Modelling Variability in the Leader Algorithm Family: A Testable Model and Implementation

Barton, Alan

https://doi.org/10.4224/5763719

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC · CNRC

Modelling Variability in the Leader Algorithm Family: A Testable Model and Implementation *

Barton, A.
December 2004

* published as NRC/ERB-1119. 54 Pages. December 20, 2004. NRC 47429.

Modelling Variability in the Leader Algorithm Family: A Testable Model and Implementation

Alan J. Barton
Integrated Reasoning Group
Institute for Information Technology
National Research Council Canada
Ottawa, Canada, K1A 0R6
alan.barton@nrc-cnrc.gc.ca

December 20, 2004

Abstract

This final project report[1] is a Type *I*: Modelling variability into a testable model project. The main themes of the course revolve around the concepts of *variability* (in this project variability is associated with the variation of an algorithm in time i.e. evolutionary variability and with variability in definitions of concepts), *traceability* and *verification*. Hence these concepts form the core focus of this project, with more emphasis being placed on the creation of a traceable, testable domain model. In particular, two questions posed during the offering of this course are: *i)* "Can tests be used to document how family members are different?", and *ii)* "How can features and their dependencies be properly documented?"

[1]Submitted to Professor Jean-Pierre Corriveau in partial fulfillment of the requirements of the computer science course *COMP 5104 - Object Oriented Software Engineering* and to the National Research Council as an internal technical report.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

T_d	A dissimilarity (or distance) threshold
T_s	A similarity threshold (See §5.1)
X	The original (possibly huge) data set
$O(g(n))$	$= \{f(n) : \exists c, n_0 > 0, \text{ s.t. } 0 \leq f(n) \leq c \cdot g(n), \forall n \geq n_0\}$

Chapter 1

Introduction

The Method of Multiple Working Hypotheses – *T.C. Chamberlin 1890*[1]

There are many things in this world that are not *certainly* known or understood by humans. It is best, therefore, to consider multiple competing (variable) hypotheses[2] in order to attempt to understand[3] the inherent complexity in the world around us and to be able to disseminate that knowledge (communicate these hypothesis for others). In particular, it is assumed that the larger the variability in the set of hypotheses under consideration, then the higher the likelihood one of them may very accurately describe the complex world.

In the field of software engineering, one software system is usually built for a particular domain in order to help domain experts (users) understand some aspects of their domain. It has been proposed that it might be better to consider all possible systems (a family) that could be built for a particular domain, instead of only one system at a time (classical approach). This concept, called system family engineering, attempts to more explicitly explicate ("clearly explain" would be another variation of this word combination) the tradeoffs between design (implementation) alternatives, individually known as family members.

Further, the classical testing approach focuses on the consideration of one software system at a time (called the system under test). However, a competing approach considers testing from the point of view of a whole family of software systems and at the much more abstract and user centered domain modelling level. This approach leads to objective tests that are directly derived from a *model of the whole family of systems*.

[1]Thomas C. Chamberlin (1843-1928) wrote about the method of multiple working hypotheses in [4].
[2]Multiple hypotheses could also be called a family of hypotheses.
[3]Objectivity is chosen over subjectivity (or intuition) whenever possible in this project.

Chapter 2

Project

The family of algorithms that will be investigated, as the domain under consideration, are related to clustering data. That is, the *Leader Algorithm* originally proposed by John A. Hartigan in [10] along with variants that Dr. Julio Valdés proposed will be investigated. A fortran implementation of the original algorithm is available in [10] and a variant was made available by Dr. Julio Valdés in ANSI C. This project proposal is a research topic in its domain, as well as being a research topic for the purposes of the course.

Definition 1 (Program Family) *In 1976 David L. Parnas [12] wrote that:*
We consider a set of programs to constitute a family, whenever it is worthwhile to study programs from the set by first studying the common properties of the set and then determining the special properties of the individual family members.

2.1 Project Conceptualization

The project aims to investigate how abstract models of a domain (in this case *Testable Feature Contracts*) can lead, via a structured and systematic approach to abstract tests. See Fig-2.1 for a visual representation of the project goals (i.e. *i)* Testable Domain Models and *ii)* Derived Tests) and their relationship to the project work products (e.g. Feature Contracts are intended to be a testable domain model because, for example, dynamic tests can be derived from them in a systematic way).

2.2 Project Domain Scope

This project will require the implementation of Hartigan's Leader Algorithm and variants, in the American National Standards Institute (ANSI) C Standard, which has been adopted as an international standard (ISO/IEC 9899:1990) [16]. Even though this course is a course in object oriented software engineering, the focus is on the *essence* of object orientation (as revealed during one lecture by Professor Corriveau when questioned by the author), and so the implementation language of ANSI C was deemed to be appropriate.

The aforementioned implementation work, and the building of testable models lie within the scope of this project. However, it is recognized that the implementation work will not be for credit towards the course objectives, but, none-the-less, it is considered as a possibly useful contribution for the ongoing research in the Integrated Reasoning Group of the Institute for Information Technology of the National Research Council Canada.

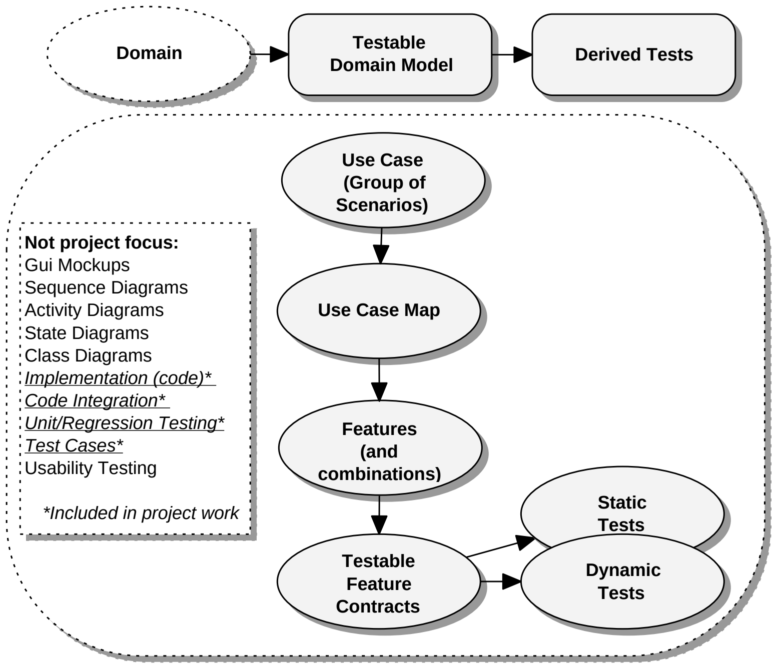

Figure 2.1: Relationship of Project Goals to Project Work Products

In addition, it is recognized that there are certainly other clustering algorithms that could be considered. They are, however, variants of a more general, and much larger family, which would involve too much time (than the duration of the course) to investigate and document thoroughly. For example, other clustering algorithms include: *i)* kmeans, *ii)* SOM, *iii)* TaxMap, *iv)* Hierarchical divisive, *v)* Hierarchical agglomerative, etc. of which one possible taxonomy (concept hierarchy variant) of these methods is given in ([14] p.201).

Only a small subset of the Leader Algorithm Family will be considered. In particular, *i)* parallel or distributed [2] variants of the leader algorithm will not be considered, as no currently known implementation exists that could be modelled, and *ii)* at most two distance (no similarity) functions will be implemented, as the investigation of an appropriate function is left to the scientific investigation of the particular data set and its associated application domain properties. That is, the data may come from very many possible domains, but only a simulated data set domain will be considered, as an algorithm variant works the same on any given data set domain; so, considering data domain variants is also outside the scope of this project. In particular, data sets could be *i)* biological, genetic (cDNA microarray data [20]), or proteomic (mass spectrometry data), or *ii)* in another domain entirely, such as in astronomy for grouping similar galaxies [17]. Therefore, deriving all possible tests from the domain model for a family member, would require knowing about the data set used, which would require knowing something about the domain. This is completely outside the scope of this project, and so a data set that the author is familiar with will be used. That is, an artificially constructed data set will be used for conformance-directed and fault-directed testing.

Nonfunctional features could be considered in terms of, for example, performance, because the different family members (variants) will potentially have different computational complexity. However, nonfunctional features will not be considered in this project.

2.3 Project Constraints and Coverage

For this project, time was a large constraint, because implementation and then modelling work needed to be performed. Indeed, a large body of material was covered in the course from many sources, of which [3] and [6] are the main references; while no material was directly used from the presented works of [11] and [9]. For example, [8] covers material related to *i)* Objects and UML, *ii)* basic concepts of real-time systems and safety-critical systems, *iii)* rapid object-oriented process for embedded systems, *iv)* requirements analysis for real-time systems, *v)* structural and behavioral object analysis, *vi)* architectural, mechanistic and detailed design, *vii)* advanced real-time object modelling including threads and schedulability, dynamic modelling and real-time frameworks.

In addition, the classical papers ([7] and [12]) were obtained and read; in order to understand the original motivations for structured programming and programming families.

Template metaprogramming [6] was not used, because, *i)* it is a C++ specific programming language construct and this project has the constraint that it should be ANSI C, *ii)* if it was used, it may be very difficult to debug the code if it turned out that testing the generated code revealed problems, and *iii)* the code should not be dependant on a particular compiler, that is, the code should be able to compile under Red Hat/GNU Linux, MS Windows or other OS.

Chapter 3

Domain

Clustering (the grouping of similar objects ([10] p.1)) could be considered part of Exploratory Data Analysis (more recently known as the Knowledge Discovery and Data Mining Process), where the clusters are considered to be the knowledge (concepts) that are being discovered or revealed. The data to be clustered is partitioned by the clustering algorithm yielding a family of clusters (concepts), where each object lies in just one member of the partition. Other definitions of a cluster may be given. For example, one object may belong to multiple clusters (have multiple –partially or wholly– related concepts associated with it), as in the field of Soft-Computing. However, for the purposes of this report, only one specific family of partition clustering algorithms (single concept per object) is considered; namely, the Leader Algorithm Family.

It is suggested in the field of Cognitive Science, in particular ([13] p.3), that

> ...the various views of concepts can be partly understood in terms of two fundamental questions: *(1)* Is there a single or unitary description for all members of the class? and *(2)* Are the properties specified in a unitary description true of all members of the class? The classical view says yes to both questions; the probabilistic view says yes to the first but no to the second; and the exemplar view says no the first question, thereby making the second one irrelevant.

