# ISOTONIC REGRESSION THROUGH THE MERGE AND CHOP ALGORITHM FOR APPLICATION IN STATISTICAL INFERENCE

A Thesis by

Jacob Thomas Skala

Bachelor of Applied Mathematics, Sterling College, May 2014

Submitted to the Department of Mathematics, Statistics, and Physics
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

May 2016

**ISOTONIC REGRESSION THROUGH THE MERGE AND CHOP ALGORITHM FOR APPLICATION IN STATISTICAL INFERENCE**

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Mathematics.

---

Xiaomi Hu, Committee Chair

---

Chunsheng Ma, Committee Member

---

Kirk Lancaster, Committee Member

---

Susan Castro, Committee Member

## DEDICATION

To my dad, my mom, my sisters, and my beloved friends

## ACKNOWLEDGEMENTS

I would like to thank my advisor, Xiaomi Hu, for supporting me and helping me with all my inane questions. I would also like to thank all of my professors and mathematics teachers for years of instruction and guidance, Conner Sorensen for his aid in developing my programming proficiency, and Ken Troyer for challenging me to achieve.

## ABSTRACT

In this paper, the theory for the application of the Merge and Chop Algorithm are defined and proven. The algorithm is used to find isotonic regressions in more situations than comparable methods. A program is included with this paper for computing the desired isotonic regressions through the algorithm. These isotonic regressions can then be used to perform more accurate and powerful hypothesis tests. This paper also defines these possible areas of utilization with use of restricted maximum likelihood estimators and likelihood ratio tests.

## TABLE OF CONTENTS

**Chapter** **Page**

## TABLE OF CONTENTS (continued)

**Chapter** **Page**

## LIST OF FIGURES

**Figure** **Page**

## LIST OF ABBREVIATIONS

| | |
|---|---|
| LRT | Likelihood Ratio Test |
| MCA | Merge and Chop Algorithm |
| MLE | Maximum Likelihood Estimate |
| MVA | Minimum Violator Algorithm |
| PAVA | Pool Adjacent Violators Algorithm |
| RWD | Robertson, Wright, and Dykstra |
| UpwardMC | Upward Merge or Chop |
| DownwardMC | Downward Merge or Chop |
| $CSS$ | Corrected Sum of Squares |
| $SSE$ | Sum of Squares of Error |

## LIST OF SYMBOLS

| | |
|---|---|
| $x$ | An input vector |
| $x^*$ | Isotonic Regression of $x$ |
| $P_C(x)$ | Projection of $x$ onto $C$ |
| $f(x_1, ..., x_p)$ | Multiple argument function |
| $(i, j)$ | Order restriction from $i$ to $j$ |
| $\preceq$ | Quasi order |
| $\Omega$ | Domain of order restrictions |
| $A = \{(i, j), ...\}$ | Set of order restrictions |
| $R^p$ | Real valued space with $p$ dimensions |
| $C$ | Closed, convex cone |
| $V$ | Hilbert Space |
| $w_i$ | Weight of $i$-th value |
| $U$ | Upper Set |
| $L$ | Lower Set |
| $\mathcal{B}$ | Level sets of input |
| $B_i$ | $i$-th level set |
| $b_i$ | Value of $i$-th level set |
| $\mathcal{T}$ | Level sets of isotonic regression |

## LIST OF SYMBOLS (continued)

| | |
|---|---|
| $Av(x, D)$ | Average value of $x$ on $D$ |
| $f^*\|_D$ | Isotonic regression of $f$ restricted to $D$ |
| $M$ | Base order matrix |
| $m_{ij}$ | Entry at $i$-th row and $j$-th column of matrix $M$ |
| $\mu$ | Population mean |
| $\bar{x}$ | Sample mean |
| $\bar{\bar{x}}$ | Weighted average of $x$ |
| $\sigma^2$ | Population variance |
| $L(\mu, \sigma^2)$ | Likelihood function of $\mu$ and $\sigma^2$ |
| $N(\mu, \sigma^2)$ | Normal variable with mean $\mu$ and variance $\sigma^2$ |
| $H_0$ | Null hypothesis |
| $H_1$ | Order restricted hypothesis |
| $H_a$ | Alternative hypothesis |
| $\mathcal{L}$ | Null hypothesis domain |
| $\Lambda$ | Likelihood ratio |
| $T^2$ | LRT statistic related to normal variables under order restrictions |
| $T^2_{ob}$ | Observed test statistic value |
| $T^2_*$ | Test statistic with variance unknown |

## LIST OF SYMBOLS (continued)

| | |
|---|---|
| $\chi^2$ | Chi-squared distribution |
| $\mathcal{B}(\alpha_i, \beta_i)$ | Random variable with beta distribution |
| $p_i$ | $i$-th level probability |
| $\bar{\chi^2}$ | Chi-bar squared distribution |
| $\bar{E^2}$ | $E$-bar squared distribution |

# CHAPTER 1

## Introduction

When making statistical inferences, known information regarding populations can be used to assume a restraint upon the studied sample statistics. These restrictions can be mathematically described as orderings. Examining samples under an ordering can lead to a more accurate conclusion than studying unrestrained samples. For example, Potthoff and Roy (1964) conducted a study on the size of the pituitary fissure of young girls aged 8, 10, 12, and 14. In the study, it was feasible to assume that the size of the pituitary fissure increased as the age of the subjects increased. If the results of the traditional method for finding sample statistics did not follow the order restriction, then a regression of the sample statistics could be used to estimate the population parameters. There are many methods for obtaining this regression in order restricted settings. This paper aims to study one of the methods known as the Merge and Chop Algorithm.

The Merge and Chop Algorithm (MCA) is a method for producing isotonic regressions in an order restricted space. The produced isotonic regression is a projection of the original sample onto the order restricted space. This projection can be utilized to find a more accurate estimation of the overall population than the traditional sample statistics. This algorithm allows for the use of a larger variety of orderings than similar algorithms such as the Pool-Adjacent-Violator Algorithm (PAVA) introduced by Ayer et al. (1955), or the Minimum-Violator Algorithm (MVA) by Thompson (1962). For instance, PAVA only works for simple orderings, while the MCA can work for simple orderings, umbrella orderings, simple tree orderings, and more.

This thesis covers the theory behind the MCA in chapter 2, the process of the algorithm in chapter 3, a program that automates the algorithm in chapter 4, the applications of such a program in chapter 5, and concludes in chapter 6.

# CHAPTER 2

## Theory

Using order restrictions for statistical inference first requires an understanding of the framework for isotonic regression. To understand these settings for using the Merge and Chop Algorithm, this paper defines order restrictions, order restricted spaces, isotonic regression, and average values on level sets of $x^* = P_C(x)$.

### 2.1 Order Restrictions

Let $f(x_1, ..., x_p)$ be a function with an input vector $x$ such that $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}$. Order restrictions can be implemented onto $x$ to reduce the domain of $f$. A common order restriction is $x_i \leq x_j$ for some pair $(i, j)$ where $i, j \in \Omega = \{1, ..., p\}$. A set of order restrictions, $A$, institutes a restriction, or ordering, on all of $x$. For example, let $A = \{(1, 2), (2, 3)\}$ be the ordering on $f(x_1, x_2, x_3)$. Then, $x_1 \leq x_2 \leq x_3$. This is known as a simple ordering as noted by Robertson, Wright, and Dykstra (1988), referred to as RWD from now on, in section 1.3. Another example of a simple ordering would be $A = \{(3, 2), (2, 1)\}$ where $x_3 \leq x_2 \leq x_1$. Two other types of common orderings are simple tree orderings and umbrella orderings. A simple tree ordering occurs when one single element is ordered below every other element, or vice versa. For example, $x_1 \leq x_i$ for all $i = 2, ..., p$ defines a simple tree ordering. An umbrella ordering would be similar to the form $x_1 \leq x_2 \geq x_3$. The diagrams in figure 2.1 provide visual representation of these orderings.

Orderings can be considered of Type I or Type II if they follow certain restrictions. A Type I ordering occurs if each $k \in \Omega$ has either 1 or 0 upward relations. A Type II order occurs if each $k \in \Omega$ has either 1 or 0 downward relation. Simple orderings and simple tree orderings are always of Type I or Type II. Some algorithms, such as the PAVA, only work

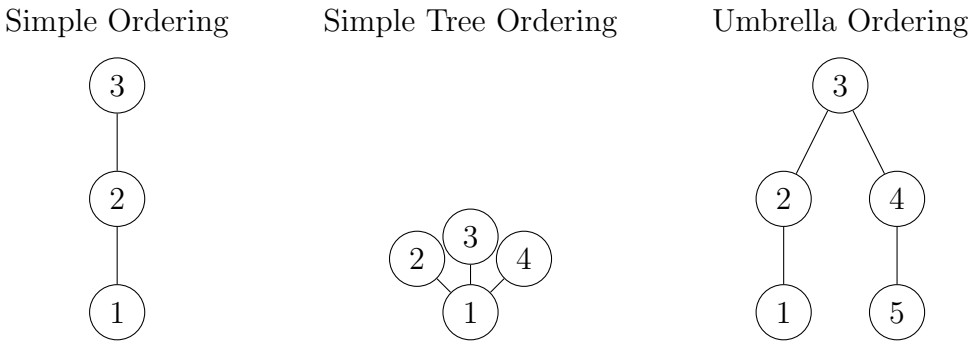


Figure 2.1: Examples of Orderings

for orderings that are Type I and Type II. The MCA can possibly work for orderings that do not fall into one of these two types. It is important to note that the MCA works for any given ordering that is Type I or Type II. For example, in figure 2.1, the umbrella ordering cannot be of Type II as the node numbered 3 has two downward relations, but since it is of Type I, the MCA can compute the isotonic regression.

