2018

Methods to Calculate Cut Volumes for Fault Trees with Dependencies Induced by Spatial Locations

Phillip Hanes
University of Central Florida

METHODS TO CALCULATE CUT VOLUMES FOR FAULT TREES WITH DEPENDENCIES INDUCED BY SPATIAL LOCATIONS

by

PHILLIP JEFFREY HANES
B.S. University of Arizona, 1986
M.S. Johns Hopkins University, 1992

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctorate of Philosophy in Modeling and Simulation
in the College of Engineering and Computer Science
at the University of Central Florida

Spring Term
2018

Major Professor: R. Paul Wiegand

ABSTRACT

Fault tree analysis (FTA) is used to find and mitigate vulnerabilities in systems based on their constituent components. Methods exist to efficiently find minimal cut sets (MCS), which are combinations of components whose failure causes the overall system to fail. However, traditional FTA ignores the physical location of the components. Components in close proximity to each other could be defeated by a single event with a radius of effect, such as an explosion or fire. Events such as the Deepwater Horizon explosion and subsequent oil spill demonstrate the potentially devastating risk posed by such spatial dependencies. This motivates the search for techniques to identify this type of vulnerability. Adding physical locations to the fault tree structure can help identify possible points of failure in the overall system caused by localized disasters. Since existing FTA methods cannot address these concerns, using this information requires extending existing solution methods or developing entirely new ones.

A problem complicating research in FTA is the lack of benchmark problems for evaluating methods, especially for fault trees over one hundred components. This research presents a method of using Lindenmeyer systems (L-systems) to generate fault trees that are reproducible, capable of producing fault trees with similar properties to real-world designs, and scalable while maintaining predictable structural properties. This approach will be useful for testing and analyzing different methodologies for FTA tasks at different scales and under different conditions.

Using a set of benchmark fault trees derived from L-systems, three approaches to finding these vulnerabilities were explored in this research. These approaches were compared by defining a metric called "minimal cut volumes" (MCV) for describing volumes of effect that defeat the system. Since no existing methods are known for solving this problem, the methods are compared to each other to evaluate performance.

- The control method executes traditional FTA software to find minimal cut sets (MCS), then extends this approach by searching for clusters in the resulting MCS to find MCV.
- The next method starts by searching for clusters of components in the three dimensional space, then evaluates combinations of clusters to find MCV that defeat the system.
- The last method uses an evolutionary algorithm to search the space directly by selecting center points, then using the radius of the smallest sphere(s) as the fitness value for identifying MCV.

Results generated using each method are presented. The performance of the methods are compared to the control method and their utilities evaluated accordingly.

First and foremost, I dedicate this work to my Lord and Savior, Jesus Christ, to whom I owe everything. I also dedicate this to the memory of my stepfather, William T. Vomocil, who pointed me on the path to this achievement years ago and who always believed in me.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Paul Wiegand, for his participation in my doctoral research. His constant advice, ideas, editorial services, statistical review and friendship have made this process educational and fun. I appreciate Dr. Ronald DeMara, Dr. Annie Wu, and Dr. Zixia Song for serving as members of my dissertation committee.

My sincere thanks go to the many bosses and mentors who have provided both the inspiration and the motivation to make this journey: Dr. Paul Deitz, Dr. Mike McGlockton, Dr. Dave Jerome, Dr. Dave Artman, Dr. Brian Kent, Dr. John Wilcox and Dr. Bruce Simpson. There isn't enough space here to mention all the ways they have helped me. They are much more than supervisors; they are true leaders who expanded my vision of what is possible and have helped me to become both a better researcher and (hopefully) a better leader.

Most of all, I would like to thank my family. My mom, who has been praying for me long before I started this. My wife, Ceci, and daughters, Danielle and Siobhan, who have patiently put up with my late nights and absences for years. Without their support, encouragement, and "family fun days," I don't think I would have made it through this.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

Acronyms used in this dissertation and where they are introduced.

Acronym	Meaning	Introduced
AM	Analytical Method	Subsection 4.2.2
BDD	Binary Decision Diagram	Section 1.1
CM	Cluster Method	Section 4.3
EA	Evolutionary Algorithm	Section 1.2.1
FTA	Fault Tree Analysis	Section 1.1
GA	Genetic Algorithm	Subsection 2.5.1
MCS	Minimal Cut Set	Subsection 2.1.4
MCV	Minimal Cut Volume	Section 4.1
NP	Nondeterministic Polynomial time	Subsection 2.1.4
PRA	Probability Risk Analysis	Subsection 2.1.4

CHAPTER 1: INTRODUCTION

The goal for the research described in this document was to explore the use of deterministic and heuristic methods to solve complex fault trees when system components are coupled due to constraints imposed by their spatial location.

To understand the need for this research, it is necessary to look briefly at fault trees, what they are and how they are analyzed. This will be followed with a description of one of the key limitations of fault tree analysis in its current form and motivate some reasons why it is important to find a way to solve them.

1.1 Fault Tree Analysis

A fault tree is a logical structure used to describe the dependency of systems on the components that comprise them. It uses a tree structure with logical operators at each node to define the relationship of each subsystem with its components. Fault trees will be defined in more detail in Section 2.1.2.

Fault tree analysis (FTA) is used widely in engineering applications to improve the reliability of systems by identifying weaknesses in design. Revealing potential problems enables designers to mitigate them by adding back-up systems or eliminating single points of failure. It has been used widely in industries where the costs of failure are unacceptably high, such as nuclear power [Vesely et al., 1981] or space launch vehicles [Stamatelatos et al., 2002]. It has also become widely applied in manufacturing, to improve mass produced items such as automobiles before they go to market.

Fault trees combine lower level components into subsystems and systems using a series of gates in a directed acyclic graph. Gates use the logic operators OR and AND, plus the combinatorial

operator K-OF-N to define the components that must fail to cause failure in the top level system. The construct is general enough to be used to study business [Geum et al., 2009] or bureaucratic processes [Lacey, 2011] as well as hardware and software systems. The lowest level entities (leaf nodes) in the tree are referred to generically as "basic events" when they refer to elements that are not hardware. In my research, the basic events are the failure of components regardless of the cause. Therefore, this document will use the term "component" throughout.

Fault trees can be used to calculate the probability of system failure or to find the combinations of components that can cause failure of the top level system. Calculating the probability is referred to as "quantitative analysis" and is performed using the rules of probability through logic operators. Finding the combinations of components (called "cut sets") that will cause failure is termed "qualitative analysis" and has historically been performed using Boolean reduction of the graph [Vesely et al., 1981] and more recently using binary decision diagrams (BDD) [Rauzy, 1993].

These algorithms provide closed form solutions and are deterministic in nature. That is, they always produce the same answer when they are executed with the same input parameters. They will usually execute very quickly for small to medium sized systems. The qualitative techniques will identify all of the possible combinations within a defined threshold. However, as the size of the system grows, the algorithms used to solve the fault trees require more memory and time to execute and produce an exponentially greater number of solutions.

Quantitative solutions show the effectiveness of improving the reliability of specific components or of changing the structure of the fault tree. In contrast, qualitative solutions reveal the combinations of components that constitute vulnerabilities. Hence, qualitative solutions are often used to find ways to mitigate problems in system design and quantitative solutions are used to evaluate their effectiveness. This research will focus on qualitative solutions for fault trees; they will be used to identify vulnerabilities in complex urban systems to disaster situations as described below.

1.2 Challenges with Fault Tree Analysis

An important limitation of the deterministic algorithms used for qualitative analysis is that they are based on the assumption of independence of the component failures. They are treated as simultaneous events without regard to physical location. However, real world events are always bounded by space and time. Thus, events affecting the top level system occur at a specific location. The interaction of event locations and the proximity of certain components of a system may cause dependencies that are ignored by the classic algorithms used to solve fault trees. These dependencies can have profound effects on the results of the fault trees.

Another challenge within fault tree research is communcating them clearly with other researchers. Graphics are the best tool for showing fault trees, but they are limited by the size of fault trees that can be shown on a single page. Of course, it is possible to share computer files with fault tree definitions, but that only works if those sharing them are using the same software package. Thus, there is no clear standard for sharing fault trees and larger fault trees are often discussed in the literature but seldom shown. Moreover, there are no existing parameterized models for constructing benchmark problems with well-defined charactersitics.

These issues will be discussed in the sections that follow.

1.2.1 Fault Trees with Location-based Constraints

It is not immediately clear how best to incorporate spatial constraints in the structure of the fault tree. Alterations to the fault tree structure would need to represent both the logical connectivity and the physical proximity. This may not be possible without significant conceptual changes to the fault tree. If such changes are made, however, there remains the question of whether it is possible to develop a deterministic algorithm to solve the resulting graph. Changes to the structure of the

fault tree would represent a major increase in the complexity of finding solutions.

As a consequence, little academic research has explored the question of physical proximity in a fault tree. However, this lack ignores some important interactions. A trivial example, shown in Figure 1.1, will suffice to show the potential impact. Assume, for example, that a building has a generator to provide power, and it also has a backup generator; but both generators are housed in the same room. A vulnerability is introduced into the building by the proximity of the generator and its backup (circles) as shown in the figure on the left. The solution, moving the backup to a different room, is shown on the right. In the first case, a fire in one room could render the building without power.

A classic fault tree would show that the power system is more robust because it requires two independent events to cause failure of the top level system, but common sense says they could both be damaged by a single common cause, such as a fire in the room containing the generators. Changing the layout by putting the generators on opposite corners of the building would be more robust, but classic FTA would treat the two designs as identical.

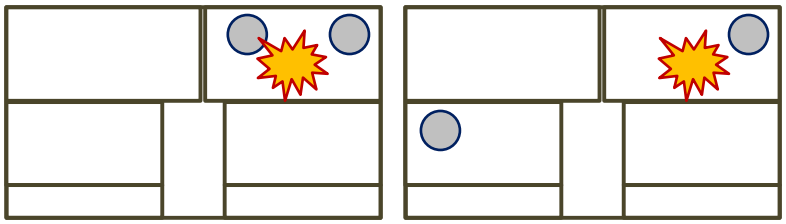

Figure 1.1: Illustration of the generator example.

Thus, classic FTA readily captures effects like wear and tear on the two generators, which proceed independently, but not an accident that impacts everything in the room in which they are housed. This difference can be critical when the research must focus on events such as a disaster in a

building or an urban area, where a common cause can affect many components simultaneously due to their proximity to one another, but may not affect more distant components. The independence assumption requires analysts to use other tools, such as an event-based simulation, to determine the state of the components, then combine that information with FTA to evaluate the overall impact on the system.

1.2.2 Example Application: Urban disasters

As seen above, fault trees have been applied to virtually every area of engineering where there is a need to analyze complex systems for weaknesses. Perhaps one of the most complex and diverse system of systems in the world today is a city. It has many interdependent systems with very different physical characteristics. Fault trees have been used to study the vulnerability of urban areas to various types of disasters [ten Veldhuis et al., 2009] [Sui and Pan, 2011]. These analyses tend to focus on using quantitative FTA to find a single component or type of component that can be improved to reduce the overall probability of failure. This is an important contribution to improving the robustness of the system, but it will not capture common cause failures such as the two generator situation described earlier.

In a recent report [Sundermann et al., 2014], the Swiss insurance company SwissRe analyzed 616 cities for vulnerability to 5 types of natural disaster. Similarly, a UN report [Gu et al., 2015] shows that hundreds of millions of people worldwide live in urban zones that expose them to mortality from one or more types of natural disasters as shown in Figure 1.2. The primary conclusions from these reports are:

- about 2.2 billion people worldwide live in zones that are significantly vulnerable to natural disaster, and

- it is imperative to find means to improve the robustness of urban areas to natural disasters.

The natural disasters studied by these reports (earthquakes, floods, cyclones, volcanoes, landslides) tend to affect very wide areas at once. Other disasters, such as tornadoes, tend to be more localized, while man-made disasters (fires, chemical spills, terrorist attacks) are usually centered at a single point and extend from there. In either case, a single event can affect many systems simultaneously. Urban planners need to account for these large areas of effect when deciding how to improve current infrastructure and where to locate backup systems.

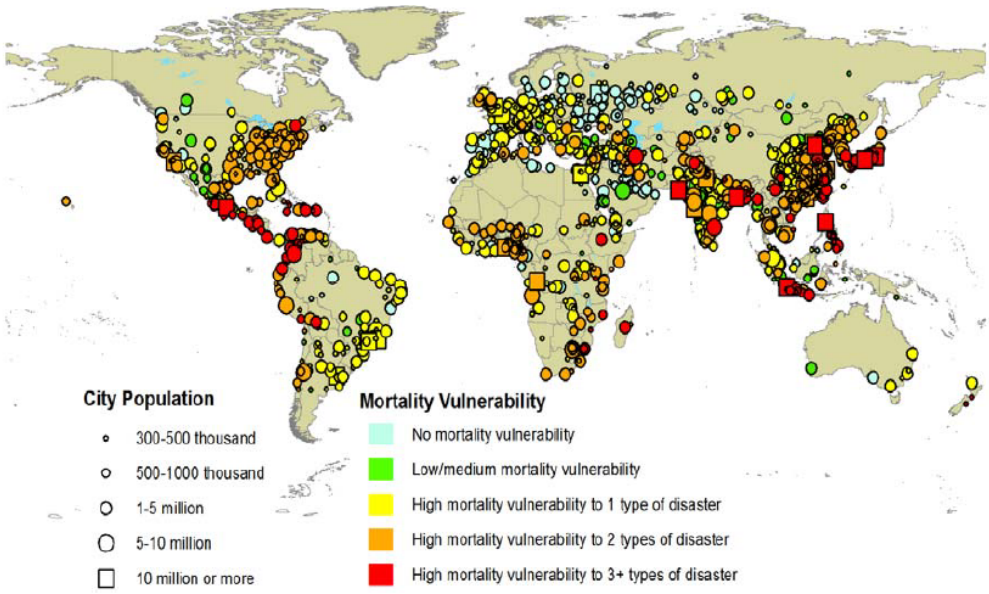

Figure 1.2: Map of global urban vulnerability to natural disaster. (*image credit:* [Gu et al., 2015])

Despite the obvious application, FTA in its current form does not provide a way to answer the question: What is the smallest volume of damaged components that will cause the overall system to fail? This information would be extremely useful if one were searching for vulnerabilities in an urban area in order to make it more robust to disaster scenarios. One could imagine a terrorist attack or perhaps an industrial explosion damaging everything within some distance of a particular

location. Finding ways to reduce the vulnerability of systems to these radii of effect will help planners to make urban areas more robust in the face of disaster.