Definition 2 (Domain) *([6] p.34) states that a domain is an area of knowledge*

- *Scoped to maximize the satisfaction of the requirements of its stakeholders*
- *Includes a set of concepts and terminology understood by practitioners in that area*
- *Includes the knowledge of how to build software systems (or parts of software systems) in that area*

Definition 3 (Knowledge Discovery and Data Mining) *The nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. (Fayyad, Piatesky-Shapiro, Smith 1996)*

Definition 4 (Data Mining) *Data Mining is the analysis of huge data sets. ([1] p.219)*

Definition 5 (Soft Computing) *A set of computational disciplines which in contradistinction with classical techniques, make emphasis in tolerating imprecision, uncertainty and working with partial truth notions (Zadeh 1994)*

Definition 6 (Object) *The classical software engineering ([6] p.9) definition for objects: an object has an identity, state, and behaviour. (This is a reference to Grady Booch). In particular, something related to an object oriented programming language construct.*
But in machine learning a class (or concept/category/cluster) means the class to which an object (or case/data row/instance) belongs.

3.1 Family Members

The algorithm variants in the Leader Algorithm Family are listed in Table-3.1, of which only the first variant will be described with pseudo-code. The other variants will be described in terms of the additional features they offer for the researcher (user of the algorithm) in order to focus on the differences and not replicate the similarities. A table of mappings between members and features was created (Table-4.4) but tables for feature-feature and feature-parameter relationships were not, as that is the intention of the feature contracts section §4.4.1.

№	Name	Brief Description	See Also
A_1	Hartigan's	Translation of algorithm in [10]	§3.1.1
A_2	Extension 1	Sort and then run A_1	Table-4.4
A_3	Extension 2	A_1 and search reverse	Table-4.4
A_4	Extension 3	Sort and then run A_3	Table-4.4
A_5	Extension 4	A_1 and search best	Table-4.4
A_6	Extension 5	Sort and then run A_5	Table-4.4
A_7	Extension 6	Integrate A_3 and A_5	Table-4.4
A_8	Extension 7	Sort and then run A_7	Table-4.4
A_9	Extension 8	A_1 and supervised	Table-4.4
A_{10}	Extension 9	Sort and then run A_9	Table-4.4
A_{11}	Extension 10	Integrate A_7 and A_9	Table-4.4
A_{12}	Extension 11	Sort and then run A_{11}	Table-4.4

Table 3.1: *Leader Algorithm Family Members.*

3.1.1 Hartigan's Leader Algorithm

The Leader Algorithm, as originally described ([10] p.74), begins with some motivation for this particular quick partition algorithm:

> It is desired to construct a partition of a set of M cases, a division of the cases into a number of disjoint sets or clusters. It is assumed that a rule for computing the distance D between any pair of objects, and a threshold T are given. The algorithm constructs a partition of the cases (a number of clusters of cases) and a leading case for each cluster, such that every case in a cluster is within a distance T of the leading case. The threshold T is thus a measure of the diameter of each cluster. The clusters are numbered $1, 2, 3, ..., K$. Case I lies in cluster $P(I)[1 \leqslant P(I) \leqslant K]$. The leading case associated with cluster J is denoted by $L(J)$. The algorithm makes one pass through the cases, assigning

each case to the first cluster whose leader is close enough and making a new cluster, and a new leader, for cases that are not close to any existing leaders.

The algorithmic description of the algorithm ([10] p.75) is:

STEP 1. Begin with case $I = 1$. Let the number of clusters be $K = 1$, classify the first case into the first cluster, $P(1) = 1$, and define $L(1) = 1$ to be the leading case of the first cluster.

STEP 2. Increase I by 1. If $I > M$, stop. If $I \leq M$, begin working with the cluster $J = 1$.

STEP 3. If $D(I, J) > T$, go to Step 4. If $D(I, J) \leq T$, case I is assigned to cluster J, $P(I) = J$). Return to Step 2.

STEP 4. Increase J to $J + 1$. If $J \leq K$, return to Step 3. If $J > K$, a new cluster is created, with K increased by 1. Set $P(I) = K$, $L(K) = I$, and return to Step 2. □

However, for the purposes of *i)* clarity, and *ii)* demonstration of description variability (i.e. a description of the same algorithm can be variable), a translation of the Leader Algorithm using modern terminology was performed (e.g. removing **goto** statements). In addition, the original algorithm used the terminology $D(i, j)^1$, but it is believed that $D(i, L(j))^2$, may be more clear, and so the translation described in A_1 uses this change of terminology.

Algorithm 1 Hartigan's Leader Algorithm (Translation)

Input: Data X, number of cases M, distance threshold T_d

Algorithm Negative Properties [20]: *i)* the first data object always defines a cluster and therefore, appears as a leader *ii)* the partition formed is not invariant under a permutation of the data objects *iii)* the algorithm is biased, as the first clusters tend to be larger than the later ones since they get first chance at "absorbing" each object

```
1:  $k \Leftarrow 1$                                   ▷ The current number of clusters
2:  $P(1) \Leftarrow 1$                                ▷ Classify the first case into the first cluster
3:  $L(1) \Leftarrow 1$                                ▷ Define the leading case of the first cluster
4:  for $i \Leftarrow 2$ to $i \leq M$ by $i \Leftarrow i + 1$ do   ▷ For every case but the first in the data set
5:    $P(i) \Leftarrow -1$                             ▷ Case $i$ is not assigned to a cluster yet
6:    for $j \Leftarrow 1$ to $j \leq k$ by $j \Leftarrow j + 1$ do  ▷ For each currently known cluster
7:      if $D(i, L(j)) \leq T_d$ then                 ▷ Current case is within the threshold
8:        $P(i) \Leftarrow j$                          ▷ Case $i$ is assigned to cluster $j$
9:        break for
10:     end if
11:   end for
12:   if $P(i) = -1$ then                ▷ Case $i$ isn't close enough to one of the existing leaders
13:     $k \Leftarrow k + 1$                           ▷ Create a new cluster
14:     $P(i) \Leftarrow k$                            ▷ Classify case $i$ to the new cluster
15:     $L(k) \Leftarrow i$                            ▷ Define the leader of the new cluster
16:   end if
17: end for
```

[1]meaning that case i is measured in terms of distance to cluster j

[2]meaning that case i and case $L(j)$ are measured in terms of distance to each other, where $L(j)$ is the leader for cluster j

Chapter 4

System Family Engineering Domain Models

The selected domain may be modelled (abstracted) in very many different ways. For example, if domain terminology, notation and theory exist then they may be modelled directly in their native textual or visual form. But for software engineering purposes, particular types of models are built with the view to implementing and testing software on a computer. But the software that is being built is usually written in one or more languages, which themselves are abstractions of the underlying assembly code, which is related to machine code, which may be executed in some manner by a processor (e.g. multi-tasking or parallel execution of the instructions) or by a set of processors with, for example, shared-distributed memory. In spite of all of these issues, the system family engineering domain models that will be built are being built with the intention of deriving tests from the models directly.

Definition 7 (Domain Engineering) *([6] p.20-21) Domain Engineering is the activity of collecting, organizing, and storing past experience in building systems or parts of systems in a particular domain in the form of reusable assets (i.e., reusable work products), as well as providing an adequate means for reusing these assets (i.e., retrieval, qualification, dissemination, adaptation, assembly, and so on) when building new systems. It encompasses Domain Analysis, Domain Design, and Domain Implementation. Conventional software engineering (or single-system engineering) concentrates on satisfying the requirements for a single system, whereas Domain Engineering concentrates on providing reusable solutions for a family of systems.*

Definition 8 (Product Line vs. System Family) *From ([6] p.31) The terms product line and system family are closely related, but have different meanings. A system family denotes a set of systems sharing enough common properties to be built from a common set of assets. A product line is a set of systems scoped to satisfy a given market. A product line need not be a system family, although that is how its greatest benefits can be achieved. Likewise, a system family need not constitute a product line if the member systems differ too much in terms of market target, that is, a system family could serve as a basis for several product lines. Historically, the term product line is a younger term than system family. A domain encapsulates the knowledge needed to build the systems of a system family or product line. More precisely, system families and product lines constitute two different scoping strategies for domains.*

4.1 Textual Domain Representation

All family members (See Table-3.1), when applied to the original data set X, must:

- reduce X by selecting a set of representative objects[1], as typified in Fig-4.1. It could also be thought of that the "leader" is reducing the variability in the data. That is, trying to get to the core of the data set, rather than all of the minor perturbations.
- satisfy a mathematical property, as exemplified in (4.1).
- contend with the problem of incompleteness (missing data).
- contend with the problem of data type heterogeneity. For an example of using heterogeneous time series data, see [18].

$$\{(\forall j \in [1, |X|])(\exists i \in [1, |L|])s(\vec{l_i}, \vec{x_j}) \geq T_s\} = L(X, T_s) \subseteq X \tag{4.1}$$

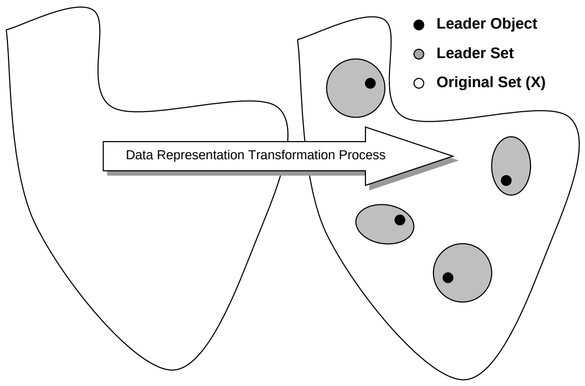

Figure 4.1: Conceptualization of Clustering Process

4.2 Visual Domain Models

4.2.1 Use Case

The way in which the Leader Algorithm System will be used is shown in Fig-4.2. In particular, one motivation of the system is to reduce the size of the data set when given to another system, and yet retain the core structure of the original data.

The focus of the project is testable feature contracts and their associated derived tests, so this use case was not elaborated further.

Definition 9 (Use Case) *http://www.foruse.com/articles/structurestyle2.pdf states:*
Jacobsons original definition [Jacobson et al.,1992]: A use case is a specific way of using the system by using some part of the functionality. [A use case] constitutes a complete course of interaction that takes place between an actor and the system.

[1]cases and objects are used interchangeably

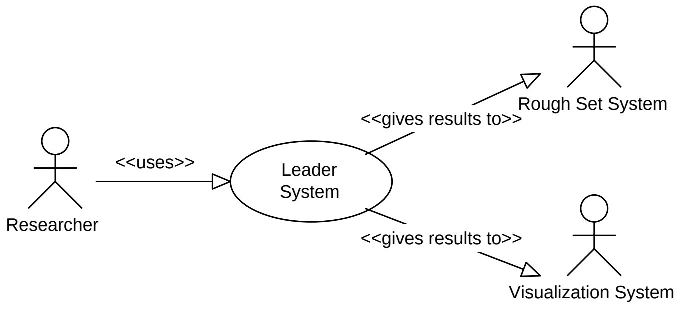

Figure 4.2: *Use Case* (UC) for Leader Algorithm Family

The specification of sequences of actions, including variant sequences and error sequences, that a system, subsystem, or class can perform by interacting with outside actors [Rumbaugh et al, 1999: 488].

