2016

Aspect Mining Using Multiobjective Genetic Clustering Algorithms

David G. Bethelmy
Nova Southeastern University, bethelmyd@gmail.com

Aspect Mining Using Multiobjective Genetic Clustering Algorithms

by

David G. Bethelmy

A dissertation submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy

in

Computer Science

College of Engineering and Computing

Nova Southeastern University

2016

We hereby certify that this dissertation, submitted by David Bethelmy, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

_____ _____

Francisco J. Mitropoulos, Ph.D. Date
Chairperson of Dissertation Committee

_____ _____

Sumitra Mukherjee, Ph.D. Date
Dissertation Committee Member

_____ _____

Renata Rand McFadden, Ph.D. Date
Dissertation Committee Member

Approved:

_____ _____

Amon B. Seagull, Ph.D. Date
Interim Dean, College of Engineering and Computing

College of Engineering and Computing
Nova Southeastern University

2016

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Aspect Mining Using Multiobjective Genetic Clustering Algorithms

by
David G. Bethelmy
March 2016

In legacy software, non-functional concerns tend to cut across the system and manifest themselves as tangled or scattered code. If these crosscutting concerns could be modularized and the system refactored, then the system would become easier to understand, modify, and maintain. Modularized crosscutting concerns are known as aspects and the process of identifying aspect candidates in legacy software is called aspect mining.

One of the techniques used in aspect mining is clustering and there are many clustering algorithms. Current aspect mining clustering algorithms attempt to form clusters by optimizing one objective function. However, the objective function to be optimized tends to bias the formation of clusters towards the data model implicitly defined by that function. One solution is to use algorithms that try to optimize more than one objective function. These multiobjective algorithms have been used successfully in data mining but, as far as this author knows, have not been applied to aspect mining.

This study investigated the feasibility of using multiobjective evolutionary algorithms, in particular, multiobjective genetic algorithms, in aspect mining. The study utilized an existing multiobjective genetic algorithm, MOCK, which had already been tested against several popular single objective clustering algorithms. MOCK has been shown to be, on average, as good as, and sometimes better than, those algorithms. Since some of those data mining algorithms have counterparts in aspect mining, it was reasonable to assume that MOCK would perform at least as good in an aspect mining context.

Since MOCK's objective functions were not directly trying to optimize aspect mining metrics, the study also implemented another multiobjective genetic algorithm, AMMOC, based on MOCK but tailored to optimize those metrics. The reasoning hinged on the fact that, since the goal was to determine if a clustering method resulted in optimizing these quality metrics, it made sense to attempt to optimize these functions directly instead of a posteriori.

This study determined that these multiobjective algorithms performed at least as good as two popular aspect mining algorithms, k-means and hierarchical agglomerative. As a result, this study has contributed to both the theoretical body of knowledge in the field of aspect mining as well as provide a practical tool for the field.

Acknowledgements

First of all, I would like to thank my wife for putting up with all that is involved with pursuing a doctorate and having faith that I will finish someday. Then I would like to thank my committee members, Dr. Mitropoulos (chair), Dr. Rand McFadden, and Dr. Mukherjee, for their guidance and helpful feedback. I would especially like to thank Dr. Rand McFadden for her invaluable help understanding her work. Without her help this arduous task would have been even more difficult. Lastly, I would like to thank Dr. Julia Handl, one of MOCK's main creators, who helped with understanding how MOCK worked and how to get it to work properly.

Table of Contents

List of Tables

Tables

List of Figures

Figures

Chapter 1

Introduction

Background

Aspect oriented programming (AOP), introduced by Kiczales et al. (1997), refers to the process of designing software so that the designer focuses on the functional concerns of a software system as opposed to non-functional concerns such as logging and authentication. Non-functional concerns, although important to the system under development, tend to cut across the system rendering the system difficult to understand, maintain, and modify (Kiczales et al., 1997). Current paradigms, such as object oriented programming, are unable to modularize these crosscutting concerns (Kiczales et al., 1997).

AOP allows a developer to modularize crosscutting concerns into entities called aspects (Kiczales et al., 1997). These aspects can then be woven into the code by an aspect weaver either at compile time or run time. AOP techniques, however, are applied at the design and implementation stages of software development. Legacy software, on the other hand, already contains these crosscutting concerns. To be able to maintain and modify legacy code it is desirable that crosscutting concerns be identified and the software put through a process termed aspect refactoring (Van Deursen, Marin, & Moonen, 2003). Since crosscutting concerns tend to manifest themselves in tangled[1] and

[1] Tangled code refers to code that implements crosscutting concerns which are intertwined with functional concerns as well as other crosscutting concerns.

scattered[2] code (Kiczales et al., 1997) it is possible to mine legacy code for aspect candidates by looking for these symptoms. The process of identifying crosscutting candidates in legacy software is called aspect mining (Kellens & Mens, 2005; Kellens, Mens, & Tonella, 2007).

Researchers have developed several aspect mining techniques, many of which have been borrowed from data mining. In their 2007 survey, Kellens, Mens, and Tonella provided an exhaustive list of aspect mining approaches including formal concept analysis, cluster analysis, dynamic analysis, program slicing, software metrics and heuristics, clone detection, and pattern matching. The survey did not list all of the algorithms that implemented these approaches. It merely provided a framework for comparing techniques used in aspect mining. The goal was to help practitioners in the aspect mining field select the right tool(s) for the job. No one technique was proffered as being the best and it was even suggested that a combination of techniques may be more successful than simply using one. This author's research focused on the clustering approach.

Clustering methods look for patterns in the data based on some similarity metric (Han, Kamber, & Pei, 2011; Jain, 2010). Similar objects are grouped into clusters with the final objective being that the objects in a cluster are more similar to each other than to those in other clusters. The degree of similarity is determined by optimizing an objective function, for example, intra-cluster variance. Clustering can be applied to aspect mining if a software system is viewed as being a set, S, of elements (statements, classes,

---

[2] Scattered code is code that implements one concern across many modules and classes.

modules, etc.) and a crosscutting concern as being a subset of those elements that implement that concern (Moldovan & Serban, 2006a). The goal is to partition S into n (n > 1) clusters where each of n-1 clusters contains elements relating to a crosscutting concern with the one remaining cluster containing all the elements related to functional concerns.

Several clustering algorithms have been ported from data mining to aspect mining. Examples of the more popular algorithms ported to aspect mining are the k-means, k-medoids, and hierarchical agglomerative clustering (HAC) algorithms. These algorithms have had some success at isolating crosscutting concerns but have not achieved the ideal hence the need for further research into developing more algorithms. A big part of the reason why these algorithms have failed to achieve the ideal is that there is no one-size-fits-all clustering algorithm (Jain, Murty, & Flynn, 1999; Kleinberg, 2002). This is a direct consequence of the difficulty in determining the precise data distribution in any given data set especially if different parts of the data space have different sizes and densities (Law, Topchy, & Jain, 2004). Also, there will always be a need for better-performing algorithms or algorithms that target specific domains.

One class of algorithms that has been used successfully in data mining, but has not gained much of a foothold in the aspect mining domain, is the class of evolutionary algorithms. Evolutionary algorithms work on the solution space and not on the individual objects in the data space. Heuristic algorithms like k-means and HAC can get stuck at local optima whereas evolutionary algorithms can, theoretically, arrive at global optima (Jain et al., 1999). Evolutionary algorithms are able to achieve this because they

manipulate a population of solutions as opposed to the actual data, resulting in a better global view of the solution space.

One popular type of evolutionary algorithm is a genetic algorithm (GA). GAs mimic evolution in order to arrive at an optimal solution. They do so through the use of selection, crossover, and mutation operators. Selection guides the algorithm towards an optimal solution. Crossover provides diversity. And mutation allows a better view of the global landscape. Even with the ability to home in on global optima, GAs have not had success in aspect mining. Actually, this researcher is only aware of one attempt at using a GA for aspect mining.

Serban and Moldovan (2006c) developed a genetic algorithm for aspect mining (GAM) and tested it against an aspect mining version of the k-means algorithm. Unfortunately, GAM performed poorly in that test as well as in a subsequent test by the same authors in Cojocar and Czibula (2008). Serban and Moldovan made suggestions for modifications that could potentially enhance the performance of GAM. Two of those modifications involved the use of multiple objective functions and a better vector space model[3].

As mentioned earlier, data mining researchers have been using GAs for some time now. In general, GAs (and other heuristic algorithms) attempt to find global solutions by optimizing one objective function. The problem with this is that the objective function being optimized makes some assumptions about the underlying data distribution. This biases the results towards data sets that conform to that distribution. Therefore, other data

[3] Each item in the data set is assigned a vector representing its attributes. The structure of the vector defines the vector space model.

distributions provide challenges to the algorithms optimizing those objective functions and the structure of the data under analysis may be missed altogether. Since there are different objective functions that target different data distributions, some data mining researchers for example, Handl and Knowles (2007a), have experimented with algorithms that attempt to simultaneously optimize different objective functions. These researchers have shown that their multiobjective algorithms can compete successfully against single objective ones.

As far as this researcher was aware of, no multiobjective algorithm has been used in the aspect mining domain. Since multiobjectective algorithms have performed well in data mining, especially when compared to popular data mining algorithms that have counterparts in aspect mining, it was reasonable to assume that those algorithms would have a very good chance of outperforming some of the current aspect mining algorithms.

Problem Statement

Mens, Kellens, and Krinke (2008), discussing the ability of aspect mining techniques to actually find valid candidates, stated that one problem is that most of the techniques will look for only certain symptoms of the presence of crosscutting concerns. This means that only some of the crosscutting concerns are targeted and many will be missed. This is especially true of current clustering methods which attempt to optimize one objective function. In almost all of the cases, this objective function looks for scattering symptoms. Since being able to specify one objective function that tackles more than one symptom is hard, having an algorithm that will simultaneously optimize multiple objective functions will allow each symptom to be addressed by a different

function. Furthermore, the popular algorithms, like k-means and hierarchical agglomerative, tend to gravitate towards local optima. Therefore, this dissertation attempted to solve the problem of how not to be constrained by the underlying data distribution while getting a better view of the global solution space.

Goal

The goal of this research was to show that multiobjective genetic clustering algorithms can perform as good as, if not better than, the aspect mining versions of the k-means and hierarchical agglomerative algorithms. This study looked at two such algorithms, MOCK (MultiObjective Clustering with automatic K determination), created by Handl and Knowles (2004, 2007a), and AMMOC (Aspect Mining using MultiObjective Clustering), a modification of MOCK created specifically for this study. MOCK was chosen because its creators showed that, when used in data mining, MOCK performed better overall than the k-means and hierarchical agglomerative algorithms. Therefore, it seemed reasonable to assume that it would exhibit similar performance against those algorithms which had counterparts in the aspect mining domain. However, since MOCK's objective functions did not directly optimize aspect mining metrics, AMMOC was developed with objective functions that did optimize those metrics.

Research Question

Since multiobjective genetic algorithms have been shown to outperform their single objective counterparts and MOCK, in particular, performed better than popular data mining algorithms like the k-means and hierarchical agglomerative algorithms overall, the question was whether multiobjective genetic algorithms would perform as

good as or better than the aspect mining versions of the k-means and hierarchical agglomerative algorithms.

Relevance and Significance

As mentioned previously, mining legacy systems for aspect candidates provides the opportunity to modularize crosscutting concerns within the software and refactor the system so that it is easier to understand, modify, and maintain. A lot of research has been done in developing aspect mining techniques (Kellens & Mens, 2005; Kellens et al., 2007). However, no one technique has emerged as the best technique. Researchers in the aspect mining community continue to search for better methods. Their fellow researchers in data mining, who also continue the search for better methods, have found that multiobjective algorithms are a promising avenue of research (Law et al., 2004; Zhou et al., 2011). MOCK is an example of one such algorithm (Handl & Knowles, 2004, 2007a).

MOCK also addresses a deficiency with most clustering algorithms and that is the need to supply them with the number of clusters as a parameter. MOCK can automatically determine the number of clusters in one run. Model-based algorithms can also arrive at the number of clusters (Fraley & Raftery, 1998; McNicholas, 2011). However, these algorithms assume that each cluster conforms to a certain probability distribution and select among several models made up of a finite mixture of probability distributions. The model that best fits the data is then selected using a selection criterion such as the Bayesian Information Criterion (Fraley & Raftery, 1998; McNicholas, 2011). MOCK arrives at the number of clusters without making any assumption about the underlying data distribution.

Handl and Knowles (2004, 2007a) tested MOCK against the k-means algorithm as well as an average linkage algorithm, a single linkage algorithm, and an ensemble algorithm and showed that, overall, MOCK outperformed those algorithms. Since those algorithms are widely used in the aspect mining community, having an algorithm that outperforms them will be a definite contribution to aspect mining research.

By experimenting with MOCK and AMMOC in the aspect mining domain, this study contributed to the overall knowledge in that domain by showing that these types of algorithms could be effectively used to detect aspect candidates. To the best of this researcher's knowledge, this has not been done before.

Barriers and Issues

To be successful, evolutionary algorithms must have a good encoding for individuals in their data space, objective functions that make sense, and well-designed evolutionary operators - selection, mutation, and crossover. Furthermore, they must all be designed to work together (Handl & Knowles, 2007a) in order to reduce the search space and control the search. Also, the objective functions used in an MOEA (MultiObjective Evolutionary Algorithm) must be complementary. That is, they must target different aspects of the objective space (Handl & Knowles, 2007a) otherwise the MOEA will be as biased (maybe even more so) as a single objective algorithm. Another issue is that an MOEA can potentially produce a very large set of possible solutions. Although MOCK can be configured to select one of the solutions from its solution set, the result is the best solution from the clustering point of view. This does not mean that the solution is the best from the aspect mining point of view. Finally, genetic algorithms tend to be slower and

require more resources than their non-evolutionary counterparts. For example, Handl and Knowles (2007a) reported that MOCK took approximately 20 minutes on a 10-dimensional data set which had 3565 data elements and 10 clusters and approximately 44 minutes on a 100-dimensional data set which had 2892 elements and 10 clusters.

A couple of these barriers and issues manifested themselves in this study. MOCK and AMMOC took much longer than the non-evolutionary algorithms, especially MOCK which had run times that could last a very long time on high-dimensional data sets. This was because MOCK attempted to determine the best solutions from the set of solutions already obtained. Since AMMOC was not designed to look for such solutions, it ran much faster than MOCK but still took longer on some data sets than the non-evolutionary algorithms.

MOCK produced over 100 solutions per run and they all had to be analyzed since the suggested best solutions were not always the best from an aspect mining point of view. AMMOC produced many solutions as well but not as many as MOCK. AMMOC did not suffer from the latter problem since it optimized aspect mining functions directly.

Even with these concerns it is the opinion of this researcher that the superior performance of those algorithms justified the longer times. However, these algorithms cannot be used in a real time environment.

Limitations

This research was limited by the lack of benchmarks with a well-defined list of aspects (Rand McFadden & Mitropoulos, 2013b). JHotDraw (Gamma & Eggenschwiler, n.d.) is currently the only benchmark that has a well-defined list and is the de facto

benchmark used in aspect mining research. However, as mentioned by Rand McFadden and Mitropoulos (2013b), JHotDraw is a relatively small program that is not reflective of non-aspect oriented software due to its well-structured nature. Therefore, this research could only compare the performance of MOCK against other results based on the analysis of JHotDraw.

MOCK came with its own limitations. Due to its heuristic nature, MOCK can only approximate the true Pareto front and so cannot guarantee the overall best solution. Related to this is the determination of a best solution from the generated set of solutions since that depends on how good the solution and control sets are and hence there were fluctuations from run to run.

Definition of Terms

AMMOC: A multiobjective genetic algorithm designed to optimize two or more aspect mining functions.

Aspect: A modularized non-functional concern.

Aspect mining: The search for aspect candidates in legacy code.

Aspect refactoring: Re-engineering legacy code so that crosscutting concerns are modularized as aspects.

Clustering: The collecting of like entities into groups with unlike entities being in separate groups.

Code scattering: The dispersal of non-functional code across modules and/or classes.

Code tangling: The intermingling of non-functional with functional and/or other non-functional code.

Crosscutting concern: A non-functional concern that is spread across modules and/or classes in the software system.

Genetic algorithm: An algorithm modeled after the process of natural selection. A genetic algorithm represents solutions to an optimization problem as genomes and modifies the population of genes by selecting new individuals based on gene recombination and mutation.

MOCK: A genetic multiobjective clustering algorithm with automatic determination of the number of clusters.

Multiobjective optimization: The process of attempting to simultaneously optimize two or more objective functions.

Non-dominated solution: A feasible solution to a multiobjective optimization problem that has no other solution that is "better" at simultaneously optimizing the individual objective functions.

Pareto front: The image of the Pareto optimal set in objective space.

Pareto optimal set: In multiobjective optimization, feasible solutions represent trade-offs with respect to the optimization of the individual objective functions. Those solutions that are not dominated by any other feasible solution belong to the Pareto optimal set.

Summary

Aspect mining looks for crosscutting concerns in legacy systems so that these concerns can be modularized into aspects and the systems refactored to reflect this modularity. Clustering is one approach to aspect mining. Many clustering algorithms have been used in aspect mining (Cojocar, Czibula, & Czibula, 2009; Kellens & Mens, 2005; Kellens et al., 2007) but currently, all of them attempt to optimize one objective. Kleinberg (2002) proved that it was impossible to have one objective function work on all data distributions because each function had a bias towards one particular distribution. Multiobjective algorithms, which try to optimize several objective functions simultaneously, have been used successfully in the data mining domain (Zhou et al., 2011). In particular, Handl & Knowles (2004) have shown that their multiobjective algorithm, MOCK, can outperform several of the popular singleobjective algorithms like the k-means and hierarchical agglomerative algorithms. Since multiobjective algorithms have not been used in aspect mining but algorithms like k-means and hierarchical agglomerative algorithms have, it was reasonable to assume that multiobjective algorithms, like MOCK and AMMOC, would perform at least as good as these algorithms. This research showed that MOCK and AMMOC generally performed better than k-means and hierarchical agglomerative algorithms in the aspect mining domain.

Chapter 2

Review of the Literature

Kiczales et al. (1997) introduced aspect oriented programming to address the problem of crosscutting concerns in software design. Placing all crosscutting concerns into aspects is the aim of aspect oriented programming (Elrad, Aksit, Kiczales, Lieberherr, & Ossher, 2001; Kiczales et al., 1997). However, aspect oriented programming techniques can only be applied when developing new software. Legacy software still retains crosscutting traits. Aspectizing such code is desirable because it will make the code more understandable, modifiable, and maintainable (Kellens et al., 2007). But in order to aspectize the code, the code's crosscutting concerns need to be identified. The process of identifying crosscutting concerns in non-aspect-oriented code is referred to as aspect mining (Kellens & Mens, 2005; Kellens et al., 2007) and the conversion of these concerns into aspects is called aspect refactoring (Van Deursen et al., 2003).