This question is probably not going to be answered by searching for any MCS that kills the fault tree, rather it requires cut sets that occupy small volumes or arbitrary shapes; therefore it resembles a multi-objective optimization problem in a combinatorial space. My research focused on exploring methods to answer the question above.

1.2.3 Generating & Communicating Benchmark Problems

Fault trees for real system engineering problems can grow quite large in practice; up to hundreds or even thousands of components. Since the most common graphical representation of fault trees is to draw components as rectangles with connecting arrows, they can take up a lot of space. A typical fault tree graph will be illegible if it includes more than about 40 to 50 components on a single page. Attempting to show a full size fault tree with several hundred components in this manner would take up many tens of pages in a typical journal, which would likely distract from the point the author is trying to communicate. Additionally, larger fault trees for real world systems are generally not available in the literature because 1) they are proprietary to a particular company or 2) they are part of classified military applications (remember, FTA was originally created to study the Minuteman missile system).

Because of these reasons, sample problems found in the literature are small (ten to forty components), which is not large enough to challenge new algorithms described in the papers that include them. Thus, it is difficult to find useful benchmark problems that can be used to test new methods being researched or to compare them with other methods.

Nevertheless, as research in FTA continues, there is a need for benchmark problems that conform

to desired characteristics so that methods can be tested in a consistent manner. It is also important that such benchmark fault trees scale up to sizes encountered in engineering analysis so that new methods can be shown to address the needs of the analytical community.

1.3 Addressing these Challenges

The focus of this research was to extend FTA to incorporate the effect of imposing limitations of spatial proximity on the components that form the top level system. The question I am attempting to answer is, "where in the city will the smallest radius of effect defeat the top level system of systems?"

Developing a formalism to characterize the effects of proximity (or lack thereof) in the fault tree itself is a daunting task. One approach I explored was to use methods drawn from graph theory, namely clustering algorithms, to attempt to capture this phenomenon. I hoped that combining clusters into groups could provide a means to solve such a graph. In the end, the cluster-based methods attempted did not perform as well as other approaches explored in this research.

Another approach to solving the spatial proximity problem was inspired by my earlier research using genetic algorithms to solve fault trees directly. Genetic algorithms are a form of evolutionary algorithm (EA) that starts with random guesses for candidate solutions (called "genomes"), then progressively improves on them using the processes of combination and mutation. The new solutions created in each round of mutations (called a "generation") have been shown to converge on sets of input values that tend to produce near optimal results [Mitchell, 1998] [De Jong, 2006]. These processes will be described in more detail later.

The research described in this dissertation explored the use of EAs to find qualitative solutions to fault trees with spatial constraints. Specifically, EAs were used to identify volumes in which the

components formed valid cut sets for the top level system and this algorithm attempted to find the smallest such volumes. This simulates the hazards of placing related items too close together.

The evolutionary and cluster methods were compared to a control method based on an extension of existing FTA tools so that the performance could be anchored in a method relevant to the reliability community. The first control method was not a true contender; it simply used a grid of points and found the smallest volume centered at that point which would defeat the top level system. Even for two such locations, the execution time is too large to be useful. However, the second control method, using a traditional FTA tool then calculating clusters in the resulting solution, proved to perform well in this research.

A preliminary component of this research was to develop a means to automatically generate test cases to use for comparing the algorithms. In the literature explored, researchers use small sample fault trees to demonstrate the performance of their method; these usually have less than 20 components.

I developed software to generate fault trees using Lindenmayer systems (L-systems). L-systems are a method that uses a recursive symbolic replacement grammar to create tree structures; the selection of replacement rules govern the characteristics of the resulting tree. The goal was to use L-systems with varying rule sets to generate fault trees for testing the algorithms above. These generated fault tree structures were augmented with a stochastic approach to define the position of components. This constituted the complete definition of the problems evaluated by each solution method.

The generated fault trees ranged in size from 82 to almost 1000 components. However, they can be fully described with a few lines of rules and a depth to which the rules are expanded. This technique enables generation of large fault trees that demonstrate desired characteristics while stressing the capabilities of the algorithms used to solve them. Defining fault trees in this way can provide a real

benefit to the reliability community by allowing them to fully describe large, non-proprietary fault trees in a few lines of text — thus giving researchers a means to share large complex fault trees for testing advances in methodology. This makes it possible to create a design of experiments that measures the performance capabilities of the methods over a range of controllable conditions.

Measuring the performance of the various algorithms explored points the way to a set of analytical tools for solving spatially constrained fault trees. The techniques developed in this research can be used to explore the effects of component proximity on high level system failure. This information should point the way to means of improving the disaster resistance of urban areas or highly complex installations such as power plants or oil rigs. Other applications could include crash resistance for vehicles and improved resilience for software security systems.

1.4 Related Research

The problem of finding volumes of damage in fault trees is related to a few fields of study which focus on discovering vulnerabilities in complex search spaces. One such research area revolves around using evolutionary algorithms to improve resilience of computational hardware [Pyle et al., 2015], often by means of self-correcting mechanisms embedded in dynamic hardware resources [Imran and DeMara, 2011]. Another such area is known as constraint satisfaction problems (CSP), which are mathematical problems defined as sets of values whose collective state must satisfy a collection of constraints on those values [Dechter and Pearl, 1987]. CSPs often focus on homogeneous constraints, although mine are heterogeneous (binary for state and continuous for locations). These problems are generally highly complex, requiring significant computing resources to identify solutions as the problems grow larger.

In the context of CSP, this problem can be considered a specific case where the constraints lie in two

different dimensional spaces. Despite the extensive literature on CSPs, I cannot find any references to searching for vulnerabilities in fault trees. However, the notion of satisfying constraints in multiple dimensions is central to the definition of CSPs. A common method for solving CSPs is to use hybrid algorithms [Gogate and Dechter, 2005], in which two distinct solution methods are combined to exploit their complementary strengths in solving the problem — both of my clustering approaches are hybrid methods. Another widely used approach is to apply heuristics such as evolutionary algorithms to solve CSPs, as I have done here.

1.5 Contributions

The focus on generating benchmark problems is expected to provide benefits to the reliability engineering community independent of the main focus of this research.

- The use of L-systems for generating large fault trees that can be simply described is expected to help researchers define and communicate large problems for testing methodology.
- Creating metrics to describe and compare fault trees is also expected to help with research in fault tree analysis, as future methods can be defined as working better for specific types of problems.

The overall research focus is expected to enhance the robustness of system designs by helping to identify vulnerabilities caused by packing too many critical components too close to each other. In general, it is expected that separating these components will be sufficient to mitigate the issues discovered, although that is left to the design engineers. My goal has been to provide the tools that will answer the questions.

1.6 Dissertation Organization

The next chapter will discuss relevant Background information for the techniques explored in this research.

Because the development of benchmark problems is foundational to the rest of the research, the third chapter will be devoted to that phase of the work. It will focus on the methods used to develop the test set of fault trees and the results of that research (only). The benchmark fault tree instances generated in this work form the problem set for subsequent chapters.

The fourth chapter will discuss the various Approaches taken to solve for volumes in the fault trees.

The fifth chapter will show the Results of executing the control, cluster and evolutionary methods on the benchmark fault trees.

The sixth chapter will draw Conclusions and discuss potential directions for future research in this area.

CHAPTER 2: BACKGROUND

The focus of my research is to extend FTA to integrate the locations of components with the logical tree structure and so to find potential vulnerabilities caused by proximity of those components. To accomplish this, I used a variety of analytical methods including traditional FTA, evolutionary algorithms, cluster analysis and replacement grammars. The basis for each of these methods will be described in more detail in the following sections.

Whereas traditional FTA focuses on sets of components whose loss causes the top level system to cease to function (*i.e.*, "kills" the system), my research focuses on finding *volumes* that contain such sets of components. Thus, if every component within the volume is damaged, the result will cause the top level system to cease to function.

2.1 Fault Tree Analysis

FTA is a set of analysis techniques used to identify the components that constitute the greatest vulnerability for a system. That is, the components most likely to cause failure or combinations of components whose collective loss will cause the system to fail.

2.1.1 History

Fault trees were invented in the 1960s, when engineers at Boeing working on the Minuteman Missile program sponsored research at AT&T's Bell Labs to analyze the missile's launch control system [Ericson, 1999]. In that study, H.A. Watson proposed using a logic tree to describe systems in terms of the subsystems and components that comprise them. This permitted exploration of the

effect of damage to components and subsystems to the functionality of upper level systems. Use of this innovative technique spread throughout Boeing to many other government and commercial projects. One of the key proponents of fault tree analysis at this time was Boeing program manager Dave Haasl.

FTA grew significantly in exposure when the Nuclear Regulatory Commission began to use it to analyze nuclear power plants after the Three Mile Island incident in 1979. The handbook developed by the NRC [Vesely et al., 1981] became the standard text on the subject for decades.

Later, in the aftermath of the Challenger disaster in 1986, NASA began to use FTA to improve the safety of rocket and spacecraft designs. Their manual [Stamatelatos et al., 2002] is probably the most complete description of the standard approach to FTA available today.

The application areas mentioned above (*i.e.*, space travel and nuclear power) involve high profile systems where the cost of a single system failure is extremely high, in terms of lives, dollars and finally, in prestige for the agency or industry involved in the incident. As a result, the proponents of those systems are willing to invest a great deal of time, money and effort to reduce the likelihood of failure. FTA provides tools to identify vulnerabilities in the system and to evaluate the effectiveness of potential solutions.

With the ready availability of software tools to help in the construction and analysis of fault trees, the technique has spread to many other industries, such as the automotive industry [Ruppert and Bertsche, 2001], where the cost of a single failure may not be as high, but the number of units sold make the potential damage from systemic problems disastrous for the company. Recalls in the automotive industry have cost companies billions of dollars in a single year [Isidore, 2015]. These companies are highly motivated to ensure that their designs are less likely to fail under normal use and to protect passengers from catastrophic events. Carefully creating and analyzing fault trees can help designers identify and mitigate single points of failure and other flaws in a design.

Fault trees are widely used in simulations to describe the response of complex systems to combinations of simple events. They make it possible to link physical events in a simulation with the capabilities of the entities being modeled [Deitz et al., 2009] [Driels, 2013]. In this way, they enhance the realism of the simulation.

2.1.2 Structure of Fault Trees

Mathematically, fault trees are directed acyclic graphs that use logic operators to describe the interconnections among the components of a system. A fault tree consists of nodes and edges connected in a tree structure; the nodes represent either systems or components. The root node of the tree represents the system being defined for study, while the leaf nodes represent the basic components; in between may be any number of subsystems to define the specific elements of the system. Links connect a system or subsystem to the nodes (subsystems or components) that form it. System and subsystem nodes also have operators that indicate how the lower level nodes affect their functionality.

Formally, a fault tree can be defined as follows:

Definition 1 *A **fault tree** is a* 4*-tuple* $FT := \langle C, O, T, I \rangle$*, where:*

- C *is the set of* components *that can be affected by spontaneous events;*
- O *is the set of* operators *that aggregate events from lower-level components;*
- $T :\mapsto Y$ *is a function mapping an operator to a particular type of operation;*
- $Y := \{\text{AND}, \text{OR}, \text{K-OF-N}\}$ *are the types of operations;*
- $S := C \cup O$ *is the set of all* sub-system *roots in the whole system;*

- $I : O \mapsto \mathcal{P}(S)$ *is a function that describes the inputs of each operator;*

The state of any node (*i.e.*, component or system) is defined as a Boolean value indicating the *failure* of that node. That is, *true* (1) represents "failure", while *false* (0) represents "no failure". Thus, an undamaged component is in a *false* state — this is opposite the way Boolean values are used in many other contexts, so it may seem counterintuitive to those not familiar with FTA. Remembering that fault trees focus on the *failure* of the system will help to disambiguate the meaning.

The states of lower level nodes are combined to find the state of the parent node using operators which, for a classic fault tree, include AND, OR and K-OF-N. AND refers to the case in which all nodes below the operator must fail in order to cause failure in the system above; this represents redundancy in the system represented by the fault tree. OR is the case in which failure of any single element will cause failure of the system; this represents serial dependency on the components. K-OF-N refers to cases where there are n elements and it requires any combination of k [1] failed elements to cause failure of the system (where $1 < k < n$). Systems composed of these operators may be combined in arbitrary arrangements and to any depth required to define the subsystems that describe the functionality of the top level system.

A simple example system with OR and AND operators is shown in Figure 2.1. In this figure, a) shows a simple top level system with three subsystems comprised of two components each. The top level system A is an OR system and thus, will be defeated when any one of its subsystems fails. In turn, subsystems B and C are both AND systems and will only be defeated when both of their components fail. Finally, subsystem D is also an OR system. b) shows that killing two components

[1]The variable k is used here to denote the number of items in a combination. Elsewhere in this document, k is used to denote the number of clusters sought. Although this introduces some ambiguity, both uses have historical precedence, and it is confusing to introduce new variables to describe either value. Therefore, I use k in both senses and expect that the meaning will be clear from the context.

