DYNAMIC METHOD TO IDENTIFY AND VISUALIZE ACCIDENT CONTRIBUTING FACTORS AT HIGHWAY-RAIL GRADE CROSSINGS

BY

JACOB MATHEW

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Civil and Environmental Engineering in the Graduate College of the University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Adviser:

Professor Rahim F. Benekohal
ABSTRACT

Highway–Rail grade crossings are important locations on a highway network and safety at these locations is a critical issue for both highway and railroad network. Even though safety at grade crossings have improved over the years, over 200 people still lose their lives every year at grade crossings across the United States. Also, the injury and fatality rates are significantly higher for grade crossing crashes than other types of traffic crashes. Therefore, need for improving the safety at grade crossings is relevant today and more so especially since the demand in both highway and railroad systems continue to increase.

Safety improvements at grade crossing locations can be suggested by establishing the contributing factors for accidents at each individual grade crossing and spearheading the safety improvement recommendation based on those contributing factors. Manually identifying the most significant contributing factors from the accident database at each grade crossing is not feasible due to various reasons. This thesis presents a new, easy method to extract the most frequent nested accident trends and helps an analyst visualize these trends using a tree based structure. This procedure is called the M+C method. The work presented in this thesis builds on top of the previous research conducted in this area by considering a more comprehensive set of accident attributes and by introducing a data-driven method of determining the order of the contributing factors to accidents at each location. The algorithm presented in this paper is also implemented in a computer program using the C++ language to automate the procedure thus reducing human effort and error.

Various examples illustrating the utility of the procedure to extract accident trends and contributing factors at a grade crossing. The use of this procedure to simultaneously analyze multiple crossings including all crossings along a corridor, all crossings within a county, all crossings with single accident are demonstrated in this thesis. This procedure is also used to identify new attributes to be used in accident prediction models. This computerized procedure, combined with a user-friendly interface could be a very useful tool that practitioners can use for quick and easy analysis of accidents at grade crossings.
ACKNOWLEDGMENTS

There are several people that I would like to thank for the successful completion of this thesis.

I am extremely grateful to my adviser Prof. Rahim Benekohal for accepting me into his research group and for all the productive meetings that we went through.

I would also like to thank National University Rail Center (NURail) and the Roadway Safety Institute (RSI) for funding this project.

Gratitude is owed to Juan C. Medina, Research Assistant Professor at the University of Utah, for introducing me to the ideas on which this project is based on and for assisting me as I was getting into the project.

Last but not the least, I would like to thank my family and friends by who I am constantly surrounded.
# TABLE OF CONTENTS

| LIST OF TABLES | .......................................................... | vi |
| LIST OF FIGURES | .......................................................... | vii |

## CHAPTER 1  INTRODUCTION
1.1 Need for the Study ........................................... 1
1.2 Grade Crossing Safety Improvement Approaches ................. 1
1.3 The Static Method of Accident Data Visualization .......... 4
1.4 Objectives of the Research ................................... 6
1.5 Thesis Organization ............................................ 7

## CHAPTER 2  DYNAMIC METHOD OF ACCIDENT ANALYSIS .......... 8
2.1 Source of Data .................................................. 9
2.2 Selection of Attributes to be used in Dynamic Method ......... 9
2.3 Determining the Hierarchy of Attributes ..................... 12
2.4 Method to Determine Hierarchy of Attribute ................. 17
2.5 Example of Accident Analysis at Grade Crossing using Various Algorithms .. 17

## CHAPTER 3  ANALYSIS OF GRADE CROSSING ACCIDENTS USING DYNAMIC TREE METHOD .......... 23
3.1 Selected Crossing from Illinois ............................... 23
3.2 Selected Crossings from California ............................ 26
3.3 Selected Crossings from Indiana .............................. 32

## CHAPTER 4  DYNAMIC TREE ANALYSIS FOR MULTIPLE LOCATIONS .... 39
4.1 Corridor Analysis ............................................... 39
4.2 Regional Analysis ............................................. 47
4.3 Single Accident Locations .................................... 50

## CHAPTER 5  IDENTIFYING NEW ATTRIBUTES TO BE USED IN PREDICTION MODELS .......... 56
5.1 “HWYNEAR” ...................................................... 56
5.2 “XANGLE” ....................................................... 64

## CHAPTER 6  CONCLUSIONS AND RECOMMENDATIONS ............ 72
6.1 Recommendations for Database Improvement ................... 72
LIST OF TABLES

1.1 Attributes used in Static Method ........................................ 5
2.1 Attributes Used in Dynamic Method ................................. 11
2.2 Accident Details for Hypothetical Crossing ...................... 13
2.3 Hypothetical Database to Explain Procedure to Resolve Ties .... 16
4.1 Number of Accidents in Crossings in Corridor 1 ............... 40
4.2 Number of Accidents in Crossings in Corridor 2 ............... 43
5.1 Number of Accidents split by “HWYNEAR” and Warning Device 56
5.2 Number of Accidents split by “XANGLE” and Warning Device 64
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Dynamic Tree Using Absolute Sorting for Hypothetical Crossing</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Dynamic Tree Using Nested Sorting for Hypothetical Crossing</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>Aerial view of crossing 173887G</td>
<td>18</td>
</tr>
<tr>
<td>2.4</td>
<td>Static Tree Visualization of Accidents at Crossing 173887G</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>Accident Tree Visualization of Accidents at Crossing 173887G using AS algorithm</td>
<td>19</td>
</tr>
<tr>
<td>2.6</td>
<td>Accident Tree Visualization of Accidents at Crossing 173887G using NS algorithm</td>
<td>20</td>
</tr>
<tr>
<td>2.7</td>
<td>Accident Tree Visualization of Accidents at Crossing 173887G using MNS+CC algorithm</td>
<td>21</td>
</tr>
<tr>
<td>3.1</td>
<td>Aerial View of Crossing 608311K</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Accident Tree Visualization for Accidents at Crossing 608311K using Static Method</td>
<td>24</td>
</tr>
<tr>
<td>3.3</td>
<td>Accident Tree Visualization for Accidents at Crossing 608311K using M+C method</td>
<td>25</td>
</tr>
<tr>
<td>3.4</td>
<td>Aerial View of Crossing 811479J</td>
<td>26</td>
</tr>
<tr>
<td>3.5</td>
<td>Accident Tree Visualization for Accidents at Crossing 811479J using Static Method</td>
<td>26</td>
</tr>
<tr>
<td>3.6</td>
<td>Accident Tree Visualization for Accidents at Crossing 811479J using M+C method</td>
<td>27</td>
</tr>
<tr>
<td>3.7</td>
<td>Aerial View of Crossing 028380R</td>
<td>28</td>
</tr>
<tr>
<td>3.8</td>
<td>Accident Tree Visualization for Accidents at Crossing 028380R using Static Method</td>
<td>29</td>
</tr>
<tr>
<td>3.9</td>
<td>Accident Tree Visualization for Accidents at Crossing 028380R using M+C method</td>
<td>29</td>
</tr>
<tr>
<td>3.10</td>
<td>Aerial View of Crossing 026517B</td>
<td>30</td>
</tr>
<tr>
<td>3.11</td>
<td>Accident Tree Visualization for Accidents at Crossing 026517B using Static Method</td>
<td>31</td>
</tr>
<tr>
<td>3.12</td>
<td>Accident Tree Visualization for Accidents at Crossing 026517B using M+C method</td>
<td>31</td>
</tr>
<tr>
<td>3.13</td>
<td>Aerial View of Crossing 522646H</td>
<td>32</td>
</tr>
<tr>
<td>3.14</td>
<td>Accident Tree Visualization for Accidents at Crossing 522646H using Static Method</td>
<td>33</td>
</tr>
</tbody>
</table>
3.15 Accident Tree Visualization for Accidents at Crossing 522646H using M+C method .................................................. 33
3.16 Aerial View of Crossing 879204S .................................................. 34
3.17 Accident Tree Visualization for Accidents at Crossing 879204S using Static Method .................................................. 35
3.18 Accident Tree Visualization for Accidents at Crossing 879204S using M+C Method .................................................. 36
3.19 Accident Tree Visualization for Accidents involving Vehicles Stopped at Crossing 879204S using M+C Method .................................................. 37

4.1 Crossings in Corridor 1 .................................................. 40
4.2 Accident Tree Visualization of Accidents at Corridor 1 using Static Method .................................................. 41
4.3 Accident Tree Visualization of Accidents at Corridor 1 using M+C algorithm .................................................. 42
4.4 Crossings in Corridor 2 .................................................. 44
4.5 Accident Tree Visualization of Accidents at Corridor 2 using Static Method .................................................. 44
4.6 Accident Tree Visualization of Accidents at Corridor 2 using M+C algorithm .................................................. 45
4.7 Accident Tree Visualization of all Motorist Accidents at Corridor 2 using M+C algorithm .................................................. 46
4.8 Accident Tree Visualization of all accidents within Franklin County and Marion County .................................................. 49
4.9 Accident Tree Visualization of all accidents within Bureau County and Effingham County .................................................. 50
4.10 Accident Tree Visualization of all locations with 1 accident in Illinois between 2002 and 2011 .................................................. 51
4.11 Accident Tree Visualization of all locations with Xbucks with 1 accident in Illinois between 2002 and 2011 .................................................. 52
4.12 Accident Tree Visualization of all locations with Flashing Lights with 1 accident in Illinois between 2002 and 2011 .................................................. 53
4.13 Accident Tree Visualization of all locations with Gates with 1 accident in Illinois between 2002 and 2011 .................................................. 54

5.1 Accidents at Gated Crossings with nearby Highway under 75 feet .................................................. 57
5.2 Accidents at Gated Crossings with nearby Highway between 75 feet and 500 feet .................................................. 58
5.3 Accidents at Gated Crossings with nearby Highway over 500 feet .................................................. 59
5.4 Accidents at Crossings with flashing lights with nearby Highway under 75 feet .................................................. 60
5.5 Accidents at Crossings with flashing lights with nearby Highway over 500 feet .................................................. 61
5.6 Accidents at Crossings with crossbucks with nearby Highway under 75 feet .................................................. 62
5.7 Accidents at Crossings with crossbucks nearby Highway over 500 feet .................................................. 63
5.8 Accidents at Gated Crossings with crossing angle under 60 degrees .................................................. 65
5.9 Accidents at Gated Crossings with crossing angle over 60 degrees .................................................. 66
5.10 Accidents at Crossings with flashing lights with crossing angle under 60 degrees .................................................. 67
5.11 Accidents at Crossings with flashing lights with crossing angle over 60 degrees .................................................. 68
5.12 Accidents at Crossings with crossbucks with angle under 60 degrees .................................................. 69
5.13 Accidents at Crossings with crossbucks with angle over 60 degrees .................................................. 70
1.1 Need for the Study

Over the years, the number of fatalities and injuries at a highway rail grade crossing has seen a declining trend. Between the years 2001 and 2011, the number of fatalities and injuries have decreased by nearly 40% [1]. Even though this safety picture has improved over the years, over 200 people still lose their lives at railroad crossings in the United States each year [2]. It should also be noted that “...because of the substantial mass difference between train and vehicle, the train vehicle crash injury and fatality rates are much higher than other types of traffic crashes” [3]. Therefore need for improving the safety at grade crossings is relevant today and more so especially since the demand in both highway and railroad systems continue to increase.

1.2 Grade Crossing Safety Improvement Approaches

Various researchers and institutions have used different approaches to improve the railroad grade crossing safety as listed below.

Yeh et. al. did a literature review to better understand the driver behavior at grade crossings in order to develop countermeasures to discourage dangerous driver behavior. Some of the findings from the literature review indicate that the drivers were “quite willing to violate active signals.” The authors suggests an explicit method (use of barrier gates) as well as an implicit method (improve credibility of the warning device) to improve compliance. Countermeasures suggested to compensate for the reduced driver skills include techniques for improving the detection of the crossing at night and installing additional signs indicating the required action from the highway user. Countermeasures suggested to compensate for driver attitude and driving attitude include information campaigns, reduction in the perception of peer approval for committing violations [4].

Coleman et. al. discusses the state of the practice on railroad-highway grade crossings
with a look to the future. They identified that the current state of practice at the railroad grade crossing is the use of passive (i.e. crossbucks, pavement markings etc.) and active warning devices (flashing lights, gates, audible bells etc.). They also discuss various emerging technologies and practices (Ohio buckeye shield for passive crossings, use of LED flashing signals, four–quadrant gates etc.) which are being evaluated in the efforts to improve safety at grade crossings [5].

Mok et. al. looked into reasons for improved safety at railroad–highway grade crossings between 1975 and 2001 years and identified the relative contribution of the factors to safety improvement. They identified that two-fifths of the decrease in the number of incidents is attributed to reduction in drunk driving and improved emergency response, a fifth of the decrease is due to installation of gates and flashing lights, a seventh of the reduction was due to Operation Lifesaver public education campaign and installation of lights on locomotives while a tenth of the reduction was due to crossing closures/consolidation of little used crossings [6].

Horton et.al conducted literature review and identified 11 success factors which are likely contributors to improvement in grade crossing safety. The factors identified are rule makings, changes or advances in the grade crossing and transportation environment, and political, societal, and economic changes [7, 8, 9].

The main element in the grade crossing safety improvement program along the Metro Blue Line (MBL) in Los Angeles was the expansion of the grade crossing enforcement efforts. [10]. The Los Angeles County Metropolitan Transportation Authority used photo enforcement systems along MBL which has lead to a significant reduction in the number of grade crossing violations. Accident experience at crossing was used as a factor to determine the problem locations. Potential areas of concern were determined by studying locations with broken gate arms.

