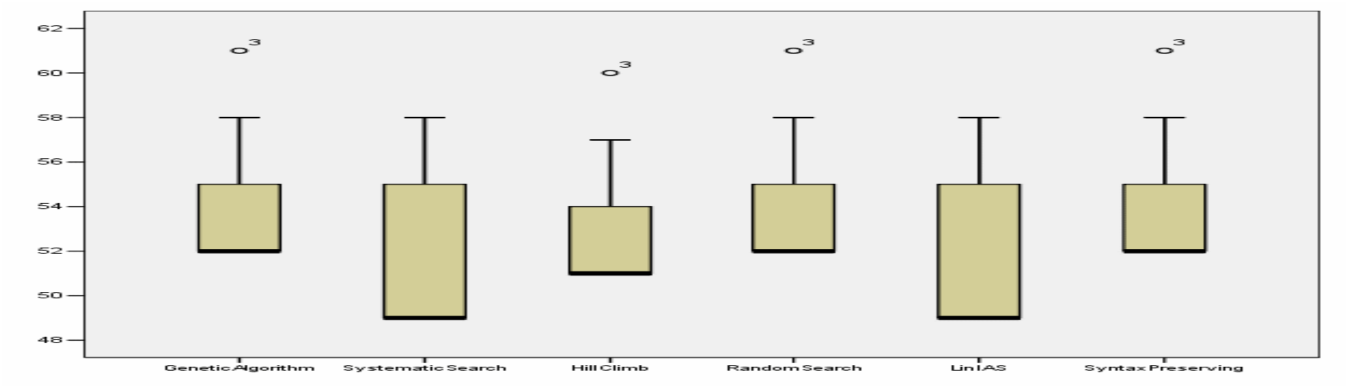

Figure 6. Results of executing different search techniques to compute amorphous slices and the analytic slicing system on the Tax subject program

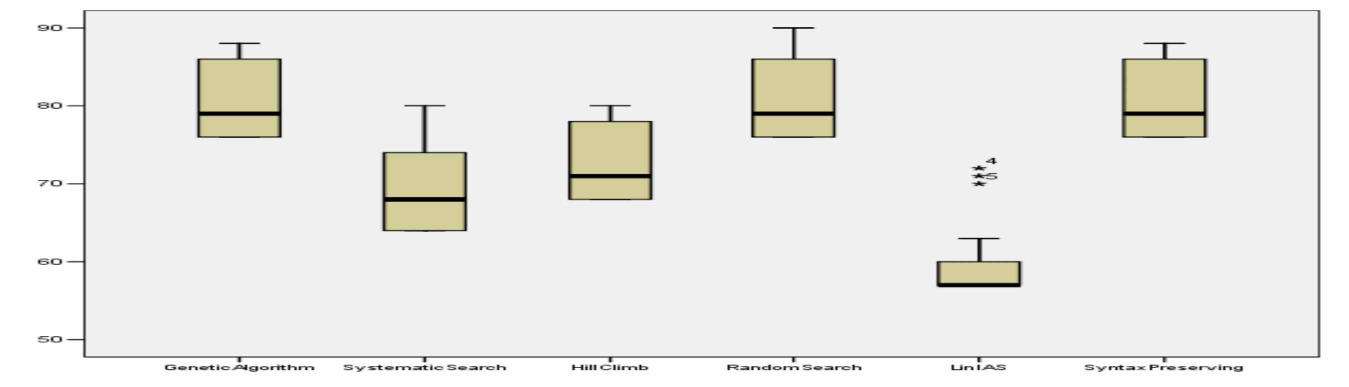

Figure 7. Results of executing different search techniques to compute amorphous slices and the analytic slicing system on the Marks subject program

their pool of transformations. The presence of such a traditional slicing transformation rule in the pool of transformation rules available to the search algorithms may result in dramatically improved results than those already observed. The good of this work is to see whether transformation *can* be formulated as a search problem and to investigate the differences between various search algorithms. However, it is very encouraging that these techniques are able to return good and occassionally competitive results despite this handicap.

6.1 A Surprising Result

It has been widely observed that search techniques are good at producing unexpected answers. This happens because the techniques are not hindered by implicit human assumptions. One example is the discovery of a patented digital filter using a novel evolutionary approach [28]. Another example is the discovery of patented antenna designs [22] which are available commercially. The human formalises their (explicit) assumptions as a fitness function. The machine uses this fitness function to guide the search. Should the search produce unexpected results then this reveals some *implicit* assumptions and/or challenges the human's intuition about the problem.

Unlike human–based search, automated search techniques carry with them no bias. They automatically scour the search space for the solutions which best fit the (stated) human assumptions in the fitness function. This is one of the central strengths of the approach.