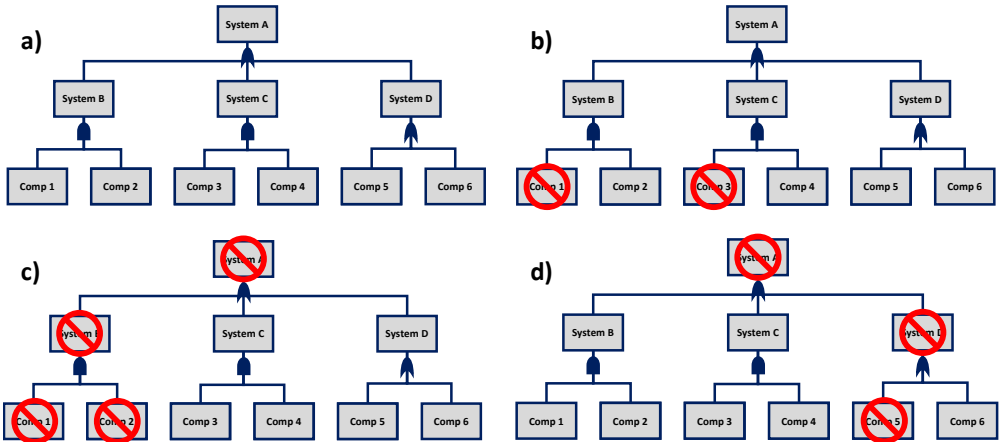

Figure 2.1: Example system structure and cut sets.

in subsystems B & C (different AND subsystems) is not sufficient to defeat system A. c) shows that killing two components in subsystem B (an AND subsystem) will cause system A to fail. Lastly, d) shows that killing a single component in the lower level OR subsystem will propagate through the tree to defeat system A.

Constructing accurate fault trees for large systems requires a significant investment of work investigating the functionality of the top level system and the constituent parts that make it function properly [Stamatelatos et al., 2002]. Depending on the complexity of the system, this process can take several man-years of effort to complete. One factor that can reduce the workload compared to other reliability techniques is that the practitioner is not required to evaluate all possible components, only those that are deemed to be most likely to cause failure need be elaborated. Others can be ignored; in fact, they are often subsumed in the larger events. For example, an automobile engine could be modeled as a single item, or as the block, pistons, fuel injector, etc. depending on the goals of the analysis. This subjectivity contributes to the power of FTA, but also requires experience on the part of the analyst developing the tree to provide "just enough" detail.

2.1.3 Functional Description vs. Physical Description

It is important to bear in mind that fault trees do not model processes, rather, they map the interdependencies of the lower level elements needed to execute a process. Thus, they answer the question, "if specific elements are no longer operational, is the system still functional?" Fault tree nodes are generally described as "top level events" and "basic events". This generalization allows the method to be applied to many situations that are not directly related to a physical analog. For this research, I use the terms "system" and "component", since my focus is on the physical infrastructure systems of an urban zone. These terms are grounded in physical systems and thus provide readily understandable concepts for description. However, my expectation is that the techniques developed in this research will have broader applicability to analyze any form of system dependencies.

2.1.4 Methods to Solve Fault Trees

There are two primary types of analysis performed on fault trees. The first is qualitative in nature; its goal is to find "minimal cut sets" (MCS). This involves searching for sets of components whose failure will cause the upper level system to fail when all of them fail. Any set of components that kills the top level system is called a "cut set." If no components can be removed without losing the distinction as a cut set, then the set is a MCS. Methods of generating cut sets (not necessarily minimal) will be a key theme throughout this dissertation.

Solving for all possible MCS is a #P complete problem [Ball, 1986], the equivalent of "NP complete" for counting problems. "NP complete" and "NP hard" are terms from computational complexity theory referring to problems that require Nondeterministic Polynomial time to solve. This means that the time required to solve the problem using currently known algorithms increases at

least as fast as a polynomial function of the size of the problem; often the time required grows exponentially or faster.

The other type of analysis is quantitative; it is referred to as "probability risk analysis" (PRA). This involves combining the probabilities of the lower level components to calculate the overall probability of failure of the top level system. The primary challenge for this type of analysis is obtaining reliable values for the individual component probabilities of failure. If there are no historical data, information must be generated by other means (*e.g.*, simulation or expert opinion) for use in rolling up the probabilities. PRA is particularly effective for finding single components that have the greatest impact on system robustness, which informs designers where to focus their efforts to improve system reliability. It is not so good at evaluating combinations that kill the top level system. For that, reliability engineers turn to qualitative analysis.

Classic FTA applications, such as SAPHIRE, perform qualitative analysis using a method called "Boolean reduction" [Vesely et al., 1981]. The fault tree is converted to an internal representation, then the rules of Boolean algebra are used to simplify, or reduce, complex system representations into their basic components. In particular, using the laws of associativity and commutativity, one may rearrange equations into convenient forms, then remove portions using the law of absorption. De Morgan's theorem is also particularly useful for rearranging terms into simpler forms. This process is repeated recursively through levels of the tree to find progressively simpler cut sets. Once this reduction process has found the simplest sets for defeating the top level system, they are reported as the minimal cut sets. Because this approach uses clearly-defined rules of Boolean algebra, it will find all MCS for small to medium sized fault trees as long as the number of possible MCS does not grow too large.

However, since larger systems may contain 10^{30} or more MCS, software implementing Boolean reduction usually allows the user to set a threshold value for the maximum size of MCS which

will be reported. Failure to do so can cause the algorithm to exceed memory limits for typical computer systems or reach run times that analysts will find unacceptable. The exact conditions under which this algorithm exceeds machine limits depends greatly on the form of the tree, the number of components, and the computer being used. In other words, it is not always easy to determine *a priori* whether the algorithm will be successful based only on a simple characteristic like the number of components in the system.

In the 1990s another approach was developed based on binary decision diagrams (BDD) [Rauzy, 1993] [Stamatelatos et al., 2002]. The basic approach is to start with a single component and branch into killed and not killed trees. Software based on BDDs has proven to find MCS faster than that implementing Boolean reduction [Gauthier et al., 2007], so many recent software packages have applied this approach.

The key to making BDDs work is to order the components advantageously so that kill conditions for intermediate systems are reached quickly. The problem of finding the *best* variable ordering is NP-hard [Meinel and Slobodová, 1994], but an efficient ordering can be found in polynomial time [Rauzy, 2001] and is normally good enough to provide fast solutions for small- to medium-sized systems.

However, the larger the size of the MCS sought, the larger and more complex the BDD becomes. Similar to Boolean reduction, the diagram can become so large that it overwhelms the memory of available computer systems and causes the application to stall. Therefore, this method is also dependent upon the practice of limiting the sizes of the MCS. Because the time required for these deterministic approaches scales exponentially with the size of the FT, analysts will always encounter a resource limitation when the problem reaches a certain size. My preliminary research on a typical PC (16GB of main memory) showed that both of these methods tend to exceed memory limits when the number of MCS is in the millions and the number of components required to defeat

the system is greater than 10 or so components, which can occur for systems with as few as 70 or 80 components.

Some researchers explored genetic algorithms in the 1980s as a means for solving fault trees [Ericson, 1999]. However, the combinatorial explosion of possibilities limited their usefulness on computer processors of the 1980s and 90s vintage and the focus of research moved to the deterministic methods mentioned above. My recent research [Hanes and Fay, 2015] demonstrated that they are useful for finding cut sets for larger systems and that they may be less susceptible to memory issues when solving these systems. They are subject to the usual caveats for stochastic solutions; namely 1) they are not guaranteed to find the minimum on every execution of the method and 2) they will generally not achieve the same answer(s) on consecutive executions. Nevertheless, they can be a compelling approach for solving larger fault tree problems. The fact that they make few assumptions about the structure of the problem means they can be adapted to finding MCS for very complex fault trees where traditional methods tend to exceed computational limits.

2.2 Applications of FTA

Since its invention, FTA has spread to virtually every field of engineering [Henley and Kumamoto, 1981]. A search for articles published on the subject in the past 5 years produced thousands of results. These include such diverse applications as mechanical systems (automobiles, aircraft [Saglimbene, 2009], buildings, etc.), electronics [Pan et al., 2008], and complex systems of systems such as power plants [Agarwal and Kansal, 2012] or even cities [ten Veldhuis et al., 2009]. In the general case, FTA can be used to describe any sort of dependent organizational structure, including bureaucracies [Lacey, 2011], businesses [Geum et al., 2009], or software [Tian et al., 2009]. The practical needs of these real world applications often exceed the capabilities of traditional FTA and motivate researchers to extend or adapt FTA techniques to solve their analytical problems.

2.3 Fault Trees in the Real World

Fault trees are highly useful for evaluating the functionality of systems in the real world, but they are limited due to a fundamental assumption in the nature of fault tree. That is, FTA assumes that the component failures (or "basic events") are mutually independent; to do otherwise would complicate the algorithms for solving fault trees to the point that they might not be tractable in reasonable time. In addition, the abstract nature of trees necessarily ignores the location of components, even though component proximity can result in dependent probabilities and common cause failure modes.

Components in the real world always have a physical location and failures happen at a particular time. These facts place constraints on system failures in the real world that are not captured in classical FTA. Nevertheless, they have implications for FTA that cannot always be ignored. Recognizing this fact has led practitioners to develop a variety of extensions to FTA [Ruijters and Stoelinga, 2015] that address specific limitations encountered in their research.

For example, Dugan et al. [1990], added time-ordered events to fault trees, which has grown into a subfield of study known as "dynamic fault trees" (DFTs). That is, fault trees in which the result can vary depending upon the order in which the components are killed. This was accomplished by adding three types of gates to the classic fault tree structure. Once a temporal sequence is added to the cut set, the number of possible permutations (not just combinations — the order matters) grows astronomically, which complicates direct algorithmic solutions. Consequently, DFTs have so far not been solvable by direct algebraic methods, although some researchers [Merle et al., 2010] [Merle et al., 2011] have developed structure functions for a subset of dynamic fault tree operators. The most common approach so far to solving DFTs has been to use Markov chains, the approach applied by the original developers of the concept [Dugan et al., 1992] [Boudali et al., 2007]. However, since Markov chain models grow exponentially with the size of the problem, this

approach has limitations for medium to large problems.

Although DFTs have enabled modeling time dependent nature of some systems, I found no methodology in the academic literature to capture the implications of the physical location of the components. Simulations of physical phenomena (*e.g.*, fires, floods, weapons, software, etc.) provide tools that can help address this lack, but they are generally used in a "feed forward" mode [Deitz et al., 2009] [Driels, 2013], so the physical analysis is not informed by the logical relations in the FT. Rather, the FT is used to evaluate the results of events described by the simulation, generally by passing assessed probabilities of component failure (or "probabilities of kill") to FTA in order to evaluate the probability of the top level event. The connection between the physical location and the system dependencies represented by the fault tree has not received consistent academic treatment.

Tobler's first law of geography [Hecht and Moxley, 2009] states that "everything is related to everything else, but near things are more closely related than distant things." To put this in the context of FTA, the closer components are to each other, the more likely they are to violate the independence assumption inherent in classic FTA.

The implications of the spatial constraints of real world events are the focus of my research. In particular, the role that physical proximity plays in system failure is key to modeling real world events with fault trees. This is particularly true with physical disasters, such as floods or tornadoes. Items that are close together are often more likely to fail as a group due to the common cause created by the disaster.

If we can better understand the vulnerabilities implicit in the physical arrangement of the components, then we can improve our chances of preventing them or at least preparing for them. This is easier said than done, however. Attempts to incorporate common cause failures into fault trees dramatically complicate the process of qualitative evaluation [Vaurio, 2003]. Most robust FTA

software incorporates common cause failures by treating it as a distinct basic event, while the individual component failures are treated separately. Qualitative analysis using such common cause failures will only identify the cases where the common cause is known to the analyst *a priori*. While this does allow inclusion of common cause failures in a fault tree, it does not provide a means to find the failures in the first place. This is not surprising given that the fault tree itself does not have a means to express information about the relation of the components to each other apart from their inclusion in a single system.

2.4 Cluster Analysis

One obvious approach for finding cut volumes in the fault tree can be borrowed from graph theory. Since fault trees are graphs, they can be explored using some of the tools and metrics of graph theory [Chartrand, 1977]. One area of research intersecting graph theory that has received a lot of attention is cluster analysis.

Cluster analysis focuses on the relationships of items to each other. The idea is that elements of a set that are more like each other will form a cluster that has characteristics distinct from another cluster of elements that are also similar to each other. These characteristics can be either Euclidean proximity or similarity in some other set of dimensions. A common difficulty with cluster analysis is that the way of defining "similar" can vary dramatically depending on the application and the nature of the elements that are being analyzed.

Cluster analysis combines elements of statistics, graph theory, and computer science [Han et al., 2001]. It is used widely in machine learning, pattern recognition, bioinformatics, and image analysis, to name a few applications. Early cluster algorithms focused on centroids of the locations of elements of the set; this is the basis for Lloyd's algorithm (a.k.a. the "k-means algorithm"), which

is described in detail in Chapter 4. k-means is often considered outdated and ineffective since it sometimes ignores the proximity of elements to each other and large gaps that can occur. These irregularities are easy for a human to spot, but difficult to define in algorithmic terms.

Much of the focus in cluster analysis research has been to find algorithms that detect irregular-shaped clusters with different distributions or formed by groups with a similar density in some region. Such clusters are often obvious upon visual inspection by a human, but it has taken many years of research to develop algorithms that find these clusters reliably. Each such algorithm is focused on defining clusters suitable to a particular requirement.

For this application, I determined that the k-means algorithm was appropriate for finding the type of clusters needed to represent a sphere of effect. Since I was focused on the location of components in three dimensional (*i.e.*, Euclidean) coordinates, it was necessary to cluster items according to their proximity to a central point. k-means generally performs well at finding uniformly-sized spherical clusters, even though it has difficulty with irregular shaped clusters and variable-sized spheres. For my research, the goal was to emulate the radius of a blast or fire, which is reasonably approximated by a sphere in open spaces. Therefore, it is sufficient to find regular-shaped clusters in the search space.

Once the clusters were defined, the set of components contained in each cluster could then be checked versus the fault tree to determine their impact on the top level system. Then, combinations of clusters could be checked similarly up to the value of k being studied.

2.5 Evolutionary Algorithms

Stochastic optimization methods have gained in prominence during the past decade as they have shown their ability to find good solutions to problems that have not so far been solved using de-

terministic methods. The word "stochastic" refers to the use of random numbers in the algorithm; that is, these methods rely on some form of random exploration of the search space to find better solutions. Thus, they usually do not produce the same answer each time they are executed. In comparison, deterministic methods will always produce the same answer. Therefore, stochastic solutions are typically used in cases where there is no deterministic solution or when said solution takes too much time to execute for normal-sized problems.

