2010-10-18

Assessment of aCGH Clustering Methodologies

Serena F. Baker
Brigham Young University - Provo

Assessment of aCGH Clustering Methodologies

Serena F. Baker

A project submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

David A. Engler, Chair
C. Shane Reese
Natalie J. Blades

Department of Statistics

Brigham Young University

December 2010

ABSTRACT

Assessment of aCGH Clustering Methodologies

Serena F. Baker

Department of Statistics

Master of Science

Array comparative genomic hybridization (aCGH) is a technique for identifying duplications and deletions of DNA at specific locations across a genome. Potential objectives of aCGH analysis are the identification of (1) altered regions for a given subject, (2) altered regions across a set of individuals, and (3) clinically relevant clusters of hybridizations. aCGH analysis can be particularly useful when it identifies previously unknown clusters with clinical relevance. This project focuses on the assessment of existing aCGH clustering methodologies. Three methodologies are considered: hierarchical clustering, weighted clustering of called aCGH data, and clustering based on probabilistic recurrent regions of alteration within subsets of individuals.

Assessment is conducted first through the analysis of aCGH data obtained from patients with ovarian cancer and then through simulations. Performance assessment for the data analysis is based on cluster assignment correlation with clinical outcomes (e.g., survival). For each method, 1,000 simulations are summarized with Cohen's kappa coefficient, interpreted as the proportion of correct cluster assignments beyond random chance. Both the data analysis and the simulation results suggest that hierarchical clustering tends to find more clinically relevant clusters when compared to the other methods. Additionally, these clusters are composed of more patients who belong in the clusters to which they are assigned.

Keywords: array CGH, hierarchical clustering, WECCA

ACKNOWLEDGMENTS

I would like to thank everyone who has helped me complete this project. Special thanks to Dr. David Engler for providing the R code to generate aCGH data, as well as the code used to format the data and find regions for the WECCA method. His assistance throughout this project was invaluable, in the way of direction, encouragement, and expertise. I am also indebted to the Statistics department faculty for their interest in my success and for all the statistical training and experience that I have received. Thanks, also, to my fellow students for their advice, support, and willingness to listen to my thinking out loud. And to my mom, for her continual thoughts and prayers.

CONTENTS

Contents iv

List of Tables vi

List of Figures vii

1 Introduction 1

2 Literature Review 3

2.1 Identifying Copy Number Aberration for Single Hybridizations 3

2.2 Identifying Copy Number Aberration Across Multiple Hybridizations 5

2.3 Clustering Hybridizations 8

3 Methods 14

3.1 Modifications to the Rueda/Diaz-Uriarte Method 15

4 Data Analysis 16

4.1 Hierarchical Clustering 16

4.2 WECCA 20

4.3 Discussion 21

5 Simulations 24

5.1 Generating aCGH Data 24

5.2 Simulation Set-Up 26

5.3 Results 27

6 Conclusions 30

Bibliography 32

Appendices 37

Appendix A: Source Code 38

A.1 CGH Regions 38

A.2 WECCA Functions 45

A.3 Willenbrock Data Generation 53

Appendix B: Script Code 59

B.1 CGH Regions Script 59

B.2 WECCA Script 62

Appendix C: Data Analysis Code 67

C.1 Data 67

C.2 Hierarchical Clustering 68

C.3 WECCA 70

C.4 Comparison of HC and WECCA 76

Appendix D: Simulation Code 78

D.1 Empirical Data 78

D.2 Simulations 80

LIST OF TABLES

4.1 Patient counts by stage number (2, 3, 4) and cluster (1, 2) with marginal totals. In parentheses are cell proportions out of 70 serous histology patients. Two patients from each cluster have missing values. Stage number is a significant predictor of cluster ($p = 0.0184$). Clusters are assigned by hierarchical clustering. 18

4.2 Patients with differering cluster assignments by HC and WECCA. CS2688 is the only patient who is initially assigned to Cluster 3 by WECCA. For analysis, CS2688 is reassigned to WECCA Cluster 1. 23

5.1 Selected running times for the RJaCGH function with varying numbers of samples and markers. 27

5.2 Tabulated results for each clustering method, including a varying probability criterion, p_w, for RDU. Rows represent the clusters assigned to generated profiles by the indicated clustering method. Columns represent the two known aCGH profiles, with 5,000 profiles generated from each. Table totals are 10,000 cluster assignments. 28

5.3 Estimates, standard errors, and 95% confidence limits for Cohen's kappa are computed from the tabulated cluster assignments from 1,000 simulations. 29

LIST OF FIGURES

4.1 Dendrogram for hierarchical clustering using Ward's method. Performed on the $\log_2$ ratios for 78 ovarian cancer patients with clinical information. 17
4.2 Dendrogram for hierarchical clustering using Ward's method. Performed on the $\log_2$ ratios for the 74 patients with serous histology. 17
4.3 Kaplan-Meier estimates of overall survival for 74 patients by HC cluster with 95% confidence limits ($p = 0.777$). 19
4.4 Kaplan-Meier estimates of progression-free survival for 74 patients by HC cluster with 95% confidence limit ($p = 0.033$). 19
4.5 WECCA dendrogram. Based on called aCGH data, WECCA clusters 74 serous histology patients into 3 groups of sizes 1, 45, and 28. Part of the methodology includes hierarchical clustering; this dendrogram is produced using average linkage with the concordance measure. Clusters 1 and 2 (sizes 46 and 28) are used throughout the WECCA analysis. 20
4.6 Kaplan-Meier estimates of overall survival for 74 patients by WECCA cluster with 95% confidence limits ($p = 0.848$). 21
4.7 Kaplan-Meier estimates of progression-free survival for 74 patients by WECCA cluster with 95% confidence limits ($p = 0.150$). 22
4.8 Kaplan-Meier estimates of progression-free survival by HC cluster ($p = 0.038$) and WECCA cluster ($p = 0.150$) for 74 serous histology patients. 22

CHAPTER 1

INTRODUCTION

Array comparative genomic hybridization (aCGH) is a technique for identifying duplications and deletions of DNA at specific locations across a genome. The number of duplicate segments of DNA at any genomic location is the copy number corresponding to that location. Each genomic location typically has two copies of DNA. Because many cancers involve alterations in copy number, aCGH data is useful in identifying genes that directly relate to tumor progression. Other applications for aCGH technology are in diagnosis, cancer classification, human variation, interspecies comparison, and characterization of genetic syndromes.

To collect the data, an array is prepared with DNA sequences to serve as the target for the binding of the test sample and the reference sample. In cancer studies, the test sample is a fragment of DNA that is extracted from a tumor. A reference sample is a corresponding DNA fragment that is drawn from normal tissue. The test and reference samples from a single patient are each labeled with a different florescent dye before they are mixed and allowed to hybridize with the target DNA on the microarray. Smaller sections of target DNA increase genomic resolution, allowing for the detection of smaller copy number aberrations and more precise identification of the locations where they occur.

A microarray scanner produces an image for each array location that is converted to florescence intensities. The $\log_2$ ratio of these intensities quantifies the presence of DNA from the test sample relative to the reference sample. If more copies of a gene are present in the test sample, these genes will bind to the target DNA more frequently than will those in the reference sample. Changes from zero in the $\log_2$ intensity ratios then presumably reflect alterations in copy number.

Aside from experimental error, the variation in the observed $\log_2$ ratios could come from tissue heterogeneity in the test or reference sample (Lockwood et al. 2006) or from markers that typically exhibit a copy number not equal to two (Iafrate et al. 2oo4). Large-scale copy number variation (longer segments of DNA with gains or losses in copy number) may be inherited or caused by mutation. Thus, copy number sequences are expected to differ by individual, especially since recent studies have shown that copy number variants are responsible for much of the genomic variation in humans (Freeman et al. 2006). Numerous studies have found copy number variants which contain genes that have been previously associated with cancer or genetic diseases.

Potential objectives of aCGH analysis are the identification of (1) altered regions for a given subject, (2) altered regions across a set of individuals, and (3) clinically relevant clusters of hybridizations. These goals have motivated many statistical methodologies.

In this paper, the assessment of existing aCGH clustering methodologies is of interest. Assessment is conducted first through the analysis of aCGH data obtained from patients with ovarian cancer and then through simulations.

CHAPTER 2

LITERATURE REVIEW

This section describes existing methods for identifying DNA copy number aberration for both single and multiple hybridizations; it also contains a review of clustering methods that have been used for grouping samples according to aCGH profiles.

2.1 Identifying Copy Number Aberration for Single Hybridizations

For a single sample of aCGH data, it is often useful to identify the nature of copy number aberration and where it occurs in the genome. A variety of methods exist for identifying copy number gains and losses.

Pollack et al. (2002) finds genes of significant gain or loss by comparing a moving average of flourescence ratios to a threshold based on standard deviations of ratios produced from the hybridization of a normal sample to another normal sample. Calculating thresholds in this way allows the cutoff point to vary from gene to gene.

Binary segmentation can be used to estimate locations of mean copy number change, called change points, throughout a genome (Braun and Mueller 1998). Olshen and Venkatraman (2002) modify this procedure in order to improve the detection of small segments of change. While binary segmentation is only capable of detecting one change point at a time, circular binary segmentation (CBS) can detect two change points at once. Originally derived from the assumption that data points are normally distributed around each segmented mean, CBS is extended to non-normal data using a permutation approach to identify significant change points (Olshen et al. 2004).

Some model-based methods of breakpoint detection use a penalized likelihood (Jong et al. 2003; Picard et al. 2005). Jong's breakpoint detection method is applied to each chromosome to cluster aCGH values so that the numbers of clones in each group are equal.

After identifying potential sets of breakpoints, maximum likelihood estimation with the addition of a penalty for the number of breakpoints is used to find the breakpoints that best fit the data. Maximum likelihood is also used to estimate the copy number for each cluster of aCGH values, assuming that the values between each breakpoint follow a normal distribution with unique parameters. GLAD (Gain and Loss Analysis of DNA) is another method for detecting breakpoints in copy number patterns and assigning each chromosomal section a status of normal, gain, or loss (Hupe et al. 2004). Detecting breakpoints stems from adaptive weights smoothing (Polzehl and Spokoiny 2000), a procedure for smoothing discontinuous data while preserving contrasts between regions.

Wang et al. (2005) propose a method for calling gains and losses in a single sample that uncovers the underlying structure of genomic alterations. Clustering along chromosomes (CLAC) builds hierarchical clusters of adjacent clones that may be assessed as gain, loss, or normal based on the order in which the cluster formed, the number of genes in the cluster, and the mean value of the $\log_2$ ratios for genes in the cluster. An estimate for the false discovery rate (FDR) is used to determine which of the called gains and losses are legitimate.

In order to utilize information about nearby clones, Fridlyand et al. (2004) employ a hidden Markov model (HMM) approach to map and characterize the number of copies of DNA from normalized $\log_2$ ratios. The first step segments clones into sets with the same underlying copy number, which are represented by states in an HMM. The number of states that minimizes the penalized log-likelihood is an intermediate estimate for the number of segments that are present. Then the states with the smallest difference in medians are combined, resulting in one fewer state, until the distance between states exceeds a specified threshold. The genomic alterations are characterized in four types using the median of median absolute deviations (MMAD) for clones in a given segment. Guha's Bayesian HMM (Guha et al. 2008) uses posterior probabilities to locate small and large-scale copy number aberrations.

A pseudolikelihood approach to aCGH analysis also incorporates commonalities across chromosomes and hybridizations (Engler et al. 2006). This method assigns a Gaussian mixture model to the distribution of $\log_2$ ratios and uses the hidden Markov dependence of neighboring clones to generate probabilities of gain, loss, or no-change at each location on the genome. Also, conditional probabilities of the classification of three adjacent clones can be used to identify likely breakpoints. Combining concepts from several aforementioned methods, van de Wiel and van Wieringen (2007) introduce the CGHcall algorithm, which uses a mixture model with six states to distinguish single and double deletions and gains in copy number.

2.2 Identifying Copy Number Aberration Across Multiple Hybridizations

Several methods have also been published on identifying copy number alteration that is common to a set of aCGH profiles.

Cheng et al. (2003) use regression analysis to find aberrations in the mean flourescence ratio for each clone across multiple arrays, controlling the Type 1 error rate by calculating p-values with a permutation test. Wang et al. (2005) use an estimate of the positive false discovery rate (Storey 2002) to provide a consensus summary of gains and losses for multiple hybridizations after applying CLAC to each individual array. In the pseudolikelihood approach by Engler et al. (2006), the model accounts for variation between hybridizations, as well as within and between chromosomes, and requires no modification for the joint analysis of multiple samples.

Another method designed for analyzing changes in copy number across multiple samples is called Genomic Identification of Significant Targets in Cancer (GISTIC). After identifying aberrations in individual tumors, the amplitudes of aberrations at a specific marker are summed over all tumors, yielding a G-score for each location on the genome. Permuting the locations generates a reference distribution of G-scores, from which a significance threshold can be determined for assessing the significance of the observed G-scores. The significant

genomic locations mark common gains and losses across multiple samples (Beroukhim et al. 2007).

CGHregions (van de Wiel and van Wieringen 2007) is a method of reducing multiple aCGH samples from thousands of probes to tens or hundreds of regions. A region is a subset of consecutive probes whose distance is within a specified threshold; the distance function essentially counts the discrepancies in the called copy number data. Another set of rules determines region boundaries. The data reduction makes the aCGH analysis easier to interpret in biological applications and speeds the computing time of later analyses. The number of regions found is controlled to minimize the loss of information; the sequences of probes that overlap across a set of samples constitute the probes that are excluded in the reduced data, as they provide no information for differentiating samples.

Rueda and Diaz-Uriarte (2009) propose two different approaches for finding genomic regions of copy number alteration (CNA) that are common to several samples: pREC-A (probabilistic recurrent copy number regions, common threshold over all arrays) and pREC-S (probabilistic recurrent copy number regions, subset of arrays). Both methods produce probabilities of regions being altered over a set of arrays. The pREC-A method is intended for studying a homogeneous group of individuals, while pREC-S is intended for discovering subgroups with common CNA regions that are part of a larger heterogeneous group of subjects. Thus pREC-S may be used for clustering. Following a description of how joint probabilities are computed are general descriptions of the pREC-A and pREC-S algorithms. Rueda and Diaz-Uriarte (2009) illustrate these algorithms using simple numerical examples. Both methods use joint probabilities of alteration while considering sequences of probes for recurrent regions.

Computing Probabilities

Both pREC-A and pREC-S require the probability that a set of contiguous probes is altered as either a gain or as a loss. Because these probes are within a close genomic distance,

their states of gain or loss are not independent, and their marginal probabilities of alteration cannot be multiplied. The marginal probabilities are computed prior to implementing the pREC algorithms with a Bayesian hidden Markov model.

Using Markov chain Monte Carlo (MCMC), Rueda and Diaz-Uriarte (2009) calculate the joint probability of an alteration for a particular sequence of probes as the proportion of those sequences that are altered in the total number of maximum a posteriori sequences (MAPs). Rather than sampling from the distribution of hidden states, sampling is from the distribution of MAPs. Although the MCMC iterations produce conditional probabilities of altered probes, these are not averaged, in keeping with the Markovian property, because the model parameters are adjusted each time. These joint probabilities are for particular sequences of probes in separate arrays (individuals).

To find regions that are common to a set of arrays, the joint probabilities (of an altered sequence of probes) can be averaged over all arrays because the data have been summarized in classes, or states, of gain, loss, or no change, which are comparable across individuals.

pREC-A

This algorithm finds all the regions with an average probability of alteration over all arrays that is at least p_a. It begins with the first probe of each chromosome. If the probability of alteration at this probe over all arrays is greater than p_a, then this probe is considered the beginning of a region. Probes are added to the region consecutively while the joint probability of alteration is greater than p_a. When the joint probability of alteration falls below the threshold, the region ending with the previous probe is stored as a starting and an ending postition with the average probability of alteration for that region. The current probe becomes the starting position for the next region if its probability of alteration exceeds p_a.

pREC-S

This algorithm finds all the regions with a probability of alteration of at least p_w within a subset of arrays. The smallest number of arrays in a subset is determined by the parameter $freq.arrays$. The algorithm sequentially considers each probe as the starting position for a region. An initial set of arrays consists of those with a marginal probability of alteration greater than p_w at this location. Provided the number of candidate arrays for sharing a common CNA region is greater than $freq.arrays$, the joint probability of alteration over the starting probe and the next probe is calculated for each candidate array. As long as the joint probabilities for every array are greater than p_w, the region under consideration is extended by one probe and the probabilities recalculated.

Eventually the region either reaches the end of a chromosome or the probability falls below the threshold for at least one of the arrays. The latter case indicates that over the set of current candidate arrays, the region of common copy number alteration occurs from the starting probe to the previously considered probe position. This information is saved as a region.

Now the smaller set of arrays for which joint probabilities through the current probe are greater than p_w becomes the new set of candidate arrays. The regions are continually extended one probe at a time until the end of a chromosome or until the number of arrays that satisfy the p_w threshold drops below $freq.arrays$, the minimum number of arrays required to form a region. If the region is not completely contained in a previously stored region, the algorithm saves the region and considers regions for sets of arrays that begin with the next probe in the sequence.

2.3 Clustering Hybridizations

Based on patterns of copy number aberration in aCGH profiles, individuals often form natural groups that offer useful biological insights. Some methods that have been used for clustering aCGH samples are described in this section.

Hierarchical clustering has been used frequently in the analysis of aCGH data (Wilhelm et al. 2002; Pollack et al. 2002; Massion et al. 2002; Hackett et al. 2003; Diskin et al. 2006). The implementation is straightforward, fast, and visually helpful; choosing the number of clusters may be done in a variety of ways, not discussed here (Everitt et al. 2001).

Liu et al. (2006) applies distance-based clustering to CGH data with three measures of similarity using bottom-up, top-down, and k-means clustering algorithms. A "segment-based similarity" called Sim, which counts the number of overlapping segment pairs, is the distance metric that produces the best clustering results, accounting for correlation between adjacent genomic locations.

The first method developed specifically for clustering called aCGH data, termed WECCA (weighted clustering of called aCGH data), allows certain chromosomal regions to be assigned more weight in hierarchical clustering by modifying the similarity measure (van Wieringen et al. 2008). Rather than clustering on called aCGH data, Shah et al. (2009) use an EM-like algorithm to simultaneously determine the cluster profiles and assign samples to clusters directly from the aCGH ratios. This is a model-based approach where each cluster profile is defined by a hidden Markov model; hence, it is called HMM-Mix.

Pan and Shen (2007) use a penalized likelihood approach designed specifically for multivariate clustering; the penalty function acts as a threshold to select variables to use for clustering samples. There are other semi-supervised learning algorthims that have been applied to gene expression data (McLachlan et al. 2002; Alexandridis et al. 2004). Details of hierarchical clustering and WECCA are described in the next sections. Additionally mentioned is how the results from the pREC-S method for analyzing multiple hybridizations may be used for clustering.

Hierarchical Clustering

Hierarchical clustering is a method for forming natural groups, or clusters, of observations without prior knowledge of the existence or nature of the groups. Agglomerative clustering

begins with each observation in a separate group. The two most similar groups are combined, forming a new cluster. This step is repeated until all observations have been combined into a single cluster. Divisive clustering begins with a single cluster and successively divides the most dissimilar groups until one observation remains in each group. The hierarchy of cluster assignments is visually represented in a tree structure called a dendrogram. The pattern and length of the dendrogram branches reflect the similarity of the samples being clustered.

Some measure of similarity is necessary for determining the closeness of individual observations. For continuous data, this is often a distance or dissimilarity measure. For any observations x_1, x_2, and x_3, a distance metric d satisfies the following properties: identity, where $d(x_1, x_1) = 0$; uniqueness, where $d(x_1, x_2) = 0$ implies $x_1 = x_2$; symmetry, where $d(x_1, x_2) = d(x_2, x_1)$; and the triangle inequality, $d(x_1, x_2) + d(x_2, x_3) \geq d(x_1, x_3)$ (Veltkamp and Latecki 2006). Some examples of distance metrics are Euclidean distance, Minkowski distance, and Canberra distance. Depending on the data, other properties may desirable for similarity measures.

Linkage methods describe the way in which distance is defined between two clusters C_1 and C_2 when one or more of the clusters contain multiple observations. Single linkage defines the distance between two clusters as the smallest distance between any two observations x_1 in C_1 and x_2 in C_2. It tends to find long and thin clusters, where observations on opposite ends of the same cluster may actually be quite different. This chaining makes single linkage useless for finding well-separated clusters. Complete linkage tends to find more compact clusters by defining the distance between two clusters as the maximum distance between any two members, one from each cluster. Average linkage describes the distance between two clusters as the average of all pairwise distances $d(x_1, x_2)$, where $x_1 \in C_1$, $x_2 \in C_2$. The nature of this linkage method causes clusters with small variances to merge first. It also incorporates cluster structure, unlike single and complete linkage (Everitt et al. 2001).

Ward's method merges two clusters if the total loss of information is minimized. Information loss is typically taken to be the total error sum of squares within clusters. This

method is sensitive to outliers and tends to find spherical clusters of similar sizes (Everitt et al. 2001).

Weighted Clustering of Called Array CGH Data

Weighted clustering of called aCGH data (WECCA) was the first method developed specifically for clustering called aCGH data (van Wieringen et al. 2008). Called data refers to copy number ratios that have already been classified as loss, normal, or gain. In some cases, called data may have four categories: loss, normal, gain, and amplification.

The model that van Wieringen et al. (2008) use as a framework for defining two similarity measures makes no distributional assumptions. It does assume independence between n samples (indexed by i), where every sample is from group g, out of m unknown groups, or clusters. Each sample consists of copy number measurements for p clones (indexed by j). While individual clones may not be independent, the model assumes that every clone belongs to exactly one of r independent regions (indexed by s). The probability that the DNA copy number X from sample i and clone j equals k is denoted by $P(X_{ij} = k)$, where $k = 1, ..., a$ and a is the number of possible categories to which a call may belong.

Because ordinal measurements are in distinct categories, it is certainly possible that the copy number at an arbitrary clone j is the same for two samples. That is, the samples agree. The agreement of two samples over all clones is more likely for samples that are from the same unknown cluster. The probability of agreement between two samples is the first measure that van Wieringen et al. (2008) propose for assessing the similarity of two called aCGH samples.

The second measure for similarity stems from the ordering in magnitude of ordinal categories. An equal and same-directional change in magnitude between any two clones j_1 and j_2 for two samples i_1 and i_2 shows concordance between two samples. Clones with equal copy numbers also contribute to the concordance of two samples, as this may indicate a shared underlying mechanism. The probability of concordance between two arbitrary

samples may be written generally as

$$\begin{aligned} P(\text{concordance}) =& P(X_{i_1 j_1} < X_{i_1 j_2}, X_{i_2 j_1} < X_{i_2 j_2}) \\ &+ P(X_{i_1 j_1} = X_{i_1 j_2}, X_{i_2 j_1} = X_{2 j_2}) \\ &+ P(X_{i_1 j_1} > X_{i_1 j_2}, X_{i_2 j_1} > X_{i_2 j_2}). \end{aligned}$$

Both similarity measures are modified to incorporate weights, by clone or by region, into the clustering process. Clustering weighted by clone requires some prior knowledge of which clones are useful in providing information about real groups that we expect to emerge from the data. By default, clustering weighted by region assigns each region equal weight so that regions of smaller size are given the same importance in forming clusters as regions of substantial length. If desired, the code can be modified to incorporate different weights for regions. Simulations that evaluate WECCA show that the best clustering results from using region data with concordance as the similarity measure and average linkage. With positive weights w_j for each clone j, $j = 1, ..., p$, the estimator for the probability of concordance is given by

$$\begin{aligned} \hat{P}(\text{concordance}) = & \frac{1}{\sum_{\substack{j_1, j_2 = 1 \\ j_1 \neq j_2}}^{p} w_{j_1} w_{j_2}} \sum_{k_1=2}^{a} \sum_{k_2=1}^{k_1 - 1} \cdots \sum_{k_{2r-1}=2}^{a} \sum_{k_{2r}=1}^{k_{2r-1}-1} \\ & \times \sum_{\substack{j_1, j_2 = 1 \\ j_1 \neq j_2}}^{p} w_{j_1} w_{j_2} I_{\left\{X_{i_1 j_1} = k_1, X_{i_1 j_2} = k_1\right\}} \cdots I_{\left\{X_{i_r j_1} = k_{2r-1}, X_{i_r j_2} = k_{2r-1}\right\}} \\ & + \frac{1}{\sum_{\substack{j_1, j_2 = 1 \\ j_1 \neq j_2}}^{p} w_{j_1} w_{j_2}} \sum_{k=1}^{a} \sum_{\substack{j_1, j_2 = 1 \\ j_1 \neq j_2}}^{p} w_{j_1} w_{j_2} I_{\left\{X_{i_1 j_1} = k, X_{i_1 j_2} = k\right\}} \cdots I_{\left\{X_{i_r j_1} = k, X_{i_r j_2} = k\right\}} \\ & + \frac{1}{\sum_{\substack{j_1, j_2 = 1 \\ j_1 \neq j_2}}^{p} w_{j_1} w_{j_2}} \sum_{k_1=1}^{a-1} \sum_{k_2=k_1+1}^{a} \cdots \sum_{k_{2r-1}=1}^{a-1} \sum_{k_{2r}=k_{2r-1}+1}^{a} \\ & \times \sum_{\substack{j_1, j_2 = 1 \\ j_1 \neq j_2}}^{p} w_{j_1} w_{j_2} I_{\left\{X_{i_1 j_1} = k_1, X_{i_1 j_2} = k_1\right\}} \cdots I_{\left\{X_{i_r j_1} = k_{2r-1}, X_{i_r j_2} = k_{2r-1}\right\}}. \end{aligned}$$

Total linkage is a new type of linkage for defining the similarity between clusters. Total linkage for agreement and concordance is the probability that all samples are in agreement

or in concordance, respectively. In the supplementary material, van Wieringen shows that tighter clusters are more likely to form when using total linkage than when using average or complete linkage. Hierarchical clustering is then performed with the number of clusters chosen so that clusters are compact and well-separated.