In a small way, this insight-yielding advantage of search was observed in our experiments on search–based amorphous slicing. For one of the examples — (tax program) in Figure 6 — the amorphous slice found by the search tech-

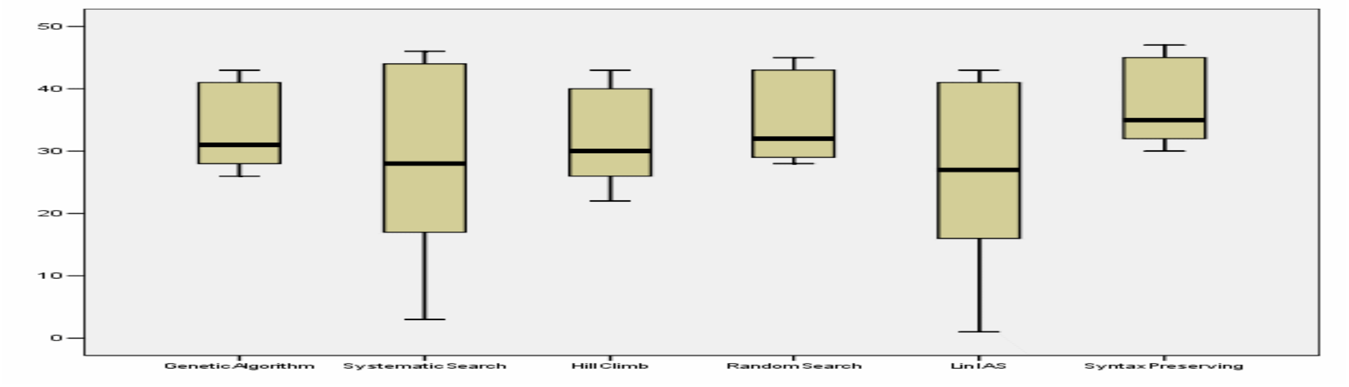

Figure 8. Results of executing different search techniques to compute amorphous slices and the analytic slicing system on the Rear-End Window Controller subject program

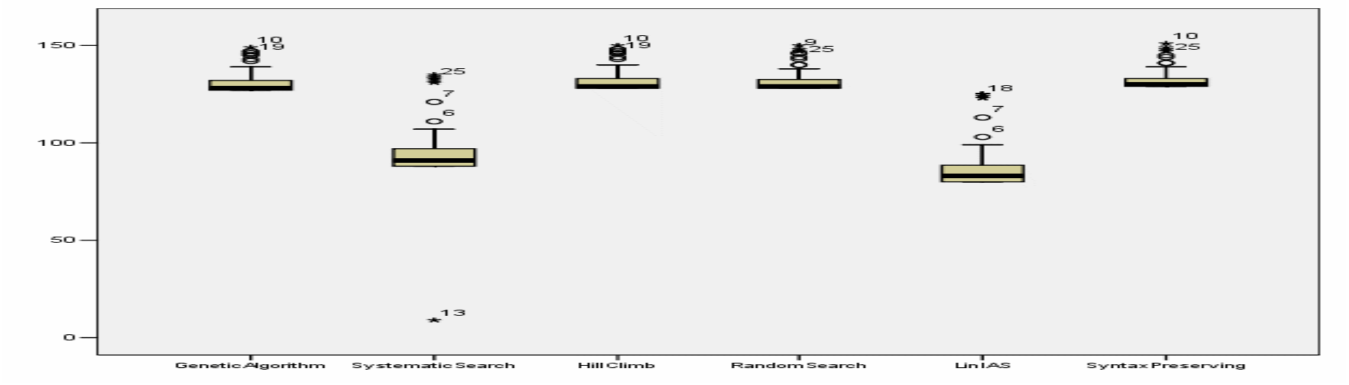

Figure 9. Results of executing different search techniques to compute amorphous slices and the analytic slicing system on the Braking Controller subject program

niques was *smaller* than that produced by the analytic algorithm. This seemed impossible at first because the analytic algorithm is designed to produce excellent results with precisely the type of programs under study. Closer examination revealed that the syntax–preserving slices had a weakness in it. It produced an over–large slice, which fed into the amorphous slicing algorithm used by LinIAS. For the particular program and slicing criterion it missed a chunk of code which could be deleted.

7 Related Work

Slicing was introduced by Mark Weiser in 1979 [33] and has been the subject of extensive study since then. However studies [25, 20, 21, 7] carried out from then through to the early 90s employed statement deletion as the sole simplifying transformation used to create slices.

Several authors have indicated that the syntactic subset requirement of syntax-preserving slicing has been a hindrance to the computation of small slices [9]. Indeed, in his thesis, Weiser immediately recognized and acknowledged ([33], page 6) that it would not always be possible for a slice to be constructed as a purely faithful subset of the original program's syntax.