Evolutionary computation refers to a class of stochastic optimization algorithms in which a collection of initial guesses (usually randomly generated) are iteratively refined through a series of operations which transform them while they are evaluated for their closeness to the desired answer, called "fitness." Less fit solutions are abandoned while more fit solutions are refined in the next step [Mitchell, 1998] [De Jong, 2006]. A conceptual illustration of this process is shown in Figure 2.2.

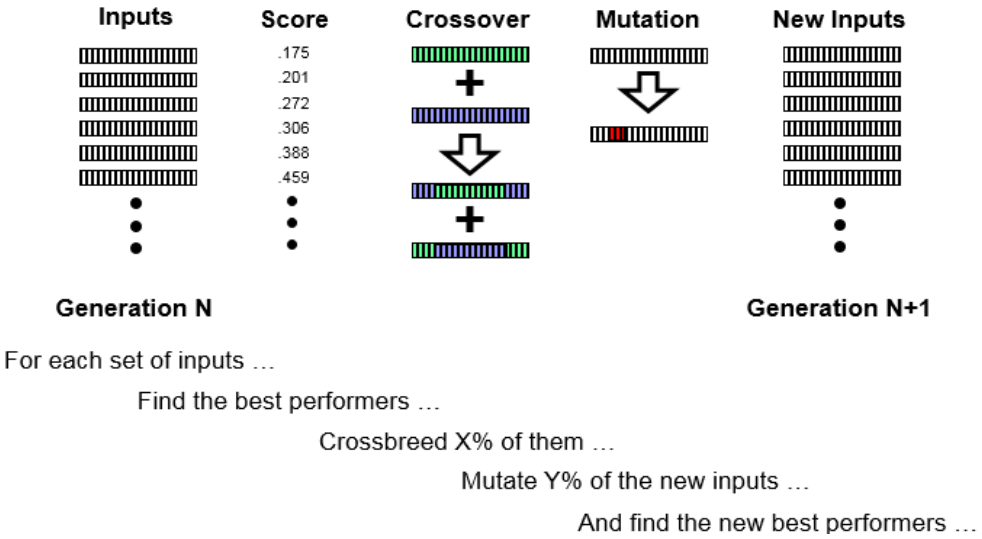

Figure 2.2: Conceptual illustration of one generation for an Evolutionary Algorithm

In practice, evolutionary algorithms (EAs) have proven to be a very flexible tool for solving difficult optimization problems that have defeated other methods. Their flexibility is closely related to the

generality of the genomes and the fact that the fitness function can be literally any numerical metric that the user defines. This flexibility has proven to be a great strength for solving problems. It also points to one of the difficulties of EAs: there are so many ways to structure the algorithm that it can be difficult to find the right representation for a given problem, but this flexibility is the key to effective use of EAs. With the right representation, EAs have been shown to solve complex optimization problems that have defeated other methods. For some types of complex problems, they have been demonstrated to be the best known solution method for finding good approximations for the multi-objective shortest path problem in polynomial time [Horoba, 2009]. In addition, other research [Zhu et al., 2014] shows that they provide superior performance for optimization problems that must balance different sets of objectives simultaneously.

2.5.1 Application of Genetic Algorithms to Fault Tree Analysis

There were early attempts to use EAs to solve fault trees [Ericson, 1999], but the speed of deterministic methods, particularly BDDs, rendered EAs unnecessary for typical FTA problems, such as finding MCS. Although deterministic methods have difficulty generating all cut sets for large fault trees, this is a result of hardware limitations, not a problem with the algorithms themselves. In practice, smaller solution sets are not a problem since most analysts would not have the need or ability to search through 10^{20} or more MCS. A sampling of the smallest sets is usually enough for most applications.

Antoine Rauzy (personal communication, June 16, 2014) indicates that GAs have two distinct problems when attempting to find MCS for fault trees. 1) The nature of the top level event is binary, thus each genome will receive a fitness of 0 or 1, with no opportunity for values in between. This sabotages the normal GA operation of gradually approaching optimal solutions by finding successively more fit genomes. 2) The fact that there is not necessarily a natural order to the

list of components means that crossover operations may randomly eliminate otherwise promising genomes by "turning off" a component that is crucial to creating a particular cut set. The fact that a given component set "almost works" will be lost due to problem (1). These are valid concerns and need to be addressed to configure an EA approach that consistently produces useful results. My approach to resolving these issues will be described in Chapter 4.

As a result of the issues mentioned above, EAs are not the tool of choice for qualitative analysis of fault trees. However, as noted in my earlier work [Hanes and Fay, 2015], they can be a useful technique for resolving large fault trees.

On the other hand, if spatial constraints are added to the fault tree, a more complex representation is required. Since this representation has yet to be defined, neither defined metrics nor deterministic approaches exist to find solutions for the most damaging cases.

The example of DFTs is illustrative. They were first introduced in 1990, and the developers used Markov Chain models to solve them; these were only suitable for small problems due to the exponential growth in execution times. To date, no faster algorithms have been developed, despite decades of research. In the meantime, heuristic methods have been applied to solve DFTs because researchers face large, time-dependent problems that need to be solved in the near term, and they cannot wait for fast-running deterministic solutions to emerge.

If someone were to develop generic means to include spatial constraints in the fault tree itself, we could expect it to take decades for the academic community to find fast deterministic algorithms to solve reasonable sized problems, assuming that it is possible. Therefore, when spatial constraints are added to the fault tree, EAs become an attractive method that is worth considering in comparison with approaches based in deterministic methods.

2.6 Lindenmeyer Systems

Finally, to exercise thoroughly the methods in this research, it is important to use a variety of test cases incorporating many different sizes, depths, and branching characteristics. Lack of variety in test cases is a common deficiency observed in FTA literature. Many papers found apply a method to one or two relatively small sample systems (*e.g.*, [Pan et al., 2008], [Sui and Pan, 2011], [ten Veldhuis et al., 2009], and [Xiang and Yanoo, 2010]); the reader is left to infer that the method described works for larger cases. My research focused on large systems that are difficult for the standard deterministic algorithms to solve even without the addition of spatial constraints. Therefore, I needed several large systems that demonstrate this difficulty. Unfortunately, while it is possible to randomly generate fault trees, it would be difficult to do so in a way that preserves the important characteristics of the fault trees as they grow larger and that is reproducible for academic purposes.

To that end, a preliminary step in this research was to develop a method to automatically generate fault trees with desired characteristics. The method is parameter-driven so that I can control the types of trees generated and explore performance of the methods as these characteristics change in predictable ways.

Lindenmayer systems ("L-systems" for short) are a grammar used to recursively generate sequences of items through symbolic replacement. The method was initially created to generate computer models of plants [Prusinkiewicz and Lindenmayer, 1996], but it has been applied to many other fields as diverse as music generation [Manousakis, 2006] and procedural creation of buildings for video games [Watson et al., 2008]. The extension to generating fault trees follows naturally with the simple addition of an operator to each replacement rule so that lower level branches become subsystems with a specific type of function.

Moreover, L-systems have been used successfully by researchers to construct benchmark problems for other fields. For example, Martin et al. [2010] used functional L-systems within automated scenario generation tools to build up scenarios in simulations and games used for training. Additionally, Ahammed and Moscato [2011] use L-systems to design challenging traveling salesman problem (TSP) instances for benchmarking TSP solvers. They demonstrate that increasingly more challenging TSP instances could be produced while maintaining salient properties of existing, known problems at varying scales.

As an example of an L-system grammar with operators for each node, a simple set of rules could be defined in the following manner:

start: A

A: (2 of 3) BCB

B: (OR) CBC

C: (AND) BA

The colon indicates that on each iteration, the indicated letter on the left is replaced with the sequence to the right. One can generate a fault tree by defining a specific number of iterations, which would create the desired branching behavior as the tree grows through several iterations. Thus, the overall system produced by the rule set shown above after zero, one, and two iterations would look as follows:

0th iteration: [A]

1st iteration: (2 of 3) [B][C][B]

2nd iteration: (2 of 3) {(OR) [C][B][C]} {(AND) [B][A]} {(OR) [C][B][C]}

where "{ }" encapsulates a subsystem. A better way to understand this progression is to represent it graphically, showing the FT operators and each rule element as a box with the appropriate letter, as shown in Figure 2.3. The final step to using this in a FT would be to give each subsystem and

each leaf node a unique identifier so the analysis software can distinguish between them.

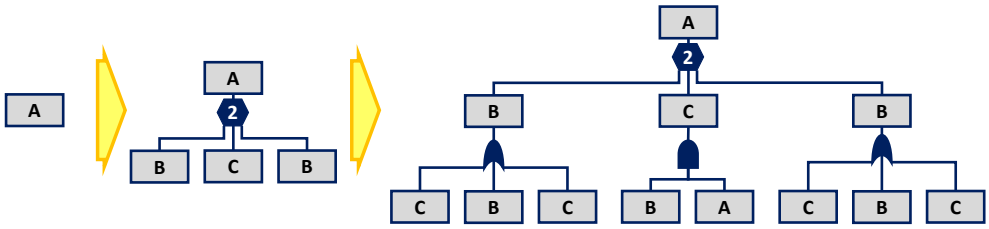

Figure 2.3: Example of growth for a simple L-system grammar

As this system progressed through more iterations, the system definition would grow larger and more complicated. The rule set above could be expanded in several ways by adding more letters with alternate operators and numbers of subsystems at each level. It could also add terminators in some cases so that the fault tree did not always reach the same depth on all branches.

With these tools, I developed a straightforward application to generate fault trees for use in the methods described in earlier sections. More work will be required to add the ability to repeat subsystems and components in different portions of the fault tree, which is one of the complicating factors in real world FTA.

2.7 Conclusion to Background

The discussion above shows a compelling application for fault trees constrained by physical proximity to find vulnerabilities in urban areas to natural and man-made disasters, as well as a lack of straightforward techniques to address this problem. In this research, I explored Evolutionary Algorithms and Cluster Analysis to find a repeatable approach to solving this problem efficiently. As a basis for comparison, I also developed a control method using a BDD-based solver to find MCS, then applied the k-means algorithm to evaluate volumes. To thoroughly test these methods

under a variety of conditions, I used L-systems to generate a robust set of fault trees with varying characteristics as measured using graph-based metrics.

CHAPTER 3: METRICS & BENCHMARKING

The first step in my research was to obtain fault trees for use in testing the algorithms of interest. To be useful for showing differences between the methods, the test cases must be large enough to be difficult to solve using traditional methods. At the same time, they needed to be useful from an academic point of view.

3.1 Generating the Test Set

Experience with fault trees in an industrial environment has taught several lessons. Unfortunately, due to their proprietary nature, specific fault trees encountered in such settings typically cannot be published in an open format. Important observations from this experience include:

- Fault trees with several hundred components are normal;
- There is a need for metrics to describe structural variances in fault trees to help understand analytical differences;
- There is a need for non-proprietary benchmark fault trees that can be shared in open literature.

As a precursor to this research, I searched for test cases in the academic literature that could be used for benchmarking the methods. However, the fault trees I found all had less than 50 components and could be solved by existing FTA software in a fraction of a second. The preponderance of small fault trees is not surprising since a legible graphical image of a fault tree containing 50 components will fill a page in a journal. Thus, larger fault trees would quickly take up the entire space allotted

to a typical article before spending any space on the method under discussion. Nevertheless, the paucity of examples leaves researchers with no standard benchmark cases that can be used to compare developmental methodologies.

To determine how the methods performed at scale, it is necessary to generate large fault trees that stretch their capabilities. Therefore, a method is needed to create fault trees that meet the necessary criteria. To provide consistent benchmarks for researchers, the method must produce fault trees that are:

1. Reproducible,
2. Similar to real world examples,
3. Scalable in measurable ways from small trees to large trees.

Evaluating both #2 and #3 require some sort of metrics for comparing fault trees.

3.2 Metrics for Fault Trees

The primary dimensions that I applied in this research are: Size, Operator proportion, and the Size of the smallest MCS.

Size: The number of components in a fault tree has been shown to be one of the factors that impacts performance of deterministic methods and makes them unsuitable for very large problems [Gauthier et al., 2007]. Therefore, I exercised methodologies using smaller systems initially to prove each concept, then applied them to larger systems to evaluate performance in more difficult cases. The size of the fault tree can be increased simply by executing more replacement cycles of the L-system; this can be performed as often as required to generate the size of fault trees needed

to provide a challenge to the methods. The largest systems used for this research contained close to a thousand components; which is a stressing size for FTA in industrial applications [Gauthier et al., 2007].

Size of smallest MCS (s_{MCS}): This appears to be a key component contributing to the complexity of fault trees. That is, it makes them difficult to solve. Therefore, the fault tree generator was programmed to use this value as a part of the multi-objective fitness function. If the s_{MCS} is small (*e.g.*, 5 components or less), then the fault tree is easy to solve using traditional methods; at least, it is easy to find small MCS that kill the system. But, as s_{MCS} grows larger, classical methods begin to struggle to find any MCS and other methods will be required. Not surprisingly, during the development of the test set, FTs were generated with MCS that reached ridiculous sizes; in one case, s_{MCS} was almost 160 components for a fault tree with about 300 components. This is too large for any real world system as it would indicate an extraordinary level of redundancy. This led to putting an upper limit on the size of the MCS as well as a lower bound. Thus, the goal was to generate fault tree that had an s_{MCS} that was not too small and not too large.

Operator proportion: Defined as $n_{OR}/(n_{OR} + n_{AND} + n_{KofN})$. The overall percentage of AND, OR and K-OF-N operators has an effect on the number of MCS found in the fault tree. The effect is not a straightforward function of this proportion, however, since the determining factor for creating more MCS is whether AND and OR operators appear in alternating layers of the fault tree. If an OR system has an OR system below it, then by the Boolean Law of Absorption, the components of the lower system actually functions as part of the parent system, as shown in Figure 3.1. The same holds true for systems of AND operators.