To create the ordering by defining an order for each set $(i, j)$ in $A$, the relation between the values $x_i$ and $x_j$ must follow specific properties.

### 2.1.1 Quasi Orders

The weakest order is known as a quasi order, or a preorder, as shown by RWD (1988). Let $T$ be a set of elements. Define a relation $\preceq$ on $T$ such that the following statements are satisfied:

- $\preceq$ is reflexive: $x \preceq x$ for all $x \in T$.
- $\preceq$ is transitive: $x \preceq y$ and $y \preceq z$ implies $x \preceq z$.

Then, $\preceq$ is called a quasi order on $T$.

### 2.1.2 Further Orders

There are more stringent properties for other types of orders. For example, a relation $\preceq$ is known as a partial order if:

- $\preceq$ is a quasi order.
- $\preceq$ is antisymmetric: $x \preceq y$ and $y \preceq x$ implies $x = y$.

A relation $\preceq$ is a total order if:

- $\preceq$ is a partial order.
- For all $x, y \in T$, either $x \preceq y$ or $y \preceq x$.

For example, umbrella orderings may not use a total ordering. Suppose $x_1 \preceq x_2$ and $x_3 \preceq x_2$. Since no relation is explicitly present for the pair $(1, 3)$, this order may not be a total order. For real numbers, $\leq$ is always a total order. This relation is reflexive, transitive, antisymmetric, and for any two real numbers $x$ and $y$, either $x \leq y$ or $y \leq x$.

One last order is an equivalence relation. A relation $\preceq$ is an equivalence relation if:

- $\preceq$ is a quasi order.
- $\preceq$ is symmetric: if $x \preceq y$, then $y \preceq$ x.

For real numbers, $=$ is an equivalence relation as it is reflexive, transitive, and symmetric. In general, equivalence relations are denoted as $\equiv$ or $\sim$.

### 2.2 Order Restricted Spaces

Order restricted spaces exhibit many properties that are essential in finding an isotonic regression. For a function $f(x_1, ..., x_p)$ under an order restriction $A$, the space of all possible input vectors, is an order restricted space. If there are no restrictions, then $x \in R^p$. If some order restriction is present, then $x \in C \subset R^p$. In the next sub sections, three inherent properties of order restricted spaces are defined and proved, and can be verified in the publication by RWD (1988). These three properties show that a closed, convex cone is induced if an order restriction is present. The last section shows that a closed, convex cone can also induce an order relation.

### 2.2.1 Cones

The first property that an order restricted space exhibits is that it is a cone. A space $C$ is a cone if for all $x \in C$, $\alpha x \in C$ for all $\alpha \geq 0$.

Proof:

Suppose that the space $C$ has the ordering $x_i \leq x_j$ for all $(i, j)$ in $A$. Then, for all $\alpha \geq 0$, $\alpha x_i \leq \alpha x_j$ for all $(i, j)$. Thus, $\alpha x \in C$ for all $\alpha \geq 0$, and $C$ is a cone.$\square$

The use of cones for the MCA can be applied to polyhedral cones, as proven by Raubertas, Lee, and Nordheim (1986). Polyhedral cones are cones that are finitely generated, or every element in the cone is a non-negative linear combination of generators.

### 2.2.2 Convex Sets

$C$ is a convex set if for all $x, y \in C$, $\alpha x + (1 - \alpha)y \in C$ for all $\alpha \in (0, 1)$. If $C$ is an order restricted space, then it is a convex set.

Proof:

Suppose $C$ is an order restricted space such that for all $x$ and $y$ in $C$, $x_i \leq x_j$ and $y_i \leq y_j$ for all $(i, j) \in A$. Then, $\alpha x_i \leq \alpha x_j$ and $(1 - \alpha)y_i \leq (1 - \alpha)y_j$ for all $\alpha \in (0, 1)$. It follows that $\alpha x_i + (1 - \alpha)y_i \leq \alpha x_j + (1 - \alpha)y_j$ for all $\alpha \in (0, 1)$. Thus, $\alpha x + (1 - \alpha)y \in C$, and $C$ is a convex set.$\square$

### 2.2.3 Closed Set

Lastly, each order restricted space is a closed set. $C$ is a closed set if for every sequence $x^{(n)} \in C$, where $x^{(n)} \to x$, $x \in C$.

Proof:

Suppose the sequence $x^{(n)} \in C$ and $x^{(n)} \to x$. If $C$ has an order restriction $A$, and $x_k^{(n)} \to x_k$ for all $k$, then $x_i^{(n)} \leq x_j^{(n)}$ for all $(i, j) \in A$. Therefore, if $x^{(n)} \in C$, then $x \in C$, and $C$ is a closed set.$\square$

### 2.2.4 Order Relations Induced by Closed, Convex Cones

The previous three properties show that an order restriction can induce a closed, convex cone. It is important to note that a closed, convex cone can induce an order restriction. If an order $\preceq$ is present in a Hilbert space $V$, i.e. a complete linear space with a norm induced from an inner product, then the cone defines an order relation.

Proof:

Suppose $C$ is a closed, convex cone in a Hilbert Space $V$. For $x$ and $y$ in $V$, define $x \preceq y$ if $y - x \in C$. Then the following are true:

(a) Since $x \in V$, then $x - x = 0 \in C$. Thus, $x \preceq x$.

(b) If $x \preceq y$ and $y \preceq z$, then $y - x, z - y \in C$. Since $C$ is a convex set, let $\alpha = \frac{1}{2}$ and $dfrac12(z-y)+\frac{1}{2}(y-x) \in C$. By the definition of a cone, $2[dfrac12(z-y)+\frac{1}{2}(y-x) \in C] \in C = z - x \in C$. Thus, $x \preceq z$.

(c) Let $x_1 \preceq y_1$ and $x_2 \preceq y_2$. Then, $y_1 - x_1, y_2 - x_2 \in C$. This implies that $\alpha(y_1 - x_1) + \beta(y_2 - x_2) \in C$ for all $\alpha \geq 0, \beta \geq 0$. Then, $(\alpha y_1 + \beta y_2) - (\alpha x_1 + \beta x_2) \in C$, and $\alpha x_1 + \beta x_2 \preceq \alpha y_1 + \beta y_2$ for all $\alpha \geq 0, \beta \geq 0$.

(d) For $x_n \preceq y_n$, assume $x_n \to x$ and $y_n \to y$. Then, $y_n - x_n \in C$, and $y_n - x_n \to y - x \in C$. Thus, $x \preceq y$.$\square$

Note that part $(a)$ and $(b)$ define the reflexivity and transitivity of a quasi order, and $(c)$ and $(d)$ are properties of such a relation. See RWD (1988) page 15 for more on order restrictions and closed, convex cones.

## 2.3 Isotonic Regression

The isotonic regression of a function $f$ is the projection of $f$ onto a convex cone, $C$, induced by an order restriction. Commonly, these functions are paired with a weight function that details the relative weight of each value. To understand these pieces of isotonic

regression, the next subsections define isotonic functions and projections onto closed, convex cones.

### 2.3.1 Isotonic Functions

Consider a function $f(x_1, ..., x_p)$ with the quasi order $\preceq$. The function $f$ is called an isotonic function if $x_i \preceq x_j$ implies $f(x_i) \leq f(x_j)$. The word isotonic refers to a trend that is non-decreasing. These functions can be labeled monotonic, but monotonic encapsulates non-increasing functions as well, or anti-tonic functions. The collection of all isotonic functions is a closed, convex cone. When finding an isotonic regression of a function, the function is projected onto an isotonic cone induced by an order restriction as shown by RWD (1988) on page 17 of their publication. A isotonic regression can also be described as a least-squares fit for the data, where the fit is a non-decreasing function. The image below gives an example of a produced isotonic regression.

### 2.3.2 Projections onto Closed, Convex Cones

The isotonic regression of some $x$ with respect to a closed, convex cone $C$, induced by an order restriction, is the projection denoted by $x^* = P_C(x)$. To find this projection, consider $\Omega = \{x_1, ..., x_p\}$ with a relation $\preceq$ that satisfies the properties of a quasi order. Let $C$ be the convex cone in $R^p$ that is the collection of all isotonic functions on $\Omega$. RWD (1988) prove that there is a unique function $f^* \in C$ that minimizes $\parallel f - g \parallel$ for all $g \in C$. This $f^*$ is the projection of $f$ onto $C$, called the isotonic regression of $f$. In order for $f^* = P_C(f)$, then the following are necessary and sufficient conditions:

- $f^* \in C$
- $\langle f - f^*, f^* \rangle = 0$
- $\langle f - f^*, g \rangle \leq 0$ for all $g \in C$.