The Texas department of transportation’s Texas Highway-Rail Grade Crossing Safety Action Plan is designed to improve grade crossing safety. Texas DOT and FRA developed action plan strategies based on the significant findings of the analysis of the grade crossing accidents in the state between 2003 and 2007. Some of the evaluation/engineering strategies recommended include identification and mitigation of signal preemption issues at signalized crossings, improving crossing inventory data on crossings with signal preemption, upgrade passive warning devices to active warning devices at un-signalized crossings etc. [11]

The Minnesota department of transportation focuses its grade crossing improvement efforts on high hazard locations. Areas of local concern and areas with antiquated equipment are also considered for review for safety improvements. The MnDOT uses the USDOT accident prediction formula to determine the hazardous locations. The local concerns are
identified from local road authorities, railroad, local planning organization or MnDOT district staff etc. Locations with signals older than 25–30 years of age (normal lifetime of a railroad highway grade crossing signal) are also considered for safety improvement [12].

This literature review didn’t reveal a procedure to identify accident contributing factors specific which is unique to each grade crossing which is a research gap. The methodology described in this thesis is an attempt to bridge the gap.

The Federal Railroad Administration maintains a database of grade crossing accidents across the U.S. which is publicly available online [13]. There is a need to analyze the order of the contributing factors of accidents to recommend appropriate safety improvements at each crossing. If this order is not established properly, any trends and significant contributing factors of the accidents may remain hidden within the database. Establishing the order of contributing factors of accidents at a grade crossing is a challenging task because

1. The number of possible orders of the contributing factors to accidents could be very high.
   The number of possible orders for 6 contributing factors would be 720 (which is 6!).
   This is a very large number to handle manually.

2. The order of the contributing factors for different grade crossings could be different.
   The order of the contributing factors identified for one grade crossing cannot be used for another grade crossing because accidents at different crossings could be different. Therefore, each crossing should be analyzed individually to suggest safety improvements.

3. The complexity of this process increases when multiple crossings are considered.
   When multiple crossings along a corridor or over a region is considered to be analyzed simultaneously, then the number of possible order of the contributing factor for accidents would increase exponentially with each additional crossing.

4. Including more factors into the analysis increases the number permutations for its order.
   Using 20 attributes available in the FRA accident database can result in over $2.4 \times 10^{18}$ attributes. This is practically impossible to analyze manually.

Due to the above mentioned reasons, a computerized program to determine the order of the contributing factor of grade crossing accidents is required. It becomes even more relevant when the analysis is to be repeated on an annual or a semi-annual basis. This thesis presents a new, easy, data-driven method to extract the most frequent nested accident trends. This
method uses a tree structure to visualize the accident frequencies and to show the hierarchy of the most common accident attributes at a grade crossing or a group of crossings analyzed simultaneously.

1.3 The Static Method of Accident Data Visualization

This research builds on the “static method” of tree based visualization of grade crossing accidents. The static method is a microscopic approach to accident analysis where individual characteristics of accidents are investigated to determine the potential contributing factors at the location. The accident attributes such as driver characteristics, visibility, speed and direction of the vehicles involved etc. are taken into account in the micro-level analysis approach [14, 15, 16]. Table 1.1 lists out the attributes which were used in the static method.
Table 1.1: Attributes used in Static Method

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Definition</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPVEH</td>
<td>Highway User Type</td>
<td>Pedestrian, motorist or others</td>
</tr>
<tr>
<td>MOTORIST</td>
<td>Action of Highway User</td>
<td>1. Drove around or thru gate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Stopped and then proceeded</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Did not stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Stopped on crossing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. Others</td>
</tr>
<tr>
<td>VEHDIR</td>
<td>Highway User Direction</td>
<td>1. North</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. South</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. East</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. West</td>
</tr>
<tr>
<td>TRNDIR</td>
<td>Timetable Direction</td>
<td>1. North</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. South</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. East</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. West</td>
</tr>
<tr>
<td>TYPACC</td>
<td>Circumstance of Accident</td>
<td>1. Train hit highway vehicle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Highway vehicle hit train</td>
</tr>
<tr>
<td>DRIVAGE</td>
<td>Vehicle Driver Age</td>
<td>Numerical value of the highway user’s age</td>
</tr>
<tr>
<td>DRIVGEN</td>
<td>Vehicle Driver Gender</td>
<td>1. Male</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Female</td>
</tr>
<tr>
<td>WEATHER</td>
<td>Weather Conditions</td>
<td>1. Clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Cloudy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Rain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Fog</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. Sleet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6. Snow</td>
</tr>
<tr>
<td>VISIBILITY</td>
<td>Visibility</td>
<td>1. Dawn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Day</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Dusk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Dark</td>
</tr>
</tbody>
</table>
The nine attributes listed in Table 1.1 are used in the same order as listed in the table to produce a “static tree”. This order of the attributes is pre-determined and remains the same for all the crossings beings analyzed.

1.3.1 Limitations and Possible Improvements of Static Method

The static method has several limitations as listed below

1. Use of only 9 attributes. Explore the use of other attributes in FRA accident database. The static method uses just 9 attributes which limits the potential of information extraction from the database using the static method. The use of other attributes in the FRA accident database like “POSITION” (position of highway user), “VIEW” (Primary obstruction of track view) etc. can be included in a micro-level analysis to be able to extract further information about the accident from the database.

2. Accidents trends are identified only according to the pre-determined hierarchy. Use a data-driven method to determine the hierarchy of the contributing factors. A pre-determined hierarchy of the accident attributes are used in the static method. This could limit the information that could be extracted from the database and a dynamic, data-driven method is required to replace it.

3. Use of a computerized procedure. As mentioned earlier in this section, manual analysis to determine the order of the accident contributing factor is not a feasible task and a computerized procedure is required to do the same. Computerization of this process can eliminate human error, reduce time of analysis and also expand avenues for further research.

1.4 Objectives of the Research

The objectives of this research are

1. Explore the FRA accident database and identify accident attributes which could be used in the analysis in addition to the attributes used in the static method.

2. Develop a data-driven methodology to prioritize the accident attributes to determine the order of the contributing factor of the accident.

3. Write a computer program to implement the methodology developed.
This method proposed in this thesis is a micro-level analysis approach and builds on the
tree-style visualization of the accident trends used in the static method. 22 attributes are
chosen from the FRA accident database to include in this analysis. This method uses a
data-driven approach to develop the order of the contributing factors rather than relying
on a pre-determined hierarchy of the accident attributes. A computer code is also written
(given in Appendix) in C++ to automate the algorithm for quick and easy use. This method
could be used by practitioners to inspect grade crossing to determine the contributing factors
for accidents and recommend improvements and can complement macroscopic procedures
like USDOT accident prediction formula [17].

1.5 Thesis Organization

This thesis is organized into 6 chapters including the introduction. Chapter 2 briefly lists
out the selection of the accident attributes to be used in the analysis and explains in detail
the development of the proposed method. This various methods discussed in the chapter
reveals chronologically the thought process that leads to the development of the final method.
Chapters 3 and 4 shows the utility of the method for single crossing analysis and multiple
crossing analysis. Various examples comparing the static method and the new method of the
crossing are also given in these chapters. Chapter 5 details out the the use of the dynamic tree
to identify additional factors which may be used in the macro model for accident prediction.
Chapter 6 discusses the suggested procedures to improve information extraction using the
proposed method. Recommendations for further research are also listed in this chapter
followed by a short conclusion.

The C++ code used to implement the developed algorithm is given in the Appendix.
Various limitations were identified in the static tree method of accident analysis. To overcome these limitations, the dynamic tree method of accident analysis was developed. The dynamic tree method builds on the static tree method. The highlights of this method include

1. Consider all potential attributes in the FRA accident database.

   The static tree method is limited in its ability to extract information by the number of attributes that were considered in its methodology. The dynamic tree method discussed in this thesis considers twenty two attributes from the FRA accident database. The selected attributes and the basis of selection of these attributes are given in Section 2.2

2. Data driven selection of the hierarchy of the attributes.

   The attributes that gives us information about the accidents at the crossing may be unique for each crossing. The hierarchy of the attributes could also be unique for each crossing. Thus a predetermined hierarchy of the attributes which was used in the static tree method is of little value. The dynamic tree method relies on a data driven procedure to determine the hierarchy of the attributes thus creating a unique hierarchy of attributes for each of the crossing. The unique hierarchy created for each of the crossing(s) depends on the accident characteristics at the crossing(s) and thus improves the potential for discovering more useful information. The three different algorithms developed to generate a hierarchy of the attributes are discussed in Section 2.3

3. Computerization of the process.

   The dynamic tree algorithm is computerized. This reduces the time required to consider all the selected attributes to determine the hierarchy of the attributes. Another advantage of computerization of the process is the reduction in the possibility of human error. The C++ code written to implement the procedure is given in Appendix A.
The following sections of this chapter describes the source of data, selection of attributes, algorithms to determine the hierarchy of the attributes with their advantages and disadvantages and details out the procedure selected in the dynamic tree procedure. The chapter concludes with an illustration of the three algorithms discussed along with the algorithms selected so that a comparison could be made between all the 4 methods.

2.1 Source of Data

FRA maintains the Highway Rail Accident (HRA) Database which was used in this procedure. This database is populated via the forms FRA F 6180.57 (Highway-Rail Grade Crossing Accident/Incident Report) and FRA F 6180.150 (Highway User Injury Inquiry Form) and contains information regarding “any impact, regardless of severity, between railroad on-track equipment and a highway user at a highway-rail grade crossing site” [18]. Data for all accidents that occurred in the state of Illinois between 2002 and 2011 was used while developing the dynamic tree method.

2.2 Selection of Attributes to be used in Dynamic Method

The HRA Database consists of 103 attributes. Those attributes were selected which described

1. the time of the accident (e.g. MONTH, AMPM etc.)

2. the crossing condition at the time of the accident (LOCWARN, WARNSIG, VIEW etc.)

3. the characteristics and behavior of the highway vehicle driver involved in the accident (MOTORIST, DRIVAGE etc.)

4. the speed and direction of the vehicle/train involved at the crossing (VEHSPD, TRNDIR etc.)

22 attributes were chosen among the 103 attributes available in the FRA accident database. The attributes which were not selected include

1. those specifying the location of the crossing (e.g. STATION, COUNTY etc.)

2. redundant attributes (e.g. IYR, IYR2, YEAR etc. which carry the same value)
3. describing the consequence of accident (e.g. TOTALKLD, TOTALINJ etc.)

4. accident narrative (NARR1, NARR2 etc.)

5. blank dummy fields. (DUMMY1, DUMMY2 etc. which are blank data expansion fields)
## Table 2.1: Attributes Used in Dynamic Method

<table>
<thead>
<tr>
<th>No.</th>
<th>Attribute</th>
<th>Number of Sub-Categories</th>
<th>Definition</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MONTH</td>
<td>4</td>
<td>Month of Incident</td>
<td>Month of Incident. Classified based on Season</td>
</tr>
<tr>
<td>2</td>
<td>AMPM</td>
<td>2</td>
<td>AM or PM</td>
<td>AM or PM</td>
</tr>
<tr>
<td>3</td>
<td>VEHSPD</td>
<td>4</td>
<td>Vehicle estimated speed</td>
<td>&lt;20, 20-40, 40-60, &gt;60</td>
</tr>
<tr>
<td>4</td>
<td>TYPVEH</td>
<td>3</td>
<td>Highway user type</td>
<td>Motorist, Pedestrian, Others</td>
</tr>
<tr>
<td>5</td>
<td>VEHDIR</td>
<td>4</td>
<td>Highway user direction</td>
<td>North, South, East or West</td>
</tr>
<tr>
<td>6</td>
<td>POSITION</td>
<td>4</td>
<td>Position of Highway user</td>
<td>Stalled, Stopped, Moving, Trapped</td>
</tr>
<tr>
<td>7</td>
<td>TYPACC</td>
<td>2</td>
<td>Circumstance of accident</td>
<td>Train struck Highway user, Train struck by Highway User</td>
</tr>
<tr>
<td>8</td>
<td>VISIBILITY</td>
<td>4</td>
<td>Visibility</td>
<td>Dawn, Day, Dusk, Dark</td>
</tr>
<tr>
<td>9</td>
<td>WEATHER</td>
<td>6</td>
<td>Weather condition</td>
<td>Clear, Cloudy, Rain, Fog, Sleet, Snow</td>
</tr>
<tr>
<td>10</td>
<td>TYPTRK</td>
<td>4</td>
<td>Type of track</td>
<td>Main, Yard, Siding, Industry</td>
</tr>
<tr>
<td>11</td>
<td>TRKCLAS</td>
<td>10</td>
<td>FRA Track Class</td>
<td>FRA track class: 1-9, X [19]</td>
</tr>
<tr>
<td>12</td>
<td>TRNSPD</td>
<td>4</td>
<td>Speed of Train</td>
<td>&lt;20, 20-40, 40-60, &gt;60</td>
</tr>
<tr>
<td>13</td>
<td>TRNDIR</td>
<td>4</td>
<td>Timetable Direction</td>
<td>North, South, East or West</td>
</tr>
<tr>
<td>14</td>
<td>LOCWARN</td>
<td>3</td>
<td>Location of Warning Device</td>
<td>Both sides, Side of vehicle approach, Opposite side</td>
</tr>
<tr>
<td>15</td>
<td>WARNSIG</td>
<td>3</td>
<td>Crossing warning interconnected with Highway Signal</td>
<td>Yes, No or Unknown</td>
</tr>
<tr>
<td>16</td>
<td>LIGHTS</td>
<td>3</td>
<td>Crossing illuminated by street lights or special lights</td>
<td>Yes, No or Unknown</td>
</tr>
<tr>
<td>17</td>
<td>MOTORIST</td>
<td>5</td>
<td>Action of highway user</td>
<td>Drove around or through gate, Stopped and then proceeded, Did not stop, Stopped on crossing, Others</td>
</tr>
<tr>
<td>18</td>
<td>VIEW</td>
<td>8</td>
<td>Primary obstruction to track view</td>
<td>Permanent structure, Standing RR equipment, Passing train, Topography, Vegetation, Highway vehicles, other not obstructed</td>
</tr>
<tr>
<td>19</td>
<td>CROSSING</td>
<td>2</td>
<td>Type of warning device at crossing</td>
<td>Some warning device at the crossing, No warning device at the crossing</td>
</tr>
<tr>
<td>20</td>
<td>PUBLIC</td>
<td>2</td>
<td>Public crossing</td>
<td>Public or Private</td>
</tr>
<tr>
<td>21</td>
<td>DRIVAGE</td>
<td>3</td>
<td>Highway user’s age</td>
<td>&lt;30, 30-60, 60-60</td>
</tr>
<tr>
<td>22</td>
<td>DRIVGEN</td>
<td>2</td>
<td>Highway user’s gender</td>
<td>Male or Female</td>
</tr>
</tbody>
</table>
2.3 Determining the Hierarchy of Attributes

Various algorithms were considered to determine the hierarchy. This section of the chapter discusses the three different algorithms developed to determine the hierarchy. The order in which the algorithms are presented show how the final method evolved. The three algorithms are

1. Absolute Sorting Prioritization Algorithm
2. Nested Sorting Prioritization Algorithm
3. Modified Nested Sorting Prioritization Algorithm

2.3.1 Absolute Sorting Prioritization Algorithm

The absolute sorting (AS) prioritization algorithm is a simple algorithm that determines the hierarchy of the attributes based on the frequency with which each attribute clusters the accidents at the crossing(s) analyzed into its sub-attributes. A step by step procedure explaining this algorithm is given below.