Many other authors have suggested ways of combining slicing and transformation for a variety of applications including refining the precision of syntax-preserving slicing[29], assisting testing [16], identifying unobservable components in optimising task scheduling [14] and register-allocation optimisation [24].

Amorphous slicing was first introduced by Harman and Danicic [17], and has been developed by Binkley [2] and Harman, Binkley and Danicic [15] and by Ward [32]. Binkley's approach uses the System Dependence Graph [20], while Ward's approach uses a novel syntax–preserving slicing algorithm,which is currently under development into an augmented system for producing semantic slices [32],which are closely related to amorphous slices. Binkley et al. [5, 15] have shown that amorphous slicing aids program comprehension. Hierons, Harman and Danicic [19] have shown how amorphous slices can be used to (partly) ameliorate the equivalent mutant problem for mutation testing.

Search–Based transformation have been researched by several authors: Fatiregun et al. [13] examined using search based techniques to (fully) automate program transformation at source–code level. Cooper et al. [10] examined using biased random sampling search means to find ideal sequences for perfoming compiler optimisations. Nisbet [23] focused on using GAs to find program restructuring transformations for FORTRAN programs to execute on parallel architectures and Ryan [27] worked on using search techniques to automate parallelisation for super–computers.

8 Future Work

The following issues will be addressed in future work. First, research into the scalability of this approach by applying the techniques proposed for very large-scale subject programs.

Secondly, in the present study, the search–based approaches were implemented independently. We propose experimenting with other search–based heuristic and implementing a hybrid algorithm, for example, we could use systematic search to produce a set of transformations, which will then be used to seed a hill–climbing algorithm. The results of 'different versions' of the systematic search/hill–climb may then be used as an initial population for a GA and then finally hill–climb on the results of the GA.

Thirdly, the current approach with the search algorithms is to keep the program under test fixed (static) for each application of an individual transformation sequence. For algorithms that require transformation sequence as unique individuals such as the GA and HC, each application of an individual transforms an old copy of the original program under consideration. We propose to dynamically change the program under consideration each time a tansformation sequence is applied and returns a smaller slice of the original. That way, subsequent sequences act upon improved versions of the program under consideration and should ultimately produce much smaller slices.

9 Conclusion

This paper introduces the idea of using search–based approaches to compute amorphous slices. By reformulating the amorphous slicing problem as a redundancy removal problem, we are able to apply transformations capable of eliminating redundancies and simplifying dependencies to reduce the size of program slices and preserve the computations for a particular variable of interest.

We implement a Genetic Algorithm, Hill–Climb and System Search Algorithm to traverse the search space for transformation sequences and report that all three algorithms produce valid amorphous slices that are smaller in size than the original program under study and present a case where a search–based approach performs better than a dedicated analytic amorphous slicing system.

References

[1] D. W. Binkley. The application of program slicing to regression testing. In M. Harman and K. Gallagher, editors, *Information and Software Technology Special Issue on Program Slicing*, volume 40, pages 583–594. Elsevier Science B. V., 1998.

[2] D. W. Binkley. Computing amorphous program slices using dependence graphs and a data-flow model. In *ACM Symposium on Applied Computing*, pages 519–525, The Menger, San Antonio, Texas, U.S.A., 1999. ACM Press, New York, NY, USA.

[3] D. W. Binkley and K. B. Gallagher. Program slicing. In M. Zelkowitz, editor, *Advances in Computing, Volume 43*, pages 1–50. Academic Press, 1996.

[4] D. W. Binkley and M. Harman. A survey of empirical results on program slicing. *Advances in Computers*, 62:105–178, 2004.

[5] D. W. Binkley, M. Harman, L. R. Raszewski, and C. Smith. An empirical study of amorphous slicing as a program comprehension support tool. In 8^{th} *IEEE International Workshop on Program Comprehension (IWPC 2000)*, pages 161–170, Limerick, Ireland, June 2000. IEEE Computer Society Press, Los Alamitos, California, USA.

[6] T. Bull. *Software maintenance by program transformation in a wide spectrum language*. PhD thesis, University of Durham, UK, School of Engineering and Computer Science, 1994.

[7] G. Canfora, A. Cimitile, and A. De Lucia. Conditioned program slicing. In M. Harman and K. Gallagher, editors, *Information and Software Technology Special Issue on Program Slicing*, volume 40, pages 595–607. Elsevier Science B. V., 1998.

[8] G. Canfora, A. Cimitile, A. De Lucia, and G. A. D. Lucca. Software salvaging based on conditions. In *International Conference on Software Maintenance (ICSM'96)*, pages 424–433, Victoria, Canada, Sept. 1994. IEEE Computer Society Press, Los Alamitos, California, USA.

