A Heuristic Approach to Solving the Software Clustering Problem

Brian S. Mitchell
Department of Computer Science
Drexel University, Philadelphia, PA, USA
bmitchell@drexel.edu

Abstract

This paper provides an overview of the author's Ph.D. thesis [8]. The primary contribution of this research involved developing techniques to extract architectural information about a system directly from its source code. To accomplish this objective a series of software clustering algorithms were developed. These algorithms use metaheuristic search techniques to partition a directed graph generated from the entities and relations in the source code into subsystems. Determining the optimal solution to this problem was shown to be NP-hard, thus significant emphasis was placed on finding solutions that were regarded as "good enough" quickly. Several evaluation techniques were developed to gauge solution quality, and all of the software clustering tools created to support this work were made available for download over the Internet.

1. Introduction

Software supports many business, government, and social institutions. As the processes of these institutions change, so must the software that supports them. Changing software systems that support complex processes can be quite difficult, as these systems can be large (*e.g.*, millions of lines of code) and dynamic.

There is a need to develop sound methods and tools to help software engineers understand large and complex systems so that they can modify the functionality or repair the known faults of these systems. Understanding how the software is structured – at various levels of granularity – is one of several kinds of understanding that is important to a software engineer. As the software structure can itself be very complex, the appropriate abstractions of a system's structure must be determined. Techniques and tools can then be designed and implemented to support the creation of these abstractions.

Automatic design extraction methods have been proposed to create abstract views of a system's structure. Such views help software engineers cope with the complexity of software development and maintenance. Design extraction starts by parsing the source code to determine the components and relations of the software. The parsed code is then analyzed to produce a variety of views of the software structure, at varying levels of abstraction.

Detailed views of software structure are appropriate when the software engineer has isolated the subsystems that are relevant to an analysis task. However, abstract (architectural) views are more appropriate when the software engineer is trying to understand the global structure of the software. Software clustering is used to produce such abstract views. These views feature module-level components and relations contained within subsystems. The source code components and relations can be determined using source code analysis tools. The subsystems, however, are not found in the source code. Rather, they are inferred from the source code components and relations either automatically, using a clustering tool, or manually, when tools are not available.

The problem of automatically creating abstract views of software structure is computationally expensive,[1] so a hope for finding a general solution to the software clustering problem is unlikely. To deal with this complexity the author's research focused on using using metaheuristic search techniques [6, 1, 5] to find "good enough" solutions quickly.

2. Contributions of this Research

Many of the software clustering algorithms presented in the literature use different criteria and techniques to decompose the structure of software systems into clusters (subsystems). Given a particular system, the results produced by the clustering algorithms differ

[1] For example, the software clustering approach described in the dissertation uses graph partitioning, which is a NP-hard [3] problem.