1. The total number of accidents is divided into the sub-attributes of the 22 attributes.
2. The sub-attributes in each attribute which holds the highest number of accidents is identified.
3. The largest sub-attributes (found in step 2) are sorted in the descending order. This order gives the hierarchy of attributes.
4. The tree is built following the order of attributes.

To explain AS prioritization algorithm, a hypothetical crossing is considered. This hypothetical crossing has 12 accidents. Only three attributes (A1, A2 and A3) are considered for the purpose of illustration. Each of the attributes have 2 sub-attributes i.e. attribute A1 has sub-attributes A11 and A12, attribute A2 has sub-attributes A21 and A22 while attribute A3 has sub-attributes A31 and A32. The details of the accidents are given in Table 2.2
For this hypothetical crossing, out of the 12 accidents, 10 come under the sub-attribute A11 while 2 come under the sub-attribute A12; 7 accidents come under the sub-attribute A21 while 5 attributes come under the sub-attribute A22; 8 accidents come under the sub-attribute A31 while 4 accidents come under the sub-attribute A32. In the second step of the AS algorithm, we identify the sub-attributes that holds the highest number of accidents in each attribute i.e. A11, A21 and A31 in this illustration. The hierarchy of the attributes is formed in step three of the AS algorithm by sorting largest sub-attributes in each attribute in descending order i.e. since A11 > A31 > A21, the hierarchy determined by AS algorithm is A1, A3, A2. The dynamic tree is build based on this hierarchy and the tree built for the hypothetical crossing is shown in Figure 2.1.
The advantage of the AS algorithm over the procedure used in static method is the inclusion of more attributes in the analysis. However the AS algorithm does not follow through the “main branch” (the branch that groups the largest number of accidents). Nonetheless, this algorithm is relatively simple and the hierarchy obtained depends on the distribution of the accidents into subcategories.

2.3.2 Nested Sorting Prioritization Algorithm

The Nested Sorting Prioritization Algorithm (NS algorithm) is an improvement over the AS algorithm. The idea behind the NS algorithm is to emphasize focus on the “main” branch of the dynamic tree. The NS algorithm looks into that node of the tree which clusters the highest number of accidents while determining the hierarchy of the attributes at the crossing(s). A step by step procedure explaining the algorithm is given below.

1. The highest ranking attribute is selected as the attribute at the top level of the tree using the procedure described in absolute sorting

2. The accidents in the largest subcategory in the attribute selected in the previous step are further divided into sub-attributes using the attributes which are not selected yet.

3. The attribute which gives the highest concentration of accidents in a sub-attributes is selected as 2nd attribute in the hierarchy dynamically.

4. Steps 2 and 3 are repeated until all the attributes are selected and the dynamic tree is formed.

It can be seen from the above two algorithms that the NS algorithm determines the hierarchy of the attributes as the dynamic tree is formed while the AS algorithm determines the hierarchy before the formation of the dynamic tree (the hierarchy based on the AS algorithm is determined in step 3 of the algorithm before the tree is formed). The dynamic tree is build based on this hierarchy and the tree built for the hypothetical crossing of Table 2.2 is shown in Figure 2.2.

The NS algorithm is an improvement over the AS algorithm but it has it’s own limitations. The limitations of this algorithm include

1. Inability to resolve ties while determining the hierarchy of the attributes
   The algorithm selects that attribute which appears earlier in the database over the attribute that appears later in such a scenario.
2. This algorithm focuses on the “main branch” of the tree. The analysis of the accidents on the main branch may take the focus of the analysis away from the accidents on the other branch. To address these limitations, the Modified Nested Sorting Method was developed.

2.3.3 Modified Nested Sorting Prioritization Algorithm

The modified nested sorting (MNS) method improves the NS method by removing the first limitation of the NS method mentioned above. The limitation due to ties in ranking is resolved by computing a Tie Score for each attribute and ranking them based on the Tie Score values. The Tie Score is computed using the historic accident data from the database. The Tie Score for an attribute is the sum of the number of accidents in the largest subcategory of that attribute in all the crossings present in the database analyzed. To explain the calculation of the Tie Score values, consider a hypothetical database with 10 locations (crossings). Each of the crossings have 3 attributes i.e. A1, A2 and A3 and each of the attributes have 2 sub-attributes. This is given in Table 2.3.
### Table 2.3: Hypothetical Database to Explain Procedure to Resolve Ties

<table>
<thead>
<tr>
<th>Locations</th>
<th>No. of Accidents at Location</th>
<th>Attribute A1</th>
<th>Attribute A1</th>
<th>Attribute A3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A11</td>
<td>A12</td>
<td>A21</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>10</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>5</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>6</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>6</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tie Score</td>
<td></td>
<td>35</td>
<td>28</td>
<td>24</td>
</tr>
</tbody>
</table>

*Accident frequencies in bold are used to estimate the Tie Scores

The Tie Score of attribute A1 is calculated as the sum of the values in the largest sub-categories in all 10 locations. For A1 this number is the sum of 10, 5, 2, 1, 6, 2, 6, 1, 1 and 1; the sum is equal to 35. The Tie Scores of attributes A2 and A3 are calculated similarly, resulting in 28 and 24, respectively. Since the Tie Score for an attribute is calculated based all the crossings in the database, these values gives an indication as to how well an attribute can group more number of accidents in each crossing into one of its sub-attributes. The numerical value of a Tie Score therefore indicates a tendency of an attribute to cluster accidents into one of its sub-attributes. An attribute with a higher Tie Score has a stronger tendency to cluster more accidents into one of its sub-attributes as opposed to an attribute with a lower Tie Score. Hence, an attribute with a higher Tie Score will appear in the hierarchy above an attribute with a lower Tie Score in case of a tie. A step by step procedure explaining the algorithm is given below.

1. The same as in Step 1 of the NS method.
2. The same as in Step 2 of the NS method.
3. The same as in Step 3 of the NS method.
4. If two or more attributes has a tie in the hierarchy, compute the Tie Score and use it to break the tie.
5. The procedure is repeated until all the attributes are ranked and the dynamic tree is formed.
The MNS algorithm does not address the second limitation of the NS method. The contribution of the accidents in the other branches of the tree should also be represented in the dynamic tree so that a complete picture of the accidents at a crossing(s) is obtained.

2.4 Method to Determine Hierarchy of Attribute

To resolve the second limitation of the NS algorithm, a new variable was defined called the “Crossing Cluster”. The crossing cluster is calculated for each of the attributes in the hierarchy as returned by the MNS algorithm. The crossing cluster is defined as the sum of the number of accidents in a sub-attribute of an attribute across all branches of the dynamic tree.

The final chosen algorithm is a combination of the tree developed using a dynamic hierarchy of attributes returned by the MNS algorithm and the crossing cluster. This is called the M+C algorithm, short for Modified Nested Sorting + Crossing Cluster algorithm. Examples of the static tree and the dynamic trees generated for a crossing is given in the following subsection

2.5 Example of Accident Analysis at Grade Crossing using Various Algorithms

The crossing analyzed is 173887G located in the city of Chicago. 9 accidents occurred at this crossing during the years between 2002 and 2011. The Figure 2.3 shows an aerial view of the crossing. Figure 2.4 gives the tree generated using the static method. Figure 2.6 illustrates the tree generated using the hierarchy as determined by the MNS algorithm. Finally Figure 2.7 showcases the tree developed using the M+C method.
It is noted that the variable VEHDIR recorded in the database was interpreted as southbound if the values were 2 (south) or 4 (west), and northbound if the values were 1 (north) or 3 (east). Similarly, the variable TRNDIR was interpreted as eastbound if the values were 2 (south) or 3 (east) and westbound if the values were 1 (north) or 4 (west).
Figure 2.4: Static Tree Visualization of Accidents at Crossing 173887G

Figure 2.4 shows the implementation of the static tree for the crossing 173887G. This tree follows the fixed hierarchy as given in Table 1.1. It can be seen from the static tree that there were 4 cases of gate violations and 4 cases where the action of the motorist was “others” (stopped at the crossing before the gate was lowered as read from the narrative). This tree also tells the analyst that 7 accidents involved a train traveling east and that 6 of the accidents involved a train striking a highway vehicle.

Figure 2.5: Accident Tree Visualization of Accidents at Crossing 173887G using AS algorithm

*one accident record didn’t have the vehicle speed entry.*
Figure 2.5 shows the implementation of the dynamic tree using the hierarchy as returned by the AS algorithm. Even though the more attributes are used in the dynamic tree as compared to the static tree, this method is not able to reveal any further information about the accidents at the crossing. It can also be seen that the hierarchy of the attributes is dependent on the accidents that occurred at the crossing.

*one accident record didn’t have the vehicle speed entry.

Figure 2.6: Accident Tree Visualization of Accidents at Crossing 173887G using NS algorithm

Figure 2.6 shows the implementation of the dynamic tree using the hierarchy as returned by the NS algorithm. From the tree it can be seen that

1. 8 accidents involved highway vehicles at speeds under 20 mph
2. 7 out of the 8 accidents involved a train traveling east
3. 6 out of the 7 accidents involved a train striking a highway user
4. All of the above 6 accidents involved a highway vehicle driver between the ages of 30 and 60
5. 5 out of the 6 accidents involved the vehicles stopping on the crossing
8. 4 out of the 5 vehicles were stopped on the crossing before gates were lowered. (This information is obtained from the narrative column in the accident database).

Figure 2.7 shows the implementation of the dynamic tree using the hierarchy as returned by the MNS algorithm. The crossing cluster is also shown in this figure.

![Dynamic Tree and Attributes Diagram]

**Figure 2.7: Accident Tree Visualization of Accidents at Crossing 173887G using MNS+CC algorithm**

All the observations that were made from the dynamic tree using NS method could be made using the M+C method from Figure 2.7. The M+C algorithm gives us further information about the driver behavior as it is seen in the crossing cluster that half of the motor vehicles involved in accidents tried to drive around the gate (the last entry in the crossing cluster). The figure also reveals that 5 of the accidents involved a south bound highway user.

Based on the Figure 2.7 the trends in the accidents at crossing 173887G identified are

1. 7 out of the 8 motor vehicle accidents involved an east bound train.

2. 50% of the motor vehicle drivers who were involved in the accident were gate violators.
3. Around 60% of the motor vehicle drivers had stopped on the crossing before the train struck them. An analyst should be interested in exploring the reason as to why this is the case.

Thus, M+C method of accident data visualization aids an analyst in determining the trends and potential contributing factors to the accidents at grade crossing.
CHAPTER 3

ANALYSIS OF GRADE CROSSING ACCIDENTS USING DYNAMIC TREE METHOD

This chapter gives further examples of accident analysis at grade crossings using the M+C method. We use recent accident data (2005 to 2014) and applied the static method and M+C method to compare the two methods. In order to validate this method and to show that the M+C approach is not restricted to state of Illinois, accident analysis for crossings in Indiana and California are also considered.

3.1 Selected Crossing from Illinois

The crossing in Illinois with the highest accident frequency between the years 2005 and 2014 was selected. The crossing was 608311K and had 7 accidents between 2005 and 2014. It is located at the crossing of rail line and W 119th Street in Blue Island. Figure 3.1 gives the aerial view of the crossing.

It is noted that the variable VEHDIR recorded in the database was interpreted as eastbound if the values were 2 (south) or 3 (east), and westbound if the values were 1 (north) or 4 (west). Similarly, the variable TRNDIR was interpreted as northbound if the values were 1 (north) or 3 (east) and southbound if the values were 2 (south) or 4 (west). Figure 3.2 and Figure 3.3 are respectively the trees developed using the static method and the M+C method for accidents at 608311K.
Figure 3.1: Aerial View of Crossing 608311K

Figure 3.2: Accident Tree Visualization for Accidents at Crossing 608311K using Static Method
From both Figure 3.2 and Figure 3.3 it can be observed that 5 out of 7 motor vehicles involved in accidents drove around the gate. Further information is revealed from Figure 3.3 which is listed below.

1. 3 out of the 7 accidents involved a highway user striking a train
2. In all of the above 3 accidents, the train was headed in the North direction
3. 4 out of the 7 accidents occurred on FRA track class 1, indicating trains with lower speeds.

Thus, it can be seen that the dynamic tree using M+C method is able to easily reveal more information from the database thus aiding the analyst to make informed decisions regarding safety improvements at the crossing.
3.2 Selected Crossings from California

Crossing 811479J is located at Nogales St and UP rail line in Los Angeles. There were 8 accidents involving motor vehicles at this location during the analysis period. The Figure 3.4 gives the aerial view of the crossing. The Figure 3.5 shows the static tree generated for the accidents at the crossing while the Figure 3.6 gives the dynamic tree generated using the M+C method.