According to Kellens et al. (2007), it is difficult to manually identify crosscutting concerns and it is also prone to error. Therefore, the trend is towards more automated aspect mining systems (Kellens et al., 2007). Several semi-automated systems exist which require some human interaction. Kellens et al. (2007) carried out a survey of automated aspect mining techniques in which they defined a set of criteria for comparing these techniques as well as a taxonomy of the different techniques. The survey only listed a few algorithms that fell into those categories and did not specify a standard set of algorithms to be used by an aspect mining practitioner. Interestingly, Kellens et al. stated

that combinations of techniques may be more successful than any single technique. To the best of this researcher's knowledge, no current survey on aspect mining techniques has been done. And, although other techniques, like model-based clustering (Rand McFadden, 2011), have been applied to aspect mining since the survey by Kellens et al., there still does not seem to be any standard set of techniques as far as this researcher has been able to determine. It is therefore necessary to design algorithms that can become part of a standard set. But to be included in this set they would have to be validated by exercising them on a standard set of benchmarks and measuring their validity through standard metrics that measure their precision as well as their recall at different levels of granularity (Kellens et al., 2007; Mens et al., 2008; Rand McFadden & Mitropoulos, 2013a; Rand McFadden & Mitropoulos, 2013b).

Kellens et al. (2007) placed aspect mining techniques into two broad categories: those techniques that work with some feature of the source code and those that work with data from the execution of the code. The authors arrived at the conclusion that all the aspect mining techniques work either by using data mining and data analysis techniques like formal concept analysis and cluster analysis or by using techniques like dynamic analysis, program slicing, software metrics and heuristics, clone detection, and pattern matching, to name a few. Nora, Said, and Fadila (2005) also categorized aspect mining techniques but their categories depended on the types of concern symptoms looked for and the type of analysis that is performed on the system to be mined. This led the authors to have two main categories, one for techniques that use code duplication as a symptom and one for those that look for scattering. The researchers assigned techniques like clone

detection and formal concept analysis to the first category and placed fan-in analysis and analysis of recurring patterns in execution traces into the second.

Aspect mining

There is a lot of research that compares the performance of the various aspect mining techniques. For example, Roy, Uddin, Roy, and Dean (2007) applied fan-in analysis, dynamic analysis, and identifier analysis to the four applications: JHotDraw, JDraw, JsokoApplet, and SquareRootDisk. Fan-in analysis and identifier analysis look for symptoms of scattering within the code base. Dynamic analysis can look for both scattering and tangling but by scanning execution traces. The researchers found that, on average, dynamic analysis found a larger percentage of concerns. This was followed by identifier analysis and then fan-in analysis. They noted that the techniques did not match on all concerns. However, they suggested that some combination of the techniques should be able to recover all concerns. Interestingly, they stated that the lack of a solid definition of what a crosscutting concern is could threaten the results of their experiments. It is hard to compare Roy et al.'s findings against a prior similar experiment by Marin, van Deursen, and Moonen (2004) as the latter group did not perform any quantitative analysis. Also, the only application that the two sets of researchers had in common was JHotDraw.

Ceccato et al. (2005) had also done a similar experiment prior to Roy et al.'s (2007) except that they only applied the techniques to JHotDraw. Ceccato et al.'s experiment was a qualitative one so no quantitative comparison could be made with Roy et al.'s experiment. However, both sets of researchers concluded that a combination of the

techniques would be the most effective. They arrived at this because the dynamic analysis technique and the fan-in technique worked in a complementary manner with the identifier analysis technique basically covering some of the concerns that the other two missed. This hypothesis has particular relevance to the research proposed by this paper since the point of multi-objectivity is to optimize objective functions that target different concerns which have complementary effects on the clustering process.

Cluster Analysis

Cluster analysis is a popular technique used in data mining. This technique attempts to take a set of data and organize the data into groups or clusters based on some similarity between data members (Jain et al., 1999). Similarity is determined based on an objective function. The goal of clustering algorithms is to form clusters from the data by optimizing the associated objective functions (Jain et al., 1999). Jain et al. (1999) noted that there was no clustering technique that could be universally applied to multidimensional data sets with the hope of uncovering all structures within those sets. They stated that this had a lot to do with the fact that such techniques come with implicit assumptions about the structure of the data and the fact that objective functions usually target one aspect of the structure. This implies that those functions may miss other facets of the data. Jain et al. attributed the thousands of clustering algorithms described in the literature to this difficulty in designing a general-purpose clustering algorithm.

Clustering algorithms fall into two broad categories: partitional and hierarchical (Jain et al., 1999). Hierarchical algorithms recursively detect clusters either in a top-down manner (divisive algorithms) or a bottom-up (agglomerative algorithms) (Jain et al.,

1999). Either way, the algorithms form a hierarchy of partitions usually represented by dendrograms (Bandyopadhyay, & Saha, 2012). Some examples of hierarchical algorithms are single-link and complete-link. Partitional algorithms determine all the clusters, as a partition of the data, at one time and don't impose a hierarchical structure on the data (Jain et al., 1999). An example of a partitional algorithm is k-means.

Clustering algorithms have been applied to aspect mining with varied success. Ideally, each cluster should be associated with one crosscutting concern and exactly one crosscutting concern should be in a cluster with a cluster for every crosscutting concern. Furthermore, all functional concerns should belong to their own single cluster. Current aspect mining clustering algorithms have not been able to achieve the ideal. (This is another reason why there is continued research into developing algorithms for aspect mining.) Moldovan and Serban (2006a) gave a formal definition for the ideal partition containing aspects. The authors also defined quality metrics (DISP, DIV, PANE, and PREC) for determining how close to the ideal a given partition is.

Many of the current aspect mining clustering algorithms are derived from data mining algorithms. For example, Serban and Moldovan (2006a) modified k-means to create kAM by using their own heuristic for choosing the initial centroids (means) which define a cluster for that algorithm. They modified k-means so that it is relevant to aspect mining. They defined two vector-space models (of dimension L) for encoding the data elements (in this case the methods of a program) that will make up the data space. The first model, M_1, used the vector {FIV, CC} as method attributes where FIV is Marin et al.'s (2004) fan-in value and CC is the number of calling classes. The second model, M_2, used the vector {FIV, B_1, B_2, ..., B_{L-1}} where each B_i is 1 if the method is called from an

application class C_i and 0 otherwise. They defined similarity based on the Euclidean distance metric. They also defined quality metrics InterD and IntraD to measure the degree of clustering and used two of their old metrics (PREC and PAM) for measuring aspect mining performance. They ran their algorithm on Laffra's implementation of Dijkstra's Shortest Path Algorithm as well as the JHotDraw application. For Laffra, they found that, from the clustering perspective, they could not tell which model was better but from the aspect mining point of view M_2 was better. They stated, categorically, that the inability to decide in the clustering case was due to the vector space model. For JHotDraw, M_1 was better from the aspect mining point of view but M_2 was better from the clustering viewpoint. Here they postulated that the lack of correlation between the two viewpoints could be due to the vector space model. They did not compare the performance of kAM with any other algorithm.

In another experiment, Serban and Moldovan (2006b) compared the KM[4] (kAM), FCM, and HAC algorithms. (They used the KM algorithm to determine the number of clusters before running FCM and HAC.) They used the squared sum error (SSE) function to partition the data and used ACC and PAM quality metrics to evaluate the partition from the aspect mining perspective. (ACC was newly created by them.) They ran their algorithms on Laffra's implementation of Dijkstra's Shortest Path Algorithm as well as the JHotDraw application. For Laffra, M_1 proved to be better for all algorithms with FCM showing the best results and for JHotDraw, M_1 worked better for KM and HAC with M_2

[4] It is this researcher's belief that KM and kAM are the same algorithm. This is based on the observation that Cojocar and Czibula (2008) referred to the same original paper for a description of both algorithms. And although there were discrepancies in the results for the DISP and DIV metrics in those papers, the discrepancies could have been the result of k-means' finicky behavior even under different runs using the same data.

being better for FCM. The authors' overall conclusion was that the vector space models needed to be improved. The authors conducted a similar experiment using kAM and HAC on the Theatre, JHotBox, and Laffra applications looking for symptoms of scattering (Moldovan & Serban, 2006b). They manually analyzed the resulting clusters and found that the first set of clusters obtained had basically the same methods implementing crosscutting concerns regardless of the clustering algorithm.

Cojocar and Czibula (2008) investigated the performance of the KM, kAM, FCM, HAC, HAM, and GAM algorithms using the vector models of Serban and Moldovan (2006a). The researchers ran their algorithms on the JHotDraw application. They used the DISP, DIV, PREC, and ACTE quality measures. In general, they found that:

- GAM, a genetic algorithm developed by Serban and Moldovan (2006c), consistently performed the worst for model M_1 and ran for too long under M_2 to be able to get any results. They concluded that the algorithm wasn't appropriate for aspect mining. Interestingly, when Serban and Moldovan (2006c) had tested GAM using vector space model M_2 and compared it to kAM, kAM outperformed it but not by much and only in one quality metric. However, Serban and Moldovan claimed that the vector space model had a lot to do with GAM's performance. It should be noted that kAM and GAM were only tested on Laffra's application (which has much less lines of code and much less methods than JHotDraw) and there were only two quality measures used, ACC and PAME.

- None of the algorithms were able to identify all of the concerns and for almost all of the crosscutting concerns there were elements that did not belong to the clusters upon analysis.

- They were unable to determine which combination of algorithm and vector space model was right for aspect mining.

Zhang, Guo, and Chen (2008) developed a technique called clustering-based fan-in analysis (CBFA) in which they assigned a fan-in value (FIV) to an entire cluster and then used the cluster fan-in to rank the clusters. The ranking was used to recommend the clusters most likely to have crosscutting concerns. The cluster FIV was simply the sum of the individual method FIVs. The authors used a simple binary vector model and the Jaccard coefficient as the similarity measure. They ran their algorithm on JHotDraw and a subset of the Linux system. They used two quality metrics called concern coverage and true positives. They compared their technique against the fan-in analysis, the dynamic analysis, and the identifier analysis techniques. Their results showed that CBFA performed better than or as good as the other techniques on both JHotDraw and the Linux subsystem.

Cojocar, Czibula, and Czibula (2009) did a solid comparison of the kAM, HAM, PACO, and HACO techniques. Here the authors compared two algorithms, kAM and HAM, that used vector space models against two algorithms, HACO and PACO, that didn't. The first two algorithms used the M_2 model and measured distance between objects with the Euclidean metric. PACO was based on the k-medoids algorithm that was modified so that it initialized the first set of medoids like kAM. PACO used three

distance semi-metrics: one that targeted scattering concerns, one that targeted tangling concerns, and one that targeted both. HACO was based on the hierarchical agglomerative algorithm but it used a heuristic to determine the initial number of cluster centers. The authors used the DIV and DISP quality measures, as defined by Moldovan and Serban (2006b), to evaluate the clustering techniques. Some of the authors' main findings were:

- The algorithm that performed best was HACO which was able to group crosscutting concerns better.
- All algorithms spread crosscutting concerns over two or more clusters.
- HACO appeared to be the best algorithm for use in aspect mining.
- No optimal partitions were obtained.

The authors concluded that they would see an improvement in HACO's results for the DISP measure by improving the distance semi-metric that looked for scattering and tangling symptoms.

Czibula, Cojocar, and Czibula (2011) defined two new quality metrics, CORE and CODI, and showed where they would be applicable in aspect mining. However, they did not test their metrics so the performance of those metrics is unknown. CORE was designed to measure the degree of grouping of crosscutting concerns which the authors referred to as partitioning. CODI was designed to determine how important the ordering of the clusters was when analyzing them. The authors stated that, in their opinion, as far as what to look for in aspect mining, partitioning was the most important and ordering the least.

Rand McFadden and Mitropoulos (2012) evaluated six clustering methods three of which were model-based and three were heuristic-based. They applied these methods to six vector-space models, three previously-defined models (fanIn_numCallers, fanIn_hasMethod, sigTokens) and three models that they developed (fanIn_sigTokens, fanIn_numCallers_sigTokens, fanIn_numCallers_hasMethod_sigTokens). They used PREC, DISP, DIV2, and MTA as their quality metrics. (PREC and DISP were defined by Moldovan and Serban (2006a) whereas DIV2 and MTA were defined by Rand McFadden (2011).) Their findings indicated that, overall, the model-based methods outperformed the heuristic-based ones and that their new vector-space models were a contributing factor in many of the cases.

Tribbey and Mitropoulos (2012) designed three vector space models based on FIV. FIV is an aggregate measure and does not retain any information about the data that was used to compute it. The researchers kept a binary N x N matrix of method call relationships instead of just keeping the FIV aggregate which was calculated by summing the rows of the matrix. The model that used that matrix was referred to as M_{FIV}. The transpose of the matrix allowed the calculation of fan-out values (FOV) and the related model was labeled M_{FOV}. The product of the two matrices with the subsequent division of each of its rows by that row's diagonal value (as long as that diagonal value was non-zero) yielded a third matrix and its model was referred to as M_{COM}. Since the authors were going to test their models on JHotDraw and JHotDraw (after filtering) contained 2248 methods, they decided to use principal component analysis (PCA) to reduce the dimensionality of the models. They analyzed JHotDraw using the k-means algorithm which had its original cluster configuration initialized using the k-means++ seeding

algorithm (Arthur & Vassilvitskii, 2007). The authors measured aspect mining quality using the DIV, DISP, and KPREC metrics with the latter measuring recall. They also measured cluster quality configuration with a validity index, RS, obtained from the literature. They concluded that, overall, their models gave results that, although not stellar, made their approach viable. However, they admitted that there was no one model that they could identify as the best.

Fillus and Vergilio (2012) introduced a clustering based approach, named CAAMPI (Clustering Based Approach for Aspect Mining and Pointcut Identification), that included an integrated process to discover aspect candidates and to identify pointcuts for the purpose of refactoring. The integrated process was achieved by aggregating three distance measures. One measure addressed scattering. Another measure took care of operation cloning. And a third handled naming conventions. The resulting measure was a linear combination of the three with the sum of the weights being equal to 1. Once groups were identified they were ranked using four ranking measures that the authors defined. Groups with higher ranking scores were those most likely to be crosscutting concerns. The authors did not define an aggregate ranking score so it is unclear how the four scores were used to rank the groups. The authors used quality measures for clustering (DISP, DIV2), for ranking (RANK), and for pointcut identification (COV and USE). The authors performed experiments using k-medoids, hierarchical, and CHAMELEON clustering algorithms using distance measures reported in the literature as well as their own aggregate distance measure. They tested their method on JHotDraw, Apache Tomcat, and HSQLDB. The authors' results showed that the hierarchical method using their distance measure performed the best followed by CHAMELEON which also used their measure.

The idea behind the research is a good one but some of the formulas don't seem to be correct so it is difficult to agree with the results.

Genetic Clustering Algorithms

Genetic algorithms belong to a class of algorithms called evolutionary algorithms (Freitas, 2008; Jain et al., 1999; Srinivas & Patnaik, 1994). Evolutionary algorithms are modeled after life's evolutionary process where populations are made up of individuals comprised of genetic material. Future populations are made by selecting the "fittest" individuals from the current populations, combining their genetic material in some way to produce new individuals with the chance that some of that genetic material may be modified by some mutation agent. The genetic material is represented by chromosomes which contain genes where each gene has a value that affects some feature of the particular individual to whom that chromosome belongs. The genetic materials manipulated by an evolutionary algorithm represent solutions to some optimization problem (Freitas, 2008; Jain et al., 1999; Srinivas & Patnaik, 1994). It should be noted that evolutionary algorithms must start with solutions, not necessarily good ones, to a problem under investigation. The role of the evolutionary algorithm is to find the globally optimal solution starting with an initial population of solutions (Jain et al., 1999).

Genetic algorithms (GAs) have been around since the 1970's (Srinivas & Patnaik, 1994). They have been used in data mining (Freitas, 2008; Naldi, de Carvalho & Campell, 2008), clustering (Hruschka, Campello, Freitas & De Carvalho, 2009), and in aspect mining (Serban & Moldovan, 2006c). They differ from other evolutionary algorithms mainly in the use of a crossover operator (Srinivas & Patnaik, 1994). Such an

operator combines two chromosomes by exchanging genetic material between them. The evolutionary operators (selection, crossover, and mutation) are the key to having the GAs cover a global space and converge on a global optimum. Choosing the right mixture of chromosome encoding, operators, and objective function is crucial to the success of a GA (Handl & Knowles, 2007a). Since selection depends on the "fitness" of an individual, the fitness function must also be carefully selected (Hruschka et al., 2009).

As far as clustering goes, k-means tends to be the benchmark against which GAs are measured. There is good reason for this; k-means is simple to implement, it runs roughly in O(n) time, and it is popular (Jain et al., 1999). GAs tend to be more complex and are computationally expensive because of either their fitness functions and/or their crossover strategies (Krishna & Narasimha Murty, 1999). GAs, however, can search a global solution space looking for a global optimum. K-means tends to find local optima and is affected by its initial cluster configuration including the number of clusters (k) (Jain, 2010).

Maulik and Bandyopadhyay (2000) developed a GA that outperformed k-means when tested on four artificial data sets and three real-life data sets. The data set dimensions ranged from two to ten and the number of clusters from two to nine. They measured performance strictly on the value of their clustering metric which had to be minimized. No data on time or space complexity was given.

Many researchers try to take advantage of k-means' efficiency by hybridizing k-means with a GA. Krishna and Narasimha Murty (1999) developed a hybrid they called GKA (genetic k-means algorithm) that used a one-step k-means algorithm to replace

crossover in their GA. They also defined a distance-based mutation operator instead of just a random one. The mutation operator basically considered whether the change made sense cluster-wise before making the change. Although their results indicated that GKA was better than k-means, the experiments were done on small data sets with low-dimensional data and a small number of clusters. This is important since the researchers noted that the search for a global optimum would become harder, hence take more time, for larger numbers of clusters as the search space would increase combinatorially.

Genetic Aspect Mining Clustering Algorithms

To the best of this researcher's knowledge the only genetic algorithm adapted for aspect mining is one by Serban and Moldovan (2006c). Their algorithm used the M_2 vector space model (defined earlier) to represent each data element. The actual chromosomes were an n-dimensional integer vector with n being the number of items in the data set. Each entry in the vector represented which cluster the associated data item belonged to. The authors used their own heuristic for determining the number of clusters. Their objective function was the sum of squared Euclidean distances from each data item to its respective cluster center. The fitness function was the difference between the maximum sum of squared distances over the entire data set and their objective function. So maximizing the fitness meant minimizing the objective function. The authors compared GAM with their own kAM (Serban & Moldovan, 2006a) because k-means minimized a similar objective function. They used their own aspect mining quality measures, ACC and PAME, to determine how good their algorithm was at identifying candidates for aspectizing. The results of their experiment indicated that GAM performed slightly worse than kAM. They attributed the poor performance mainly to the vector

space model that was used to define the attributes of their data items. The researchers gave several suggestions that they claimed would improve GAM's performance, two of which were the use of a different vector space model and the use of multiple objective functions.

As mentioned earlier in this paper, later research by Cojocar and Czibula (2008) compared GAM with several other non-genetic algorithms and concluded that GAM was not appropriate for aspect mining. However, that version of GAM was the original version and Cojocar and Czibula used the same vector space model. It is therefore unclear how a modified version of GAM would perform.

