A New Representation and Crossover Operator for Search-Based Optimization of Software Modularization

Mark Harman and Robert Hierons
Brunel University,
Uxbridge, Middlesex, UB8 3PH.
{Mark.Harman,Rob.Hierons}@brunel.ac.uk

Mark Proctor,
CISCO Systems, 250 Longwater Avenue,
Reading, UK.
Mark.proctor@bigfoot.com

Abstract

This paper reports experiments with automated software modularization and remodularization, using search-based algorithms, the fitness functions of which are derived from measures of module granularity, cohesion and coupling. The paper introdeuces a new representation and crossover operator for this problem and reports initial results based on simple component topologies.

1 INTRODUCTION

It is well established in the software engineering community that good modularization of software leads to system which are easier to design, develop, test, maintain and evolve. Given a set of program components, there are many ways in which the module boundaries can be drawn, each of which corresponds to a different 'modularization' of the software. The problem is a graph partitioning problem, which is known to be NP hard and therefore seems suited to a metaheuristic search-based approach.

Macoridis et al. [15] showed that the problem of modularizing software can be reformulated as a search problem. Initially, they used an exhaustive search and hill climber [15], but later experimented with a simple genetic algorithm [4] to search the space of possible modularizations. Mitchell [16] provides a survey of work on modularization together with experience using exhaustive search, hill climbing and genetic algorithms. Mitchell reports that exhaustive search becomes impractical for networks of more than 15 components. He describes the Bunch [14] a modularization tool which uses a hill climbing approach to implement search based modularization. The Bunch tool uses hill climbing, rather than a genetic algorithm, because it was found that the hill climber produced more consistently high quality results [16]. Mitchell indicates that the genetic approach requires more work.

This paper attempts to further explore the application of genetic algorithms to the problem, and makes two modest contributions to the application of genetic algorithms to the modularization problem.

- We introduce a new representation which allows only one representation per modularization.
- We introduce a new crossover operator which attempts to preserve building blocks.

In our work, we were principally concerned with the problem of reverse engineering a system whose modularization has degraded as the system is maintained. For such a system some components may no longer be in suitable modules and re-modularization of the system might be appropriate. The granularity of a modularization is the number of modules it uses. Our problem is therefore to search the space of possible modularizations around the current granularity to see if there exists a better allocation of components to modules.

The rest of the paper is organized as follows. Section 2 describes the approach adopted to formulating modularization as a search problem. Section 3 presents the results of applying the genetic algorithm to example modularization problems. Section 4 presents related work and Section 6 concludes.

2 REPRESENTATION, FITNESS AND OPERATORS

The starting point for the application of search-based techniques to software engineering is the definition of a suitable representation, fitness function and operators

[7]. This section introduces a new representation for the allocation of components to modules and a new crossover operator. We also describe our approach to the definition of the fitness function, as this differs from that used in previous work [15, 4, 16].

2.1 REPRESENTATION

The first problem which presents itself when attempting to formulate a software engineering problem as a search-based problem, is that of representation. In the case of modularization there is a need to identify each possible way of modularizing a system in a *unique* way so that there is only one representation per modularization. Non-unique representations of modularizations artificially increase the search space size, inhibiting search-based approaches to the problem.

The approach we adopted was to normalize the representation in the following way: Modules are numbered, and elements allocated to module numbers using a simple look-up table. Component number one is always allocated to module number one. All components in the same module as component number one are allocated to module number one. Next, the lowest numbered component, n, not in module one, is allocated to module number two. All components in the same module as component number n are allocated to module number two. This process is repeated, choosing each lowest numbered unallocated component as the defining element for the module.

This representation must be renormalized when components move as the result of mutation and crossover, but it has the significant advantage that a particular allocation of components to modules has but one single representation.

2.2 FITNESS FUNCTION

Following Constantine and Yourdon [2], approaches to modularization typically attempt to maximize cohesion and minimize coupling in line with software engineering principles which indicate that this leads to good quality results. Constantine and Yourdon defined seven levels of coupling and seven levels of cohesion. These seven levels of cohesion provide a qualitative measure of a systems overall cohesion and coupling. Unfortunately the levels are of little use as an input to a fitness function as they are too subjectively defined. Lakhotia [10] formalised the seven levels within a dependency analysis framework. This work allows the levels to form the input to a fitness function. However, Lakhotia's measure would yield only a seven point scale, resulting in a fitness landscape which would be coarse and would, therefore, be inappropriate for a search-based solution.