*The highway user involved represented by others was a trailer

Figure 3.4: Aerial View of Crossing 811479J

Figure 3.5: Accident Tree Visualization for Accidents at Crossing 811479J using Static Method
*The highway user involved represented by others was a trailer.

**Figure 3.6: Accident Tree Visualization for Accidents at Crossing 811479J using M+C method**

From the static tree given in the Figure 3.5 it could be seen that 5 out of the 9 accidents involved a highway user driving being stopped on the crossing. It can also be seen that the highway user was headed in the southbound direction in all the 5 cases. The tree developed using M+C method as shown in Figure 3.6 helps to reveal further information regarding the accidents at this crossing. It can be seen from the dynamic tree that

1. All the highway vehicles involved in the accident were traveling south.

2. 8 out of the 9 accidents occurred during the PM hours.

From the crossing cluster, it is observed that
1. 5 out of the 9 vehicles were stopped on the crossing.

2. 3 out of the 9 vehicles were trapped at the crossing.

Based on the information gained from the M+C method, an analyst should be interested in the reasons behind the high frequency of accidents involving southbound vehicles stopped/trapped on the crossing.

The second crossing selected from California was 028380R. This crossing is located at the intersection between Kratzmeyer Road and BNSF rail line in Bakersfield, CA. Six accidents were observed at this location between 2005 to 2014. Figure 3.7 gives the aerial view of the crossing. The Figure 3.8 shows the static tree generated for the accidents at the crossing while the Figure 3.9 gives the dynamic tree generated using the M+C method. It is to be noted that VEHDIR values coded as 1 and 3 are considered east bound vehicles while the VEHDIR values coded and 2 and 4 are considered west bound vehicles.

Figure 3.7: Aerial View of Crossing 028380R
Figure 3.8: Accident Tree Visualization for Accidents at Crossing 028380R using Static Method

Figure 3.9: Accident Tree Visualization for Accidents at Crossing 028380R using M+C method
From the static tree shown in Figure 3.8 it can be seen that all 6 accidents involved a motor vehicle. In 3 of the cases the motor vehicle didn’t stop before the crossing while in 2 cases, the motor vehicle was stopped on the crossing. The dynamic tree generated using the M+C method as shown in Figure 3.9 reveals that

1. 5 of the accidents involved a west bound train
2. 5 of the accidents involved a west bound vehicle
3. 4 of the accidents involved trains traveling at more than 60 mph. It is also seen that 5 accidents involved trains on FRA track class 4 indicating a maximum train speed between 60 and 80 mph.
4. 4 of the accidents occurred during day time
5. 2 of the accidents had track view obstructions caused by standing railroad equipment.

The third crossing selected from California is 026517B which is located at the crossing of Magnolia Avenue and BNSF rail line in San Bernardino. This crossing observed 6 accidents between the years 2005 and 2014. Figure 3.10 gives the aerial view of the crossing. The Figure 3.11 shows the static tree generated for the accidents at the crossing while the Figure 3.12 gives the dynamic tree generated using the M+C method.

![Figure 3.10: Aerial View of Crossing 026517B](image)
Figure 3.11: Accident Tree Visualization for Accidents at Crossing 026517B using Static Method

Figure 3.12: Accident Tree Visualization for Accidents at Crossing 026517B using M+C method
From both Figure 3.11 and Figure 3.12, it can be seen that all the accidents involve an east bound vehicle as well as an eastbound train. It is also seen that in 5 of the accidents, the vehicle was stopped on the crossing. Additional information extracted from Figure 3.12 include

1. All the accidents involved the train striking the highway user
2. All the accidents occurred during PM hours

The directionality issue is very clear from the figures since all the accidents involve both vehicles and the train traveling eastbound. This could be caused due to the tight angle between the rail line and the highway causing reduced visibility for the highway user.

### 3.3 Selected Crossings from Indiana

The first crossing selected from the state of Indiana is 522646H. This crossing is located at the intersection of Clark Road and NS rail line in New Castle. During the analysis period between 2005 and 2014, 6 accidents were observed at this crossing. Figure 3.13 gives the aerial view of the crossing. The Figure 3.14 shows the static tree generated for the accidents at the crossing while the Figure 3.15 gives the dynamic tree generated using the M+C method. It is to be noted that TRNDIR values 1 and 4 were considered westbound trains.

![Figure 3.13: Aerial View of Crossing 522646H](image-url)
Figure 3.14: Accident Tree Visualization for Accidents at Crossing 522646H using Static Method

Figure 3.15: Accident Tree Visualization for Accidents at Crossing 522646H using M+C method
From Figure 3.14 and Figure 3.15, it is observed that

1. All the accidents involved southbound vehicles.
2. 5 of the accidents involved an eastbound train
3. All the accidents involved a train striking a HW vehicle.
4. All the accidents involved vehicles stopped on the crossing.

Additional information from the dynamic tree in Figure 3.15 include

1. 5 of the accidents occurred during PM hours.
2. 4 of the accidents occurred during the day.

In this example, both the static and the dynamic methods were able to identify similar trends.

The second crossing selected from Indiana is 879204S located at the intersection of McGalliard Road and NS rail line in the town of Muncie. This crossing had 15 accidents between 2005 and 2014. Figure 3.16 gives the aerial view of the crossing. The Figure 3.17 shows the static tree generated for the accidents at the crossing while the Figure 3.18 gives the dynamic tree generated using the M+C method. It is to be noted that VEHDIR values of 2 and 3 are considered as eastbound while the others are coded as westbound. TRNDIR values of 1 and 3 are considered as northbound trains while others are considered as southbound.

Figure 3.16: Aerial View of Crossing 879204S
Figure 3.17: Accident Tree Visualization for Accidents at Crossing 879204S using Static Method

From the static tree given in Figure 3.17, it can be seen that the accidents are distributed mostly between those where the vehicles did not stop before the crossing (8 out of 15) and those where vehicles were actually stopped at the crossing when the accident happened (6 out of 15). Among the 6 vehicles stopped on the crossing, 5 motor vehicles were traveling east. Figure 3.18 gives the dynamic tree visualization for the accidents at the crossing.
It can be seen from Figure 3.18 that

1. 14 out of 15 accidents occurred during the PM hours.

2. 13 out of 15 accidents involved a train striking a highway user.

3. 13 out of 15 accidents involved vehicle speeds under 20 mph

4. 8 accidents involved vehicles that did not stop before the crossing while 6 crossings were stopped at the crossing when the accident happened.
Further exploration of the accidents can be carried out by using the M+C method on just a subset of accidents. For example, Figure 3.19 shows the dynamic tree visualization of the 6 accidents that involved vehicles stopped on the crossing.

Figure 3.19: Accident Tree Visualization for Accidents involving Vehicles Stopped at Crossing 879204S using M+C Method

Figure 3.19 shows that

1. All the accidents involved a train striking a highway user.
2. All the accidents involving vehicles stopped on the crossing happened during PM hours.
3. 5 out of the 6 accidents involving the vehicles stopped on the crossing involved an east bound vehicle
4. The vehicle speed was under 20 mph in 5 of the cases while the train speed was between 20 and 40 in 5 of the cases.
Therefore the dynamic tree can be used for further exploration of the accident trends by carefully selecting a subset of accidents to be analyzed.

The next chapter talks about how the M+C method could be used for multiple locations simultaneously. Dynamic tree method of analysis for a corridor and a region are considered for analysis over multiple locations.
In the previous chapter, the M+C method was explained and various examples were used to illustrate usefulness in visualizing accidents occurring at grade crossings. The process, though effective at crossings with a high frequency of accidents ($\geq 5$ accidents at a crossing), is not beneficial for crossings with a low frequency of accidents. Low frequency of accidents at a crossing poses a challenge in identifying accident trends at a location. This is because those accidents could be a result of a random event and thus may not contribute to an accident trend at a crossing.

Analyzing multiple locations simultaneously can overcome this challenge and this procedure helps the analyst to visually observe the data and identify the accident trends over various locations. This is explained in the following sections.

4.1 Corridor Analysis

4.1.1 Corridor I: Metra Rail Rock Island District line, Chicago, IL

The first corridor considered for this analysis is along the Rock Island District line of the Metra Rail in Chicago. Eight crossings along the corridor was crossings selected for this study. Twenty three accidents were observed along this corridor during the period of analysis. Table 4.1 gives the crossing ID and the number of accidents observed at each crossings between 2002 and 2011.
Table 4.1: Number of Accidents in Crossings in Corridor 1

<table>
<thead>
<tr>
<th>Crossing ID</th>
<th>Number of Accidents</th>
</tr>
</thead>
<tbody>
<tr>
<td>608846J</td>
<td>4</td>
</tr>
<tr>
<td>608311K</td>
<td>7</td>
</tr>
<tr>
<td>608310D</td>
<td>2</td>
</tr>
<tr>
<td>608309J</td>
<td>2</td>
</tr>
<tr>
<td>608308C</td>
<td>2</td>
</tr>
<tr>
<td>608304A</td>
<td>2</td>
</tr>
<tr>
<td>609012G</td>
<td>1</td>
</tr>
<tr>
<td>609011A</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 4.1 is an aerial view of the crossing. Figure 4.2 shows the static tree for the corridor and Figure 4.3 shows the dynamic tree generated for the corridor. It is noted that VEHDIR values recorded in the database was interpreted as eastbound if the values were 1 (North) or 3 (east) while the others were considered westbound, and TRNDIR values recorded as 1 (North) or 3 (East) were coded as northbound while the others were considered were southbound.

Figure 4.1: Crossings in Corridor 1
The static tree could identify that there were 15 accidents that involved a gate violation. Much more observations that should be of interest to an analyst can be made from the dynamic tree method. Figure 4.3 shows the dynamic tree and corridor cluster using the M+C method.
The various observations made from Figure 4.3 are listed below.

1. Along the main branch, out of the 17 accidents involving a train striking a highway user, 14 accidents involved a southbound train while only 3 accidents involved a northbound train.

*one accident didn’t have the vehicle speed entry*
2. In 6 instances involving a driver moving over the crossing, the driver drove around the gate.

3. From the crossing cluster it can be seen that 19 out of the 23 accidents (82% of the accidents) involved a train striking a highway user.

4. 15 out of the 23 accidents (65% of the accidents) involved accidents where the drivers drove around the gate.

5. 14 out of the 23 accidents (60% of the accidents) occurred during PM hours.

This information could be valuable at the corridor level and can point out to some issues that may require further investigation at the individual crossing level. Based on these observations, an analyst should be interested in determining the reasons for frequent accidents involving southbound trains and the reason for frequent gate violations by drivers. Corridor analysis becomes important here because analysis at crossings with very few accidents (2 or 3) may not be able to reveal the observations that were made above.

4.1.2 Corridor II: BNSF Rail Line, Chicago, IL

The second corridor analyzed is located in the Chicago area between Union Station and Aurora along a BNSF rail line. This corridor is used both by passenger trains and freight trains. Eight crossings were selected in the corridor as shown in Figure 4.4. Nineteen accidents were recorded at these locations during the analysis period. Table 4.2 gives the crossing ID and the number of accidents observed at each crossings between 2002 and 2011.

<table>
<thead>
<tr>
<th>Crossing ID</th>
<th>Number of Accidents</th>
</tr>
</thead>
<tbody>
<tr>
<td>079508Y</td>
<td>4</td>
</tr>
<tr>
<td>079503P</td>
<td>2</td>
</tr>
<tr>
<td>079501B</td>
<td>1</td>
</tr>
<tr>
<td>079498V</td>
<td>2</td>
</tr>
<tr>
<td>079493L</td>
<td>4</td>
</tr>
<tr>
<td>079491X</td>
<td>2</td>
</tr>
<tr>
<td>079498W</td>
<td>0</td>
</tr>
<tr>
<td>079488P</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 4.4 is an aerial view of the crossing. Figure 4.5 shows the static tree for the corridor and Figure 4.6 shows the dynamic tree and corridor cluster generated for the corridor.
Figure 4.4: Crossings in Corridor 2

Figure 4.5: Accident Tree Visualization of Accidents at Corridor 2 using Static Method

The static tree indicates a high number of pedestrian accidents on the crossings along this corridor. This is because the pedestrian traffic is heavy in this area. The static tree focuses on the accidents involving motor vehicles and no trends could be identified among these accidents from Figure 4.5.
Figure 4.6: Accident Tree Visualization of Accidents at Corridor 2 using M+C algorithm

From the dynamic tree and the corridor cluster as shown in Figure 4.6, we could see that
1. 17 out of the 19 accidents (89% of accidents) involved a train hitting a highway user.
2. 15 out of the 19 accidents (79% of accidents) occurred during daytime.
3. A high number of pedestrian accidents (7 pedestrian accidents out of 19) were also observed. This suggested the analysis of accidents involving a motor vehicles by excluding the accidents involving pedestrians.
Figure 4.7 shows the dynamic tree and corridor cluster for all the motor vehicle accidents along the corridor.

From Figure 4.7, it is further revealed that

1. 7 out of 12 motorist accidents occurred during cloudy days

2. 7 out of 12 motorist accidents involved the motor vehicle moving over the crossing while the accident happened.

An analyst should be interested in the reason for such motorist behavior.
4.2 Regional Analysis

The dynamic tree method could be used on multiple crossings spread across a region. To illustrate this feature of the dynamic tree, comparison of accidents within "similar counties" in Illinois was analyzed using the dynamic tree method. As a first attempt to identify similar counties, regression analysis was carried out between the accident history within the county for 10 years (between 2005–14) and various explanatory variables for the county as listed below.

1. Population
2. Count of Public Crossings
3. Count of Gated Crossings
4. Count of Crossings with Flashing Lights
5. Count of Crossings with Crossbucks
6. Exposure (product of AADT and AATT)
7. Count of crossings with angle <30
8. Count of crossings with angle between 30 and 60
9. Count of crossings with angle >60
10. Count of crossings with HW intersection within 75 feet
11. Count of crossings with HW intersection between 75 feet and 200 feet.
12. Count of crossings with HW intersection between 200 feet and 500 feet.