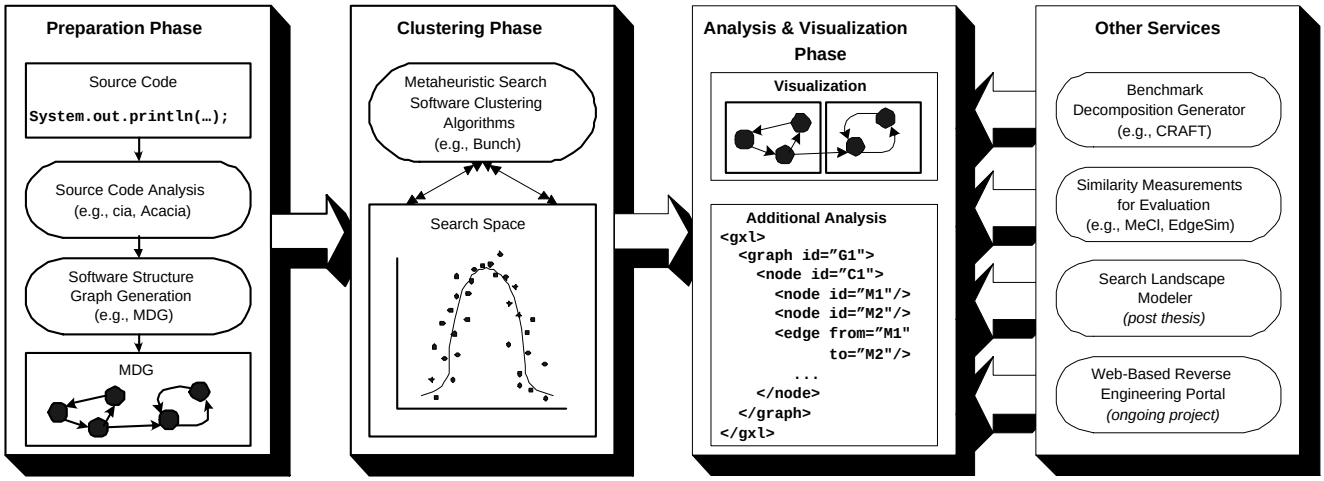

Figure 1. The Bunch Reverse Engineering Environment

because of the diversity in the approaches employed by the algorithms.

The emphasis of the author's dissertation was on creating search-based algorithms and tools to cluster large software systems well and in an efficient manner. A primary deliverable of this research was the creation of an automatic clustering tool called Bunch. Bunch offers several distinguishing capabilities as compared to other clustering tools that have been developed by the reverse engineering research community. Specifically:

- Most software clustering tools can be categorized as either fully-automatic, semi-automatic, or manual. Fully-automatic tools decompose a system's structure without any manual intervention. Semi-automatic tools require manual intervention to guide the clustering process. Manual software clustering techniques guide the user through the clustering process.

  Bunch includes a fully-automated clustering engine, and implements a suite of metaheuristic clustering algorithms. Additionally, a user may supply any known information about a system's structure to Bunch. This information is respected by the automatic clustering engine. In other words, Bunch's clustering approach provides fully-automatic, semi-automatic, and manual clustering features.

- Most software clustering algorithms are deterministic, and as such, always produce the same result for a given input. Since Bunch's clustering engine uses randomized heuristic search techniques, repeated runs on the same system often do not produce the same result, but instead produce a family of related results. The thesis describes several beneficial aspects of this behavior.

- Most software clustering algorithms are computationally intensive. This limits their ability to cluster large systems within a reasonable amount of time. A primary requirement for this research was to support clustering real world systems quickly.[2]

- Most software clustering algorithms are evaluated by soliciting the opinion of a system expert to comment on the quality of the software clustering results. Since an alternative to subjective evaluation was needed, a significant effort was expended on developing better evaluation techniques. Specifically, a tool called CRAFT [10] was created that uses Bunch's software clustering algorithms to generate a reference decomposition, and several similarity measurements [9] were developed to compare and evaluate software clustering results.

3. Bunch's Reverse Engineering Services

Figure 1 illustrates the software clustering services provided by Bunch's reverse engineering environment. These services can be classified into clustering services, evaluation services, and analysis services.

3.1. Software Clustering Services

The goal of the software clustering process is to partition a graph of the source-level entities and relations into a set of clusters such that the clusters represent subsystems. In the preparation phase, source code analysis tools are used to parse the code and build a

[2]The current release of Bunch is able to cluster a system the size of the Linux kernel, with approximately 1,000 modules and 10,000 relations, in a few minutes.

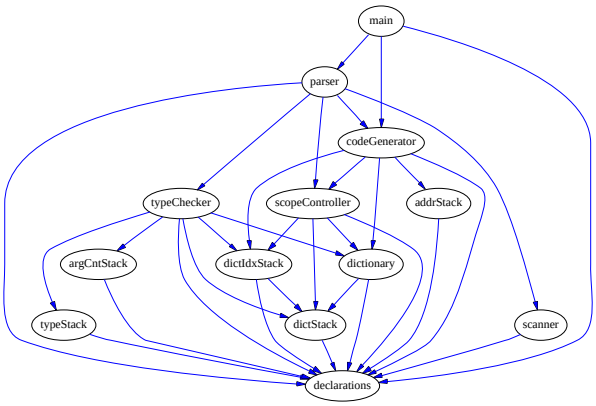

Figure 2. The MDG for a Small Compiler

repository of information about the entities and relations in the system. A series of scripts are then executed to query the repository and create the *Module Dependency Graph* (*MDG*). The *MDG* is a graph where the source code components are modeled as nodes, and the source code dependencies are modeled as edges.