In this paper cohesion and coupling will be measured simply in terms of dependencies between the components of a module. The module's components will be assumed to be a set of procedures, functions and variables. Dependence arises between procedure (or function) p and procedure (or function) p' iff p calls p'. Similarly, a dependence (or association) arises between a procedure (or function) p and variable v iff p reads from or writes to v.

The problem of finding good modularization is therefore a graph theoretic problem of finding subgraphs with the maximum connectivity (cohesion), with the minimal association between subgraphs (coupling), and for a desired number of identified subgraphs (target granularity).

Cohesion for a network is measured as the average number of associations per module with respect to the maximum possible number of associations. More formally, let $\mathcal{A}$ be a function from modules to the number of associations within the module. Let N be a function from a module to the number of components in the module and let K be the number of modules in the network. The cohesion $C(m)$, of a module m is defined as follows:

$$\mathrm{C}(m) = \begin{cases} 1 & \text{if N(m) = 1} \\ \frac{\mathcal{A}(m)}{\mathcal{N}(m).(\mathcal{N}(m)-1)} & \text{otherwise} \end{cases}$$

The cohesion of the system, S containing K modules is defined as follows:

$$\mathrm{Cohesion(S)} = \frac{\sum_{m \in S} C(m)}{K}$$

This value is renormalized to a percentage (by multiplying by 100), so that 100% indicates that all components are related to all others within their own module.

The coupling *unfitness*[1], of the system is expressed as the total number of inter-module associations divided by the total number possible for the network. Coupling fitness is simply the inverse of coupling unfitness. Once again this is renormalized to a percentage, where 100% indicates that there is no coupling between modules.

In order to capture the additional requirement that the modularization produced has a granularity not too dissimilar to the current granularity, a polynomial punishment factor was introduced into the fitness function,

[1]recall that coupling is considered to be bad, so it is to be minimized, hence 'unfitness'.

to reward solutions polynomially as they approach the target value for granularity of the modularization.

The aim is to allow some deviation from the target granularity, where this can allow dramatic improvement in cohesion and coupling values for the overall system, but to encourage the search to consider solutions on and around the target granularity.

The granularity component of the fitness function is calculated in terms of the actual granularity of the modularization AG and the target granularity TG. Once again this is normalized to a percentage, so that 100% represents the situation where the actual and target values of granularity are identical. The value of the actual granularity is allowed to range from 0 to twice the target granularity, which each of these extreme values scoring zero fitness for the granularity component of the fitness function.

The three fitness components: cohesion, coupling and granularity are each given equally weight in computing the overall fitness of the system.

2.3 CROSSOVER

To attempt to promote the formation, retention and propagation of good building blocks [5, 23] within the genetic algorithm, a crossover operator was defined which attempts to preserve partial module allocations from parents to children.

Rather than selecting an arbitrary point of crossover within the two parents, an arbitrary parent is selected and one of its arbitrarily chosen modules is copied to the child. This results in a partial allocation of components to modules in the child. The components allocated are removed from both parents. This removal can be thought of as a form of 'pre-emptive repair' as it prevents duplication of components in the child when further modules are copied from one or other parent to the child.

The process of selecting a module from a parent and copying to the child is repeated and the components copied are removed from both parents until the child contains a complete allocation (that is, when both parents have no modules left to copy).

This approach ensures that at least one (randomly chosen) module from the parents is preserved (in entirety) in the child, and that parts of other modules will also be preserved.

A standard genetic algorithm was implemented with single point crossover, to allow comparison with the novel crossover operator.

Mutation was set to an unusually low value (after crossover, an individual chromosome had only a 5% chance of mutation). The population size was also relatively low at 30 individuals, in keeping with prior work [16]. This allows the possible effects of the crossover operator to dominate our results, facilitating a comparison of the novel and standard crossover techniques.

3 RESULTS

Figures 1 and 2, show the results of applying the two genetic algorithms and hill climbing against a random search baseline. The labellings are 'Random' for the random search, 'HC' for the Random Mutation Hill Climbing search, 'GA' for the simple genetic algorithm with standard single point crossover and 'GA+' for the genetic algorithm with the novel crossover operator. Results are averaged over five separate runs for each search algorithm to account for random effects.