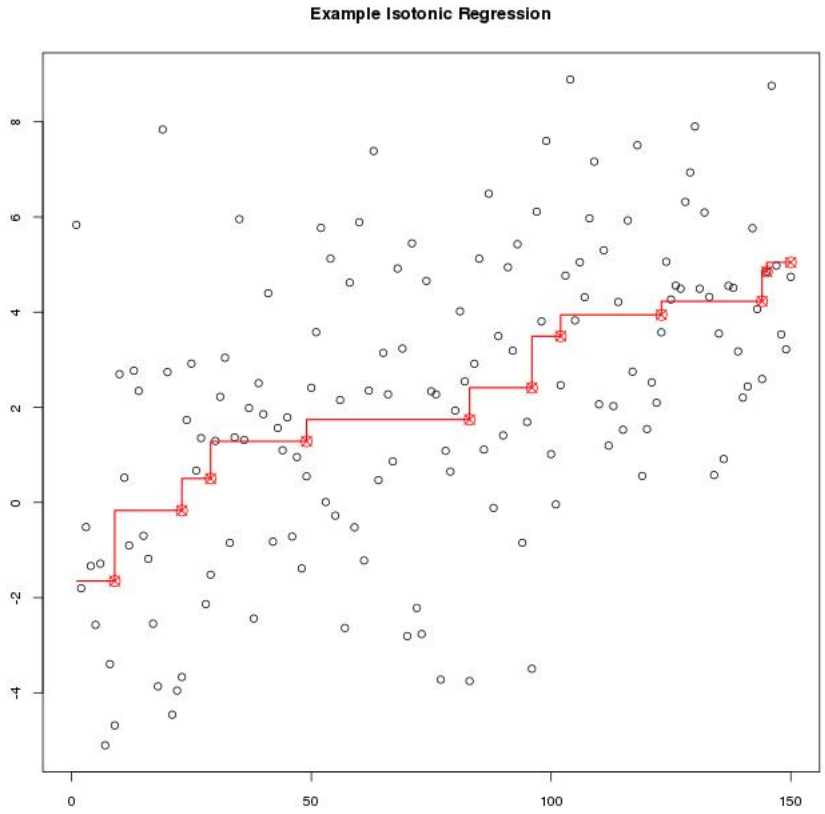


Figure 2.2: Isotonic Regression

The proof of these items can be seen in Zarantonello (1971) Lemma 1.1. These minimization and uniqueness properties are necessary as they dictate the desired projection to be the best approximation of the original input. This approximation follows the desired order restriction.

## 2.4 Average Values on Level Sets

During the process of the MCA, average values are used to correct instances where the order restriction is not satisfied. Each value of the initial vector begins in its own level set. When two values are averaged, the new value is an amalgam into a single level set. The next three sections detail the definition of level sets, and how this averaging can be used to find the desired projection.

### 2.4.1 Level Sets

Assume that $x^*$ is the projection of $x$ onto a closed, convex cone $C$ induced from an order restriction. Let $\Omega = \{1, 2, ..., p\}$ and define a quasi order $\preceq$ on $\Omega$. A subset $U$ of $\Omega$ is an upper set if for all $i \in U$, $i \preceq j$ implies $j \in U$. A subset $L$ of $\Omega$ is a lower set if for all $i \in L$, $j \preceq i$ implies $j \in L$. A subset $B$ of $\Omega$ is a level set if and only if there exists a lower set $L$ and upper set $U$ such that $B = L \cap U$.

If $x$ assumes $q$ distinct values, where $q \leq p$ then $x$ has $q$ levels. These levels are represented as $b_1 < b_2 < ... < b_q$. The level sets of $x$ are defined by $B_j = [x = b_j] = \{i \in \Omega : x_i = b_j\}$. Level sets can also be denoted by $[x \leq b_j] \cap [x \geq b_j]$ where $[x = b_j]$ is the level set for a value $b_j$. Using this notation, when $x$ is an isotonic vector, $[x \leq b_j]$ is a lower set by definition, and $[x \geq b_j]$ is an upper set. Thus, $\Omega$ is partitioned by these level sets as $B_i \cap B_j = \emptyset$ for all $i \neq j$, and $B_1 \cup ... \cup B_q = \Omega$.

### 2.4.2 Average Values on Sets

The average value of a vector or function depends on each individual value and the corresponding weight of each value. The average value of a vector $x$ on some subset $D \subset \Omega$ is given by

$$Av(x, D) = \sum_{i \in D} \frac{w_i}{\sum_{j \in D} w_j} x_i = \frac{\sum_{i \in D} w_i x_i}{\sum_{i \in D} w_i}.$$

In context of statistics, the sample size of each value divided by variance could be used as the weight of a value. This average value is also called the weighted average.

### 2.4.3 Average Value of $x$ on Level Sets of $P_C(x)$

Hu and Hanshom (2007) show in Theorem 1 of their paper that the isotonic regression of a function $f$, $f^*$, is completely determined by the collection of level sets $\mathcal{T} = \{T_1, ..., T_q\}$ of $f^*$. The level sets of the isotonic regression are imposed upon the input vector to determine the desired projection. Thus, $T = \{x : f^*(x) = t\}$ and $t = Av(T|f, w)$ where $w$ is the weight function. This theorem also shows that a given $T$ is the largest maximum valued upper set in $\{x : f^*(x) \leq t\}$, and this $T$ is the largest minimum valued lower set in $\{x : f^*(x) \geq t\}$.

The MCA aims to find these level sets that determine the isotonic regression and the values that each level set expresses.

# CHAPTER 3

## Merge and Chop Algorithm

With the background theory of the previous chapter, the Merge and Chop Algorithm (MCA) can now be defined. The MCA has two major processes, merging and chopping. Merging is the weighted average of two level sets, while chopping is the notion that the considered level set is in the desired isotonic regression. These process happen in either an upward direction or a downward direction determined by paths. The MCA is a mixed use of upward and downward processes. The algorithm works until either the isotonic regression is found, or upward and downward merge and chop simultaneously fail. Even though then MCA works for more orderings than PAVA or MVA, it can still fail to find an isotonic regression. Chapter 6 illustrates an example where MCA will fail with figure 6.2. Throughout this section, let $\Omega = \{1, .., p\}$, where $\preceq$ is an order in $\Omega$ that induces a closed, convex cone $C \subset R^n$. Assume that $x^* = P_C(x)$ is the desired isotonic regression of a vector $x$. This section covers paths, the merge process, the chop process, and an example of the algorithm finding an isotonic regression.

### 3.1 Paths

In order to determine if values should be merged or chopped, the algorithm first finds the number of paths connected to a value. Paths are concerned with the order restriction that generates the closed, convex cone onto which the input vector $x$ will be projected. A path, or base order, is a specified ordering on $i_0$ and $j_0$ such that the following are satisfied:

- $i_0 \preceq j_0$
- $i_0 \neq j_0$
- For $k \in \Omega$, the set $\{k : i_0 \preceq k \text{ and } k \preceq j_0\} = \emptyset$.

The base order $i \preceq j$ is called an upward path from $i$ to $j$, or a downward path from $j$ to $i$. In the process of upward merge and chop, only upward paths are considered, and only downward paths are needed for downward merge and chop. These paths can also be considered as paths between sets. For example, consider the sets $B_1, B_2 \subset \Omega$. If $x \in B_1$, $y \in B_2$, and $x \preceq y$, then there exists an upward path from $B_1$ to $B_2$. It is important to note that while the relation $\preceq$ must be at least a quasi order, base paths do not account for the reflexivity or transitivity of the order.

### 3.2 Merge Process

The merge process takes two level sets of the input vector, $B_1, B_2 \in \Omega$, and finds the weighted average of the sets if they are in the same level set of the isotonic regression. To prove this process, two lemmas must first be introduced that help prove to a theorem detailing the process.

**Lemma 1:** Let $B \in \Omega$ and $\preceq$ be a quasi order.

(a) If $B$ has a unique upper path from $x$ to $y$, and there exists $u \in B$ and $v \notin B$ such that $u \preceq v$, then $u \preceq x$ and $y \preceq v$.

(b) If $B$ has a unique lower path from $x$ to $y$, and there exists $u \in B$ and $v \notin B$ such that $v \preceq u$, then $x \preceq u$ and $v \preceq y$.

(c) If $B$ has no upper path and there is $x \in B$ such that $x \preceq y$, then $y \in B$.

(d) If $B$ has no lower path and there is $x \in B$ such that $y \preceq x$, then $y \in B$.

Proof:

(a) If $B$ has a unique upper path from $x$ to $y$ and for $u \in B$, $u \preceq v$ where $v \notin B$, then the relation $u \preceq v$ must pass through the path from $x$ to $y$. Thus, by transitivity, $u \preceq x$, and $u \preceq y$. It follows that if $u \preceq y$ and $u \preceq v$ where $x$ to $y$ is a unique upper path, then $y \preceq v$.

(b) The proof of this part is similar to the logic of the proof of (a).

(c) Suppose $y \notin B$. Then, if $x \preceq y$, $B$ must have an upper path, a contradiction. Therefore, if $x \preceq y$ and $B$ has no upper paths, $y \in B$.