[Fowler, 1997: 43]: A use case is a typical interaction between a user and a computer system [that] captures some user-visible function [and] achieves a discrete goal for the user.

Use cases ... are, of course, only one of many potential ways of modelling tasks, ranging from, on the one end of the spectrum, rigorous and highly structured approaches that are of greatest interest to researchers and academics, to, on the other end, informal movie-style storyboards and free-form scenarios.

From `www.alike.com/html/main_html/glossary.html`*: A methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal. It consists of a group of elements (for example, classes and interfaces) that can be used together in a way that will have an effect larger than the sum of the separate elements combined. The use case should contain all system activities that have significance to the users.*

4.2.2 Use Case Maps

The use case map in Fig-4.3 represents the responsibilities within the Leader Algorithm System that was implemented. The UCM also maps the responsibilities onto abstract conceptual components. For example, in Fig-4.3 the *Object Presentation Order* responsibility is mapped onto an abstract component called *Preprocessing*, which in turn may be mapped onto a concrete component or components in a particular family member[2].

The mapping from abstract components to concrete components is described in §5, along with other details of the project implementation work.

Definition 10 (Use Case Map - UCM) *http://www.usecasemaps.org states: Use case maps can help you describe and understand emergent behaviour of complex and dynamic systems.*

[2]The mapping from abstract component to concrete component may or may not be done in different ways for different family members.

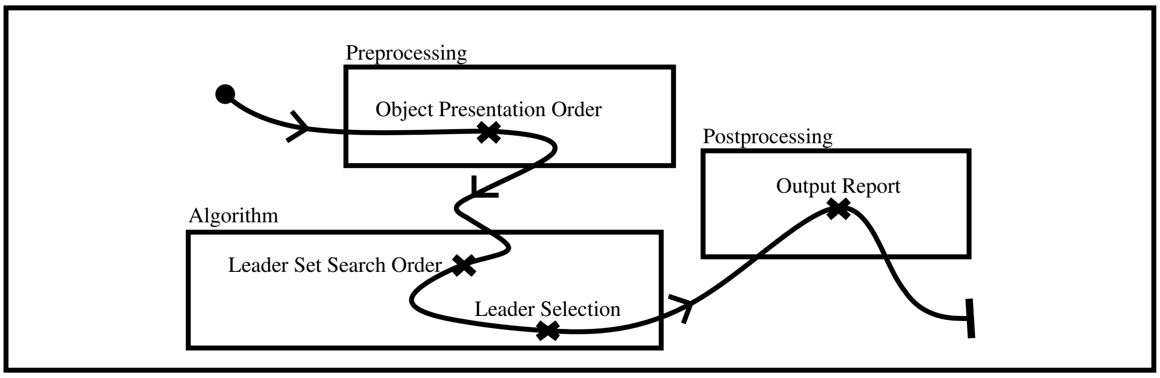

Figure 4.3: *Use Case Map* (UCM) for Leader Algorithm Family

4.3 Textual Domain Model

Domain models may be represented visually or textually. Visual domain representations, as in the previous section, take advantage of natural human visual understanding, and so may possibly be favoured over textual representations in certain circumstances. However, textual representations may be favoured due to their ability to describe structured languages — natural [e.g. Inuktitut, which is the major language of the Circumpolar region stretching from Alaska to Greenland and the main language for Nunavut[3]], artificial [Tengwar, which was invented by J.R.R. Tolkien], or computer [PROLOG, for PROgramming in LOGic].

Definition 11 (Testable Feature Contracts) *Testable feature contracts are a textual feature model of a domain, from which tests (hopefully) can be directly derived. Feature contracts address feature coupling through explicit recording of feature interactions and behavioral dependencies between features.* *Summarized from course*

4.3.1 Features and Parameters

How were the features in Table-4.1 and parameters in Table-4.2 of this domain chosen? Based on a perceived notion of what was directly related to the leader algorithm family, implying that the features may not completely cover the domain being modelled. How could the features be proved to *cover* the domain? No known answer exists, unless, possibly, if the domain is formally defined via, for example, axioms. What would happen if we have a domain, but were missing a feature? It depends on what feature was missing. If enough other information existed in the model, then the missing feature could be derived from the other information, then possibly a tool could be built to help in this regard. However, in the general case, the missing feature will become apparent to someone with appropriate domain knowledge and then the correction will be made. Such latent changes incur potentially high cost. One advantage of feature contracts could be to try and find such missing features via the structured derivation of tests, forcing the model builder to more thoroughly consider the domain and ask the "right" questions of the "right" experts. What if we had a feature that wasn't really a part of the domain? This should be found as long as the domain was

[3]Information obtained from the National Research Council Canada's Interactive Information Group's web site `http://iit-iti.nrc-cnrc.gc.ca/projects-projets/uqausiit_e.html`

clearly understood and bounded, or the domain could be enlarged or shrunken to include or exclude that feature.

№	Feature [Value] Name	Additional Information
F_1	Object Presentation Order	
F_1V_1	`DataSetOrder`	Original File Order
F_1V_2	`ObjectCompletenessOrder`	
F_2	Leader Set Search Order	
F_2V_1	`Forward`	$i \leftarrow 1$ upto $i = N$ by 1
F_2V_2	`Reverse`	$i \leftarrow N$ downto $i = 1$ by 1
F_3	Leader Selection (for i-th Object)	
F_3V_1	`First`	Select First Encountered
F_3V_2	`Best`	Select Closest Distance Encountered
F_3V_3	`FirstMostComplete`	
F_3V_4	`BestMostComplete`	
F_4	j-th Leader Set's Members	
F_4V_1	`MayBeHeterogeneous`	Unsupervised Clustering
F_4V_2	`MustBeHomogeneous`	Supervised Clustering
F_5	Leader (Prototype) Reporting	
F_5V_1	`Original`	As determined by algorithm
F_5V_2	`ClassMajority`	Ties broken randomly
F_6	Output	
F_6V_1	`HumanReadable`	
F_6V_2	`MachineParsable`	

Table 4.1: *Feature* and *Feature Value* Identifiers. F_1V_2 means feature value 2 of feature 1. Or put another way, the 2^{nd} possible variant of the feature.

Definition 12 (Parameter) *From the Ring Election thesis p47: a parameter defines a meaningful attribute input to the system. Contrary to a feature, we view a parameter not as a functional unit, but as a datum (e.g. length or weight). A feature may have parameters associated with it. These parameters may fully or only partially define a feature.*

The possible binary interactions are listed in Table-4.3 and the mapping of features onto family members is list in Table-4.4.

№	Parameter [Value] Name	Additional Information
P_1	ThresholdType	See §5.1
P_1V_1	`DistanceThreshold`	T_d (used for `DistanceFunction`)
P_1V_2	`DissimilarityThreshold`	(used for `DissimilarityFunction`)
P_1V_3	`SimilarityThreshold`	T_s (used for `SimilarityFunction`)
P_2	FunctionType	See §5.1
P_2V_1	`DistanceFunction`	
P_2V_2	`DissimilarityFunction`	
P_2V_3	`SimilarityFunction`	
P_3	SortType	
P_3V_1	`StableSort`	Input order preserving sort
P_3V_2	`UnstableSort`	Ties may be ordered in any way
P_4	DataUsage	
P_4V_1	`OneCopyOfData`	Only one copy of the data exists
P_4V_2	`MultipleCopiesOfData`	There may be multiple copies of the data
P_5	MemoryUsage	
P_5V_1	`PrimaryMemory`	All data is loaded
P_5V_2	`ExternalMemory`	Not all data loaded
P_6	DistributionType	
P_6V_1	`UniformRandom`	$f(x, A, B) = \frac{1}{B-A}$
P_6V_2	`Gaussian`	$f(x, \mu, \sigma) = \frac{e^{-(x-\mu)^2/(2\sigma^2)}}{\sigma\sqrt{2\pi}}$ (Also called Normal)
P_6V_3	`Poisson`	$p(x, \lambda) = \frac{e^{-\lambda}\lambda^x}{x!}$ for $x = 0, 1, 2, \cdots$
P_7	SoftwareSystem	
P_7V_1	`VisualizationSystem`	See [19]
P_7V_2	`RoughSetSystem`	See [19]
P_8	Computer	
P_8V_1	`Sequential`	
P_8V_2	`Distributed`	See [2]
P_8V_3	`Parallel`	See [2]

Table 4.2: *Parameter* and *Parameter Value* Identifiers. P_1V_2 means parameter value 2 of parameter 1. Or put another way, the 2^{nd} possible variant of the parameter.

№	Interaction Description
$\checkmark_0$	applicable (e.g. not applicable ✗$_0$)
$\checkmark_1$	is equivalent to
$\checkmark_2$	requires
$\checkmark_3$	has
$\checkmark_4$	must be before
$\checkmark_5$	is the parent of
$\checkmark_6$	with

Table 4.3: Possible binary relations (interactions) that may occur. Not all are used in modelling this domain, the table certainly does not contain all possible relations, and no higher order (e.g. tertiary etc.) relations are proposed. The negation of the relation may be used ✗$_1$. In general $\Re(F_1, F_2V_2)$ means that F_1 is related to F_2V_2 in the manner specified by the traceability code $\Re$. In particular, ✗$_3$ would mean *does not have* and $\checkmark_{4,5}$ would mean the conjunction of two interactions; in this case, *must be before* –and– *is the parent of*.

№	$A_1\rightarrow$	$A_2\rightarrow$	$A_3\rightarrow$	$A_4\rightarrow$	$A_5\rightarrow$	$A_6\rightarrow$	$A_7\rightarrow$	$A_8\rightarrow$	$A_9\rightarrow$	$A_{10}\rightarrow$	$A_{11}\rightarrow$	$A_{12}\rightsquigarrow$
F_1												
F_1V_1	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$
F_1V_2		$\boxed{\checkmark_3}$		$\boxed{\checkmark_3}$		$\boxed{\checkmark_3}$		$\boxed{\checkmark_3}$		$\boxed{\checkmark_3}$		$\boxed{\checkmark_3}$
F_2												
F_2V_1	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$
F_2V_2			$\boxed{\checkmark_3}$	$\checkmark_3$			$\boxed{\checkmark_3}$	$\checkmark_3$			$\boxed{\checkmark_3}$	$\checkmark_3$
F_3												
F_3V_1	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$
F_3V_2					$\boxed{\checkmark_3}$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$			$\boxed{\checkmark_3}$	$\checkmark_3$
F_3V_3												
F_3V_4												
F_4												
F_4V_1	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$
F_4V_2									$\boxed{\checkmark_3}$	$\checkmark_3$	$\checkmark_3$	$\checkmark_3$
F_5	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$
F_5V_1												
F_5V_2												
F_6	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$	✗$_0$
F_6V_1												
F_6V_2												

Table 4.4: *Member–Feature* Relationships. For example, $\checkmark_3(A_1,F_1V_1)$ means algorithm A_1 has the feature variant F_1V_1. The table has been designed based on ideas from John Tukey 1977. (For example, the most important data and relationships should clearly stand out in a visualization.)