Rueda and Diaz-Uriarte Method

The pREC-S method that Rueda and Diaz-Uriarte (2009) propose for finding recurrent CNA regions among subsets of samples uses clustering to represent patterns of similarity among groups of arrays. For any two arrays, the algorithm returns the number of common probes in recurrent regions of alteration, as well as the number of common regions. The authors mention that either of these quantities may be used to measure similarity between samples, after which any clustering method may be applied. A graphical representation of hierachical clustering with single linkage is the default output for the pREC-S method.

CHAPTER 3

METHODS

Performance of the three methods of unsupervised clustering described in this paper are assessed in `R` on aCGH data from ovarian cancer patients and through simulation studies. The aCGH data consists of $\log_2$ intensity ratios for over 99,000 genetic markers for each of 78 ovarian cancer patients. Hierarchical clustering on the $\log_2$ ratios is carried out using Ward's minimum variance method; hierarchical clustering (HC) has been commonly used in aCGH analysis.

WECCA is also used because it has been developed specifically for clustering called aCGH data, where each marker has been identified as, say, gain, loss, or no change. This method employs hierarchical clustering, which is implemented using average linkage with concordance as the similarity measure. Classification data for regions rather than individual markers is used; regions are weighted equally.

There are two main components of the Rueda/Diaz-Uriarte (RDU) method. The first component fits a Bayesian hidden Markov model to aCGH data using Reversible Jump MCMC. This produces a marginal probability of either gain or loss at each marker; these probabilities are used as input for the pREC-S algorithm, the second component. The pREC-S algorithm finds regions of gain or regions of loss above a certain probability threshold that are shared by a specified proportion of patients. These regions, common to subsets of individuals, are used as a foundation for clustering on the number of common regions between any two individuals. A visual representation of hierachical clustering with single linkage is the default output for the pREC-S method. The first component requires the bulk of the running time for the entire RDU method.

In the data analysis for HC and WECCA, performance assessment is based on cluster assignment correlation with clinical outcomes (e.g., survival). The nature of the RDU method is not conducive to analyzing large data sets, so this analysis is not included. Simulation studies demonstrate how well each clustering method correctly groups patients generated from common underlying aCGH profiles.

3.1 Modifications to the Rueda/Diaz-Uriarte Method

The pREC-S algorithm computes probabilities of either gains or losses, not alterations in general. Thus the distance matrices containing the number of common markers in regions of either gain or loss, not both, lead to two sets of cluster assignments. One set of cluster assignments is more useful for comparing the simulated results from Rueda/Diaz-Uriate (RDU) to the simulated results that are obtained by HC and WECCA.

Because the number of shared regions of gain for any two samples will be different than the shared regions of loss, the distance matrices for gains and losses may be added, provided the samples in each matrix are ordered so that the entries correspond. The resulting distance matrix includes the number of recurrent regions of both gain and loss. Thus hierarchical clustering performed on this matrix produces a single set of cluster assignments based on both gain and loss regions, which can be compared to the known clusters from which profiles were generated, as with the other clustering methods.

Additionally, hierarchical clustering is performed with complete linkage to avoid the chain-like clusters that are often produced with single linkage.

CHAPTER 4

DATA ANALYSIS

This chapter contains cluster analyses with HC and WECCA applied to ovarian cancer data. Because the RDU method has difficulty processing large amounts of data, it is not applied to this data set. Further explanation of its limitations are given later.

The aCGH data consists of $\log_2$ intensity ratios for 99,264 genetic markers for each of 78 ovarian cancer patients. Up to 53 clinical variables were collected from these patients. The clinical data contains information on a patient's cancer progression, survival, and treatments received (Engler et al. 2010). Logistic regression is used to identify clinical variables that relate to a method's clustering results. Variables that are not considered are dates and those with nearly identical values across patients.

4.1 Hierarchical Clustering

Using the $\log_2$ ratios of the 78 cancer patients, HC produces the dendrogram in Figure 4.1. There appear to be three distinct groups. The two larger clusters merge before joining the third cluster of only four subjects. Although there are only four subjects in this data set with nonserous histology, only two of these subjects belong to the small cluster.

Clustering only the 74 subjects with serous histology produces the dendrogram with two distinct groups in Figure 4.2. Cluster 1 contains 44 patients. Cluster 2 contains 30 patients. These cluster assignments are used for assessing the correlation between HC clusters and clinical variables.

The marginal effects of about 30 clinical variables are examined using logistic regression with the cluster assignment as the response (Cluster 1 = 0, Cluster 2 = 1). With a significance level of 0.05, only stage number (2, 3, or 4) as a continuous variable appears to

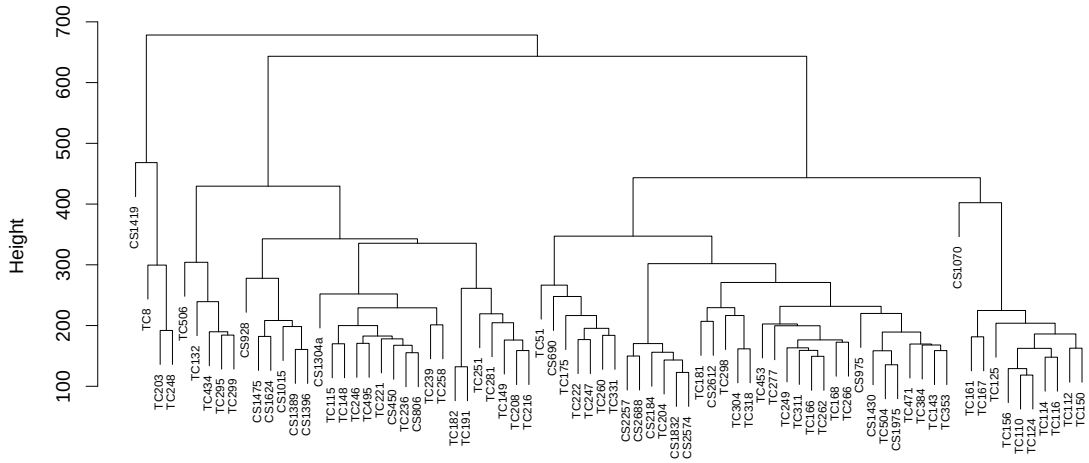

Figure 4.1: Dendrogram for hierarchical clustering using Ward's method. Performed on the $\log_2$ ratios for 78 ovarian cancer patients with clinical information.

Figure 4.2: Dendrogram for hierarchical clustering using Ward's method. Performed on the $\log_2$ ratios for the 74 patients with serous histology.

be associated with cluster ($p = 0.0184$). There are four patients with missing stage numbers, two from each cluster. Counts by stage number and cluster are found in Table 4.1.

Table 4.1: Patient counts by stage number (2, 3, 4) and cluster (1, 2) with marginal totals. In parentheses are cell proportions out of 70 serous histology patients. Two patients from each cluster have missing values. Stage number is a significant predictor of cluster ($p = 0.0184$). Clusters are assigned by hierarchical clustering.

		Cluster		
		1	2	Row Totals
Stage Number	2	0 (0.00)	5 (0.07)	5 (0.07)
	3	26 (0.37)	17 (0.24)	43 (0.61)
	4	16 (0.23)	6 (0.09)	22 (0.32)
Column Totals		42 (0.60)	28 (0.40)	70 (1.00)

It is clear from the Kaplan-Meier (K-M) curves in Figure 4.3 that clusters are not related to overall survival. The p-value for the logrank test is 0.777. Figure 4.4 shows the K-M estimates of progression-free survival (PFS) by cluster. In the first 4 months, the patients in Cluster 1 have lower risk of cancer recurrence. This trend reverses after four months, and Cluster 2 patients have longer time to cancer recurrence than do Cluster 1 patients. Based on the variability of the PFS estimates, shown in the confidence intervals, it is possible that this slight crossing of survival curves is only due to chance. After the crossing point, which occurs relatively early, the separation between curves increases, especially in later months. Thus it is reasonable to assume that the assumption of proportional hazards is not violated, so the logrank test is appropriate for testing the equivalence of two survival curves. Despite some overlap in the confidence intervals, the logrank test ($p = 0.033$) shows a statistically significant ($\alpha = 0.05$) difference in PFS by cluster.

A Cox proportional hazards regression model can further quantify the relationship between cluster and PFS. The estimated coefficient for cluster as the only covariate is -0.6636 with a corresponding p-value of 0.0388. (As this is equivalent to the logrank test, the p-values are very close.) The log hazard ratio of Cluster 2 relative to Cluster 1 can be computed as $e^{-0.6636} = 0.515$, with a 95% confidence interval of (0.2744, 0.9664). While the risk of death for Cluster 2 patients is estimated to be 0.515 times that of Cluster 1, decrease in risk for Cluster 2 patients could be as high as 72.5% or as low as 3.4%.

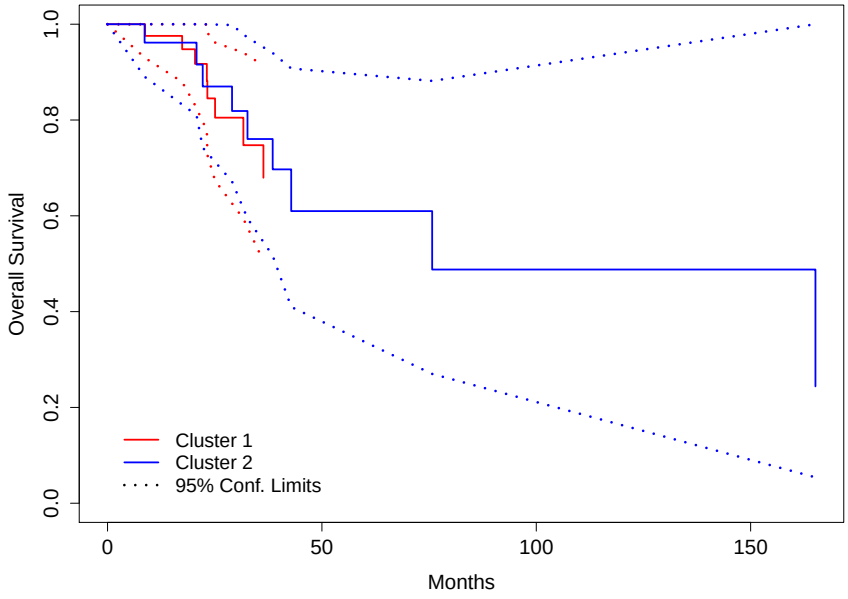

Figure 4.3: Kaplan-Meier estimates of overall survival for 74 patients by HC cluster with 95% confidence limits ($p = 0.777$).

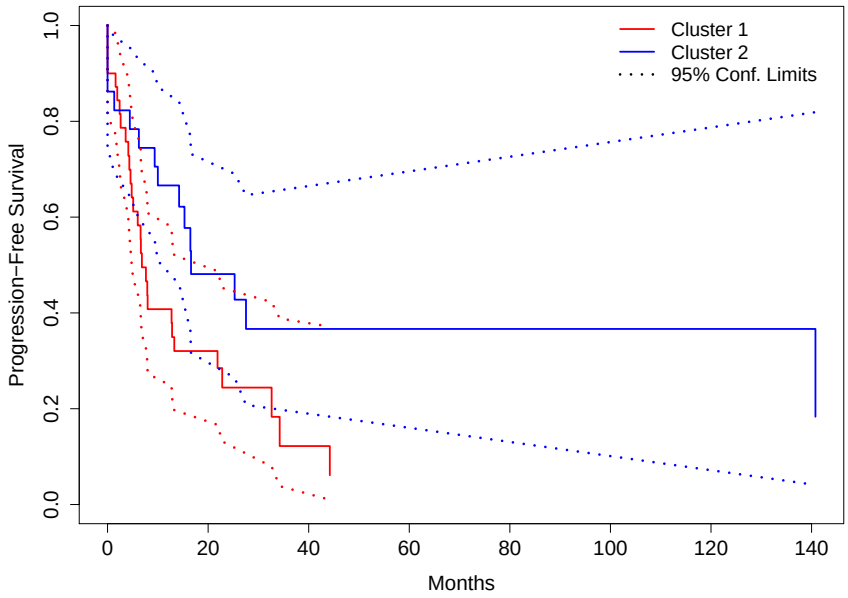

Figure 4.4: Kaplan-Meier estimates of progression-free survival for 74 patients by HC cluster with 95% confidence limit ($p = 0.033$).

4.2 WECCA

The WECCA method clusters 73 of the 74 serous histology patients into two main groups with a single patient (sample ID: CS2688) in a separate cluster (see Figure 4.5). To relate these clusters to the clinical data, it is useful to consider only two main clusters. The patient CS2688 will be combined with the largest group for a total of 46 patients in Cluster 1. Assuming that the cluster of 45 patients has more variability than does the cluster of size 28, making Patient CS2688 part of Cluster 1 will introduce the least within-cluster variation. Aside from forming a single cluster, it is not obvious from the clinical data that there is any reason to exclude this patient. Cluster 2 refers to the cluster of size 28.

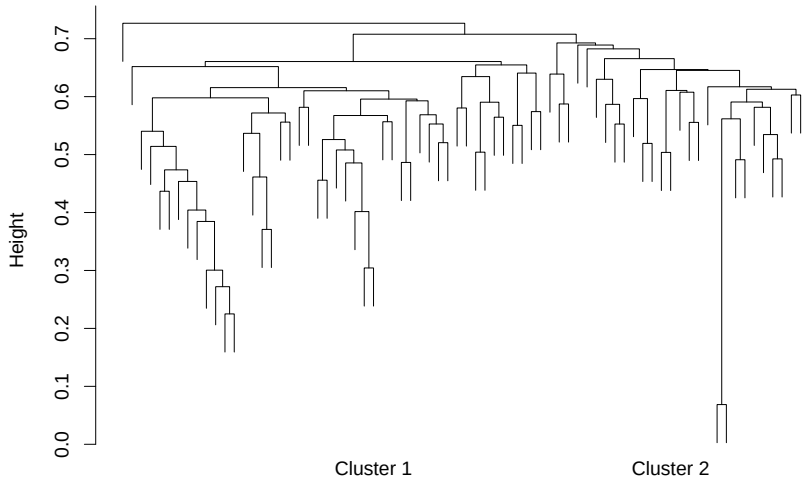

Figure 4.5: WECCA dendrogram. Based on called aCGH data, WECCA clusters 74 serous histology patients into 3 groups of sizes 1, 45, and 28. Part of the methodology includes hierarchical clustering; this dendrogram is produced using average linkage with the concordance measure. Clusters 1 and 2 (sizes 46 and 28) are used throughout the WECCA analysis.

None of the clinical variables are significant predictors for assigning these patients to Clusters 1 or 2. Figures 4.6 and 4.7 show the K-M estimates of overall survival and progression-free survival by clusters. For overall survival, the curves are essentially the same; the logrank p-value is 0.848. Based on the similarity of survival trends for Clusters 1 and 2 through the first 42 months, there is no reason to attribute this difference to anything

more than chance. In Figure 4.7 the PFS curves are very similar for approximately five months. After that time, Cluster 2 patients are estimated to have lower risk for recurrence of cancer. Assuming proportional hazards, a logrank test for the difference in survival curves is not significant with a p-value of 0.150.

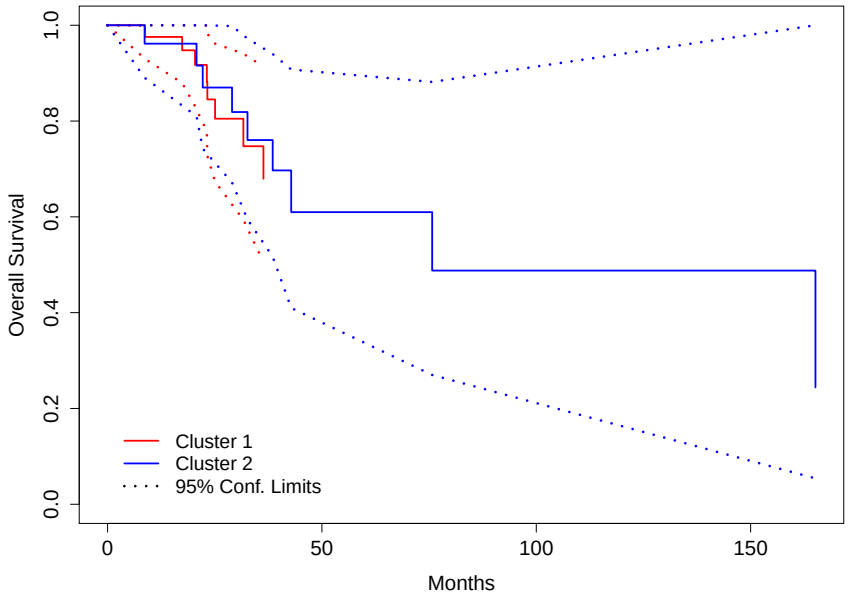

Figure 4.6: Kaplan-Meier estimates of overall survival for 74 patients by WECCA cluster with 95% confidence limits ($p = 0.848$).

4.3 Discussion

The sizes of the clusters produced by HC (44, 30) and WECCA (46, 28) are fairly similar. In both methods (see Figures 4.2 and 4.5), the larger cluster forms first (and thus is identified as Cluster 1). Except for stage number of cancer for HC clusters, none of the clinical variables in this data set are associated with HC or WECCA clusters. Within each method, the clusters are not associated with overall survival. Figure 4.8 shows PFS curves by cluster for both methods; the survival curves by cluster are quite similar across methods. However, HC clusters appear to be associated with PFS, while WECCA clusters are not.

Although HC and WECCA naturally have variation in cluster composition because they are different clustering methods, the similarity in cluster sizes and survival by clusters

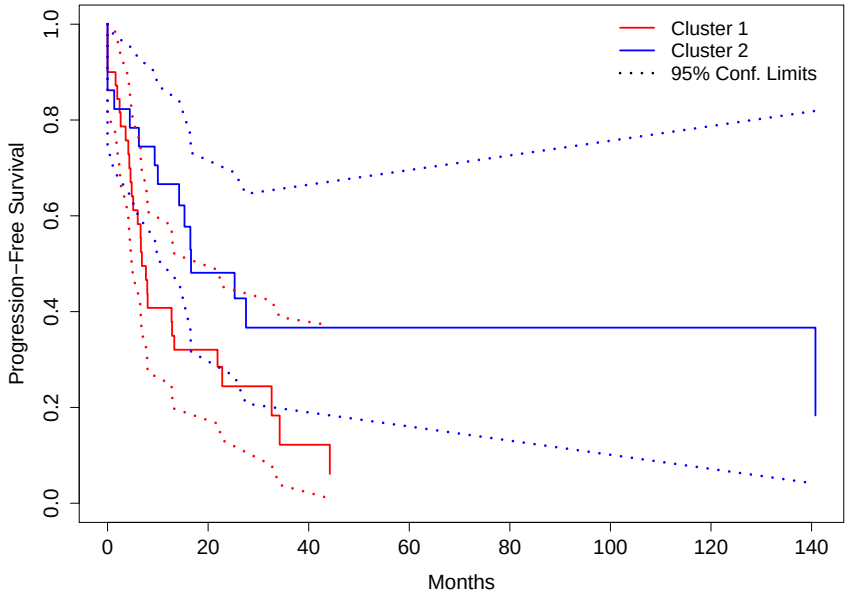

Figure 4.7: Kaplan-Meier estimates of progression-free survival for 74 patients by WECCA cluster with 95% confidence limits ($p = 0.150$).

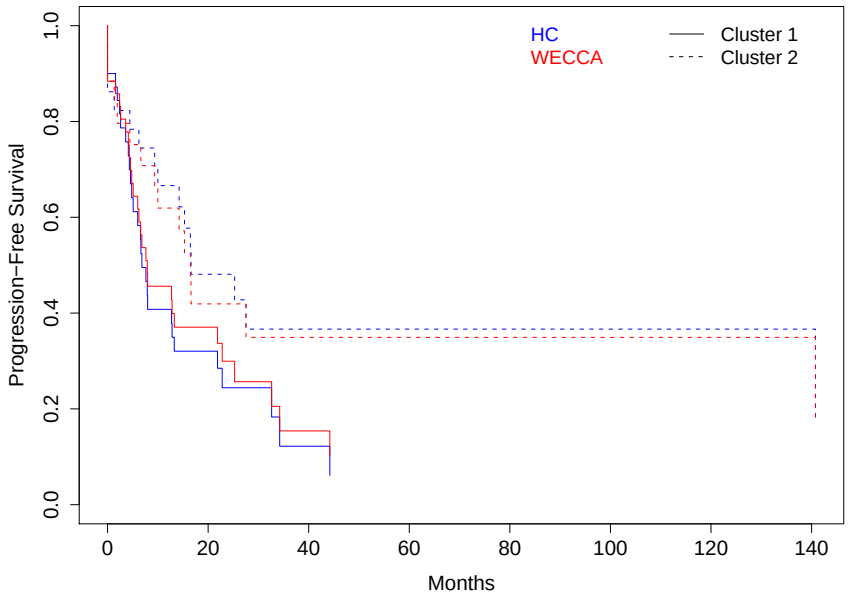

Figure 4.8: Kaplan-Meier estimates of progression-free survival by HC cluster ($p = 0.038$) and WECCA cluster ($p = 0.150$) for 74 serous histology patients.

suggests that both methods identify many of the same patients as being similar. In fact, only eleven out of the 74 patients with serous histology are assigned to different clusters across the methods of HC and WECCA (see Table 4.2). All other patients are placed in the same clusters by HC and WECCA. The difference in logrank p-values for PFS by cluster indicates that these eleven patients do change the cluster composition in relation to progression-free survival. Even though WECCA is designed to cluster patients according to the ordinal nature of the data, hierarchical clustering on the $\log_2$ ratios seems to produce more clinically relevant clusters from this data.

Table 4.2: Patients with differering cluster assignments by HC and WECCA. CS2688 is the only patient who is initially assigned to Cluster 3 by WECCA. For analysis, CS2688 is reassigned to WECCA Cluster 1.

Patient	HC	WECCA
TC161	1	2
TC167	1	2
TC204	1	2
TC221	2	1
TC453	1	2
TC506	2	1
TC8	2	1
CS1389	2	1
CS1396	2	1
CS1419	2	1
CS2688	1	3

Computational Considerations

Hierarchical clustering on 74 patients runs for about 10 seconds. WECCA clusters the same data in about 2.6 hours. In an attempt to apply the RDU method to the 78 patients with clinical data, the first component of RDU was implemented with the RJaCGH function to obtain a set of input probabilities that is required for the pREC-S algorithm. The running time for this function was about five days, after which the pREC-S algorithm appeared to be running indefinitely. Clearly, this method is not practical for clustering large data sets.

CHAPTER 5

SIMULATIONS

Simulations demonstrate how well each clustering method correctly groups patients generated from common underlying aCGH profiles. The following sections discuss the structure of the simulation study, the process of generating aCGH profiles, and the summarized results. Cohen's kappa (Cohen 1960) is a coefficient of agreement for nominal scales that is used for comparing clustering results across methods.

5.1 Generating aCGH Data

Simulated aCGH data is generated based on observed aCGH data from the ovarian cancer data set previously analyzed. CBS (Olshen et al. 2004) divides each patient profile of $\log_2$ intensity ratios into segments of varying lengths. Each segment contains markers with similar $\log_2$ ratios and has an associated mean. This method of generating aCGH data uses segmented means and their associated classifications of loss, gain, and no change to create an empirical distribution of segment lengths for each classification.

A single aCGH profile is generated in sections specified by marker indices for likely regions of gain, loss, or no change. Probabilities for each alteration are also specified. Within each section, a classification is selected according to probabilities of loss, gain, or no change, which are based on the specified probability p. For likely sections of loss (gain), 90% of $1-p$ becomes the probability of no change for the region, with the remaining 10% of probability allotted toward gain (loss). For likely sections of no change, the probabilities of gain and loss divide $1-p$ equally. After selecting the region classification, the length for the region is drawn from the corresponding distribution of segment lengths. This process will continue as long as the selected segment length is less than the total length of the section of markers

that are being generated. Otherwise the section length is adjusted to complete this section of the generated profile.

The classified sections are then ordered according to the given marker indices and expanded according to their assigned segment lengths. The $\log_2$ ratios are generated based on their corresponding classifications according to the method introduced by Willenbrock and Fridlyand (2005). This set of aCGH data can be thought of as a single chromosome and the method repeated to generate other chromosomes with different structures.

For this simulated data, the segmented means and classifications from 74 serous ovarian cancer patients are used. The probability that defines how likely a region is to receive its associated classification is drawn from a uniform distribution defined from 0.7 to 0.95 for each simulation. Probabilities for loss, gain, and no change are calculated in each section of generated data as mentioned earlier.