The left hand section of the figure shows the parameters for the problem and illustrates the components and their associations, depicted in a manner which suggests the 'ideal' modularization (namely, that which maximizes fitness).

The right-hand section of the figure shows the corresponding results for the three search-based algorithms, together with random search (as a base-line performance). The graphs plot average fitness (over five runs) against generation number. Recall that fitness is denoted by a percentage, where 100% is the maximum fitness obtainable for a 'perfect' modularization. A perfect modularization has reached exact target granularity, no associations between modules and has every component within a module related to every other component. Such a perfect fitness may be prohibited by the structure of the associations between components and so 100% is not reached by any of the search algorithms in some cases.

Figure 1 shows the behaviour of the search algorithms when the target granularity is appropriate. That is, the target granularity is set to the number of modules in the 'ideal' modularization. Figure 2 shows the behaviour when the target granularity is misleading.

3.1 APPROPRIATE TARGET GRANULARITY

The results for these simple networks confirm that search-based algorithms outperform random search[2]

[2]In the most simple problem (at the top of the figure), the genetic algorithm with the novel crossover technique is

and that the gap between the search-based algorithms and random search increases with the size of the problem.

As expected, GA+, the genetic algorithm with the novel crossover operator outperforms the simple genetic algorithm, GA, which uses a single point crossover operator. However, the novel crossover quickly becomes trapped by a local optimum.

For these simple networks, hill climbing outperforms all other techniques. This is not surprising since the genetic algorithms are focused upon the use of crossover and allow very little mutation.

3.2 MISLEADING TARGET GRANULARITY

The results for a selection of simple problems, where the value of target granularity is set to a misleading value were also collected and are depicted in Figure 2. These results still show that the hill climber and simple genetic algorithm out-perform random search and that hill climbing is far superior, but they indicate a worse performance for the GA+ search algorithm which employs the novel crossover technique. It is worth noting that that GA+ still performs well where the modules are very clearly defined by the associations (the third case).

These results suggest that GA+ is more sensitive to inappropriate choices of target granularity than any of the other approaches. This sensitively can be exploited, because it will suggest more radical remodularization for very badly degraded, heavily maintain software, where a complete repartitioning of the system will produce better results than merely moving a few components between modules, or perhaps adding or removing a module or two.

4 RELATED AND FUTURE WORK

The work reported here is most closely related to work on the Bunch tool, by Macoridis et al. [15, 4, 16, 14], who introduced the search-based approach to software modularization. Macoridis et al. use a standard genetic algorithm with single point crossover and a representation which allocates a module number to each component. This representation has the drawback that it allows many representations of a single modularization, for example the strings (1,1,2,1,2,3,3), (3,3,1,3,1,2,2) and (2,2,3,2,3,2,2) all represent a modularization consisting of three modules, which places components 1,2 and 4 in one module, components 3 and 5 in another module and which places components 6 and 7 in a third module.

worse than random, but in all other cases all search algorithms are better than random.

The principal difference between our work and that of Macoridis et al. lies in the novel crossover technique and the instruction of a normalized representation for the modularization problem.

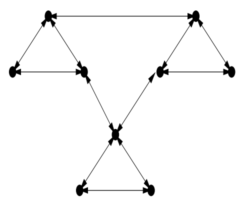
A related problem of hierarchical decomposition of software is considered by Lutz [13]. Lutz is concerned with the problem of decomposition of software into hierarchies at different levels of abstraction, whereas the present work is concerned with only a single level of abstraction (the implementation level). Lutz therefore considers designs rather than code. However, there is no reason, in principle, why the approach adopted by Lutz could not be also applied to the modularization problem considered in the present paper.

The approach adopted by Lutz differs strongly from the approach adopted in the present paper with regard to the choice of fitness function. The fitness function used by Lutz is based upon an information-theoretic formulation inspired by Shannon [19]. The function awards high fitness scores to hierarchies which can be expressed most simply (in information theoretic terms), with the aim of rewarding the more 'understandable' designs. Such a fitness function is possibly more semantic than the comparatively structural approaches adopted in the present paper and in the work of Macoridis et al. More work is required to compare the results produced by these two approaches.