4.4 Changes to Feature Contracts

Changes (semantic/syntactic/presentation) were made to the feature contracts proposed in the course in order to simplify the applicability for the Leader Algorithm Family domain.

C1 Overall, tried to make section names like attributes of a feature that can be referenced. For example, a feature contract could be thought of like an interface.

C2 Removed superfluous textual strings (e.g. Interaction and Behaviour) because the meaning of a section (such as VARIANTS) implies that it is an interaction or a behaviour. This assumes that the reader understands what a section means.

C3 In VARIATION section, removed $< type >\in$. Old idea was F_1.type and it has been replaced by the ability to refer to a variant of a feature by using VARIANT. For example, the current variant of Feature 1, may be referred to as F_1.VARIANT from another feature such as F_2 and as VARIANT from within F_1. The additional notation of THIS.VARIANT is not needed because the feature that the reader is currently reading is always assumed to be THIS. Consequently, the syntax is simplified, and VARIANT is thought of like an attribute of a feature that is instantiated with a value. Further, consistency is introduced because we don't have many different aliases for what boils down to referring to a variant. This requires less cognitive stress on the reader as fewer things need to be explicitly remembered.

C4 Removed CONTRACT section, as it is superfluous because this information can be determined within the MAPPING section (by a tool, if needed).

C5 Renamed MAPPING section to INTERACTIONS section in order to make it more clear that other features interact with this feature in the mentioned ways. CONSTRAINTS is another possible word, but was rejected due to the fact that data interactions may occur. For example, see F_3.INTERACTIONS section.

C6 Each section is thought of as being an ordered operation (like the production rules in an expert system). For example, the accumulation $+=$ operation in F_5.INTERACTIONS

C7 Added a DESCRIPTION section to describe the feature in natural language.

C8 Moved old DESCRIPTION section to be contained within the COMPOSITION section.

C9 Added details to RESPONSIBILITIES section.

C10 Feature and parameter names must be unique. Avoids parameter-feature confusion.

C11 Composition can be conditional. See, for example, F_1.

C12 Added ability to specify a set of responsibilities via set notation in conjunction with a template for a responsibility name. See, for example, F_3.DOCUMENTATION for `leader_algX_Y()`.

C13 Overall, tried to remove obfuscating syntax, and allow broader understanding of the intention of a contract without having a reader remember the meaning of particular symbols. For example, the syntax of PASCAL versus C.

C14 Removed use of NULL. Instead used NONE. This is to move more towards a domain, and away from a computer language, because it is trivial for a computer scientist to understand NULL, but not necessarily for a domain expert. This, of course, assumes that domain experts may read these feature contracts.

C15 Changed PARAMETERS notation to be consistent with the set notation used.

C16 Renamed VARIATION to VARIANTS.

C17 Added separating lines around the identifier and name of the feature so that the feature can be seen easily in a document.

C18 Lined names of items up, and separated them from content of items. This allows one to scan quickly for a particular item, such as VARIANTS.

4.4.1 Feature Contracts

One of the course goals is to investigate how to document both feature interactions[4] and behavioral dependencies[5]. For example, Feature–Feature, Feature–Parameter and other configuration rules (combination rules) are described within the feature contracts.

When writing a particular feature contract, the INTERACTIONS section was interpreted as being all of those things in other features that constrain, or are used by, this feature. In particular, the converse way of interpreting this section was not used. That is, the INTERACTIONS section does not document those things in this feature that are used by other features. For example, when constructing F_i, it may be constrained to a particular variant, say VARIANT_k because F_j has a variant within a set of possibilities.

The following feature contracts are a model for the project implementation work on the Leader Algorithm Family. In addition, some generalization was performed in order to conceive of possible variants that were not implemented.

F_1: **Object Presentation Order**	
DESCRIPTION:	*The objects may be presented to the main body of the leader algorithm in many different ways. For example, they may be presented in the order that they are read from a file, or they may be sorted using any total ordering relation (which may be comprised of a partial order in conjunction with, for example, the data set order) over the objects.*
VARIANTS:	{`DataSetOrder, ObjectCompletenessOrder`}
INTERACTIONS:	`None`

PARAMETERS:	PARAMETERS={`SortType,DataUsage,MemoryUsage`}
	VARIANT ∈ {`ObjectCompletenessOrder`} ⇒
	    PARAMETERS={`StableSort,OneCopyOfData,PrimaryMemory`}
	VARIANT ∈ {`DataSetOrder`} ⇒
	    PARAMETERS=`None`

DOCUMENTATION:	`sortMatrixf()` create a new, reordered data matrix `[Matrixf]` original matrix of floating point numbers (See Fig-5.1) `[MatrixfOut]` sorted version of the input matrix (Share same data, just different pointers) `[IndexPointers]` sorted pointers to data objects (See Fig-5.4) `[IndexMap]` identity map or mapping from index of `[IndexPointers]` to index of original data objects `[Bins]` bin i contains i missing values (See Fig-5.3)

COMPOSITION:	COMPOSITION={`[Matrixf],[IndexMap]`}
	VARIANT ∉ {`DataSetOrder`} ⇒
	    COMPOSITION+={`[MatrixfOut],[IndexPointers],[Bins]`}

RESPONSIBILITIES:	`sortMatrixf() {`
	    `create [MatrixfOut], [IndexPointers], [Bins]`
	`}`

[4]Feature interactions pertain to how the selection of a variant for a feature constrains the selection of a variant for another feature. [From course]

[5]Behavioral dependencies pertain to how a service offered by a feature depends on a service of another feature. [From course]

F_2: Leader Set Search Order

DESCRIPTION:	*The currently grown set of leaders (there are k of them) may be searched in many different ways when object i's membership to a leader needs to be determined.*
VARIANTS:	{`Forward`,`Reverse`}
INTERACTIONS:	F_3.`VARIANT` $\in$ {`Best`,`BestMostComplete`} $\Rightarrow$ `VARIANT`={`Forward`} F_3.`VARIANT` $\in$ {`BestMostComplete`} $\Rightarrow$ `INTERACTIONS`={F_1.`[Bins]`}
PARAMETERS:	`None`
DOCUMENTATION:	`[start]` first leader index to consider `[stop]` last leader index to consider `comparisonFcn()` tests if there are more leaders to consider `iterationFcn()` increment or decrement an iteration variable
COMPOSITION:	`COMPOSITION`={`[start], [stop]`}

RESPONSIBILITIES:	`comparisonFcn(){`
	 `}`
	 `iterationFcn(){`
	 `}`

F_3: Leader Selection (for i-th Object)

DESCRIPTION:	*There are many ways to chose a leader to represent an object. The case of choosing multiple leaders for an object is not considered, therefore, exactly one leader will be chosen for object i. For all variants, if no leader in the current leader set exists that is close enough to object i, then object i will be promoted to becoming a leader, and hence selected as the leader for object i.*
VARIANTS:	{`First`,`Best`,`FirstMostComplete`,`BestMostComplete`[6]}

INTERACTIONS:	INTERACTIONS = {F_1.`[IndexMap]`}
	VARIANT ∈ {`BestMostComplete`} ⇒
	    INTERACTIONS += {F_1.`[Bins]`}
	F_4.VARIANT ∈ {`MustBeHomogeneous`} ⇒
	    INTERACTIONS += {F_1.`[IndexMap]`.`[ClassInfo]`}

PARAMETERS:	{`DistanceThreshold`,`DistanceFunction`}
DOCUMENTATION:	`[k]` cardinality of the leader set at any given point in time `[P]` object to leader mapping. ($P[i] = j$, case i is assigned to cluster j) has size exactly equal to the number of objects `[L]` the leader set. ($L[j] = i$, cluster j is represented by object i) has size exactly equal to `[k]` with elements being objects from the input data set The `[IndexPointers]` are (polymorphically) represented as a `[Matrixf]`. This frees the implementation of this feature from having to know about the existence of F_1, and therefore reduces feature coupling. `leader_algX_Y()` where $\{(X,Y)\} = \{(1, hartigan), (3, ext2), (5, ext4), \cdots, (11, ext10)\}$
COMPOSITION:	COMPOSITION={`[k]`,`[P]`,`[L]`}

RESPONSIBILITIES:	`leader_algX_Y(){`
	    `See Table-4.4 for details`
	`}`

[6]This has not been implemented due to time constraints of the project. This variant considers all objects in bin j before bin $j+1$. For example, if leader L_1 has 2 missing values and another leader L_2 has no missing values, and if L_1 was "closer" than L_2 to object i, then L_2 should still be selected even though it is not the first encountered because it has fewer missing values... making it the "better" choice.

F_4: j-th Leader Set's Members

Description:	*An object's class may be used as additional information when considering adding it to the membership of a particular leader's members. In particular, the classes of all of the member objects belonging to a leader may be heterogeneous (i.e. the class information is ignored) or the object's classes must be homogeneous.*
Variants:	{`MayBeHeterogeneous`,`MustBeHomogeneous`}
Interactions:	{F_1`.[IndexMap].[ClassInfo]`}
Parameters:	`None`
Documentation:	`leader_algX_Y()` contains implementation
Composition:	Composition={}
Responsibilities:	`leader_algX_Y() where` $\{(X,Y)\} = \{(1, hartigan), (2, ext1), (3, ext2), \cdots, (12, ext11)\}$

F_5: Leader (Prototype) Reporting

Description:	*The leader representative may be reported as being the same as what was originally selected, or if the data contains class information and the object members of a leader have heterogeneous classes, then the most abundant class may be chosen as a better representative of the set.*
Variants:	{`Original, ClassMajority`}
Interactions:	Interactions={F_1`.[IndexMap]`} F_4.Variant $\in$ {`MustBeHomogeneous`} $\Rightarrow$ Interactions += {F_1`.[IndexMap].[ClassInfo]`}
Parameters:	`None`
Documentation:	`utility()` contains implementation
Composition:	Composition={}
Responsibilities:	`utility() {` `}`

F_6: **Output**	
DESCRIPTION:	*The output from the Leader Algorithm System may be intended to be read by a human or parsed by another computer program with the purpose of further analyzing the original input data set.*
VARIANTS:	{`HumanReadable, MachineParsable`}
INTERACTIONS:	INTERACTIONS={F_5`.utility()`}
PARAMETERS:	`None`
DOCUMENTATION:	`createHumanReadableReportForASCII()` text-based report `createHumanReadableReportForLaTeX()` LaTeX-based report `createMachineReadableReportForOptions()` system options database `createMachineReadableReportForData()` leader database
COMPOSITION:	COMPOSITION={}

RESPONSIBILITIES:	`createHumanReadableReportForASCII()` {
	  `uses` F_5`.utility()`
	}
	`createHumanReadableReportForLaTeX()` {
	  `uses` F_5`.utility()`
	}
	`createMachineReadableReportForOptions()` {
	}
	`createMachineReadableReportForData()` {
	}

4.5 Derived Tests (from Feature Contracts)

Now that documenting feature interactions and behavioral dependencies has been attempted via feature contracts[7], the following tests may be derived. In particular, static tests may be derived from feature interactions, while dynamic tests may be derived from the behavioral dependencies. The intention is to systematically derive as many *kinds* of tests as possible, in such a way that the resulting set of tests are fully traceable (i.e. Each test can be justified because of such and such a reason).