Choice of Profiles

Generally, most aCGH data across individuals is quite similar; relatively few portions of the genome are altered. We create two profiles to reflect this tendency; 15% of the markers in the first and 5% of the markers in the second are set up for likely alteration. Both profiles will contain 3,000 markers. For profile 1, markers 1–150 form a region of likely loss and 1501–1800 form a region of likely gain. For profile 2, markers 601–700 and 2401–2450 form regions of likely loss, with no regions of likely gain. Based on the empirical distributions of segment lengths for regions of loss, gain, and no change from 74 serous ovarian cancer patients, these segment lengths are fairly typical and seem reasonable relative to the total length of one sample. While larger regions of alteration may create more obviously distinct profiles, the goal of defining these profiles is to investigate how well clustering methods distinguish profiles based on relatively few alterations. For the profiles defined here, none of the regions with tendencies toward alteration overlap across profiles.

5.2 Simulation Set-Up

A total of 1,000 simulations are run for the HC, WECCA, and RDU methods. Each iteration generates aCGH data based on specifically defined true profiles, runs the three clustering methods on the simulated data, and saves the cluster assignments for two groups. For the RDU method, simulations are run using $freq.arrays = 2$. The p_w parameter is allowed to vary, producing cluster assignments corresponding to $p_w = 0.5, 0.65$, and 0.8. These are marginal probabilities of alteration that specify the first rule for a marker's potential inclusion in a region of recurrent copy number alteration. Lower probabilities for p_w are less stringent and typically allow the pREC-S algorithm to find more regions, which could result in larger clusters with greater within-cluster variability. Higher probabilities for p_w are intended for finding smaller subsets of profiles with a high degree of similarity. By default, the first component of RDU fits a six-state model using one chain with 10,000 iterations following a burn-in period of 10,000.

While it is ideal to run the clustering methods on simulated data that reflects typical aCGH profile lengths with enough samples (synonymous with patients in the data analysis) to consider more than two clusters, computing time poses some practical limitations. The limits of this simulation study are determined by RDU, as it is the slowest of the three methods under consideration.

We consider two hours to be a workable running time for RDU to cluster a single set of simulated data. As the first component of the RDU method requires the bulk of the entire method's running time, the RJaCGH function was timed for randomly selected subsets of aCGH data, with varying individual samples and profile lengths. From the data subsets that finished in about two hours (see Table 5.1), we select ten samples of 3,000 markers as the size for the simulated data. This allows for 5 sample profiles to be generated from each of two aCGH profiles. The number of markers is assumed to be large enough to characterize two different aCGH profiles.

Table 5.1: Selected running times for the RJaCGH function with varying numbers of samples and markers.

Samples	Markers	Time (min)
5	10,000	121
10	3,000	105
15	1,000	102
20	500	144

Using ten samples of 3,000 markers, ten simulations of HC finish in about 8.7 seconds, and ten simulations of WECCA finish in about 3 minutes. Ten simulations of all three methods run for 20 hours and 52 minutes. These times illustrate the lack of speed for RDU relative to both HC and WECCA.

5.3 Results

Because each of 1,000 simulations cluster ten aCGH samples, five from each known profile, a total of 10,000 samples are generated (5,000 from profile 1 and 5,000 from profile 2). The misclassification tables in Table 5.2 show counts of the number of profiles assigned by the clustering method (rows) in the context of the correct cluster profiles (columns) from which they are generated. HC is the method that correctly clusters the most profiles (see Table 5.2a). WECCA correctly clusters almost as many profiles to Cluster 1 as HC, but it also assigns Cluster 1 to more than 1,000 profiles that are generated from Cluster 2 (see Table 5.2(b)). Tables 5.2(c)–(e) indicate that RDU does not perform well. Regardless of p_w, RDU tends to assign samples to Cluster 1 more often than to Cluster 2. This pattern is more pronounced as p_w increases.

Rather than simply reporting the percentage of agreement between the known clusters and the method assignments, Cohen's kappa adjusts for the proportion of agreement that is expected to occur by chance. Each method's cluster assignments for the simulated data are compared to the known profiles from which the data were generated. Agreement refers to a method's correct cluster assignment. Cohen's kappa is an appropriate measure to analyze

(a) HC

	1	2
1	4135	1823
2	865	3177

(b) WECCA

	1	2
1	4082	3098
2	918	1902

(c) RDU: $p_w = 0.5$

	1	2
1	3432	3510
2	1568	1490

(d) RDU: $p_w = 0.65$

	1	2
1	3683	3653
2	1317	1347

(e) RDU: $p_w = 0.8$

	1	2
1	4548	4035
2	452	965

Table 5.2: Tabulated results for each clustering method, including a varying probability criterion, p_w, for RDU. Rows represent the clusters assigned to generated profiles by the indicated clustering method. Columns represent the two known aCGH profiles, with 5,000 profiles generated from each. Table totals are 10,000 cluster assignments.

these results for the following reasons. As the known profiles were designed to reflect the differences between two independent aCGH samples, the cluster categories 1 and 2 may be considered independent. In the simulation context, both the generated profiles and clustering assignments are mutually exclusive and exhaustive. Also, each method assigns samples to a cluster without any knowledge of the underlying profiles, and incorrect cluster assignments are regarded equally.

Cohen's kappa is calculated by

$$\kappa = \frac{p_o - p_c}{1 - p_c},$$

where p_o is the proportion of observed agreement and p_c is the proportion of chance agreement. Coefficient estimates, standard errors, and 95% confidence intervals are given in Table 5.3. Excluding chance-correct classifications, HC correctly clusters 46.2% of the simulated profiles. WECCA correctly clusters 19.7% of the profiles beyond expected chance agreement. $\text{RDU}_{0.5}$ and $\text{RDU}_{0.65}$ are no different than randomly assigning profiles to clusters. While $\text{RDU}_{0.8}$ shows some improvement by clustering between 8.2% and 12.3% of the profiles correctly (chance correctness excluded), it is still too low to be useful when compared with HC and WECCA.

Method	κ	s_κ	κ_l	κ_u
HC	0.4624	0.0121	0.4387	0.4861
WECCA	0.1968	0.0109	0.1754	0.2182
$\text{RDU}_{0.8}$	0.1026	0.0105	0.0820	0.1232
$\text{RDU}_{0.65}$	0.0060	0.0100	-0.0137	0.0257
$\text{RDU}_{0.5}$	-0.0156	0.0099	-0.0350	0.0038

Table 5.3: Estimates, standard errors, and 95% confidence limits for Cohen's kappa are computed from the tabulated cluster assignments from 1,000 simulations.

CHAPTER 6

CONCLUSIONS

Hierarchical clustering produces clusters in the data analysis that are associated with progression-free survival and cancer stage number, while the WECCA clusters show no association to clinical variables. Although many patients are similarly clustered in this analysis, HC differentiates patients in a more clinically relevant way. Clustering patients based on their observed $\log_2$ ratios seems to do better than clustering patients based on called data. Of course, these conclusions are specific to the ovarian cancer data that is analyzed in chapter 4.

In the simulation results, Cohen's kappa for HC is more than twice that of the kappa coefficient for WECCA, again suggesting that clustering improves by using $\log_2$ ratios rather than called data. Both HC and WECCA are superior to $\text{RDU}_{0.8}$, which is only slightly better than random chance assignment to clusters. Even though the kappa coefficient for HC (0.4642) shows the most correct clustering beyond random chance of the three methodologies considered, there is clearly room for improvement in clustering aCGH data.

Not surprisingly, $\text{RDU}_{0.5}$ and $\text{RDU}_{0.65}$ do not cluster aCGH samples any better than random assignment. Lower p_w parameters result in larger subsets of samples that meet the probability requirement for a region of alteration. However, some of these samples are almost just as likely not to follow this particular pattern of alteration for the markers in the region. Thus clustering on the similarity matrix containing the number of common regions (which were determined from lower probabilities) actually forms clusters of samples with profiles that may not really be similar at all in the regions of interest.

Naturally, $\text{RDU}_{0.8}$ shows improved clustering performance over $\text{RDU}_{0.5}$ and $\text{RDU}_{0.65}$, but it still is not good, only correctly clustering 10% of samples beyond chance correctness.

It is possible that higher p_w parameters (0.85, 0.9, or 0.95) may cause improved clustering performance, in which case, RDU may be comparable with WECCA. However, these parameters are not considered in this project. If this method is applied for clustering, p_w should not be lower than 0.8.

With regard to the simulations, another factor that could be explored is how the results are affected by known profiles that are defined with different likely regions of alteration. Excluding RDU from future aCGH simulation studies will allow for the inclusion of more than two underlying profiles so that cluster assignments for more than two groups can be considered. In this case, the true profiles could also include more markers, allowing for the inclusion of longer (still typical) likely regions of alteration.

HC and WECCA are both viable clustering methods; however, HC is markedly faster than WECCA. Given the long running time for RDU, as well as its difficulty producing cluster assignments for large data sets, it is not practical for anything more than small-scale analyses. Even then, HC or WECCA seem to be superior in terms of the accuracy of cluster assignments, as well as running time. Perhaps computational improvements can render RDU more useful in the future.

Overall, hierarchical clustering is recommended over WECCA and the Rueda/Diaz-Uriarte method. Aside from its short implementation time, clinical relevance and clustering accuracy support hierarchical clustering as a reasonable method for clustering aCGH data. Certainly, the development of other clustering methodologies for aCGH data may provide useful alternatives to hierarchical clustering in a variety of clinical settings.

BIBLIOGRAPHY

Alexandridis, R., Lin, S., and Irwin, M. (2004), "Class discovery and classification of tumor samples using mixture modeling of gene expression data," *Bioinformatics*, 20, 2546–2552.

Beroukhim, R., Getz, G., Nghiemphu, L., Barretina, J., Hsueh, T., Linhart, D., Vivanco, I., Lee, J. C., Huang, J. H., Alexander, S., Du, J., Kau, T., Thomas, R. K., Shah, K., Soto, H., Perner, S., Prensner, J., Debiasi, R. M., Demichelis, F., Hatton, C., Rubin, M. A., Garraway, L. A., Nelson, S. F., Liau, L., Mischel, P. S., Cloughesy, T. F., Meyerson, M., Golub, T. A., Lander, E. S., Mellinghoff, I. K., and Sellers, W. R. (2007), "Assessing the significance of chromosomal aberrations in cancer: Methodology and application to glioma," in *Proceedings of the National Academy of Sciences: USA*, Vol. 104, pp. 20007–20012.

Braun, J. V., and Mueller, H.-G. (1998), "Statistical methods for DNA sequence segmentation," *Statistical Science*, 13, 142–162.

Cheng, C., Kimmel, R., Neiman, P., and Zhao, L. P. (2003), "Array rank order regression analysis for the detection of gene copy-number changes in human cancer," *Genomics*, 82, 122–129.

Cohen, J. (1960), "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, 20, 37.

Diskin, S. J., Eck, T., Greshock, J., Mosse, Y. P., Naylor, T., Stoeckert, C. J., Weber, B. L., Maris, J. M., and Grant, G. R. (2006), "STAC: A method for testing the significance of DNA copy number aberrations across multiple array-CGH experiments," *Genome Research*, 16(9), 1149–1158.

Engler, D. A., Gupta, S., Nitta, M., Growdon, W. B., Sergent, P. A., Drapkin, R. I., Baker, S. F., Gross, J., Kuo, W.-L., Karlan, B. Y., Rueda, B. R., Orsulic, S., Gray, J. W., and Mohapatra, G. (2010), "Genome wide DNA copy number analysis of serous type ovarian carcinomas identifies genetic markers predictive of clinical outcome," In preparation.

Engler, D. A., Mohapatra, G., Louis, D. N., and Betensky, R. A. (2006), "A pseudolikelihood approach for simultaneous analysis of array comparative genomic hybridizations," *Biostatistics*, 7, 399–421.

Everitt, B. S., Landau, S., and Leese, M. (2001), *Cluster Analysis* (4th ed.), New York, NY: Arnold.

Freeman, J. L., Perry, G. H., Feuk, L., Redon, R., McCarroll, S. A., Altshuler, D. M., H, A., Jones, K. W., Tyler-Smith, C., Hurles, M. E., Carter, N. P., Scherer, S. W., and Lee, C. (2006), "Copy Number Variation: New Insights in Genome Diversity," *Genome Research*, 16, 949–961.

Fridlyand, J., Snijders, A. M., Pinkel, D., Albertson, D. G., and Jain, A. N. (2004), "Hidden Markov models approach to the analysis of array CGH data," *Journal of Multivariate Analysis*, 90, 132–153.

Guha, S., Li, Y., and Neuberg, D. (2008), "Bayesian hidden markov modeling of array cgh data," *Journal of the American Statistical Association*, 103, 485–497.

Hackett, C. S., Hodgson, J. G., Law, M. E., Fridlyand, J., Osoegawa, K., de Jong, P. J., Nowak, N. J., Pinkel, D., Albertson, D. G., Jain, A., Jenkins, R., Gray, J. W., and Weiss, W. A. (2003), "Genome-wide array CGH analysis of murine neuroblastoma reveals distinct genomic aberrations which parallel those in human tumors," *Cancer Research*, 63, 5266–5273.

Hupe, P., Stransky, N., Thiery, J. P., Radvanyi, F., and Barillot, E. (2004), "Analysis of array CGH data: from signal ratio to gain and loss of DNA regions," *Bioinformatics*, 20, 3413–3422.

Iafrate, A. J., Feuk, L., Rivera, M. N., Listewnik, M. L., Donahoe, P. K., Qi, Y., Scherer, S. W., and Lee, C. (2oo4), "Detection of large-scale variation in the human genome," *Nature Genetics*, 36, 949–951.

Jong, K., Marchiori, E., van der Vaart, A., Ylstra, B., Meijer, G., and Weiss, M. (2003), "Chromosomal breakpoint detection in human cancer," *Lecture Notes in Computer Science: Springer-Verlag, Berlin*, 2611, 54–65.

Liu, J., Mohammed, J., Carter, J., Ranka, S., Kahveci, T., and Baudis, M. (2006), "Distance-based clustering of CGH data," *Bioinformatics*, 22, 1971–1978.

Lockwood, W. W., Chari, R., Chi, B., and Lam, W. L. (2006), "Recent advances in array comparative genomic hybridization technologies and their applications in human genetics," *European Journal of Human Genetics*, 14, 139–148.

Massion, P. P., Kuo, W.-L., Stokoe, D., Olshen, A. B., Treseler, P. A., Chin, K., Chen, C., Polikoff, D., Jain, A. N., Pinkel, D., Albertson, D. G., Jablons, D. M., and Gray, J. W. (2002), "Genomic copy number analysis of non-small cell lung cancer using array comparative genomic hybridization: implications of the phosphatidylinositol 3-kinase pathway," *Cancer Research*, 62, 3636–3640.

McLachlan, G. J., Bean, R. W., and Peel, D. (2002), "A mixture model-based approach to the clustering of microarray expression data," *Bioinformatics*, 18, 413–422.

Olshen, A. B., and Venkatraman, E. S. (2002), "Change-point analysis of array-based comparative genomic hybridization data," in *Proceedings of the Joint Statistical Meetings, American Statistical Association*, Alexandria, VA, pp. 2530–2535.

Olshen, A. B., Venkatraman, E. S., Lucito, R., and Wigler, M. (2004), "Circular binary segmentation for the analysis of array-based DNA copy number data," *Biostatistics*, 5, 557–572.

Pan, W., and Shen, X. (2007), "Penalized Model-Based Clustering With Application to Variable Selection," *Journal of Machine Learning Research*, 8, 1145–1164.

Picard, F., Robin, S., Lavielle, M., Vaisse, C., and Daudin, J. J. (2005), "A statistical approach for array CGH data analysis," *BMC Bioinformatics*, 6, 27.

Pollack, J. R., Srlie, T., Perou, C. M., Rees, C. A., Jeffrey, S. S., Lonning, P. E., Tibshirani, R., Botstein, D., Brresen-Dale, A.-L., and Brown, P. O. (2002), "Microarray analysis reveals a major direct role of DNA copy number alteration in the transcriptional program of human breast tumors," in *Proceedings of the National Academy of Sciences*, Vol. 99, pp. 12963–12968.

Polzehl, J., and Spokoiny, V. (2000), "Adaptive weights smoothing with applications to image restoration," *The Journal of the Royal Statistical Society, Series B*, 62, 335–54.

Rueda, O. M., and Diaz-Uriarte, R. (2009), "Detection of recurrent copy number alterations in the genome: taking among-subject heterogeneity seriously," *BMC Bioinformatics*, 10, 308.

Shah, S. P., Cheung, K. J., Johnson, N. A., Alain, G., Gascoyne, R. D., Horsman, D. E., Ng, R. T., and Murphy, K. P. (2009), "Model-based clustering of array CGH data," *Bioinformatics*, 25(12), i30–38.

Storey, J. D. (2002), "A direct approach to false discovery rates," *Journal of the Royal Statistical Society*, 64, 479–498.

van de Wiel, M. A., and van Wieringen, W. N. (2007), "CGHregions: dimension reduction for array CGH data with minimal information loss," *Cancer Informatics*, 2, 55–63.

van Wieringen, W. N. N., van de Wiel, M. A. A., and Ylstra, B. (2008), "Weighted clustering of called array CGH data," *Biostatistics*, 9, 484–500.

Veltkamp, R. C., and Latecki, L. J. (2006), "Properties and Performance of Shape Similarity Measures," in *Proceedings of IFCS 2006 Conference: Data Science and Classification*.

Wang, P., Kim, Y., Pollack, J., Narasimhan, B., and Tibshirani, R. (2005), "A method for calling gains and losses in array CGH data," *Biostatistics*, 6, 45–58.

Wilhelm, M., Veltman, J. A., Olshen, A. B., Jain, A. N., Moore, D. H., Presti, J., Kovacs, G., and Waldman, F. M. (2002), "Array-based comparative genomic hybridization for the differential diagnosis of renal cell cancer," *Cancer Research*, 62, 957–960.

Willenbrock, H., and Fridlyand, J. (2005), "A comparison study: applying segmentation to array CGH data for downstream analyses," *Bioinformatics*, 21, 4084–4091.

APPENDICES

APPENDIX A

SOURCE CODE

The following code contains all of the R functions that are required for running CGHregions and the WECCA method, as well as for generating aCGH data (code provided by David Engler) in the simulations.