This is similar to the Boolean Satisfiability Problem (SAT) in which the satisfiability of a disjunction of conjunctions can be solved in linear time, but the satisfiability of a conjunction of disjunctions requires exponential time to solve. Generally, with randomly created rule sets, there is a

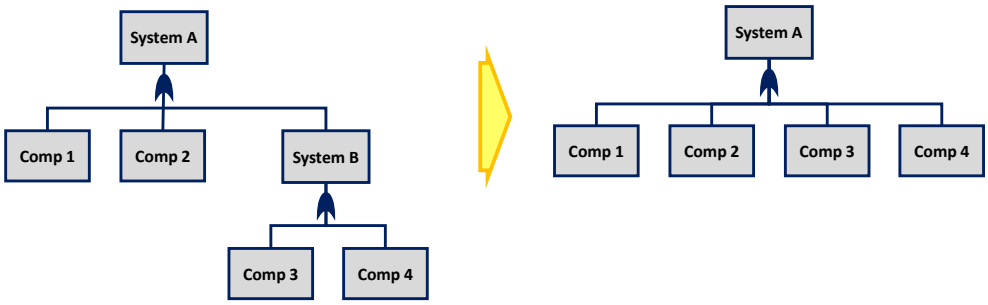

Figure 3.1: Law of Absorption for systems of OR operators

mix of both operators at every level of replication. This creates a cascading series of conjunctions and disjunctions — each conjunction (AND operator) with disjunctions below it increases s_{MCS}. Deterministic methods for finding the MCS are roughly equivalent to the process of converting the Boolean expression to disjunctive normal form, a process which can generate an exponential number of branches.

Finally, the nature of the K-OF-N operator means that it will *not* absorb a subsystem with a K-OF-N operator; it is not a Boolean operation and the Law of Absorption does not hold true for it. However, a K-OF-N node can be reduced to an equivalent disjunctive normal form, which is subject to the laws of Boolean algebra. The number of nodes in such a reduction grows exponentially since it is governed by the combination formula. Therefore, the K-OF-N operator will add complexity by increasing the number of possible solutions and the number of components in each one. Its presence is an indicator that the system in question will have a larger number of MCS than a system with the same number of components that does not have this operator.

Out Degree: this reflects the amount of branching at each level. The measure used is derived from graph theory and reflects the number of links exiting from each node. Every system and subsystem node has a single "in" link and two or more "out" links (component nodes have no "out" links).

Two of these metrics (Number of Components & Size of Smallest MCS) increase as the rule set is expanded through more iterations. The other two (Operator Proportion & Out Degree) relate to the structure of the fault tree and should not vary significantly as the rule set is expanded past the first three or four iterations. Experience shows that there may be significant variation with the first few iterations, but that these metrics asymptotically approach steady state values as the fault trees grow larger with more iterations.

3.2.1 Calculating Number of MCS

The number of possible MCS for a given fault tree is a useful metric for assessing the complexity of fault trees. If a fault tree does not reuse subsystems or components, then the total number of MCS can be calculated directly by a recursive function without solving the system. Many fault trees are cyclic in the graph sense; that is, they repeat the use of some systems or components. This method will *not* work for such systems.

The number of MCS is calculated by propagating the counts of component combinations through the gates of the fault tree. The calculation is different for each type of operator that defines a node in the tree.

If a node has n elements, each with a number of MCS M_1 through M_n, then the total number of MCS for each type of node is:

$$M_{OR} = \sum_{i=1}^{n} M_i$$

$$M_{AND} = \prod_{i=1}^{n} M_i$$

$$M_{KofN} = f(k, 1)$$

Where $f()$ is a recursive function applied to $(M_1, ..., M_n)$ as follows:

$$f(k, i) = M_i \cdot f((k-1), (i+1)) + f(k, (i+1))$$

The first argument, *k*, indicates the number of elements to be chosen; the second, *i*, indicates the element with which to begin. This function is applied recursively to progressively smaller problems until it arrives at one of two cases. If $k = 1$, this indicates 1 of $(n-i)$ elements, which is equivalent to the OR formula. If $k = (n-i)$, this indicates $(n-i)$ of $(n-i)$ elements, which calls for the AND formula.

The number of MCS is calculated by executing a depth-first traverse of the tree. Every leaf node (basic event) has exactly 1 MCS, which contributes to the number of MCS for the system(s) above it. The values for each node are propagated up through progressively higher level nodes until the top node is reached.

I verified the accuracy of this formula by comparing its results with the count of MCS obtained from *xfta* for small to medium fault trees that I found or generated. It proved to be accurate on systems with up to several million MCS.

Knowing the number of MCS provided insight into the results derived from some of the methods used, but did not necessarily provide the desired indicator of complexity. In the course of the research, I discovered that the *size* of the MCS was actually a more useful indicator of the complexity of solving a fault tree.

Table 3.1: *Metrics for Fault Trees from the literature.*

source	c	# MCS	OR	AND	out	in
Song et al. [2009]	8	7	0.83	0.17	2.17	1
Lacey [2011]	17	15	0.71	0.29	2.14	1
Sui and Pan [2011]	20	24	0.75	0.25	2.83	1.03
Shahriar et al. [2012]	42	92	0.70	0.30	2.83	1.15

3.3 Matching Fault Trees Found in the Literature

With these metrics in mind, my foundational hypothesis focused on the feasibility of using L-systems to produce fault trees of various sizes with predictable characteristics.

Hypothesis H0: L-systems can be used to generate fault trees that retain measurable characteristics that scale as they increase in size with more iterations of the rule base.

Success criteria: For a given rule set, size metrics (Number of Components & s_{MCS}) grow in a predictable way, while structural metrics (Operator, Out Degree) do not vary by more than 10% of average as size increases.

I used fault trees found in the literature as a baseline for comparison with the generated trees. Although I did not find large fault trees, I found some with a few tens of components. The sources are listed in Table 3.1 along with calculated metrics to characterize their structure. The metrics include: number of components ("c"), the number of MCS for that fault tree ("# MCS"), the proportion of system nodes using the OR operator ("OR"), and the out-degree of the system nodes ("out").

Since these fault trees were few in number, I matched their structural metrics by manipulating L-system rule sets by hand until the resulting fault trees were close to the desired values. The resulting rule sets can be found in Appendix A. Though this type of manual process is time-consuming, the

Table 3.2: *Metrics for L-System Fault Trees matching those found in the the literature*

source	i	c	MCS	OR	AND	out	in
Example 1	3	7	6	0.80	0.20	2.20	1
Example 2	4	17	31	0.71	0.29	2.14	1
Example 3	3	20	17	0.73	0.27	2.82	1.03
Example 4	5	40	68	0.70	0.30	2.81	1.15

control it provides over the process helps to develop intuition regarding the relationship between input and output which helped for developing the Benchmark test set shown in Section 3.5.

The first set of fault trees generated using the manually-generated L-systems demonstrated good agreement with the metrics for the target systems. Table 3.2 shows the metrics for the fault trees generated using L-system grammars; they can be compared with those in Table 3.1. Although these L-system grammars were generated by hand, they demonstrated that the approach is feasible for creating fault trees with desired characteristics. The process of generating L-systems by hand also helped me appreciate the difficulty of balancing the metrics in a given rule set. Seemingly small changes in the rules often produce large effects in the characteristics of the resulting fault trees.

The rule sets for the L-systems are similar to those shown in the Background discussion. They can be altered to produce trees with a higher or lower degree of branching (i.e. trees with more sub-systems at each level) and to incorporate the operators in different proportions to reflect different types of system design. Additionally, expanding the tree more levels creates different sized trees.

One factor that influenced the use of these rule sets is that some of the structural metrics will often vary widely in the first several iterations, then asymptotically approach a stable value. Sometimes they approach from one direction, but other times they will bounce above and below until reaching the final value. This is because L-systems are dynamical systems and, like many such systems,

pass through a transient phase before settling down. This suggests that researchers who use this approach should ensure that the L-systems passes through sufficient rule replacement iterations. (*e.g.*, after 4 cycles, my L-systems generally approached steady state values). Paying careful attention to the metrics of each successive fault tree in the iteration sequence is required to ensure that the generated trees show the desired characteristics.

Results produced by attempting to emulate the limited sample of fault trees found in academic literature reveal that L-systems with relatively few rules are able to produce fault trees with characteristics that closely resemble these real world systems. The initial rule sets were generated in a series of manual cycles, making adjustments to one or more rules, then generating a new fault tree to determine whether its metrics were closer to the desired characteristics. While the small sample shown here is not conclusive, it is sufficient show that L-systems can be used to create fault trees with desired characteristics to explore the functional boundaries of various approaches to solving them.

3.4 Automatic Generation of L-System Rules

The success of the manually created L-systems at closely emulating fault trees was sufficient encouragement to provide the impetus to develop an EA to create fault trees that would provide a realistic set of cases to explore the capabilities of the research methods.

In practice, developing EAs that search for appropriate L-systems proved to be quite challenging. This is in part because of the nature of the rule sets themselves, which are not easy to characterize as a genome, and in part because of the difficulty of identifying appropriate fitness metrics.

1) **EA Representation:** Each candidate solution in an EA is represented by a genome. For this problem, the genome comprised a collection of genes, each of which contains a single rule, where

a rule is defined as an operator + references. Each reference had to correspond to another gene, so the selection of references was limited to the number of genes in the genome. Each gene therefore represented a single replacement rule in the L-system.

To use the genome, the fault tree generator interpreted the rules starting from the initial rule (which was always 'A'). In each replacement cycle, when that letter was encountered in a leaf node, it was replaced by another node with the designated operator and a set of leaf nodes with the given letters.

One problem encountered in the operation was that some rules could end up not being referenced by other rules. Although this would limit the variation, it did not result in a rule that could not be interpreted, so it was not checked for during the process.

In addition to the normal fault tree operators (AND, OR, and K-OF-N), I added a terminator operator ("T") which would cause the expansion to cease along that particular node. This was intended to prevent the unnatural situation in which all branches of a fault tree terminated at the same level. With the terminator as an operator, some branches could end at various levels, creating a more realistic structure.

Crossover worked by combining genes from each genome using a single point crossover with the point selected at random from the range 2 to $n-1$ where n was the number of genes in the smaller genome.

Mutation was designed to change either the operator or the arguments in a single gene selected at random. 25% of the time it would change the operator to another selected randomly; if necessary, it would add or remove arguments. 75% of the time it would change the arguments for a rule that has them. With equal probability it would 1) remove the last argument, 2) add one randomly selected argument, or 3) change one argument at random.

2) **EA Fitness:**

To evaluate the fitness of each genome, I generated a corresponding fault tree by expanding the rule set to a random depth. Then I used a multi-objective fitness function combining structural and performance metrics to characterize the fault trees that I was seeking. My approach to using multiple objectives is to randomly choose one evaluation objective each time genomes are selected for the next generation.

The objectives were in two dimensions. First, I minimized scaled deviations from desired structural metrics (out degree, operator proportion & number of components). Specifically, I combined the structural metrics into a single value and set the EA to minimize it:

$$val_T = \sum_{i=1}^{3} \left(\frac{val_{i,c}}{val_{i,d}} - 1 \right)^2$$

where $val_{i,c}$ is the actual metric calculated for that fault tree and $val_{i,d}$ is the desired metric targeted by the EA.

Second, I checked for the "complexity" of each fault tree by calculating the number of components in the smallest MCS. Simply maximizing that value produced some very unrealistic results (*e.g.*, in one case, it generated a fault tree with about 300 components that had over 100 components in the smallest MCS). Thus, my metric seeks values within a selected range and penalizes genomes the further they are from that range.

To find this value, it was necessary to perform quantitative analysis on each fault tree to determine the size of the smallest MCS. Generally, an analyst will tune *xfta* to find the right execution parameters for the fault tree being solved. With an EA generating about a hundred fault trees each generation, this was not possible. Therefore, I set the fitness function to execute a loop in which it gradually grows the size of the MCS to search for until *xfta* found a solution or grew too large and threw an exception due to lack of memory. In the case of a memory fault, I switched to a Monte

Carlo elimination scheme (described in more detail in Section 4.2.2) as an alternative means to find the size of the smallest MCS.

3) **EA Dynamics & Selection:** My EA employs both parent and survival selection. Parents are selected using a variety of tournament selection: four individuals are selected uniformly at random from the population and the two most fit (smallest fitness value for one randomly chosen objective) are then chosen to produce two children via the crossover and mutation operator described above.

Survival selection uses a $(\mu + \lambda)$ strategy: 50 parents were used to produce 50 children, then the most fit 50 were selected from those 100 individuals to survive to the next generation. The $(\mu + \lambda)$ dynamic was chosen primarily because of the time required to compute the MCS and a desire to retain the "best" answers found in the process.

This structure ended up being fragile. After each cycle of crossover and mutation, it was necessary to add a function to check all arguments and make sure they made sense. That is, it had to ensure that all references still existed, that operators still had 2 or more arguments, and that all rules had at least one other rule that referenced them — thus, the function cleaned up rules as needed after the check.

Due to the long run times for the fitness function, I elected to use a steady state EA, so the population retained the best genomes from all generations. Even with this advantage, due to the computational difficulty and extensive run times (sometimes it took several days to execute 40 generations), the EA was not usually able to converge.

Thus, I chose to use the best genomes found in the final generations. I made this decision since the focus of my research was *not* to find an all-purpose fault tree generator, although that would be a valuable contribution, but to generate a set of fault trees to test the abilities of my approaches to finding MCV. This is an area for further research that could benefit the reliability community.

3.4.1 Lessons Learned

The application of an EA to generating L-systems with the desired metrics was a limited success. Although the process showed that grammars could successfully be created by an EA, the fitness function proved to be problematic. Specifically, the use of *xfta* to find the size of the smallest MCS did not fit well with the application environment. When I shifted to using the Monte Carlo Elimination technique, progress was slow due to the significant increase in execution times.

xfta is designed to be used by FTA professionals who are working with a specific fault tree. It does not check memory requirements vs. system resources as it decomposes the problem and forms the internal representation to find MCS. Thus, if the problem is too large for the memory limits of the system being used, it will crash, leaving the analyst with no results. Analysts can dial this behavior by selecting the relevant size of the MCS to be found, effectively telling *xfta* to ignore anything larger. The memory crashes can also be mitigated by using a system with larger memory; however, *xfta* will then search for all MCS that it can find within the size limit; if that limit is too large, then execution time can grow unacceptably large. Balancing this behavior requires an understanding of the relative requirements for memory and execution speed of a particular fault tree. Analysts working with a *particular* fault tree would quickly develop a sense of the right settings to solve the problem at hand and whether larger resources are needed to find the answers required for the analysis at hand.