(d) The proof of this part is similar to the logic of the proof of (c) .$\square$

Let $\mathcal{B} = \{B_1, ..., B_p\}$ be a partition of $\Omega$ and suppose that $\mathcal{T} = \{T_1, ...T_m\}$ is the collection of level sets of the isotonic regression $f^*$. The collection $\mathcal{B}$ is said to be finer than $\mathcal{T}$ if there exists a mapping $h : \{1, ..., p\} \to \{1, ..., m\}$ such that $B_i \subset T_{h(i)}$ for all $i \in \{1, ..., p\}$.

**Lemma 2:** Suppose the partition $\mathcal{B}$ is finer than $\mathcal{T}$.

(a) If $B_{i^*}$ has a unique upper path from $x^*$ to $y^* \in B_{j^*}$, then $B_{i^*} \cup T_{h(j^*)}$ is an upper set in $\{x : f^*(x) \leq t_{h(j^*)}\}$.

(b) If $B_{i^*}$ has a unique lower path from $x^*$ to $y^* \in B_{j^*}$, then $B_{i^*} \cup T_{h(j^*)}$ is a lower set in $\{x : f^*(x) \geq t_{h(j^*)}\}$.

Proof:

The proof of this lemma can be found in the paper by Hu and Hansohm (2007) by use of Lemma 1.

Using these two lemmas, the following theorem can be proven for the merge process.

**Theorem 1:** Let $\mathcal{B}$ be a finer partition of $\Omega$ than $\mathcal{T}$. Suppose one of the following statements is true.

(a) $B_i$ is a maximum valued set in $\mathcal{B}$ and it has a unique upper path from $x \in B_i$ to $y \in B_j$.

(b) $B_i$ is a minimum valued set in $\mathcal{B}$ and it has a unique lower path from $x \in B_i$ to $y \in B_j$.

Then, $B_i$ and $B_j$ are in the same level set of $f^*$. Thus, if two sets in $\mathcal{D}$ are merged together, the resulting $\mathcal{B}$ is still a finer partition of $\Omega$ than $\mathcal{T}$.

Proof:

(a) Suppose $B_j$ is in the level set $T_{h(j)}$. By part (a) of Lemma 2, $B_i \cup T_{h(j)}$ is an upper set in $\{x : f^*(x) \leq t_{h(j)}\}$. Since $T_{h(j)}$ contains some amount of sets in $\mathcal{B}$ and $B_i$ is a maximum valued set in $\mathcal{B}$, then $Av(B_i \cup T_{h(j)})|f, w)$ is no less than $Av(T_{h(j)}|f, w)$. By the theorem referenced in section 2.4.3, $T_{h(j)}$ is the largest maximum valued set in $\{x : f^*(x) \leq t_{h(j)}\}$. Thus, $B_i \cup T_{h(j)}$ is a maximum valued upper set in $\{x : f^*(x) \leq t_{h(j)}\}$, and $B_i \cup T_{h(j)} \subset T_{h(j)}$. Therefore, both $B_j$ and $B_i$ are in $T_{h(j)}$.

(b) The proof of this part is similar to the logic of the proof of (a). □

Under this theorem, the merge process can take place. If a set $B_i$ is the largest maximum valued set in $\mathcal{B}$ and it has a unique upper path to a set $B_j$, then both of these sets are in the same level set of the isotonic regression. If these two sets are in the same level set, they must have the same value, which is found by computing the weighted average of the two sets. A similar process occurs for downward merge, where $B_i$ is the largest minimum valued set in $\mathcal{B}$ with a unique lower path to a set $B_j$.

### 3.3 Chop Process

When finding the isotonic regression by merging sets together, it is imperative to locate when a set no longer needs to be merged. This occurs when a set is the same as a level set that partitions the isotonic regression. This process is known as chopping the set. The following lemma shows that after a set has been considered chopped, that the remaining set can still be used to find the rest of the isotonic regression.

**Lemma 3:** Suppose $f^*$ is the isotonic regression of a function $f$ with a weight vector $w$, where $B$ is either a maximum valued upper set or a minimum valued lower set. Define $D = \Omega - B$. Then, the following statements are true.

(a) If $x \in B$, then $f^*(x) = Av(B|f, w)$.

(b) $f^*|_D = P_{C_D}(f|_D)$.

Proof:

The proof of this lemma can be found in the paper by Hu and Hansohm (2007).

This lemma allows the algorithm to disregard any set $B$ found to be a maximum valued upper set or minimum valued lower set. In other words, this lemma allows the algorithm to chop off a set and resume the process of finding the isotonic regression on everything that remains in $\mathcal{B}$. The next theorem identifies which sets can be chopped.

**Theorem 2:** Let $\mathcal{B}$ be a finer partition of $\Omega$ than $\mathcal{T}$.

(a) If $B_i$ is a maximum valued set in $\mathcal{B}$ and it has no upper paths, then $B_i$ is a maximum valued upper set in $\Omega$.

(b) If $B_i$ is a minimum valued set in $\mathcal{B}$ and it has no lower paths, then $B_i$ is a minimum valued lower set in $\Omega$.

Proof:

(a) Suppose that $x$ and $y$ are in $\Omega$, $x \preceq y$, and $x \in B_i$. By part (c) of Lemma 1, $y \in B_i$ as $B_i$ has no upper paths. Then, $B_i$ is a maximum valued set in $\mathcal{B}$. Since $B_i$ is a maximum valued set in $\mathcal{B}$, and the largest maximum valued upper set is partitioned bu some sets on $\mathcal{B}$, then $Av(B_i|f, w)$ is the maximum value over all upper sets in $\Omega$. Therefore, $B_i$ is a maximum valued upper set in $\Omega$.

(b) The proof of this part is similar to the logic of the proof of (a). □

For example, if the algorithm is working in an upward direction and the largest maximum valued set has no upper paths, then this set is chopped. If the algorithm is working downward, then the largest minimum valued set with no lower paths can be chopped. After a set is chopped it is no longer consider during the process for the merge and chop algorithm. The algorithm continues until all sets are considered chopped.

### 3.4 Merge and Chop Algorithm Process

With the theory behind the processes of merging and chopping defined, the steps of the algorithm will now be explained. The first step of the algorithm is to identify the collection of sets, $\mathcal{B}$, that partition $\Omega$. Then, the paths between these sets can be shown. Given that merges and chops work for either maximum valued sets or minimum valued sets, the algorithm can begin in either direction. For the purpose of consistency, the upward direction will be chosen first.

When considering the initial level sets of a vector $x \in R^p$, every value can be regarded as in an individual level set. This does not change the process or outcome of the MCA. After determining the sets and paths, an upward merge or chop can be attempted. The algorithm begins by locating a maximum valued set in $\mathcal{B}$. After locating the desired set, one of three actions may be performed:

1. If the set has no upper paths, chop the set.

2. If the set has a single upward path, merge these two sets together.

3. If the set has more than one upward path, the upward merge or chop process has failed.

If an upward chop was performed on a set $B_i$, the algorithm attempts another upward merge or chop on the reduced set $\Omega - B_i$. If an upward merge was performed on $B_i$ and $B_j$ to create a new set $B_k$, the upward merge and chop is attempted again on the now coarser collection $\mathcal{B}$. If the maximum valued set is merged, then the paths from any merged set are considered, unless they are paths within the merge. If the process fails, then the algorithm moves on to attempt a downward merge or chop. The downward merge and chop process begins by locating a minimum valued set. One of three actions are then performed on this set:

1. If the set has no lower paths, chop the set.

2. If the set has a single downward path, merge these two sets together.

3. If the set has more than one downward path, the downward merge or chop process has failed.

Assuming that an upward merge and chop was performed first, the process will revert back to attempting upward merges and chops if the downward merge and chop process was successful. The upward process will be performed on the new $\mathcal{B}$, whether it is coarser from a merge, or reduced from a chop.

If both the upward and downward processes fail simultaneously, then the overall Merge and Chop Algorithm fails to produce the desired isotonic regression. If the ordering relevant to the process is of Type I or Type II, then the MCA will not fail. When all sets are considered chopped, all the sets correspond to a single level set of the isotonic regression.

For example, consider the vector $x = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$ with a corresponding weight vector $w = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$. Suppose the algorithm is being used to find the isotonic regression $x^* = P_C(x)$, where $C$ is the cone induced by the order restriction $A = \{(1,2),(2,3)\}$. The process would begin by considering each value of $x$ as an individual set $B_i$. The largest set in the collection $\mathcal{B}$ would be $B_2 = 3$. Since this set has a single upward path from $B_2$ to $B_3$, using the ordering $(2,3)$, these sets need to be merged. The weighted average would be calculated as:

$$\frac{3(1)+2(1)}{1+1} = \frac{5}{2} = 2.5.$$

Thus, the new vector would be $x = \begin{bmatrix} 1 \\ 2.5 \\ 2.5 \end{bmatrix}$ with level sets $B_1$ and $B_2$. The algorithm would repeat the upward merge and chop procedure by identifying the new set $B_2$ as the maximum valued set in $\mathcal{B}$. Since this set no longer has any upward paths, it would be chopped. The only remaining set $B_1$ would be also be chopped in the next step of the algorithm. Since

all sets are considered chopped, the procedure would be complete. Therefore, $x^* = \begin{bmatrix} 1 \\ 2.5 \\ 2.5 \end{bmatrix}$ would be the desired isotonic regression of $x$.