A.1 CGH Regions

```
#find basepair pos
findbp <- function(reg,bppos)
        {
        st <- bppos[reg[1]]
        end <- bppos[reg[2]]
        c(st,end)
        }


#find chromosome
findchr <- function(reg,chr)
        {
        chr[reg[1]]
        }


#impute missings
imp_missing <- function(cghdataraw)
{
nrsm <- nrow(cghdataraw)
imp <- function(smkol)
{
smkol[1]<- ifelse(is.na(smkol[1]),0, smkol[1])
for (i in (2:nrsm))
{
smkol[i]<-ifelse(is.na(smkol[i]),smkol[i-1],smkol[i])
}
smkol
}
return(apply(cghdataraw,2,imp))
}



#distance function
# modified name so that R's dist function is not overwritten -SFB
dist1 <- function(c1,c2)
        {
        d <- abs(c1-c2)
        apply(d,1,sum)
        }


dist2 <- function(c1,c2)
        {
        c3 <- c2
        if (nrow(c1) >1) for (i in 1:(nrow(c1)-1)) c3<-rbind(c3,c2)
        d <- abs(c1-c3)
        apply(d,1,sum)
        }
```

```
#function insbr inserts breaks per region based on distance larger than c
#some problems with null-regions: therefore always insert c(0,0)
#Also determine mono-regions: distance to both neighbours > 3*c/4.

insbr <- function(reg,c,ct)
         {
         datareg <- ct[reg[1]:reg[2],-1]
         #print(dim(datareg))
         datareg2 <- rbind(datareg,datareg[nrow(datareg),])[-1,]
       totdist <- dist1(datareg,datareg2)
         totdist2 <- append(totdist,0,0)[-length(totdist+1)]
         indc <- ct[reg[1]:reg[2],1]
         totdisti <- cbind(indc,totdist,totdist2)
         newbr <- rbind(c(0,0,0),totdisti[totdist>c,])
         newbr1 <- as.vector(newbr[,1])
         mono <- rbind(c(0,0,0),totdisti[totdist>floor(3*c/4)&totdist2>floor(3*c/4),])
         mono1 <- as.vector(mono[,1])[-1]
         mono2 <- c(mono1,mono1-1)
         sort(unique(c(newbr1,mono2)))
         }

#jump size, this function computes the jump-sizes, it returns the
#absolute clone index and its index within the region
jump <- function(reg,ctdat)
         {
         datareg <-  td(reg,ctdat)
         ind <- tdind(reg,ctdat)
         datareg2 <- rbind(datareg,datareg[nrow(datareg),])[-1,]
       totdist <- dist1(datareg,datareg2)
         maxim <- max(totdist)
       selectmax <- cbind(ind,totdist)[totdist==maxim,1]
         selectmax
         }



#concatenate regions
concat <- function(reg,c,ctdat,breakchr)
{
regionsst <- c(1)
regionsend <- c()
#regionstart <- reg[1,1]
for (i in 1:(nrow(reg)-1))
{
#i<-2
regionstart <- reg[i+1,1]
c1 <- td(c(reg[i,2],reg[i,2]),ctdat)
c2 <- td(c(reg[i+1,1],reg[i+1,1]),ctdat)
#print(list(i,c1,c2,ctdat[8050:8065,]))
#print(i)
if (dist1(c1,c2) > c | is.element(regionstart,breakchr))
    #do not concatenate if distance is too large or chr. border
{
regionsst <- c(regionsst,regionstart)
regionsend <- c(regionsend,reg[i,2])
}
}
#regionsst <- c(regionsst,regionstart)
regionsend <- c(regionsend,reg[nrow(reg),2])
#regionsst
cbind(regionsst,regionsend)
}

#take datarows from counts data
td <- function(reg,ct)
      {
     ct[ct$ind<=reg[2] & ct$ind>= reg[1],][,-1]
      }
```

```
ntd <- function(reg,ct) ifelse(!is.null(dim(td(reg,ct))),nrow(td(reg,ct)),1)

tdind <- function(reg,ct)
      {
     ct[ct$ind<=reg[2]&ct$ind>= reg[1],][,1]
      }


#check basic condition, check it on first and last and 6 equally spaced clones maximally
checkcond <- function(reg,c,ctdat)
         {
         ctr <- td(reg,ctdat)
         ncl <- nrow(ctr)
         if (ncl > 10)
         {
         skip <- floor(ncl/6)
         takerow <- c(seq(1,ncl,skip),ncl)
         }
         else {takerow <- 1:ncl}
         ctreg <- ctr[takerow,]
         ctuni <- unique(ctreg)
         luni <- dim(ctuni)[1]
         i<-1
         cond <- 0
       while (i < (luni-1) & cond == 0)
         {
         for (j in luni:(i+1))
         {
         totdist <- dist1(ctuni[i,],ctuni[j,])
         if (totdist>c) cond <- 1
         }
         i<-i+1
         }
         cond
         }

#function for unique confs within region
check <- function(reg,c)
         {
         ncl <- reg[2]-reg[1] + 1
         if (ncl > 10)
         {
         skip <- floor(ncl/6)
         takerow <- c(seq(1,ncl,skip),ncl)
         }
         else {takerow <- 1:ncl}
         ctreg <- (counts[reg[1]:reg[2],-1])[takerow,]
         ctuni <- unique(ctreg)
ctuni
}

#compute right-gradient
rightgrad <- function(cl,ct,nro)
       {
         mincl <- nro-cl+1
         if (mincl==1) rgrad <- 0 else
         {
         minim <- min(5,mincl)
         cllst <- ct[(cl+1):(cl+minim-1),]
         distvec <- dist2(cllst,ct[cl,])
         weightvec <- 1:(minim-1)
         rgrad <- (distvec %*% weightvec)/sum(weightvec)
         rgrad
         }
         }
#rightgrad(84,td(newreg[1,],countsdel),142)

wh <- function(x,i) which(x==i)
```

```
gradients <- function(reg,index,ctdat)
             {
             ctreg <- td(reg,ctdat)
             ctind <- tdind(reg,ctdat)
             clones <- sapply(index,wh,x=ctind)
             nr <- nrow(ctreg)
             grads <- sapply(clones,rightgrad,ct=ctreg,nro=nr)
             maxim<-max(grads)
           selectmax <- cbind(index,grads)[grads==maxim,1]
             selectmax
             }
#this function is only applied if both the jump-size and the gradient
#for two clones are equal. Currently not re-computed!!!
dist2mid <-  function(reg,index,ctdat) {
             ncl <- nrow(td(reg,ctdat))
             nmid <- (ncl+1)/2
         ctreg <- td(reg,ctdat)
             ctind <- tdind(reg,ctdat)
             clones <- sapply(index,wh,x=ctind)
             di2mi <- abs((1:ncl) - nmid +0.25)
             cl <- which.min(di2mi[clones])
             index[cl]
             }
#defines new breaks
breek <- function(reg,ctdat)
         {
         jee <- jump(reg,ctdat)
         if (length(jee)==1)
         br <- jee[1]
         else
         {
         grads <- gradients(reg,jee,ctdat)
         if (length(grads)==1) br <- grads[1]
       else
        {
        br <- dist2mid(reg,grads,ctdat)
        }
         }
         br
         }

#compute gradient
grad <- function(cl,ct,nro)
       {
         mincl <- nro-cl+1
         if (cl==1) 0
         else
         {
         minim <- min(5,cl)
         cllst <- ct[(cl-minim+1):(cl-1),]
         distvec <- dist2(cllst,ct[cl,])
         weightvec <- (minim-1):1
         lgrad <- (distvec %*% weightvec)/sum(weightvec)
         if (mincl==1) rgrad <- 0 else
         {
         minim <- min(6,mincl)
         cllst <- ct[(cl+1):(cl+minim-1),]
         distvec <- dist2(cllst,ct[cl,])
         weightvec <- 1:(minim-1)
         rgrad <- (distvec %*% weightvec)/sum(weightvec)
         }
         max(lgrad,rgrad)
         }
         }

#COMPUTE unique signatures within region and their frequency
```

```
distmat <- function(uni)
{
el <- nrow(uni)
cmat <- array(NA,c(el,el))
for (i in (1:el))
{
for (j in (1:el))
{
ifelse(j<i,cmat[i,j]<-cmat[j,i],cmat[i,j]<-dist1(uni[i,],uni[j,]))
}
}
cmat
}

countrow <- function(unireg,datareg)
{
testje <- function(unireg,reg1)
{
min(reg1==unireg)
}
filt <- apply(datareg,1,testje,unireg=unireg)
datarel <- cbind(filt,datareg)[filt==1,-1]
ifelse(is.null(dim(datarel)),1,nrow(datarel))
}

#this is the SLOW function!!!!!
whichsign <- function(reg,ctdat)
{
datareg <-td(reg,ctdat)
uni <- unique(datareg)
if (!is.null(dim(uni)))
{
afst <- distmat(uni)%*%apply(uni,1,countrow,datareg=datareg)
#afst <- distmat(uni)
#afst <- apply(uni,1,countrow,datareg=datareg)
minim <- which.min(afst[,1])
return(list(uni[minim,],afst[minim,1]))
#return(apply(uni,1,countrow,datareg=datareg))
}
else return(list(uni,0))
}

#this is the FAST function!
whichsign2 <- function(reg,ctdat,levels)
{
datareg <-td(reg,ctdat)
uni <- unique(datareg)
if (!is.null(dim(uni)))
{
#nclone <- nrow(datareg)
cummat <- apply(datareg,2,countlevels,levels=levels)
afst <- apply(uni,1,dm,cm = cummat,levels=levels)
minim <- which.min(afst)
return(list(uni[minim,],afst[minim]))
}
else return(list(uni,0))
}


countlevels <- function(datacol, levels)
{
sapply(levels,whl <- function(x,i) length(which(x==i)),x=datacol)
}

dm <- function(uniseq,cm,levels)
{
ele <- length(levels)+1
all <- rbind(cm,uniseq)
```

```
concol <-function(colvec,el=ele,lev)
    {
    elem <- colvec[el]
    dt <- colvec[1:(el-1)]
    crossprod(dt,abs(elem-lev))
    }
sum(apply(all,2,concol,lev=levels))
}


regionact <- function(reg)
{
datareg <- countsnordel[reg[1]:reg[2],]
return(sum(apply(datareg,1,sum))/nrow(datareg))
}

regionact2 <- function(reg,ctdat)
{
datareg <- td(reg,ctdat)
countf <- function(dr)
{
length(dr[dr!=normstate])
}
return(sum(apply(datareg,1,countf))/nrow(datareg))
}




##################END OF FUNCTIONS
#setwd("D:\Documenten en settings\\Mark\\Mijn documenten\\CGHRegions")
#setwd("D:\\VUData\\CGHdata")
#setwd("D:\\Hub\\DemoData")
#getwd()
#Format first column: chr nr; 2nd: Mb position; rest: loss, normal, gain, ampl
#Missing data are not allowed.


#initilisation: inserts breaks at chromosome borders;
#breakpoint is index of first clone in new region
deterreg <- function(CGHdata,crit,ncolm,normstate,levels)
{
chr <- as.numeric(CGHdata[,1])
numcl <- length(chr)
ind <- 1:numcl
chrsh <- append(chr,1,0)
chrsh <- chrsh[-length(chrsh)]
dif <- chr-chrsh
difind <- cbind(ind,dif)


counts <- cbind(ind,CGHdata[,-(1:2)])
counts <- counts[,1:(ncolm+1)]

nchr <- length(unique(chr))

if (nchr > 1)
    {
    if (nchr>2) breaks <- difind[dif != 0,][,1] else breaks <- difind[dif != 0,][1]
    regions <- cbind(append(breaks,1,0),append(breaks-1,numcl,length(breaks)))
    allbreaks <- as.vector(apply(regions,1,insbr,c=crit,ct=counts))
    newb <- c()
    for (i in 1:length(allbreaks))
    {
    tp <- allbreaks[[i]]
    newb <- c(newb,tp[tp != 0]+1)
    }
    }
```

```
if (nchr <= 1)
      {
      breaks <- c()
      regions <- c(1,numcl)
      allbreaks <- insbr(regions,crit,counts)
    newb <- c()
    for (i in 1:length(allbreaks))
    {
    tp <- allbreaks[[i]]
    newb <- c(newb,tp[tp != 0]+1)
    }

    }


#merge with old breaks
allb <- sort(c(breaks,newb))
regions <- cbind(append(allb,1,0),append(allb-1,numcl,length(allb)))
del <- regions[(regions[,1]-regions[,2])==0,1]
countsdel <- if (length(del)>0) counts[-del,] else counts
if(nrow(regions)>1) { #adapted 23/08
        regions2 <- regions[(regions[,1]-regions[,2])<0,]
        if(nrow(regions2) > 1) regions3 <- concat(regions2,crit,countsdel,breaks) else regions3 <- regions2
        } else {
        regions3 <- regions
        }
allcond <- apply(regions3,1,checkcond,c=crit,ctdat=countsdel)

if(max(allcond)==0) {selreg <- regions3} else #only apply gradient ruler if necessary
{
violreg <- regions3[which.max(allcond),]
regions3 <- rbind(regions3,violreg)
selreg <- c()
stop <- 0

#recursive application of gradient ruler
while (stop==0)
{
allcond <- apply(regions3,1,checkcond,c=crit,ctdat=countsdel)
#note '0' indicates that region satisfies criterion
selreg0 <- cbind(regions3,allcond)
selreg <- rbind(selreg,selreg0[allcond==0,-3])
newreg <- selreg0[allcond==1,-3]

if (!is.null(dim(newreg)) && dim(newreg)[1] != 0)
{
newbr<-apply(newreg,1,breek,ctdat=countsdel)
#insert newbr into newreg
lbr <- length(newbr)
newreg2 <- c()
for (i in (1:(lbr-1)))
{
reg1 <- c(newreg[i,1],newbr[i])
reg2 <- c(newbr[i]+1,newreg[i,2])
newreg2 <- rbind(newreg2,c(newreg[i,1],newbr[i]),c(newbr[i]+1,newreg[i,2]))
}
regions3<-newreg2
regions3 <- rbind(regions3,violreg)
}
if (is.null(dim(newreg)) || dim(newreg)[1] == 0) {
    stop <- 1
}
}
}
#DELETE MONO-REGIONS FROM SELREG

selnew <- selreg[(selreg[,1]-selreg[,2])!=0,]
```

```
#selects ACTIVE regions, assumes order loss, normal, gain.

#seqnone <- seq(colnor,length(counts[1,-1]), by = nclass)
#countsnordel <- counts[,-1][,-seqnone]


sortedact <- sort(apply(selnew,1,regionact2,ctdat=countsdel),index.return=TRUE,decreasing=TRUE)
indicesActReg <- sortedact$ix[1:ceiling(0.25*nrow(selnew))]
ActReg25perc <- selnew[indicesActReg,]
if(is.null(nrow(ActReg25perc))) {ActReg25perc <-rbind(ActReg25perc,ActReg25perc)}
   # create two rows for rare case in which ActReg25perc consists of only 1 row

#ASSUMES  missings are absent!!
nsam <- ncolm
nclone <- sum(apply(ActReg25perc,1,function(y){y[2]-y[1]+1}))
all<-apply(ActReg25perc,1,whichsign2,ctdat=countsdel,levels=levels)
avedist <- sum(as.vector(lapply(all,function(x) {x[[2]]}),mode="double"))/(nclone*nsam)
return(list(avedist,selnew,countsdel))
#return(list(ActReg25perc,all))

}
```

A.2 WECCA Functions

```
#############################################################################################
#
# SPECIFICS
#
# Name            : WECCA.functions.R
# Authors         : Wessel N. van Wieringen
# Email             : wvanwie@few.vu.nl
# Last updated : 25 - 08 - 2007
# Description        : R-functions for WECCA: weighted clustering of called aCGH data.
#                         It corresponds to the paper "Weighted clustering of called aCGH data" (2007)
#                         by Wessel N. van Wieringen, Mark A. van de Wiel, Bauke Ylstra.
#
# Amsterdam, August 25, 2007, Wessel N. van Wieringen, wvanwie@few.vu.nl
#
#############################################################################################

#############################################################################################
#
# This function is a wrapper, merely selecting the right clustering method.
# Then, it produces the dendrogram with the selected method.
#
#############################################################################################

WECCA <- function(data,a,sim,link,strict,corr){

        ###########################################################################################
        #
        # Function that transforms a hierarchy into a dendogram.
        #
        ###########################################################################################

        hierarchy2dendrogram <- function(hierarchy,order){
                    tree <- list(merge = hierarchy[,1:2], height= hierarchy[,3], method=NULL,
                                                call = match.call(), order = order, dist.method = "whatever")
                class(tree) <- "hclust"
                    return(tree)
        }


        ###########################################################################################
        #
```

```
# Function that constructs the heights and merging pattern,
# with the weighted agreement similarity and average linkage.
#
##############################################################################################

merge.agree.avlink.weight <- function(X,a,corr){
        clusters <- cluster.list(dim(X)[2]-1)
        D0 <- agree.comlink.weight.dist(X,clusters,a)
        if (corr){ D0 <- matrix(as.numeric(lapply(as.numeric(D0),
          function(x) max((x-1/a)/(1-1/a),0))),ncol=dim(D0)[1]) }
        D0[upper.tri(D0)] <- t(D0)[upper.tri(t(D0))]
        diag(D0) <- 1
        hc <- hclust(as.dist(1-D0),method="average")
        hc
}


##############################################################################################
#
# Function that constructs the heights and merging pattern,
# with the weighted agreement similarity and complete linkage
# in the traditional sense.
#
##############################################################################################

merge.agree.comtradlink.weight <- function(X,a,corr){
        clusters <- cluster.list(dim(X)[2]-1)
        D0 <- agree.comlink.weight.dist(X,clusters,a)
        if (corr){ D0 <- matrix(as.numeric(lapply(as.numeric(D0),
                                    function(x) max((x-1/a)/(1-1/a),0))),ncol=dim(D0)[1]) }
        D0[upper.tri(D0)] <- t(D0)[upper.tri(t(D0))]
        diag(D0) <- 1
        hc <- hclust(as.dist(1-D0),method="complete")
        return(hc)
}


##############################################################################################
#
# Function that constructs the heights and merging pattern,
# with the weighted agreement similarity and complete linkage
# in the traditional sense.
#
##############################################################################################

merge.conc.comtradlink.weight <- function(X,a,corr,strict){
        clusters <- cluster.list(dim(X)[2]-1)
      D0 <- conc.comlink.weight.dist(X,clusters,a,strict)
      if (!strict){
           if (corr){ D0 <- matrix(as.numeric(lapply(as.numeric(D0),
             function(x) max((x-((a^2-2*a+1)/(2*a^2) + 1/a^3))/(1-((a^2-2*a+1)/(2*a^2) + 1/a^3)),0))),
             ncol=dim(D0)[1]) }
        } else {
           if (corr){ D0 <- matrix(as.numeric(lapply(as.numeric(D0),
             function(x) max((x-(a^2-2*a+1)/(2*a^2))/(1-(a^2-2*a+1)/(2*a^2)),0))), ncol=dim(D0)[1]) }
      }
      D0[upper.tri(D0)] <- t(D0)[upper.tri(t(D0))]
        diag(D0) <- 1
        hc <- hclust(as.dist(1-D0),method="complete")
        return(hc)
}

##############################################################################################
#
# Function that constructs the heights and merging pattern,
# with the weighted agreement similarity and average linkage.
#
##############################################################################################
```

```
merge.conc.avlink.weight <- function(X,a,corr,strict){
        clusters <- cluster.list(dim(X)[2]-1)
        D0 <- conc.comlink.weight.dist(X,clusters,a,strict)
      if (!strict){
           if (corr){ D0 <- matrix(as.numeric(lapply(as.numeric(D0),
             function(x) max((x-((a^2-2*a+1)/(2*a^2) + 1/a^3))/(1-((a^2-2*a+1)/(2*a^2) + 1/a^3)),0))),
             ncol=dim(D0)[1]) }
        } else {
           if (corr){ D0 <- matrix(as.numeric(lapply(as.numeric(D0),
             function(x) max((x-(a^2-2*a+1)/(2*a^2))/(1-(a^2-2*a+1)/(2*a^2)),0))),
             ncol=dim(D0)[1]) }
      }
      D0[upper.tri(D0)] <- t(D0)[upper.tri(t(D0))]
        diag(D0) <- 1
        hc <- hclust(as.dist(1-D0),method="average")
        return(hc)
}

dist.conc.avlink.weight <- function(X,a,corr,strict){
        clusters <- cluster.list(dim(X)[2]-1)
        D0 <- conc.comlink.weight.dist(X,clusters,a,strict)
      if (!strict){
           if (corr){ D0 <- matrix(as.numeric(lapply(as.numeric(D0),
             function(x) max((x-(a^2-2*a+3)/(2*a^2))/(1-(a^2-2*a+3)/(2*a^2)),0))), ncol=dim(D0)[1]) }
        } else {
           if (corr){ D0 <- matrix(as.numeric(lapply(as.numeric(D0),
             function(x) max((x-(a^2-2*a+1)/(2*a^2))/(1-(a^2-2*a+1)/(2*a^2)),0))), ncol=dim(D0)[1]) }
      }
      D0[upper.tri(D0)] <- t(D0)[upper.tri(t(D0))]
        diag(D0) <- 1
        return(D0)
}


##############################################################################################
#
# Function that constructs the heights and merging pattern,
# with the weighted agreement similarity and complete linkage.
#
##############################################################################################

merge.agree.comlink.weight <- function(X,a,corr){
        clusters <- cluster.list(dim(X)[2]-1)
        D0 <- agree.comlink.weight.dist(X,clusters,a)
        if (corr){ D0 <- matrix(as.numeric(lapply(as.numeric(D0),
          function(x) max((x-1/a)/(1-1/a),0))),ncol=dim(D0)[1]) }
        height <- NULL
        merge <- NULL
        for (j in 1:(dim(D0)[1]-1)){
                height <- c(height,max(D0[lower.tri(D0)]))
                if (height[length(height)] != 0){
                   ind <- which(D0 == max(D0[lower.tri(D0)]),arr.ind=TRUE)[1,]
                   new.cluster <- sort(c(clusters[[ind[1]]],clusters[[ind[2]]]))
                } else {
               ind <- which(lapply(1:length(clusters),
                                   function(i, clusters){ is.na(max(match(clusters[[i]],
                                                                          clusters[[1]]))) }, clusters )==TRUE)[1]
                   new.cluster <- sort(c(clusters[[1]],clusters[[ind]]))
                }
                for (i in 1:length(new.cluster)){ clusters[[new.cluster[i]]] <- new.cluster }
                when <- rep(0,dim(D0)[1])
                when[new.cluster] <- j
                merge <- rbind(merge,when)
                dist.vec <- NULL
                for (i in 1:dim(D0)[1]){
                    if (corr){ chance <- 1/a^(length(new.cluster)+length(clusters[i])) }
                    else { chance <- 0 }
```

```
                dist.vec <- c(dist.vec, max((prob.agree.comlink.weight(X,1,c(new.cluster),
                                        c(clusters[[i]]),a)-chance)/(1-chance),0))
              }
              dist.vec[new.cluster] <- 0
              for (i in 1:length(new.cluster)){ D0[new.cluster[i],] <- dist.vec        }
              D0[upper.tri(D0,diag=TRUE)] <- 0
       }
       return(list(merge=merge,height=height))
}


#############################################################################################
#
# Function that constructs the heights and merging pattern,
# with the weighted concordance similarity and complete linkage.
#
#############################################################################################

merge.conc.comlink.weight <- function(X,a,corr,strict){
       clusters <- cluster.list(dim(X)[2]-1)
       D0 <- conc.comlink.weight.dist(X,clusters,a,strict)
     if (!strict){
         if (corr){ D0 <- matrix(as.numeric(lapply(as.numeric(D0),
           function(x) max((x-((a^2-2*a+1)/(2*a^2) + 1/a^3))/(1-((a^2-2*a+1)/(2*a^2) + 1/a^3)),0))),
           ncol=dim(D0)[1]) }
       } else {
          if (corr){ D0 <- matrix(as.numeric(lapply(as.numeric(D0),
            function(x) max((x-(a^2-2*a+1)/(2*a^2))/(1-(a^2-2*a+1)/(2*a^2)),0))), ncol=dim(D0)[1]) }
     }
       height <- NULL
       merge <- NULL
       for (j in 1:(dim(D0)[1]-1)){
              height <- c(height,max(D0[lower.tri(D0)]))
              if (height[length(height)] != 0){
                 ind <- which(D0 == max(D0[lower.tri(D0)]),arr.ind=TRUE)[1,]
                 new.cluster <- sort(c(clusters[[ind[1]]],clusters[[ind[2]]]))
              } else {
             ind <- which(lapply(1:length(clusters),
          function(i, clusters){ is.na(max(match(clusters[[i]],
           clusters[[1]]))) }, clusters )==TRUE)[1]
                 new.cluster <- sort(c(clusters[[1]],clusters[[ind]]))
              }
              for (i in 1:length(new.cluster)){ clusters[[new.cluster[i]]] <- new.cluster }
              when <- rep(0,dim(D0)[1])
              when[new.cluster] <- j
              merge <- rbind(merge,when)
              dist.vec <- NULL
              for (i in 1:dim(D0)[1]){
                  if (!strict){
                     if (corr){ chance <- 2*((a-1)/(2*a))^(length(new.cluster)+length(clusters[i]))
                       + 1/a^(2*(length(new.cluster)+length(clusters[i]))-1) }
                     else { chance <- 0 }
                } else {
                     if (corr){ chance <- 2*((a-1)/(2*a))^(length(new.cluster)+length(clusters[i]))}
                     else { chance <- 0 }
              }
                dist.vec <- c(dist.vec, max((prob.conc.comlink.weight(X,1,c(new.cluster),
                                        c(clusters[[i]]),a,strict)-chance)/(1-chance),0))
              }
              dist.vec[new.cluster] <- 0
              for (i in 1:length(new.cluster)){ D0[new.cluster[i],] <- dist.vec        }
              D0[upper.tri(D0,diag=TRUE)] <- 0
       }
       return(list(merge=merge,height=height))
}


#############################################################################################
```

```
#
# Function that transforms the merging patterns into an efficient one and the permutation
# order for efficient plotting, and turns this into a dendogram.
#
#############################################################################################

build.dendro <- function(merge,height){
        H1 <- apply(merge,2,cummax)
        H <- NULL
        for (j in 1:dim(merge)[2]){
                H <- c(H,which(H1[,j]==min(H1[(H1[,j]>0),j]),arr.ind=TRUE)[1])
        }
        M <- (H==1)*c(1:dim(merge)[2])
        M <- -1*M[(M!=0)]
        for (j in 2:dim(merge)[1]){
                id <- (merge[j,]!=0)*c(1:dim(merge)[2])
                id <- id[(id!=0)]
                H1a <- H1[j-1,id]
                for (k1 in 1:length(id)){
                        if(H1a[k1]==0){ H1a[k1] <- -id[k1] }
                }
                M <- rbind(M,as.numeric(names(table(H1a))))
        }
        colnames(merge) <- c(1:dim(merge)[2])
        for (j in 1:(dim(merge)[1]-1)){
                merge <- merge[,order(merge[j,])]
        }
        order <- as.numeric(colnames(merge))
      mergePheight <- cbind(M,1-height)
        colnames(mergePheight) <- NULL
        rownames(mergePheight) <- NULL
        hc <- hierarchy2dendrogram(mergePheight,order)
        return(hc)
}


#############################################################################
#
# Function that constructs a list for n items equal to 1,...,n.
#
#############################################################################

cluster.list <- function(n){
        c.null <- list()
        for (i in 1:n){ c.null[[i]] <- c(i) }
        return(c.null)
}


#############################################################################################
#
# Function that constructs the concordance similarity matrix with complete linkage.
#
#############################################################################################

conc.comlink.weight.dist <- function(X,clusters,a,strict){
    dist.mat <- matrix(0,nrow=length(clusters),ncol=length(clusters))
    for (i1 in 1:(length(clusters)-1)){
      for (i2 in (i1+1):length(clusters)){
           dist.mat[i2,i1] <- prob.conc.comlink.weight(X,1,as.numeric(clusters[i1]),
                                                       as.numeric(clusters[i2]),a,strict)
        }
    }
    return(dist.mat)
}


#############################################################################################
```

```
#
# Function that constructs the agreement similarity matrix with complete linkage.
#
#############################################################################################

agree.comlink.weight.dist <- function(X,clusters,a){
    dist.mat <- matrix(0,nrow=length(clusters),ncol=length(clusters))
    for (i1 in 1:(length(clusters)-1)){
      for (i2 in (i1+1):length(clusters)){
            dist.mat[i2,i1] <- prob.agree.comlink.weight(X,1,as.numeric(clusters[i1]),
                                                  as.numeric(clusters[i2]),a)
        }
    }
    return(dist.mat)
}


#############################################################################################
#
# Function that calculates the weighted probability of agreement with complete linkage.
#
#############################################################################################

prob.agree.comlink.weight <- function(X,w.col,c1.cols,c2.cols,a){
    X1 <- matrix(X[,c1.cols+1],ncol=length(c1.cols),nrow=dim(X)[1])
    X2 <- matrix(X[,c2.cols+1],ncol=length(c2.cols),nrow=dim(X)[1])
    Xt <- cbind(X[,w.col],X1,X2)
    index <- as.numeric(factor(apply(Xt[,2:dim(Xt)[2]], 1, paste, collapse=":")))
    Xt <- cbind(Xt,index)
    Xt <- Xt[order(Xt[,dim(Xt)[2]]),]
    counter <- NULL
    for (i in 1:length(table(index))){
          counter <- c(counter, cumsum(Xt[(Xt[,dim(Xt)[2]]==i),dim(Xt)[2]])/i)
    }
    Xt <- cbind(Xt,counter,1)
    Xc <- NULL
    for (i in 1:length(table(index))){
          Xc <- rbind(Xc,c(i,sum(Xt[(Xt[,dim(Xt)[2]-2]==i),dim(Xt)[2]]),Xt[((Xt[,dim(Xt)[2]-2]==i)&
                          (Xt[,dim(Xt)[2]-1]==1)),2:(dim(Xt)[2]-3)],sum(Xt[(Xt[,dim(Xt)[2]-2]==i),1]),
                          sum((Xt[(Xt[,dim(Xt)[2]-2]==i),1])^2)))
    }
    Pa <- 0
    for (i in 1:a){
          K <- rep(i,length(c1.cols)+length(c2.cols))
          P <- matrix((Xc[,3:(length(c1.cols)+length(c2.cols)+2)]==K),
                                                ncol=(length(c1.cols)+length(c2.cols)+2)-(3-1))
          H <- apply(P,1,prod)*c(1:dim(P)[1])
          if (sum(H)>0){ Pa <- Pa + Xc[H,(length(c1.cols)+length(c2.cols)+3)] }
    }
    Pa <- Pa/sum(X[,w.col])
    return(Pa)
}


#############################################################################################
#
# Function that calculates the weighted probability of concordance with complete linkage.
#
#############################################################################################

prob.conc.comlink.weight <- function(X,w.col,c1.cols,c2.cols,a,strict){
    X <- cbind(X[,c(w.col,c1.cols+1,c2.cols+1)])
    X <- cbind(X,as.numeric(factor(apply(X[,2:dim(X)[2]], 1, paste, collapse=":"))) )
    X <- X[order(X[,dim(X)[2]]),]
    counter <- NULL
    for (i in 1:length(table(X[,dim(X)[2]]))){
          counter <- c(counter, cumsum(X[(X[,dim(X)[2]]==i),dim(X)[2]])/i)
    }
```

```
    X <- cbind(X,counter,1)
    Xc <- NULL
    for (i in 1:length(table(X[,dim(X)[2]-2]))){
          Xc <- rbind(Xc,c(i,sum(X[(X[,dim(X)[2]-2]==i),dim(X)[2]]),X[((X[,dim(X)[2]-2]==i)&
                          (X[,dim(X)[2]-1]==1)),2:(dim(X)[2]-3)],sum(X[(X[,dim(X)[2]-2]==i),1]),
                          sum((X[(X[,dim(X)[2]-2]==i),1])^2)))
    }
    m <- Xc[,3:(dim(Xc)[2]-2)]
    if (is.matrix(m)){
        m <- m[,do.call("order",lapply(1:nrow(m),function(i) m[i,]))]
        m <- m[do.call("order",lapply(1:ncol(m),function(i) m[,i])),]
    }
    if (dim(Xc)[1]==1){
            Xc <- matrix(c(Xc[,c(1:2)],m,Xc[,c((dim(Xc)[2]-1):dim(Xc)[2])]),nrow=1)
    } else {
        Xc <- cbind(Xc[,c(1:2)],m,Xc[,c((dim(Xc)[2]-1):dim(Xc)[2])])
    }
    equal <- function(X,P){ all(X==P) }
    welke <- NULL
    if (is.matrix(m)){
            for (i in 1:a){ welke <- cbind(welke,apply(m,1,equal,i)) }
    }
    if (is.matrix(m)==FALSE){
            for (i in 1:a){ welke <- cbind(welke,all(m==i)) }
    }
    if (is.matrix(welke)){
            Pc.e <- sum(Xc[apply(welke,1,any),dim(Xc)[2]-1]^2) - sum(Xc[apply(welke,1,any), dim(Xc)[2]])
    }
    if (is.numeric(welke)){
            Pc.e <- sum(Xc[apply(welke,1,any),dim(Xc)[2]-1]^2) - sum(Xc[apply(welke,1,any), dim(Xc)[2]])
    }
    Pc.l <- 0
    bigger <- function(X,P){ all(X>P) }
    if (dim(Xc)[1]>1){
            U  <- apply(Xc[,3:(dim(Xc)[2]-2)],1,"bigger",apply(Xc[1:dim(Xc)[1],
                                                  3:(dim(Xc)[2]-2)],2,min))*c(1:dim(Xc)[1])
            U <- U[(U!=0)]
    } else {
            U <- all( Xc[,3:(dim(Xc)[2]-2)] > min(Xc[1:dim(Xc)[1],3:(dim(Xc)[2]-2)]))*c(1:dim(Xc)[1])
            U <- U[(U!=0)]
    }
    for (i in 1:(dim(Xc)[1]-1)){
            if (length(U)>1){
               Ui <- apply(Xc[U,3:(dim(Xc)[2]-2)],1,"bigger",Xc[i,3:(dim(Xc)[2]-2)])*U
                 Ui <- Ui[(Ui!=0)]
                 if (length(Ui) != 0){ Pc.l <- Pc.l + Xc[i,(dim(Xc)[2]-1)]*sum(Xc[Ui,(dim(Xc)[2]-1)]) }
               U  <- apply(Xc[U,3:(dim(Xc)[2]-2)],1,"bigger",apply(Xc[i:dim(Xc)[1],
                                                        3:(dim(Xc)[2]-2)],2,min))*U
            }
                if (length(U)==1){
               Ui <- bigger(Xc[U,3:(dim(Xc)[2]-2)],Xc[i,3:(dim(Xc)[2]-2)])*U
             Ui <- Ui[(Ui!=0)]
               if (length(Ui) != 0){ Pc.l <- Pc.l + Xc[i,(dim(Xc)[2]-1)]*sum(Xc[Ui,(dim(Xc)[2]-1)]) }
               U  <- bigger(Xc[U,3:(dim(Xc)[2]-2)],apply(Xc[i:dim(Xc)[1], 3:(dim(Xc)[2]-2)],2,min))*U
            }
            U <- U[(U!=0)]
            if (length(U) == 0){ break }
    }
    if (!strict){ Pc <- (2*Pc.l+Pc.e)/(sum(X[,w.col])^2 - sum(X[,w.col]^2)) }
    if (strict){ Pc <- (2*Pc.l)/(sum(X[,w.col])^2 - sum(X[,w.col]^2)) }
    return(Pc)
}

##############################################################################################
#
# Actual start of function WECCA.
#
##############################################################################################
```

```
        data <- as.matrix(data[,-1:-2])
        if (sim=="conc" && link=="total" && corr==FALSE && strict==FALSE){
                hier <- merge.conc.comlink.weight(data,a,corr=FALSE,strict=FALSE)
                dendro <- build.dendro(hier$merge,hier$height)
        }
        if (sim=="conc" && link=="total" && corr==FALSE && strict==TRUE){
                hier <- merge.conc.comlink.weight(data,a,corr=FALSE,strict=TRUE)
                dendro <- build.dendro(hier$merge,hier$height)
        }
        if (sim=="conc" && link=="total" && corr==TRUE && strict==FALSE){
                hier <- merge.conc.comlink.weight(data,a,corr=TRUE,strict=FALSE)
                dendro <- build.dendro(hier$merge,hier$height)
        }
        if (sim=="conc" && link=="total" && corr==TRUE && strict==TRUE){
                hier <- merge.conc.comlink.weight(data,a,corr=TRUE,strict=TRUE)
                dendro <- build.dendro(hier$merge,hier$height)
        }
        if (sim=="conc" && link=="complete" && corr==FALSE && strict==TRUE){
                dendro <- merge.conc.comtradlink.weight(data,a,corr=FALSE,strict=TRUE)
        }
        if (sim=="conc" && link=="complete" && corr==FALSE && strict==FALSE){
                dendro <- merge.conc.comtradlink.weight(data,a,corr=FALSE,strict=FALSE)
        }
        if (sim=="conc" && link=="complete" && corr==TRUE && strict==TRUE){
                dendro <- merge.conc.comtradlink.weight(data,a,corr=TRUE,strict=TRUE)
        }
        if (sim=="conc" && link=="complete" && corr==TRUE && strict==FALSE){
                dendro <- merge.conc.comtradlink.weight(data,a,corr=TRUE,strict=FALSE)
        }
        if (sim=="conc" && link=="average" && corr==FALSE && strict==FALSE){
                dendro <- merge.conc.avlink.weight(data,a,corr=FALSE,strict=FALSE)
        }
        if (sim=="conc" && link=="average" && corr==FALSE && strict==TRUE){
                dendro <- merge.conc.avlink.weight(data,a,corr=FALSE,strict=TRUE)
        }
        if (sim=="conc" && link=="average" && corr==TRUE && strict==FALSE){
                dendro <- merge.conc.avlink.weight(data,a,corr=TRUE,strict=FALSE)
        }
        if (sim=="conc" && link=="average" && corr==TRUE && strict==TRUE){
                dendro <- merge.conc.avlink.weight(data,a,corr=TRUE,strict=TRUE)
        }
        if (sim=="agree" && link=="total" && corr==FALSE){
                hier <- merge.agree.comlink.weight(data,a,corr=FALSE)
                dendro <- build.dendro(hier$merge,hier$height)
        }
        if (sim=="agree" && link=="total" && corr==TRUE){
                hier <- merge.agree.comlink.weight(data,a,corr=TRUE)
                dendro <- build.dendro(hier$merge,hier$height)
        }
        if (sim=="agree" && link=="complete" && corr==FALSE){
                dendro <- merge.agree.comtradlink.weight(data,a,corr=FALSE)
        }
        if (sim=="agree" && link=="complete" && corr==TRUE){
                dendro <- merge.agree.comtradlink.weight(data,a,corr=TRUE)
        }
        if (sim=="agree" && link=="average" && corr==FALSE){
                dendro <- merge.agree.avlink.weight(data,a,corr=FALSE)
        }
        if (sim=="agree" && link=="average" && corr==TRUE){
                dendro <- merge.agree.avlink.weight(data,a,corr=TRUE)
        }
        return(dendro)
}
```