Multiobjective Clustering Algorithms

Objective functions used in clustering have biases as to what the underlying properties of the data set are and tend to look for clusters that match those biases. For example, k-means will find compact, (hyper-) spherical clusters because of its objective function but it will miss other features of the data space, for example, a spiral set of data elements (Law et al., 2004). Therefore, unless the data space has a homogeneous distribution, a clustering algorithm with a single objective function won't be able to find all the correct clusters (Law et al., 2004). Kleinberg (2002) actually proved that it was impossible for any clustering algorithm that used one objective to be effective at clustering all data distributions. Multiobjective algorithms address this issue by attempting to find partitions by simultaneously using the objective functions that target the different parts of the data space (Law et al., 2004).

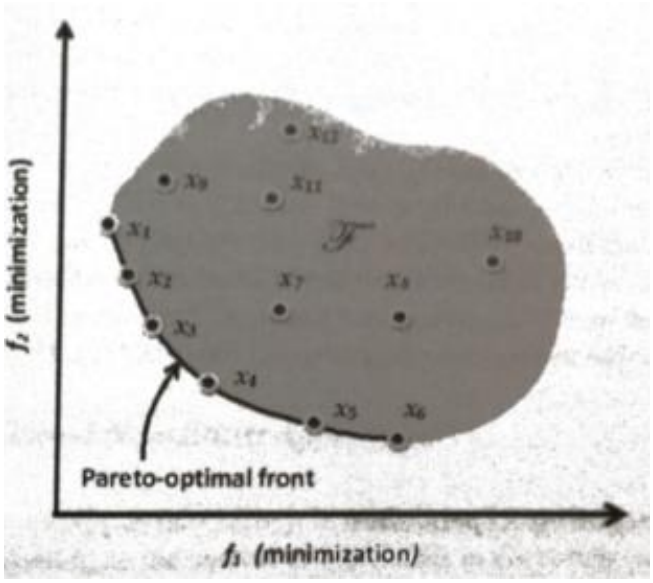
Multiobjective Evolutionary Clustering Algorithms.

In a 2011 paper, Zhou et al. observed that even though evolutionary multiobjective optimization was still in its early stages, there were very many publications. The authors referred to data that showed that by January 2011, there were more than 5600 publications on evolutionary multiobjective optimization. In a more recent survey on multiobjective evolutionary clustering, Mukhopadhyay, Maulik, and Bandyopadhyay (2015) mentioned that there was a great deal of literature on multiobjective evolutionary clustering algorithms. Their survey gives a modern, comprehensive introduction to research in multiobjective evolutionary clustering.

Many multiobjective evolutionary algorithms (MOEAs) take a different approach at attempting to find an optimal solution. The theory behind this approach is based on the concept of Pareto optimality. Informally, the set of Pareto optimal solutions is the set of feasible solutions that are not dominated by any other feasible solution (Coello, 1999; Maulik, Bandyopadhyay, & Mukhopadhyay, 2011; Mukhopadhyay, Maulik, & Bandyopadhyay, 2015). Basically, a solution is Pareto optimal if there is no other feasible solution that is as good at optimizing all objective functions and is better at optimizing at least one of the functions. Note that there may be feasible solutions that do not dominate each other. The collection of these non-dominated solutions forms the Pareto optimal set. The image of the Pareto optimal set in objective space is known as the Pareto front. For example, in Figure 1, x_1 through x_6 represent the image in objective space of the set of Pareto optimal solutions for the images of feasible solutions shown in the shaded region. The "best" solution, which is a subjective one based on the application, will be retrieved

from the Pareto optimal set. (See Appendix A for a more formal definition of Pareto optimality.)

Figure 1 - Image of a Pareto Front (Maulik, Bandyopadhyay, & Mukhopadhyay, 2011)



It is important to note that, given any set of feasible solutions, there will always be a set of non-dominated solutions since it will either be the case that, given any two solutions in the space, one dominates the other or neither dominates the other. The resulting set of non-dominated solutions may or may not belong to the Pareto optimal set. A good MOEA will strive to find a set of non-dominated solutions which are as diverse as possible and which are as close as possible to the Pareto optimal front (Deb, 1999, 2001). Deb (1999, 2001) pointed out that these are not easy goals. He listed multi-modality, deception, isolated optima, and collateral noise as some key features that would prevent a MOEA from generating solutions that are very close to the Pareto optimal front. He also stated that convexity or non-convexity in the Pareto optimal front, discontinuity or discreteness in the Pareto optimal front, and non-uniform distribution of solutions in the search space and in the Pareto optimal front would prevent a MOEA from

maintaining a diverse set of Pareto optimal solutions (Deb, 1999, 2001). Deb's text, *Multi-Objective Optimization Using Evolutionary Algorithms* (2001), provides a thorough coverage of MOEAs.

MOCK – MultiObjective Clustering with Automatic K-determination

Handl and Knowles (2004, 2007a) developed a MOEA, called MOCK, that is designed to work on numerical data[5]. Handl and Knowles showed that this algorithm could find clusters that are closer to the optimal solutions than those obtained from single-objective clustering techniques. Furthermore, when using MOCK, there is no need to specify the number of clusters a priori. The authors showed that MOCK performed consistently across a wide range of data although when the data was skewed towards one of the objectives MOCK performed a little worse than the corresponding single-objective algorithm. The authors did state that the use of MOCK depended on knowledge of the structure of the data and that if the data has well-defined properties that are aligned with a particular objective, then MOCK should not be used.

The authors used two objective functions in their algorithm although MOCK is capable of handling more than two. One objective function measured overall deviation and the other measured connectedness. To represent a chromosome the authors used a locus-based adjacency vector which is a graph-based representation such that connected segments of the graph represent clusters. The authors stated that they could not use an encoding based on cluster centers because such an encoding would not allow the evolution of solutions in conflict with the underlying spherical cluster model imposed on

[5] Later on they developed a version of MOCK referred to as MOCK-around-medoids (Handl & Knowles, 2005b) designed to cluster similarity data.

the data by their overall deviation objective. This would give more weight to that objective. The authors' experimental data showed that, overall, the performance of MOCK was better than k-means, average linkage, single linkage, and Strehl's ensemble method. The researchers demonstrated that it had to be the simultaneous application of the two objective functions that resulted in MOCK's superior performance (Handl & Knowles, 2005c). More recently, Handl and Knowles (2012) compared the use of MOCK's two objective functions against three other pairs in order to determine if the choice of objective functions mattered. They showed that the quality of the clustering solutions does depend on the choice of the objective functions and the results of their experiments seemed to indicate that the ones used by MOCK produced higher quality clusters in general.

One of the problems with the original version of MOCK was scalability. The authors stated that this was addressed in a second version of MOCK (Handl & Knowles, 2005a, 2007a). However, the largest data set tested had approximately 5000 items, the highest dimensionality was 100, and the most amount of clusters was 40. Furthermore, these did not occur in the same test suite.

Phases of MOCK.

MOCK's algorithm (Handl & Knowles, 2004, 2007a) consists of two phases: a clustering phase and a model selection phase. The clustering phase is responsible for generating a set of mutually non-dominated clustering solutions. The algorithm that implements this phase is heavily based on one of their earlier algorithms, a MOEA called PESA-II (Corne, Jerram, Knowles, & Oates, 2001). After the set of non-dominated

solutions is generated, the model selection algorithm examines those solutions and provides a subset of solutions that it considers to be more promising from a clustering point of view. This phase of MOCK can even produce what it considers to be a "best" solution.

Clustering phase. PESA-II is a MOEA that uses elitism to help converge solutions towards the Pareto optimal front and niching to help spread the solutions uniformly across the front. The algorithm keeps track of two populations, an internal population (IP) and an external population (EP). The IP is of fixed size and the EP is of variable but limited size. The IP is used to generate new solutions and it is the population that undergoes the evolutionary operations of selection, recombination, and mutation. The EP holds non-dominated solutions in niches which are implemented as a hypergrid of equally sized cells in objective space.

First, the EP is initialized with non-dominated solutions. Then, for each generation, the following occurs:

1. Solutions from the EP are selected at random from randomly selected populated niches and placed in the IP. The number of solutions placed in the IP is limited by the fixed size of the IP.
2. Solutions in the IP now undergo recombination and mutation based on probabilities of such operations occurring.
3. Once the population in the IP has evolved an attempt is made to place each of its non-dominated elements into the EP. A solution from the IP can only get into the EP if it is non-dominated by the solutions in the EP. Any solutions in the EP that

are dominated by the new solution are removed from the EP. If no solutions are dominated by the new solution and the size of the EP is not at its maximum, then the new solution is placed in its niche. If the EP is full, then the new solution can only get into its niche if that niche has less members than some other niche. If that is the case, the new solution is placed in its niche and another solution is removed, at random, from the most populated niche.

After the algorithm iterates through all generations, the solutions stored in the EP are returned as the final solutions.

Handl and Knowles (2004, 2007a) pointed out that the implementation of the PESA-II algorithm in MOCK had to be adapted for the purpose of clustering by specifying two or more appropriate objective functions, a suitable genetic encoding that defined a cluster partition, one or more genetic operators such as mutation and crossover, and a method for initializing the EP. They stressed the importance of making the right choices for each of these requirements as the performance and scalability of the algorithms heavily depended on those choices. They also stated that, in order to design an effective evolutionary algorithm for clustering, the genetic encoding, the genetic operators, and the objective functions must work together harmoniously in order to reduce the search space while effectively guiding the search.

As stated earlier, the authors chose a graph-based encoding for representing a chromosome (cluster partition). In such an encoding, there are as many genes as there are data items and the value of each gene ranges from 1 up to the number of data items. A value of *j* for the *i*th gene indicates that data item *i* is connected to data item *j* implying

that both data items are in the same cluster. This makes cluster reassignment easy since all that needs to be done is to change the value of the gene in question. Identifying clusters now becomes a matter of identifying the connected components in the graph.

MOCK's two objective functions, overall deviation and connectivity, were chosen in order to measure cluster compactness and connectedness, respectfully. The first objective function, overall deviation, is simply the overall summed distances between data items and their cluster centers. This measure is similar to intracluster variance used by many clustering algorithms. The actual definition given by Handl and Knowles (2007a) is

$$Dev(C) = \sum_{C_k \in C} \sum_{i \in C_k} \delta(i, \mu_k)$$

where C is the set of all clusters, μ_k is the centroid of cluster C_k, and δ is the chosen distance function. For the latter, the Euclidean distance function is used (Handl & Knowles, 2007a). The second, connectivity, measures the degree to which data points are placed in the same clusters as their neighbors. The computation therefore requires a user-defined parameter indicating how many neighbors contribute to connectivity. The formula for connectivity given by the authors is

$$Conn(C) = \sum_{i=1}^{N} \left(\sum_{j=1}^{L} x_{i,nn_{ij}} \right)$$

where $x_{r,s} = \begin{cases} \frac{1}{j} & if\ \nexists C_k : r \in C_k \wedge s \in C_k \\ 0 & otherwise \end{cases}$, nn_{ij} is the jth nearest neighbor of data item i, N is the size of the data set, and L is the parameter that determines how many neighbors

play a part in the calculation of connectivity. However, it is the belief of this researcher that the previous 'where' clause was incorrectly written and should have been:

$$\text{where } x_{i,nn_{ij}} = \begin{cases} \frac{1}{j} & if \nexists C_k : i \in C_k \wedge nn_{ij} \in C_k \\ 0 & otherwise \end{cases}$$

It should be noted that the parameter *L* is also used to determine what a solution can mutate to. Since a solution in MOCK is represented as a set of connected components of a graph, mutation means breaking a connection and making another by changing the value of a particular gene. If the new value for the gene could be any of the values from 1 to the number of data items, *N*, this would mean that the clustering algorithm was always looking at the entire search space which is of size N^N. Handl and Knowles therefore used what they call a nearest-neighbor mutation that allows a chromosome to mutate only if the new value represented one of its *L* nearest neighbors. This reduces the size of the space that mutation sees to just L^N with *L* being much smaller than *N*. Furthermore, this neighborhood-biased mutation strategy extends to the probability that a gene will mutate by adding a bias that leans more towards breaking a link to a neighbor that is further away and creating a link to a closer neighbor.

MOCK's initial population is created from the data in two ways. Half of the initial population is created from a minimum spanning tree (MST) derived from the entire data set. This results in a single cluster. The algorithm then removes what are considered to be "interesting" links. These "interesting" links are those that lead to the creation of a true cluster as opposed to outliers. *L* is used to determine "interesting" links. The other half of the initial population is generated by running the k-means algorithm for different numbers of clusters. Handl and Knowles (2007a) justified this initialization strategy

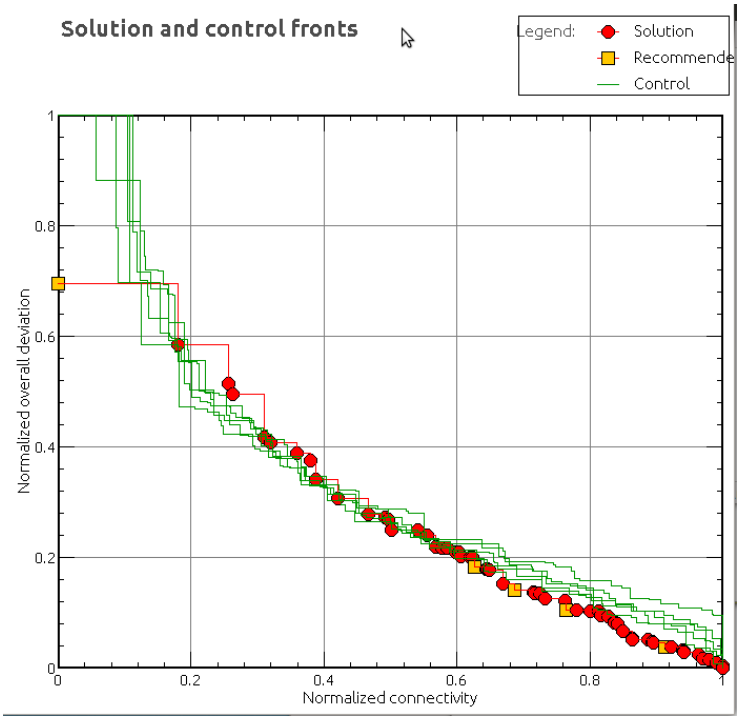
based on the fact that the MST part targets solutions that perform well under connectivity whereas the k-means part targets solutions that perform well under overall deviation. (The version of PESA-II in the 2004 paper does not use kmeans to generate any part of the initial population.)

Model selection phase. In this phase MOCK selects solutions that it considers as promising from its resulting Pareto optimal approximation set and even proffers what it considers to be the "best" of those solutions. MOCK does this by adapting a statistical method called the Gap statistic (Tibshirani, Walther, & Hastie, 2001). The Gap statistic determines the number of clusters in a data set based on a plot of the performance of the clustering method, as measured by an internal evaluation metric, as a function of the number of clusters. This statistic expects a "knee" (a bump) to occur in the plot at the place where the most suitable number of clusters occurs. However, Handl and Knowles (2007a) pointed out that the Gap statistic is basically a single objective approach to model selection since it only considers a single clustering objective. Therefore, the authors adapted the statistic to take into account multiple objectives. Knees are now points in objective space that correspond to solutions where a small improvement in the value of one objective would result in a large decline in the value of another objective (Branke, Deb, Dierolf, & Osswald, 2004). To help identify these knees Handl and Knowles defined what they termed an attainment score for each point in the solution front.[6] This score is the minimum Euclidean distance between a given point in the solution front and sets of Pareto optimal control fronts obtained from running MOCK on randomly generated data from the same data space. When these attainment scores are plotted as a

[6] It should be noted that Branke et al. (2004) used different methods for determining the knees.

function of the number of clusters those solutions with scores that are local maxima are considered as promising solutions while the global maximum is taken as the "best" solution. Figure 2 shows the results obtained by running MOCK once. The solutions are represented by the red dots with the recommended solutions (knees) in yellow. The green lines are the control fronts. In the figure the number of clusters increases from left to right with the recommendations having clusters 4, 24, 26, 28, and 33 respectively. The "best" solution has 4 clusters with an attainment score of 0.0906423, 0 connectivity, and 0.696032 overall deviation.

Figure 2 - Image of a solution front from one run of MOCK using five control fronts



MIE-MOCK

Tsai, Chen, and Chiang (2012) transformed MOCK into MIE-MOCK (Multiple Information Exchange - MultiObjective Clustering with automatic K determination). The

researchers made two changes to MOCK. The first was that they used pools of crossover and mutation operators to choose from as opposed to MOCK's use of only one crossover and one mutation operator. They claimed that the use of multiple recombination operators increased the search diversity. The second change involved the use of a different method of obtaining the one "best" solution from the Pareto optimal set. Instead of the GAP statistic they used the PBM index as well as the Davies-Bouldin (DB) index. However, they didn't give any reason for using these indices.

Tsai et al. compared the performance of MOCK with that of MIE-MOCK using five measures: overall deviation, connectivity, accuracy, PBM, and DB. Of the five, accuracy was the only measure that the algorithm did not attempt to optimize and PBM and DB were not used by MOCK for model selection. The researchers never used MOCK's modified GAP statistic as a measure of performance so it is questionable whether the choice of the metric had anything to do with the better performance of MIE-MOCK. Regardless, the use of a choice of recombination operators seems to be a reasonable modification to make since it shouldn't make the clustering quality worse than using single operators.

Chapter 3

Methodology

Research Methodology

The proposed research adopted the methodology of several researchers in aspect mining (Moldovan & Serban, 2006a; Rand McFadden & Mitropoulos, 2012; Zhang et al., 2008) by defining a software system to be made up of a set of methods, assigning attributes to each method, applying a clustering algorithm on the set of attribute vectors (in this case, MOCK and AMMOC), analyzing the data using quality metrics, and presenting tabular reports based on the findings. The software under investigation was JHotDraw 5.4b1 (Gamma & Eggenschwiler, n.d.). This is the de facto standard in aspect mining because its crosscutting concerns are well documented. Although other software, such as Laffra's version of Dijkstra's algorithm and Tomcat, have been analyzed, JHotDraw is the one most analyzed by researchers (Rand McFadden & Mitropoulos, 2013b). This means that results are readily available from a wide range of sources.

Vector Space Model Data Generation

This research used the data set generated by the FINT tool (Marin, Moonen, & van Deursen, 2006) run on JHotDraw 5.4b1. The data created by FINT consisted of method signatures, their fan-in values (FIVs), and a list of their calling methods for JHotDraw. FINT required a filtering threshold to be set and this researcher chose to use a threshold of 0 since it was the threshold value used by other researchers.

A Java program was written and given the aforementioned data set as input. The program transformed the data into vector representations as defined by the vector space models described in Rand McFadden and Mitropoulos (2012) and defined in the next section.

Vector Space Models

Moldovan and Serban (2006a) defined a formal model for clustering-based aspect mining. In this model, a software system is a set $S = \{s_1, s_2, ..., s_n\}$ of n elements where each element can be a statement, a method, a class, etc. In this research each one of the n elements was a method and each method was described as an x-dimensional vector. The following vector space models were used to describe methods in the system:

Model 1. **fanIn_NumCallers** (Moldovan & Serban, 2006b): Each method is described by a vector (FIV, CC) where FIV is the fan-in value or number of methods that call this method and CC is the number of calling classes. Here x = 2.