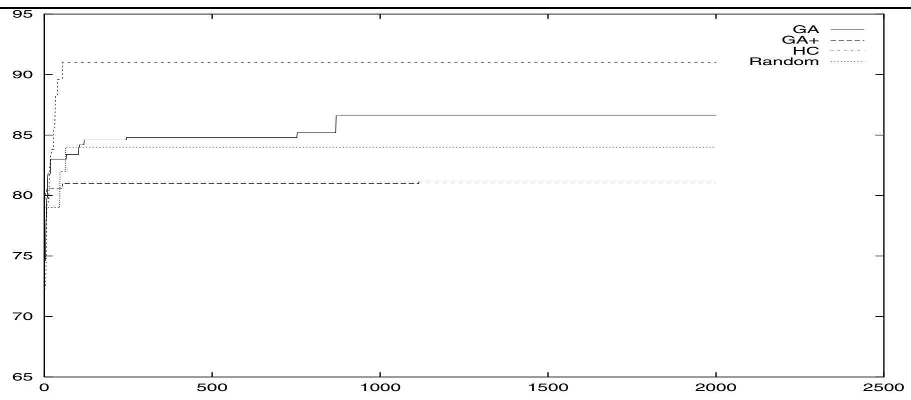
Other work on software re-modularization has adopted analytical solutions based upon formal concept analysis and clustering metrics [9, 21, 11] and sets of heuristic rules [18]. More work is required to assess the comparative performance of these non search-based approaches with the search-based strategy introduced here.

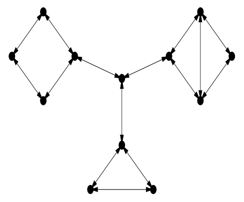
Some metrics for coupling and cohesion [20, 17, 1, 8] have attempted to give a more 'continuous' real-valued quantitative metric based on a variety of criteria, derived from program slicing [3, 22, 6]. Using these metrics, it is possible to allocate weights to associations between components. This would allow the search to be more attuned to the relative impact of a particular association between modules. This would be particularly useful in systems where there are many associations, and so clustering based merely upon the presence or absence of an association becomes relatively arbitrary.

In order to exploit association weights, a search based clustering algorithm such as the GGA [12] is required.

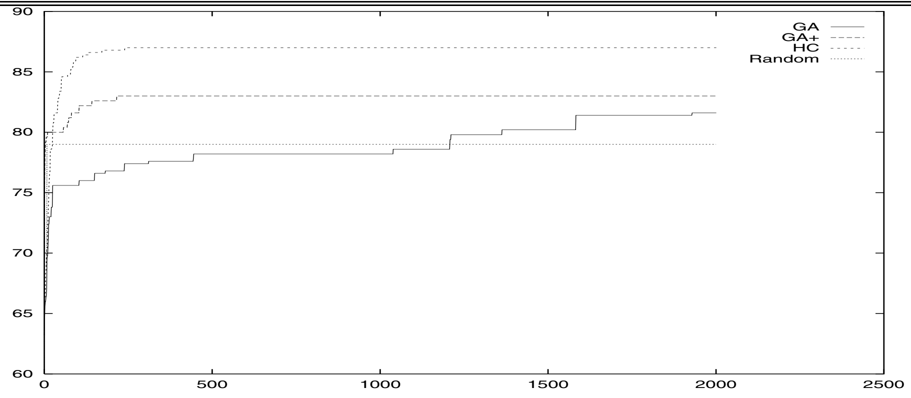


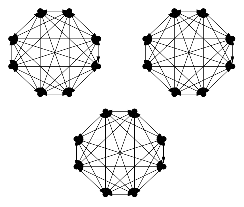
Components = 9
Target Granularity = 3
Population = 30
P_{mut} = 5%