# CHAPTER 4

## Programming the Merge and Chop Algorithm

The author of this paper has created a program that automates the Merge and Chop Algorithm in the programming language Python. The program code has been attached in as Appendix A at the end of this thesis for examination. This section aims to discuss the programming techniques used to perform the algorithm. One useful aspect of this program is that it gives a step-by-step readout of how the algorithm created the isotonic regression. This is especially advantageous if the algorithm fails, as the program will detail all steps performed leading to the failure.

The main process of the program works by recursion. Since the program continues until all sets have been chopped, the program calls itself until all values are marked as chopped. In the case where the MCA fails to find an isotonic regression, the loop that dictates the recursion is broken.

To examine the procedure of the program, this section details the initialization of the program, the execution of the algorithm, and the output process of the program.

### 4.1 Initialization

The program works by using three input files. Before the algorithm is performed, the program first checks the input files, creates and displays an order matrix, and initializes a status matrix.

The input files are text files that give the initial vector, the corresponding weight vector, and a list of ordered pairs detailing the order restrictions. All of these files are in a text format with each value or ordered pair on a new line. The program transforms the initial vector and weight vector into single dimensional floating point arrays. The ordered pairs are used to create a single dimensional integer array. Before moving to the next step, the program checks the length of each array in the function called "first_check". If any array

has a length of zero, the program will not perform the algorithm. This is to make sure that program has successfully loaded and read the desired files. A message will displayed if the initial check is successful or failed.

If the initial check has been passed, the program will continue by creating a matrix $M$ from the given orderings in the function called "create_matrix_order". The matrix is first initialized by creating a two dimensional array of zeros of size $n$ by $n$, where $n$ is the length of the input vector. The program will examine each ordered pair given. For example, if an ordered pair is $(i, j)$, then the program will change the value located at $m_{ij}$ to 1. This 1 signifies that the desired isotonic regression follows the quasi order $i \preceq j$. After all ordered pairs have been considered, the matrix $M$ is known as the base-order matrix. It is important to note that this matrix does not indicate the reflexive or transitive orderings that the quasi order $\preceq$ follows. Since the MCA only considers base orders for locating paths, the matrix $M$ is only used to identify these base orders. To verify the successful creation of the base order matrix, the program outputs the matrix to a file called "matrix.txt" from the function "matrix_check".

After checking each input and creating the base order matrix, the initialization process creates another matrix. This matrix, $L$, denotes if any values have been merged, chopped, or untouched. The matrix is initialized as an $n$ by $n$ two dimensional array of all zeros. Each zero indicates that a value has not been considered in the algorithm. A value of 1 indicates that a set has been chopped, while a 2 marks a set as merged. For example, if the value at $l_{ii}$ is a 1, then the set $B_i$ has been chopped. If the value located at $l_{ij}$ is marked as a 2, then the sets $B_i$ and $B_j$ have been merged together. It is important to note that the algorithm does not remove sets, it only considers sets as merged. This is only for avoiding indexing issues while the program is computing the isotonic regression.

### 4.2 Execution of the Merge and Chop Algorithm

The external function in the program is called "MCA". To perform the MCA, this function calls the main function "merge_and_chop". As stated previously, the algorithm

works by means of recursion. The recursion occurs as the function "merge_and_chop" calls itself until completion or failure of the algorithm. The function creates a while loop that checks each value in the status matrix $L$. If each value along the diagonal of $L$ is marked with a 1, then every set is chopped and the function is completed. The rest of the function "merge_and_chop" can be broken down in to four main pieces: locating the desired set, upward merge and chop, downward merge and chop, and completion.

### 4.2.1 Locating the Maximum or Minimum Valued Set

To execute the algorithm, the largest valued set is located first. As previously assumed, upward merges and chops are considered before downward merges and chops for consistency. First, the chopped values are removed, and the resulting reduced vector is used to locate the maximum value. The algorithm does not discriminate between multiple sets with the maximum value. For example, if the set $B_i$ and the set $B_j$ have the maximum value $k$, then the algorithm chooses the set with the lower index. This does not change the resulting isotonic regression as Theorem 1 and Theorem 2 only rely on a maximum valued set, not a specific maximum valued set.

### 4.2.2 Upward Merge and Chop

After a maximum valued set has been located, the program moves on to the function "UpwardMC". The next step of the program depends on if a set has been merged. If the maximum valued set is a single value, the function "UpwardMC" is used. If the status matrix finds that the maximum valued set is merged with another set, the function "UpwardMCMergeProtocol" is used instead. The difference between the two functions is that the function "UpwardMCMergeProtocol" includes extra steps to handle merged sets.

Both functions begin by identifying all upward paths from the maximum valued set. The function "UpwardMCMergeProtocol" function must take extra precaution to not count paths between sets that have been merged together. If the number of paths in 0, the maximum valued set is chopped and the corresponding value in the status matrix $L$ is

changed to a 1. If the number of paths is 1, the merge process is performed for the relevant sets. In either of these cases, the variable "complete" is changed from a 0 to a 1. This signifies that the function successfully performed a merge or a chop. If complete has a value of 1, then the program will revert to attempt an upward merge or chop again.

In the case where the variable called "complete" is not changed to a 1, the upward merge and chop process has failed. This means that the maximum valued set had two or more upward paths. When this occurs, the program moves on to attempt a downward merge or chop.

### 4.2.3 Downward Merge and Chop

A downward merge or chop only occurs after the failure of an upward merge or chop. When this transpires, the flow of the program returns to the function "merge_and_chop". The program will next use the same technique to locate a minimum valued set. Analogously, the program finds the minimum valued set with the lowest index value, as Theorem 1 and Theorem 2 do not require a specific minimum valued set. This set is then passed to the function "DownwardMC". Comparable to the process of an upward merge and chop, the program determines if the minimum valued set is merged. If it is merged, it is passed to the function "DownwardMCMergeProtocol" for extra provisions to handle the merged sets. These functions determine the number of paths, and execute a merge or chop if applicable. In the case where a downward merge or chop cannot be completed, the overall merge and chop algorithm has failed. The variable "complete" is used to determine a failure. If the variable is not changed from a value of 0, the algorithm will break all loops and print out a message stating the MCA has failed to find an isotonic regression.

### 4.2.4 Completion

When the final iteration of the function "merge_and_chop" is completed, the returned vector will be the final isotonic regression. If the MCA manages to chop every set without running into an error, then one final function, "outputfile", is called. This function prints

out the isotonic regression in a text file called "output.txt". If the MCA was unable to find an isotonic regression of the initial vector, then this file will still be created. This output could be embedded into other programs or functions for application.

# CHAPTER 5

## Application

The isotonic regression found through the use of the Merge and Chop Algorithm can be used extensively in the field of Statistics. While projections onto closed, convex cones induced by order restrictions can be applicable in other fields, this paper will focus on the statistical implications of the algorithm. The two main applications for isotonic regression in Statistics are Restricted Maximum Likelihood Estimators, and Hypothesis Testing through Likelihood Ratio Tests.

### 5.1 Restricted Maximum Likelihood Estimators

Maximum likelihood estimation is a foundational method of approximating parameters of statistical models. For example, assume that $\{X_n\}$ is a sequence of independent and identically distributed normal random variables with mean $\mu$ and variance $\sigma^2$, or $\{X_n\} \sim N(\mu, \sigma^2)$. Using the likelihood function,

$$L(\mu, \sigma^2) = (2\pi\sigma^2)^{-n/2} \exp\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2\right],$$

it is well known that the maximum likelihood estimator (MLE) of $\mu$ is given by the sample mean, $\bar{x}$. Refer to proof by Johnson and Wichern (2007) by result 4.11 on page 171 for an example of deriving this MLE. Specifically, $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the MLE of $\mu$ because it maximizes the likelihood function $L(\mu)$.

When working in order restricted spaces with multiple normal populations distributed as $N(\mu_i, \sigma_i^2)$ where $i = 1, ..., p$, $\mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_p \end{bmatrix}$ is unknown, and $\sigma_i^2$ are known, this process is analogous to $\bar{x} = \begin{bmatrix} \bar{x}_1 \\ \vdots \\ \bar{x}_p \end{bmatrix}$ minimizing $\| \bar{x} - \mu \|^2$ where $\mu \in R^p$. This norm can be used instead,

as the inner product $\langle x, y\rangle = \sum_i x_i y_i \frac{n_i}{\sigma_i^2}$ for $x, y \in R^p$, $w_i = \frac{n_i}{\sigma_i^2}$ and $i = 1, ..., p$ is defined in order restricted spaces. The norm $\| \bar{x} - \mu \|^2 = \langle \bar{x} - \mu, \bar{x} - \mu \rangle = \sum_i (\bar{x}_i - \mu_i)^2 \frac{n_i}{\sigma_i^2}$ can be substituted into the likelihood function $L(\mu)$. Thus, the likelihood function, determined by multiplying all of the probability density functions together, could be expressed as

$$L(\mu) = \prod_{i=1}^{p} \prod_{j=1}^{n_i} (2\pi\sigma^2)^{-1/2} \exp\left[-(2\sigma_i^2)^{-1}(x_{ij} - \mu_i)^2\right]$$

$$= (2\pi)^{n/2} \prod_{i=1}^{p} (\sigma_i^2)^{-n_i/2} \exp\left[-\frac{1}{2}\sum_{i=1}^{p} \frac{CSS_i}{\sigma_i^2}\right] \exp\left[-\frac{1}{2}\sum_{i=1}^{p} (\bar{x}_i - \mu_i)^2 \frac{n_i}{\sigma_i^2}\right]$$

$$= (2\pi)^{n/2} \prod_{i=1}^{p} (\sigma_i^2)^{-n_i/2} \exp\left[-\frac{1}{2}\sum_{i=1}^{p} \frac{CSS_i}{\sigma_i^2}\right] \exp\left[-\frac{1}{2} \| \bar{x} - \mu \|^2\right],$$

where $CSS_i = \sum_{j=1}^{n_i} (x_{ij} - \bar{x}_i)^2$.