Since such a large number of variables are considered, Principal Component Analysis was used for dimensionality reduction [20]. The database is normalized to project the original data in the direction with the maximum variance [21]. The first principal component obtained after the dimensionality reduction was able to explain 88% in the variance of the data and hence the regression analysis was carried out between the accident count within the county and the first principal component value within the county. The following regression equation was obtained

\[
\text{Accident Count} = 15.0891 + 18.5155 \times \text{Principal Component 1}
\]

with \( R^2 = 0.9271 \)
It was suggested to use the Principal component values to as a measure to establish similarity between counties.

Even though the regression analysis gave an exceptional goodness of fit, this idea was rejected. This is because similar principal component values doesn’t guarantee that the variables considered during the principal component analysis would have similar values between the two counties. It was observed that none of the pairs of counties in Illinois had all the variables similar within reasonable error even though the principal component values calculated for both counties were identical.

The method chosen to identify similar counties is defined below. We define the conditions for two counties to be considered similar:

1. The difference in population between the two counties is $\leq 5000$
2. The difference in the number of public at-grade crossings is $\leq 6$
3. The difference in the number of public at-grade crossings with gates as warning device is $\leq 3$
4. The difference in the number of public at-grade crossings with flashing lights as warning device is $\leq 3$

These chosen values for the thresholds are approximately 5% of the average of the respective values over all counties in Illinois.

8 pairs of counties were identified in Illinois which satisfy the 4 criterion mentioned above. Out of the 8 pairs of counties, only 2 counties had $\geq 5$ accidents in both the counties. The two pairs are

1. Franklin County and Marion County
2. Bureau County and Effingham County

The dynamic tree was used to compare the accidents within the pair of counties. Franklin county had a population of 39018 as per the 2010 census while Marion county had a population of 41691[22] There are 143 public at-grade crossings, 56 gated crossings and 36 crossings with flashing lights in Franklin county while there are 141 public at-grade crossings, 56 gated crossings and 33 crossings with flashing lights in Marion county. Figure 4.8 shows the dynamic tree and the cluster generated for the two counties.
From the dynamic trees for the two counties few similarities and differences could be observed. The similarities include

1. Both counties have high proportion of accidents during the PM hours.

2. Both counties had most accidents (∼80%) involving low speed vehicles

The main difference between the accidents in the counties is the driver gender of the highway user involved.

1. Franklin county had ∼64% of accidents involving female drivers.

2. Marion county had 70% of accidents involving male drivers

The second pair of counties compared include Bureau County and Effingham County. The population of Bureau county was 35503 in 2010 while the population of Effingham was 34264 in 2010 [22]. There are 148 public at-grade crossings, 55 gated crossings and 20 crossings with flashing lights in Bureau county while there are 143 public at-grade crossings, 58 gated crossings and 17 crossings with flashing lights in Effingham county. The dynamic tree was used to compare the accidents within the pair of counties and are given in Figure 4.9.
Figure 4.9: Accident Tree Visualization of all accidents within Bureau County and Effingham County

From Figure 4.9, similarities and differences between the accidents within the counties could be observed.

1. Both Bureau County and Effingham county has around 50% of its accidents happening on lines classified as FRA Track Class 4.

2. Only 66% of accidents in Bureau county involved vehicles moving while 89% of the accidents in Effingham county involved moving vehicles at the time of the accident.

4.3 Single Accident Locations

Another example of dynamic tree analysis of multiple location is to select the crossings with only one accident in a 10-year time period (2002-2011). The dynamic tree for all single accident locations in Illinois during that time period is given in Figure 4.10.
Figure 4.10: Accident Tree Visualization of all locations with 1 accident in Illinois between 2002 and 2011

The following observations can be made from Figure 4.10:

1. 60 accidents were ped accidents

2. 72% of accidents involve a train striking a highway user

3. 60% of the vehicles were traveling under 20 mph at the time of the accident

In order to identify trends in single accident locations separated based on the warning device at the crossing, three separate analysis were done on crossings with single accidents on locations with crossbucks, flashing lights and gates. The following three figures show the dynamic tree visualization of accidents for single accident locations separated by the warning devices. Figure 4.11 gives the dynamic tree for all single accident locations in Illinois with crossbucks.
The following observations can be made from Figure 4.11:

1. Only 1 accident involved a pedestrian.

2. \( \sim 75\% \) of accidents occurred during day time.

3. A little over two thirds of the accidents involved vehicle speeds under 20 mph.

Figure 4.12 gives the dynamic tree for all single accident locations in Illinois with Flashing Lights. Various observations were made from this figure which are mentioned below.
1. Very few accidents involved pedestrians (This trend was observed in single accident locations with crossbucks as well)
2. About 82% of the accidents involved moving vehicles
3. About two third of the vehicles did not stop before entering the crossing.

Figure 4.12: Accident Tree Visualization of all locations with Flashing Lights with 1 accident in Illinois between 2002 and 2011
Figure 4.13 gives the dynamic tree for all single accident locations in Illinois with Gates.

The following observations can be made from Figure 4.13:

1. Almost all the pedestrian accidents among locations with single accidents occurred at gated crossings.
2. Majority of the accidents (70%) of the accidents involved vehicles with speeds under 20 mph.
3. Almost half the accidents occurred during dark time.
4. Around 76% of the accidents involved a train striking a HW vehicle. This ratio is similar for the locations with crossbucks.

Figure 4.13: Accident Tree Visualization of all locations with Gates with 1 accident in Illinois between 2002 and 2011
In all of the above cases, accident trends could be observed among seemingly unrelated crossings. These could be of interest to a practitioner.

From the various examples illustrating the use of the M+C algorithm on various crossings, it is seen that the algorithm developed has the potential to identify significant contributing factors and accident trends at grade crossings. A practitioner analyzing at-grade crossings can easily detect accident trends which otherwise would require a time-consuming exercise to extract. The use of the M+C algorithm on multiple locations to extract accident trends is also illustrated in this thesis. Overall, this method is expected to be very useful for analysts and practitioners to quickly measure detect accident contributing factors at a grade crossing or a group of crossings for more accurate recommendations for safety improvement.

The next chapter details out the use of the dynamic tree to identify additional factors which may be used in the macro model for accident prediction.
This chapter explains the attempts made to identify new attributes to be used in the accident prediction models. Two different attributes are tried in this chapter which are “HWYNEAR” and “XANGLE”. The values of these variables are available in the inventory database [23]. Accidents in Illinois between 2005 and 2014 are used for the analysis done in this chapter.

5.1 “HWYNEAR”

The “HWYNEAR” attribute available in the inventory database divides the database based on the distance of the crossing to it’s nearest highway intersection. This attribute could have four values namely, highway intersection within 75 feet of the crossing, highway intersection between 75 and 200 feet of the crossing, highway intersection between 200 and 500 feet of the crossing and N/A. The table 5.1 gives the number of accidents at each group of locations analyzed in this section. Note that only public crossings are considered in this analysis.

Table 5.1: Number of Accidents split by “HWYNEAR” and Warning Device

<table>
<thead>
<tr>
<th>HWYNEAR</th>
<th>Xbucks</th>
<th>Flashing Lights</th>
<th>Gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;75</td>
<td>59</td>
<td>62</td>
<td>373</td>
</tr>
<tr>
<td>75-200</td>
<td>0</td>
<td>2</td>
<td>51</td>
</tr>
<tr>
<td>200-500</td>
<td>1</td>
<td>0</td>
<td>27</td>
</tr>
<tr>
<td>N/A</td>
<td>104</td>
<td>91</td>
<td>333</td>
</tr>
</tbody>
</table>

5.1.1 Comparison between Dynamic Trees of Gated Crossings divided by distance to nearby Highway Intersection

Figures 5.1, 5.2 and 5.3 show the dynamic tree visualization for gated crossings with a highway intersection under 75 feet, between 75 and 500 feet and over 500 feet from the crossing respectively.
Figure 5.1: Accidents at Gated Crossings with nearby Highway under 75 feet
Figure 5.2: Accidents at Gated Crossings with nearby Highway between 75 feet and 500 feet
From figures 5.1, 5.2 and 5.3 the following observations could be noted.

1. The variable “MOTORIST” (action of the motorist) doesn’t appear in Figures 5.2 and 5.3 but appears in Figure 5.1 and shows that around 30% of the accidents involved motor vehicles violating the gate.

2. The “TRNSPD” attribute (speed of the train) does not show up in Figure 5.1 but appears in Figures 5.2 and 5.3.

3. The number of accidents involving a train striking the highway user is nearly 75% in Figures 5.2 and 5.3 and 82% in Figure 5.6. The percentage of vehicles moving over the crossing is observed to increase as the distance to the nearby intersecting highway intersection increases.
Therefore, from the three dynamic tree, differences in type of accident, action of motorist, train speeds and position of the highway user at the time of the accident are observed. For this reason, the “HWYNEAR” variable should be explored further for gated crossings to establish it’s significance.

5.1.2 Comparison between Dynamic Trees of Crossings with Flashing Lights divided by distance to nearby Highway Intersection

The second comparison considered in this section is within crossings with flashing lights as a warning device. Figures 5.4 and 5.5 show the dynamic tree visualization for crossings with flashing lights and a highway intersection under 75 feet and over 500 feet from the crossing respectively.

Figure 5.4: Accidents at Crossings with flashing lights with nearby Highway under 75 feet
From figures 5.4 and 5.5 the following observations could be noted.

1. The percentage of moving vehicles (attribute “POSITION”) at the time of accident is around 85% in both the cases.

2. The percentage of vehicles that didn’t stop at the crossing (attribute “MOTORIST”) at the time of the accident have similar values (67% in Figure 5.4 and 62% in Figure 5.5).

3. The proportion of accidents involving train striking the HW vehicle similar in both the cases (61% in Figure 5.4 and 57% in Figure 5.5).
4. The hierarchy of the attributes as determined by the M+C method is almost identical in both cases.

Therefore, from the dynamic tree, no evidence is observed for the significance of HWYN-EAR variable for crossings with flashing lights.

5.1.3 Comparison between Dynamic Trees of Crossings with Crossbucks divided by distance to nearby Highway Intersection

Figures 5.6 and 5.7 show the dynamic tree visualization for crossings with crossbucks and a highway intersection under 75 feet and over 500 feet from the crossing respectively.

Figure 5.6: Accidents at Crossings with crossbucks with nearby Highway under 75 feet
From Figures 5.6 and 5.7 the following observations could be noted.

1. In both the cases the number of accidents involving moving vehicles are very similar (81.3% in Figure 5.6 and 84.6% in Figure 5.7)

2. The number of accidents which involving vehicles that didn’t stop at the crossing are similar in both the cases (77.9% in Figure 5.6 and 75.9% in Figure 5.7)

3. In both the cases the number of highway users involved in accidents with vehicle speeds under 20 mph is similar in both cases (66.34% in Figure 5.6 and 69.69% in Figure 5.7)

Therefore, from the dynamic tree, no evidence is observed for the significance of HWYN EAR variable for crossings with crossbucks.
5.1.4 Comparison Across Warning Device Types

A comparison between the trees across the warning device types was also done. Figures 5.1, 5.4 and 5.6 are used to compare the dynamic tree visualizations for accidents at locations with a highway intersection within 75 feet of the crossing between gated locations, locations with flashing lights and locations with crossbucks.

The following observations about the dynamic trees could be made across the three dynamic trees:

1. The number of highway vehicles moving over the crossing at the time of the accident is the most at crossbuck locations, followed by locations with flashing lights and locations with gates.

2. The number of highway vehicles with speeds under 20 mph are higher in gated locations than crossbuck locations.

3. The number of vehicles that didn’t stop at the crossing at the time of the accident is higher at crossbuck locations than locations with flashing lights.

These observations are intuitive and emphasizes the fact that the warning device type is a significant variable crossings with a highway intersection within 75 feet.

5.2 “XANGLE”

The “XANGLE” attribute available in the inventory database divides the database based smallest angle between the rail line and the highway. This attributes could have 3 values namely, angle < 30 degrees, angle between 30 and 60 degrees and angle > 60 degrees. The table 5.2 gives the number of accidents at each group of locations analyzed in this section.

Table 5.2: Number of Accidents split by “XANGLE” and Warning Device

<table>
<thead>
<tr>
<th>XANGLE</th>
<th>Xbucks</th>
<th>Flashing Lights</th>
<th>Gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;30</td>
<td>6</td>
<td>12</td>
<td>37</td>
</tr>
<tr>
<td>30-60</td>
<td>26</td>
<td>24</td>
<td>132</td>
</tr>
<tr>
<td>&gt;60</td>
<td>132</td>
<td>119</td>
<td>615</td>
</tr>
</tbody>
</table>

Since the accident counts for crossings in the crossing angle under 30 degrees category was limited, analysis was done by combining the crossings with angle under 30 degrees with the crossings with angle between 30 and 60 degrees.
5.2.1 Comparison between Dynamic Trees of Gated Crossings divided by Crossing Angle

Figures 5.8 and 5.9 show the dynamic tree visualization for gated crossings with crossing angle under 60 degrees and over 60 degrees respectively.

Figure 5.8: Accidents at Gated Crossings with crossing angle under 60 degrees
From figures 5.8 and 5.9 the following observations could be noted.

1. The proportion of accidents involving trains striking the highway user is similar in both the cases (83% in Figure 5.8 and 77% in Figure 5.9).

2. The speed of the highway vehicles involved in the accident were under 20 mph in most of the accidents (76% in Figure 5.8 and 72% in Figure 5.9).

3. The “MOTORIST” attribute appears towards the end of the hierarchy in both the figures and this attribute clusters around 30% of the accidents in both the cases into the category “Drove around or thru gate”.