Model 2. **fanIn_hasMethod** (Moldovan & Serban, 2006b): Each method is described by a vector (FIV, B_1, B_2, ..., B_m) where FIV is the fan-in value and each B_i (i = 1, ..., m) is a 1 if the method is called by at least one method from an application class, C_i (i = 1, ..., m), in the system and 0 otherwise. Here x = m+1.

Model 3. **sigTokens (**Zhang et al., 2008): Each method is described by a vector (O_1, O_2, ..., O_p) where each O_i is a binary value that depends on whether the method has attribute A_i (i = 1, ..., p) or not. Each attribute is a token obtained

from splitting the signatures of all methods in the system, consolidating the tokens, and eliminating redundancies and trivial tokens like "in". Here x = p.

Model 4. **fanIn_sigTokens** (Rand McFadden, 2011): Each method is described by a vector (FIV, O_1, O_2, ..., O_p) where FIV is the same as that used in models 1 and 2 and the O_i's are from model 3. Here x = p+1.

Model 5. **fanIn_numCallers_sigTokens** (Rand McFadden, 2011): A combination of models 1 and 3.

Model 6. **fanIn_numCallers_hasMethod_sigTokens** (Rand McFadden, 2011): A combination of models 1, 2, and 3.

Experimental Procedure

The R implementations of the k-means (kmeans) and agglomerative hierarchical clustering (agnes) algorithms were executed on the data sets generated from the vector space models. These algorithms required that k be specified. A Java program was written to generate a set of centroids based on a heuristic defined in Serban and Moldovan (2006a). The number of centroids provided the values of k that were used for kmeans and agnes.

MOCK (Version 1.1) was executed in Ubuntu 12.04 LTS and also given the data sets generated from the vector space models. MOCK has a Java user interface that allowed this researcher to select the data file to be clustered and set some of the parameters required by that part of the code that actually does the clustering. The Java interface allowed the selection of the distance function to be used in overall deviation, the

maximum number of clusters, whether there was normalization or not, the value of *L* (the number of nearest neighbors under consideration), and the number of control surfaces. All other parameters required by the clustering stage were either the default values or required that the code be modified slightly. (See Appendix B for a list of parameters and their default values.) Several runs were made with different parameter settings before a suitable set was found. The actual set of values are shown in a later section of this paper.

One benefit of using the Java interface was that it displayed the solution and control fronts graphically. It also gave a graphical representation of the clustering for a particular solution. However, the latter was not as helpful, especially for large data sets, since the display was crowded and hard to read.

An adaptation of MOCK called AMMOC (Aspect Mining using MultiObjective Clustering) was written in Java and designed to optimize the aspect mining functions DISP and DIV. (AMMOC implements a version of MOCK's PESA-II engine that does the clustering.) The reasoning behind the decision to optimize DISP and DIV instead of overall deviation and connectivity was based on the observation that DISP and DIV behaved similarly to MOCK's objective functions from a clustering point of view and, instead of retroactively determining whether MOCK's solutions resulted in optimal clusters as far as those aspect mining functions were concerned, it made more sense to create a program that attempted to optimize them directly. AMMOC consists of 13 classes and roughly 2000 lines of code in all.

The current version of the AMMOC was not designed to suggest a "best" solution since, after initial testing, it was determined that the number of solutions generated by the

program on each run was small enough to allow analysis of all of them and that implementing that facet of MOCK would only slow down the program needlessly. (The determination of the "best" cluster size is one of the most computationally expensive parts of MOCK.)

Aspect Mining Quality Metrics

The clustering results produced by all of the methods were analyzed using Java programs, written by this researcher, designed to calculate the aspect mining metrics PREC (precision), DISP (dispersion), and DIV (diversity) (Moldovan & Serban, 2006a) as well as DIV2 (a modified diversity), and MTA (methods to analyze) (Rand McFadden & Mitropoulos, 2012). The programs also produced results of the distribution of crosscutting concerns across clusters. The aspect mining metrics used determine the quality of the cluster partitions from an aspect mining viewpoint. To be able to define these metrics it is necessary to have a formal definition of what it means to cluster a set of crosscutting concerns. The definition used will be the one proffered by Moldovan and Serban (2006a).

Let S = {s_1, s_2, ..., s_n} be a set of n elements representing a software system where each element is a method. Let a crosscutting concern C be a subset of S. Let CCC = {C_1, C_2, ..., C_q}, be the set of all crosscutting concerns where q is the number of elements in the set. And let NCCC = $S - \bigcup_{i=1}^{q} C_i$ be the set of all elements that do not represent a crosscutting concern. The goal of a hard clustering algorithm is to partition S into a set, K = {K_1, K_2, ..., K_p}, of p clusters (p ≥ q) where $S = \bigcup_{i=1}^{p} K_i$ such that $K_i \cap K_j = \emptyset, 1 \leq i, j \leq p, i \neq j$, and, ideally, there is a 1-to-1 map, m, from the set CCC to the set K. The

map m relates an element of CCC with one of K based on set equality. That is, m(C_i) = K_j if and only if $C_i = K_j$, i ϵ {1, ..., q}, j ϵ {1, ..., p}. Note that in the ideal case, if NCCC is not empty, p should be equal to q+1 and NCCC should be equal to K - {C_1, C_2, ..., C_q}. In other words, in the ideal case, each crosscutting concern is equal to one and only one cluster in the partition K and all non-crosscutting concerns belong to the one remaining cluster in the partition. Based on this formal definition the following quality metrics are defined:

PREC (**PREC**ision). This measures the percentage of found crosscutting concerns (Moldovan & Serban, 2006a).

Let *T* be an aspect mining clustering technique. Then the precision with which *T* can find crosscutting concerns CCC in a partition K is defined as:

$$PREC(CCC, K, T) = \frac{1}{|CCC|} \sum_{i=1}^{|CCC|} prec(C_i, K, T)$$

$$where\ prec(C_i, K, T) = \begin{cases} 1\ if\ C_i\ was\ discovered\ by\ T \\ 0\ otherwise \end{cases}$$

$$0 \leq PREC \leq 1\ and\ should\ be\ maximized.$$

Note that a concern is considered found if at least one method that implements the concern is found.

DISP (**DISP**ersion). This measures the degree to which crosscutting concerns are spread across clusters (Moldovan & Serban, 2006a). Ideally, each crosscutting concern should be in its own cluster and nothing else should be in that cluster.

The dispersion of the set CCC in the partition K is defined as:

$$DISP(CCC, K) = \frac{1}{|CCC|} \sum_{i=1}^{|CCC|} disp(C_i, K)$$

where *disp(C, K)* is the dispersion of a crosscutting concern C across the partition K and is defined as:

$$disp(C, K) = \frac{1}{|D_C|} \; where \; D_C = \{k \mid k \in K \; and \; k \cap C \neq \emptyset\}$$

$$0 < DISP \leq 1 \; and \; should \; be \; maximized.$$

DIV (**DIV**ersity). This measures the degree to which each cluster has crosscutting concerns that are different from other concerns (Moldovan & Serban, 2006a). Ideally, each cluster should be equal to one crosscutting concern.

The diversity of a partition K with respect to a set CCC is defined as:

$$DIV(CCC, K) = \frac{1}{|K|} \sum_{i=1}^{|K|} div(CCC, K_i)$$

where *div(CCC, k)* is the diversity of a cluster k and is defined as:

$$div(CCC, k) = \frac{1}{|V_k| + \tau(k)}$$

where $V_k = \{C \mid C \in CCC \; and \; C \cap k \neq \emptyset\}$ and $\tau(k) = \begin{cases} 1 \; if \; k \cap NCCC \neq \emptyset \\ 0 \; if \; k \cap NCCC = \emptyset \end{cases}$

$$0 < DIV \leq 1 \; and \; should \; be \; maximized.$$

DIV2 (**DIV**ersity). Similar to DIV but only considers one of the clusters with no crosscutting concerns (Rand McFadden & Mitropoulos, 2012; Rand McFadden & Mitropoulos, 2013a). DIV2 also needs to be maximized.

MTA (**M**ethods **T**o **A**nalyze). This measures the number of methods that need to be analyzed in a given ordering of the clusters in the partition before all methods that implement all crosscutting concerns can be found. The lower this value is the better the clustering (Rand McFadden & Mitropoulos, 2012; Rand McFadden & Mitropoulos, 2013a).

Based on the theory of Cojocar and Czibula (2008) and Moldovan and Serban (2006c, as cited by Rand McFadden, 2011, p. 40) the following hold with respect to a clustering technique T:

> A clustering partition K is considered optimal if DIV(CCC, K) = 1 and DISP(CCC, K) = 1 (Cojocar & Czibula, 2008).
>
> Given two clsutering partitions, K_1 and K_2, K_1 is considered "better" than K_2 relative to aspect mining if DIV(CCC, K_1) > DIV(CCC, K_2) and DISP(CCC, K_1) > DISP(CCC, K_2) (Moldovan & Serban, 2006c).
>
> If the previous inequalities do not hold, then K_1 is considered "better" than K_2 relative to aspect mining if DISP(CCC, K_1) + DIV(CCC, K_1) > DISP(CCC, K_2) + DIV(CCC, K_2) (Moldovan & Serban, 2006c).

Analysis and Presentation of Results

The results of the experiments are presented and interpreted in the following chapter. The analysis includes a comparison of the results with those from previous results obtained from the literature. Some of the data are presented in tables.

Resources

The FINT (Software Engineering Research Group, 2008) and JHotDraw 5.4b1 (Gamma & Eggenschwiler, n.d.) software were obtained from the Web. The k-means and hierarchical agglomerative clustering algorithms (kmeans and agnes packages) from R were used to generate the respective data for comparison. The current version of MOCK (1.1) was downloaded from the Web (Handl & Knowles, 2007b). AMMOC and programs to generate the vector models and analyze the clustering results were implemented in Java by this researcher. Microsoft Excel 2013 was used to clean up some of the data and facilitate interpretation of the aspect mining results.

The research was done on a laptop with an AMD A8 Quad Core processor running at 1.70 GHz. The system was a 64-bit system with 8 GB of RAM and a 1 TB hard drive. The system was set up so that it was able to dual boot into Windows 10 Home Edition or Ubuntu 14.04 LTS.

Chapter 4

Results

The purpose of this research was to show that multiobjective genetic clustering algorithms were at least as good at clustering aspect mining data as the singleobjective k-means and hierarchical agglomerative algorithms and it is the opinion of this researcher that the goal was achieved. The multiobjective algorithms used were MOCK, created by Handl and Knowles (2004, 2007b) and AMMOC, created specifically for this research. MOCK optimized two generic functions, connectivity and connectedness, whereas AMMOC targeted the two aspect mining metrics, DISP and DIV. The data that these algorithms clustered were obtained from running a FINT analysis on JHotdraw 5.4b1. JHotDraw 5.4b1 was chosen because it is the de facto benchmark in aspect mining and its crosscutting concerns are well documented.

Although Moldovan and Serban (2006b) analyzed JHotDraw 5.2, the FINT analysis for this research was done on JHotDraw 5.4b1 because the available list of concerns and fan-in values came from the Web site of the Software Engineering Research Group (2008) and Moldovan and Serban did not list the concerns used in their experiment.

Data Generation

The FINT threshold for filtering was set to 0 with the following being filtered based on information from research results found on the Software Engineering Research

Group Web site (http://swerl.tudelft.nl/bin/view/AMR/FanInAnalysisResults) as well as results from Rand McFadden (2011):

> Setters and getters as identified by Java naming conventions. (At first setters and getters by function were also filtered but that resulted in at least one concern not being found.)
>
> All libraries.
>
> CH.ifa.draw.test.* (also filtered from the set of callers)
> CH.ifa.draw.util.collections.* (this includes CH.ifa.draw.util.collections.jdk11.* and CH.ifa.draw.util.collections.jdk12.*)
> CH.ifa.draw.util.CollectionsFactory
> CH.ifa.draw.util.ReverseListEnumerator
> CH.ifa.draw.standard.ReverseFigureEnumerator
> CH.ifa.draw.standard.HandleAndEnumerator
> CH.ifa.draw.standard.SingleFigureEnumerator
> CH.ifa.draw.standard.FigureAndEnumerator
> CH.ifa.draw.standard.HandleEnumerator
> CH.ifa.draw.standard.FigureEnumerator
> CH.ifa.draw.framework.FigureEnumeration
> CH.ifa.draw.framework.HandleEnumeration

At first, it did not seem to make sense to include the samples package (CH.ifa.draw.samples) that was in both callees and callers but that resulted in one of the concerns being missed indicating that previous analyses included that package. Therefore, the FINT analysis was rerun with the samples package included. This filtering resulted in 2381 callees for threshold 0. Once the FINT files were generated, Java code was written to do the following:

- Generate a file with method names and another file with class names.
- Generate vector files for each of the 6 vector models in formats required by R and MOCK.

- Generate centroids (and hence cluster numbers) for k-means for each of the 6 vector models. For each vector model, centroids were generated using a set of minimum distance thresholds as required by the version of k-means used by Serban and Moldovan (2006a).

After generating the vector data and centroids, the kmeans and agnes algorithms were run in R. First, the kmeans algorithm was run using the centroids created from the heuristic that used a threshold of 2. (The reason for choosing this threshold will be addressed later on in this paper.) Then it was run using the cluster numbers based on the centroids from the heuristic but with random centroid generation. The agnes algorithm was then run and the dendrograms for each vector model were cut at the places specified by the cluster numbers used for kmeans.

MOCK was run on the raw data for each vector model. For each run, MOCK generated at least 100 clustering files in addition to other supporting files. One of the supporting files contained recommendations that MOCK considered the most promising clustering solutions. MOCK also generated a solutions file which had information on each solution giving the values of the objective functions that each solution was attempting to optimize. At first, whenever the recommendations file had inadequate information, this solutions file was visually analyzed to determine what this researcher considered to be the most likely candidates for analysis. The choices were based on the values for the two objective functions that a clustering solution was to simultaneously minimize. Likely candidates were determined first by those values that were small but close to each other. The number of clusters was also a factor in determining the choices. It was reasoned that cluster values which were closer to the optimal (10 clusters each

with methods from one concern plus 1 cluster with all other non-concern methods) would produce better aspect mining results but that did not always turn out to be the case. Later on, all MOCK's solutions for all 6 vector models were analyzed.

When the clustering files were created by all three methods, Java programs were written to clean up the data and get it in a form for analyzing. Java code was also written to analyze the clustering results using the aspect mining metrics, DIV, DISP, DIV2, PREC, and MTA. Once the aspect mining results were created Microsoft Excel 2013 was used to help interpret those results. From those results (which will be discussed in more detail later) a decision was made to create an adaptation of MOCK that would attempt to optimize the aspect mining metrics DISP and DIV directly. This program, Aspect Mining using MultiObjective Clustering (AMMOC), was written in Java and was used to cluster all vector model data. Since the program was written to optimize DISP and DIV, aspect mining analysis of the resulting clusters gave corresponding values for those metrics and that analysis was run primarily to gain information concerning the other metrics as well as the distribution of concerns across clusters.

After getting aspect mining analysis results from MOCK and AMMOC, kmeans and agnes were rerun using the clustering values from MOCK and AMMOC's top ten clustering solutions based on their DISP+DIV values. The resulting clustering partitions from all runs were then analyzed using the aspect mining metrics.

Data Analysis

From the FINT files, 96 caller methods and 2381 callee methods were generated. From those methods, 296 classes were distilled. From their analysis of JHotDraw 5.2

Moldovan and Serban (2006b) stated that it had 190 classes. A quick comparison between JHotDraw 5.4b1 and 5.2 revealed that there are 234 files in the CH source folder for JHotDraw 5.2 and 555 files and 39 folders for JHotDraw 5.4b1. This would explain the increase in methods and classes. (It is interesting to note that those researchers referenced a paper by Marin, van Deursen, and Moonen (2004) who analyzed JHotDraw 5.4b1 instead of version 5.2.)

The methods and classes were used to create vector files for each of the 6 vector models. The following were the vector dimensions for each model:

Vector Model 1 (FIV_CC): 2

Vector Model 2 (FIV_HasMethod): 297

Vector Model 3 (sigTokens): 681

Vector Model 4 (FIV_sigTokens): 682

Vector Model 5 (FIV_CC_sigTokens): 683

Vector Model 6 (FIV_CC_hasMethod_sigTokens): 979

Vector model 1 had a large range of values in the first two dimensions and, as a result, those dimensions carried the weight when model 1 was merged to form models 2, 4, 5, and 6. In the first dimension, model 1 had a maximum of 90 and a minimum of 0 and the second dimension had a maximum of 58 and a minimum of 0. All the other dimensions in models 1, 2, 4, 5, and 6 were binary. Model 3 was completely binary.

This research mainly used raw data values in the runs following the research method used by Rand McFadden (2011). However, just for comparison purposes, MOCK was set to normalize data in some of the runs. The results did not turn out to be significantly different from the runs that used non-normalized data and hence are not

presented here. The results of the aspect mining analysis were synthesized in Microsoft Excel 2013. For each vector model the data was sorted in descending order based on the sum of the aspect mining metrics DISP and DIV and appropriately filtered in order to interpret the results for different methods and vector models.

Individual Clustering Method Analysis

Method 1: kmeans with heuristic

First, centroids were generated using the heuristic of Serban and Moldovan (2006a). Centroids were generated because k-means (as well as agglomerative hierarchical clustering algorithms) requires that the number of clusters be specified. Serban and Moldovan developed this heuristic to determine the optimal number of centroids, and hence clusters, with respect to aspect mining. In their paper that described this heuristic, the authors used a minimum distance of 1. (Setting a distance threshold is required by the algorithm that generates the centroids.) When the minimum distance of 1 was used by this researcher, 649 centroids were generated just for vector model 2. This implied that higher-dimensional vectors would yield higher cluster numbers. But, for a vector space of 2381 vectors, that would result in many singletons. Minimum distances of 2, 3, and 4 were evaluated but when threshold 4 was used in this study only one centroid was generated for vector model 3. After researching the reason for this it was found that the most tokens for any method was 6. This meant that the highest number of positions that two vectors could differ in was 12. For a pure binary vector this would give a distance of $\sqrt{12}$ which is less than 4 resulting in only the initial centroid being generated by the algorithm. Serban and Moldovan stated that they used a threshold of 1 but did not state what optimal numbers of clusters were obtained. In another paper by the same

authors (Moldovan and Serban, 2006b) they stated that the number of clusters generated by their heuristic when run on vectors obtained from JHotDraw 5.2 was 20 for vector model 1 and 34 for vector model 2. However, they did not state what threshold they used. This study only used centroids generated from thresholds 2 and 3 because of the reasons stated earlier. The number of centroids obtained in this study from using the threshold values of 2 and 3 is shown in Table 1.

Table 1 - Cluster Numbers from the Heuristic Algorithm for Thresholds 2 and 3

Vector Model	Threshold of 2	Threshold of 3
1	42	31
2	189	84
3	87	4
4	182	19
5	240	39
6	622	189

From this table it is evident that the optimal cluster numbers for models 1 and 2 did not come close to those recorded by Moldovan and Serban (2006b) but without knowing what their threshold was it is not conclusive that this study's results are incorrect.

After the centroids were generated, the kmeans function from the statistical program R used those centroids to cluster the data sets for the various models. The aspect mining analysis of the clustering results are shown in Table 2.