If there is an order restriction on $\mu$, then $\mu \in C$, where $C$ is the closed, convex cone induced by the order restriction. The estimator that maximizes the likelihood function $L(\mu)$ over $\mu \in C$ is known as the restricted maximum likelihood estimator (RMLE). The RMLE is found by minimizing $\| \bar{x} - \mu \|^2$ where $\mu \in C$. As previously stated, this is minimized by $\bar{x}$, but since $\mu \in C$, then the RMLE is given by the unique projection $\bar{x}^* = P_C(\bar{x})$. The existence and minimization condition of this projection are proven by the Hilbert Projection Theorem. A detailed proof of the uniqueness portion of this theorem can be found in the publication by Luenberger (1969) in Section 3.3 under Theorem 1. Also, Theorem 4.10 in the textbook by Rudin (1966) details a proof of the Projection Theorem with an emphasis on closed, convex sets. Therefore, $\| \bar{x} - \mu \|^2 \geq \| \bar{x} - \bar{x}^* \|^2$ for all $\mu \in C$. This RMLE can be determined by using the MCA, where the corresponding weight function is determined by sample size divided by variance, or $\frac{n_i}{\sigma_i^2}$.

Similar RMLEs can be verified for normal variances, Analysis of Variance settings, and Bernoulli, Geometric, Poisson, Multinomial, and Gamma distributions. These RMLEs are described on pages 34-39 in context of Theorem 1.52 in the publication by RWD (1988), and can be used to perform p-value tests.

## 5.2 Hypothesis Testing

RMLEs can be used for calculating the p-values of hypothesis testing. In classical hypothesis testing for normal means, the null hypothesis $H_0$ is the statement that $\mu_1 = \mu_2 = ... = \mu_p$, which can be expressed as $\mu \in \mathcal{L}$ where $\mathcal{L} = \{\mu \in R^p : \mu_1 = ... = \mu_p\}$. The classic alternative hypothesis, $H_a$, is the statement that $\mu_i \neq \mu_j$ for some $i, j = 1, ..., p$. In order restricted spaces, this alternative hypothesis would change to test $H_0$ against $H_1 - H_0$, or $H_a = H_1 - H_0$, where $H_1$ is the hypothesis that $\mu \in C$. Thus, $H_a$ tests if $\mu \in C - \mathcal{L}$. To test this hypothesis, a likelihood ratio test statistic can be derived, and the observed significance levels can then be computed.

### 5.2.1 Likelihood Ratio Test Statistic

To test $H_0$ against $H_1 - H_0$, a likelihood ratio test statistic can be derived from

$$\Lambda = \frac{\max_{\mu \in H_0} L(\mu, \sigma^2)}{\max_{\mu \in H_1} L(\mu, \sigma^2)}.$$

If the variance is known, then RWD (1988) derive the test statistic to be

$$T^2 = \sum_{i=1}^{k} w_i (\bar{x_i}^* - \bar{\bar{x}}_i)^2,$$

where $\bar{\bar{x}}_i$ is the weighted sample mean, and $\bar{x}^*$ is the projection of $\bar{x}$ onto $C$. Note that in hypothesis testing, the null hypothesis is rejected for small $\Lambda$, or large $T^2$, as $T^2$ is strictly monotonic. Also, the weighted mean is a vector of length $p$, where $\bar{\bar{x}}_1 = ... = \bar{\bar{x}}_p$. Given that the norm of $\langle x, y \rangle$ is defined to be $\sum_{i=1}^{p} x_i y_i w_i$, then

$$T^2 = \langle \bar{x}^* - \bar{\bar{x}}, \bar{x}^* - \bar{\bar{x}} \rangle = \| \bar{x}^* - \bar{\bar{x}} \|^2 .$$

If this test statistic is considered under the hypothesis spaces $\mathcal{L}$ and $C$, then

$$T^2 = \| P_C(\bar{x}) - P_{\mathcal{L}}(\bar{x}) \|^2 .$$

This test statistic follows a $\bar{\chi}^2$ distribution, as noted by RWD (1988).

A similar test statistic can also be derived in the case where the variance is unknown. This test statistic is given by

$$T_*^2 = \frac{T^2}{SSE + \| \bar{x} - P_C(\bar{x}) \|^2},$$

where $SSE = \sum_{i=1}^{p} CSS_i$, which follows an $\bar{E}^2$ distribution. The MCA is useful for calculating $P_C(\bar{x})$, when $C$ is a closed, convex cone induced by an order restriction. Once this computation is performed, the resulting test statistics can be used for finding p-values.

### 5.2.2 Observed Significance Levels

In order to give meaning to a LRT statistic, level probabilities can be used to determine levels of significance in the form of p-values. Given that the distributions for $T^2$ and $T_*^2$ are invariant under choice of $\mu \in \mathcal{L}$, assume that $\mu = 0$. Then, the formula for finding the levels of significance are given by RWD (1988) as

$$P(T^2 > t | \mu = 0) = \sum_i p_i P(\chi^2(i) > t),$$

$$P(T_*^2 > t | \mu = 0) = \sum_i p_i P(\mathcal{B}(\alpha_i, \beta_i) > t),$$

where $p_i$ is the level probability, and $\mathcal{B}$ is a beta variable. Thus, the p-values can be found by evaluating

$$P(T^2 > T_{ob}^2 | \mu = 0) = \sum_i p_i P(\chi^2(i) > T_{ob}^2),$$

$$P(T_*^2 > T_{*ob}^2 | \mu = 0) = \sum_i p_i P(\mathcal{B}(\alpha_i, \beta_i) > T_{*ob}^2),$$

where $T_{ob}^2$ and $T_{*ob}^2$ are the observed test statistic obtained from evaluating statistical samples. Given the difficult nature of evaluating the level probabilities as shown by RWD (1988) in Section 2.2, the desired p-values can instead be approximated my simulation.

### 5.2.3 Simulated Example

If $n_1, .., n_p$ and $t$ are given, Monte-Carlo simulation can be used to find $N$ sets of samples with $\mu = 0$. These $N$ sets could derive $N$ values of $T^2$ to compare against $T_{ob}^2$. The simulated p-value would then be the relative frequency of the event that $T^2$ is greater than

$T^2_{ob}$. This simulated value approximates the exact value $P(T^2 > T^2_{ob}|\mu = 0)$. This approach requires only the isotonic regression found through the MCA. Future work could attach the MCA program into a simulated p-value calculator. One specific example of using Monte-Carlo simulations for ordered restricted spaces has been shown by Lemke (1983). Lemke uses a simple loop ordering to perform hypothesis testing. This ordering is not of Type I or Type II. The following diagram portrays the simple loop ordering Lemke employed.

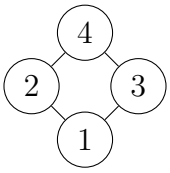


Figure 5.1: Simple Loop Ordering

This simple loop has only four nodes, but the MCA could find isotonic regressions for simple loop orderings of higher magnitudes. The values of this simulation have been arranged in the appendix of the publication by RWD (1988).

# CHAPTER 6

## Conclusion

The Merge and Chop Algorithm can be considered as a strong alternative to classic algorithms such as the Pool Adjacent Violator Algorithm for computing isotonic regressions. This strength can be identified as the ability for the MCA to use more complex orderings than PAVA. Only orderings of Type I and Type II can be used for PAVA, but the MCA can use orderings that do not fall into either one of these categories. For example, consider the following diagram of an order restriction for the input vector $x = \begin{bmatrix} 6 \\ 7 \\ 9 \\ 1 \\ 10 \\ 6 \\ 8.5 \\ 8 \end{bmatrix}$ and the order restriction $A = \{(1,3),(2,3),(3,4),(3,5),(4,6),(5,6),(6,7),(6,8)\}$. This ordering is not of