4. The “POSITION” attribute appears in Figure 5.8 but does not appear in Figure 5.9. From the above observations made from the dynamic tree, only very little evidence is observed for the significance of XANGLE variable for gated crossings.
5.2.2 Comparison between Dynamic Trees of crossings with Flashing Lights divided by Crossing Angle

Figures 5.10 and 5.11 show the dynamic tree visualization for crossings with flashing lights and crossing angle under 60 degrees and over 60 degrees respectively.

Figure 5.10: Accidents at Crossings with flashing lights with crossing angle under 60 degrees
From figures 5.10 and 5.11 the following observations could be noted.

1. Around 77% of the accidents at crossings with angle <60 degrees involved the highway user moving while over 87% of the accidents at crossings with angle >60 degrees involved the highway user moving (attribute “POSITION”).

2. Around 53% of the highway users involved in accidents at crossings with angle <60 degrees didn’t stop at the crossing while 68% of the highway users didn’t stop at crossing where the crossing angle was >60 degrees (attribute “MOTORIST”).

3. Around 72% of the accidents at crossings with angle <60 degrees involved a train striking a highway user while only 54% of the accidents at crossings with angle >60 degrees involved a train striking a highway user.
Therefore, from the dynamic tree it can be seen that, even though more accidents involved moving vehicles and vehicles that didn’t stop at the crossing where the angle was over 60 degrees, there were higher number of accidents involving a train striking a highway user. A possible explanation for this could be the visibility issue at crossings with tight angles.

5.2.3 Comparison between Dynamic Trees of crossings with Crossbucks divided by Crossing Angle

Figures 5.12 and 5.13 show the dynamic tree visualization for crossings with flashing lights and crossing angle under 60 degrees and over 60 degrees respectively.

Figure 5.12: Accidents at Crossings with crossbucks with angle under 60 degrees
From figures 5.12 and 5.13 the following observations could be noted.

1. Around 78% of the accidents at crossings with angle <60 degrees involved a train striking a highway user while only 71% of the accidents at crossings with angle > 60 degrees involved a train striking a highway user.

2. The attributes “POSITION” and “MOTORIST” appear much lower in the hierarchy in Figure 5.13 but appears earlier in the hierarchy in Figure 5.12.

From the dynamic tree it can be seen that, the accidents involving a train striking a highway vehicle has a greater relative frequency for crossings with a tighter angle than crossings with angles above 60 degrees. It is also observed that the attributes representing the action of the motorist and position of the highway user at the time of the accident were more important attributes at crossing locations with tighter angles than at locations with angles over 60 degrees.
Though this exercise cannot establish a significance of a contributing factor towards accidents, it definitely suggests that the distance to the nearby highway intersection (for gated crossings) and angle of the crossing (for crossings with flashing lights or crossbucks) is to be explored as an additional variable in the accident prediction formula.

The next chapter summarizes the thesis. Various suggestions to improve the database to enable further information extraction are also mentioned.
CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

Identifying the order of the contributing factors of the accidents and extracting useful information from it is a time consuming process and manual analysis of individual crossings or groups of crossings is not feasible. The dynamic tree is very useful in extracting useful information from such a database and the tree based visualization method enables a practitioner to quickly identify significant accident contributing factors and any trends in the accidents analyzed.

Currently, the method uses attributes that are directly available in the accident database. Additional information regarding the accidents could be extracted using an improved database. Various suggestions to improve the database are given below.

6.1 Recommendations for Database Improvement

The following are some of the suggestions to improve the accident database to be used for the algorithm

1. Use of Hybrid Attributes: Various new attributes could be generated using the existing attributes of the database. For example, the "Day of the week" information does not exist in the database but it could be obtained by combining the day (attribute name: DAY), month (attribute name: MONTH) and year (attribute name: YEAR) information along with some calculations. This new variable could be used to distinguish between weekday accidents vs weekend accidents. Another example is "Time of the day" which could be generated using the hour (attribute name: HOUR), minute (attribute name: MINUTE) and AM/PM information (attribute name: AMPM). This could be useful to detect accident contributing factors like AM-peak or PM-peak.

2. Use of Additional Databases: The FRA accident database could be combined with additional databases. For example, the FRA inventory database [23] has details like angle of crossing (attribute name: XANGLE), type of highway (attribute name: HWYSYS) which could be useful attributes in determining the accident contributing factors.
3. Use of Population Information: The dynamic tree method does not assume a binary nature for the data. The dynamic tree classifies the accidents into as many subcategories as the data provides and all of them are used to build on the hierarchy. Population information like demographics of the people surrounding the area, directional distribution of trains and highway users etc are very useful information when an analyst explores the reason for any data skew that appears on the dynamic tree.

The dynamic tree method is a significant improvement over the static method which used a fixed and pre-determined hierarchy of the accident attributes for all the crossings. A more comprehensive set of variables was used in the dynamic method as compared to the static method. Furthermore, a data driven approach utilized in this method makes identification of the order of the contributing factors of the accidents at grade crossings easier for an analyst. A computer program implementing the M+C method was also written which significantly reduces the analysis time and reduces human effort required in the analysis. To demonstrate the application of the dynamic tree method to extract useful information from an accident database, various examples are also illustrated in the thesis. Examples showing single location, multiple locations including crossings along a corridor, crossings within counties and single accident locations were analyzed and information was easily extracted from the database.

This method was validated over various databases spread over different time frames. It has been shown that this method is not just restricted to the database used to develop it (i.e. database 6180.57 for the state of Illinois between 2002 and 2011). Validation using accident database for the state of Illinois between 2005 and 2014, accident database for the state of California between 2005 and 2014 as well as for the state of Indiana between 2005 and 2014 are used for this purpose and are reported in this thesis.

The dynamic tree was also used to explore additional variables to be used in macro models. Two variables, namely “HWYNEAR” and “XANGLE” representing “presence of adjacent highway intersection” and the “smallest crossing angle” were tried. The dynamic tree method using the “HWYNEAR” attribute was able to identify evidence for further exploration for the attribute for gated crossings. It was also observed that for warning device type Flashing Lights and Crossbucks, those crossings with angle below 60 degrees tend to have a higher relative frequency for accidents involving train striking the highway. Based on these observations, the attributes “HWYNEAR” and “XANGLE” should be explored for significance in macro analysis of accidents to improve the accident prediction models.

Safety at grade crossings relevant today with the increasing congestion on both the highway and railroad systems. The proposed computerized method, combined with a user friendly
interface, can be a very useful tool that practitioners can use for quick and easy analysis of accidents at grade crossings.
REFERENCES


#include<iostream>
#include<iomanip>
#include<fstream>
#include<string>
#include<string.h>
#include<sstream>

struct Tree
{
    int data;
    int parent;
    int arr[1000];
    int node;
    int print;
    int outlier;
    Tree()
    {
        data = 0;
        parent = 0;
        node = 0;
        outlier = 0;
        print = 0;
        for (int i = 0; i < 10; i++)
            arr[i] = -1;
    }
    ~Tree(){
}
Tree t[23][950];
Tree b[23][950];

using namespace std;

string titles[22][11] = {
    {"1","2","3","4","","","","","",""},
    {"AM","PM","","","","","","","",""},
    {"<20","20-40","40-60",">60","","","","","",""},
    {"Ped","Motor","Others","","","","","","",""},
    {"North","South","East","West","","","","","",""},
    {"Stalled","Stopped","Moving","Trapped","","","","","",""},
    {"Rail->HW","HW->Rail","","","","","","","",""},
    {"Dawn","Day","Dusk","Dark","","","","","",""},
    {"Clear","Cloudy","Rain","Snow","Sleet","Snow","","","",""},
    {"Main","Yard","Siding","Industry","","","","","",""},
    {"X","1","2","3","4","5","6","7","8","9"},
    {"<20","20-40","40-60",">60","","","","","",""},
    {"North","South","East","West","","","","","",""},
    {"Both","Same.Side","Other.Side","","","","","","",""},
    {"Yes","No","Unknown","","","","","","",""},
    {"Yes","No","Unknown","","","","","","",""},
    {"Around.Gate","Stop.&.Proceed","Didn’t.Stop","Stop.on.Xing"},
    {"Perm.Str","RR.Equip","Passing.Train","Topography","Vegetation","HW.Vehicle","Other","Unobstructed","",""},
    {"No","Yes","","","","","","","",""},
    {"Public","Private","","","","","","","",""},
    {"<30","30-60",">60","","","","","","",""},
    {"Male","Female","","","","","","","",""}
};

string attribute_names[22] = {
    {"","","","","","","","","",""},
"MONTH",
"AMPM",
"VEHSPD",
"TYPVEH",
"VEHDIR",
"POSITION",
"TYPACC",
"VISIBILITY",
"WEATHER",
"TYPTRK",
"TRKCLAS",
"TRNSPD",
"TRNDIR",
"LOCWARN",
"WARNSIG",
"LIGHTS",
"MOTORIST",
"VIEW",
"CROSSING",
"PUBLIC",
"DRIVAGE",
"DRIVGEN"

int printingfunction (int data, int attribute, int leaf, int space, int highest[], int priority[], ofstream& treeinprint)
{
    stringstream sstm;
    string stringtoprint;
    sstm<"titles[priority[attribute]][leaf]<"\n("<<data<<")"
    stringtoprint = sstm.str();
    cout<"setw(space);
    cout<"stringtoprint;
    treeinprint<"setw(space);
    treeinprint<"stringtoprint;
    if (data == highest[priority[attribute]])
{  
    return 1;
}

return 0;

int main()
{
    const int attributes = 22;
    const int num_accidents = 950;
    const int max_divisions = 11;
    const int max_locations = 15;
    int largest_modifier[attributes]= {0};
    float score[attributes] = {0};
    string xno = "ABCD";
    double long_number[max_locations][5] = {0};
    int a[num_accidents][attributes][max_divisions] = {0};
    int temp1[attributes][max_divisions] = {0};
    int temp = 0;
    int max_subcat[attributes] = {0};
    int number_counter = 0;
    int new_test = 0;
    int test1 = 0;
    int tie = -1;
    float expectation[attributes][max_divisions] = {0};
    int std_arr[attributes] =
        {4,2,3,3,4,4,2,4,6,4,10,4,4,3,3,5,8,2,2,3,2};
    int count_all[attributes][max_divisions][2] = {0};
    int largest_first_level[attributes] = {0};
    int largest_leaf[attributes] = {0};
    int highest[attributes] = {0};
    string line;
    string dummy;
    ifstream myfile;
    ifstream locationfile;
    ifstream expectationfile;
    ofstream treeinprint;
// expectationfile.open("Expectations_Historic.csv");
// int expectation_counter = 0;
// if (!expectationfile.is_open())
// {
// cout<<"Expectations file not found"<<endl;
// cin.get();
// return 0;
// }
// while (!expectationfile.eof())
// {
// getline(expectationfile,line);
// if (line.compare("") != 0)
// {
// istringstream ss (line);
//  int j = 0;
//  while (getline(ss,dummy,','))
//  {
//   float exp;
//   stringstream(dummy) >> exp;
//   expectation[expectation_counter][j] = exp;
//   j++;
//  }
//  }
//  expectation_counter++;
// }
// expectationfile.close();

int no_of_lines = 0;
int accCount = 0;
// cout<<"Enter file name"<<endl;
string filename;
string location[950];

filename = "Illinois Compiled Database 2002-11";
cout<<"Reading from file "<filename<<endl;
// std::getline(cin, filename);
// myfile.open("Illinois_Compiled_Accident.csv"); //Not relevant now.

myfile.open(filename + "_.csv");
if (!myfile.is_open())
{
    cout << "File not found" << endl;
    cin.get();
    cin.get();
    return 0;
}
while (!myfile.eof())
{
    getline(myfile, line);
    no_of_lines = no_of_lines + 1;
    if (no_of_lines == 1)
    {
        continue;
    }

    int counter = 0;
    istringstream ss(line);
    while (getline(ss, dummy, ','))
    {
        counter++;
        if (counter == 16)
        {
            if (dummy.compare(xno) == 0)
            {
                accCount = accCount + 1;
                xno = dummy;
            }
            else
            {
            }
        }
    }
}

if (accCount > 0)
for (int i = 0; i < accCount; 
    i++) // to check all 
    accidents 
{
    for (int j = 0; j < 
        attributes; j++) // 
        to check all 
        conditions 
    {
        for (int k = 0; 
            k < std_arr[j ]; k++) // 
                division in 
                condition 
    {
        if (a[i ][j][k] 
            == 1)
        {
            temp1 
            [ 
                j 
            ][ 
                k 
            ]
        
            =
        
            temp1 
            [ 
                j 
            ][ 
                k 
            ]+1;
for (int j = 0; j < attributes; j++)
{
    for (int k = 0; k < std_arr[j]; k++)
    {
        if (largest_modifier[j] < temp1[j][k])
        {
            largest_modifier[j] = temp1[j][k];
        }
    }
}
for (int j = 0; j < attributes; j++)
{
    // score[j] = score[j] + (10-(accCount-largest_modifier[j]+1));
    // cout<<"here"<<endl;
    score[j] = score[j] + accCount*
largest_modifier[j];
    //float(accCount);
    // cout<<j" | "<<score[j]" | "<<largest_modifier[j]" | "<<accCount" | "<<xno"<<endl;
    // cin.get();
}
for (int j = 0; j < attributes; j++) {
    for (int k = 0; k < max_divisions; k++) {
        temp1[j][k] = 0;
    }
    largest_modifier[j] = 0;
}

xno = dummy;
for (int i = 0; i < num_accidents; i++) {
    for (int j = 0; j < attributes; j++) {
        for (int k = 0; k < std_arr[j]; k++) {
            a[i][j][k] = 0;
        }
    }
    accCount = 1;
    continue;
}

if (counter == 18) //Month
{

    if (!dummy.empty())
    {
        int month;

        // cout<<dummy<<endl;
        // cin.get();

stringstream(dummy) >> month;
temp = (month-1)/3;