Once the *MDG* is created, Bunch generates a random partition of the *MDG* and evaluates the "quality" of this partition using a fitness function that is called *Modularization Quality* (*MQ*). Several *MQ* functions are described in the full thesis, all having the property of rewarding cohesive clusters and penalizing excessive inter-cluster coupling.

Given that the fitness of an individual partition can be measured with *MQ*, metaheuristic search algorithms are used in the clustering phase of Figure 1 in an attempt to improve the *MQ* of the randomly generated partition. Bunch implements several hill-climbing algorithms [6, 8] and a genetic algorithm (GA) [1, 8].

Once Bunch's search algorithms converge, a result can be viewed as GXL [4] (an XML schema for graphs) or using the dotty [2] graph visualization tool.

To illustrate the clustering process an example *MDG* for a small compiler developed at the University of Toronto is illustrated in Figure 2. A sample partition of this *MDG*, as created by Bunch, is shown in Figure 3. Notice how Bunch automatically created four subsystems to represent the abstract structure of a compiler. Specifically, there are subsystems for code generation, scope management, type checking, and parsing.

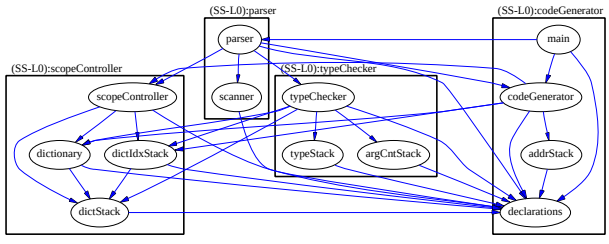

Figure 3. The Partitioned MDG for a Small Compiler

3.2. Evaluation Services

One requirement for evaluating software clustering results is to be able to compare them to each other objectively. To address this requirement our research defined two new similarity measurements for evaluation. These measurements enable the results of clustering algorithms to be compared to each other, and preferably to be compared to an agreed upon "benchmark" standard. Note that the "benchmark" standard needn't be the optimal solution in a theoretical sense. Rather, it is a solution that is perceived by several people to be "good enough". These similarity measurements, called EdgeSim and MeCl [9], were integrated into the Bunch user interface and programming interface to support automated similarity analysis.

Comparing software clustering results to a benchmark standard is useful and often, in the process, exposes interesting and surprising design flaws in the software being studied. However, this kind of evaluation is not always possible, because benchmarks are often not documented, and the developers are not always accessible for consultation. To address this problem another tool called CRAFT [10] was developed to show how the results produced by Bunch and other clustering tools can be evaluated in the absence of a reference design document and without the participation of the original designers of the system. CRAFT essentially creates a "reference decomposition" by exploiting similarities in the results produced by several clustering algorithms. The underlying assumption is that, if several clustering algorithms agree on certain clusters, these clusters become part of the generated reference decomposition.

3.3. Other Analysis Services

Since Bunch exposes all of its functionality via an application programming interface (API), our research group was able to integrate clustering services into some of its other tools. One of our services is called REportal [7], which is an online reverse engineering portal. REportal essentially integrates a set of disparate reverse engineering tools into a set of reverse engineering services. Some of these services include: source code analysis, source code browsing, reachability analysis, impact analysis, visualization and clustering. The REportal website (`http://reportal.cs.drexel.edu`) continues to evolve.