On the other hand, when used in a EA, there is no such intuition available. Trying to find settings that work for a large set of randomly generated fault trees without causing crashes or encountering extremely long execution times proved to be difficult and often resulted in EAs running for days or weeks, especially when the desired fault trees were both large and complex.

Overall, the most effective way to generate realistic fault trees was to use the EA to generate a set

of possible L-system grammars then manually adjust the rules to obtain the desired characteristics. This was sufficient for the purpose of generating benchmark problems for this research; however, there is much room for future researchers to focus on developing fully automated approaches. Since my objective was to create a test set that would provide a reasonable challenge for both the control methods and research methods, I chose to focus on finding rule sets that would serve the overall research goals rather than continually improving the EA in hopes of developing an all-purpose fault tree generator.

3.4.2 Component Locations

Since the focus of this research is on solving FTs with physical location constraints, it is not enough to generate the logical structure of the tree; the FTs must also have locations for every component. In this case, I chose to separate the process of generating the trees (described above) from the process of finding locations for each component in the system. The latter were generated randomly in the following way.

Once the fault tree is created, the top level system is placed within a bounding box that is approximately the size of an urban area, several kilometers on a side. For each level of the fault tree, the lower level nodes subdivide the bounding box as described below. A few Valid subdivision options are shown in Figure 3.2.

1. for each extremum (*i.e., minimum* or *maximum*) of each dimension of the system's bounding box (*x, y, z*), select a child node at random
    (a) if the node is a subsystem, set its corresponding extremum to that value
    (b) if the node is a component and its opposite extremum in that dimension is not set, set its corresponding extremum to that value

2. for each node in the system

   (a) for each dimension

      i. if the node is a subsystem and an extremum is not set, set it to a random value between the center of the bounding box and the corresponding system extremum

      ii. if the node is a component and one extremum is set, set the opposite extremum to a value within one meter away from it

      iii. if the node is a component and neither extremum is set, set them to random values between the system extrema and within one meter of each other

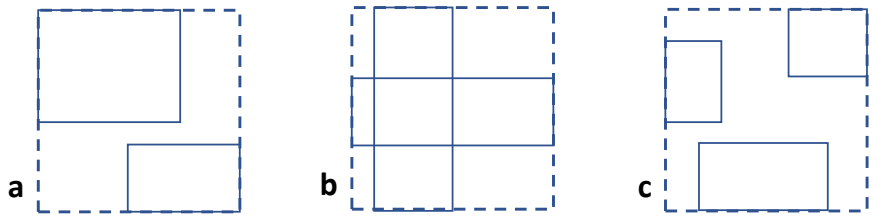

Figure 3.2: Illustration of possible means to subdivide system space

Once the component locations are assigned, they are incorporated into the fault tree XML format as a property of each component. That way, both logical and positional information about the FT is stored in a single file, but portions of that information are stripped out for the software (such as *xfta*) that does not recognize it.

3.5 Benchmark Test Set of Fault Trees for this Dissertation

After I established the ability of L-systems to create fault trees with scalable characteristics [Hanes and Wiegand, 2016], my next step was to generate the set of test cases for the experiment using L-systems. These test cases are fault trees that vary in multiple dimensions designed to stress

the capabilities of the methods. Also, showing that a method is applicable to different types of problems demonstrates its generality. These test cases are used for the research in the rest of this dissertation.

I measured the same set of metrics for each tree to verify that the larger trees have close to the same values as the smaller fault trees used in developing the methods. This shows that I did not bias the answers by changing fundamental characteristics of the fault trees as I moved to larger trees. The fault trees were stored in OpenPSA format and all software developed in this research reads this format. OpenPSA is based on XML and its creators include the developers of *xfta*.

Table 3.3: *L-System Rule Sets used to generate Fault Tree test set*

start:	A	start:	A
A:	(AND) BDE	A:	(AND) CCDD
B:	(OR) ACA	B:	(OR) EA
C:	(T)	C:	(AND) GDF
D:	(AND) BDBB	D:	(AND) AB
E:	(OR) BBBA	E:	(OR) AADBEA
		F:	(OR) BAAACC
		G:	(T)

Two rule sets were used for generating the fault trees explored using the methods described in Chapter 4. The rule sets are shown in Table 3.3 — the one on the left is referred to as "rule set A" throughout this dissertation, while the one on the right is referred to as "rule set B." Each rule set was expanded to 4, 5 and 6 levels to create a set of six fault trees for testing.

The metrics for these fault trees are shown in Table 3.4. Some, such as component count and size of smallest MCS (s_{MCS}), are more related to the complexity of the fault tree while others, such as operator proportion and out degree, are defined as structural in nature. These values are also visualized in the graphs in Figure 3.3 and Figure 3.4. Note that L-systems are deterministic in

Table 3.4: *Metrics for L-system rule sets expanded up to 7 levels (test set is expanded 4 through 6 levels)*

Rule	Depth	# comps	MCS count	s_{MCS}	Or proportion	Out Degree
A	2	11	12	6	0.5	3.5
	3	32	810	9	0.571429	3.21429
	4	89	382725	12	0.435897	3.25641
	5	269	1.63408E+13	15	0.53913	3.33043
	6	761	1.43941E+28	18	0.52071	3.24852
	7	2180	1.16205E+61	21	0.492662	3.28406
B	2	10	1	10	0	2.8
	3	30	144	18	0.307692	3.23077
	4	94	9604	36	0.243902	3.26829
	5	282	83109970944	64	0.24	3.248
	6	862	2.04107E+24	108	0.281167	3.28382
	7	2634	1.56722E+51	208	0.250644	3.26009

nature, so there is no need to expand these rules more than once; they will always produce the same fault trees

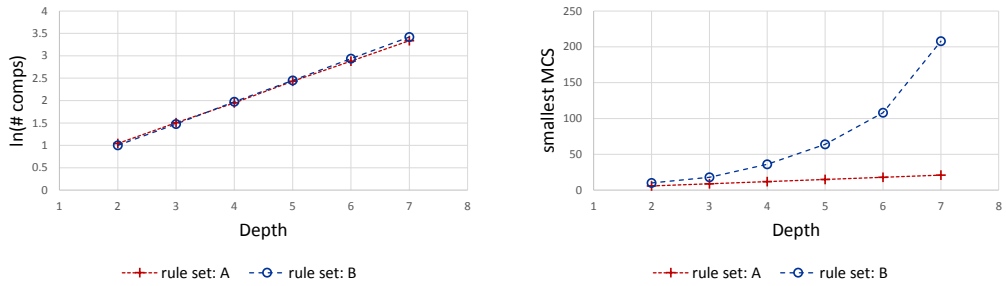

Figure 3.3: Graphs of fault tree complexity metrics for final rule sets showing the natural logarithm of component count (left) and size of smallest MCS (right) vs. expansion depth (X axis)

The intent of producing these fault trees was to vary the size and complexity in three levels for each one. I used the resulting rule sets to generate the test set of fault trees used for testing the

performance of the methods. My focus was not to find rule sets that conformed to specific metrics, but rather to test the effectiveness of the MCS methods acting on a test set of fault trees that have consistent metrics at different scales.

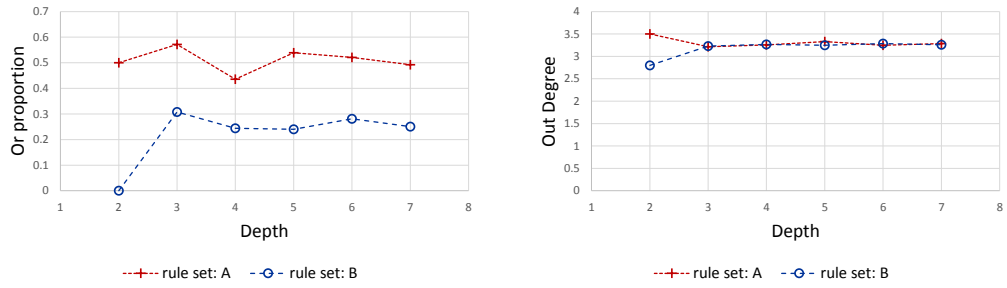

Figure 3.4: Graphs of fault tree structural metrics for final rule sets showing the proportion of operators (left) and out degree (right) vs. expansion depth (X axis)

Figures 3.3 and 3.4 show that the complexity metrics (component count and smallest MCS) grow in predictable ways and that the structural metrics (operator proportion and out degree) stay within 10% of the average value after the fourth replacement cycle. These results show that it is indeed possible to generate fault trees that grow in predictable ways and that maintain consistent structural properties at scale.

Status of H0: Hypothesis supported by the data.

The caveat is that there is sometimes more than 10% variation in the first 2 to 4 replacement cycles, then the metrics generally converge toward stable values. That does not make these fault trees unusable; it simply dictates that analysts must be aware of the differences and ensure that the fault trees they use meet their requirements, particularly when the depth of expansion is small.

3.6 Conclusion to Benchmarking

These results show that L-systems can be defined to generate fault trees that conform to desired characteristics. This initial phase of the research produced a set of fault trees with consistent characteristics that will also provide a challenge for the methods.

As can be seen from the metrics in Table 3.4, Rule set A produces fault trees with moderate redundancy that one might see in a factory or high risk system where there is a focus on maintain a high degree of reliability. Rule set B shows very high redundancy that is probably unrealistic for real world systems, especially after 5 or 6 replacement cycles. However, this makes it a good case for testing the performance of the methods on very difficult cases.

CHAPTER 4: APPROACH TO FINDING CUT VOLUMES

To find vulnerabilities caused by the proximity of critical infrastructure components, I developed approaches to search for minimal volumes of components that kill fault trees using two methods not traditionally applied to this field, cluster analysis and evolutionary algorithms. To provide a baseline for comparison and experimental control, I also applied traditional FTA methods through the direct exercise of fault trees in a discretized search space and through the use of a deterministic FTA tool called *xfta* [Rauzy, 2012b] to find MCS. The results generated by *xfta* were examined using cluster analysis to form a hybrid (deterministic/stochastic) method that served as a comparison for assessing the effectiveness of the two other methods. In this chapter, I provide formalization of the underlying problem as well as detailed descriptions of the control methods, cluster analysis, and evolutionary analysis. I begin with an informal description of the problem.

4.1 Map of Urban Vulnerability

Returning to the urban disaster scenario described in Section 1.2.1, an urban area contains many interconnected systems, all of which contribute to keeping the city functioning. I assume, for the sake of simplicity, that a disaster centered at one location with a radius of effect, such as an industrial accident or terrorist attack, will destroy all of the components contained inside that radius; for this research, probabilities, delayed effects and partial damage are ignored. The question I am attempting to answer is, "where in the city will the smallest radius of effect defeat the top level system of systems?"

The simplest approach to solving this question is to search a grid of locations to find single volumes that defeat the top level fault trees. This is analogous to finding a single point of failure; the differ-

ence is that a volume located at any point in the space can be expanded until it encompasses enough components to kill the system. However, doing so will often require a volume that comprises a significant percentage of the entire volume occupied by the systems and is much larger than the effect of a typical explosion. Only those volumes that are similar in size to some likely physical phenomenon are of concern. After demonstrating the proof of concept using the grid approach, all of the other methods were used to search for combinations of two, three or more volumes since these searches represent challenging analyses and realistic situations for which planners would wish to prepare.

If we look at the disaster in this manner, then the city is most vulnerable where the cumulative volume of the zone causing damage is smallest. The smallest such volumes will be called "minimal cut volumes" (or MCV) throughout this research as an analogy to minimal cut sets in classic FTA. For this research, I define MCV as a combination of spheres with equal radius such that reducing the radius will cause the volume to no longer defeat the top level system. Although it is possible to search for spheres of different sizes, I chose to simplify the problem for this initial exploration of the concept. Future research could focus on generalizing this problem class in several ways.

Conceptually, one might explore an urban system by discretizing the space as illustrated in Figure 4.1. This approach starts with a set of evenly spaced points in the urban area. For accuracy, the spacing should be less than the radius of the effect being investigated in the study, although a lower bound would be needed to prevent very large execution times.

I developed a program to find the smallest radius of damage at each point in the grid; the algorithm is described in Subsection 4.2.1. The results produced by this program can be visualized by coloring a cell according to the size of the radius: red meaning a small radius and blue meaning a large radius. A "heat map" derived by visualizing the grid of effective radii for one of the systems in the trade space is shown in Figure 4.2.

Figure 4.1: Grid of search locations superimposed on an urban layout

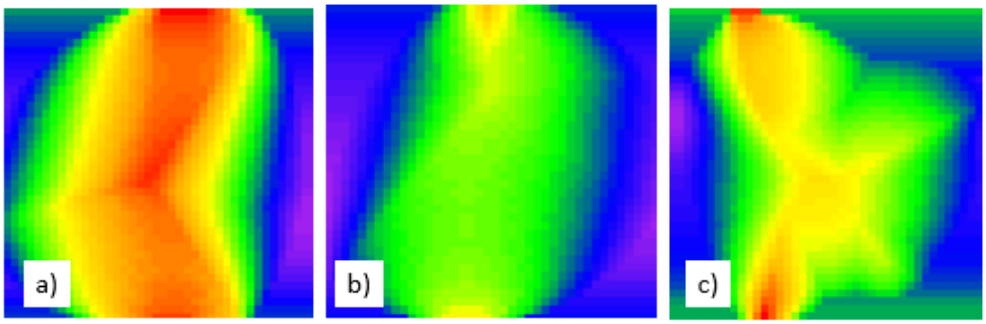

Figure 4.2: "Heat map" derived for a fault tree with randomly generated locations

A map such as this for a real city would provide urban planners valuable information they could use to evaluate and eliminate vulnerabilities in the infrastructure that have the potential to cause failures in a disaster situation. Given such a map, the analyst could then evaluate all cases where the effective radius is lower than the radius for some phenomenon of concern.