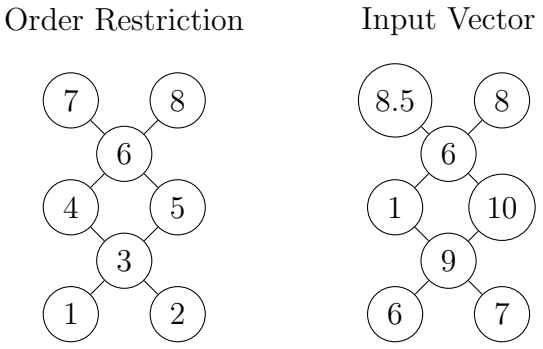


Figure 6.1: MCA Applicable Vector and Ordering

Type I or Type II as some nodes have 2 upper paths, or two lower paths. The PAVA would

not be able to compute an isotonic regression of this form, while the MCA is still applicable. The following is a detailed output given by the MCA program included with this thesis.

```
Load successful.
First check complete.
The checked vector for max is:
[  6.    7.    9.    1.   10.    6.    8.5   8. ]
Finding max. Max is 10.0 located at 4
The path count is: 1
[(4, 5)]
Merging values...
[ 6.   7.   9.   1.   8.   8.   8.5  8. ]
The checked vector for max is:
[ 6.   7.   9.   1.   8.   8.   8.5  8. ]
Finding max. Max is 9.0 located at 2
The path count is: 2
[(2, 3), (2, 4)]
Upward MC failed. Moving to Downward MC
[ 6.   7.   9.   1.   8.   8.   8.5  8. ]
The checked vector for min is:
[ 6.   7.   9.   1.   8.   8.   8.5  8. ]
Finding min. Min is 1.0 located at 3
The path count is: 1
[(2, 3)]
Merging values...
The checked vector for max is:
[ 6.   7.   5.   5.   8.   8.   8.5  8. ]
Finding max. Max is 8.5 located at 6
```

```
The path count is: 0
[]
Chopping value...
[ 6.   7.   5.   5.   8.   8.   8.5  8. ]
The checked vector for max is:
[ 6.  7.  5.  5.  8.  8.  8.]
Finding max. Max is 8.0 located at 4
Procedure is attempting upward MCA for merged values
The current merges are at: [4, 5]
The number of paths is 0
Chopping values...
[ 6.   7.   5.   5.   8.   8.   8.5  8. ]
The checked vector for max is:
[ 6.  7.  5.  5.  8.]
Finding max. Max is 8.0 located at 7
The path count is: 0
[]
Chopping value...
[ 6.   7.   5.   5.   8.   8.   8.5  8. ]
The checked vector for max is:
[ 6.  7.  5.  5.]
Finding max. Max is 7.0 located at 1
The path count is: 1
[(1, 2)]
Merging values...
[ 6.  5.66666667  5.66666667  5.66666667  8.  8.  8.5  8. ]
The checked vector for max is:
```

```
[ 6.  5.66666667  5.66666667  5.66666667]
Finding max. Max is 6.0 located at 0
The path count is: 1
[(0, 2)]
Merging values...
[ 5.75  5.75  5.75  5.75  8.    8.    8.5   8.  ]
The checked vector for max is:
[ 5.75  5.75  5.75  5.75]
Finding max. Max is 5.75 located at 0
Procedure is attempting upward MCA for merged values
The current merges are at: [0, 1, 2, 3]
The number of paths is 0
Chopping values...
[ 5.75  5.75  5.75  5.75  8.    8.    8.5   8.  ]
All values have been chopped.
```

The given output vector, $x^* = \begin{bmatrix} 5.75 \\ 5.75 \\ 5.75 \\ 5.75 \\ 8 \\ 8 \\ 8.5 \\ 8 \end{bmatrix}$ follows the desired order restriction, and is the isotonic regression of the original input vector, $x$. This output vector could then be used as a restricted maximum likelihood estimator in hypothesis testing.

It is worth noting that there are cases of similar orderings where the MCA will fail to produce an isotonic regression. If the previous example is changed to the following diagram,

the MCA will be incapable of computing the projection. The order restriction is assumed to be the same.

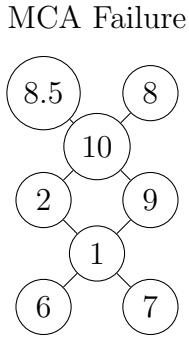


Figure 6.2: Failed MCA Example

Given that the max value of 10 has two upward paths, upward merge and chop fails. Attempting a downward merge or chop, the algorithm locates the value of 1 as the smallest value. Again, the algorithm fails as this node has two paths downward. The MCA would then terminate.

This thesis details the theory, programming, and application of the Merge and Chop Algorithm. While this paper focuses on the statistical applications of the algorithm, other fields of mathematics may be able to utilize the algorithm. Given that the algorithm is usable for partial orderings, abstract algebra may find application for the MCA. At its base, the MCA is used for finding projections to specific spaces. Thus, any field that makes use of projections onto special spaces may also find use of the algorithm. Since the MCA can be applied for polyhedral cones, the field of geometry could be one possible endeavor. Further work and study could also improve the results and of this thesis to contain a p-value calculator for testing hypotheses under order restrictions. These tests could more accurately approximate hypotheses where assumptions can be made upon the studied populations. Under normal circumstances, no restrictions would be placed upon the populations. Noticeably, not all hypothesis testing would benefit from these assumptions. Despite this, in the cases

where an order restriction is applicable, the resulting hypothesis tests are more powerful and precise. The paper written by Bartholomew (1961) proves the strength of the $\bar{E}^2$ tests compared to similar tests. Thus, order restricted statistical inference can be more meaningful and accurate.

# REFERENCES

## REFERENCES


[1] Ayer, M., Brunk, H.D., Ewing, G.M., Reid, W.T., and Silverman, E., 1955. 'An Empirical Distribution Function for Sampling with Incomplete Information' *Annals of Mathematical Statistics* **26**, 607-616.

[2] Bartholomew, D.J., 1961. 'Ordered Tests in the Analysis of Variance' *Biometrika* **48**, 325-32.

[3] Hu, X., and Hansohm, J., 2008. 'Merge and Chop in the Computation for Isotonic Regressions' *Journal of Statistical Planning and Inference* **138**, 3099-3106.

[4] Johnson, R., and Wichern, D., 2007. *Applied Multivariate Statistical Analysis* Pearson, New Jersey.

[5] Lemke, J.H., 1983. 'Estimation and Testing for Two-Way Contingency Tables within Order Restricted Inference Parameter Spaces' Ph.D Thesis, Pennsylvania State University.

[6] Luenberger, D., 1969. *Optimization by Vector Space Methods* Wiley, New York.

[7] Potthoff, R. F., and S. N. Roy, 1964. 'A Generalized Multivariate Analysis of Variance Model Useful Especially for Growth Curve Problems' *Biometrika* **51**, 313-26.

[8] Raubertas, R.F., C.I.C. Lee, and Nordheim, E.V., 1986. 'Hypothesis Tests for Normal Means Constrained by Linear Inequalities' *Communication in Statistics - Theory and Methods* **15**, 2809-33.

[9] Robertson, T., Wright, F. T., and Dykstra, R. L., 1988. *Order Restricted Statistical Inference* Wiley, New York.

[10] Rudin, W., 1966. *Real and Complex Analysis* McGraw-Hill, New York.

[11] Thompson Jr., W.A., 1962. 'The Problem of Negative Estimates of Variance Components' *Annals of Mathematical Statistics* **33**, 273-289.

[12] Zarantonello, E.H., 1971. 'Projection on Convex Sets in Hilbert Space and Spectral Theory' *Contributions to Nonlinear Functional Analysis* Academic Press, New York, 237-424.