Definition 13 (Algorithm) *Prof. Nussbaum (Carleton) taught in COMP3804 that:*
An algorithm is a well defined computational procedure that takes some value, or set of values as input and produces some value, or set of values, as output. An algorithm is correct if for every input sequence the algorithm: i) halts, and ii) produces a correct output. An incorrect algorithm may: i) never halt, or ii) may halt and produce an incorrect answer.
And another variant of the definition is from the field of Cognitive Science ([15] p.5):
Information processes that transform symbolic input structures into symbolic outputs can be defined in terms of syntactic structures of the inputs and outputs. Such information processes analyze the syntactic structures of inputs and build syntactically structured outputs. Such information processes are also called algorithms. (p.6) An algorithm is defined completely in terms of processes that operate on a representation. The processes do not operate on the domain being represented. They are not even defined in terms of the meaning

[7]This is one of the course goals.

of the representation, which is carried separately by the semantic mapping from the representation to the domain. An algorithm is a -formal- procedure or system, because it is defined in terms of the form of the representation rather than its meaning. It is purely a matter of manipulating patterns in the representation.

Definition 14 (Test) *([3] p.1112) states that a test could be (1) An activity in which a system or a component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component. (2) To conduct an activity as in (1). (3) A set of one or more test cases. (4) A set of one or more test procedures. (5) A set of one or more test cases and procedures [IEEE 610].*

Definition 15 (Test case) *([3] p.1113) states that a test case is a set of inputs, execution conditions, and expected results developed for a particular objective. A representation or implementation that defines a pretest state of the IUT (Implementation Under Test —in this paper called a family member—) and its environment, test inputs or conditions, and the expected result.*

4.6 Derived Tests

Tests of feature contracts were derived from the point of view a compiler error report. Only tests need to be written; tests should say what to test rather than how. No testcases need to be written, but some are included in the appendix. Examples of each kind of test are provided when applied to the Leader Algorithm feature contracts. Syntax checks are not included. For example, not closing a { or forgetting an = or ∈.

1. *Ambiguous (feature and parameter) reference for:* `Name`.
   If F_1's VARIANTS is:
   > `VARIANT`={`DataSetOrder`}
   and if F_6's PARAMETERS is:
   > `PARAMETERS`={`DataSetOrder`}
   Then the following should be reported:
   > *Ambiguous (feature and parameter) reference for:* {`DataSetOrder`}.

2. *Invalid feature variant referenced:* `Name`.
   If F_2's INTERACTIONS is:
   > F_3`.VARIANT` ∈ {`Bests,BestMostComplete`} ⇒
   Then the following should be reported:
   > *Invalid feature variant referenced:* `Bests`.
   Similar kinds of errors could also be specified for:
   > *Invalid responsibility referenced:* `Name`.
   > *Invalid data referenced:* `Name`.

3. *Reference before definition for:* `Name`.
   If F_2's INTERACTIONS is:
   > F_3`.VARIANT` ∈ {`Best,BestMostComplete`} ⇒ `VARIANT`+={`Forward`}
   Then the following should be reported:
   > *Reference before definition for:* `VARIANT`+=.
   Because it should be `VARIANT`=.

4. *Expression can never be true:* `Name`.
   If F_2's INTERACTIONS is:
   > F_3`.VARIANT` ∈ {`Best,BestMostComplete`} ⇒ `VARIANT`={`Forward`}
   and If F_3's INTERACTIONS is:
   > F_4`.VARIANT` ∈ {`MayBeHeterogeneous`} ⇒ `VARIANT`={`First`}
   > F_4`.VARIANT` ∈ {`MustBeHomogeneous`} ⇒ `VARIANT`={`FirstMostComplete`}
   Then the following should be reported:
   > *Expression can never be true:* F_3`.VARIANT` ∈ {`Best,BestMostComplete`}.

5. *No documentation exists for:* `Name`.
   A list of data and responsibilities can be made, and then the DOCUMENTATION section can be checked to ensure that all items are documented.
   Similar kinds of errors could also be specified for:
   Not used outside documentation section: `Name`.
   Responsibility not used anywhere other than definition: `Name`.
   Data not used anywhere other than definition: `Name`.
   Parameter not used anywhere other than definition: `Name`.

6. *Invalid variant combination.*
   `Leader = {`F_1, F_2, F_3, F_4, F_5, F_6`}`
   `> Leader1 = {ObjectCompletenessOrder,Reverse,Best,`
   `MayBeHeterogeneous,ClassMajority,HumanReadable}`
   This is only only combination that was found to be illegal, because `Reverse` should be `Forward`. However, this is not quite true, because it is not an error to search `Reverse` because when combined with `Best`, all of the leader set needs to be considered. This condition was only put into the feature contract so that a feature combination could be demonstrated that was not valid.

There is not enough time to discuss in depth about pre- and post- conditions for the tests; nor for domain specific tests (See §5.4). A small list will merely be mentioned:

1. cardinality of k must always equal size of the leader set `[L]`.

2. class information read from the data file must be in range < 1..`NumClasses`>, where `NumClasses` is data domain specific, but none-the-less, assumed to be contiguous.

3. in terms of vectors, the size of an `int` must always be less than or equal to the size of a vector element. This is because an `int` is written to the bytes of an element. A different implementation is possible. For example, a mapping index for an element could be made such that it would allow constant time access to a vector of elements of non-uniform size (See §6.1).

4. the test code assumes that the class information is the last column in the data matrix.
   `inputs->classIndex = size_matrixf_cols(inputs->data);`
   But it could be possible to add this information to the data matrix itself, thereby increasing flexibility for the user.

5. the count function used in the implementation of `sort_matrixf()` must return an `int` value ∈ [0..size_matrixf_cols(`[Matrixf]`)]

6. there may not be enough memory

7. a file may not be openable, or writable

The appendix has a test case suite that demonstrates some of the differences between family members in the Leader Algorithm Family.

Chapter 5

Implementation

The implementation strategy was iterative. In particular, Hartigan's algorithm was implemented first, and then additional features were added. This time dependance is illustrated in the heading of Table-4.4. The process (iterative, tightly integrated, small waterfalls, semi-automated testing) is illustrated as follows: *i)* consider one feature that was not in Hartigan's algorithm, *ii)* extend Hartigan's algorithm to incorporate this feature, *iii)* test the extension, *iv)* make a second extension by adding a preprocessing sort wrapper, *v)* test these extensions, *vi)* integrate these extensions into a larger algorithm that incorporates all features up until this point, and *vii)* then move on to the next feature.

5.1 Threshold Function

The only threshold function that is currently acceptable is a `DistanceFunction`. However, this is easily extended to support a `SimilarityFunction` by the addition of a parameter that specifies the comparison relation to use with respect to the threshold. As the focus of the project is not on the implementation of different `DistanceFunction`'s, but on the implementation of the algorithm, it is left to the `DistanceFunction` implementation to transform itself into a `SimilarityFunction`. For example, one possible similarity function that could be used is a non-linear transformation of a modified Euclidean distance ($s = 1/(1+d)$, where s is a similarity and d a distance), accepting missing values (See [19], but originally from [5]). In particular, given two vectors $\overleftarrow{x} =< x_1, \cdots, x_n >, \overleftarrow{y} =< y_1, \cdots, y_n >\in \mathbb{R}^n$, defined by a set of variables (i.e. attributes) $A = \{A_1, \cdots, A_n\}$, let $A_c \subseteq A$ be the subset of attributes s.t. $x_i \neq \mathtt{X}$ and $y_i \neq \mathtt{X}$. The corresponding distance function is $d_e = (1/card(A_c)) \sum_{A_c} (x_i - y_i)^2$. This is a normalized distance and therefore, independent of the number of attributes. Consequently, *no imputation* of missing values to the data set is performed. See Table-5.1 for possible distance functions.

Name	Distance
Euclidean	$\frac{\sum_{A_c}(x_i-y_i)^2}{card(A_c)}$
Clark	$\frac{\sum_{A_c}\frac{(x_i-y_i)^2}{(x_i+y_i)^2}}{card(A_c)}$
Canberra	$\frac{\sum_{A_c}\frac{\lvert x_i-y_i\rvert}{(x_i+y_i)}}{card(A_c)}$
Others...	...

Table 5.1: Three possible modified distances ([19]p.1947)

5.2 UCM Component Mapping: Abstract → Concrete

The use case map (UCM) in Fig-4.3 contains abstract components wrapping the responsibilities. These components are implemented as concrete entities in the code. The mapping from abstract component to concrete component for each of the abstract components are listed in Table-5.2 for the Preprocessing abstract component, Table-5.3 for the Algorithm abstract component, and Table-5.4 for the Postprocessing abstract component.

matrix_io.h	matrixf **read_matrixf**(FILE *f)
matrix_sort.h	void **sort_matrixf**(matrixf m, matrixf* outM, vectori* outIndices)
leader_utils.h	void **swapDataAndSortedData**(leader_alg_in_t* inputs)

Table 5.2: Mapping of abstract preprocessing component to concrete components

leader_utils.h	void **initSearchParms**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg1_hartigan**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg2_ext1**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg3_ext2**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg4_ext3**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg5_ext4**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg6_ext5**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg7_ext6**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg8_ext7**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg9_ext8**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg10_ext9**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg11_ext10**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)
leader.h	void **leader_alg12_ext11**(leader_alg_in_t* inputs, leader_alg_out_t* outputs)

Table 5.3: Mapping of abstract algorithm component to concrete components

leader_io.h	void **write_leader_reports**(leader_alg_in_t* inputs, leader_alg_out_t* outputs, leader_report_t* rep)

Table 5.4: Mapping of abstract postprocessing component to concrete components

5.3 Design decisions

DD1 *2004-OCT-05* A discussion between the author and Dr. Valdés revolved around how sorting could be done efficiently. One possibility, which involves determining the number of missing values per data object and either appending to a *completeness* list or inserting into another list of varying *incompleteness*, is diagrammed in Fig-5.2. A second feasible implementation, which was selected because it is faster, is *range sort*, which is diagrammed in Fig-5.3.