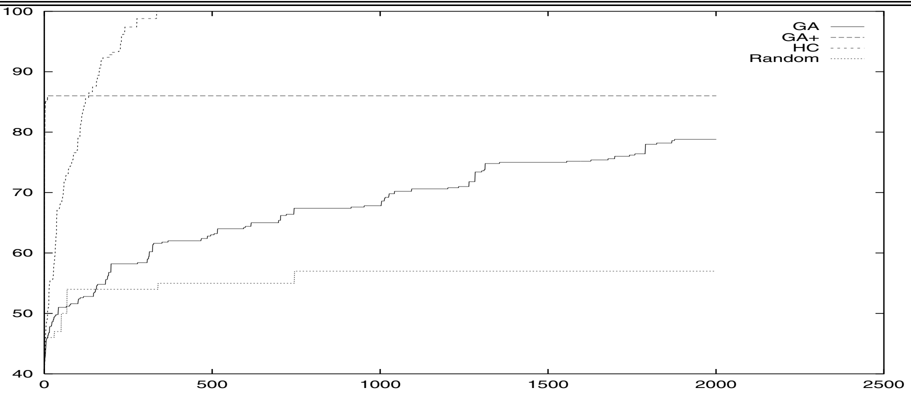


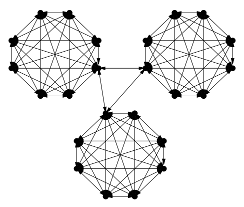
Components = 12
Target Granularity = 4
Population = 30
P_{mut} = 5%





Components = 24
Target Granularity = 3
Population = 30
P_{mut} = 5%





Components = 24
Target Granularity = 3
Population = 30
P_{mut} = 5%

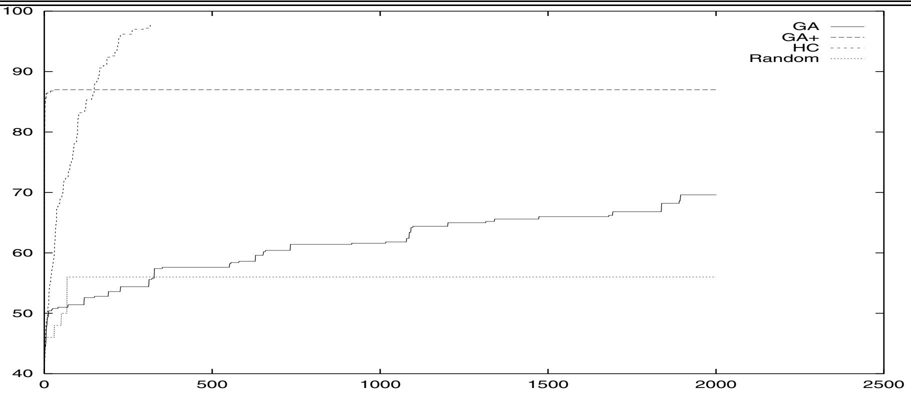
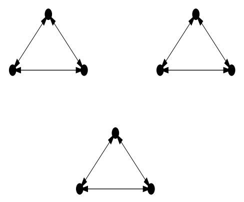
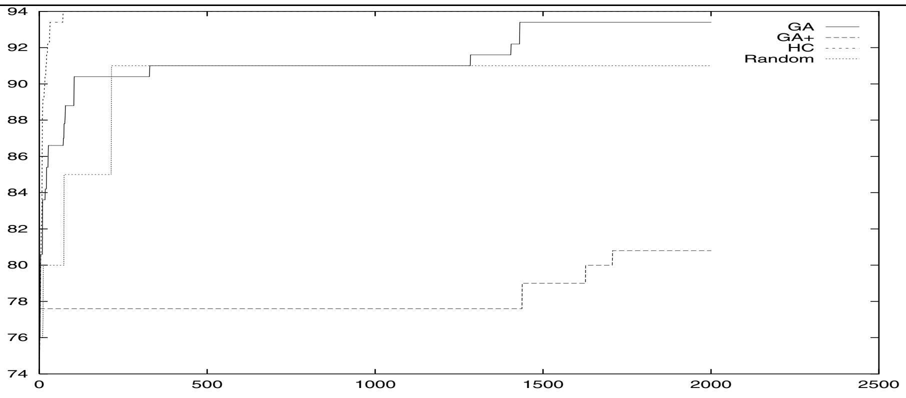
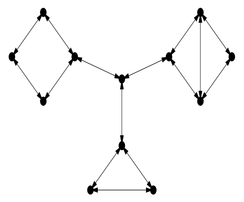