// cout<<month" | "<<temp<<endl;
// cin.get();

a[accCount-1][0][temp] = 1;
count_all[0][temp][0]++;
}
continue;
}
if (counter == 22) //AMPM
{
    if (!dummy.empty())
    {
        if (!dummy.compare("AM"))
        {
            a[accCount-1][1][0] = 1;
            count_all[1][0][0]++;
        }
        else
        {
            a[accCount-1][1][1] = 1;
            count_all[1][1][0]++;
        }
    }
    continue;
}
if (counter == 30) //Vehicle Speed
{
    if (!dummy.empty())
    {
        int vehspd;
        stringstream(dummy) >> vehspd;
        temp = (vehspd)/20;
        a[accCount-1][2][temp] = 1;
count_all[2][temp][0]++;
}
continue;
}
if (counter == 31) // Type of highway user
{
    if (!dummy.empty())
    {
        if (!dummy.compare("K")
        {
            a[accCount-1][3][0] = 1;
            count_all[3][0][0]++;
        }
        else if (!dummy.compare("A")
            || !dummy.compare("B")
            || !dummy.compare("C")
            || !dummy.compare("D")
            || !dummy.compare("E")
            || !dummy.compare("F")
            || !dummy.compare("G")
            || !dummy.compare("H")
            || !dummy.compare("I")
            || !dummy.compare("J")
        {
            a[accCount-1][3][1] = 1;
            count_all[3][1][0]++;
        }
        else
        {
            a[accCount-1][3][2] = 1;
            count_all[3][2][0]++;
        }
    }
}
continue;
}
if (counter == 32) //Highway user direction
{
    if (!dummy.empty())
    {
        int vehdir;
        stringstream(dummy) >> vehdir;
        a[accCount-1][4][vehdir-1] = 1;
        count_all[4][vehdir-1][0]++;
    }
    continue;
}
if (counter == 33) // Position of vehicle
{
    if (!dummy.empty())
    {
        int position;
        stringstream(dummy) >> position;
        a[accCount-1][5][position-1] = 1;
        count_all[5][position-1][0]++;
    }
    continue;
}
if (counter == 36) // Type of Accident
{
    if (!dummy.empty())
    {
        int typacc;
        stringstream(dummy) >> typacc;
        a[accCount-1][6][typacc-1] = 1;
        count_all[6][typacc-1][0]++;
    }
}
} 
continue;
} 
if (counter == 39) //Visibility 
{
    if (!dummy.empty())
    {
        int visiblty;
        stringstream(dummy) >> visiblty;
        a[accCount-1][7][visiblty-1] = 1;
        count_all[7][visiblty-1][0]++;
    }
    continue;
} 
if (counter == 40) //Weather 
{
    if (!dummy.empty())
    {
        int weather;
        stringstream(dummy) >> weather ;
        a[accCount-1][8][weather-1] = 1;
        count_all[8][weather-1][0]++;
    }
    continue;
} 
if (counter == 42) //Type of track 
{
    if (!dummy.empty())
    {
        int typtrk;
        stringstream(dummy) >> typtrk;

a[accCount-1][9][typtrk-1] = 1;
count_all[9][typtrk-1][0]++;
}
continue;
}
if (counter == 44) //Track Class
{
  if (!dummy.empty())
  {
    if (!dummy.compare("X"))
    {
      a[accCount-1][10][0] = 1;
count_all[10][0][0]++;
      continue;
    }
    int trkclas;
    stringstream(dummy) >> trkclas;
a[accCount-1][10][trkclas] = 1;
count_all[10][trkclas][0]++;
  }
  continue;
}
if (counter == 47) //Train Speed
{
  if (!dummy.empty())
  {
    int trnspd;
    stringstream(dummy) >> trnspd;
temp = (trnspd)/20;
a[accCount-1][11][temp] = 1;
count_all[11][temp][0]++;
    if (trnspd > 60)
{ 
a[accCount-1][11][3] = 1;
count_all[11][3][0]++;
}
}
continue;
}
if (counter == 49) //Train Direction
{
    if (!dummy.empty())
    {
        int trndir;
        stringstream(dummy) >> trndir;
        a[accCount-1][12][trndir-1] = 1;
count_all[12][trndir-1][0]++;
    }
continue;
}
if (counter == 51) //Location of warning device
{
    if (!dummy.empty())
    {
        int locwarn;
        stringstream(dummy) >> locwarn;
        a[accCount-1][13][locwarn-1] = 1;
count_all[13][locwarn-1][0]++;
    }
continue;
}
if (counter == 52) //Warning sign connected to highway signs

{ 
    if (!dummy.empty())
    {
        int warnsign;
        stringstream(dummy) >> 
            warnsign;
        a[accCount-1][14][warnsign-1] = 1;
        count_all[14][warnsign -1][0]++;
    }
    continue;
}
if (counter == 53) //lights at crossing
{
    if (!dummy.empty())
    {
        int lights;
        stringstream(dummy) >> lights;
        a[accCount-1][15][lights-1] = 1;
        count_all[15][lights-1][0]++;
    }
    continue;
}
if (counter == 56) // Action of motorist
{
    if (!dummy.empty())
    {
        int motorist;
        stringstream(dummy) >> 
            motorist;
        a[accCount-1][16][motorist-1] = 1;
        count_all[16][motorist -1][0]++;
    }
}
if (counter == 57) //view
{
    if (!dummy.empty())
    {
        int view;
        stringstream(dummy) >> view;
        a[accCount-1][17][view-1] = 1;
        count_all[17][view-1][0]++;
    }
    continue;
}
if (counter == 70) //warning devices
{
    if (!dummy.empty())
    {
        if (!dummy.compare("12"))
        {
            a[accCount-1][18][0] = 1;
            count_all[18][0][0]++;
        }
        else
        {
            a[accCount-1][18][1] = 1;
            count_all[18][1][0]++;
        }
    }
    continue;
}
continue;
if (counter == 75) // public or private
{
    if (!dummy.empty())
    {
        if (!dummy.compare("Y"))
        {
            a[accCount-1][19][0] = 1;
            count_all[19][0][0]++;
        }
        else
        {
            a[accCount-1][19][1] = 1;
            count_all[19][0][0]++;
        }
    }
    continue;
}
if (counter == 84) // driver age
{
    if (!dummy.empty())
    {
        int age;
        stringstream(dummy) >> age;
        temp = age/30;
        a[accCount-1][20][temp] = 1;
        count_all[20][temp][0]++;
    }
    continue;
}

if (counter == 85)
{
    if (!dummy.empty())
    {
        int gen;
        stringstream(dummy) >> gen;
        a[accCount-1][21][gen-1] = 1;
        count_all[21][gen-1][0]++;
    }
    continue;
}

// cout<<accCount<<endl;
// cin.get();

for (int i = 0; i < attributes; i++)
    for (int j = 0; j < max_divisions; j++)
    {
        count_all[i][j][0] = 0;
    }

int sorted_index[attributes];
int tie_break[attributes] = {0};
for (int i = 0; i < 22; i++)
{
    sorted_index[i] = i;
}
int swap, swap2 = 0;
for (int i = 0 ; i < attributes; i++)
{
    for (int j = 0; j < attributes; j++)
    {
        if (score[i] > score[j])
        {
        

swap = score[i];
score[i] = score[j];
score[j] = swap;
swap2 = sorted_index[i];
sorted_index[i] = sorted_index[j];
sorted_index[j] = swap2;
}
}
}
for (int i = 0; i < attributes; i++)
{
    tie_break[sorted_index[i]] = i;
}
for (int i = 0; i < attributes; i++)
{
    cout<<i+1<"\t\t"<<sorted_index[i]+1<"\t\t"<<score[i]<<endl;
}

//FINAL METHOD B BEGINS HERE

cout<<"Reading␣from␣Corridor.txt␣file"<<endl;
locationfile.open("Crossings/IL␣Crossing.txt");
int location_counter = 0;
while (!locationfile.eof())
{
    getline(locationfile,xno);
    if (xno.compare("") != 0)
    {
        location[location_counter] = xno; //you can go up
to a max of 3000. Change index if you want more.
        location_counter++;
    }
    else
    {
        continue;
    }
}
locationfile.close();

int priority[attributes] = {0};
int selected[attributes] = {0};
int store_in_z = 0, z = 0, m = 0, j = 0, counter = 0, test = 0,
    large = 0, largest = 0, num_levels = 0;
int var = 0; // to check if we reach a breakdown to 1 for the
    first time. This is relevent in prioritizing
int q = 0, r = 0, w = 0, i = 0, p = 0; // just counters for loops

myfile.clear();
myfile.seekg(0,myfile.beg); // this prevents repeted
    opening and closing of the same file

for (q = 0; q < num_accidents; q++)
{
    for (r = 0; r < attributes; r++)
    {
        for (w = 0; w < max_divisions; w++)
        {
            a[q][r][w] = 0; // reinitializing the
                variables
        }
    }
}

accCount = 0;

// where the file is read and a is filled.

while (!myfile.eof())
{
    getline(myfile,line);
    int counter = 0;
    istringstream ss(line);
    while (getline(ss,dummy,',',','))
    {

counter++;  
if (counter == 16)  
{
    int test = 0;
    for (int l = 0; l < location_counter;  
l++)  
    {
        xno = location[l];  
        if (dummy.compare(xno) != 0) //
            if dummy not in Corridor.txt  
                file, then there is no need  
                to read the rest  
        {
            // basically do nothing  
        }  
        else  
        {
            // cout<<xno<<" | "<<accCount+1<<endl;
            test = 1;  
            // cin.get();  
            test = 1;
            break;
        }
    }
    if (test == 1)  
    {
        accCount++;
    }
    else  
    {
        break;
    }
}
if (counter == 18) // Month  
{
    if (!dummy.empty())  
    {
        //
    98
int month;
stringstream(dummy) >> month;
temp = (month-1)/3;
a[accCount-1][0][temp] = 1;
count_all[0][temp][0]++;
}
continue;
}
if (counter == 22) //AMPM
{
    if (!dummy.empty())
    {
        if (!dummy.compare("AM"))
        {
            a[accCount-1][1][0] = 1;
            count_all[1][0][0]++;
        }
        else
        {
            a[accCount-1][1][1] = 1;
            count_all[1][1][0]++;
        }
    }
    continue;
}
if (counter == 30) //Vehicle Speed
{
    if (!dummy.empty())
    {
        int vehspd;
        stringstream(dummy) >> vehspd;
temp = (vehspd)/20;
a[accCount-1][2][temp] = 1;
count_all[2][temp][0]++;
}
if (counter == 31) //Type of highway user
{
  if (!dummy.empty())
  {
    if (!dummy.compare("K"))
    {
      a[accCount-1][3][0] = 1;
      count_all[3][0][0]++;
    }
  
    else if (!dummy.compare("A") || !dummy.compare("B") || !dummy.compare("C") || !dummy.compare("D") || !dummy.compare("E") || !dummy.compare("F") || !dummy.compare("G") || !dummy.compare("H") || !dummy.compare("I") || !dummy.compare("J"))
    {
      a[accCount-1][3][1] = 1;
      count_all[3][1][0]++;
    }

    else
    {
      a[accCount-1][3][2] = 1;
      count_all[3][2][0]++;
    }
  
  }

  continue;
}

100
if (counter == 32) //Highway user direction
{
    if (!dummy.empty())
    {
        int vehdir;
        stringstream(dummy) >> vehdir;
        // a[accCount-1][4][vehdir-1] = 1;
        // count_all[4][vehdir-1][0]++;
        /* if (vehdir == 1)
        {
            a[accCount-1][4][2] =
            1;
            count_all[4][2][0]++;
        }
        else
        {
            a[accCount-1][4][3] =
            1;
            count_all[4][3][0]++;
        }*/
        if (vehdir == 2 || vehdir ==
            4)
        {
            a[accCount-1][4][1] =
            1;
            count_all[4][1][0]++;
        }
        else if (vehdir == 1 || vehdir
            == 3)
        {
            a[accCount-1][4][0] =
            1;
            count_all[4][0][0]++;
        }
    }
}
if (counter == 33) // Position of vehicle
{
    if (!dummy.empty())
    {
        int position;
        stringstream(dummy) >> position;
        a[accCount-1][5][position-1] = 1;
        count_all[5][position-1][0]++;
    }
    continue;
}
if (counter == 36) // Type of Accident
{
    if (!dummy.empty())
    {
        int typacc;
        stringstream(dummy) >> typacc;
        a[accCount-1][6][typacc-1] = 1;
        count_all[6][typacc-1][0]++;
    }
    continue;
}
if (counter == 39) // Visibility
{
    if (!dummy.empty())
    {
        int visiblty;
        stringstream(dummy) >> visiblty;
        a[accCount-1][7][visiblty-1] = 1;
    }
}
count_all[7][visiblty-1][0]++;
}
continue;
}
if (counter == 40) //Weather
{
    if (!dummy.empty())
    {
        int weather;
        stringstream(dummy) >> weather ;
        a[accCount-1][8][weather-1] = 1;
        count_all[8][weather-1][0]++;
    }
    continue;
}
if (counter == 42) //Type of track
{
    if (!dummy.empty())
    {
        int typtrk;
        stringstream(dummy) >> typtrk;
        a[accCount-1][9][typtrk-1] = 1;
        count_all[9][typtrk-1][0]++;
    }
    continue;
}
if (counter == 44) //Track Class
{
    if (!dummy.empty())
    {
        if (!dummy.compare("X"))
        {
            