Table 2 - Aspect Mining Results for kmeans (Predetermined Centroids) Based on DISP+DIV

Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2
3	87	0.692	0.921	1.613	0.507
3	4	0.950	0.648	1.598	0.530
4	182	0.548	0.942	1.490	0.523
5	240	0.485	0.963	1.448	0.712
6	622	0.413	0.989	1.402	0.829
6	189	0.419	0.961	1.380	0.796
2	189	0.418	0.961	1.379	0.807
2	84	0.439	0.896	1.335	0.743
1	42	0.503	0.770	1.273	0.613
5	39	0.513	0.754	1.267	0.544
4	19	0.631	0.628	1.259	0.411
1	31	0.508	0.692	1.200	0.566

As Table 2 shows, kmeans performed the best on vector model 3 (87 clusters) based on its DISP+DIV (1.613) value with the DIV (0.921) value dominating the DISP (0.692) value. The DIV2 value dropped drastically which indicates that there were many clusters that only contained non-crosscutting concerns. These results are supported by the following cluster distribution among crosscutting concerns. (The number in parentheses refers to the number of methods that make up the concern. The clusters where the concern was found follow the parenthetical value.)

Consistent behavior (21) 12 26 31 36 63 72

Decorator (6) 15 29 50 63

Composite (12) 63 64

Observer (10) 38 63

Adapter (1) 63

Command (2) 63

Contract enforcement (3) 63

Persistence (6) 6 63

Undo (3) 63

Exception handling (1) 58

The distribution also shows why the DISP was relatively low; five of the concerns were spread out among different clusters, especially the consistent behavior concern.

Method 2: kmeans with random centroids

Table 3 - Aspect Mining Results for kmeans (Random Centroids) Based on DISP+DIV

Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2
6	622	0.626	0.987	1.613	0.675
5	240	0.631	0.962	1.593	0.497
6	189	0.634	0.957	1.591	0.421
3	87	0.679	0.904	1.583	0.507
4	182	0.611	0.957	1.568	0.441
2	84	0.639	0.913	1.552	0.439
4	19	0.717	0.83	1.547	0.461
5	39	0.640	0.859	1.499	0.390
2	189	0.530	0.953	1.483	0.553
3	4	0.900	0.523	1.423	0.523
1	31	0.631	0.678	1.309	0.475
1	42	0.499	0.746	1.245	0.573

The kmeans function was again run in R using the number of clusters equivalent to the number of centroids but with random centroid generation. As shown in Table 3, it performed the best on vector model 6 (622 clusters) with a DISP value of 0.626 and a DIV value of 0.987. Its DIV2 value was 0.675. The high DIV value was due to the fact that only a few clusters had more than one crosscutting concern. Again, the large drop from DIV to DIV2 can be understood by looking at its clustering distribution.

Consistent behavior (21) 20 164 180 323 368 379 386 573 612

Decorator (6) 152 162 175 275 484

Composite (12) 162 329 354 556 575

Observer (10) 16 527 556 587

Adapter (1) 527

Command (2) 573

Contract enforcement (3) 573

Persistence (6) 422 515

Undo (3) 573

Exception handling (1) 303

Here, an extremely large number of clusters had only non-crosscutting concerns.

Method 3: agnes

The agnes function from the R clustering library is a hierarchical agglomerative algorithm that starts with n (number of elements) one-element clusters and builds a dendrogram of clustering solutions where the clustering partition at level i (counting from the bottom of the dendrogram) has less clusters than the solution at level i-1. The highest level contains all of the elements in one cluster. To obtain a clustering solution the dendrogram is cut at the level containing the number of required clusters, k. In this experiment the values for k were the same as prescribed by the heuristic used for kmeans.

Table 4 - Aspect Mining Results for agnes Based on DISP+DIV

Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2
3	4	0.950	0.648	1.598	0.530
3	87	0.675	0.915	1.590	0.507
4	182	0.553	0.944	1.497	0.465
5	240	0.503	0.965	1.468	0.696
6	622	0.418	0.989	1.407	0.825
6	189	0.435	0.963	1.398	0.802
2	189	0.419	0.961	1.380	0.796
2	84	0.444	0.900	1.344	0.752
1	31	0.605	0.708	1.313	0.569
5	39	0.513	0.756	1.269	0.587
1	42	0.503	0.748	1.251	0.577
4	19	0.579	0.618	1.197	0.395

Table 4 shows that this function performed the best on vector model 3 with a 4-cluster result. The high DISP (0.950) and relatively low DIV (0.648) indicate that agnes was successful at not spreading out the concerns but was unable to keep each concern in a separate cluster. This is not that surprising given the low number of clusters. The fact that the DIV and DIV2 values were so close supports this conclusion as does the following distribution.

Consistent behavior (21) 2

Decorator (6) 2

Composite (12) 1 2

Observer (10) 2

Adapter (1) 2

Command (2) 2

Contract enforcement (3) 2

Persistence (6) 2

Undo (3) 2

Exception handling (1) 2

As the distribution shows, all the concerns were found in cluster 2 with only the Composite concern being spread across 2 clusters. The other two clusters have no crosscutting concerns. This again supports the close DIV and DIV2 values.

Method 4: MOCK

As mentioned earlier in this paper MOCK attempts to simultaneously optimize two objective functions, one measuring compactness and the other measuring connectedness. Its performance was, in general, competitive with the other algorithms. MOCK generated several hundred clustering solutions for all of the vector models combined only some of which are reported in this paper.

MOCK required that certain parameters be configured prior to execution. Handl and Knowles (2007a) suggested a set of parameter configurations that worked for them. (See Appendix B.) However, this set did not have very good results for this research so the following configuration set, shown in Table 5, was used:

Table 5 - Parameter Values Used in MOCK

Parameter	Value
Maximum clusters	150
Control fronts	5
Distance metric	Euclidean
L (connectivity)	10
Number of generations	1000
Recombination rate	0.7

The main difference between the sets is that the maximum number of clusters was set to 150 and the number of control fronts was set to 5. Actually, the authors did suggest that

researchers might wish to increase those two parameters from the values used by them. The decision to increase the number of clusters was based on cluster values returned by the heuristic for k-means. Those numbers tended to be larger than 50 which was the default. The number was first raised to 100 but some of the clustering solutions gave numbers of clusters close enough to warrant increasing the limit. Therefore, it was increased to 150. The number was not changed any more since analysis of clustering solutions showed that lower numbers of clusters tended to give very good aspect mining results. MOCK uses the control fronts to determine the cluster values it suggests as the most promising ones but, using the original setting of 3, MOCK's suggested values did not always give better aspect mining results than MOCK's other partitions so it made sense to increase the number of control fronts. This resulted, as indicated by Handl and Knowles, in increasing MOCK's run time considerably and only gave clustering configurations that resulted in slightly better aspect mining values. It was therefore decided that no further increase in the control fronts would be made. Obviously, it was possible to tweak parameters further but there are infinitely many combinations even when reasoning is based on potential outcome. This researcher did not feel that it was necessary to continue tweaking when current settings were producing good results.

Table 6 - MOCK's Top 10 Aspect Mining Results Based on DISP+DIV

Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2	DISP+DIV2	MTA
2	6	1.000	0.848	1.848	0.545	1.545	2249
2	6	1.000	0.848	1.848	0.545	1.545	2254
2	5	1.000	0.818	1.818	0.545	1.545	2274
4	5	1.000	0.818	1.818	0.545	1.545	2288
2	5	1.000	0.818	1.818	0.545	1.545	2297
2	5	1.000	0.818	1.818	0.545	1.545	2302
2	8	0.950	0.824	1.774	0.530	1.480	2215
2	8	0.950	0.824	1.774	0.530	1.480	2221
4	4	1.000	0.773	1.773	0.545	1.545	2302
5	4	1.000	0.773	1.773	0.545	1.545	2307

As shown in Table 6, vector model 2 dominated with a DISP of 1 and DIV of 0.848 for both solutions with 6 clusters. (Many of the solutions shown in the table differ only in their MTA values. Also, the table contains only the top 10 results as there are too many results to list.) Interestingly, the DISP values were perfect for all but the 8-cluster solutions for vector model 2. MOCK was doing an excellent job of not spreading concerns across clusters but this came at a price for diversity. It turned out that all of the crosscutting concerns ended up in one cluster along with some non-crosscutting concerns. This was supported by the associated distribution of concerns. (The fact that the one cluster with crosscutting concerns also had non-crosscutting concerns is not evident from the distribution but was confirmed by other analyses.)

Consistent behavior (21) 1

Decorator (6) 1

Composite (12) 1

Observer (10) 1

Adapter (1) 1

Command (2) 1

Contract enforcement (3) 1

Persistence (6) 1

Undo (3) 1

Exception handling (1) 1

Method 5: AMMOC

Table 7 - AMMOC's Top 10 Aspect Mining Results Based on DISP+DIV

Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2	DISP+DIV2	MTA
3	42	1.000	0.978	1.978	0.545	1.545	2277
3	44	1.000	0.968	1.968	0.533	1.533	2310
3	119	0.900	0.980	1.880	0.520	1.420	2169
4	10	0.950	0.859	1.809	0.530	1.480	2344
4	32	0.833	0.920	1.753	0.572	1.406	2335
4	22	0.850	0.902	1.752	0.463	1.313	2316
5	9	0.900	0.843	1.743	0.648	1.548	2376
2	6	0.950	0.765	1.715	0.530	1.480	2329
5	6	0.950	0.765	1.715	0.530	1.480	2373
6	6	0.950	0.765	1.715	0.530	1.480	2373

AMMOC is based on MOCK's clustering engine PESA-II but designed to directly optimize the aspect mining metrics DISP and DIV. It was run with the same set of configurations as the one used by MOCK in this research. Table 7 shows that the results from AMMOC were extremely good. Vector model 3 gave the best results with a perfect DISP (1.0) and almost perfect DIV (0.978). Again, the drastic drop in the DIV2 score (0.545) indicates that there were many clusters with only non-crosscutting concerns contributing to the calculation for DIV and masking the true diversity. (Again, the table contains only the top 10 results as there are too many results to list.)

The distribution of concerns for the solution with 42 clusters follows.

Consistent behavior (21) 1

Decorator (6) 1

Composite (12) 1

Observer (10) 1

Adapter (1) 1

Command (2) 1

Contract enforcement (3) 1

Persistence (6) 1

Undo (3) 1

Exception handling (1) 1

As can be seen, all concerns end up in cluster 1, a fact that is corroborated by the very high DIV but mediocre DIV2.

Vector Model Analysis

Vector Model 1: fanIn_NumCallers

Table 8 - Top 10 Methods for Vector Model 1 Based on DISP+DIV

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2
AMMOC	1	6	0.950	0.765	1.715	0.530
MOCK	1	8	0.900	0.700	1.600	0.520
AMMOC	1	2	1.000	0.545	1.545	0.545
MOCK	1	18	0.717	0.820	1.537	0.461
MOCK	1	16	0.717	0.798	1.515	0.461
MOCK	1	20	0.700	0.807	1.507	0.448
MOCK	1	15	0.717	0.784	1.501	0.461
AMMOC	1	20	0.692	0.809	1.500	0.706
MOCK	1	6	0.900	0.600	1.500	0.520
MOCK	1	19	0.700	0.796	1.496	0.448

This vector model had only 2 dimensions with relatively large values per dimension when compared to the other dimensions of the other vector models, especially vector model 3. Table 8 shows that with this vector model AMMOC and MOCK had the best results among all the methods as determined by the DISP+DIV values. Not only did they have the best results, they exceeded the DISP+DIV values for the heuristic method

closest to them by a large amount. As shown in the figure, AMMOC had the highest DISP+DIV value of 1.715 with a DISP of 0.95 and a DIV of 0.765. This is followed by MOCK with a DISP+DIV of 1.6, a DISP of 0.9, and a DIV of 0.7. The nearest method to MOCK and AMMOC based on DISP+DIV was the kmeans with heuristic with a DISP+DIV of 1.273, a DISP of 0.503, and a DIV of 0.77 for 42 clusters. Following that was agnes with a DISP+DIV of 1.251, a DISP of 0.503, and a DIV of 0.718 for the same number of clusters. Much lower in the table was kmeans with random centroids with a DISP+DIV of 1.245, a DISP of 0.499, and a DIV of 0.746 again with the same number of clusters.

Vector Model 2: fanIn_hasMethod

Table 9 - Top 11 Methods for Vector Model 2 Based on DISP+DIV

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2
MOCK	2	6	1.000	0.848	1.848	0.545
MOCK	2	6	1.000	0.848	1.848	0.545
MOCK	2	5	1.000	0.818	1.818	0.545
MOCK	2	5	1.000	0.818	1.818	0.545
MOCK	2	5	1.000	0.818	1.818	0.545
MOCK	2	8	0.950	0.824	1.774	0.530
MOCK	2	8	0.950	0.824	1.774	0.530
MOCK	2	4	1.000	0.773	1.773	0.545
MOCK	2	7	0.950	0.799	1.749	0.530
MOCK	2	7	0.950	0.799	1.749	0.530
AMMOC	2	6	0.950	0.765	1.715	0.530

This vector model had 297 dimensions with all but the first dimension being binary. Again, the genetic methods had some results for DISP+DIV that far exceeded those of the heuristic methods. Table 9 shows the top 11 values which are all from the genetic methods. (A few of the rows in the table have the same DISP, DIV, and DIV2 values but they differ for MTA.) The kmeans with random centroids, occurring

considerably lower down, was the first of the heuristic methods to appear with a DISP of 0.530 and a DIV of 0.953 giving a DISP+DIV of 1.483 which was significantly lower than both the top MOCK value of 1.848 and AMMOC's 1.715. Next was the agnes algorithm with a DISP+DIV of 1.38, a DISP of 0.419, and a DIV of 0.960. Following agnes closely was the kmeans with heuristic which only differed by having a DISP of 0.418. All three heuristic methods worked on 189 clusters. MOCK was obviously the clear winner for this vector model. It even had 10 values that were better than AMMOC's. It is unclear why MOCK outperformed AMMOC to that extent since AMMOC was targeting the aspect mining metrics while MOCK was targeting arbitrary clustering objectives.

Vector Model 3: sigTokens

Table 10 - Top 10 Methods for Vector Model 3 Based on DISP+DIV

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2
AMMOC	3	42	1.000	0.978	1.978	0.545
AMMOC	3	44	1.000	0.968	1.968	0.533
AMMOC	3	119	0.900	0.980	1.880	0.520
MOCK	3	14	0.800	0.846	1.646	0.460
MOCK	3	8	0.850	0.793	1.643	0.447
MOCK	3	13	0.800	0.834	1.634	0.460
kmeansCentroids	3	87	0.692	0.921	1.613	0.507
MOCK	3	15	0.783	0.823	1.606	0.468
agnes	3	4	0.950	0.648	1.598	0.530
kmeansCentroids	3	4	0.950	0.648	1.598	0.530

This vector model had only binary values in each of its 681 dimensions. As shown in Table 10, the genetic methods had 6 values for DISP+DIV that were better than those of the first occurrence of a heuristic method which was kmeans with predetermined centroids. AMMOC was the clear winner here with two of its results having perfect

scores for DISP (1, 1) and nearly perfect scores for DIV (0.978, 0.968). As previously noted, the values for DIV2 (0.545, 0.533) for those rows in the table put things in a better perspective by showing that the high DIV values were due to a large number of clusters with only non-crosscutting concerns in them. The distribution of crosscutting concerns for the first row of the table showed that all crosscutting concerns ended up in one cluster leaving 41 clusters with only non-crosscutting concerns. There was a similar result for the second row which had 9 of the concerns together in one cluster and the 10th in another cluster leaving 42 of the 44 clusters with only non-crosscutting concerns. The DIV2 values for the top two rows with AMMOC were higher than all other DIV2 values. Even though AMMOC and MOCK had the best results among all of the methods, it is interesting that two of the heuristic methods, kmeans with heuristic and agnes, placed within the top 10 for this vector model.

Vector Model 4: fanIn_sigTokens

Table 11 - Top 10 Methods for Vector Model 4 Based on DISP+DIV

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2
MOCK	4	5	1.000	0.818	1.818	0.545
AMMOC	4	10	0.950	0.859	1.809	0.530
MOCK	4	4	1.000	0.773	1.773	0.545
MOCK	4	4	1.000	0.773	1.773	0.545
AMMOC	4	32	0.833	0.920	1.753	0.572
AMMOC	4	22	0.850	0.902	1.752	0.463
MOCK	4	3	1.000	0.697	1.697	0.545
AMMOC	4	18	0.833	0.857	1.691	0.572
AMMOC	4	5	0.950	0.718	1.668	0.530
AMMOC	4	25	0.758	0.867	1.626	0.526

This vector model had 682 dimensions of which 681 were binary. Although both genetic algorithms still dominated the others, MOCK edged out AMMOC for top spot with a perfect DISP that contributed to a DISP+DIV of 1.818 versus AMMOC's

DISP+DIV of 1.809. (See Table 11.) Even so AMMOC had a better DIV value (0.859) than MOCK's (0.818). Again, however, the DIV2 values for both indicate that there were many clusters with only non-crosscutting concerns. The distribution of concerns supports this as it showed that all of the crosscutting concerns were in one cluster. MOCK left 4 out of 5 clusters without any crosscutting concerns while AMMOC had 8 out of 10 clusters without any concerns. All 10 concerns were in cluster 1 and one concern was spread across clusters 1 and 2.

None of the heuristic methods made it into the top 10. The first heuristic method, kmeans with random centroids, showed up very far down in the ranking. It had 182 clusters, a DISP of 0.611, and a DIV of 0.957 giving a DISP+DIV of 1.568. Its DIV2, however, was 0.441 which again indicates that many of the 182 clusters had no crosscutting concerns in them. This is supported by its cluster distribution shown below.

Consistent behavior (21) 5 9 13 53 78 103 112 159 175
Decorator (6) 5 78 103 162
Composite (12) 5 13 53 103
Observer (10) 13 53 131 159
Adapter (1) 131
Command (2) 53
Contract enforcement (3) 53
Persistence (6) 13 78 128 159
Undo (3) 53
Exception handling (1) 142

Vector Model 5: fanIn_numCallers_sigTokens

Table 12 - Top 10 Methods for Vector Model 5 Based on DISP+DIV

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2
MOCK	5	4	1.000	0.773	1.773	0.545
MOCK	5	7	0.950	0.799	1.749	0.530
MOCK	5	7	0.950	0.799	1.749	0.530
AMMOC	5	9	0.900	0.843	1.743	0.648
AMMOC	5	6	0.950	0.765	1.715	0.530
MOCK	5	6	0.950	0.765	1.715	0.530
MOCK	5	6	0.950	0.765	1.715	0.530
MOCK	5	6	0.950	0.765	1.715	0.530
MOCK	5	5	0.950	0.718	1.668	0.530
MOCK	5	8	0.900	0.742	1.642	0.483

This vector model had 683 dimensions of which 681 were binary. Again, MOCK and AMMOC had better results than any of the other methods. The top 10 results are shown in Table 12. As seen in the table, MOCK had three DISP+DIV results that were better than the best result for AMMOC. However, MOCK's top results were primarily due to a higher DISP value. AMMOC had the highest DIV value (0.843) of the set. DIV2 values continued to be low because most of the concern methods ended up in one or two clusters.