A.3 Willenbrock Data Generation

```
##############################################################################################
gen.data.Willenbrock <- function(segmean,seglength,segclassify,n.sample,n.clone,noise,
                                 rL1=0,rL1p=0,
                                 rL2=0,rL2p=0,
                                 rG1=0,rG1p=0,
                                 rG2=0,rG2p=0,
                                 r01=0,r01p=0,
                                 r02=0,r02p=0,
                                 r03=0,r03p=0,
                                 r04=0,r04p=0,
                                 r05=0,r05p=0) {
  #if(length(segmean) > length(seglength)) {
  #  segmean <- sample(segmean,length(seglength))
  #}
  ## create empirical distribution of segment lengths for each classification
  seglength.gain <- seglength[segclassify == 1]
  seglength.nochange <- seglength[segclassify == 0]
  seglength.loss <- seglength[segclassify == -1]
  ## first identify which segmeans are gain, loss, no change (then sample from each with desired prob)
  segmean.gain <- segmean[segclassify==1]
  segmean.nochange <- segmean[segclassify==0]
  segmean.loss <- segmean[segclassify==-1]

  sim.classify <- array(NA,dim=c(n.clone,(n.sample+2)))
  sim.dat <- array(NA,dim=c(n.clone,(n.sample+2)))

  sim.dat[,1] <- sim.classify[,1] <- rep(1,n.clone)
  sim.dat[,2] <- sim.classify[,2] <- c(1:n.clone)


  for(i in 1:n.sample) {

    # create data sections separately, based on gain/loss propensity

    # r01
    if(length(r01) > 1) {
      sec.length.r01 <- 0
      sec.classify.r01 <- NA
      sec.res.r01 <- NA
      sec.index.r01 <- 1
      while(sec.length.r01 < length(r01)) {
        rGp <- rLp <- (1-r01p)/2
        #choose underlying classification for first segment in section of interest
        sec.classify.r01[sec.index.r01] <- sample(c(1,0,-1),1,replace=T,c(rGp,r01p,rLp))
        # choose segment length (dependent on classification)
        if(sec.classify.r01[sec.index.r01] == 1) {
          sec.res.r01[sec.index.r01] <- sample(seglength.gain,1)
        }
        if(sec.classify.r01[sec.index.r01] == 0) {
          sec.res.r01[sec.index.r01] <- sample(seglength.nochange,1)
        }
        if(sec.classify.r01[sec.index.r01] == -1) {
          sec.res.r01[sec.index.r01] <- sample(seglength.loss,1)
        }
        # ensure selected segment length is not longer than total section length
        if(sec.res.r01[sec.index.r01] >= length(r01)) sec.res.r01[sec.index.r01] <- length(r01)
        # ensure sum of segment lengths does not exceed total section length
        if((sec.res.r01[sec.index.r01] + sec.length.r01) >= length(r01))
          sec.res.r01[sec.index.r01] <- length(r01) - sec.length.r01
        sec.length.r01 <- sec.length.r01 + sec.res.r01[sec.index.r01]
        sec.index.r01 <- sec.index.r01 + 1
      }
    } else {
      sec.res.r01 <- 0
      sec.classify.r01 <- 0
```

```
}


# r02
if(length(r02) > 1) {
  sec.length.r02 <- 0
  sec.classify.r02 <- NA
  sec.res.r02 <- NA
  sec.index.r02 <- 1
  while(sec.length.r02 < length(r02)) {
    rGp <- rLp <- (1-r02p)/2
    #choose underlying classification for first segment in section of interest
    sec.classify.r02[sec.index.r02] <- sample(c(1,0,-1),1,replace=T,c(rGp,r02p,rLp))
    # choose segment length (dependent on classification)
    if(sec.classify.r02[sec.index.r02] == 1) {
      sec.res.r02[sec.index.r02] <- sample(seglength.gain,1)
    }
    if(sec.classify.r02[sec.index.r02] == 0) {
      sec.res.r02[sec.index.r02] <- sample(seglength.nochange,1)
    }
    if(sec.classify.r02[sec.index.r02] == -1) {
      sec.res.r02[sec.index.r02] <- sample(seglength.loss,1)
    }
    # ensure selected segment length is not longer than total section length
    if(sec.res.r02[sec.index.r02] >= length(r02)) sec.res.r02[sec.index.r02] <- length(r02)
    # ensure sum of segment lengths does not exceed total section length
    if((sec.res.r02[sec.index.r02] + sec.length.r02) >= length(r02))
      sec.res.r02[sec.index.r02] <- length(r02) - sec.length.r02
    sec.length.r02 <- sec.length.r02 + sec.res.r02[sec.index.r02]
    sec.index.r02 <- sec.index.r02 + 1
  }
} else {
  sec.res.r02 <- 0
  sec.classify.r02 <- 0
}

# r03
if(length(r03) > 1) {
  sec.length.r03 <- 0
  sec.classify.r03 <- NA
  sec.res.r03 <- NA
  sec.index.r03 <- 1
  while(sec.length.r03 < length(r03)) {
    rGp <- rLp <- (1-r03p)/2
    #choose underlying classification for first segment in section of interest
    sec.classify.r03[sec.index.r03] <- sample(c(1,0,-1),1,replace=T,c(rGp,r03p,rLp))
    # choose segment length (dependent on classification)
    if(sec.classify.r03[sec.index.r03] == 1) {
      sec.res.r03[sec.index.r03] <- sample(seglength.gain,1)
    }
    if(sec.classify.r03[sec.index.r03] == 0) {
      sec.res.r03[sec.index.r03] <- sample(seglength.nochange,1)
    }
    if(sec.classify.r03[sec.index.r03] == -1) {
      sec.res.r03[sec.index.r03] <- sample(seglength.loss,1)
    }
    # ensure selected segment length is not longer than total section length
    if(sec.res.r03[sec.index.r03] >= length(r03)) sec.res.r03[sec.index.r03] <- length(r03)
    # ensure sum of segment lengths does not exceed total section length
    if((sec.res.r03[sec.index.r03] + sec.length.r03) >= length(r03))
      sec.res.r03[sec.index.r03] <- length(r03) - sec.length.r03
    sec.length.r03 <- sec.length.r03 + sec.res.r03[sec.index.r03]
    sec.index.r03 <- sec.index.r03 + 1
  }
} else {
  sec.res.r03 <- 0
  sec.classify.r03 <- 0
}
```

```
# r04
if(length(r04) > 1) {
  sec.length.r04 <- 0
  sec.classify.r04 <- NA
  sec.res.r04 <- NA
  sec.index.r04 <- 1
  while(sec.length.r04 < length(r04)) {
    rGp <- rLp <- (1-r04p)/2
    #choose underlying classification for first segment in section of interest
    sec.classify.r04[sec.index.r04] <- sample(c(1,0,-1),1,replace=T,c(rGp,r04p,rLp))
    # choose segment length (dependent on classification)
    if(sec.classify.r04[sec.index.r04] == 1) {
      sec.res.r04[sec.index.r04] <- sample(seglength.gain,1)
    }
    if(sec.classify.r04[sec.index.r04] == 0) {
      sec.res.r04[sec.index.r04] <- sample(seglength.nochange,1)
    }
    if(sec.classify.r04[sec.index.r04] == -1) {
      sec.res.r04[sec.index.r04] <- sample(seglength.loss,1)
    }
    # ensure selected segment length is not longer than total section length
    if(sec.res.r04[sec.index.r04] >= length(r04)) sec.res.r04[sec.index.r04] <- length(r04)
    # ensure sum of segment lengths does not exceed total section length
    if((sec.res.r04[sec.index.r04] + sec.length.r04) >= length(r04))
      sec.res.r04[sec.index.r04] <- length(r04) - sec.length.r04
    sec.length.r04 <- sec.length.r04 + sec.res.r04[sec.index.r04]
    sec.index.r04 <- sec.index.r04 + 1
  }
} else {
  sec.res.r04 <- 0
  sec.classify.r04 <- 0
}

# r05
if(length(r05) > 1) {
  sec.length.r05 <- 0
  sec.classify.r05 <- NA
  sec.res.r05 <- NA
  sec.index.r05 <- 1
  while(sec.length.r05 < length(r05)) {
    rGp <- rLp <- (1-r05p)/2
    #choose underlying classification for first segment in section of interest
    sec.classify.r05[sec.index.r05] <- sample(c(1,0,-1),1,replace=T,c(rGp,r05p,rLp))
    # choose segment length (dependent on classification)
    if(sec.classify.r05[sec.index.r05] == 1) {
      sec.res.r05[sec.index.r05] <- sample(seglength.gain,1)
    }
    if(sec.classify.r05[sec.index.r05] == 0) {
      sec.res.r05[sec.index.r05] <- sample(seglength.nochange,1)
    }
    if(sec.classify.r05[sec.index.r05] == -1) {
      sec.res.r05[sec.index.r05] <- sample(seglength.loss,1)
    }
    # ensure selected segment length is not longer than total section length
    if(sec.res.r05[sec.index.r05] >= length(r05)) sec.res.r05[sec.index.r05] <- length(r05)
    # ensure sum of segment lengths does not exceed total section length
    if((sec.res.r05[sec.index.r05] + sec.length.r05) >= length(r05))
      sec.res.r05[sec.index.r05] <- length(r05) - sec.length.r05
    sec.length.r05 <- sec.length.r05 + sec.res.r05[sec.index.r05]
    sec.index.r05 <- sec.index.r05 + 1
  }
} else {
  sec.res.r05 <- 0
  sec.classify.r05 <- 0
}
```

```
# rL1
if(length(rL1) > 1) {
  sec.length.rL1 <- 0
  sec.classify.rL1 <- NA
  sec.res.rL1 <- NA
  sec.index.rL1 <- 1
  while(sec.length.rL1 < length(rL1)) {
    rGp <- (1-rL1p)*.10
    r0p <- (1-rL1p)*.90
    #choose underlying classification for first segment in section of interest
    sec.classify.rL1[sec.index.rL1] <- sample(c(1,0,-1),1,replace=T,c(rGp,r0p,rL1p))
    # choose segment length (dependent on classification)
    if(sec.classify.rL1[sec.index.rL1] == 1) {
      sec.res.rL1[sec.index.rL1] <- sample(seglength.gain,1)
    }
    if(sec.classify.rL1[sec.index.rL1] == 0) {
      sec.res.rL1[sec.index.rL1] <- sample(seglength.nochange,1)
    }
    if(sec.classify.rL1[sec.index.rL1] == -1) {
      sec.res.rL1[sec.index.rL1] <- sample(seglength.loss,1)
    }
    # ensure selected segment length is not longer than total section length
    if(sec.res.rL1[sec.index.rL1] >= length(rL1)) sec.res.rL1[sec.index.rL1] <- length(rL1)
    # ensure sum of segment lengths does not exceed total section length
    if((sec.res.rL1[sec.index.rL1] + sec.length.rL1) >= length(rL1))
      sec.res.rL1[sec.index.rL1] <- length(rL1) - sec.length.rL1
    sec.length.rL1 <- sec.length.rL1 + sec.res.rL1[sec.index.rL1]
    sec.index.rL1 <- sec.index.rL1 + 1
  }
} else {
  sec.res.rL1 <- 0
  sec.classify.rL1 <- 0
}

# rL2
if(length(rL2) > 1) {
  sec.length.rL2 <- 0
  sec.classify.rL2 <- NA
  sec.res.rL2 <- NA
  sec.index.rL2 <- 1
  while(sec.length.rL2 < length(rL2)) {
    rGp <- (1-rL2p)*.10
    r0p <- (1-rL2p)*.90
    #choose underlying classification for first segment in section of interest
    sec.classify.rL2[sec.index.rL2] <- sample(c(1,0,-1),1,replace=T,c(rGp,r0p,rL2p))
    # choose segment length (dependent on classification)
    if(sec.classify.rL2[sec.index.rL2] == 1) {
      sec.res.rL2[sec.index.rL2] <- sample(seglength.gain,1)
    }
    if(sec.classify.rL2[sec.index.rL2] == 0) {
      sec.res.rL2[sec.index.rL2] <- sample(seglength.nochange,1)
    }
    if(sec.classify.rL2[sec.index.rL2] == -1) {
      sec.res.rL2[sec.index.rL2] <- sample(seglength.loss,1)
    }
    # ensure selected segment length is not longer than total section length
    if(sec.res.rL2[sec.index.rL2] >= length(rL2)) sec.res.rL2[sec.index.rL2] <- length(rL2)
    # ensure sum of segment lengths does not exceed total section length
    if((sec.res.rL2[sec.index.rL2] + sec.length.rL2) >= length(rL2))
      sec.res.rL2[sec.index.rL2] <- length(rL2) - sec.length.rL2
    sec.length.rL2 <- sec.length.rL2 + sec.res.rL2[sec.index.rL2]
    sec.index.rL2 <- sec.index.rL2 + 1
  }
} else {
  sec.res.rL2 <- 0
  sec.classify.rL2 <- 0
}
```

```
# rG1
if(length(rG1) > 1) {
  sec.length.rG1 <- 0
  sec.classify.rG1 <- NA
  sec.res.rG1 <- NA
  sec.index.rG1 <- 1
  while(sec.length.rG1 < length(rG1)) {
    rLp <- (1-rG1p)*.10
    r0p <- (1-rG1p)*.90
    #choose underlying classification for first segment in section of interest
    sec.classify.rG1[sec.index.rG1] <- sample(c(1,0,-1),1,replace=T,c(rG1p,r0p,rLp))
    # choose segment length (dependent on classification)
    if(sec.classify.rG1[sec.index.rG1] == 1) {
      sec.res.rG1[sec.index.rG1] <- sample(seglength.gain,1)
    }
    if(sec.classify.rG1[sec.index.rG1] == 0) {
      sec.res.rG1[sec.index.rG1] <- sample(seglength.nochange,1)
    }
    if(sec.classify.rG1[sec.index.rG1] == -1) {
      sec.res.rG1[sec.index.rG1] <- sample(seglength.loss,1)
    }
    # ensure selected segment length is not longer than total section length
    if(sec.res.rG1[sec.index.rG1] >= length(rG1)) sec.res.rG1[sec.index.rG1] <- length(rG1)
    # ensure sum of segment lengths does not exceed total section length
    if((sec.res.rG1[sec.index.rG1] + sec.length.rG1) >= length(rG1))
      sec.res.rG1[sec.index.rG1] <- length(rG1) - sec.length.rG1
    sec.length.rG1 <- sec.length.rG1 + sec.res.rG1[sec.index.rG1]
    sec.index.rG1 <- sec.index.rG1 + 1
  }
} else {
  sec.res.rG1 <- 0
  sec.classify.rG1 <- 0
}

# rG2
if(length(rG2) > 1) {
  sec.length.rG2 <- 0
  sec.classify.rG2 <- NA
  sec.res.rG2 <- NA
  sec.index.rG2 <- 1
  while(sec.length.rG2 < length(rG2)) {
    rLp <- (1-rG2p)*.10
    r0p <- (1-rG2p)*.90
    #choose underlying classification for first segment in section of interest
    sec.classify.rG2[sec.index.rG2] <- sample(c(1,0,-1),1,replace=T,c(rG2p,r0p,rLp))
    # choose segment length (dependent on classification)
    if(sec.classify.rG2[sec.index.rG2] == 1) {
      sec.res.rG2[sec.index.rG2] <- sample(seglength.gain,1)
    }
    if(sec.classify.rG2[sec.index.rG2] == 0) {
      sec.res.rG2[sec.index.rG2] <- sample(seglength.nochange,1)
    }
    if(sec.classify.rG2[sec.index.rG2] == -1) {
      sec.res.rG2[sec.index.rG2] <- sample(seglength.loss,1)
    }
    # ensure selected segment length is not longer than total section length
    if(sec.res.rG2[sec.index.rG2] >= length(rG2)) sec.res.rG2[sec.index.rG2] <- length(rG2)
    # ensure sum of segment lengths does not exceed total section length
    if((sec.res.rG2[sec.index.rG2] + sec.length.rG2) >= length(rG2))
      sec.res.rG2[sec.index.rG2] <- length(rG2) - sec.length.rG2
    sec.length.rG2 <- sec.length.rG2 + sec.res.rG2[sec.index.rG2]
    sec.index.rG2 <- sec.index.rG2 + 1
  }
} else {
  sec.res.rG2 <- 0
  sec.classify.rG2 <- 0
}
```

```
    # determine order of sections
    section.list <- list(rL1,rL2,rG1,rG2,r01,r02,r03,r04,r05)
    sec.res.list <- list(sec.res.rL1,sec.res.rL2,sec.res.rG1,sec.res.rG2,
                         sec.res.r01,sec.res.r02,sec.res.r03,sec.res.r04,sec.res.r05)
    sec.classify.list <- list(sec.classify.rL1,sec.classify.rL2,sec.classify.rG1,sec.classify.rG2,
                              sec.classify.r01,sec.classify.r02,sec.classify.r03,
                              sec.classify.r04,sec.classify.r05)
    min.list <- c(min(rL1),min(rL2),min(rG1),min(rG2),
                  min(r01),min(r02),min(r03),min(r04),min(r05))
    id <- sort.list(min.list)
    section.list.order <- section.list[id]
    sec.res.order <- sec.res.list[id]
    sec.res.order.unlist <- unlist(sec.res.order)
    sec.res.tot <- sec.res.order.unlist[sec.res.order.unlist > 0]
    sec.classify.tot <- unlist(sec.classify.list[id])[sec.res.order.unlist > 0]

    ## expand sec.classify.tot
    sec.classify.expand <- rep(sec.classify.tot,sec.res.tot)

    # generate log2 ratios based on classification information
    sim.classify.new <- NA
    for(jj in 1:length(sec.classify.expand)) {
      if(sec.classify.expand[jj]==-1) sim.classify.new[jj] <- 0
      if(sec.classify.expand[jj]==0) sim.classify.new[jj] <- 2
      if(sec.classify.expand[jj]==1) sim.classify.new[jj] <- 6
    }
    ## select proportion of tumor cells0
    p.ind <- runif(1,.3,.7)
    ## find expected log2 ratio for each clone
    log2.expected <- log2((sim.classify.new*p.ind + 2*(1-p.ind))/2)
    ## add noise to data
    if(length(noise) > 1) {
      noise.sample <- sample(noise,1)
      mean.ind.tot.fin <- rnorm(n.clone,log2.expected,noise.sample)
    } else {
      mean.ind.tot.fin <- rnorm(n.clone,log2.expected,noise)
    }

    # store classification, log2 results
    sim.classify[,(i+2)] <- sec.classify.expand
    sim.dat[,(i+2)] <- mean.ind.tot.fin

  } # end subject loop
  sim.res <- list(sim.dat,sim.classify)
  return(sim.res)
}
##############################################################################################
```

APPENDIX B

SCRIPT CODE

Included below are the R scripts that were used to run the WECCA method in both the data analysis and simulations. CGHregions finds genomic regions of copy number alteration. With the appropriate user input, the WECCA script produces cluster assignments for the region data.