Figure 1: Guiding Value for Target Granularity

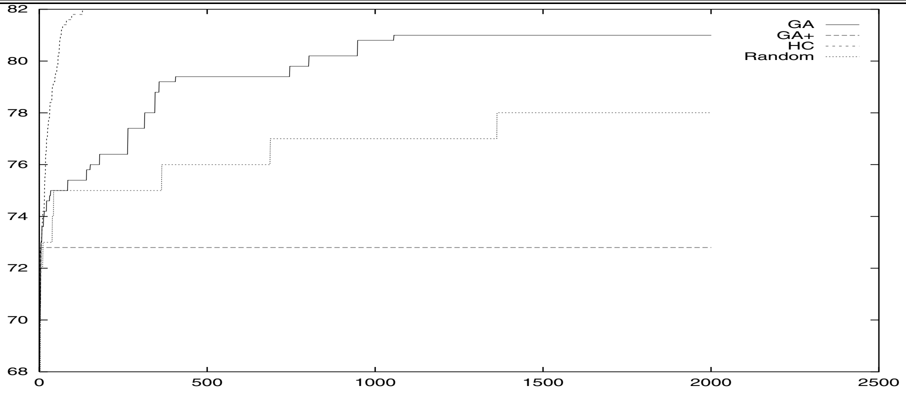


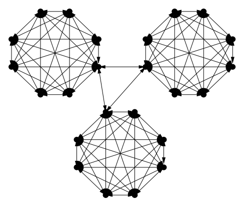
# Components = 9
Target Granularity = 5
Population = 30
P_{mut} = 5%





# Components = 12
Target Granularity = 8
Population = 30
P_{mut} = 5%





# Components = 24
Target Granularity = 8
Population = 30
P_{mut} = 5%

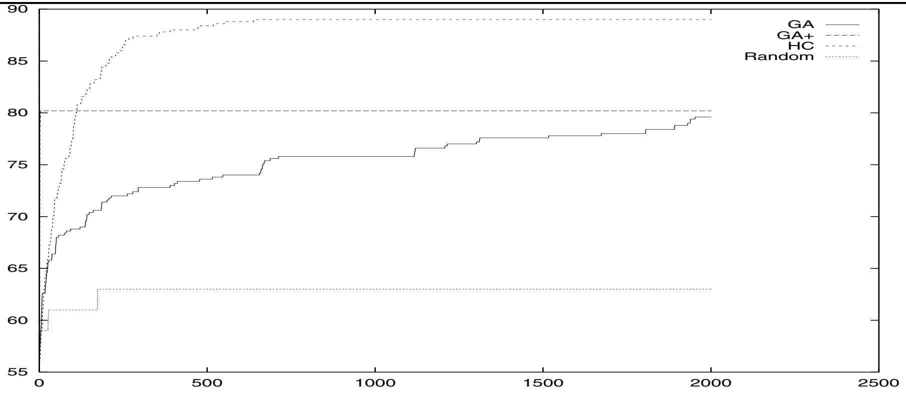


Figure 2: Misleading Value for Target Granularity

The GGA clusters n-dimensional nodes according to the internode distance in the n dimensional space. The modularization problem is a special case of this, where the nodes are components and the distance is the weight of association. In the present paper this 'weight' has simply been 0 (indicating no association) or 1 (indicating the presence of an association). Using the GGA and slice-based measurement of cohesion and coupling, a more 'impact sensitive' approach can be pursued. The authors intend to explore this possibility in future work.

5 ACKNOWLEDGEMENTS

The authors benefitted greatly from discussion of earlier versions of this work with members of the EPSRC-funded SEMINAL network. In particular, Vic-Rayward Smith (at the University of East Anglia, UK) pointed out the importance of a normalized representation. The authors would also like to thank Stephen Swift, Xiaohui Liu and Allan Tucker (at Brunel) for helpful comments on this work and for suggesting and offering the use of the GGA in future work and Spiros Mancoridis and Brian Mitchell (at Drexel) for detailed discussion about their BUNCH system.

6 CONCLUSION

This paper makes a modest contribution to work on the software modularization problem. Previous work in this area has either not used search-based techniques at all, or has largely been concerned with hill-climbing and exhaustive searches.

We introduce a normalized representation for a software modularization, which will reduce the size of the search space and may improve results for genetic algorithms which are known to perform poorly where there is a many-to-one mapping from genotype to phenotype. We also suggest a new crossover operator which is designed to promote the formation and retention of building blocks. Initial work suggests that this crossover technique may be better suited to genetic approaches than standard crossover.