Of course, in urban planning, one would concentrate on the regions with an effective radius smaller than some characteristic effect size. The desired size could be defined by reviewing historical ter-

rorist actions or industrial accidents. Then the method could be applied to the urban area to identify "danger zones" where effects of that size could cause high level failure to the infrastructure. Once these "danger zones" have been identified, a plan could be developed to move critical components to new locations or protect them so that they are less likely to be damaged by an effect at a single location. This process would then reduce the vulnerability of the urban services to common cause failures induced by the disaster.

Naturally, there are special cases. A tornado would create a "corridor of effect" rather than a radius; storm surge from a hurricane would tend to affect all low-lying components along the seaward side of a city. However, the same principle can be extended to these cases. This same information can also be calculated in three dimensions to provide a full picture of the vulnerabilities above and below ground as well as at the surface.

As described so far, considering single locations in a discrete space, this problem could be solved in polynomial time by finding the minimum radius of effect at each grid location on the map. However, searching for *combinations* of volumes that kill the top level system is exponentially more time consuming to solve directly as the number of locations increases, even in a discrete space. If we consider the problem of searching for multiple locations in a continuous search space, it becomes even more challenging. Thus, I chose to approach the problem from several directions in the hope that one of them would prove to be clearly superior to the others.

The methods explored in this research can be viewed as attempts to find the "danger zones" in the graph of effective radii without having to compute all of the combinations for the entire urban zone. I used the research methods to search for k-event combinations that defeat the systems in the urban system, for $k = \{2, 3, 4, 6, 8, 10\}$. This conceptual experiment suggests at least two metrics for evaluating the utility of the methods under investigation. If these methods cannot find smaller "danger zones" than the control method or if they cannot find them in a useful period of time, they

would not be considered adequate for finding and mitigating urban hazard zones.

To find these MCV, I pursued multiple methods in the hopes of finding a general method. I will outline them here for clarity, then expand on each step in the discussion that follows.

1. Extend standard methods (*i.e.*, deterministic with cluster analysis) to create a control to define baseline performance.
   (a) Create a grid of points and calculate effective radius at each grid point.
   (b) Run deterministic algorithm to find MCS, then search for clusters in the resulting sets.
2. Define clusters of components in the urban space, then determine whether killing components in some combination of clusters defeats the top level urban system
3. Execute EA to find combinations of locations that collectively cause failure of the top level system.
4. Calculate performance metrics and compare methods to the control method.

4.1.1 Formal Statement of Problem & MCV

Formally, the location-based fault tree analysis problem and the metric used for evaluating solutions can be stated as follows:

Definition 2 *A **location-constrained fault tree** is a* 5*-tuple defined as follows.* $FT := \langle C, O, T, I, L \rangle$*, where:*

- C *is the set of* components *that can be affected by spontaneous events;*

- O *is the set of* operators *that aggregate events from lower-level components;*
- $T :\mapsto Y$ *is a function mapping an operator to a particular type of operation;*
- $Y := \{\text{AND}, \text{OR}, \text{K-OF-N}\}$ *are the types of operations;*
- $S := C \cup O$ *is the set of all* sub-system *roots in the whole system;*
- $I : O \mapsto \mathcal{P}(S)$ *is a function that describes the inputs of each operator;*
- $L : X \mapsto \mathbb{R}^d$ *is a function that gives a* location *for each component in some space of dimensionality* d*.*

Note from the above definition that $\langle S, I \rangle$ must form a directed, acyclic graph with a single *root*, which is denoted by r.

Definition 3 *A* ***locations-based failure event*** *is a* 3*-tuple defined as follows.* $E := \langle V, \varrho, \Delta \rangle$*, where:*

- $V \subset \mathbb{R}^d$ *is a finite subset of locations of physical events;*
- $\varrho \in \mathbb{R}$ *is the* radius of effect *of these locations;*
- $\Delta : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ *is a function giving a* metric distance *between two locations.*

Definition 4 *Given a* particular *locations-based failure event* $e = \langle v, \rho, \delta \rangle$ *and particular fault tree* $f = \langle c, o, t, i, \ell \rangle$*, the* ***failed component set*** $FCS_{e,f}$ *is the set of all components that fall within* ρ *distance of any fault location:*

$$FCS_{e,f} := \{c_j \in c \mid \exists v_i \in v \textit{ such that } \delta\left(v_i, \ell(c_j)\right) \leq \rho\}$$

Let $\pi_e\left(FCS_{e,f}, x\right)$ be a function indicating whether component x fails given a particular $FCS_{e,f}$. This function is defined as follows:

$\forall c_j \in c$, let $\pi_e\left(FCS_{e,f}, c\right) := 1$ if $c_j \in FCS_{e,f}$

$\forall o_j \in o$ and where $t(o_j) =$ AND, let

$\pi_e\left(FCS_{e,f}, o_j\right) := \wedge_{x \in I(o_j)} \pi_e\left(FCS_{e,f}, x\right)$

$\forall o_j \in o$ and where $t(o_j) =$ OR, let

$\pi_e\left(FCS_{e,f}, o_j\right) := \vee_{x \in I(o_j)} \pi_e\left(FCS_{e,f}, x\right)$

$\forall o_j \in o$ and where $t(o_j) =$ K-OF-N, let

$$\pi_e\left(FCS_{e,f}, o_j\right) := \begin{cases} 1 & \text{if } \sum_{x \in I(o_j)} \pi_e\left(FCS_{e,f}, x\right) \geq k \\ 0 & \text{otherwise} \end{cases}$$

Since r is the top-most operator at the root of the fault tree, if $\pi_e\left(FCS_{e,f}, r\right) = 1$, then $FCS_{e,f}$ forms a **cut set**. Given all this, the optimization problem is as follows. Find some location-based failure event such that:

$$\arg \min_{e=\langle v,\rho,\delta\rangle \in E} \rho|v|$$

subject to:

$$\pi_e\left(FCS_{e,f}, r\right) = 1$$

Any solution to this problem is called a **minimal cut volume**. A minimal cut volume is guaranteed to cover components that represent a cut set, though that cut set may include components that are not strictly necessary for system failure since ancillary components may be physically contained in the volume.

Conversely, I assert that for any given MCS, there exists a set of location-based failure events e such that

$$MCS \in FCS_{e,f}$$

I say that each of these events *contains* the MCS.

4.2 Control Methods

Two methods were used as a control to provide a comparative basis to evaluate the performance of the research methods. The first was not explored as a serious candidate for finding the smallest MCV, but as a means to illustrate the combinatorial issues that arise when approaching the problem in a naive way. The second is an application of real world qualitative FTA software combined with a stochastic post-processing stage that organizes MCS into geographic clusters to find vulnerabilities in the system. This method proved to more than just a comparative benchmark for the other two methods — it performs well in many cases and will be shown to be the preferred method under some circumstances.

4.2.1 Heat Map

The first control method is described in the conceptual outline at the beginning of this chapter. Using the bounding box of the top level system, a grid is defined and the minimum effective radius calculated at each point on the grid. These radii are plotted in a heat map that can be used to visualize the nature of the vulnerabilities in the urban space. The goal was to determine the overall granularity of the grid that would be needed to discern an effect in the trade space.

For each point in the grid, the method finds the smallest radius of effect that will defeat the top

level system. To do so, it starts with a radius large enough to contain all components in the system — if the effect kills every component, then obviously that kills the system. Then the radius is divided in half and a set is defined containing all components inside the new sphere. If this set defeats the top level system, then it is a cut set and the radius is cut in half again until a set is found that does not defeat the top level system. If the set does not defeat the top level system, then the radius is increased to halfway between the current radius and the smallest radius known to defeat the system. The radius is varied in step-wise fashion, increasing or decreasing depending on the success of the previous step. Once the difference between the largest and smallest radii found is less than 5 meters, the process exits and returns the smallest radius that encloses sufficient components to defeat the system. We define this value as the **effective radius** at that point. This process is also used by the EA as part of the fitness function for evaluating genomes as discussed in Section 4.4.

This process looks something like Figure 4.3, where each circle represents a sphere and the numbers show the order in which they are evaluated. In this figure, the yellow circles indicate a radius that causes defeat to the top level system; they are followed by a smaller radius. Similarly, the blue circles indicate a radius that does not defeat the top level system; they are followed by a larger radius. In this illustration, the next circle would fall halfway between circles 3 and 4.

If this process is repeated for every point on the grid, the result can be visualized by showing the effective radius as a color continuum, where red indicates the smallest values and transitioning through yellow and green to blue as the effective radius grows larger as seen in Figure 4.2. The "heat map" indicates areas where the urban zone is potentially vulnerable to the effect being studied.

As mentioned before, this approach only works for single locations. To generate results for two or more locations simultaneously, one would perform the shrinking radius operation on two grid

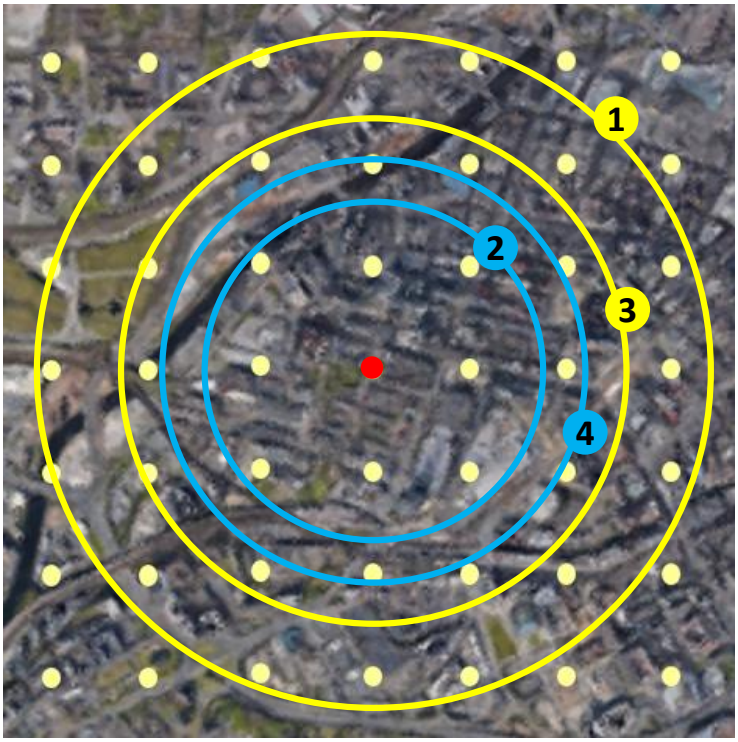

Figure 4.3: Urban space showing calculation of radius of effect

locations at the same time. Taking the locations that produce the smallest radius found would identify the vulnerable locations for $k = 2$. It would also require n times as much computation time (where n is the total number of points in the grid) since the operation must be performed for every pair of grid points. Expanding to three locations would require n^2 as much computation time. Since the time for single location grid search was over an hour, it is clear that this would be an extremely long process for two locations and impractical for three or more. This confirmed my intuition that the grid approach would be unsuitable as a control for comparing with the research methods for $k > 1$.

4.2.2 *Extending Traditional Analysis Tools*

A more realistic way to approach this problem using traditional tools is to use software implementing one of the deterministic algorithms for qualitative analysis. For this research, I used *xfta* [1] It is based on a deterministic algorithm [Rauzy, 2012a] and developed by current researchers in the field of FTA [Baklouti et al., 2017] , so it is a good choice for a "state of the art" implementation of the deterministic approach. Since this is based on FTA software, I refer to it as the **Analytical Method** (AM) throughout this dissertation.

In this approach, I used *xfta* to find MCS for the top level system. For small sized systems, it was able to find all MCS; but for medium systems, it was necessary to limit the size of the cut sets because of both run times and storage capacity. Therefore, many possible solutions were missed. For the larger systems, I used a backup method of finding the MCS as described in the next subsection; however, it was not able to provide a complete set of MCS either. This phenomenon and the actual fraction of MCS used will be discussed in more detail in Section 5.1.

4.2.3 *Monte Carlo Elimination*

For some of the fault trees, *xfta* was not able to find MCS within the limits of the hardware available. Therefore, I developed an alternate method so that I would have MCS to use for the second step of this method.

The alternate approach eliminates components using a Monte Carlo (random) draw to compare with component indices. The elimination method starts with a set in which all components are damaged (*true*), then randomly eliminates them from the damaged set (*i.e.*, returns them to an undamaged state) until it settles on a MCS. Initially, the top level system evaluates to failed, since

[1] *xfta* can be downloaded at no cost from **http://altarica-association.org/contents/xfta.html**

all components are failed. It is re-evaluated after each elimination to determine whether the reduced set still causes the system to fail. If the top level system still evaluates as defeated, the component is left undamaged. If the top level system is no longer defeated, the component is returned to the failed set and marked so that it will not be selected again. Then, another component is randomly selected and the evaluation steps are repeated. After all components have been eliminated or marked, the remaining components form a MCS. It is guaranteed to be minimal since the process has shown that removing any one of them will return the top level system to functional. However, the cut set may not be the smallest possible MCS for this fault tree, so it is necessary to execute the process thousands of times; all MCS found in this manner were combined into a single file of MCS. Comparison with MCS found by executing *xfta* for the same fault tree showed that the elimination technique generally found the smallest MCS with fewer than one thousand attempts.

For this research, the goal of the Monte Carlo Elimination was to generate at least 20,000 MCS (of any size) for use in the next step of the AM. Thus, the Monte Carlo sampling was repeated numerous times to generate independent results, then these results were merged into a single file for later use. Although the choice of 20,000 was arbitrary, it resulted in a sample size that was empirically comparable to the number of solutions found by *xfta* before crashing. In some cases, *xfta* found less than 10,000 MCS before it exceeded memory limits or ran so long that it did not produce results in reasonable time (less than a day). In other cases, it found over 100,000 MCS for a single fault tree — however, the resulting file size made the cluster algorithm time-consuming, because it performed cluster searches 10 times for each MCS in the file.