B.1 CGH Regions Script

```
            ############## HOW TO RUN THIS SCRIPT  #############
#This script generates CGH region data from called array CGH data. It corresponds to the
#(submitted) paper: "CGHregions: dimension reduction for array CGH data with minimal information loss" (2006)
#by Mark A. van de Wiel and Wessel N. van Wieringen.

#Running this script:
# 1. Save the scripts 'ForCGHRegionsUsers.R' and 'CGHRegions.R' in the same folder
# 2. Check the input file format (see below and example file)
# 3. Set the input parameters in the USER INPUT section below.
# 4. Select ALL text in this script (CTRL + A), and copy (CTRL + C) it into the R-console.
# 5. Wait. You can observe progression by switching off 'Buffered output' in the RGui 'Misc' submenu.



#Amsterdam, December 20 2006, Mark A. van de Wiel, mark.vdwiel@vumc.nl



             ############## USER INPUT ###############

#SET YOUR WORKING DIRECTORY: FOLDER THAT CONTAINS THE INPUT DATA
#setwd("C:\\Synchr\\Microarrays\\CGHRegions\\Data\\")
#setwd("C:\\VUData\\Boudewijn")
#setwd("C:\\VUData\\Tineke\\TinekeCGH\\")
#setwd("C:\\MyCGHData\\")
#setwd("C:\\VUData\\lymphoma")

#USE imputemiss <- "no" ONLY WHEN CERTAIN THAT DATA DOES NOT CONTAIN MISSINGS.
imputemiss <- "no"
#imputemiss <- "yes"

#LEVELS WHICH DISTINGUISH THE LOSS, NORMAL, GAIN (AND AMPLIFICATION) STATES IN THE INPUT FILE
#levels <- c(-1,0,1)  #-1 = loss, 0 = normal, 1 = gain
levels <- c(-1,0,1,2) #-1 = loss, 0 = normal, 1 = gain, 2 = amp
#levels <- c(1,2,3) #1 = loss, 2 = normal, 3 = gain

#LEVEL CORRESPONDDING TO NORMAL STATE
normstate <- 0
#normstate <- 2

#AVERAGE ERROR RATE (T) IN 25% MOST ACTIVE REGIONS
#thresh <- 0.025
thresh <- 0.01 #Recommended for unsupervised analysis (clustering)

#NAME OF INPUTFILE, WHICH SHOULD BE TAB-SEPARATED .txt
#inputfile <- "CALLS_Imp_stanford_tineke_ruw_49 cases.txt"
#inputfile <- "Test_Called.txt"

#FOLDER IN WHICH THE CGHRegions.R SCRIPT IS LOCATED. ALWAYS END WITH "\\".
```

```
#Rfolder <- "C:\\Synchr\\Rscripts\\CGH\\CGHRegions\\"
#Rfolder <- "C:\\Synchr\\Microarrays\\CGHRegions\\Data\\"

#PREFIX TO PUT IN FRONT OF INPUTFILE TO CREATE NAME OUTPUTFILE
#prefix <- "regions_"

            ################ END USER INPUT ###########



            ################ READ DATA: NOTE ABOUT DATA FORMAT  ###########

#DATA SHOULD CONTAIN A HEADER, IF NOT USE 'Header = False';
#DATA FORMAT: 1ST COLUMN NAME, 2ND CHROMOSOME, 3RD BP POSITION (OR SOME ROW INDEX), REST: PROFILES (CALLS)
#A SIMPLE IMPUTATION SCHEME IS USED FOR MISSING CALLS: SUBSTITUTION BY CALL FOR PREVIOUS CLONE.
#WE RECOMMEND TO DELETE CLONES WITH TOO MANY MISSINGS FIRST
#MISSING CHROMOSOME OR BP POSITION IS ALLOWED; THESE CLONES WILL BE SKIPPED

#outputfile <- paste(prefix,inputfile,sep="")
#CGHdata <- read.table(inputfile,header=TRUE,sep="\t",fill=TRUE) #SEPARATOR = TAB
#CGHdata <- read.table(inputfile,header=TRUE,sep=",",fill=TRUE) #SEPARATOR = ,

            ############## END READ DATA  ######################



            #############  CGHRegion  ################

#srcCGHReg <- paste(Rfolder,"CGHRegions.R",sep="")
#source(srcCGHReg)

kolnam <- colnames(CGHdata)
kolnam <- kolnam[-(1:3)]
#column with names is not used
CGHdata <- CGHdata[,-1]

#delete clones with missing chromosome or base pair information
CGHdata <- CGHdata[!is.na(CGHdata[,1]) & !is.na(CGHdata[,2]) & (CGHdata[,1] > 0) & (CGHdata[,2]>0),]

#CGHdata <- load(info.CGH)
ncolm <- ncol(CGHdata)-2
critst <- max(1,floor(ncolm/10))
stepsize <- max(1,round(ncolm/20))

if(imputemiss=="yes")
{
twocols <- CGHdata[,1:2]
notwocols <- CGHdata[,-(1:2)]
CGHdataimp <- imp_missing(notwocols)
CGHdata <- cbind(twocols,CGHdataimp)
print("End imputation...")
}

#SET STARTING VALUE MANUALLY
#critst <- 7
#bppos and chromo
chromo <- CGHdata[,1]
bppos <- CGHdata[,2]

srcCGHReg <- paste(Rfolder,"CGHRegions.R",sep="")
source(srcCGHReg)

numbr <- nrow(CGHdata)
if(numbr<=10000)
{
startnew <- floor(numbr/3)
minim <- min(400,startnew)
CGHdataTry <- rbind(CGHdata[1:minim,],CGHdata[startnew:(startnew+minim),],
```

```
    CGHdata[(2*startnew):(2*startnew+minim),])
} else #use somewhat more (but smaller) tuning regions for oligo data
{
startnew <- floor(numbr/6)
minim <- min(400,startnew)
CGHdataTry <- rbind(CGHdata[1:minim,],CGHdata[startnew:(startnew+minim),],
    CGHdata[(2*startnew):(2*startnew+minim),],CGHdata[(3*startnew):(3*startnew+minim),],
    CGHdata[(4*startnew):(4*startnew+minim),],CGHdata[(5*startnew):(5*startnew+minim),])
}
#cspr <- deterreg(CGHdata=CGHdata,5,ncolm,normstate,levels)

#First: find starting value using small data set;
#then large data set.
pmt<- proc.time()
stoploop <- 0
cspr <- deterreg(CGHdata=CGHdataTry,critst,ncolm,normstate,levels)
critsatpr <- cspr[[1]]
print(c(critst,cspr[[1]],nrow(cspr[[2]])))
ifelse(critsatpr<=thresh,cr <- critst+stepsize,cr <- max(0,critst-stepsize))
while (stoploop==0)
{
cs <- deterreg(CGHdata=CGHdataTry,cr,ncolm,normstate,levels)
critsat <- cs[[1]]
print(c(cr,cs[[1]],nrow(cs[[2]])))
if (critsat>= thresh)
{
   if (critsatpr<=thresh)
   {
   stoploop<-1
   critfound <- cr-stepsize + floor(stepsize*(thresh-critsatpr)/(critsat-critsatpr))
   }
   else {cr <- max(0,cr-stepsize)}
}
else
{
   if (critsatpr>=thresh)
   {
   stoploop<-1
   critfound <- cr + floor(stepsize*(thresh-critsat)/(critsatpr-critsat))
   }
   else
   {
   if (critsat==critsatpr)
   {
   stoploop<-1
   critfound <- cr
   } else {cr <- cr+stepsize}}
}
critsatpr <- critsat
}
critfound
proc.time()-pmt
print("Tuning on small data set finished...started with entire data set")

pmt <- proc.time()
#now the LARGE data set

#critfound <- 4
#CGHdataHalf <- CGHdata[,1:(floor((ncolm+2)/2))]
#ncolm <- ncol(CGHdataHalf)-2

stoploop <- 0
cspr <- deterreg(CGHdata=CGHdata,critfound,ncolm,normstate,levels)
critsatpr <- cspr[[1]]
print(c(critfound,cspr[[1]],nrow(cspr[[2]]),nrow(CGHdata)-nrow(cspr[[3]])))
ifelse(critsatpr<=thresh,cr <- critfound+1,cr <- critfound-1)
while (stoploop==0)
{
```

```
cs <- deterreg(CGHdata=CGHdata,cr,ncolm,normstate,levels)
critsat <- cs[[1]]
print(c(cr,cs[[1]],nrow(cs[[2]])))
if (critsat>= thresh)
{
   if (critsatpr<=thresh) #stop with loop
   {
   stoploop<-1
   critfound <- cr-1
   regionsfound <- cspr[[2]]
   countnomono <- cspr[[3]]
   }
   else  #continue with loop
   {
   cr <- cr-1
   critsatpr <- critsat
   cspr <- cs
   }
}
else
{
   if (critsatpr>=thresh) #stop with loop
   {
   stoploop<-1
   critfound <- cr
   regionsfound <- cs[[2]]
   countnomono <- cs[[3]]
   }
   else #continue with loop
   {
   cr <- cr+1
   critsatpr <- critsat
   cspr <- cs
   }
}
}
critfound
proc.time()-pmt
print(paste("c = ",critfound,", nr of regions: ", nrow(regionsfound), sep=""))
print("Finished with entire data set...writing output file")
res <- apply(regionsfound,1,whichsign2,ctdat=countnomono,levels=levels)
prof <- t(sapply(res,function(x) {as.vector(x[[1]],mode="numeric")}))
ntd <- function(reg,ct) {
datareg <- td(reg,ct)
return(ifelse(!is.null(dim(datareg)),nrow(datareg),1))
}
nclone <- apply(regionsfound,1,ntd,ct=countnomono)
aved <- signif(as.vector(lapply(res,function(x) {x[[2]]}),mode="numeric")/nclone,digits=3)
bp <- t(apply(regionsfound,1,findbp,bppos = bppos))
chrreg <- apply(regionsfound,1,findchr,chr = chromo)
towrite <- cbind(regionsfound,bp,chrreg,nclone,aved,prof)
od <- order(towrite[,1])
towrite <- towrite[od,-(1:2)]
kolnamnew <- c("bp start", "bp end","chromosome","nclone","Ave Dist",kolnam)
colnames(towrite) <- kolnamnew
write.table(towrite, file=outputfile,row.names=FALSE, sep = "\t")
#write.table(towrite, file=outputfile,row.names=FALSE, sep = ",")

         #############  END CGHRegion  ################
```

B.2 WECCA Script

```
#########################################################################################################################
#
# HOW TO RUN THIS SCRIPT
#
```

```
# This script clusters called array CGH data. It corresponds to the paper
# "Weighted clustering of called aCGH data" (2007)
# by Wessel N. van Wieringen, Mark A. van de Wiel, Bauke Ylstra
#
# Running this script:
# 1. Save the scripts 'ForWECCAUsers.R' and 'WECCA.R' in the same folder
# 2. Check the input file format (see below and example file)
# 3. Set the input parameters in the USER INPUT section below.
# 4. Select ALL text in this script (CTRL + A), and copy (CTRL + C) it into the R-console.
# 5. Wait. You can observe progression by switching off 'Buffered output' in the RGui 'Misc' submenu.
#
#
# Amsterdam, August 25, 2007, Wessel N. van Wieringen, wvanwie@few.vu.nl
#
#########################################################################################################################



#########################################################################################################################
#
# BEGIN OF USER INPUT
#
#########################################################################################################################

# SET YOUR WORKING DIRECTORY: FOLDER THAT CONTAINS THE INPUT DATA
# setwd("C:\\MyCGHData\\")

# SPECIFY THE NUMBER OF LEVELS IN THE INPUT FILE
number.of.levels <- 3   # 3 with only loss, normal and gain states, 4 when amplifications are present also

# SPECIFY TYPE OF LINKAGE TO BE USED ("average", "complete", or "total")
# NOTE: TOTAL LINKAGE TAKES LONG. FOR EXAMPLE: APPROXIMATE TEN MINUTES FOR 50 SAMPLES AND 250 REGIONS
linkage <- "average"

# SPECIFY TYPE OF SIMILARITY TO BE USED ("agree", or "conc")
similarity <- "conc"

# SPECIFY TYPE OF WEIGHTS TO BE USED ("as.is", "all.equal" or "heterogeneity")
weight.type <- "all.equal"

# NAME OF INPUTFILE, WHICH SHOULD BE TAB-SEPARATED .txt
#inputfile <- "regions_called data arrayCGH.txt"

# FOLDER IN WHICH THE WECCA.functions.R SCRIPT IS LOCATED. ALWAYS END WITH "\\".
# Rfolder <- "C:\\Rscripts\\CGH\\"


#########################################################################################################################
#
# END OF USER INPUT
#
# NOTE: ADDITIONAL USER INPUT IS REQUIRED AT THE END OF THE SCRIPT.
#
#########################################################################################################################



#########################################################################################################################
#
# SPECIAL FEATURES
#
# PLEASE, ONLY USE THESE FEATURES WHEN YOU KNOW WHAT YOU ARE DOING!!!!!!!!!!
# IF NOT, ASK WESSEL OR MARK.
#
# WECCA offers some options not described in the paper:
# 1) The original definition of concordance is used when putting the variable 'strict' to TRUE.
# 2) The similarities are normalized when putting the variable 'corr' to TRUE.
#
```

```
#########################################################################################################################

# WHEN USING THE CONCORDANCE SIMILARITY, SPECIFY WHETHER THE STRICT OR NON-STRICT CONCORDANCE SHOULD BE USED
strict <- FALSE

# WHEN USING TOTAL LINKAGE, SPECIFY WHETHER THE SIMILARITY SHOULD BE CORRECTED FOR
# THE NUMBER OF SAMPLES IN THE CLUSTER
normalization <- FALSE

#########################################################################################################################
#
# END OF SPECIAL FEATURES
#
#########################################################################################################################



#########################################################################################################################
#
# READ DATA: NOTE ABOUT DATA FORMAT
#
# DATA SHOULD CONTAIN A HEADER, IF NOT USE 'header = FALSE';
# DATA FORMAT: 1ST COLUMN REGION NAME, 2ND CHROMOSOME, 3RD WEIGHT, REST: PROFILES (CALLS)
#
#########################################################################################################################

CGHdata <- read.table(inputfile, header=TRUE, sep="\t")      # SEPARATOR = TAB


#########################################################################################################################
#
# END OF READING DATA
#
#########################################################################################################################


#########################################################################################################################
#
# BEGIN OF WECCA
#
#########################################################################################################################

# Read in the necessary functions
srcWECCA <- paste(Rfolder,"WECCA.functions.R",sep="")
source(srcWECCA)

# WECCA reguires the following labelling 1 = loss, 2 = normal, 3 = gain, 4 = amplification
# If one has provided the following labelling -1 = loss, 0 = normal, 1 = gain, 2 = amplification,
# this is transformed to the labelling needed.
if (min(CGHdata[,4:dim(CGHdata)[2]]) == -1){
        CGHdata[,4:dim(CGHdata)[2]] <- CGHdata[,4:dim(CGHdata)[2]] + 2
}

# Check whether indeed number.of.levels category calls is present in the data.
call.table <- as.numeric(table(factor(as.numeric(as.matrix(CGHdata[,-c(1:3)])),levels=c(1:4))))
effective.levels <- c(1:4)[call.table!=0]
if (length(effective.levels) != number.of.levels){
    print("The number of call levels specified does not match the number observed in the data.")
    print(paste("It appears there are no",c("losses","normals","gains","amplifications")[call.table==0],
            "in the data present."))
    print("Please check the data.")
} else {
    if (any(effective.levels != 1:number.of.levels)){
            print("The number of call levels specified does not match the number observed in the data.")
            print(paste("It appears there are no",c("losses","normals","gains","amplifications")[call.table==0],
                    "in the data present."))
            print("Please check the data.")
    }
```

```
}

# Specify the weights automatically.
if (weight.type == "all.equal"){
    CGHdata[,3] <- rep(1,dim(CGHdata)[1])
}
if (weight.type == "heterogeneity"){
   library(pgirmess)
   W <- apply(CGHdata[,-c(1:3)],1,function(x, number.of.levels){ shannon(as.numeric(table(
     factor(x,levels=c(1:number.of.levels))))/length(x))[1] }, number.of.levels)
   CGHdata[,3] <- W
}

# Hierarchical clustering is done now.
dendrogram <- WECCA(CGHdata,number.of.levels,similarity,linkage,strict,normalization)

## The dendrogram is plotted, using the sample colnames as labels.
#plot(dendrogram, labels=colnames(CGHdata)[c(-1,-2,-3)])
#savePlot(file=paste("Dendrogram_of_",inputfile,sep=""),type="pdf")


## Preparation of the plotting of the heatmap
## Generate alternating colors for chromosomes.
#library(gtools)
#chr.color <- rep("blue",dim(CGHdata)[1])
#chr.color[even(CGHdata[,2])] <- c("yellow")
#
## Generate labels for begin points of chromosomes.
#Y <- rep(FALSE,dim(CGHdata)[1])
#for (i in 2:(dim(CGHdata)[1])){
#        if ((CGHdata[i-1,2]!=CGHdata[i,2])){ Y[i] <- TRUE }
#}
#begin.chr <- rep("",dim(CGHdata)[1])
#begin.chr[Y] <- CGHdata[Y,2]
#begin.chr[1] <- "1"

## The heatmap is plotted.
#color.coding <- c("red", "black", "green", "white")[effective.levels]
#heatmap(as.matrix(CGHdata[,4:dim(CGHdata)[2]]),
# Colv=as.dendrogram(dendrogram),Rowv=NA,col=color.coding,
            labRow=begin.chr,RowSideColors=chr.color,scale="none")
## savePlot(file=paste("Heatmap_of_",inputfile,sep=""),type="pdf")
#savePlot(file=paste("Heatmap_of_",(gsub(".txt","",inputfile),".pdf",sep=""),type="pdf")
#
## The dendrogram is saved in an .Rdata object for later use.
#save(dendrogram,file="dendrogram.Rdata")

#########################################################################################################################
#
# END OF WECCA
#
#########################################################################################################################


#########################################################################################################################
#
# BEGIN POST-PROCESSING: EXTRACT CLUSTERS FROM DENDROGRAM
#
#########################################################################################################################

## SPECIFY THE NUMBER OF CLUSTERS TO BE EXTRACTED FROM THE DENDROGRAM
#number.of.clusters <- 2

## A .txt-file specifying which samples belong the same cluster will now be generated.
#samples.in.clusters <- cutree(dendrogram,k=number.of.clusters)
#cluster.results <- cbind(1:length(samples.in.clusters),colnames(CGHdata[,4:dim(CGHdata)[2]]),
                    samples.in.clusters)
#colnames(cluster.results) <- c("sample.number","sample.label","cluster")
```

```
#write.table(cluster.results,file="cluster.results.txt",quote=FALSE,row.names=FALSE,sep="\t")

#########################################################################################################################
#
# END OF POST_PROCESSING: EXTRACT CLUSTERS FROM DENDROGRAM
#
#########################################################################################################################
```

APPENDIX C

DATA ANALYSIS CODE

The R code for the data analysis is divided into four sections. Listed first is the code that was used to read in and subset the aCGH data from ovarian cancer patients; this should be run prior to running the code for HC and WECCA. The next code sections for HC and WECCA run the respective methods and analyze how well the cluster assignments correlate with clinical information. HC and WECCA analyses are compared in the final section of code.

C.1 Data

```
library(survival)
ovca.log2 <- read.csv("OVCA_log2Ratios.csv", header=T) # 105 samples, 99264 markers
id <- read.csv("OVCA_SampleList_WithChipTag.csv", header=T) # 78 patients with clinical info
info <- read.csv("Ovarian database Combined WBG edit 7.3.09l.csv", skip=1, header=T)
info <- info[-(79:86),] # 78 x 54, clinical info, excludes empty rows

#####  KEEP SAMPLES WITH CLINICAL INFO (78)  #####

orderchip <- order(info$Chip.)
orderorig <- order(order(id$Chip.ID))
newinfo <- info[orderchip,]
clinical.dat <- ordinfo <- newinfo[orderorig,]
clinical.dat$Sample <- ordinfo$Sample <- id$Sample.ID

split <- strsplit(names(ovca.log2)[-(1:5)],"_")
split2 <- unlist(split) # alternating sample and chip IDs (length=105)
sq <- seq(1,length(split2))
ovcasamp <- split2[sq%%2 == 1]
clinical.samp <- which(ovcasamp %in% clinical.dat$Sample)

OVCA.105.log2 <- ovca.log2[,-(1:5)]

######  CHECK FOR MISSING DATA  ######

sum(is.na(OVCA.105.log2)) # 1
c <- which(apply(is.na(OVCA.105.log2),2,sum)==1) # CS1070_4447a
r <- which(apply(is.na(OVCA.105.log2),1,sum)==1)

# Impute missing value with the mean of adjacent observations
OVCA.105.log2[r,c] <- (OVCA.105.log2[r-1,c] + OVCA.105.log2[r+1,c])/2

######  RESTRICT ANALYSES  ######

# To samples with clinical information
OVCA.78.log2 <- ovca.log2[,-(1:5)][,clinical.samp]

# To samples with "serous" histology
OVCA.74.log2 <- OVCA.78.log2[,clinical.dat$Histology=="Serous"]
clinical.74 <- clinical.dat[clinical.dat$Histology=="Serous",]
```