# APPENDIX

# APPENDIX A

## MCA Program

The following is the code for the MCA. This code is written in Python. The language Python makes use of white space, and the following lines do not account for this guideline.

```
import os
os.chdir("C:\Thesis")

import numpy as np

V = open('vector.txt', 'r')
W = open('weight.txt', 'r')
O = open('order.txt', 'r')

v = V.read().split('\n')
w = W.read().split('\n')
o = O.read().split('\n')

v = np.asarray(v,dtype=np.float64)
w = np.asarray(w,dtype=np.float64)
o = np.asarray(o)

def first_check (a,b,c):
"This checks if the input files have been loaded correctly"
A = 0
B = 0
```

```
C = 0


if len(a) > 1:
A = 1

if len(b) > 1:
B = 1

if len(c) > 1:
C = 1

if A == 0 or B == 0 or C == 0:
print("Load failed.")
print("Either the input vector, the weight vector, or the order is empty
or the file cannot be found. Also, the method only works for vectors
with a length greater than 1.")
else:
print("Load successful.")

return print("First check complete.")

def outputfile (a):
```

```
file = open("output.txt", "w")
str1 = '\n'.join(str(e) for e in a)
file.write(str1)
file.close()

def matrix_check(m):
file = open("matrix.txt", "w")
file.write("The matrix created from the order is:\n")
out = ""
for row in m:
str1 = ' '.join(str(e) for e in row)
str1 = str1.replace("[","")
str1 = str1.replace("]","")
file.write(str1 + "\n")
file.close()

def create_matrix_order(order,vector):
n = len(vector)
M = [[0 for y in range(n)] for y in range(n)]
i = 0
for item in order:
item = item.replace("(","")
item = item.replace(")","")
item = item.split(",")
I = int(item[0])
```

```
J = int(item[1])
M[I - 1][J - 1] = 1
return M


def UpwardMC(vector,weight,matrix, status, output, mx, ind):
#Determine paths for merge, chop, or fail
complete = 0
if np.any(status[ind,:] == 2):
output,complete=UpwardMCMergeProtocol(vector,weight,matrix,status,output,mx,ind)
else:
j = 0
count = 0
paths = []
while j < len(vector):
if matrix[ind,j] == 1 and ind != j and not vector[ind] <= vector[j]:
count += 1
paths.append((ind,j))
j += 1
print("The path count is: " + str(count))
print(paths)
if count == 0: #chop
print("Chopping value...")
status[ind,ind] = 1
output[ind] = mx
complete = 1
```

```
elif count == 1: #merge
print("Merging values...")
for (a,b) in paths:
if a != b and np.all(status[b,:] != 2):
newvalue = ((vector[a]*weight[a]) + (vector[b] * weight[b]))
 / (weight[a] + weight[b])
np.put(output,a,newvalue)
np.put(output,b,newvalue)
complete = 1
status[a,b] = 2
status[b,a] = 2
elif a != b and np.any(status[b,:] == 2):
merges = [b]
i = 0
while i < len(vector):
if status[b,i] == 2:
merges.append(i)
i += 1
merges.append(a)
numerator = 0
denominator = 0
for index in merges:
numerator = numerator + (vector[index] * weight[index])
denominator = denominator + weight[index]
newvalue = numerator/denominator
```

```
for index in merges:
np.put(output,index,newvalue)
for item in merges:
for index in merges:
if item != index:
status[item,index] = 2
status[index,item] = 2
complete = 1
else: #fail
print("Upward MC failed. Moving to Downward MC")
print(output)
return(output,complete)

def DownwardMC(vector,weight,matrix,status,output, mn, ind):
#Determine paths for chop, merge, or fail
complete = 0
if np.any(status[:,ind] == 2):
output,complete=DownwardMCMergeProtocol(vector,weight,matrix,status,output,mn,ind)
else:
j = 0
count = 0
paths = []
while j < len(vector):
if matrix[j,ind]==1 and j != ind and not vector[j]<=vector[ind]:
count += 1
```

```
paths.append((j,ind))
j += 1
print("The path count is: " + str(count))
print(paths)
if count == 0: #chop
print("Chopping value...")
status[ind,ind] = 1
output[ind] = mn
complete = 1
elif count == 1: #merge
print("Merging values...")
for (a,b) in paths:
if a != b and np.any(status[a,:] != 2):
newvalue = ((vector[a]*weight[a]) + (vector[b] * weight[b]))
 / (weight[a] + weight[b])
np.put(output,a,newvalue)
np.put(output,b,newvalue)
complete = 1
status[a,b] = 2
status[b,a] = 2
elif a != b and np.any(status[a,:] == 2):
merges = [a]
i = 0
while i < len(vector):
if status[a,i] == 2:
```

```
merges.append(i)
i += 1
merges.append(b)
numerator = 0
denominator = 0
for index in merges:
numerator = numerator + (vector[index] * weight[index])
denominator = denominator + weight[index]
newvalue = numerator/denominator
for index in merges:
np.put(output,index,newvalue)
for item in merges:
for index in merges:
if item != index:
status[item,index] = 2
status[index,item] = 2
complete = 1
else: #fail
print("Upward MC and Downward MC failed. Aborting MCA.")
complete = 2
return (output,complete)

def UpwardMCMergeProtocol(vector,weight,matrix, status, output, mx, ind):
print("Procedure is attempting upward MCA for merged values")
complete = 0
```

```
#find merges
merges = []
merges.append(ind)
h = 0
while h < len(vector):
if status[ind,h] == 2:
merges.append(h)
h += 1
print("The current merges are at: " + str(merges))

#Determine paths for merge, chop, or fail
count = 0
paths = []

#Count paths
for index in merges:
s = 0
while s < len(vector):
if matrix[index,s]==1 and s not in merges and not vector[index]<=vector[s]:
paths.append((index,s))
s += 1
truepaths = []
for (a,b) in paths:
if b != truepaths[:] and status[b,b] != 1:
```

```
truepaths.append(b)


print("The number of paths is " + str(len(truepaths)))
count = len(truepaths)


if count == 0: #chop
print("Chopping values...")
for item in merges:
status[item,item] = 1
np.put(output,item,mx)
complete = 1
elif count == 1: #merge
print("Merging values...")
for value in truepaths:
merges = np.append(merges,value)
numerator = 0
denominator = 0
for index in merges:
numerator = numerator + (vector[index] * weight[index])
denominator = denominator + weight[index]


newvalue = numerator/denominator
for index in merges:
np.put(output,index,newvalue)
```

```
for item in merges:
for index in merges:
if item != index:
status[item,index] = 2
status[index,item] = 2
complete = 1

else: #fail
print("Upward MC failed. Moving to Downward MC")

return (output,complete)

def DownwardMCMergeProtocol(vector,weight,matrix, status, output, mn, ind):
print("Procedure is attempting downward MCA for merged values")
complete = 0

#find merges
merges = []
merges.append(ind)
h = 0
while h < len(vector):
if status[ind,h] == 2:
merges.append(h)
h += 1
print("The current merges are at: " + str(merges))
```

## APPENDIX A (continued)

```
#Determine paths for merge, chop, or fail
count = 0
paths = []

#Count paths
for index in merges:
s = 0
while s < len(vector):
if matrix[s,index]==1 and s not in merges and not vector[s]<=vector[index]:
paths.append((s,index))
s += 1
truepaths = []
for (a,b) in paths:
if a != truepaths[:] and status[a,a] != 1:
truepaths.append(a)

count = len(truepaths)
print("The number of paths is: " + str(count))

if count == 0: #chop
print("Chopping values...")
for item in merges:
status[item,item] = 1
np.put(output,item,mn)
```

```
complete = 1


elif count == 1: #merge
print("Merging values...")
for value in truepaths:
merges = np.append[merges,value]
numerator = 0
denominator = 0
for index in merges:
numerator = numerator + (vector[index] * weight[index])
denominator = denominator + weight[index]

newvalue = numerator/denominator
for index in merges:
np.put(output,index,newvalue)

for item in merges:
for index in merges:
if item != index:
status[item,index] = 2
status[index,item] = 2
complete = 1

else: #fail
print("Upward MC and Downward MC failed. Aborting MCA.")
```

```
complete = 2


return (output,complete)


def merge_and_chop(vector,weight, matrix, status, output):
k = 0
while k < len(vector):
if status[k,k] != 1: # check if entire output has been chopped yet
#locate max
mxcheck = vector
i = 0
deletes = []
while i < len(vector):
if status[i,i] == 1:
deletes.append(i)
i += 1
mxcheck = np.delete(mxcheck,deletes)
print("The checked vector for max is:")
print(mxcheck)


mx = np.amax(mxcheck)
indy = np.argwhere(vector==mx)
i = 0
while i < len(indy):
a = indy[i]
```

```
a = int(a)
if status[a,a] == 1:
i += 1
else:
ind = a
break
print("Finding max. Max is " + str(mx) + " located at " + str(ind))


#Perform Upward MC
complete = 0
output,complete = UpwardMC(vector,weight,matrix, status,output, mx, ind)
if complete == 1:
output = merge_and_chop(output,weight,matrix, status,output)
k = 0
else:
#locate min
mncheck = vector
i = 0
deletes = []
while i < len(vector):
if status[i,i] == 1:
deletes.append(i)
i += 1
mncheck = np.delete(mncheck,deletes)
print("The checked vector for min is:")
```

```
print(mncheck)


mn = np.amin(mncheck)
indy = np.argwhere(vector==mn)
i = 0
while i < len(indy):
a = indy[i]
a = int(a)
if status[a,a] == 1:
i += 1
else:
ind = a
break
print("Finding min. Min is " + str(mn) + " located at " + str(ind))


#Perform Downward MC
complete = 0
output,complete = DownwardMC(vector,weight,matrix,status,output, mn, ind)
if complete == 1:
output = merge_and_chop(output,weight,matrix, status,output)
k = 0
else:
print("Failure. Stopping Algorithm.")
Failure = "The procedure could not produce the isotonic regression."
return Failure
```

```
break
else:
k += 1
return output

def MCA (vector,weight,order):
first_check(vector,weight,order)
M = create_matrix_order(order,vector)
M = np.matrix(M)
matrix_check(M) #This just checks to see if the matrix created from the order
is correct

l = len(vector)
L = [[0 for y in range(l)] for y in range(l)] # 0 = nothing, 1 = chopped, 2 =
merged
L = np.asmatrix(L)
out = vector
out = np.asarray(out)

final = merge_and_chop(vector,weight,M,L,out)
outputfile(final)

countA = 0
for item in order:
item = item.replace("(","")
```

```
item = item.replace(")","")
item = item.split(",")
I = int(item[0])
J = int(item[1])
if J == I + 1:
countA += 1
if countA == (len(vector) - 1):
countB = 0
for item in order:
item = item.replace("(","")
item = item.replace(")","")
item = item.split(",")
I = int(item[0])
J = int(item[1])
if J == I - 1:
countB += 1
if countB == (len(vector) - 1):
print("This is a simple ordering.")

countC = 0
i = 0
while i < len(vector):
j = i
while j < len(vector):
if M[i,j] == 1:
```

```
countC += 1

j += 1

i += 1

if countC < len(vector):

print("This is a Type I ordering")


countD = 0

i = 0

while i < len(vector):

j = 0

while j <= i:

if M[j,i] == 1:

countD += 1

j += 1

i += 1

if countD < len(vector):

print("This is a Type II ordering")


print("All values have been chopped.")

print("The output vector has been stored in the 'output.txt' file.")


MCA(v,w,o)
```