The cluster distribution for AMMOC's 9-cluster result (shown below) had one cluster (3) that only had the methods from one crosscutting concern (Consistent Behavior) and no methods from non-crosscutting concerns. This fact is not evident from the concern distribution and was obtained from other analyses. All other clusters had at least some non-crosscutting concerns in them.

Consistent behavior (21) 1 3

Decorator (6) 1 2

Composite (12) 1

Observer (10) 1

Adapter (1) 1

Command (2) 1

Contract enforcement (3) 1

Persistence (6) 1

Undo (3) 1

Exception handling (1) 1

Vector Model 6: fanIn_numCallers_hasMethods_sigTokens

Table 13 - Top 10 Methods for Vector Model 6 Based on DISP+DIV

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2
AMMOC	6	6	0.950	0.765	1.715	0.530
AMMOC	6	18	0.783	0.857	1.640	0.571
kmeansRandom	6	622	0.626	0.987	1.613	0.675
MOCK	6	4	0.950	0.648	1.598	0.530
MOCK	6	4	0.950	0.648	1.598	0.530
kmeansRandom	6	189	0.634	0.957	1.591	0.421
MOCK	6	122	0.642	0.942	1.584	0.451
MOCK	6	123	0.642	0.942	1.584	0.451
MOCK	6	118	0.642	0.940	1.582	0.451
MOCK	6	118	0.642	0.940	1.582	0.451

This vector model had 979 dimensions of which 977 were binary. Table 13 shows that AMMOC and MOCK occupy four of the five top spots with AMMOC having the two highest DISP+DIV values (1.715, 1.64). The table also shows that the kmeans algorithm with random centroids did much better on this vector model placing just below AMMOC with the third best DISP+DIV (1.613). It did extremely well at not mixing concerns within clusters but was not that well at not spreading concern methods across clusters. This would seem to imply that, with 622 clusters, a researcher would have to go through a large number of clusters especially since the DIV value (0.987) was so high

and the DISP (0.626) so low. However, the algorithm's DIV2 value was much lower indicating that a large percentage of clusters had only non-crosscutting concerns. This conclusion was supported when the cluster distribution was analyzed.

Consistent behavior (21) 20 164 180 323 368 379 386 573 612

Decorator (6) 152 162 175 275 484

Composite (12) 162 329 354 556 575

Observer (10) 16 527 556 587

Adapter (1) 527

Command (2) 573

Contract enforcement (3) 573

Persistence (6) 422 515

Undo (3) 573

Exception handling (1) 303

Overall Vector Model Analysis

Vector model 3 had the best results overall and that was with AMMOC. The DISP+DIV for this model and method was 1.978 with a perfect DISP of 1 and a very high DIV of 0.978. AMMOC created 42 clusters with the data for this model but all crosscutting concerns ended up in cluster 1 along with some non-crosscutting concerns. This resulted in DIV2 having a much lower of 0.545. Table 14 shows the top 10 results which, as can be seen, only included models 2, 3, and 4. Vector model 5 occurred in position 16 of the table. Models 1 and 6 appeared much further down.

Interestingly, when the data were sorted by DISP+DIV2, AMMOC held the top ten spots covering all vector models as shown in Table 15. Another interesting fact seen in the table is that vector model 5 edged out vector model 3 for top spot. Then again,

since vector model 3 had many more clusters than vector model 5, that result is understandable when one considers that vector model 3 had a higher number of clusters containing only non-crosscutting concerns.

Table 14 - Top 10 Vector Model Results Based on DISP+DIV

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2
AMMOC	3	42	1.000	0.978	1.978	0.545
AMMOC	3	44	1.000	0.968	1.968	0.533
AMMOC	3	119	0.900	0.980	1.880	0.520
MOCK	2	6	1.000	0.848	1.848	0.545
MOCK	2	6	1.000	0.848	1.848	0.545
MOCK	2	5	1.000	0.818	1.818	0.545
MOCK	2	5	1.000	0.818	1.818	0.545
MOCK	2	5	1.000	0.818	1.818	0.545
MOCK	4	5	1.000	0.818	1.818	0.545
AMMOC	4	10	0.950	0.859	1.809	0.530

Table 15- Top 10 Vector Model Results Based on DISP+DIV2

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2	DISP+DIV2
AMMOC	5	9	0.900	0.843	1.743	0.648	1.548
AMMOC	3	42	1.000	0.978	1.978	0.545	1.545
AMMOC	2	3	1.000	0.697	1.697	0.545	1.545
AMMOC	5	2	1.000	0.545	1.545	0.545	1.545
AMMOC	5	2	1.000	0.545	1.545	0.545	1.545
AMMOC	2	2	1.000	0.545	1.545	0.545	1.545
AMMOC	1	2	1.000	0.545	1.545	0.545	1.545
AMMOC	6	2	1.000	0.545	1.545	0.545	1.545
AMMOC	6	2	1.000	0.545	1.545	0.545	1.545
AMMOC	4	2	1.000	0.545	1.545	0.545	1.545

Overall Clustering Analysis

Table 14 also shows that AMMOC and MOCK significantly outperformed the other methods based on DISP+DIV with AMMOC being the clear winner on the data from vector model 3. The aspect mining analysis for this cluster configuration had a DISP of 1, a DIV of 0.978, and a DISP+DIV of 1.978. The first occurrence of one of the

heuristic methods was the kmeans method with random centroids which was much further down in the table. This method created 622 clusters from the data for vector model 6 with a DISP of 0.626, a DIV of 0.987, and a DISP+DIV of 1.613. The kmeans that used centroids from the heuristic followed it in the table. It had a DISP of 0.692, a DIV of 0.921, and a DISP+DIV of 1.613 for vector model 3 with 87 clusters. The agnes clustering method turned up even further down with a DISP of 0.675, a DIV of 0.915, and a DISP+DIV of 1.59 for vector model 3 with 87 clusters.

This researcher decided to see if MOCK and AMMOC could be used to suggest cluster numbers for the heuristic algorithms in order to determine if that would lead to better results for those algorithms[7]. It turned out that the strategy worked for the most part. Table 16 shows that the results of running kmeans and agnes using cluster numbers obtained from running MOCK and AMMOC dominated the results from those algorithms that didn't use those cluster numbers. As the table shows, MOCK's suggested cluster numbers proved to be better, in general, at providing kmeans and agnes with "optimal" cluster numbers. Even so, MOCK and AMMOC gave better results than those algorithms for all but vector model 6. When the data for model 6 were compared AMMOC took the top two positions but MOCK was pushed down to about eleventh place in the table by various runs of kmeans with and without suggested cluster numbers from MOCK and AMMOC. (Note that for all of the heuristic methods to have been run with suggested cluster numbers meant that the centroids would have had to be randomly generated.)

---

[7] This strategy is not novel to data mining as other researchers have used genetic algorithms to find cluster centroids with which to start k-means (Jain, A. K., Murty, M. N., & Flynn, P. J., 1999).

Table 16 - Results of using AMMOC and MOCK to Suggest Cluster Numbers for the Heuristic Algorithms (AMMOC and MOCK results are not included in this comparison.)

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2
KmeansMOCK	3	10	0.950	0.859	1.809	0.530
AgnesMOCK	3	8	0.950	0.824	1.774	0.530
KmeansMOCK	3	12	0.900	0.841	1.741	0.523
KmeansMOCK	3	13	0.883	0.815	1.698	0.518
KmeansMOCK	3	15	0.850	0.807	1.657	0.517
AgnesMOCK	3	13	0.800	0.831	1.631	0.450
KmeansAMMOC	5	96	0.675	0.947	1.622	0.433
KmeansMOCK	6	111	0.667	0.953	1.620	0.423
KmeansMOCK	4	108	0.662	0.956	1.617	0.403
AgnesMOCK	3	12	0.800	0.817	1.617	0.450
KmeansAMMOC	6	67	0.683	0.932	1.616	0.435
KmeansAMMOC	6	77	0.675	0.939	1.614	0.417
kmeansCentroids	3	87	0.692	0.921	1.613	0.507
kmeansRandom	6	622	0.626	0.987	1.613	0.675
AgnesMOCK	3	14	0.800	0.811	1.611	0.470
KmeansMOCK	6	117	0.662	0.946	1.607	0.422
KmeansAMMOC	6	46	0.692	0.913	1.604	0.425
KmeansMOCK	5	116	0.658	0.944	1.603	0.415
KmeansMOCK	6	123	0.645	0.956	1.601	0.399
AgnesAMMOC	3	119	0.667	0.934	1.600	0.506
KmeansAMMOC	6	44	0.692	0.909	1.600	0.425
agnes	3	4	0.950	0.648	1.598	0.530

Comparison of MOCK against AMMOC

AMMOC outperformed MOCK on vector models 1, 3, and 6 based on DISP+DIV scores. For model 1, AMMOC lead MOCK by 0.115. For model 3, the difference was 0.332. For model 6, the difference was 0.117. For models 2, 4, and 5 MOCK led AMMOC by 0.133, 0.009, and 0.030, respectively. One would have thought that AMMOC would always be better considering the fact that it was directly optimizing aspect mining metrics. However, these are genetic algorithms that depend on various probabilities for recombination and mutation so there is no guarantee that AMMOC, or MOCK for that matter, converged completely. This is compounded by the fact that there is also no guarantee that multiobjective algorithms will see the entire Pareto front.

Over all models, AMMOC had the edge for model 3 with a DISP+DIV of 1.978 which was obtained from a DISP of 1, and a DIV of 0.978. The nearest MOCK score was a DISP+DIV of 1.848 obtained for a DISP of 1 and a DIV of 0.878. Even so, both had a DIV2 of 0.545 which meant that they were handling diversity equally. AMMOC obtained the highest DIV2 of 0.838 but that was accompanied by a very low DISP of 0.456. MOCK's highest DIV2 was 0.545 with a corresponding DISP of 1 and DIV of 0.848. When DISP+DIV2 values were compared, AMMOC again bested MOCK with a value of 1.548 obtained from a DISP of 0.9 and a DIV2 of 0.648. MOCK's highest DISP+DIV2 was 1.545.

Analysis using Other Aspect Mining Metrics

Although most of the discussion has hinged around the metrics, DISP, DIV, and DIV2, this study did investigate the performance of the methods and models based on precision (PREC) and methods to analyze (MTA). PREC measures the ability of the methods to find all crosscutting concerns. The maximum value and goal of this metric is 1. That turned out to be the value for every method working on every model. MTA measured the number of vectors (data representations for each method in the software under investigation) that each clustering algorithm would have to see before finding all crosscutting concerns once the vectors were organized in some particular order. In this case, the clusters were sorted in descending order based on the total fan-in values (FIV) of each method that the vector represented. Table 17 shows the top ten results with the lowest MTAs over all clustering methods and vector models.

Table 17 - Top 10 MTA Values for all Clustering Methods and all Models

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2	DISP+DIV2	PREC	MTA
MOCK	5	66	0.642	0.894	1.536	0.415	1.057	1	1518
MOCK	5	69	0.631	0.889	1.520	0.408	1.039	1	1595
MOCK	5	92	0.633	0.918	1.551	0.421	1.054	1	1644
MOCK	4	118	0.638	0.939	1.577	0.449	1.087	1	1676
MOCK	4	118	0.638	0.939	1.577	0.449	1.087	1	1679
MOCK	4	112	0.648	0.942	1.590	0.459	1.107	1	1683
MOCK	4	117	0.638	0.939	1.577	0.449	1.087	1	1683
MOCK	5	48	0.662	0.868	1.530	0.422	1.084	1	1685
MOCK	4	112	0.648	0.942	1.590	0.459	1.107	1	1687
MOCK	4	113	0.648	0.943	1.591	0.459	1.107	1	1688

As the table shows, the lowest ten MTAs were all obtained by MOCK with the lowest being 1518 from MOCK's clustering of model 5 data. (The total number of vectors was 2381.) This clustering resulted in a DISP of 0.642, a DIV of 0.894, a DISP+DIV of 1.536, and a DIV2 of 0.415. This MTA value did not belong to the clustering with the highest DISP+DIV. That MTA value of 2277 belonged to a clustering with a DISP+DIV of 1.978 that occurred with AMMOC working on model 3 data. The DISP, DIV, and DIV2 for that clustering were 1, 0.978, and 0.545, respectively.

What the table does not show is how the other clustering methods fared. When MTA alone was considered when sorting values, AMMOC appeared much lower down in the table with an MTA of 2158. This value was obtained from AMMOC clustering model 4 data which resulted in a DISP of 0.598, DIV of 0.953, a DISP+DIV of 1.551, and a DIV2 of 0.554. As a matter of fact, many of the clustering partitions made by the heuristic algorithms did much better than AMMOC at keeping MTA low. The heuristic method, agnes, had the lowest MTA (1971) of all of the heuristic methods and that

occurred working on model 6 data. The DISP, DIV, DISP+DIV, and DIV2 for that clustering were 0.435, 0.963, 1.398, and 0.802, respectively.

As one can see, it is hard to make decisions based on MTA alone as the DISP tends to be relatively low. This researcher considers DISP, DIV, and DIV2 to be more important than MTA since they directly relate to the goal of aspect mining. Therefore, MTA should be used to select clustering results where those metrics have values that are close to each other.

Comparison with Other Studies

Any comparison of this researcher's findings with those of Cojocar and Czibula (2008) must be put in perspective since they conducted their experiments on version 5.2 of JHotDraw which has less classes and methods than version 5.4b1. Even so, the discrepancy between the DISP+DIV values for their two k-means algorithms[8] and this research's DISP+DIV values for MOCK and AMMOC is significant enough to imply that MOCK and AMMOC are the better algorithms. A similar conclusion is arrived at when MOCK and AMMOC are compared against HAC and HAM. Note that the researchers did not use a DIV2 metric so it is impossible to determine, from their results, how clusters with only non-crosscutting concerns contributed to the high DIV values. Those researchers also worked only with vector models 1 and 2. Another thing to consider is that the researchers used a value of 1 for the threshold required by the program that generated centroids to be used by k-means. However, when this researcher used 1 as the threshold, 649 centroids were generated just for vector model 2. This implied that the

[8] Based on the reference by Cojocar and Czibula (2008) to the paper by Serban and Moldovan (2006a) when discussing their KM algorithm, this researcher is convinced that their KM and kAM algorithms are the same.

higher-dimensional vector models would result in much higher numbers of centroids resulting in a lot of 1-element clusters. As stated earlier, this research used centroid data from thresholds 2 and 3.

For vector model 1 Cojocar and Czibula's experimental results (Table 18) showed a DISP of 0.42 for KM, 0.441 for kAM, 0.435 for HAC, 0.441 for HAM, and 0.424 for GAM. Among the DISP+DIV values for MOCK and AMMOC that surpass DISP+DIV values for Cojocar and Czibula, MOCK had its highest DISP of 0.9 and AMMOC had its highest DISP of 1.0 for vector model 1. For vector model 2 Cojocar and Czibula's experimental results showed a DISP of 0.424 for KM, 0.422 for kAM, 0.422 for HAC, 0.422 for HAM, and no result for GAM since that algorithm ran so slowly that the researchers couldn't get a result within a reasonable time. Again, among the higher DISP+DIV values for vector model 2, MOCK and AMMOC had their highest DISP of 1.0.

Table 18 - Values of the Quality Measures for JHotDraw 5.2 (Cojocar & Czibula, 2008)

Method	Vector Model	DISP	DIV	DISP+DIV
HAM	2	0.422	0.994	1.416
HAC	2	0.422	0.993	1.415
kAM	2	0.422	0.993	1.415
KM	1	0.420	0.919	1.399
KM	2	0.424	0.950	1.374
HAC	1	0.435	0.896	1.331
HAM	1	0.441	0.890	1.331
kAM	1	0.441	0.842	1.283
GAM	1	0.424	0.797	1.221

For vector model 1 Cojocar and Czibula's experimental results showed a DIV of 0.919 for KM, 0.842 for kAM, 0.896 for HAC, 0.89 for HAM, and 0.797 for GAM. For

that same vector model, and focusing only on higher DISP+DIV values for both MOCK and AMMOC, MOCK had its highest DIV value of 0.82 and AMMOC had its highest DIV of 0.82. For vector model 2 Cojocar and Czibula's experimental results showed a DIV of 0.95 for KM, 0.993 for kAM, 0.993 for HAC, 0.994 for HAM, and no result for GAM. For vector model 2, MOCK had its highest DIV of 0.93 and AMMOC had its highest DIV of 0.947. Although the DIV values for AMMOC and MOCK were not as good as the corresponding values for every algorithm except GAM, their highest DISP+DIV values exceeded the DISP+DIV values of those algorithms. MOCK's highest DISP+DIV value of 1.6 for vector model 1 was better than the highest DISP+DIV value of 1.399 among KM, kAM, HAC, HAM, and GAM. Likewise, AMMOC's DISP+DIV value of 1.715 was much better. For vector model 2, MOCK's DISP+DIV value of 1.848 was much better than the highest value of 1.416 from among KM, kAM, HAC, and HAM. As before, AMMOC's DISP+DIV value of 1.715 was also better. Once again, the lack of DIV2 values from Cojocar and Czibula makes it difficult to say just how good their algorithms were at really handling diversity. The results from the algorithms studied by those researchers occurred so far down in the table that displaying the table is not practical.

When MOCK and AMMOC's results for all six of the vector models were compared with Cojocar and Czibula's results for vector models 1 and 2 AMMOC and MOCK's DISP+DIV values of 1.978 and 1.848, respectively, far surpassed those algorithms' DISP+DIV values. The highest DISP+DIV value from the set of algorithms studied by Cojocar and Czibula was 1.416 and that was obtained by HAM from model 2 data. However, the DIV values for HAM (0.994), HAC (0.993), and kAM (0.993) were

all higher than AMMOC and MOCK's although AMMOC's value (0.980) was third in the list. Again, without knowing what the DIV2 values were, it is difficult to really assess the ability of those algorithms to handle diversity.

Interestingly, Cojocar and Czibula discounted the genetic algorithm (GAM) as being a viable algorithm for aspect mining as its results were inferior to the others. That algorithm attempted to optimize one objective function. Its performance was the worst among all of the algorithms and it took so long to execute that there was no recorded value when it was run on the model 2 data. The fact that MOCK and AMMOC, also genetic algorithms, performed so well implies that the multiobjective nature of those algorithms is the primary factor affecting the performance. This implication is also supported by comparison with the kmeans and agnes algorithms each of which target a different one of MOCK's individual objective functions.

Although this study did not set out to compare multiobjective genetic algorithms against the model-based ones used in Rand McFadden's (2011) study, that researcher's results for the heuristic algorithms provided another benchmark against which to compare the performance of MOCK and AMMOC. The results obtained by that researcher for vector models 1 and 2 with the heuristic algorithms are shown in Table 19 and the results over all models are shown in Table 20.