DD2 *2004-NOV-19* When outputting the leader algorithm report, because we may have sorted the data, we need to keep track of the original case numbers. To do this, an extra attribute on a data object has been added, namely a case identification number (indexed from 1). For example, the i-th case in the input data file will result in the construction of a data object containing all of the data from that case in addition to a case identifier equal to the value i. The design tradeoff is that a little bit more memory ($sizeof(int) \times numCases$) will be used, versus a lot more computation to try to reverse map the case identifier for a particular data object in the sorted matrix. For example, one possible implementation might be via pointer equality, as the data objects don't move in memory. However, this would require looking through the whole array multiple times. This design decision has ramifications because the additional data object's case identification attribute can't be used by *i)* a similarity or dissimilarity function, and *ii)* a data object completeness determination function.

DD3 *2004-NOV-24* Upon further reflection, the idea of putting an additional attribute into a case when the data is read into memory is not the best way; it is better to remove the coupling between the indices and the data. The visual representation (shown in Fig-5.4) helped clarify and crystalize this concept into a more efficient implementation that allows *i)* use of the sorted data in algorithms that only know about the original data (See, for example, `leader_alg4_ext3()`) and ii) use of the indices by the reporting methods (See `leader_io.c`). Therefore, this deeper understanding has changed the previous design decision because the justification for the additional complexity for the similarity functions or data object completeness functions is not warranted.

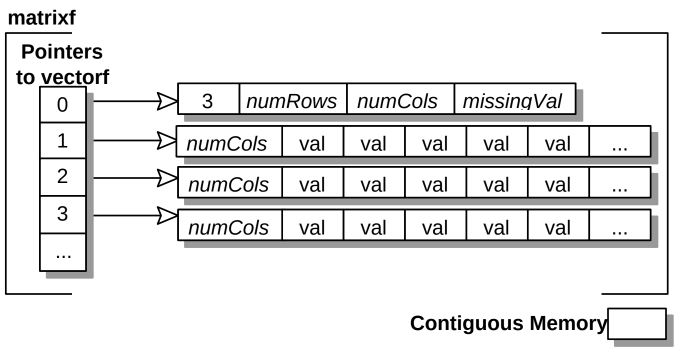

Figure 5.1: Abstract representation of a data matrix in primary memory.

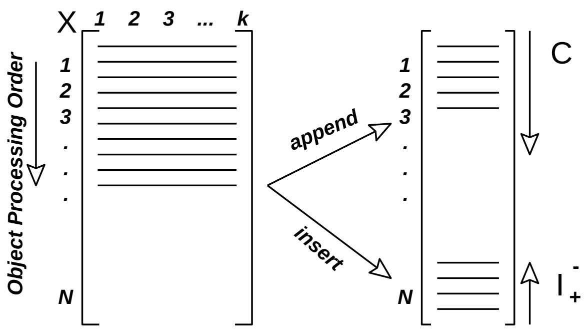

Figure 5.2: One variant of sorting incomplete data objects ($X = C \bigcup I$)

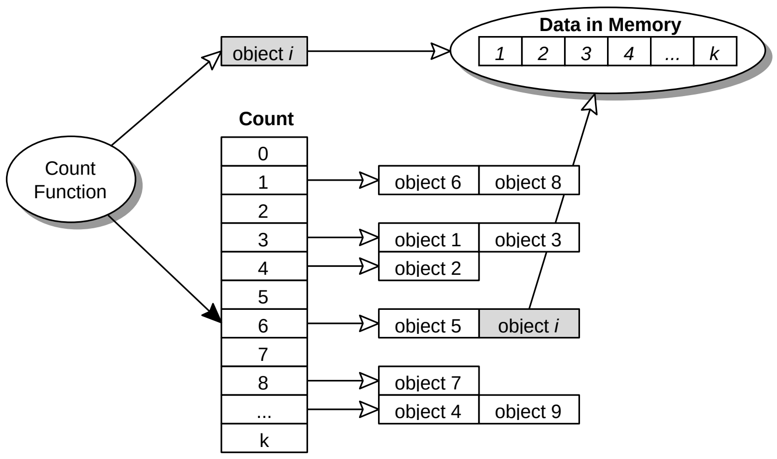

Figure 5.3: Time efficient, $O(2n)$, and space efficient, $O(sizeof(int*) \times (n + k))$, range sort of incomplete data objects using pointers.

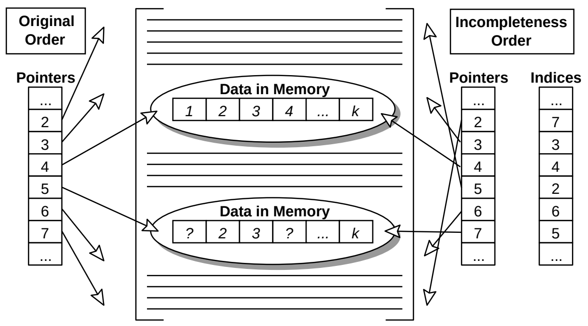

Figure 5.4: Data in primary memory indexed in two different ways: *i)* by secondary storage (data file) order, and *ii)* by incompleteness relation (count of missing values) order. Pointers are needed so that a `matrixf` can be constructed and given to a leader algorithm for either type of data. For example, if the original data is sorted before the algorithm is executed then a mapping between the original order and the sorted order is needed. The `indices` are how such a mapping is recorded.

5.4 Dynamic *"Code-based"* Unit Tests

These tests were written -during- the iterative construction of the code, and so were not explicitly planned, but were an integrated, dynamic part of a robust (and creative) development process employed in a style similar to the x-treme manifesto, whereby small unit tests are written before the code. This is in contradistinction to tests derived from features, hence the separation from those more formally (feature model based) derived tests. In other words, these tests are directly based on the code, and are related very specifically to code development activity rather than to the much higher level of the domain modelling based testing activity.

UT1 *2004-NOV-07* Implementing the sort function was intellectually stimulating.

UT2 *2004-NOV-12* It was interesting that during the construction of the translation of Hartigan's algorithm (See §3.1.1), line 4, which correctly reads, $i \Leftarrow 2$, was incorrectly translated to read $i \Leftarrow 1$, but this error was found during iterative testing while building the implementation, and so now the translation –may– be correct.

UT3 *2004-NOV-19* When running an algorithm that did nothing, the expectation was that no leaders should have been formed, but which was not the case, as some leaders were being reported with some of them having values of 0, but others having values of 19,216 (a completely random value), both of which should not have occurred. Both of these errors (non-zero value and more than zero leaders) were curious, and so it was proposed that `calloc` (when dynamically allocating a vector `allocate_vector`) was not initializing the memory to zeros. This was proved false when the `P` vector was investigated, as it had all zeros. Therefore, it was proposed that possibly the algorithm was implemented incorrectly for updating `k`, but this was proved false by the fact that the algorithm worked correctly on a previous test. Then the interaction between successive calls to the algorithm (passed as a function pointer argument to `runOneLeader`) revealed that `info` was not being initialized correctly. In particular, `void construct_leader_t(leader_t* info)` did not initialize `k` to zero, but this was not originally performed, because it was assumed that `k` would be initialized inside the algorithm e.b. `info->k = 1;` inside `leader_alg1_hartigan`. So, in hindsight, this test was probably flawed, in that an algorithm that does nothing should never actually be executed, but on-the-other-hand the code is now a little bit more robust.

UT4 *2004-NOV-24* It is important to use prefix notation when decrementing or incrementing a integer variable in a return statement (e.g. `return --i;`).

UT5 *2004-NOV-25* It was discovered that the number of columns was not being written to the data file correctly after implementation of **DD3**.
In particular, in the `matrix_io_runTestSuite()` the `matrix_io_test_01()` exhibited this failure. In order to fix this problem, it was noted that the code for implementing **DD2** still existed, and so it was decided that all code related to that decision should be deleted because that code may have been buggy. After the code was deleted, `matrix_io_test_01()` then had the expected output. The bug was probably a minus 1 that should not have existed, but there was no reason to spend time to try to track it down due to the change in design decision, and hence, lack of corresponding justification.

UT6 *2004-NOV-26* The output report does not contain the correct indices, which was discovered when running `leader_testSuite.c` while trying to complete the implementation of **DD3**.

For example, `L[2] |1| >3< [ 2]` is clearly incorrect, because the selected `>3<` does not appear in the leader set, which only consists of the index `2`. It was found that in `createHumanReadableReportForASCII()` the line which was previously written as `algOutputtedIndex = ((vectori)rep->leaderSets[ii])[jj];` should actually be referenced through the index map i.e. `inputs->data_sorted_map[...]`. The offending line now correctly reads `L[2] |1| >3< [ 3]`.

UT7 *2004-NOV-26* An access violation was occurring within `leader_testSuite.c`. It turns out that `destroy_leader_alg_in_t()` was implemented incorrectly. There were two problems: *i)* `deallocate_matrixf_andData(inputs->data);` was called before `deallocate_matrixf_andData(inputs->data_sorted);`, which should have been the other way around, and *ii)* when deallocating the sorted data, the call should not have been to the `_andDat()` variant, but to the `_butNotData()` variant.
These problems were found because the leader test suite tries more than one leader algorithm, and the particular algorithm that exhibited this problem was the extension that added the sorting capabilities. That is, the interaction between the sorting feature, and the leader algorithm were problematic.

UT8 *2004-NOV-26* The output from `leader_alg5_ext4()` for the `oneLeader` case was incorrect. That is, `P[1]=1` but `P[2]=2`, `P[3]=2` and `P[4]=2` and only one leader was expected. It was found that the implementation was incorrect. In particular, the line `outputs->P[i] = j;` should have been `outputs->P[i] = bestDistanceIndex;`

UT9 *2004-NOV-28* `inputs->distanceFcn(i,j,inputs)` should have been implemented as `inputs->distanceFcn(i,outputs->L[j],inputs)`. This was discovered while writing the code for `leader_alg9_ext8()` and not from a particular test (although a test can be written for this case). In particular, the conceptual mistake was discovered when `get_class_for_object_from_matrixf()` was written for the homogeneous case, mainly due to the fact that the only distance function that has been implemented up until this point, is the constant distance function (i.e. a distance function that always returns the same value). This decision was made in order to focus the testing on those things that have been explicitly implemented because it is not necessary to test something that hasn't been implemented, as that test will always fail. However, it is now the correct moment in time for these tests to be implemented.

UT10 *2004-DEC-15* When modifying the output in `leader_io.c` to be more informative, it was found that the vector `outputs->L` had an incorrect size, but that `k` was correct. This meant that memory was potentially being accessed when it should not. This was corrected in `leader_alg1_hartigan()` and the other relevant leader algorithms.