4. Future Research Directions

The author's Ph.D. thesis defines a metaheuristic search-based solution to the software clustering problem, and defined several techniques for evaluating software clustering results. The dissertation also outlines some future research opportunities:

- *Closing the Gap between Software Architecture Definition and Programming Language Design:* Since software clustering involves extracting architectural-level information from source code, future programming language designers may want to consider integrating software architecture specification directly into source code. This feature would simplify the process of extracting architecture facts from source code dramatically.

- *Improved Visualization Services:* Visualizing the software clustering results of large systems is cumbersome. Since visualization is one of the key tenants of programming understanding, improved visualization services would have a high impact.

- *Dynamic Analysis:* The author's Ph.D. thesis addressed clustering the static structure of a software system. Another important representation of a system's structure can be produced by modeling its dynamic behavior during execution. Clustered views, based on dynamic information, will provide further assistance to practitioners who face difficult software maintenance tasks.

- *Other Research Opportunities:* Exploring alternative representations of a system undergoing clustering, formally comparing Bunch's clustering results to other clustering approaches, and improving Bunch's GA[3] are future research opportunities.

5. Conclusions

This thesis contributes to the field of reverse engineering by showing how metaheuristic search-based techniques can be used to solve the software clustering problem. This work placed a strong emphasis on evaluating clustering results, and being able to cluster the structure of large real world software systems efficiently. Commercial quality tools that implement the software clustering algorithms were developed, along with additional automated services to support evaluation.

[3]The author's research determined that Bunch's hill-climbing algorithm consistently produced better results than the GA [8].

Software clustering is used to simplify software maintenance and improve program understanding. Thus, it is hoped that the work embodied in the author's Ph.D. thesis will help to promote the use of software clustering technologies outside of the research environment by fostering interest from the developers of commercial software engineering tools.

6. Acknowledgements

The author would like to thank his advisor, Dr. Spiros Mancoridis, for all of his hard work and superb mentoring. D. Doval, M. Traverso, Y-F. Chen and E. Gansner deserve special recognition for all of the assistance that that they provided throughout the Bunch project. The author is also indebted to the NSF for funding much of the work in his Ph.D. thesis.

References

[1] D. Doval, S. Mancoridis, and B. S. Mitchell. Automatic Clustering of Software Systems Using a Genetic Algorithm. In *Proceedings of Software Technology and Engineering Practice*, Aug. 1999.

[2] E. R. Gansner, E. Koutsofios, S. C. North, and K. Vo. A Technique for Drawing Directed Graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, Mar. 1993.

[3] M. Garey and D. Johnson. *Computers and Intractability.* W.H. Freeman, 1979.

[4] GXL: Graph eXchange Language: Online Guide. http://www.gupro.de/GXL/.

[5] S. Mancoridis, B. S. Mitchell, Y.-F. Chen, and E. R. Gansner. Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures. In *Proceedings of International Conference of Software Maintenance*, pages 50–59, Aug. 1999.

[6] S. Mancoridis, B. S. Mitchell, C. Rorres, Y.-F. Chen, and E. R. Gansner. Using Automatic Clustering to Produce High-Level System Organizations of Source Code. In *Proc. 6th Intl. Workshop on Program Comprehension*, June 1998.

[7] S. Mancoridis, T. Souder, Y.-F. Chen, E. R. Gansner, and J. L. Korn. REportal: A Web-based Portal Site for Reverse Engineering. In *Proc. Working Conference on Reverse Engineering*, Oct. 2001.

[8] B. S. Mitchell. *A Heuristic Search Approach to Solving the Software Clustering Problem.* PhD thesis, Drexel University, Philadelphia, PA, USA, 2002.

[9] B. S. Mitchell and S. Mancoridis. Comparing the Decompositions Produced by Software Clustering Algorithms using Similarity Measurements. In *Proceedings of International Conference of Software Maintenance*, Nov. 2001.

[10] B. S. Mitchell and S. Mancoridis. CRAFT: A framework for evaluating software clustering results in the absence of benchmark decompositions. In *Proc. Working Conference on Reverse Engineering*, Oct. 2001.