[9] J. Choi and J. Ferrante. Static slicing in the presence of goto statements. *ACM Transactions on Programming Languages and Systems*, 16(4):1097–1113, July 1994.

[10] K. D. Cooper, P. J. Schielke, and D. Subramanian. Optimising for reduced code space using genetic algorithms. In *Proceedings of the 1999 Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES)*, May 1999.

[11] A. De Lucia. Program slicing: Methods and applications. In 1^{st} *IEEE International Workshop on Source Code Analysis and Manipulation*, pages 142–149, Florence, Italy, 2001. IEEE Computer Society Press, Los Alamitos, California, USA.

[12] A. De Lucia and M. Munro. Program comprehension in a reuse reengineering environment. In M. Munro, editor, 1^{st} *Durham workshop on program comprehension*, Durham University, UK, July 1995.

[13] D. Fatiregun, M. Harman, and R. Hierons. Evolving transformation sequences using genetic algorithms. In 4^{th} *International Workshop on Source Code Analysis and Manipulation (SCAM 04)*, pages 65–74, Chicago, Illinois, USA, Sept. 2004. IEEE Computer Society Press, Los Alamitos, California, USA.

[14] R. Gerber and S. Hong. Slicing real-time programs for enhanced schedulability. *ACM Transactions on Programming Languages and Systems*, 19(3):525–555, May 1997.

[15] M. Harman, D. W. Binkley, and S. Danicic. Amorphous program slicing. *Journal of Systems and Software*, 68(1):45–64, Oct. 2003.

[16] M. Harman and S. Danicic. Using program slicing to simplify testing. *Software Testing, Verification and Reliability*, 5(3):143–162, Sept. 1995.

[17] M. Harman and S. Danicic. Amorphous program slicing. In 5^{th} *IEEE International Workshop on Program Comprenhesion (IWPC'97)*, pages 70–79, Dearborn, Michigan, USA, May 1997. IEEE Computer Society Press, Los Alamitos, California, USA.

[18] M. Harman, L. Hu, M. Munro, X. Zhang, D. W. Binkley, S. Danicic, M. Daoudi, and L. Ouarbya. Syntax-directed amorphous slicing. *Journal of Automated Software Engineering*, 11(1):27–61, Jan. 2004.

[19] R. M. Hierons, M. Harman, and S. Danicic. Using program slicing to assist in the detection of equivalent mutants. *Software Testing, Verification and Reliability*, 9(4):233–262, 1999.

[20] S. Horwitz, T. Reps, and D. W. Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems*, 12(1):26–61, 1990.

[21] B. Korel and J. Laski. Dynamic program slicing. *Information Processing Letters*, 29(3):155–163, Oct. 1988.

[22] D. S. Linden. Innovative antenna design using genetic algorithms. In D. W. Corne and P. J. Bentley, editors, *Creative Evolutionary Systems*, chapter 20. Elsevier, Amsterdam, The Netherland, 2002.

[23] A. Nisbet. GAPS: A compiler framework for genetic algorithm (GA) optimised parallelisation. In *HPCN Europe*, pages 987–989, 1998.

[24] C. Norris and L. L. Pollock. The design and implementation of RAP: A PDG-based register allocator. *Software Practice and Experience*, 28(4):401–424, Apr. 1998.

[25] K. J. Ottenstein and L. M. Ottenstein. The program dependence graph in software development environments. *SIGPLAN Notices*, 19(5):177–184, 1984.

[26] H. A. Partsch. *The Specification and Transformation of Programs: A Formal Approach to Software Development*. Springer, 1990.

[27] C. Ryan. *Automatic re-engineering of software using genetic programming*. Kluwer Academic Publishers, 2000.

[28] T. Schnier, X. Yao, and P. Liu. Digital filter design using multiple pareto fronts. *Soft Computing*, 8(5):332–343, April 2004.

[29] F. Tip. *Generation of Program Analysis Tools*. PhD thesis, Centrum voor Wiskunde en Informatica, Amsterdam, 1995.

[30] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, Sept. 1995.

[31] M. Ward. Assembler to C migration using the FermaT transformation system. In *IEEE International Conference on Software Maintenance (ICSM'99)*, Oxford, UK, Aug. 1999. IEEE Computer Society Press, Los Alamitos, California, USA.

[32] M. Ward. Program slicing via FermaT transformations. In 26^{th} *IEEE Annual Computer Software and Applications Conference (COMPSAC 2002)*, pages 357–362, Oxford, UK, Aug. 2002. IEEE Computer Society Press, Los Alamitos, California, USA.

[33] M. Weiser. *Program slices: Formal, psychological, and practical investigations of an automatic program abstraction method*. PhD thesis, University of Michigan, Ann Arbor, MI, 1979.

[34] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984.