Chapter 6

Project Post-mortem

Observations of important aspects were recorded throughout the project life cycle. For example, how easy was it to model the project domain? Were any changes to feature contracts performed? Were Use Case Maps applicable? If so, how? Maybe they would be more helpful in a larger domain where communication between people would be an important issue.

👍 A feature contract can be thought of as being a structured entity, where a feature contract specific language may be constructed. This application specific[1] language could be parsed (as mentioned in the course) and tools could be created to support multiple kinds of views of feature contract relationships. For example, similar to the idea of a Logical View or Component View in Rose Real Time, a Feature Contract View could be graphed. In particular, possible views could be *i)* Feature Contract Interaction View, *ii)* Feature Contract Responsibility View, *iii)* Feature Contract Data Flow View, etc. It may also be interesting to combine the view idea with the feature contract idea and allow one to switch from one view to the other, while building the contracts. This way, relationships that are not obvious in one view, may be completely trivial in another possible view.

👍 The notation $\checkmark_3(A_1,F_1V_1)$ –meaning algorithm A_1 has the feature variant F_1V_1– and ✗$_3(A_1,F_1V_1)$ –meaning algorithm A_1 does not have the feature variant F_1V_1– is easily extendible to partial has relationships. For example, $\checkmark_3{}^{75\%}(A_1,F_1V_1)$, could mean that 75% of the time, algorithm A_1 has the feature variant F_1V_1. This could be useful, perhaps, when dealing with randomized algorithms, although it was not explicitly used in this project.

👍 Writing code was beneficial in understanding the details of the selected domain even though it was known to not be towards course credit. The reason is that the concepts in the domain were understandable, but the details were essential in properly understanding *i)* possible derivations —new extensions—, and *ii)* feature modelling both in terms of constructing feature contracts and feature combination rules.

👎 Manipulating feature contracts is tedious and error prone, and gets worse with the number of contracts created (scalability problem). It would be nice if a tool to help refactor them existed because this would allow the easy modification of those parts of a feature contract that have already been formalized (e.g. this could be considered a knowledge management problem possibly with some similar aspects as in the field of case based reasoning where cases could potentially be thought of as contracts?). Some questions to consider would be: *i)* what kinds of things can be refactored? just names of things, or actual con-

[1]Application specific meaning that it is not a general programming language, but specific to the feature contract domain

tract patterns?, and *ii)* if tests have been derived from feature contracts, then how would a refactoring of a contract(s) affect the tests? e.g. feature contracts could potentially be traced (via an imagined tool) to tests and then a change in the feature contract may yield recognizable (or at least potentially predictable?) changes that would need to be performed in the tests by the modifier (user of the tool).

👍If a large number of feature contracts exists, then determining if there are similar contracts would be useful. That is, feature contracts could be clustered together (after they have been created) in order to see if any duplicate contracts (or near duplicate) exist which could then be either *i)* merged and/or deleted, or *ii)* used to determine those family members that are very dissimilar. The latter could potentially be useful for the purpose of the discovery of conceptually similar algorithms, which could potentially lead to the focussing of dynamic testing onto those family members that are very dissimilar to each other w.r.t. some measure of dissimilarity defined over the testable feature contracts. The goal would be to leverage scarce testing resources to the fullest, when considering the family of systems as a whole. That is, the overall, rather cyclic, idea is that the testable feature contracts created that define the family of Leader Algorithms, could be given to a specific Leader Algorithm (or in general, could be given to each variant in turn, and the conceptual classes could be compared, or even more generally, to a clustering algorithm, not necessarily one of the Leader Algorithm Family Members) to be clustered, and hence the contracts, which were created to conceptually help understand the algorithm, could be analysed using the algorithm to help conceptually understand the contracts.

☞Report writing, model building, implementation (including refactoring) and test writing were not all completed in sequential order. The development process was more like a multi-tasking operating system where each task has a time slice (of varying length) and is executed in random order. The code improvement process (otherwise known as refactoring) was related to *i)* simple logical consistency (e.g. moving code blocks in order to locate them faster), and *ii)* complex restructuring for the sake of introducing new design constructs. For example, the addition of the row identifiers first required changing some code to be more general, and then the subsequent deletion and introduction of a different concept (indices) into the code.

👍The feature and parameter tables (Table-4.1 and Table-4.2), along with figures Fig-5.3 and Fig-5.4 were found to be quite useful in the construction of the code.

☞The UCMs were not found to be helpful in the construction of the code, as having the features listed was more helpful; this may be due to the small size of the project (by small, it is meant less than 100,000 lines of code... large could be on the order of millions of lines of code), and the fact that the contents of the feature contracts were completely understood without the feature contracts being explicitly written. However, when trying to communicate these ideas to another person, the UCMs were found to be more helpful.

☞The feature contracts also weren't found to be helpful in the construction of the code, as simply having the features listed was found to be more helpful (i.e. Table-4.1). In fact, the feature contracts were created after the code was implemented. This may be due to the fact that many variants were being attempted to be implemented, and so it was just too complicated to try to write the contracts out in full before the implementation. In fact, what was done, was a kind of *short feature contract* where the feature list was annotated with the parameters and feature interactions at a very coarse level (kind of like a scrap paper approach). This may have proved sufficient in this case, because the whole project was limited to one person and so communication between people was not an issue until the final report, when what was done needed to be communicated with the professor; then, was when feature contracts seemed much more useful.

☟The problem with using codes (№) is that if a number is mistyped, then the syntactic mistake may be hard to detect, unless a careful semantic review of the usage is performed. For example, Table-3.1 had an algorithm specified as running A_2, when in fact it ran A_3. However, this type of semantic error is somewhat alleviated through the use of LaTeX as the document preparation system, because of its ability for *i)* cross-referencing and *ii)* importing files (For example, the appendix test cases).

☞It is not clear when a member belongs to the family of leader algorithms or doesn't. For example, extensions of Hartigan's algorithm lead to more and more complex variants that creep away from the original intention of the algorithm (i.e. being very fast). At what point in this lineage, does a variant become a new algorithm? This question is still unanswered in this project.

☞One leader algorithm could have been implemented that wraps all other leader algorithms. That is, instead of implementing the sort functionality as a separate algorithm every time, a function pointer could be passed to this wrapper algorithm. For the modelling, this doesn't change the fact that there still exist different family members. It only means that the code can represent more variants per line of code, and so this implementation was not pursued for this project. In addition, *i)* having such a variant exist, means that runtime checking would need to be performed in contradistinction with compile time checking, and *ii)* the code is more explicit about what exists, as the algorithm can directly be called, rather than having to document (correctly) that this variant may be passed as a parameter, but not that variant.

6.1 Possible Future Work

The following items were not completed before the project deadline.

- *2004-NOV-19* Additional testing: Generate many configuration files to test all of the leader algorithms. I.e. "Test Generation" from a specification for a generator. This could be considered family testing because we are testing all family members, or it could be considered member testing in the context of a particular data domain. Or one could call it *science* if a data domain was chosen that was from a specific application domain. For example, biology (e.g. cDNA microarray data, or proteomic mass spectrometry data, etc.). This requires writing code for the parameters to be read from a configuration file, and then generating the configuration files. As the focus of the course is not on implementation, this was eliminated from consideration.

- *2004-NOV-28* Write test case for **UT9**.

- *2004-NOV-28* Write test and test case for distance function.

- *2004-DEC-15* Additional feature: Object i should form a new cluster if it is incomparable with all currently grown leaders. This new feature should be straightforward to implement, but was not performed for this course due the fact that it is oriented towards a `DistanceFunction` implementation, rather than a family member.

- *2004-DEC-15* Additional feature: The maximum number of leaders allowable could be specified for the algorithm, which would have to override the threshold value given to the algorithm. One way to implement this, is for the algorithm to stop processing if this number is reached.

- *2004-DEC-16* ADDITIONAL VARIATION: The `[Matrixf]` could be generalized to a matrix of any type of data. This generalization will be pursued outside the scope of the course.

Chapter 7

Acknowledgements

The author is grateful to Julio J. Valdés, Robert Orchard and A. Fazel Famili from the Integrated Reasoning Group, Institute for Information Technology, National Research Council of Canada. J. Valdés helped in the understanding of the domain and R. Orchard and A. Famili gave their support for the author to attend the course. The author is also very grateful to Professor Jean-Pierre Corriveau of the School of Computer Science at Carleton University who taught the course and to the Canada Institute for Scientific and Technical Information (CISTI), from which the author was able to obtain the necessary research materials.

Bibliography

[1] Alan Agresti. *Categorical Data Analysis.* Number ISBN 0-471-36093-7 in Wiley Series in Probability and Statistics. Wiley-Interscience John Wiley and Sons Inc., Hoboken, New Jersey, USA, second edition, 2002.

[2] A.J. Barton. Understanding properties of i/o between compute nodes in a parallel system. Internal Report ERB 1102, NRC 45820, Institute for Information Technology, National Research Council Canada, 1200 Montreal Rd, Ottawa, Ontario, Canada, K1A 0R6, April 2003. Also submitted to Prof. Frank Dehne for COMP 5704.

[3] Robert V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools.* Number ISBN 0-201-80938-9 in The Addison-Wesley object technology series. Addison Wesley Longman, Inc., One Jacob Way, Reading Massachusetts, 01867, second printing edition, Dec 1999. *(COMP 5104 Course book).*

[4] T.C. Chamberlin. The method of multiple working hypotheses. *Science (old series)*, XV(866):92–97, February 7 1890.

[5] J.L. Chandon and S. Pinson. *Analyse Typologique: Théorie et Applications.* Masson, Paris, 1981.

[6] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications.* Number ISBN 0-210-30977-7. Addison-Wesley, One Lake Street, Upper Saddle River, NJ, 07458, May 2000. *(COMP 5104 Course book).*

[7] E. W. Dijkstra. Structured programming. In J. N. Buxton and B. Randell, editors, *Software Engineering Techniques: Report on a conference sponsored by the NATO Science Committee*, Rome, Italy, October 1969.

[8] Bruce Powel Douglass. *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns.* Number ISBN 0-201-49837-5 in The Addison-Wesley object technology series. Addison Wesley Longman, Inc., One Jacob Way, Reading, Massachusetts, 01867, May 1999. *(COMP 5104 Course book).*

[9] Hassan Gomaa. *Designing Concurrent, Distributed, and Real-Time Applications with UML.* The Addison-Wesley object technology series. Addison-Wesley, One Lake Street, Upper Saddle River, NJ, 07458, Jul 2000. *(COMP 5104 Course book).*

[10] John A. Hartigan. *Clustering Algorithms.* Wiley Series in probability and mathematical statistics. John Wiley and Sons, Inc., 1975.