103
a[accCount-1][10][0] = 1;
count_all[10][0][0]++;
continue;
}
int trkclas;
stringstream(dummy) >> trkclas;
a[accCount-1][10][trkclas] = 1;
count_all[10][trkclas][0]++;
continue;
}

if (counter == 47) //Train Speed
{
    if (!dummy.empty())
    {
        int trnspd;
        stringstream(dummy) >> trnspd;
temp = (trnspd)/20;
        if (trnspd < 60)
        {
            a[accCount-1][11][temp] = 1;
count_all[11][temp][0]++;
        }
        else
        {
            a[accCount-1][11][3] = 1;
count_all[11][3][0]++;
        }
    }
    continue;
}
if (counter == 49) //Train Direction
{
    if (!dummy.empty())
    {
        int trndir;
        stringstream(dummy) >> trndir;
        // a[accCount-1][12][trndir-1] = 1;
        // count_all[12][trndir-1][0]++;
        /* if (trndir-1 == 2)
        {
            a[accCount-1][12][0] = 1;
            count_all[12][0][0]++;
        }
        else if (trndir-1 == 3)
        {
            a[accCount-1][12][1] = 1;
            count_all[12][1][0]++;
        } */
        if (trndir == 2 || trndir == 3)
        {
            a[accCount-1][12][2] = 1;
            count_all[12][2][0]++;
        }
        else if (trndir == 1 || trndir == 4)
        {
            a[accCount-1][12][3] = 1;
            count_all[12][3][0]++;
        }
    }
}
continue;

}  
if (counter == 51) //Location of warning device 
{
    if (!dummy.empty())
    {
        int locwarn;
        stringstream(dummy) >> locwarn;
        a[accCount-1][13][locwarn-1] = 1;
        count_all[13][locwarn-1][0]++;
    }
    continue;
}
if (counter == 52) // Warning sign connected to highway signs
{
    if (!dummy.empty())
    {
        int warnsign;
        stringstream(dummy) >> warnsign;
        a[accCount-1][14][warssign-1] = 1;
        count_all[14][warssign-1][0]++;
    }
    continue;
}
if (counter == 53) //lights at crossing
{
    if (!dummy.empty())
    {
        int lights;
        a[accCount-1][15][lights-1] = 1;
        count_all[15][lights-1][0]++;
    }
    continue;
}
stringstream(dummy) >> lights;
a[accCount-1][15][lights-1] = 1;
count_all[15][lights-1][0]++;
}
    continue;
}
if (counter == 56) // Action of motorist
{
    if (!dummy.empty())
    {
        int motorist;
        stringstream(dummy) >>
            motorist;
        a[accCount-1][16][motorist-1] = 1;
        count_all[16][motorist -1][0]++;
    }
    continue;
}
if (counter == 57) // view
{
    if (!dummy.empty())
    {
        int view;
        stringstream(dummy) >> view;
        a[accCount-1][17][view-1] = 1;
        count_all[17][view-1][0]++;
    }
    continue;
}
if (counter == 70) // warning devices
{
    if (!dummy.empty())
    {
        
107
if (!dummy.compare("12"))
{
    a[accCount-1][18][0] = 1;
    count_all[18][0][0]++;
}
else
{
    a[accCount-1][18][1] = 1;
    count_all[18][1][0]++;
}
}
continue;
}
if (counter == 75) // public or private
{
    if (!dummy.empty())
    {
        if (!dummy.compare("Y"))
        {
            a[accCount-1][19][0] = 1;
            count_all[19][0][0]++;
        }
        else
        {
            a[accCount-1][19][1] = 1;
            count_all[19][1][0]++;
        }
    }
    continue;
}
if (counter == 84) // driver age
{

if (!dummy.empty())
{
    int age;
    stringstream(dummy) >> age;
    temp = age/30;
    a[accCount-1][20][temp] = 1;
    count_all[20][temp][0]++;
    
    continue;
}

if (counter == 85)
{
    if (!dummy.empty())
    {
        int gen;
        stringstream(dummy) >> gen;
        a[accCount-1][21][gen-1] = 1;
        count_all[21][gen-1][0]++;
        
        continue;
    }
}

for (int i = 0; i < attributes; i++)
{
    for (int j = 0; j < std_arr[i]; j++)
    {
        if (count_all[i][j][0] > largest_first_level[i])
        {
            largest_first_level[i] = count_all[i][j][0];
            largest_leaf[i] = j;
        }
    }
}
cout<<accCount<<endl;

//prioritizing

for (p = 0; p < attributes; p++)
{
    priority[p] = 0;  //reinitializing the variables
    selected[p] = 0;
}

z = 0;
m = 0;
j = 0;
counter = 0;
test = 0;
large = 0;
largest = 0;
num_levels = 0;
var = 0;  //to check if we reach a breakdown to 1 for the
          //first time (not relevant in this code)
t[0][0].data = accCount;
t[0][0].node = 0;
for (i = 0; i < accCount; i++)  //reinitializing the
    //structure variable
{
    t[0][0].arr[i] = i;
}

for (i = 1; i < attributes; i++)
{
    for (p = 0; p < num_accidents; p++)
    {
        t[i][p].data = 0;  //reinitializing the
                          //structure variables
        for (r = 0; r < num_accidents; r++)
        {
            t[i][p].arr[r] = 0;
        }
    }
}
for (int i = 0; i < attributes; i++)
{
    tie = -1;
    store_in_z = 0;
    if (i == attributes-1) //will not reach here in
        this code for sure.
    {
        for (z = 0; z < attributes; z++)
        {
            if (selected[z] == 0)
            {
                priority[i] = z;
                selected[z] = 1;
                break;
            }
        }
    }
    for (m = accCount-1; m > 0; m--) //Finding the
        value of m (which corresponds to the index having
        0 in next level i+1
    {
        if (t[i+1][m].data != 0)
        {
            m++;
            break;
        }
    }
    largest = 0;

    for (j = 0; j <= attributes; j++) //look at all
        unselected attributes to fill out the next level
    {
        for (int x = 0; x < m; x++) //reinitializing
            the values
        {
            t[i+1][x].data = 0;
        }
    }
t[i+1][x].node = 0;
for (int y = 0; y < num_accidents; y ++)
    t[i+1][x].arr[y] = 0;
}
m = 0;
test = 0;
if (selected[j] == 1)
{
    continue;
}
if (j == attributes)
{
    j = priority[i];
    selected[j] = 1;
    test = 1;
}
large = 0;
for (int n = 0; n < std_arr[j]; n++)
{
    for (int k = 0; k < accCount; k++)
    {
        for (int l = 0; l < accCount;
            l++)
        {
            if (t[i][z].arr[l] == k 
            )
            {
                if (a[k][j][n]
                    == 1)
                {
                    t[i+1][m]
                        .data
    = t[i
        +1][m].

data+1;
t[i+1][m].arr[
counter] = k;
counter++;
}
break;
}
}
}
if (large < t[i+1][m].data)
{
large = t[i+1][m].data;
store_in_z = m;
max_subcat[i] = n;
}

if (t[i+1][m].data != 0)
{
m++;
}
counter = 0;

}
if (largest < large) //setting up the
    priority based on the largest number
{
priority[i] = j;
largest = large;
tie = tie_break[j];
}
else if (large == largest)
{


if (tie_break[j] < tie) //using the priority that we built earlier.
{
    priority[i] = j;
    tie = tie_break[j];
}
}

if (test == 1)
{
    z = store_in_z; //since we would be splitting the largest value in the next level
    break;
}
}
} // end of outermost loop while prioritizing.
for (int i = 0; i < attributes; i++)
{
    cout<<priority[i]+1<<"|"<<largest_first_level[priority[i]]<<"|"<<largest_leaf[priority[i]]<<endl;
}
//Building the tree

cout<<"starts here"<<endl;
for (int i = 0; i < attributes; i++)
{
    for (int j = 0; j < std_arr[priority[i]]; j++)
    {
        if (count_all[priority[i]][j][0] == largest_first_level[priority[i]])
        {
            count_all[priority[i]][j][1] = 2;
        }
        if(100*((100*count_all[priority[i]][j][0]/accCount) - 100*expectation[priority[i]][j])/(100*
expectation[priority[i][j]] > 50)
{
    if (count_all[priority[i][j][0] >= 4)
    {
        cout << i+1 << priority[i]+1 << count_all[priority[i][j][0] << count_all[priority[i][j][0] << "FLAG" << endl;
        count_all[priority[i][j][1] = 1;
    }
}
// cout<<endl;
}
// reinitializing the variables
for (i = 0; i < attributes; i++)
{
    for (j = 0; j < accCount; j++)
    {
        b[i][j].data = 0;
        for (int k = 0; k < num_accidents; k++)
        {
            b[i][j].arr[k] = -1;
        }
    }
}

counter = 0;
b[0][0].data = accCount;
b[0][0].node = 0;
for (int i = 0; i < accCount; i++)
{
    b[0][0].arr[i] = i;
}

for (int i = 0; i < attributes; i++)
{
for (int j = 0; j < accCount; j++)
{
    if (b[i][j].data == 0)
    {
        break;
    }
    for (int m = accCount-1; m > 0; m--)
    {
        if (b[i+1][m].data != 0)
        {
            m++; break;
        }
    }
    if (j == 0)
    {
        m = 0;
    }
}

for (int n = 0; n < std_arr[priority[i]]; n++)
{
    for (int k = 0; k < accCount; k++)
    {
        for (int l = 0; l < accCount; l++)
        {
            if (k == b[i][j].arr[l])
            {
                if (a[k][priority[i]][n] == 1)
                {
                    
                }
b[i+1][m].data = b[i+1][m].data+1;
b[i+1][m].parent = j;
b[i+1][m].arr[counter] = k;
counter++;
}
}
}
}
}
}

b[i+1][m].node = n;

if (b[i+1][m].data == 0)
{
    continue;
}
}

if(100*((100*b[i+1][m].data/b[i][j].data) - 100*expectation[priority[i][n]]/(100*expectation[priority[i][n]])>50)
{
    b[i+1][m].outlier = 1;
}
}
else
{
    b[i+1][m].outlier = 0;
}

117
if (b[i+1][m].data != 0)
{
    m = m+1;
}
counter = 0;
}
}

//Printing the tree
cout<<"Tree for location at given crossing"<<endl;
cout<<"Level 0"<<endl<<b[0][0].data<<endl;
highest[priority[0]] = accCount;
for (int i = 0; i < attributes; i++)
{
    int large = 0;
cout<<"Level "<<i<<" attribute used "<<priority[i +1]<<endl;
    int counter = 0;
    for (int j = 0; j < accCount; j++)
    {
        if (b[i+1][j].data == 0)
        {
            continue;
        }
        else
        {
            cout<<counter<<" pnt = "<<b[i+1][j].parent<<" leaf = "<<b[i+1][j].node
            <<" data = "<<b[i+1][j].data<<" ;"
            if (b[i+1][j].outlier == 1)
            {
                cout<<"FLAG;";
            }
            counter++;
        }
        if (b[i+1][j].data > large)
        {
            large = b[i+1][j].data;
        }
    }
}
}
{ 
    large = b[i+1][j].data;
}
}

highest[priority[i]] = large;
cout<<endl;
}
b[0][0].print = 1;
for (int i = 0; i < attributes; i++)
{
    for (int j = 0; j < accCount; j++)
    {
        if (b[i+1][j].data == 0)
        {
            continue;
        }
        if (b[i][b[i+1][j].parent].print == 1 )
        {
            if (i != 0)
            {
                if (b[i][b[i+1][j].parent].data == highest[priority[i-1]])
                {
                    b[i+1][j].print = 1;
                }
            }
            else if (i == 0)
            {
                b[i+1][j].print = 1;
            }
        }
    }
}

119
//Calling the good printing function
string treefilename;
treefilename = "Trees/"+location[0]+"_"+filename+".txt";
treeinprint.open(treefilename);
int print_till = 0;
for (int i = 0; i < attributes; i ++)
{
    if (highest[priority[i]] == 1)
    {
        print_till = i;
        break;
    }
}
int print_counter = 0;
int val = 0;
int space = 20;
int high_number = 1;
int print_test = 0;
if (location_counter == 1)
{
    treeinprint<<"Tree for location at ""<<location[0]<<" "
    "is from file ""<<filename<<endl;
}
else
{
    treeinprint<<"Tree for corridor from ""<<filename<<" "
    "is printed below ""<<endl;
}
cout<<setw(19);
int move = 160;
cout<<"Total=""<<b[0][0].data;
cout<<"CROSSING CLUSTER=> ""<<endl;
treeinprint<<setw(19);
treeinprint<<"Total=""<<b[0][0].data;
treeinprint<<setw(move);
treeinprint<<"CROSSING CLUSTER"<<endl;
for (int i = 0; i < print_till+1; i++)
{
    print_counter = 0;
    high_number = 1;
    val = 0;
    print_test = 1;
    cout<<setw(move);
    cout<<attr_name[priority[i]]<<endl;
    treeinprint<<setw(move);
    treeinprint<<attr_name[priority[i]]<<
        endl;
    int space_test = 0;
    for (int k = 0; k < std_arr[priority[i]]; k++)
    {
        if (count_all[priority[i]][k][1] == 1 ||
            count_all[priority[i]][k][1] == 2)
        {
            if (space_test == 0)
            {
                cout<<setw(move);
                cout<<attr_name[priority[i]]<<"("<<count_all[priority[i]][k][0]<<")|"
                    <<endl;
                treeinprint<<setw(move);
                treeinprint<<attr_name[priority[i]]<<"("<<
                    count_all[priority[i]][k][0]<<")|"
                    <<endl;
                space_test = 1;
            }
            else
            {
                cout<<attr_name[priority[i]]<<"("<<count_all[priority[i]][k][0]<<")|"
                    <<endl;
                space_test = 1;
            }
        }
    }
}

cout<<endl;
treeinprint<<endl;
treeinprint<<endl;
for (int j = 0; j < accCount; j++)
{
    if (b[i+1][j].print == 0)
    {
        continue;
    }
    if (print_test == 1)
    {
        val = printingfunction(b[i+1][j].data
            , i, b[i+1][j].node, space-6,
            highest, priority, treeinprint);
        print_counter++;
    }
    else if (print_test == 0)
    {
        val = printingfunction(b[i+1][j].data
            , i, b[i+1][j].node, 20, highest,
            priority, treeinprint);
        print_counter++;
    }
    if (val == 1)
    {
        high_number = print_counter;
    }
    print_test = 0;
}
cout<<"\n";
treeinprint<<"\n";
if (high_number != 1)
{
    space=space+20*(high_number-1);
}
else
{
    space*=high_number;
}
}
treeinprint.close();
cin.get();