Table 19 - Values of the Quality Measures for JHotDraw 5.4 Over Models 1 and 2

(Rand McFadden, 2011)

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2	DISP+DIV2
kMeansRandomMcFadden	2	30	0.662	0.821	1.483	0.403	1.065
kMeansCentroidsMcFadden	1	21	0.667	0.676	1.343	0.477	1.144
agnesMcFadden	1	21	0.643	0.658	1.301	0.401	1.044
kMeansRandomMcFadden	1	21	0.639	0.589	1.228	0.424	1.063
kMeansCentroidsMcFadden	2	30	0.526	0.683	1.209	0.524	1.050
agnesMcFadden	2	30	0.517	0.689	1.206	0.508	1.025

Table 20 - Values of the Quality Measures for JHotDraw 5.4 Over All Models

(Rand McFadden, 2011)

Method	Vector Model	Number of Clusters	DISP	DIV	DISP+DIV	DIV2	DISP+DIV2
kMeansRandomMcFadden	5	475	0.639	0.981	1.620	0.560	1.199
kMeansRandomMcFadden	4	433	0.628	0.981	1.609	0.508	1.136
kMeansRandomMcFadden	6	529	0.544	0.984	1.528	0.563	1.107
agnesMcFadden	3	310	0.546	0.968	1.514	0.484	1.030
kMeansRandomMcFadden	2	30	0.662	0.821	1.483	0.403	1.065
kMeansCentroidsMcFadden	3	310	0.496	0.967	1.463	0.484	0.980
agnesMcFadden	4	433	0.458	0.980	1.438	0.668	1.126
kMeansCentroidsMcFadden	5	475	0.452	0.982	1.434	0.731	1.183
kMeansCentroidsMcFadden	4	433	0.453	0.979	1.432	0.662	1.115
agnesMcFadden	5	475	0.444	0.982	1.426	0.715	1.159
kMeansCentroidsMcFadden	6	530	0.435	0.986	1.421	0.773	1.208
agnesMcFadden	6	530	0.432	0.986	1.418	0.778	1.210
kMeansRandomMcFadden	3	310	0.444	0.958	1.402	0.444	0.888
kMeansCentroidsMcFadden	1	21	0.667	0.676	1.343	0.477	1.144
agnesMcFadden	1	21	0.643	0.658	1.301	0.401	1.044
kMeansRandomMcFadden	1	21	0.639	0.589	1.228	0.424	1.063
kMeansCentroidsMcFadden	2	30	0.526	0.683	1.209	0.524	1.050
agnesMcFadden	2	30	0.517	0.689	1.206	0.508	1.025

When MOCK and AMMOC's results were compared against the results of Rand McFadden the comparison yielded an outcome that was similar to the comparison against the results of Cojocar and Czibula. Just like the comparison with Cojocar and Czibula

this comparison with Rand McFadden's data must be put in perspective. The heuristic that Rand McFadden used to generate centroids, and hence cluster numbers, had 5 as its threshold. As explained earlier neither a threshold of 4 nor 5 could have been used by this study if vector model 3 was to be included.

For all vector models, MOCK and AMMOC dominated and, with the exception of the results from vector models 4 and 5, they did so by a considerable amount. Even when DIV2 was used as a deciding factor, the genetic algorithms gave much better results than those recorded by Rand McFadden for the heuristic methods for all models except model 4. For model 4, the DIV2 result from agnes as recorded by Rand McFadden was 0.668. This was accompanied by a DISP of 0.458, a DIV of 0.98, and a DISP+DIV of 1.438. For the same model, Rand McFadden's results for kmeans with predetermined centroids were a DISP of 0.453, a DIV of 0.979, a DISP+DIV of 1.432, and a DIV2 of 0.662. The DIV2 values for these two methods were the top two values. The third DIV2 value of 0.61 was obtained from AMMOC which also had a DISP of 0.628, a DIV of 0.924, and a DISP+DIV of 1.552. Considering that AMMOC gave a much better DISP and DISP+DIV than the other two methods and had a very good DIV even if it wasn't as high as the other two, the difference between 0.61 and 0.668 is not sufficient to discount AMMOC from being the best of the three. Unfortunately, although the top ten spots for this model, based on DISP+DIV, were only held by genetic algorithms, the highest DIV2 value among them was 0.572. Even so, the heuristic with the highest DISP+DIV value of 1.609 only had a DIV2 of 0.508.

Summary of Results

This study compared the clustering results from two heuristic clustering algorithms with the results from two multiobjective genetic clustering algorithms. It also compared the results from the two genetic algorithms with results obtained from previous studies. The study ran the algorithms on vectors created from six vector models and measured the quality of the clustering solutions with five aspect mining metrics. The objective was to show that the multiobjective genetic algorithms were a viable alternative to those heuristic algorithms commonly used in aspect mining.

The clustering results for all the algorithms were analyzed using the metrics PREC, DISP, DIV, DIV2, and MTA, as described in the aspect mining literature (Moldovan & Serban, 2006a, Rand McFadden & Mitropoulos, 2012; Rand McFadden & Mitropoulos, 2013a). Success was measured based on the ability of the clustering solutions to maximize DISP+DIV values. Judging from these values, the two multiobjective genetic algorithms, MOCK and AMMOC, yielded extremely good results from an aspect mining point of view and succeeded in dominating all of the heuristic algorithms used in this study. Several perfect DISP values of 1 were attained which all came from the multiobjective genetic algorithms. When all vector models and all algorithms were compared, AMMOC had the best results. When the comparison was made for individual models, at least one of the multiobjective genetic algorithms had better aspect mining results than their heuristic counterparts.

When the multiobjective genetic algorithms were compared against each other, AMMOC showed better performance on data from models 1, 3, and 6 while MOCK

performed better on data from the other three models. Over all models, AMMOC gave the best three results based on DISP+DIV and those all occurred with model 3 data.

Comparisons with results for methods obtained from previous studies in the literature had the same outcome. It should be noted that the comparisons between the two genetic algorithms and the algorithms studied by Cojocar and Czibula could only be done with vector models 1 and 2 since those researchers did not use the other vector models. Even so, none of the DISP+DIV results from the algorithms used in those researchers' experiments got anywhere close to the top DISP+DIV values achieved by MOCK and AMMOC even though the DIV values for those researchers' algorithms were the highest. Cojocar and Czibula did not use the DIV2 quality metric so it was impossible to judge how good their algorithms were at really optimizing diversity.

A few patterns emerged from analyzing the clustering results. First, for those results that contained DIV2 values, the top results had very high DISP and DIV values but the DIV values were found to be misleading when their corresponding DIV2 values were taken into consideration. For a large number of results a very high DIV2 was usually accompanied by a relatively low DISP. This applied to all the clustering methods. This meant that whenever the algorithms were good at keeping the number of concerns per cluster low, they were not that good at not dispersing concern methods across clusters. The DIV2 values did get as high as 0.838 (AMMOC on vector model 2 data) but the values higher than 0.65 were usually associated with average to low DISP values. A similar pattern occurred with MTA values. The lowest MTA recorded was 1518 (out of 2381 vectors) but that was associated with a DISP of 0.642, a DIV of 0.894, and a DIV2

of 0.415 for a 66-cluster solution. The lowest MTA value was obtained from MOCK on model 5 data.

Cluster numbers fluctuated wildly from as low as 2 to as high as 622. (No cluster numbers were reported by Cojocor and Czibula.) Although all algorithms produced clustering results with very low cluster numbers over all vector models, the highest number of clusters for the multiobjective algorithms was 143. However, a lower number of clusters did not necessarily mean a lower number of methods to analyze. For example, the three results that had 622 clusters had MTAs of 2008, 2010, and 2128 whereas results with 2 to 4 clusters had MTAs over 2300.

Overall, the results are very promising and it is fair to say that the multiobjective genetic algorithms are a viable alternative to the singleobjective heuristic algorithms.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions

This study investigated the viability of applying multiobjective genetic clustering algorithms to the problem of finding crosscutting concerns in legacy software. It did so by using two multiobjective genetic clustering algorithms, MOCK and AMMOC, to cluster sets of vectors attained from six vector models and then by comparing the clustering results against similar results derived from running two heuristic clustering algorithms on the same data. The heuristic algorithms were a version of the k-means algorithm and a version of a hierarchical agglomerative algorithm commonly used in aspect mining. One of the multiobjective algorithms, MOCK, was not adapted in any way for aspect mining but was used as is because the generic objective functions it attempted to optimize were similar to those used in aspect mining. The other algorithm, AMMOC, was a redesign of MOCK in particular, MOCK's clustering engine PESA-II. The redesigned algorithm specifically targeted aspect mining quality metrics as its objective functions since the goal of clustering in aspect mining is to generate cluster partitions that would optimize such metrics.

The multiobjective genetic clustering algorithms had the highest individual DISP (1), DISP+DIV (1.978), and DIV2 (0.838) values and the lowest MTA value (1518). (Note that these did not occur together.) Even though they did not have the highest DIV value, that DIV value of 0.994, belonging to the HAM algorithm from Cojocar and Czibula (2008), was accompanied by a DISP of 0.422 giving a DISP+DIV of 1.416.

Contrast that with the highest DISP+DIV value of 1.978, belonging to AMMOC, which had a DISP of 1 and a very good DIV of 0.978.

Handling diversity seems to have been the biggest problem for all of the algorithms run in this study based on their DIV2 values. (Again it should be remembered that there were no DIV2 values from Cojocar and Czibula so this comment does not apply to their results.) The best DIV2 value of 0.838 went along with a poor DISP of 0.456. So when diversity seemed to be contained, dispersion seemed to suffer. When one looks at the DISP+DIV2 scores, the multiobjective algorithms again dominated with 48 of the top values held by them. AMMOC came out ahead with a DISP of 0.9 and a DIV2 of 0.648 giving the highest DISP+DIV2 of 1.548. The nearest heuristic algorithm was a kmeans algorithm run with predetermined centroids. It had a DISP of 0.95, a DIV2 of 0.53, and a DISP+DIV2 of 1.48.

Although the multiobjective algorithms produced better results they did not solve the problem of finding a method that would create an ideal cluster partition from an aspect mining viewpoint. The results from analyzing MOCK and AMMOC's cluster data still had to be visually scanned in order to select the result considered to be the best for this researcher. MOCK did have the ability to suggest a "best" clustering result but the suggestions proved not to be ideal from the aspect mining point of view. That feature was not built into AMMOC because the number of clustering results that AMMOC produced was low enough to allow visual scanning of the accompanying aspect mining quality values and adding that feature meant that AMMOC would take much longer to execute than it did. (MOCK's run times for very high-dimensional vector data were very long.)

Implications

This dissertation exposed aspect mining researchers to an area of research that has been shown to be viable in data mining in general. Multiobjective genetic clustering algorithms have been used successfully in the data mining arena and this dissertation has shown that they can also be very successful in the aspect mining arena. It has also shown that designing multiobjective genetic clustering algorithms that attempt to optimize specific aspect mining metrics can lead to even better results.

Recommendations

Although AMMOC was the clear winner over all methods with all vector models, it is unclear why it was not the winner over MOCK for every vector model since its job was to optimize the aspect mining metrics directly. It could have been that MOCK had a better initial population to work with. (MOCK's initial population was generated by two algorithms, Prim's minimum spanning tree algorithm and a k-means algorithm. AMMOC's initial population was generated by Prim's algorithm alone.) But if that was the case, then one might have expected MOCK to win all of the time. It could also have been that AMMOC's objective functions were not as complementary as they should have been. That would have led AMMOC to favor particular population distributions. More than likely, the reason was that the probabilistic nature of genetic algorithms, compounded by the fact that multiobjective genetic algorithms may only approximate the true Pareto front, caused the algorithms to see different parts of the global solution space. What would also have contributed were the many parameters that had to be set with some of MOCK's parameters being inherent in the code and hence very hard to track down and

change. Regardless, the performance of both algorithms justifies their use in aspect mining.

Based on the overall better performance of AMMOC, this study recommends that more research be done in enhancing AMMOC's behavior. With respect to this, the challenges would be to do the following:

1. Find an ideal set of parameters.
2. Find a set of aspect mining objective functions that would allow a more thorough search of the solution space.
3. Consider increasing the number of objective functions being optimized simultaneously.

Since there are many more types of multiobjective algorithms used in data mining (Coello, 1999; Deb, 1999, 2001; Law et al., 2004; Maulik et al., 2011; Zhou et al., 2011), it would make sense to look at how they can be used in aspect mining. The latter is especially pertinent since genetic algorithms tend to take much longer to execute than non-genetic ones.

Summary

The purpose of this study was to determine whether multiobjective genetic clustering algorithms could perform satisfactorily in the aspect mining domain, one in which they have not been applied to the best of this researcher's knowledge. The reason for attempting this study was the need to find more ways of identifying aspect candidates in legacy code. Identifying such candidates would allow the modularization of such code,

via refactoring, to remove the scattering and tangling of crosscutting concerns. This in turn would lead to more understandable, manageable, and updateable code.

Unfortunately, there is no one algorithm that solves the problem of finding aspect candidates for all data distributions especially when using clustering techniques. One of the reasons for this is that all of the standard algorithms attempt to optimize a function that tends to target a particular underlying population distribution. This biases the clustering process and hence misses patterns that don't conform to the one targeted by the objective function. Also, many of the standard algorithms, like k-means and the hierarchical agglomerative algorithms, require that the number of clusters be known beforehand. This too adds bias to the process. Another problem that surfaces in some of these algorithms is that they tend to home in on local optima and don't see the entire solution space. There is therefore a need for algorithms that don't suffer from these drawbacks.

Genetic clustering algorithms solve two of these problems. For one thing, some genetic algorithms do not need to be supplied with the number of clusters since this is an automatic result of such algorithms. For another, genetic algorithms get a better view of the global solution space because of their ability to recombine and mutate solutions in their current population when generating potential solutions for their future populations. Note that, although they have a better chance of viewing the global solution space, they may still converge to local optima.

Unfortunately, many of these genetic algorithms try to optimize a single objective function as well. Therefore, they still suffer from being biased towards certain population

distributions. Multiobjective genetic clustering algorithms try to remove this bias by attempting to optimize multiple objective functions simultaneously. Since some of these multiobjective genetic functions have been shown to be successful in finding (near) optimal solutions in the data mining domain when they were compared against algorithms that are also used in aspect mining, there was reason to believe that they would be equally successful in the aspect mining domain. Therefore, this study conducted experiments that showed that two multiobjective algorithms performed better overall (based on DISP+DIV values) than a partitional algorithm and a hierarchical agglomerative algorithm.

The study used the k-means partitional algorithm, implemented as the kmeans function in the statistical program R. The agnes function in R was the hierarchical agglomerative algorithm. One set of runs involved giving kmeans a set of centroids as a parameter. This set of centroids was generated by a heuristic developed by Serban and Moldovan (2006a). Another set of runs used the number of centroids from the heuristic as a parameter but allowed kmeans to create its own centroids randomly. The same number of centroids was also used to determine where to cut the dendrogram created by agnes. The heuristic was one that was designed from an aspect mining viewpoint.

The multiobjective genetic clustering algorithms used were MOCK, obtained from its authors Handl and Knowles (2007b), and AMMOC, designed specifically for this study. These genetic algorithms attempted to optimize two objective functions simultaneously. While MOCK's objective functions (overall deviation and connectivity) were generic ones that behaved similarly to those found in the aspect mining literature,

AMMOC's two functions were the DISP and DIV clustering quality metrics from aspect mining.

The data to be clustered were first produced by the FINT tool working on the JHotDraw 5.4b1 program. FINT produced a set of methods along with their method callers and the number of those callers. Java programs, created by this researcher, processed that data in order to extract the different methods that were called, their method callers, and all of the classes that each method belonged to. More Java programs were written designed to make sets of vectors based on the following vector models already described in this study: fanIn_NumCallers, fanIn_hasMethod, sigTokens, fanIn_sigTokens, fanIn_hasMethod_sigTokens, fanIn_numCallers_hasMethod_sigTokens. The first two models came from Moldovan and Serban (2006b). The third model was introduced by Zhang et al. (2008) and the following three models were created by Rand McFadden (2011). The actual clustering was then done by running the kmeans and agnes functions in R and by running MOCK and AMMOC on the vector model data. The quality of the clusterings was determined using the previously defined aspect mining metrics DISP, DIV, PREC, DIV2, and MTA. The metrics DISP, DIV, and PREC were taken from research by Moldovan and Serban (2006a) whereas DIV2 and MTA were obtained from Rand McFadden (2011) and Rand McFadden and Mitropoulos (2012, 2013a).

Data from the clustering quality analysis of the various runs were compared against each other and against other data from sources in the literature, notably from Cojocar and Czibula (2008) and Rand McFadden (2011). The data from Cojocar and Czibula were results of running their versions of k-means (KM and kAM), their versions

of hierarchical agglomerative algorithms (HAC and HAM), and their genetic algorithm GAM. The data from Rand McFadden were based on that researcher's runs of the R functions kmeans and agnes on data similar to the data from this study. Although the main goal of Rand McFadden's experiments was to show that model-based algorithms were as good as, if not better than, their heuristic counterparts when used in aspect mining, the data recorded from that study's analysis of the kmeans and agnes results were used as another benchmark for comparison.

The results of this study showed that the multiobjective clustering algorithms produced superior results based on their DISP+DIV values. AMMOC was the best performer over all vector models giving its best result for vector model 3. However, when each vector model was considered separately, AMMOC was not always the best but MOCK was. The multiobjective algorithms did not have the highest DIV values although many of those values were extremely high. (The highest DIV values were recorded by the HAC, HAM, and kAM methods of Cojocar and Czibula.) However, high DIV values did not necessarily imply good diversity since the corresponding DIV2 values were significantly lower. The DIV2 values indicated that the high DIVs were as a result of the contribution of many clusters with non-crosscutting concerns. Unfortunately, the methods of Cojocar and Czibula had no DIV2 values so it was impossible to determine how good their methods were at achieving optimal diversity.

The multiobjective algorithms used in this study did not solve the problem of mining legacy data for aspect candidates but this study did introduce aspect mining researchers to a promising area of research. There are many directions that future research could take such as, looking at different types of multiobjective algorithms, or

finding better combinations of objective functions suitable for aspect mining, or increasing the number of objective functions to be optimized. Regardless of which direction the research takes, this researcher is certain that such research will move aspect mining closer to its goal.

Appendices

Appendix A

Multiobjective Optimization and Pareto Optimality

The following was obtained from Coello (1999) and from Maulik, Bandyopadhyay, and Mukhopadhyay (2011).

Multiobjective Optimization

Find the vector $\bar{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]^T$ of decision variables that satisfies the *m* inequality constraints

$$g_i(\bar{x}) \geq 0, \quad i = 1, 2, \ldots, m$$

and the *p* equality constraints

$$h_i(\bar{x}) = 0, \quad i = 1, 2, \ldots, p$$

and optimizes the vector function

$$\bar{f}(\bar{x}) = [f_1(\bar{x}), f_2(\bar{x}), \ldots, f_k(\bar{x})]^T$$

where each $f_i(\bar{x}), \ i = 1, \ldots, k$, is an objective function.

The constraints define the set of feasible solutions from which the optimal solution will be chosen. The problem is that there is no clear definition of optimality in multiobjective optimization. There may be many solutions that optimize one or more of

the objective functions but it is rare that one solution optimizes all of them. Most solutions represent some tradeoff as far as the optimization of the objective functions.

One of the theoretical tools for determining optimality in the multiobjective context is Pareto optimality. Without loss of generality, Pareto optimality will be defined for the problem of minimizing some given vector of objective functions.