[11] Jean-Marc Jézéquel, Michel Train, and Christine Mingins. *Design Patterns and Contracts.* Number ISBN 0-201-30959-9. Addison Wesley Longman, Inc., One Jacob Way, Reading, Massachusetts, 01867, Oct 1999. *(COMP 5104 Course book).*

[12] D. Parnas. On the design and development of program families. In *IEEE Transactions on Software Engineering*, volume SE-2, pages 1–9, 1976.

[13] Edward E. Smith and Douglas L. Medin. *Categories and Concepts.* Number ISBN 0-674-10275-4 in Cognitive Science Series, 4. Harvard University Press, Cambridge, Massachusetts, USA, 1981.

[14] Peter H. A. Sneath and Robert R. Sokal. *Numerical Taxonomy: The Principles and Practice of Numerical Classification.* Number ISBN 0-7167-0697-0 in A Series of Books in Biology. W. H. Freeman and Company, San Francisco, USA, 1973.

[15] N. A. Stillings, S. E. Weisler, C. H. Chase, M. H. Feinstein, J. L. Garfield, and E. L. Rissland. *Cognitive Science: An Introduction.* Number ISBN 0-262-19353-1. The MIT Press, Cambridge, second edition, 1995.

[16] `http://www.eskimo.com/~scs/C-faq/q11.1.html`. What is the “ansi c standard?”. Dec 2004.

[17] J.J. Valdés. Visual data mining of astronomic data with virtual reality spaces: Understanding the underlying structure of large data sets. Astronomical Data Analysis Software and Systems XIV, 2005. In press.

[18] J.J. Valdés and A.J. Barton. Mining multivariate heterogeneous time series models with computational intelligence techniques. In *Proceedings of the Third IASTED International Conference on Artificial Intelligence and Applications*, number ISBN: 0-88986-390-3, ISSN: 1482-7913, pages 240–245, Benalmádena, Spain, September 2003. ACTA Press, Anaheim, USA.

[19] J.J. Valdés and A.J. Barton. Multivariate time series model discovery with similarity-based neuro-fuzzy networks and genetic algorithms. In *IEEE, INNS, IJCNN 2003 International Joint Conference on Neural Networks*, number IEEE Catalog Number: 03CH37464C, ISBN: 0-7803-7899-7, pages 1945–1950, Portland, Oregon, July 2003.

[20] J.J. Valdés and A.J. Barton. Gene discovery in leukemia revisited: A computational intelligence perspective. In Bob Orchard; Chunsheng Yang; Moonis Ali, editor, *Proceedings of the 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, number LNAI 3029 in Lecture Notes in Artificial Intelligence, pages pp118–127, Ottawa, Ontario, May 2004. Springer-Verlag. Subseries of Lecture Notes in Computer Science.

Chapter 8

Appendix

The following input data:

```
4 3 -9999.99
1.1 1.2 1
2.1 -9999.99 1
3.1 3.2 2
4.1 4.2 2
```

was used for the following test cases, by calling `leader_runTestSuite()`, which in turn generated LaTeX formatted output that could be directly included in this project report; in order to report the results as accurately, and efficiently, as possible.

8.1 Test case 1

A Create only one leader set.

Report for output/05-lea-alg01-oneLeader

```
Leaders(T=1.500000)  k=1  |L|=1  min{|L[i]|}=4  max{|L[i]|}=4
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |4| >1< [ 1 2 3 4]

--- For per object analysis
P[1]=1
P[2]=1
P[3]=1
P[4]=1
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg01-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
```

```
---
L[1] |1| >1< [ 1]
L[2] |1| >2< [ 2]
L[3] |1| >3< [ 3]
L[4] |1| >4< [ 4]

--- For per object analysis
P[1]=1
P[2]=2
P[3]=3
P[4]=4
```

8.2 Test case 2

A Create only one leader set.

Report for output/05-lea-alg02-oneLeader

```
Leaders(T=1.500000)  k=1  |L|=1  min{|L[i]|}=4  max{|L[i]|}=4
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |4| >1< [ 1 3 4 2]

--- For per object analysis
P[1]=1
P[3]=1
P[4]=1
P[2]=1
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg02-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |1| >1< [ 1]
L[2] |1| >3< [ 3]
L[3] |1| >4< [ 4]
L[4] |1| >2< [ 2]

--- For per object analysis
P[1]=1
P[3]=2
P[4]=3
P[2]=4
```

8.3 Test case 3

A Create only one leader set.

Report for output/05-lea-alg03-oneLeader

```
Leaders(T=1.500000)  k=1  |L|=1  min{|L[i]|}=4  max{|L[i]|}=4
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |4| >1< [ 1 2 3 4]

--- For per object analysis
P[1]=1
P[2]=1
P[3]=1
P[4]=1
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg03-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |1| >1< [ 1]
L[2] |1| >2< [ 2]
L[3] |1| >3< [ 3]
L[4] |1| >4< [ 4]

--- For per object analysis
P[1]=1
P[2]=2
P[3]=3
P[4]=4
```

8.4 Test case 4

A Create only one leader set.

Report for output/05-lea-alg04-oneLeader

```
Leaders(T=1.500000)  k=1  |L|=1  min{|L[i]|}=4  max{|L[i]|}=4
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |4| >1< [ 1 3 4 2]

--- For per object analysis
```

```
P[1]=1
P[3]=1
P[4]=1
P[2]=1
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg04-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |1| >1< [ 1]
L[2] |1| >3< [ 3]
L[3] |1| >4< [ 4]
L[4] |1| >2< [ 2]

--- For per object analysis
P[1]=1
P[3]=2
P[4]=3
P[2]=4
```

8.5 Test case 5

A Create only one leader set.

Report for output/05-lea-alg05-oneLeader

```
Leaders(T=1.500000)  k=1  |L|=1  min{|L[i]|}=4  max{|L[i]|}=4
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |4| >1< [ 1 2 3 4]

--- For per object analysis
P[1]=1
P[2]=1
P[3]=1
P[4]=1
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg05-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
```

```
L[1] |1| >1< [ 1]
L[2] |1| >2< [ 2]
L[3] |1| >3< [ 3]
L[4] |1| >4< [ 4]

--- For per object analysis
P[1]=1
P[2]=2
P[3]=3
P[4]=4
```

8.6 Test case 6

A Create only one leader set.

Report for output/05-lea-alg06-oneLeader

```
Leaders(T=1.500000)  k=1  |L|=1  min{|L[i]|}=4  max{|L[i]|}=4
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |4| >1< [ 1 3 4 2]

--- For per object analysis
P[1]=1
P[3]=1
P[4]=1
P[2]=1
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg06-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |1| >1< [ 1]
L[2] |1| >3< [ 3]
L[3] |1| >4< [ 4]
L[4] |1| >2< [ 2]

--- For per object analysis
P[1]=1
P[3]=2
P[4]=3
P[2]=4
```

8.7 Test case 7

A Create only one leader set.

Report for output/05-lea-alg07-oneLeader

```
Leaders(T=1.500000)  k=1  |L|=1  min{|L[i]|}=4  max{|L[i]|}=4
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |4| >1< [ 1 2 3 4]

--- For per object analysis
P[1]=1
P[2]=1
P[3]=1
P[4]=1
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg07-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |1| >1< [ 1]
L[2] |1| >2< [ 2]
L[3] |1| >3< [ 3]
L[4] |1| >4< [ 4]

--- For per object analysis
P[1]=1
P[2]=2
P[3]=3
P[4]=4
```

8.8 Test case 8

A Create only one leader set.

Report for output/05-lea-alg08-oneLeader

```
Leaders(T=1.500000)  k=1  |L|=1  min{|L[i]|}=4  max{|L[i]|}=4
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |4| >1< [ 1 3 4 2]

--- For per object analysis
```

```
P[1]=1
P[3]=1
P[4]=1
P[2]=1
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg08-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |1| >1< [ 1]
L[2] |1| >3< [ 3]
L[3] |1| >4< [ 4]
L[4] |1| >2< [ 2]

--- For per object analysis
P[1]=1
P[3]=2
P[4]=3
P[2]=4
```

8.9 Test case 9

A Create only one leader set.

Report for output/05-lea-alg09-oneLeader

```
Leaders(T=1.500000)  k=2  |L|=2  min{|L[i]|}=2  max{|L[i]|}=2
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |2| >1< [ 1 2]
L[2] |2| >3< [ 3 4]

--- For per object analysis
P[1]=1
P[2]=1
P[3]=2
P[4]=2
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg09-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
```

```
---
L[1] |1| >1< [ 1]
L[2] |1| >2< [ 2]
L[3] |1| >3< [ 3]
L[4] |1| >4< [ 4]

--- For per object analysis
P[1]=1
P[2]=2
P[3]=3
P[4]=4
```

8.10 Test case 10

A Create only one leader set.

Report for output/05-lea-alg10-oneLeader

```
Leaders(T=1.500000)  k=2  |L|=2  min{|L[i]|}=2  max{|L[i]|}=2
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |2| >1< [ 1 2]
L[2] |2| >3< [ 3 4]

--- For per object analysis
P[1]=1
P[3]=2
P[4]=2
P[2]=1
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg10-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |1| >1< [ 1]
L[2] |1| >3< [ 3]
L[3] |1| >4< [ 4]
L[4] |1| >2< [ 2]

--- For per object analysis
P[1]=1
P[3]=2
P[4]=3
P[2]=4
```

8.11 Test case 11

A Create only one leader set.

Report for output/05-lea-alg11-oneLeader

```
Leaders(T=1.500000)  k=2  |L|=2  min{|L[i]|}=2  max{|L[i]|}=2
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |2| >1< [ 1 2]
L[2] |2| >3< [ 3 4]

--- For per object analysis
P[1]=1
P[2]=1
P[3]=2
P[4]=2
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg11-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |1| >1< [ 1]
L[2] |1| >2< [ 2]
L[3] |1| >3< [ 3]
L[4] |1| >4< [ 4]

--- For per object analysis
P[1]=1
P[2]=2
P[3]=3
P[4]=4
```

8.12 Test case 12

A Create only one leader set.

Report for output/05-lea-alg12-oneLeader

```
Leaders(T=1.500000)  k=2  |L|=2  min{|L[i]|}=2  max{|L[i]|}=2
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |2| >1< [ 1 2]
L[2] |2| >3< [ 3 4]
```

```
--- For per object analysis
P[1]=1
P[3]=2
P[4]=2
P[2]=1
```

B Create leader sets s.t. all objects will be a leader.

Report for output/05-lea-alg12-nLeaders

```
Leaders(T=1.400000)  k=4  |L|=4  min{|L[i]|}=1  max{|L[i]|}=1
   L[i] |cardinality| >j< [set]
      i-th leader set is represented by data object j
---
L[1] |1| >1< [ 1]
L[2] |1| >3< [ 3]
L[3] |1| >4< [ 4]
L[4] |1| >2< [ 2]

--- For per object analysis
P[1]=1
P[3]=2
P[4]=3
P[2]=4
```

Index