References

[1] Bieman, J. M., and Ott, L. M. Measuring functional cohesion. *IEEE Transactions on Software Engineering 20*, 8 (Aug. 1994), 644–657.

[2] Constantine, L. L., and Yourdon, E. *Structured Design.* Prentice Hall, 1979.

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

[4] Doval, D., Mancoridis, S., and Mitchell, B. S. Automatic clustering of software systems using a genetic algorithm. In *International Conference on Software Tools and Engineering Practice (STEP'99)* (Pittsburgh, PA, 30 August - 2 September 1999).

[5] Goldberg, D. E., and Sastry, K. A practical schema theorem for genetic algorithm design and tuning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)* (San Francisco, California, USA, 7-11 July 2001), L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds., Morgan Kaufmann, pp. 328–335.

[6] Harman, M., and Hierons, R. M. An overview of program slicing. *Software Focus 2*, 3 (2001), 85–92.

[7] Harman, M., and Jones, B. F. Search based software engineering. *Information and Software Technology 43*, 14 (Dec. 2001), 833–839.

[8] Harman, M., Okunlawon, M., Sivagurunathan, B., and Danicic, S. Slice-based measurement of coupling. In 19th *ICSE, Workshop on Process Modelling and Empirical Studies of Software Evolution* (Boston, Massachusetts, USA, May 1997), R. Harrison, Ed.

[9] Hutchens, D., and Basili, V. System structure analysis: clustering with data bindings. *IEEE Transactions on Software Engineering SE-11*, 8 (1985), 749–757. The use of cluster analysis as a tool for system modularization is examined. It appears that the clustering of data bindings provides a meaningful view of system modularization.

[10] Lakhotia, A. Rule–based approach to computing module cohesion. In *Proceedings of the* 15th *Conference on Software Engineering (ICSE-15)* (1993), pp. 34–44.

[11] Lindig, C., and Snelting, G. Assessing modular structure of legacy code based on mathematical concept analysis. In *Proceedings of the 1997 International Conference on Software Engineering* (1997), ACM Press, pp. 349–359.

[12] LIU, X., SWIFT, S., AND TUCKER, A. Using evolutionary algorithms to tackle large scale grouping problems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)* (San Francisco, California, USA, 7-11 July 2001), L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds., Morgan Kaufmann, pp. 454–460.

[13] LUTZ, R. Evolving good hierarchical decompositions of complex systems. *Journal of Systems Architecture 47* (2001), 613–634.

[14] MANCORIDIS, S., MITCHELL, B. S., CHEN, Y., AND GANSNER, E. R. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proceedings; IEEE International Conference on Software Maintenance* (1999), IEEE Computer Society Press, pp. 50–59.

[15] MANCORIDIS, S., MITCHELL, B. S., RORRES, C., CHEN, Y., AND GANSNER, E. R. Using automatic clustering to produce high-level system organizations of source code. In *International Workshop on Program Comprehension (IWPC'98)* (Ischia, Italy, 1998), IEEE Computer Society Press, Los Alamitos, California, USA, pp. 45–53.

[16] MITCHELL, B. S. *A Heuristic Search Approach to Solving the Software Clustering Problem.* PhD Thesis, Drexel University, Philadelphia, PA, Jan. 2002.

[17] OTT, L. M., AND THUSS, J. J. The relationship between slices and module cohesion. In *Proceedings of the* 11^{th} *ACM conference on Software Engineering* (May 1989), pp. 198–204.

[18] SCHWANKE, R. W. An intelligent tool for re-engineering software modularity. In *Proceedings of the 13th International Conference on Software Engineering* (May 1991), pp. 83–92.

[19] SHANNON, C. E. A mathematical theory of communication. *Bell System Technical Journal 27* (July and October 1948), 379–423 and 623–656.

[20] THUSS, J. J. An investigation into slice–based cohesion metrics. Master's thesis, Michigan Technological University, 1988.

[21] VAN DEURSEN, A., AND KUIPERS, T. Identifying objects using cluster and concept analysis. Tech. Rep. SEN-R9814, Centrum voor Wiskunde en Informatica (CWI), Sept. 1998.

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

[23] WU, A. S., AND LINDSAY, R. K. A comparison of the fixed and floating building block representation in the genetic algorithm. *Evolutionary Computation 4*, 2 (1997), 169–193.