C.2 Hierarchical Clustering

```
dist.78 <- dist(t(OVCA.78.log2))
hc.78 <- hclust(dist.78, method="ward")
plot(hc.78, xlab="", sub="", labels=F, main="")

dist.74 <- dist(t(OVCA.74.log2))#)
hc.74 <- hclust(dist.74, method="ward")

postscript("Hdend78.ps", width=40, height=23)
par(cex=4)
plot(hc.78, xlab="", sub="", main="", labels=id$Sample.ID, cex=0.6)
dev.off()

postscript("Hdend74.ps", width=40, height=23)
par(cex=4)
plot(hc.74, xlab="", sub="", main="", labels=clinical.74$Sample, cex=0.6)
mtext("Cluster 1", side=1, line=-.8, cex=4, at=23)
mtext("Cluster 2", side=1, line=-.8, cex=4, at=60)
dev.off()

# Compare cluster assignments for different size samples
# Are the common 74 samples clustered in the same groups?
clust78 <- cutree(hc.78, k=3)
clust74 <- cutree(hc.74, k=2)
subset78 <- which(names(clust78) %in% names(clust74))
table(clust74,clust78[subset78])


##########################################################
#####   RELATION OF CLUSTERS TO CLINICAL VARIABLES   #####
##########################################################

Hclust <- cutree(hc.74,k=2)
Hclust <- Hclust - 1
clin.dat <- clinical.74

# Continuous variables
summary(glm(Hclust ~ Age.at.dx, family=binomial, data=clin.dat))$coef[2,4]
summary(glm(Hclust ~ stage.number, family=binomial, data=clin.dat))$coef[2,4]

table(clin.dat$stage.number,Hclust)

# Categorical variables
summary(clin.dat)
covf.i <- c(8,10:12,16,20,25,28,30:38,43:48,52:53) # should be factor variables
covf <- names(clin.dat)[covf.i]
summary(clin.dat[covf.i])

# Change n/a and missing values to NA - then R will eliminate these from analysis
clin.dat$Grade[clin.dat$Grade %in% c("","n/a")] <- NA
clin.dat$Outcome[clin.dat$Outcome %in% c("","n/a")] <- NA
clin.dat$Recur[clin.dat$Recur %in% c("","n/a")] <- NA
summary(clin.dat[covf.i])

# Logistic regression
clust.covf.pval <- NULL
for (i in 1:length(covf)){
        clust.covf.pval[i] <- summary(glm(Hclust ~ as.factor(get(covf[i])),
                                        family=binomial,  data=clin.dat))$coef[2,4] }

cbind(data.frame(covf[which(clust.covf.pval < 0.05)]), clust.covf.pval[clust.covf.pval < 0.05])
allpval <- cbind(data.frame(covf), clust.covf.pval)
allpval[order(allpval[,2]),] # all p-values whose variables were treated as categorical

summary(glm(Hclust ~ stage.number, family=binomial, data=clin.dat))$coef[2,4]
```

```
##########################################
#####  KAPLAN-MEIER SURVIVAL CURVES  #####
##########################################

#####  OVERALL SURVIVAL  #####

time <- 12*clin.dat$time..years. # years to months
cens <- clin.dat$Censor

# Split data into groups according to clustering
grp1 <- clin.dat[Hclust==1,]
grp0 <- clin.dat[Hclust==0,]

# Create complete survival information for each group
time1 <- 12*grp1$time..years.
cens1 <- grp1$Censor
time0 <- 12*grp0$time..years.
cens0 <- grp0$Censor

# Get K-M estimates
grp1.km <- summary(survfit(Surv(time1,cens1)~1, type="kaplan-meier"))
grp0.km <- summary(survfit(Surv(time0,cens0)~1, type="kaplan-meier"))

# Add first time point
grp1.time <- c(0, grp1.km$time)
grp1.surv <- c(1, grp1.km$surv)
grp0.time <- c(0, grp0.km$time)
grp0.surv <- c(1, grp0.km$surv)

# Confidence intervals
g1.upper <- c(1,grp1.km$upper)
g1.lower <- c(1,grp1.km$lower)
g0.upper <- c(1,grp0.km$upper)
g0.lower <- c(1,grp0.km$lower)

maxtime <- max(c(grp1.time,grp0.time))

postscript("Surv-Hclust.ps", height=10, width=10)
plot(grp0.time, grp0.surv, ylim=c(0,1), xlim=c(0,maxtime), type="s", lty=1, col="red",
       xlab="Months", ylab="Overall Survival", main="",
       cex.lab=1.4, cex.main=1.4, cex.axis=1.4, lwd=2)
lines(grp1.time, grp1.surv, ylim=c(0,1), type="s", lty=1, col="blue", lwd=2)
lines(grp0.time, g0.lower, lty=3, col="red", lwd=3)
lines(grp0.time, g0.upper, lty=3, col="red", lwd=3)
lines(grp1.time, g1.lower, lty=3, col="blue", lwd=3)
lines(grp1.time, g1.upper, lty=3, col="blue", lwd=3)
legend(-0.05, 0.18, c("Cluster 1","Cluster 2","95% Conf. Limits"),
           lty=c(1,1,3), col=c("red","blue","black"), bty="n", cex=1.3, lwd=c(2,2,3))
dev.off()

#####  PROGRESSION-FREE SURVIVAL  #####

time <- clin.dat$PFS..MONTHS.
cens <- clin.dat$PFScens

grp1 <- clin.dat[Hclust==1,]
grp0 <- clin.dat[Hclust==0,]

time1 <- grp1$PFS..MONTHS.
cens1 <- grp1$PFScens
time0 <- grp0$PFS..MONTHS.
cens0 <- grp0$PFScens

grp1.km <- summary(survfit(Surv(time1,cens1)~1, type="kaplan-meier"))
grp0.km <- summary(survfit(Surv(time0,cens0)~1, type="kaplan-meier"))

grp1.time <- c(0, grp1.km$time)
grp1.surv <- c(1, grp1.km$surv)
```

```
grp0.time <- c(0, grp0.km$time)
grp0.surv <- c(1, grp0.km$surv)

g1.upper <- c(1,grp1.km$upper)
g1.lower <- c(1,grp1.km$lower)
g0.upper <- c(1,grp0.km$upper)
g0.lower <- c(1,grp0.km$lower)

maxtime <- max(c(grp1.time,grp0.time))

postscript("PFS-Hclust.ps", height=10, width=10)
plot(grp0.time, grp0.surv, ylim=c(0,1), xlim=c(0,maxtime), type="s", lty=1, col="red",
        xlab="Months", ylab="Progression-Free Survival", main="",
        cex.lab=1.4, cex.main=1.4, cex.axis=1.4, lwd=2)
lines(grp1.time, grp1.surv, ylim=c(0,1), type="s", lty=1, col="blue", lwd=2)
lines(grp0.time, g0.lower, lty=3, col="red", lwd=3)
lines(grp0.time, g0.upper, lty=3, col="red", lwd=3)
lines(grp1.time, g1.lower, lty=3, col="blue", lwd=3)
lines(grp1.time, g1.upper, lty=3, col="blue", lwd=3)
legend(0.7*maxtime, 1.04, c("Cluster 1","Cluster 2","95% Conf. Limits"),
        lty=c(1,1,3), col=c("red","blue","black"), bty="n", cex=1.3, lwd=c(2,2,3))
dev.off()

# Additional check of proportional hazards assumption - Schoenfeld residuals
# OS
cox.surv.clust <- coxph(Surv(clin.dat$time..years*12, clin.dat$Censor)~Hclust)
chk <- cox.zph(cox.surv.clust,global=T)
plot(chk)
# PFS
cox.pfs.clust <- coxph(Surv(clin.dat$PFS..MONTHS., clin.dat$PFScens)~Hclust)
chk.pfs <- cox.zph(cox.surv.clust,global=T)
plot(chk.pfs)
abline(0,0)


############################
#####  LOGRANK TESTS  #####
############################

# OS
survdiff(Surv(clin.dat$time..years*12, clin.dat$Censor)~Hclust)
# PFS
survdiff(Surv(clin.dat$PFS..MONTHS, clin.dat$PFScens)~Hclust


########################
#####  COX MODEL  #####
########################

cox.pfs.clust <- coxph(Surv(clin.dat$PFS..MONTHS, clin.dat$PFScens)~Hclust)
summary(cox.pfs.clust)
```

C.3 WECCA

```
#source("http://bioconductor.org/biocLite.R")
#biocLite("DNAcopy")
library(DNAcopy)
library(RJaCGH)
library(survival)

setwd("~/Documents/Class/Engler/Code/Simulations")
source("Willenbrock.R")
source("CGHRegions.R")
source("WECCA.functions.R")
```

```
################################################
#####  RUN "Data" CODE TO GET LOG-RATIOS  #####  See Appendix C.1.
################################################

# Impute missing value with the mean of adjacent observations
c <- which(apply(is.na(ovca.log2),2,sum)==1)
r <- which(apply(is.na(ovca.log2),1,sum)==1)
ovca.log2[r,c] <- (ovca.log2[r-1,c] + ovca.log2[r+1,c])/2


################################################
#####  GET CALLED DATA FOR WECCA INPUT  #####
################################################

#####  IDENTIFY REGIONS OF COPY NUMBER ALTERATION  #####

# Find segments and mean intensity for each subject
# (First 5 columns of ovca.log2 contain genomic location information)
cna <- CNA(genomdat=as.matrix(ovca.log2[,-(1:5)]), chrom=ovca.log2[,2],
                maploc=c(1:dim(ovca.log2)[1]), data.type="logratio", sampleid=names(ovca)[-(1:5)])

seg <- segment(cna) # 17373 x 6

# Make vectors of segmented means that correspond to markers in original data set
# Reorder segmented means to correspond to location on the genome
# Expand segmented means by number of markers with the same mean

samp.alpha <- tapply(seg$out$ID, seg$out$ID, length) # number of rows by sample (in alphabetical order)
alpha <- order(unique(seg$out$ID)) # alphabetical index
samporder <- order(alpha) # original sample order
samp <- samp.alpha[samporder] # number of rows by sample (in original order)
newcat <- rep(c(1:105),samp) # new category corresponding to ID to preserve original sample order
segcat <- cbind(newcat, seg$out)
ordcat <- order(segcat$newcat, segcat$loc.start) # orders rows by start location within samples
ordseg <- seg$out[ordcat,]

numsamp <- length(samp) # number of samples
cumsamp <- cumsum(samp) # cumulative number of rows by sample
newsamp <- c(0,cumsamp[1:numsamp-1]) + 1 # index to begin new sample
ovca2 <- ovca.log2[,] # will replace log2 ratios with segmented means

for (i in 1:numsamp){
        ovca2[,i+5] <- rep(ordseg$seg.mean[newsamp[i]:cumsamp[i]],ordseg$num.mark[newsamp[i]:cumsamp[i]])}

# Threshold for each subject - 1.11*MMAD
# MMAD - median of MAD across subjects
# MAD - median absolute distance of log2 values from segmented means

dev <- ovca.log2[,]
dev[,-(1:5)] <- abs(ovca.log2[,-(1:5)] - ovca2[,-(1:5)])

MMAD <- NULL
for (i in 1:numsamp){
        MAD <- NULL
        for (k in newsamp[i]:cumsamp[i]){
                index <- ifelse(i>1, k-cumsamp[i-1], k)
                MAD[index] <- median(dev[ordseg$loc.start[k]:ordseg$loc.end[k],i+5]) }
        MMAD[i] <- median(MAD) }

scalar <- 1.11   # 1.48*0.75

thresh <- scalar*MMAD
longthresh <- rep(thresh, samp)

gainloss <- NULL
for (k in 1:dim(ordseg)[1]){
        if (ordseg$seg.mean[k] < -longthresh[k]) class <- -1
        else if (ordseg$seg.mean[k] > longthresh[k]) class <- 1
```

```
        else class <- 0
        gainloss[k] <- class }

sum(gainloss[gainloss==1]) # 5158, 5881
length(gainloss[gainloss==-1]) # 4968, 5775

ovca3 <- ovca[,] # will replace log2 ratios with copy number gain/loss for threshold scalar 1.11
for (i in 1:numsamp){
        ovca3[,i+5] <- rep(gainloss[newsamp[i]:cumsamp[i]],ordseg$num.mark[newsamp[i]:cumsamp[i]])}

write.csv(ovca2, file="OVCA_segments.csv", row.names=F)
write.csv(ovca3, file="OVCA_gainloss.csv", row.names=F) # called data for 105 samples
save.image("segment.RData")

#####  PLOT LOG-RATIOS BY SUBJECT  #####

for (i in 6:length(names(ovca.log2))){

        ylo <- min(ovca.log2[,i]) - 0.1
        yhi <- max(ovca.log2[,i]) + 0.1
        gain <- which(ovca3[,i]==1)
        loss <- which(ovca3[,i]==-1)
        none <- which(ovca3[,i]==0)

        postscript(paste(names(ovca.log2)[i],'.ps',sep=""), height=10, width=10)
        plot(c(1:length(ovca.log2[,i])), ovca.log2[,i], pch=20, cex=0.5, xlim=c(0,99264),
            ylim=c(ylo,yhi), xaxt="n", col="grey", xlab="Marker Position",
            ylab=expression(paste(Log[2]," Intensities")), main=names(ovca)[i])
        axis(1, tick=T, at=c('10000','30000','50000','70000','90000'))
        points(loss, ovca.log2[loss,i], pch=20, cex=0.5, col="red3")
        points(gain, ovca.log2[gain,i], pch=20, cex=0.5, col="springgreen4")
        abline(0,0)
        dev.off()
}

# Segment mean before imputation
ordseg[(ordseg$ID=="CS1070_4447a") & (r > ordseg$loc.start) & (r < ordseg$loc.end),]
# Segment mean after imputation
ordseg[(ordseg$ID=="CS1070_4447a") & (r > ordseg$loc.start) & (r < ordseg$loc.end),]


################################################
#####  FORMAT DATA FOR CGHregions, WECCA  #####
################################################

# Restrict called data to 74 serous histology patients
ovca.cat <- read.csv("OVCA_gainloss.csv", header=T)
OVCA.78.cat <- ovca.cat[,-(1:5)][,clinical.samp]
OVCA.74.cat <- OVCA.78.cat[,clinical.dat$Histology=="Serous"]

# Get the number portion of the chromosome
chr <- substr(paste(ovca.log2$ChrName),start=4,stop=nchar(paste(ovca.log2$ChrName)))
chr.num <- as.numeric(ifelse(chr=="X",23,chr))

CBS.res.classify <- OVCA.74.cat

chrom <- chr.num
index <- (1:length(chr))
acghY <- OVCA.74.log2

data.log2.formatted <- cbind(index,chrom,index,acghY)
data.classify.formatted <- cbind(index,chrom,index,CBS.res.classify)

write.table(data.classify.formatted,"serous.classify.csv",sep=",",quote=F,row.names=F,col.names=F)
data.classify.formatted <- read.table("serous.classify.csv",sep=",",header=F)

write.table(CBS.res.classify,"serous.classify.dataonly.csv",sep=",",quote=F,row.names=F,col.names=F)
data.classify.dataonly <- read.table("serous.classify.dataonly.csv",sep=",",header=F)
```

```
#CGHdata <- AtM.log2.formatted # doesn't work - only works on aCGH calls (classifications)
CGHdata <- data.classify.formatted


####################################
#####  RUN CGH Regions SCRIPT  #####
####################################


##############################
#####  RUN WECCA SCRIPT  #####
##############################

save.image("WECCA.serous.Rdata")
load("WECCA.serous.Rdata")


##################################
#####  CLUSTER ASSIGNMENTS   #####
##################################

postscript("Wdend74.ps", height=23, width=40)
par(cex=4)
plot(dendrogram, xlab="", sub="", main="", labels=clinical.74$Sample, cex=0.6)
mtext("Cluster 1", side=1, line=-.8, cex=4, at=26)
mtext("Cluster 2", side=1, line=-.8, cex=4, at=58)
dev.off()

Wcluster <- cutree(dendrogram,k=2)
table(Wcluster) # one patient is an entire cluster

Wcluster <- cutree(dendrogram,k=3)
table(Wcluster)

# Investigate the single patient
only1 <- which(Wcluster==3)
clin.dat$Sample[69] # CS2688

# Is CS2688 clinically very different from the other patients?
Wclust73 <- Wcluster[-only1]
clin.dat73 <- clin.dat[-only1,]
coxph(Surv(clin.dat73$PFS..MONTHS, clin.dat73$PFScens)~Wclust73) # not significant

# Combined with Cluster 1
Wclust13 <- Wcluster
Wclust13[only1] <- 1
coxph(Surv(clin.dat$PFS..MONTHS, clin.dat$PFScens)~Wclust13) # no difference

# Combined with Cluster 2
Wclust23 <- Wcluster
Wclust23[only1] <- 2
coxph(Surv(clin.dat$PFS..MONTHS, clin.dat$PFScens)~Wclust23) # no difference

# Put CS2688 in Cluster 1 -- Now use Wclust13
clin.dat[clin.dat$Sample=="CS2688",]
summary(clin.dat[clin.dat$Sample!="CS2688",])
# Can't see any reason to exclude this patient...


############################################################
#####   RELATION OF CLUSTERS TO CLINICAL VARIABLES   #####
############################################################

clin.dat <- clinical.74
Wclust <- Wclust13 - 1
table(Wclust)
```

```
# Continuous covariates
summary(glm(Wclust ~ Age.at.dx, family=binomial, data=clin.dat))$coef[2,4]
summary(glm(Wclust ~ stage.number, family=binomial, data=clin.dat))$coef[2,4]

# Categorical covariates
summary(clin.dat)
covf.i <- c(8,10:12,16,20,25,28,30:38,43:48,52:53) # should be factor variables
covf <- names(clin.dat)[covf.i]
summary(clin.dat[covf.i])

# Change n/a and missing values to NA - then R will eliminate these from analysis
clin.dat$Grade[clin.dat$Grade %in% c("","n/a")] <- NA
clin.dat$Outcome[clin.dat$Outcome %in% c("","n/a")] <- NA
clin.dat$Recur[clin.dat$Recur %in% c("","n/a")] <- NA
summary(clin.dat[covf.i])

# Logistic regression
clust.covf.pval <- NULL
for (i in 1:length(covf)){
        clust.covf.pval[i] <- summary(glm(Wclust ~
                        as.factor(get(covf[i])), family=binomial, data=clin.dat))$coef[2,4]}

cbind(data.frame(covf[which(clust.covf.pval < 0.05)]), clust.covf.pval[clust.covf.pval < 0.05])
allpval <- cbind(data.frame(covf), clust.covf.pval)
allpval[order(allpval[,2]),] # all p-values whose variables were treated as categorical
# continuous variables have smaller p-values, but neither are significant

##########################################
#####  KAPLAN-MEIER SURVIVAL CURVES  #####
##########################################

#####  OVERALL SURVIVAL  #####

time <- 12*clin.dat$time..years. # years to months
cens <- clin.dat$Censor

# Split data into groups according to clustering
grp1 <- clin.dat[Wclust==1,]
grp0 <- clin.dat[Wclust==0,]

# Create complete survival information for each group
time1 <- 12*grp1$time..years.
cens1 <- grp1$Censor
time0 <- 12*grp0$time..years.
cens0 <- grp0$Censor

# Get K-M estimates
grp1.km <- summary(survfit(Surv(time1,cens1)~1, type="kaplan-meier"))
grp0.km <- summary(survfit(Surv(time0,cens0)~1, type="kaplan-meier"))

# Add first time point
grp1.time <- c(0, grp1.km$time)
grp1.surv <- c(1, grp1.km$surv)
grp0.time <- c(0, grp0.km$time)
grp0.surv <- c(1, grp0.km$surv)

# Confidence intervals
g1.upper <- c(1,grp1.km$upper)
g1.lower <- c(1,grp1.km$lower)
g0.upper <- c(1,grp0.km$upper)
g0.lower <- c(1,grp0.km$lower)

maxtime <- max(c(grp1.time,grp0.time))

postscript("Surv-Wclust.ps", height=10, width=10)
plot(grp0.time, grp0.surv, ylim=c(0,1), xlim=c(0,maxtime), type="s", lty=1, col="red",
        xlab="Months", ylab="Overall Survival", main="",
        cex.lab=1.4, cex.main=1.4, cex.axis=1.4, lwd=2)
```

```
lines(grp1.time, grp1.surv, ylim=c(0,1), type="s", lty=1, col="blue", lwd=2)
lines(grp0.time, g0.lower, lty=3, col="red", lwd=3)
lines(grp0.time, g0.upper, lty=3, col="red", lwd=3)
lines(grp1.time, g1.lower, lty=3, col="blue", lwd=3)
lines(grp1.time, g1.upper, lty=3, col="blue", lwd=3)
legend(-0.05, 0.18, c("Cluster 1","Cluster 2","95% Conf. Limits"),
       lty=c(1,1,3), col=c("red","blue","black"), bty="n", cex=1.3, lwd=c(2,2,3))
dev.off()

#####  PROGRESSION-FREE SURVIVAL  #####

time <- clin.dat$PFS..MONTHS.
cens <- clin.dat$PFScens

grp1 <- clin.dat[Wclust==1,]
grp0 <- clin.dat[Wclust==0,]

time1 <- grp1$PFS..MONTHS.
cens1 <- grp1$PFScens
time0 <- grp0$PFS..MONTHS.
cens0 <- grp0$PFScens

grp1.km <- summary(survfit(Surv(time1,cens1)~1, type="kaplan-meier"))
grp0.km <- summary(survfit(Surv(time0,cens0)~1, type="kaplan-meier"))

grp1.time <- c(0, grp1.km$time)
grp1.surv <- c(1, grp1.km$surv)
grp0.time <- c(0, grp0.km$time)
grp0.surv <- c(1, grp0.km$surv)

g1.upper <- c(1,grp1.km$upper)
g1.lower <- c(1,grp1.km$lower)
g0.upper <- c(1,grp0.km$upper)
g0.lower <- c(1,grp0.km$lower)

maxtime <- max(c(grp1.time,grp0.time))

postscript("PFS-Wclust.ps", height=10, width=10)
plot(grp0.time, grp0.surv, ylim=c(0,1), xlim=c(0,maxtime), type="s", lty=1, col="red",
       xlab="Months", ylab="Progression-Free Survival", main="",
       cex.lab=1.4, cex.main=1.4, cex.axis=1.4, lwd=2)
lines(grp1.time, grp1.surv, ylim=c(0,1), type="s", lty=1, col="blue", lwd=2)
lines(grp0.time, g0.lower, lty=3, col="red", lwd=3)
lines(grp0.time, g0.upper, lty=3, col="red", lwd=3)
lines(grp1.time, g1.lower, lty=3, col="blue", lwd=3)
lines(grp1.time, g1.upper, lty=3, col="blue", lwd=3)
legend(0.7*maxtime, 1.04, c("Cluster 1","Cluster 2","95% Conf. Limits"),
       lty=c(1,1,3), col=c("red","blue","black"), bty="n", cex=1.3, lwd=c(2,2,3))
dev.off()

# Additional check of proportional hazards assumption - Schoenfeld residuals
# OS
cox.surv.clust <- coxph(Surv(clin.dat$time..years*12, clin.dat$Censor)~Wclust)
chk <- cox.zph(cox.surv.clust,global=T)
plot(chk)
# PFS
cox.pfs.clust <- coxph(Surv(clin.dat$PFS..MONTHS., clin.dat$PFScens)~Wclust)
chk.pfs <- cox.zph(cox.surv.clust,global=T)
plot(chk.pfs)

############################
#####  LOGRANK TESTS  #####
############################

# OS
survdiff(Surv(12*clin.dat$time..years., clin.dat$Censor)~Wclust)
# PS
survdiff(Surv(clin.dat$PFS..MONTHS, clin.dat$PFScens)~Wclust)
```

C.4 Comparison of HC and WECCA

```
##############################
#####  COMPARE SURVIVAL  #####
##############################

Hgrp1 <- clin.dat[Hclust==1,]
Hgrp0 <- clin.dat[Hclust==0,]
Wgrp1 <- clin.dat[Wclust==1,]
Wgrp0 <- clin.dat[Wclust==0,]

Htime1 <- Hgrp1$PFS..MONTHS.
Hcens1 <- Hgrp1$PFScens
Htime0 <- Hgrp0$PFS..MONTHS.
Hcens0 <- Hgrp0$PFScens
Wtime1 <- Wgrp1$PFS..MONTHS.
Wcens1 <- Wgrp1$PFScens
Wtime0 <- Wgrp0$PFS..MONTHS.
Wcens0 <- Wgrp0$PFScens

Hgrp1.km <- summary(survfit(Surv(Htime1,Hcens1)~1, type="kaplan-meier"))
Hgrp0.km <- summary(survfit(Surv(Htime0,Hcens0)~1, type="kaplan-meier"))
Wgrp1.km <- summary(survfit(Surv(Wtime1,Wcens1)~1, type="kaplan-meier"))
Wgrp0.km <- summary(survfit(Surv(Wtime0,Wcens0)~1, type="kaplan-meier"))

Hgrp1.time <- c(0, Hgrp1.km$time)
Hgrp1.surv <- c(1, Hgrp1.km$surv)
Hgrp0.time <- c(0, Hgrp0.km$time)
Hgrp0.surv <- c(1, Hgrp0.km$surv)
Wgrp1.time <- c(0, Wgrp1.km$time)
Wgrp1.surv <- c(1, Wgrp1.km$surv)
Wgrp0.time <- c(0, Wgrp0.km$time)
Wgrp0.surv <- c(1, Wgrp0.km$surv)

maxtime <- max(c(Hgrp1.time,Hgrp0.time,Wgrp1.time,Wgrp0.time))

par(cex=1)

postscript("HWclustPFS.ps", height=10, width=10)
plot(Hgrp1.time, Hgrp1.surv, ylim=c(0,1), xlim=c(0,maxtime), type="s", lty=2, col="blue",
        xlab="Months", ylab="Progression-Free Survival", main="",
        cex.lab=1.4, cex.main=1.4, cex.axis=1.4)
lines(Hgrp0.time, Hgrp0.surv, ylim=c(0,1), type="s", lty=1, col="blue")
lines(Wgrp1.time, Wgrp1.surv, ylim=c(0,1), type="s", lty=2, col="red")
lines(Wgrp0.time, Wgrp0.surv, ylim=c(0,1), type="s", lty=1, col="red")
legend(0.55*maxtime, 1.03, c("HC","WECCA"), text.col=c("blue","red"), bty="n", cex=1.3)
legend(0.77*maxtime, 1.03, c("Cluster 1","Cluster 2"), lty=c(1,2), bty="n", cex=1.3)
dev.off()

########################################
#####  COMPARE CLUSTER COMPOSITION  #####
########################################

hc.ovca$labels # cluster assignments are displayed in the original order of the data samples being clustered
HWclust <- data.frame(cbind(Hclust,Wcluster))
HWdiff <- HWclust[HWclust$Hclust!=HWclust$Wcluster,]
HWdiff

# For the samples that were clustered differently by WECCA, where are they clustered in HC? all over, no pattern
plot(hc.ovca,cex=0.55,main="Hierarchical Clustering",xlab="",ann=F) # labels identify samples in cluster

dev.new()
plot(hc.ovca,cex=0.55,xlab="",axes=F,ann=F,label=F)
title(main="Hierarchical Clustering")
points(9,95,pch=20,col="red") # 69 W-3
points(35,127,pch=20,col="blue")
points(36,127,pch=20,col="blue")
```

```
points(11,93,pch=20,col="blue")
points(14,150,pch=20,col="blue")
points(62,127,pch=20,col="purple")
points(47,240,pch=20,col="purple")
points(45,375,pch=20,col="purple")
points(55,105,pch=20,col="purple")
points(56,105,pch=20,col="purple")
points(46,375,pch=20,col="purple")
text(4.5,645,c("WECCA"),bty="n")
text(4.5,625,c("Clusters"),bty="n")
points(4,600,pch=20,col="purple")
points(4,580,pch=20,col="blue")
points(4,560,pch=20,col="red")
text(5.5,600,"1")
text(5.5,580,"2")
text(5.5,560,"3")
```

APPENDIX D

SIMULATION CODE

The first section of the simulation code generates the empirical data that is used for generating aCGH samples. The log-ratio data is transformed into segmented means with associated classifications of loss, gain, or no change. Included in the Simulations code section is the function used to create a single distance matrix for the RDU method. Following is the code to run 1,000 simulations and save the cluster assignments for each clustering method. The CGHregions and WECCA scripts in Appendix B should be submitted before running the WECCA simulations.