A decision vector, $\bar{x}^*$, is said to be Pareto optimal if there does not exist another decision variable, $\bar{x}$, that dominates it. That is, there is no $\bar{x}$ such that

$$f_i(\bar{x}) \leq f_i(\bar{x}^*) \ \forall i \in \{1, 2, \dots, k\}$$

and

$$f_i(\bar{x}) < f_i(\bar{x}^*) \ for\ some\ i \in \{1, 2, \dots, k\}.$$

Note that Pareto optimality partially orders the set of feasible solutions as there may be pairs of solutions that do not dominate each other. The global set of non-dominated solutions forms the Pareto optimal set and its image in objective space is referred to as the Pareto front. (See Figure 1 for an example of a Pareto front.)

A multiobjective optimization algorithm strives to achieve the following:

1. The set of Pareto optimal solutions is a subset of the true Pareto optimal set.
2. The set of Pareto optimal solutions represents a uniform and diverse distribution of solutions across the Pareto front.
3. The set of Pareto optimal solutions is spread across the entire spectrum of the Pareto front.

Appendix B

Default Settings for MOCK

The following are the parameter settings for MOCK recommended by Handl and Knowles (2007a). In the table, *N* refers to the number of data items to be clustered.

Table 21 - Default Settings for MOCK

Parameter	Setting
Number of generations	1000
External population size	1000
Internal population size	10
Resolution of hypergrid per dimension	10
Maximum number of clusters, k_{user}	25 or 50
# of initial solutions, f_{size}	2 x k_{user}
Initialization	Minimum spanning tree (L = 10) and k-means
Mutation type	L nearest neighbors (L = 10)
Mutation rate, p_m	$p_m = \frac{1}{N} + \left(\frac{l}{N}\right)^2 \; where \; l \in \{1, \dots, L\}$
Recombination	Uniform crossover
Recombination rate, p_c	0.7
Objective functions	Overall deviation and connectivity (L = 10)

# of reference distributions	3

Handl and Knowles pointed out that most of those parameters were required by MOCK's PESA-II clustering engine and that those settings were the ones commonly used in the literature. They did state that increasing the IP and the number of iterations *may* improve the algorithm's accuracy. They also stated that changing the number of control distributions (fronts) required a consideration of the tradeoff between accuracy and computational cost, the latter increasing significantly with a higher number of control fronts. They remarked that k_{user} could be interpreted as an upper bound on the number of clusters expected in the data set and that its setting was not crucial. Hence, very large values could be used. They also mentioned that the choice for L affected the sensitivity of the algorithm towards small clusters. They advocated a relativity large L to prevent outliers from ending up in their own clusters but warned that too large a value for L could result in clusters being overlooked. Therefore, they suggested values in the range 5 to 20.

Appendix C

Aspect Mining using Multiobjective Clustering (AMMOC)

AMMOC is a modified version of MOCK's PESA-II engine developed by David Corne (Corne, Jerram, Knowles, & Oates, 2001). (The code was not directly attributed to Corne in the literature but was obtained from comments in the source code for MOCK, in particular, the program pesa2Clust.c.) AMMOC was implemented in Java based on the algorithms found in the paper "Multiobjective clustering with automatic determination of the number of clusters" (Handl & Knowles, 2004) as well as from the source code available on the Web (Handl & Knowles, 2007b). Like MOCK, AMMOC is a genetic algorithm designed to optimize multiple objectives simultaneously and produce a set of (near) optimal solutions. Unlike MOCK, AMMOC does not implement the part that automatically determines the number of clusters. It also does not provide a graphical user interface. The decision to exclude the automatic determination of the number of clusters was based on the fact that finding the optimal number of clusters required comparing every member of the solution against every member of a set of reference fronts. From experience running MOCK with different numbers of reference fronts it was determined that, since AMMOC was already producing a relatively small number of solutions, the addition of that component was not worth the large increase in computation time.

AMMOC uses the same elitist strategy as PESA-II by keeping track of an internal and external population. The internal population is of fixed size. The external population's size can fluctuate but is bounded. The individuals in the internal population

undergo mutation and recombination and consist of clustering solutions that explore the global solution space. The individuals in the external population are selected from the internal population and are chosen if no other individual in the internal population dominates them. (See Appendix A for the definition of 'dominate' in multiobjective optimization.) The external population consists of 'niches' (Handl & Knowles, 2004) that are implemented as a hypergrid in objective space. Niching is used to help spread solutions across the entire objective space by only allowing a solution to enter a full external population if it occupies a less crowded niche. The internal population is replenished by randomly selecting individuals uniformly from occupied niches in the external population. However, AMMOC, unlike MOCK's PESA-II algorithm, recombines and mutates individuals as they are selected from the external population as opposed to putting them into the internal population first and then applying the operators on them. It also uses a simplified calculation for the probability of mutation; the probability of mutation is simply equal to the reciprocal of the number of elements in the data set. The latter is actually derived from settings given in Table 1 of the 2004 paper by Handl & Knowles which apparently applied to an older version of MOCK.

AMMOC also uses the same locus-based adjacency representation for each clustering solution. This is a graph-based solution where each individual clustering solution, g, is made up of N genes, N being the number of vectors in the data set. Each gene can take an allele value, j, also from 1 to N. Therefore, if $gene_i$ has an allele value of j, this indicates that $gene_i$ is connected to $gene_j$ so they will end up in the same cluster. As in MOCK, the initial internal population is generated by first executing Prim's Minimum Spanning Tree algorithm on the set of vectors. This produces the first individual to be

added to the internal population. Each subsequent member, k, is generated from the first individual by removing the (k-1) longest links.

It should be noted that the algorithm for the PESA-II engine came from an earlier version of MOCK since it was the only one available and the actual updated code was a bit difficult to decipher. The earlier version only used Prim's algorithm to generate all the initial individuals for the internal population. The newer version used Prim's for half of the population and k-means for the other half. Since AMMOC was developed incrementally to make testing easier and it gave good results when Prim's was used for the entire initial population, this researcher felt that the extra time taken to include k-means and carry out more tests was not worth it at that time.

Hence, the main differences between the implementation of the PESA-II engine in the current version of MOCK and the same engine in AMMOC are:

- MOCK optimizes two objective functions, overall deviation and connectivity (see chapter 2 of this document) whereas AMMOC optimizes the aspect mining functions DISP and DIV.
- MOCK uses Prim's algorithm and k-means to initialize the internal population whereas AMMOC only uses Prim's algorithm.
- AMMOC does not implement automatic K determination.
- AMMOC, unlike MOCK's PESA-II algorithm, recombines and mutates individuals as they are selected from the external population as opposed to putting them into the internal population first and then applying the operators on them.

- In AMMOC, the probability of mutation is simply 1/N where N = number of elements in the data set.

It should also be noted that AMMOC does not provide a graphical user interface as MOCK does.

Appendix D

AMMOC's Adaptation of the PESA-II Engine

```
public void pesa2AMMOC(double probMutation, double probCrossover,
                        int numberOfGenerations)
{
                initializePopulations();  //put individuals into internal population
                                // and configure external population
                generateInitialChromosomesFromMST();
                externalPopulation.setOptimization(optimization);  //false is maximization

                for(int i = 0; i < internalPopulationSize; i++)
                {
                        Chromosome chromosome = internalPopulation.get(i);
                        int[] clusterAssignments = chromosome.getClusterAssignment();
                        functions.setClusterAssignments(clusterAssignments);
                        chromosome.setObjectiveValue(0, functions.function1());
                        chromosome.setObjectiveValue(1, functions.function2());
                        externalPopulation.updateExternalPopulation(chromosome);
                }

                for(int gen = 1; gen <= numberOfGenerations; gen++)
                {
                //Clear out internal population so that you can fill it from
                external population
                        internalPopulation.clear();
                        //Pull them all out first and work on them
                        for(int i = 0; i < internalPopulationSize; i++)
                        {
                        Chromosome chromosome1 =
                        externalPopulation.getRandomChromosomeFromPopulation();
                                if(rand.nextDouble() < probCrossover)
                                {
                            Chromosome chromosome2 =
                            externalPopulation.getRandomChromosomeFromPopulation();
                             chromosome1 =
                             crossover(chromosome1, chromosome2, probCrossover);
                                }
```

```
                    chromosome1 = mutate(chromosome1, probMutation);
                    internalPopulation.add(chromosome1);
                }

                //Now put them back in
                for(int i = 0; i < internalPopulationSize; i++)
                {
                    Chromosome chromosome = internalPopulation.get(i);
                    int[] clusterAssignments =
                        chromosome.getClusterAssignment();
                    functions.setClusterAssignments(clusterAssignments);
                    chromosome.setObjectiveValue(0, functions.function1());
                    chromosome.setObjectiveValue(1, functions.function2());
                    externalPopulation.updateExternalPopulation(chromosome);
                }

            }//end generations
}  //end pesa2AMMOC
```

Appendix E

Non-AMMOC Programs

Java classes were written to do the following:

1. Generate methods and classes from the results of the FINT analysis.
2. Generate vectors for each vector model.
3. Generate centroids, and hence cluster numbers, to be used by the k-means and hierarchical agglomerative algorithms.
4. Prepare MOCK clustering results for aspect mining analysis.
5. Prepare AMMOC clustering results for aspect mining analysis.
6. Generate individual aspect mining quality results along with cluster distributions.
7. Generate collective aspect mining quality results in table form.

References

Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 1027-1035.

Bandyopadhyay, S., & Saha, S. (2012). *Unsupervised classification: Similarity measures, classical and metaheuristic approaches, and applications*. Berlin Heidelberg: Springer-Verlag.

Branke, J., Deb, K., Dierolf, H., & Osswald, M. (2004). Finding knees in multi-objective optimization. In *Parallel Problem Solving from Nature-PPSN VIII*, 722-731.

Ceccato, M., Marin, M., Mens, K., Moonen, L., Tonella, P., & Tourwe, T. (2005). A qualitative comparison of three aspect mining techniques. In *Proceedings of the 13th International Workshop on Program Comprehension, 2005*, 13-22.

Coello, C. A. C. (1999). A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information systems*, *1*(3), 269-308.

Cojocar, G. S., & Czibula, G. (2008). On clustering based aspect mining. *The 4th International Conference on Intelligent Computer Communication and Processing*, 129−136. doi:10.1109/ICCP.2008.4648364

Cojocar, G. S., Czibula, G. & Czibula, I. G. (2009). A comparative analysis of clustering algorithms in aspect mining. In *Informatica*, *4*(1), 75-84.

Corne, D. W., Jerram, N. R., Knowles, J. D., & Oates, M. J. (2001). PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 1-8.

Czibula, G., Cojocar, G. S., & Czibula, I. G. (2011). Evaluation measures for partitioning based aspect mining techniques. *International Journal of Computers Communications & Control*, *6*(1), 72-80.

Deb, K. (1999). Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, 7(3), 205-230.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. U.K.: John Wiley & Sons.

Elrad, T., Aksit, M., Kiczales, G., Lieberherr, K., & Ossher, H. (2001). Discussing aspects of AOP. *Communications of the ACM, 44*(10), 33-38.

Fillus, E. K., & Vergilio, S. R. (2012). A clustering based approach for aspect mining and pointcut identification. In *Proceedings of the 6th Latin American Workshop on Aspect-Oriented Software Development: Advanced Modularization Techniques*, 1-6.

Fraley, C., & Raftery, A. E. (1998). How many clusters? Which clustering method? Answers via model-based cluster analysis. *The Computer Journal*, *41*(8), 578-588.

Freitas, A. A. (2008). A review of evolutionary algorithms for data mining. In *Soft Computing for Knowledge Discovery and Data Mining*, 79-111.

Gamma, E. & Eggenschwiler, T. JHotDraw [Software]. Available from http://sourceforge.net/projects/jhotdraw/?source=dlp

Han, J., Kamber, M., & Pei, J. (2011). *Data mining: concepts and techniques*. Morgan Kaufmann.

Handl, J., & Knowles, J. (2004). Multiobjective clustering with automatic determination of the number of clusters. *UMIST, Manchester, Tech. Rep. TR-COMPSYSBIO-2004-02*.

Handl, J., & Knowles, J. (2005a). Improvements to the scalability of multiobjective clustering. In *2005 IEEE Congress on Evolutionary Computation*, *3*, 2372-2379.

Handl, J. & Knowles, J. (2005b). Multiobjective clustering around medoids. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, *1*, 632-639.

Handl, J., & Knowles, J. (2005c). Exploiting the trade-off—the benefits of multiple objectives in data clustering. In *Evolutionary Multi-Criterion Optimization*, 547-560. Springer Berlin Heidelberg.

Handl, J. & Knowles, J. (2007a). An evolutionary approach to multiobjective clustering. In *IEEE Transactions on Evolutionary Computation, 11*(1), 56-76.

Handl, J. & Knowles, J. (2007b). MOCK [Software]. Available from http://personalpages.manchester.ac.uk/mbs/julia.handl/mock.html

Handl, J., & Knowles, J. (2012). Clustering criteria in multiobjective data clustering. In *Parallel Problem Solving from Nature-PPSN XII*, 32-41. Springer Berlin Heidelberg.

Hruschka, E. R., Campello, R. J., Freitas, A. A., & De Carvalho, A. P. L. F. (2009). A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 39*(2), 133-155.

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters, 31*(8), 651-666.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys (CSUR), 31*(3), 264-323.

Kellens, A., & Mens, K. (2005). A survey of aspect mining tools and techniques. *AspectLab Project IWT, 40116*, 1-20.

Kellens, A., Mens, K., & Tonella, P. (2007). A survey of automated code-level aspect mining techniques. In *Transactions on Aspect-Oriented Software Development IV, LNCS 4640*, 143-162.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J, & Irwin, J. (1997). Aspect-oriented programming. In *Proceedings of European Conference on Object-Oriented Programming*, 220−242.

Kleinberg, J. (2002). An impossibility theorem for clustering. In *NIPS 15*, 463-470.

Krishna, K., & Narasimha Murty, M. (1999). Genetic K-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 29*(3), 433-439.

Laffra, C. Dijkstra's Shortest Path Algorithm [Software]. Available from http://www.dgp.toronto.edu/people/JamesStewart/270/9798s/Laffra/GraphAlgorithm.java

Law, M. H., Topchy, A. P., & Jain, A. K. (2004). Multiobjective data clustering. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2*, 424-430.

Marin, M., Moonen, L., & van Deursen, A. (2006). FINT: Tool support for aspect mining. In *Proceedings of the 13th Working Conference on Reverse Engineering*, 299-300.

Marin, M., van Deursen, A., & Moonen, L. (2004). Identifying aspects using fan-in analysis. In *Proceedings of the 11th Working Conference on Reverse Engineering*, 132-141.

Maulik, U. & Bandyopadhyay, S. (2000). Genetic algorithm-based clustering technique. *Pattern Recognition, 33*(9), 1455-1465.

Maulik, U., Bandyopadhyay, S., & Mukhopadhyay, A. (2011). *Multiobjective Genetic Algorithms for Clustering: Applications in Data Mining and Bioinformatics*. Springer.

McNicholas, P. D. (2011). On model-based clustering, classification, and discriminant analysis. *Journal of Iranian Statistical Society, 10*(2), 181-190.

Mens, K., Kellens, A., & Krinke, J. (2008). Pitfalls in aspect mining. In *Proceedings of the 15th Working Conference on Reverse Engineering*, 113-122.

Moldovan, G. S., & Serban, G. (2006a). A formal model for clustering based aspect mining. In *Proceedings of the 8th WSEAS International Conference on Mathematical Methods and Computational Techniques in Electrical Engineering*, 70-75.

Moldovan, G. S., & Serban, G. (2006b). Aspect mining using a vector-space model based clustering approach. In *Proceedings of Linking Aspect Technology and Evolution (LATE) Workshop*, 36-40.

Moldovan, G. S., & Serban, G. (2006c). A study on distance metrics for partitioning based aspect mining. *Informatica, LI*, 53−60.

Mukhopadhyay, A., Maulik, U., & Bandyopadhyay, S. (2015). A survey of multiobjective evolutionary clustering. *ACM Computing Surveys, 47*(4), 1-46.

Naldi, M. C., de Carvalho, A. C., & Campell, R. J. G. B. (2008). Genetic clustering for data mining. In *Soft Computing for Knowledge Discovery and Data Mining*, 113-132.

Nora, B., Said, G., & Fadila, A. (2006). A comparative classification of aspect mining approaches. *Journal of Computer Science, 2*(4), 322-325. doi:10.3844/jcssp.2006.322.325

Rand McFadden, R. (2011). *Aspect mining using model-based clustering*. (Unpublished doctoral dissertation), Nova Southeastern University, Ft. Lauderdale, FL, USA.

Rand McFadden, R., & Mitropoulos, F. J. (2012). Aspect mining using model-based clustering. In *Proceedings of IEEE SoutheastCon*, 1-8.

Rand McFadden, R., & Mitropoulos, F. J. (2013a). Survey and analysis of quality measures used in aspect mining. In *Proceedings of IEEE Southeastcon 2013*, 1-8.

Rand McFadden, R., & Mitropoulos, F. J. (2013b). Survey of aspect mining case study software and benchmarks. In *Proceedings of IEEE SoutheastCon 2013*.

Roy, C. K., Uddin, M. G., Roy, B., & Dean, T. R. (2007). Evaluating aspect mining techniques: A case study. *Proceedings of 15th IEEE International Conference on Program Comprehension (ICPC '07)*, 167−176. doi:10.1109/ICPC.2007.21

Serban, G., & Moldovan, G. S. (2006a). A new k-means based clustering algorithm in aspect mining. In *Proceedings of the Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 69-74.

Serban, G., & Moldovan, G. S. (2006b). A comparison of clustering techniques in aspect mining. *Informatica, 51*(1).

Serban, G., & Moldovan, G. S. (2006c). A new genetic clustering based approach in aspect mining. In *Proceedings of the 8th WSEAS International Conference on Mathematical Methods and Computational Techniques in Electrical Engineering*, 135-140.

Software Engineering Research Group (2008). FINT [Software]. Available from http://swerl.tudelft.nl/bin/view/AMR/FINT

Srinivas, M., & Patnaik, L. M. (1994). Genetic algorithms: A survey. *Computer*, *27*(6), 17-26.

Tibshirani, R., Walther, G., & Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *63*(2), 411-423.

Tribbey, M. & Mitropoulos, F. (2012). Construction and analysis of vector space models for use in aspect mining. In *Proceedings of the 50th Annual Southeast Regional Conference*, 220-225.

Tsai, C. W., Chen, W. L., & Chiang, M. C. (2012). A modified multiobjective EA-based clustering algorithm with automatic determination of the number of clusters. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2833-2838.

Van Deursen, A., Marin, M., & Moonen, L. (2003). Aspect mining and refactoring. In *Proceedings of the 1st International Workshop on Refactoring: Achievements, Challenges, Effects (REFACE), with WCRE*, 11-21.

Zhang, D., Guo, Y., & Chen, X. (2008). Automated aspect recommendation through clustering-based fan-in analysis. In *Proceedings of the International Conference on Automated Software Engineering*, 278-287.

Zhou, A., Qu, B. Y., Li, H., Zhao, S. Z., Suganthan, P. N., & Zhang, Q. (2011). Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation, 1*(1), 32-49.