D.1 Empirical Data

```
library(DNAcopy)
library(RJaCGH)

setwd("~/Documents/Class/Engler/Code/Simulations")
source("Willenbrock.R")
source("CGHRegions.R")
source("WECCA.functions.R")

####################################
#####  SET UP EMPIRICAL DATA  #####
####################################

#####  READ DATA  #####

#source("http://bioconductor.org/biocLite.R")
#biocLite("DNAcopy")
library(DNAcopy)

ovca.log2 <- read.csv("OVCA_log2Ratios.csv", header=T)
dim(ovca.log2) # 99264 x 110 (markers x samples)
summary(ovca.log2) # one missing value in CS1070_4447a

c <- which(apply(is.na(ovca.log2),2,sum)==1)
r <- which(apply(is.na(ovca.log2),1,sum)==1)

# Impute missing value with the mean of adjacent observations
ovca.log2[r,c] <- (ovca.log2[r-1,c] + ovca.log2[r+1,c])/2

#####  IDENTIFY REGIONS OF COPY NUMBER ALTERATION  #####

# Find segments and mean intensity for each subject
cna <- CNA(genomdat=as.matrix(ovca.log2[,-(1:5)]), chrom=ovca.log2[,2],
        maploc=c(1:dim(ovca.log2)[1]), data.type="logratio", sampleid=names(ovca)[-(1:5)])

seg <- segment(cna) # 17373 x 6

# Make vectors of segmented means that correspond to markers in original data set
# Reorder segmented means to correspond to location on the genome
# Expand segmented means by number of markers with the same mean

samp.alpha <- tapply(seg$out$ID, seg$out$ID, length) # number of rows by sample (in alphabetical order)
alpha <- order(unique(seg$out$ID)) # alphabetical index
samporder <- order(alpha) # original sample order
samp <- samp.alpha[samporder] # number of rows by sample (in original order)
```

```
newcat <- rep(c(1:105),samp) # new category corresponding to ID to preserve original sample order
segcat <- cbind(newcat, seg$out)
ordcat <- order(segcat$newcat, segcat$loc.start) # orders rows by start location within samples
ordseg <- seg$out[ordcat,]

numsamp <- length(samp) # number of samples
cumsamp <- cumsum(samp) # cumulative number of rows by sample
newsamp <- c(0,cumsamp[1:numsamp-1]) + 1 # index to begin new sample
ovca2 <- ovca.log2[,] # will replace log2 ratios with segmented means

for (i in 1:numsamp){
        ovca2[,i+5] <- rep(ordseg$seg.mean[newsamp[i]:cumsamp[i]], ordseg$num.mark[newsamp[i]:cumsamp[i]]) }

# Threshold for each subject - 1.11*MMAD
# MMAD - median of MAD across subjects
# MAD - median absolute distance of log2 values from segmented means

dev <- ovca.log2[,]
dev[,-(1:5)] <- abs(ovca.log2[,-(1:5)] - ovca2[,-(1:5)])

MMAD <- NULL
for (i in 1:numsamp){
        MAD <- NULL
        for (k in newsamp[i]:cumsamp[i]){
                index <- ifelse(i>1, k-cumsamp[i-1], k)
                MAD[index] <- median(dev[ordseg$loc.start[k]:ordseg$loc.end[k],i+5]) }
        MMAD[i] <- median(MAD) }

scalar <- 1.11   # 1.48*0.75

thresh <- scalar*MMAD
longthresh <- rep(thresh, samp)

gainloss <- NULL
for (k in 1:dim(ordseg)[1]){
        if (ordseg$seg.mean[k] < -longthresh[k]) class <- -1
        else if (ordseg$seg.mean[k] > longthresh[k]) class <- 1
        else class <- 0
        gainloss[k] <- class }

sum(gainloss[gainloss==1]) # 5158, 5881
length(gainloss[gainloss==-1]) # 4968, 5775

ovca3 <- ovca[,] # will replace log2 ratios with copy number gain/loss for threshold scalar 1.11
for (i in 1:numsamp){
        ovca3[,i+5] <- rep(gainloss[newsamp[i]:cumsamp[i]], ordseg$num.mark[newsamp[i]:cumsamp[i]]) }

write.csv(ovca2, file="OVCA_segments.csv", row.names=F)
write.csv(ovca3, file="OVCA_gainloss.csv", row.names=F)
save.image("segment.RData")

ovca.log2 <- read.csv("OVCA_log2Ratios.csv", header=T)
ovca.means <- read.csv("OVCA_segments.csv", header=T)
ovca.class <- read.csv("OVCA_gainloss.csv",header=T)

#####  KEEP SAMPLES WITH CLINICAL INFO (78)  #####

id <- read.csv("OVCA_SampleList_WithChipTag.csv", header=T) # 78 x 3
info <- read.csv("Ovarian database Combined WBG edit 7.3.09l.csv", skip=1, header=T)
info <- info[-(79:86),] # 78 x 54, clinical info
orderchip <- order(info$Chip.)
orderorig <- order(order(id$Chip.ID))
newinfo <- info[orderchip,]
clinical.dat <- ordinfo <- newinfo[orderorig,]
clinical.dat$Sample <- ordinfo$Sample <- id$Sample.ID

split <- strsplit(names(ovca.log2)[-(1:5)],"_")
split2 <- unlist(split) # alternating sample and chip IDs (length=105)
```

```
sq <- seq(1,length(split2))
ovcasamp <- split2[sq%%2 == 1]
clinical.samp <- which(ovcasamp %in% clinical.dat$Sample)

#####  RESTRICT ANALYSES TO "Serous"  #####

OVCA.log2 <- ovca.log2[,-(1:5)][,clinical.samp][,clinical.dat$Histology=="Serous"]
OVCA.means <- ovca.means[,-(1:5)][,clinical.samp][,clinical.dat$Histology=="Serous"]
OVCA.class <- ovca.class[,-(1:5)][,clinical.samp][,clinical.dat$Histology=="Serous"]

####  PUT ASSOCIATED SEGMENTED MEANS, LENGTHS, AND CLASSIFICATIONS IN A VECTOR  #####

# For all clinical samples
seg.mean.all <- seg.length.all <- seg.classify.all <- NULL

# Loop through samples
for (i in 1:dim(OVCA.means)[2]){

        # find where new segments begin
        colseg <- OVCA.means[,i]
        tmp <- c(0,colseg[-length(colseg)])
        sameseg <- tmp-colseg # indicator: 0=same segment, not zero=different
        newseg <- ifelse(sameseg!=0,1,0)

        # get segment means for one sample
        newseg.index <- which(newseg==1)
        seg.mean <- colseg[newseg.index]

        # get lengths of each segment for one sample
        tmp2 <- c(newseg.index[-1],length(colseg)+1)
        seg.length <- tmp2 - newseg.index

        # get associated gains and losses with each segment
        colclass <- OVCA.class[,i]
        seg.classify <- colclass[newseg.index]

        # store information for this sample
        seg.mean.all <- c(seg.mean.all, seg.mean)
        seg.length.all <- c(seg.length.all, seg.length)
        seg.classify.all <- c(seg.classify.all, seg.classify)
        }

write.table(seg.mean.all,"seg.mean.74.csv",sep=",",quote=F,row.names=F,col.names=F)
write.table(seg.length.all,"seg.length.74.csv",sep=",",quote=F,row.names=F,col.names=F)
write.table(seg.classify.all,"seg.classify.74.csv",sep=",",quote=F,row.names=F,col.names=F)
```

D.2 Simulations

```
seg.mean <- c(as.vector(read.csv("seg.mean.74.csv",header=F)))[[1]]
seg.length <- c(as.vector(read.csv("seg.length.74.csv",header=F)))[[1]]
seg.classify <- c(as.vector(read.csv("seg.classify.74.csv",header=F)))[[1]]


################################################
#####  FUNCTION FOR RDU DISTANCE MATRIX  #####
################################################

# Get a single distance matrix incorporating both gains and losses.
# First check if there is at least 1 common region found for gains and for losses.
# Next, order rows and columns in numeric order to match Hclust and WECCA. Then add.

distgainloss <- function(reg.gain,reg.loss){

        if (length(reg.gain[1][[1]])>0) {
                dmat.gain <- plot(reg.gain)$probes
                names.gain <- colnames(dmat.gain)
```

```
                order.gain <- order(as.numeric(unlist(strsplit(names.gain,split=" "))[(1:20)%%2==0]))
                names.gain[order.gain]
                new.dmat.gain <- dmat.gain[order.gain,order.gain]
          } else { new.dmat.gain <- matrix(0,10,10) }

        if (length(reg.loss[1][[1]])>0) {
                dmat.loss <- plot(reg.loss)$probes
                names.loss <- colnames(dmat.loss)
                order.loss <- order(as.numeric(unlist(strsplit(names.loss,split=" "))[(1:20)%%2==0]))
                new.dmat.loss <- dmat.loss[order.loss,order.loss]
          } else { new.dmat.loss <- matrix(0,10,10) }

        dmat.alt <- new.dmat.gain + new.dmat.loss
        return(dmat.alt)
}


#########################
#####  SIMULATIONS  #####
#########################

# Generates 10 sample profiles for each iteration
# - 2 groups of 5 samples, 3000 markers in length

for (ii in 1:1000) {

  set.seed(ii)

  ######################################################################
  #############  Simulate Data (Willenbrock approach)  ###############

  n <- 5 # number of samples in one group
  nclone <- 600 # number of markers per chromosome
  #noise.fixed <- sqrt(.25)
  noise.fixed <- c(sqrt(.20),sqrt(.30),sqrt(.40))
  p.random <- runif(1,.7,.95)

  # group 1, chrom1
  y1.1 <- gen.data.Willenbrock(segmean=seg.mean,seglength=seg.length,
                               segclassify=seg.classify,
                               n.sample=n,n.clone=nclone,noise=noise.fixed,
                               rL1=c(1:150),rL1p=p.random,
                               rL2=0,rL2p=0,
                               rG1=0,rG1p=0,
                               rG2=0,rG2p=0,
                               r01=c(151:600),r01p=p.random,
                               r02=0,r02p=0,
                               r03=0,r03p=0,
                               r04=0,r04p=0,
                               r05=0,r05p=0)

  set1.1.chrom1 <- y1.1[[1]]
  set1.1.truth.chrom1 <- y1.1[[2]]

  # group 1, chrom2
  y1.1 <- gen.data.Willenbrock(segmean=seg.mean,seglength=seg.length,
                               segclassify=seg.classify,
                               n.sample=n,n.clone=nclone,noise=noise.fixed,
                               rL1=0,rL1p=0,
                               rL2=0,rL2p=0,
                               rG1=0,rG1p=0,
                               rG2=0,rG2p=0,
                               r01=c(1:600),r01p=p.random,
                               r02=0,r02p=0,
                               r03=0,r03p=0,
                               r04=0,r04p=0,
                               r05=0,r05p=0)
```

```
set1.1.chrom2 <- y1.1[[1]]
set1.1.truth.chrom2 <- y1.1[[2]]

# group 1, chrom3
y1.1 <- gen.data.Willenbrock(segmean=seg.mean,seglength=seg.length,
                             segclassify=seg.classify,
                             n.sample=n,n.clone=nclone,noise=noise.fixed,
                             rL1=0,rL1p=0,
                             rL2=0,rL2p=0,
                             rG1=c(301:600),rG1p=p.random,
                             rG2=0,rG2p=0,
                             r01=c(1:300),r01p=p.random,
                             r02=0,r02p=0,
                             r03=0,r03p=0,
                             r04=0,r04p=0,
                             r05=0,r05p=0)

set1.1.chrom3 <- y1.1[[1]]
set1.1.truth.chrom3 <- y1.1[[2]]

# group 1, chrom4
y1.1 <- gen.data.Willenbrock(segmean=seg.mean,seglength=seg.length,
                             segclassify=seg.classify,
                             n.sample=n,n.clone=nclone,noise=noise.fixed,
                             rL1=0,rL1p=0,
                             rL2=0,rL2p=0,
                             rG1=0,rG1p=0,
                             rG2=0,rG2p=0,
                             r01=c(1:600),r01p=p.random,
                             r02=0,r02p=0,
                             r03=0,r03p=0,
                             r04=0,r04p=0,
                             r05=0,r05p=0)

set1.1.chrom4 <- y1.1[[1]]
set1.1.truth.chrom4 <- y1.1[[2]]

# group 1, chrom5
y1.1 <- gen.data.Willenbrock(segmean=seg.mean,seglength=seg.length,
                             segclassify=seg.classify,
                             n.sample=n,n.clone=nclone,noise=noise.fixed,
                             rL1=0,rL1p=0,
                             rL2=0,rL2p=0,
                             rG1=0,rG1p=0,
                             rG2=0,rG2p=0,
                             r01=c(1:600),r01p=p.random,
                             r02=0,r02p=0,
                             r03=0,r03p=0,
                             r04=0,r04p=0,
                             r05=0,r05p=0)

set1.1.chrom5 <- y1.1[[1]]
set1.1.truth.chrom5 <- y1.1[[2]]

###############

set1.1 <- rbind(set1.1.chrom1,
                set1.1.chrom2,
                set1.1.chrom3,
                set1.1.chrom4,
                set1.1.chrom5)

set1.1.truth <- rbind(set1.1.truth.chrom1,
                      set1.1.truth.chrom2,
                      set1.1.truth.chrom3,
                      set1.1.truth.chrom4,
                      set1.1.truth.chrom5)
###############
```

```
# group 2, chrom1
y2.1 <- gen.data.Willenbrock(segmean=seg.mean,seglength=seg.length,
                             segclassify=seg.classify,
                             n.sample=n,n.clone=nclone,noise=noise.fixed,
                             rL1=0,rL1p=0,
                             rL2=0,rL2p=0,
                             rG1=0,rG1p=0,
                             rG2=0,rG2p=0,
                             r01=c(1:600),r01p=p.random,
                             r02=0,r02p=0,
                             r03=0,r03p=0,
                             r04=0,r04p=0,
                             r05=0,r05p=0)

set2.1.chrom1 <- y2.1[[1]]
set2.1.truth.chrom1 <- y2.1[[2]]

# group 2, chrom2
y2.1 <- gen.data.Willenbrock(segmean=seg.mean,seglength=seg.length,
                             segclassify=seg.classify,
                             n.sample=n,n.clone=nclone,noise=noise.fixed,
                             rL1=c(1:100),rL1p=p.random,
                             rL2=0,rL2p=0,
                             rG1=0,rG1p=0,
                             rG2=0,rG2p=0,
                             r01=c(101:600),r01p=p.random,
                             r02=0,r02p=0,
                             r03=0,r03p=0,
                             r04=0,r04p=0,
                             r05=0,r05p=0)

set2.1.chrom2 <- y2.1[[1]]
set2.1.truth.chrom2 <- y2.1[[2]]

# group 2, chrom3
y2.1 <- gen.data.Willenbrock(segmean=seg.mean,seglength=seg.length,
                             segclassify=seg.classify,
                             n.sample=n,n.clone=nclone,noise=noise.fixed,
                             rL1=0,rL1p=0,
                             rL2=0,rL2p=0,
                             rG1=0,rG1p=0,
                             rG2=0,rG2p=0,
                             r01=c(1:600),r01p=p.random,
                             r02=0,r02p=0,
                             r03=0,r03p=0,
                             r04=0,r04p=0,
                             r05=0,r05p=0)

set2.1.chrom3 <- y2.1[[1]]
set2.1.truth.chrom3 <- y2.1[[2]]

# group 2, chrom4
y2.1 <- gen.data.Willenbrock(segmean=seg.mean,seglength=seg.length,
                             segclassify=seg.classify,
                             n.sample=n,n.clone=nclone,noise=noise.fixed,
                             rL1=0,rL1p=0,
                             rL2=0,rL2p=0,
                             rG1=0,rG1p=0,
                             rG2=0,rG2p=0,
                             r01=c(1:600),r01p=p.random,
                             r02=0,r02p=0,
                             r03=0,r03p=0,
                             r04=0,r04p=0,
                             r05=0,r05p=0)

set2.1.chrom4 <- y2.1[[1]]
set2.1.truth.chrom4 <- y2.1[[2]]
```

```
# group 2, chrom5
y2.1 <- gen.data.Willenbrock(segmean=seg.mean,seglength=seg.length,
                             segclassify=seg.classify,
                             n.sample=n,n.clone=nclone,noise=noise.fixed,
                             rL1=c(1:50),rL1p=p.random,
                             rL2=0,rL2p=0,
                             rG1=0,rG1p=0,
                             rG2=0,rG2p=0,
                             r01=c(51:600),r01p=p.random,
                             r02=0,r02p=0,
                             r03=0,r03p=0,
                             r04=0,r04p=0,
                             r05=0,r05p=0)

set2.1.chrom5 <- y2.1[[1]]
set2.1.truth.chrom5 <- y2.1[[2]]

###############

set2.1 <- rbind(set2.1.chrom1,
                set2.1.chrom2,
                set2.1.chrom3,
                set2.1.chrom4,
                set2.1.chrom5)

set2.1.truth <- rbind(set2.1.truth.chrom1,
                      set2.1.truth.chrom2,
                      set2.1.truth.chrom3,
                      set2.1.truth.chrom4,
                      set2.1.truth.chrom5)
###############

group <- c(rep(0,n),rep(1,n))

set1 <- set1.1
set2 <- set2.1

set1.truth <- cbind(set1.1.truth)
set2.truth <- cbind(set2.1.truth)

sim.data <- cbind(set1[,3:(n+2)],set2[,3:(n+2)])
sim.classify <- cbind(set1.truth[,3:(n+2)],set2.truth[,3:(n+2)])

#plot(sim.data[,2])

#########################################################################
##########################  Clustering  #############################

#################
#####  HC  #####
#################

sim.dist <- dist(t(sim.data))
hc <- hclust(sim.dist, method="ward")


###################
#####  WECCA  #####
###################

# source("sim.WECCA.R")

#####  FORMAT DATA FOR CBS, CGHregions, WECCA  #####

chrom <- c(rep(1,nclone),rep(2,nclone),rep(3,nclone),rep(4,nclone),rep(5,nclone))
maploc <- c(1:length(chrom))
```

```
CBS.res.classify <- array(0,dim=dim(sim.data)) # CBS.res.smooth <-
n.tot  <- dim(sim.data)[2]

for(j in 1:n.tot) {

  genomdat <- sim.data[,j]
  index <- 1:length(chrom)
  lchrom.tot <- length(chrom)

  # create separate groupings of data, probabilities, by individual

  #lchrom <- lchrom.tot
  CNA.dat <- CNA(genomdat,chrom,index)
  seg.col <- segment(CNA.dat)
  olshen.res <- seg.col$output

  olshen.res.tot <- NA
  seg.index <- NA
  maploc.index <- 0
  for (kk in 1:dim(seg.col$output)[1]) {
    olshen.res.tot[(maploc.index + 1):(maploc.index + seg.col$output[kk,5])] <- seg.col$output[kk,6]
    seg.index[(maploc.index + 1):(maploc.index + seg.col$output[kk,5])] <- kk
    maploc.index <- maploc.index + seg.col$output[kk,5]
  }

  #####  MMAD method  #####

  ics.mmad <- NA
  seg.index <- NA
  maploc.index <- 0
  for (jjj in 1:dim(olshen.res)[1]) {
    ics.mmad[(maploc.index + 1):(maploc.index + olshen.res[jjj,5])] <- olshen.res[jjj,6]
    seg.index[(maploc.index + 1):(maploc.index + olshen.res[jjj,5])] <- jjj
    maploc.index <- maploc.index + olshen.res[jjj,5]
  }

  # chrom, log2 ratio, CBS seg val, ML val?, abs(log2 - CBS seg val), maploc, index
  ics.mmad.tot <- cbind(as.numeric(chrom), as.numeric(genomdat), as.numeric(ics.mmad),
                    as.numeric(seg.index),as.numeric(abs(genomdat-ics.mmad)),
                    as.numeric(maploc), as.numeric(index))

  abs.diff <- ics.mmad.tot[,5]
  MAD <- mad(abs.diff)
  scaled.MAD <- 1.48*MAD
  weight <- .75

  CBS.seg <- as.numeric(ics.mmad)

  MAD.loss <- sum(as.numeric(CBS.seg < -weight*scaled.MAD))         # loss
  MAD.nochange <- sum(as.numeric(abs(CBS.seg) < weight*scaled.MAD)) # no change
  MAD.gain <- sum(as.numeric(CBS.seg > weight*scaled.MAD))          # gain
  MAD.tot <- MAD.loss + MAD.gain
  ics.MAD.tot <- c(MAD.loss,MAD.gain,MAD.tot)

  log2 <- genomdat
  ML.res <- CBS.seg
  GL.indicator.loss <- GL.indicator.gain <- GL.indicator.nochange <- NA

  GL.indicator.loss <- ifelse(CBS.seg < -weight*scaled.MAD,1,0)
  GL.indicator.gain <- ifelse(CBS.seg > weight*scaled.MAD,1,0)
  GL.indicator.nochange <- ifelse(abs(CBS.seg) < weight*scaled.MAD,1,0)

  CBS.res.classify[,j] <- ifelse(GL.indicator.loss==1,-1,CBS.res.classify[,j])
  CBS.res.classify[,j] <- ifelse(GL.indicator.nochange==1,0,CBS.res.classify[,j])
  CBS.res.classify[,j] <- ifelse(GL.indicator.gain==1,1,CBS.res.classify[,j])

  #CBS.res.smooth[,j] <- CBS.seg
}
```

```
CGHdata <- cbind(maploc,chrom,maploc,CBS.res.classify)
colnames(CGHdata) <- NULL
CGHdata <- as.data.frame(CGHdata)

###################################
#####  RUN CGHregions SCRIPT  #####
###################################

###############################
#####  RUN WECCA SCRIPT  #####
###############################


##################
######  RDU  #####
##################

#source("sim.Rueda.R")

# 1) RJaCGH - produce marginal probabilities of gain and loss
# 2) pREC-S - produce joint probabilities of gain and loss for
#             regions common to subsets of samples'
# 3) Hier. clustering on number of common recurrent regions with certain prob of gain or loss

# Need chrom and position data
chrom <- c(rep(1,nclone),rep(2,nclone),rep(3,nclone),rep(4,nclone),rep(5,nclone))
maploc <- c(1:length(chrom))

# Position must be ordered within Chromosome. Data is already in this order.
order.simdata <- order(chrom, maploc)
Chrom <- chrom[order.simdata]
Position <- maploc[order.simdata]

# Find recurrent CNA regions. Return object has marginal probabilities of gain or loss.
sim.prob <- RJaCGH(y=sim.data, Pos=Position, Chrom=Chrom, model="genome")

# Find regions of gain (loss) with minimum joint probability of gain (loss) = .5, .65, and .8
reg.gain.5 <- pREC_S(sim.prob, p=0.5, freq.array=2, alteration="Gain")
reg.gain.65 <- pREC_S(sim.prob, p=0.65, freq.array=2, alteration="Gain")
reg.gain.8 <- pREC_S(sim.prob, p=0.8, freq.array=2, alteration="Gain")

reg.loss.5 <- pREC_S(sim.prob, p=0.5, freq.array=2, alteration="Loss")
reg.loss.65 <- pREC_S(sim.prob, p=0.65, freq.array=2, alteration="Loss")
reg.loss.8 <- pREC_S(sim.prob, p=0.8, freq.array=2, alteration="Loss")

# Get distance matrices
dmat.alt.5 <- distgainloss(reg.gain.5, reg.loss.5)
dmat.alt.65 <- distgainloss(reg.gain.65, reg.loss.65)
dmat.alt.8 <- distgainloss(reg.gain.8, reg.loss.8)

# Hierarchical clustering
dist.alt.5 <- as.dist(dmat.alt.5, diag=T, upper=T)
dist.alt.65 <- as.dist(dmat.alt.65, diag=T, upper=T)
dist.alt.8 <- as.dist(dmat.alt.8, diag=T, upper=T)

hc.rueda.5 <- hclust(dist.alt.5, method="complete")
hc.rueda.65 <- hclust(dist.alt.65, method="complete")
hc.rueda.8 <- hclust(dist.alt.8, method="complete")


###################################
#####  SAVE CLUSTER RESULTS  #####
###################################

# Number of clusters to extract from dendrogram
number.of.clusters <- 2
```

```
  # Cluster results
  Hclust <- cutree(hc, k=number.of.clusters) - 1
  Wclust <- cutree(hc.wecca, k=number.of.clusters) - 1
  Rclust.5 <- cutree(hc.rueda.5, k=number.of.clusters) - 1
  Rclust.65 <- cutree(hc.rueda.65, k=number.of.clusters) - 1
  Rclust.8 <- cutree(hc.rueda.8, k=number.of.clusters) - 1

  # Write to file
  write.table(t(c(ii,Hclust)),"Hres.txt",quote=F,row.names=F,col.names=F,append=T)
  write.table(t(c(ii,Wclust)),"Wres.txt",quote=F,row.names=F,col.names=F,append=T)
  write.table(t(c(ii,Rclust.5)),"Rres.5.txt",quote=F,row.names=F,col.names=F,append=T)
  write.table(t(c(ii,Rclust.65)),"Rres.65.txt",quote=F,row.names=F,col.names=F,append=T)
  write.table(t(c(ii,Rclust.8)),"Rres.8.txt",quote=F,row.names=F,col.names=F,append=T)

} ###### END SIMULATIONS #####
```