4.2.4 Clusters of MCS Components

Once the MCS are collected, regardless of the method by which they are produced, the next step is to find volumes that contain them. For any given evaluation, k must be defined: for my research,

I used $k = \{2, 3, 4, 6, 8, 10\}$. These values were intended to generate a baseline set of radii for comparison with the research methods — the larger values are also likely to exceed the number of simultaneous events that might be encountered in a true disaster, so they provide a probable upper bound for application to real world problems.

Each evaluation searched for k clusters in the set of components for *each* MCS. Algorithms to find clusters of data in multi-dimensional data sets have been an active area of research for many years [Han et al., 2001]. This has application both for geographical data sets and data mining applications. For this application, I selected the k-means clustering algorithm for finding k clusters in each MCS. The algorithm will be described in more detail in the next section.

This overall approach to finding the MCV is illustrated in Figure 4.4. Figure a) shows the (x, y) locations of all components in this instance of the fault tree, Figure b) shows the locations of the components in a single MCS superimposed as red dots over the first image, while Figure c) shows the clusters found by the k-means algorithm as circles on the image. Although there are several clusters with differing radii, the largest radius is used as the figure of merit for this solution. This process would be repeated for every known MCS until the one generating the lowest figure of merit is selected.

Although finding the *best* centroid location is NP-hard [Garey and Johnson, 1979], the k-means algorithm iteratively converges on an answer that is sufficient for this research since optimal clustering was not the focus. The physical size of the resulting clusters was converted to a sphere, from which I calculated the effective radius for those conditions.

While deterministic methods find the smallest MCS by count of components, it is possible that there exist MCS with more components that fit within a smaller volume. However, as the systems grow larger, larger MCS will be ignored to save run time. Therefore, the deterministic approach will trade fidelity in finding the "danger zones" to reduce the run time (and to save disk space).

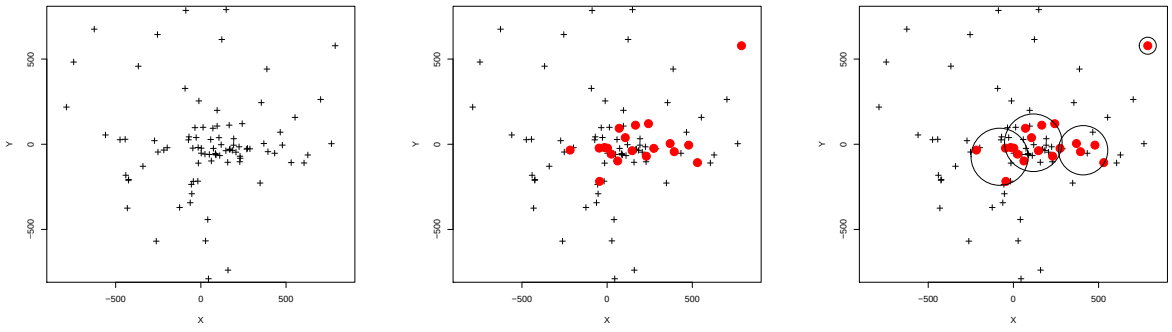

Figure 4.4: Set of figures showing the progression of the AM for a single instance of one fault tree; this example uses Fault Tree A-04.

While this is an acceptable trade for most FTA applications, it results in the deterministic approach missing a portion of the search space. As the systems grow larger in terms of component count, the number of MCS found is much less than one percent of the total MCS known to exist. Thus, the likelihood grows that an MCS exists among the 99.9999% of MCS not evaluated that fits more components into a smaller volume. That is, the larger the fault tree, the more likely it becomes that this method will miss small MCV since it filters the majority of MCS to avoid run time penalties. This leads to my first hypothesis regarding the performance of the methods described in this dissertation.

Hypothesis H1: The control method will perform less well for larger fault trees than for smaller fault trees.

Success criteria: Results showing that the radius of the MCV generated by the control method increase as fault trees increase in size as measured by number of components.

In contract to the grid approach, this control method was developed with the idea that it was a candidate for a genuine means to use qualitative analysis to achieve the research goals. It takes ad-

vantage of deterministic methods widely used in FTA to identify the MCS to be evaluated. Because the volumetric evaluation uses the stochastic k-means algorithm, the method is actually a hybrid of deterministic and stochastic algorithms. This hybrid performs well for the cases investigated and turns out to be a good measuring stick for evaluating the research methods that follow.

4.3 Cluster Analysis

The next method is, in some ways, the reverse of the AM. It searches for clusters in the full component set *before* checking whether subsets of components defined by combinations of clusters defeat the top level system. Therefore, it is referred to as the *Cluster Method* (CM) throughout this dissertation. A visual comparison of the process followed by the AM and the CM is shown in Figure 4.5.

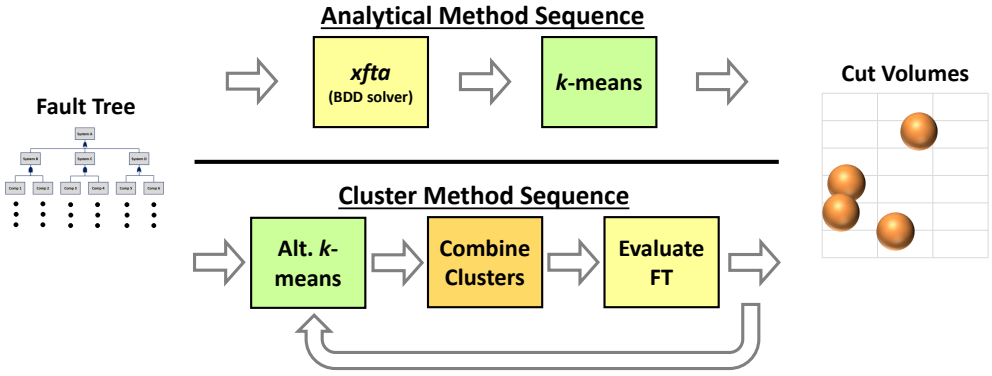

Figure 4.5: Conceptual comparison of AM vs. CM

The first objective for this approach was to divide the components into clusters based on their locations. Clustering is a research area in itself; there are many clustering algorithms suited to different types of problems [Han et al., 2001]. Although the k-means algorithm has been widely replaced in practical use by algorithms tailored for specialty areas such as finding irregular shapes

and heterogeneous-sized clusters, it is well-suited to dividing a space into groups of items that are regularly shaped and similarly sized. The resulting clusters can be characterized as volumes in which every point is in a cluster defined by its nearness to the centroid. k-means normally targets a predetermined number of clusters, iteratively refining them until they are roughly equal in size and their membership converges as illustrated in Figure 4.6. In each iteration, points are compared with each centroid and moved to the nearest cluster; the centroids are then recalculated from the new set of points defining the cluster. Sometimes, clusters will end up with no members, particularly when there are many clusters, effectively reducing the number of clusters. In my implementation, such empty clusters were reintroduced by dividing the cluster with the largest radius and assigning half the points to the empty one.

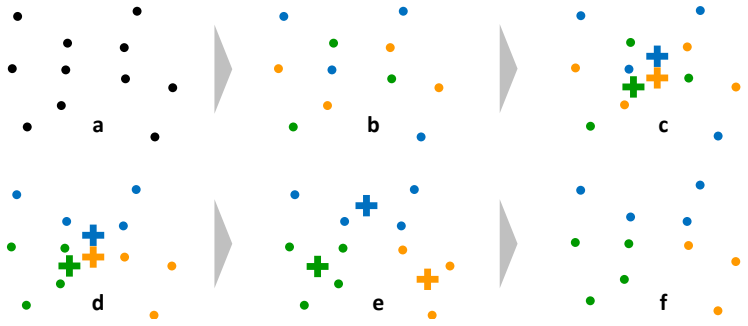

Figure 4.6: Illustration of k-means algorithm for a small set of points

This innovation inspired a modification to the normal k-means algorithm capable of targeting a geometric size. This modified algorithm periodically adds clusters until their size is below the desired maximum size. Each new cluster is created by dividing the largest cluster in two. This variant reduces the size of the clusters until they reach a size that emulates the desired size of the effect. This contrasts with the normal use of the k-means algorithm, which focuses on achieving a specific number of clusters.

Following this, the clusters were combined into sets of k clusters (where k refers to the number of

centers of effect for the execution). All possible combinations were tried in succession to identify damage volumes capable of defeating the top level system. If combinations of clusters were found that defeated the system, then the modified k-means could be executed to search for smaller clusters until a lower bound size is discovered. Conversely, if no such combinations were found, it could be executed to find larger clusters.

Hypothesis H2: Partitioning the graph into combinations of k clusters will find smaller cut volumes than those found by traditional methods.

Success criteria: Radius of MCV for k-location cut volumes found by dividing the component set into clusters is statistically the same as or smaller than those found by control methods.

Although a follow on hypothesis was proposed at the outset, it was not pursued due to reasons that will be discussed in Chapter 5.

4.4 Evolutionary Algorithms

The other approach selected for finding MCV consists of using heuristic search techniques to explore the trade space of possible solutions. These methods have been created and refined through artificial intelligence (AI) research.

Stochastic heuristics have typically not been successful or necessary in traditional FTA. Deterministic approaches to FTA run very quickly and can produce a complete set of MCS, assuming sufficient computational resources are available and the fault trees are not too large. Stochastic heuristics cannot compete with the speed of deterministic solutions and are not guaranteed to find optimal solutions. Initially, this would seem to indicate that stochastic heuristics are not reliable candidates for solving problem relating to fault trees.

However, we have established that deterministic FTA methods work because they assume independence. This implies that deterministic algorithms are not able to *identify* dependencies induced by proximity of components in the fault tree. The lack of determinism of EAs is the factor that makes AI heuristics attractive for my purposes. They have been developed to deal with ambiguity and to find patterns in extremely large search spaces that have not been solvable using deterministic analytical algorithms. It should be noted that the AM is not fully deterministic; it uses k-means, which is a heuristic. The nature of the problem defined in Subsection 4.1.1 requires some level of "guesswork", since fully deterministic approaches do not currently exist.

Heuristics are necessary in this case because solving for all MCS is #P complete, as mentioned earlier. The time to explore all possible MCS grows exponentially larger with the size of the fault tree, so there will always be problems too large to be solved using deterministic algorithms on existing hardware. Consequently, heuristics such as EAs are a valid option for finding approximate MCV in the physical space of the fault trees.

EAs (see Figure 2.2 on page 2.2) start by generating a random set of solutions and progressively refining them using stochastic search operators (known as *genetic operators*) and evaluating their suitability (known as *fitness*) for solving the problem. The best solutions are retained for continued refinement while the remainder are discarded. This progresses until a predefined end point or until solutions converge toward an apparent best answer. Readers may be familiar with "genetic algorithms" [Mitchell, 1998], which are a specific type of EA [De Jong, 2006]. One of the key advantages of EAs is their flexibility — through modifying the genetic operators and fitness function, one can adapt them to almost any problem. The challenge lies in finding the right operators for any given problem.

The specific algorithm that I used is described in more detail below:

1) **EA Representation:** Each candidate solution in an EA is represented by a genome. The ge-

netic operators combining and altering them are called *crossover* and *mutation*, respectively. The specification for how genomes are encoded and how the genetic operators work constitute the EAs representation.

For this problem, the genome is a vector of real values representing a set of locations defining the centers of physical damage (the event). Each 3D position triplet (x, y, and z) is referred to as a gene, and there are k such genes, where k is an input parameter for the search. To be consistent, the EA was run using the same values of $k = \{2, 3, 4, 6, 8, 10\}$ as were used with the AM. Crossover works by selecting one gene from each of two genomes, drawing a line connecting them, then randomly selecting a point along that line. The mutation function selects one dimension (x, y, or z) of one gene in the genome and randomly increases or decreases it by a uniformly random value within a range that anneals as the optimization progresses. Initially, the annealing size is set to half the bounding size of the dimension in question as determined during the formation of the fault tree. This range is decreased each generation according to a linear function for n generations until it is near zero in the final iteration; it would reach zero in the $n + 1$ generation. In addition to the above operators, a hypermutation mechanism was added: in each generation, twenty-five individuals are created at random (as in the initial generation) from the bounding box of the fault tree.

2) **EA Fitness:** The fitness of each genome (the quality of the solution) is estimated by a binary search to find the minimum volume of the spheres, as described for the grid approach in Section 4.2.1 and illustrated in Figure 4.3. In this case, the radii of all spheres in the genome were modified simultaneously to generate a cumulative volume of damage. The final fitness value is estimated as the smallest successful radius for the combined volume defined by the set of (x, y, z) positions in the genome. The goal of the EA was to minimize this value over the entire fault tree volume.

3) **EA Dynamics & Selection:** My EA employs both parent and survival selection. Parents are

selected using a variety of tournament selection: four individuals are selected uniformly at random from the population and the two most fit (smallest fitness value) among those four are then chosen to produce two children via the crossover and mutation operator described above. To maintain diversity in the population, I rejected children that were too close to other genomes already in the population. In this case "too close" means that they shared half of their genes (*i.e.*, effect locations) or more as determined by the Euclidean distance between locations. If any two locations in the two genome are within one meter of each other, they are considered to be shared, in which case the child is rejected and a new one produced via crossover and mutation.

Survival selection uses a $(\mu + \lambda)$ strategy: 500 parents were used to produce 400 children (375 via crossover and mutation, 25 via hypermutation), then the most fit 500 were selected from those 900 individuals to survive to the next generation.

The $(\mu + \lambda)$ dynamic was chosen primarily because of the time required to compute the solutions and a desire to retain the "best" answers found in the process. The hypermutation and distance checks discussed above were implemented in order to mitigate the possibility that such an aggressive selection method can lead to premature convergence on local minima.

The EA approach to finding the MCV is illustrated in Figure 4.7. Figure a) shows the (x, y) locations of all components in this instance of the fault tree, Figure b) shows the locations of the spheres of damage for a single genome in one generation as circles over the first image, while Figure c) shows the damaged components as red dots forming a cut set (though not necessarily minimal). Many such damage spheres are created randomly and evolved through several generations until they converge on a lower bound radius. This lower bound is used as the performance metric for comparing with the AM.

The goal was to ensure that the EA converged in all cases. So each execution of the EA ran for 40 or more generations depending on the size of k; the maximum number of generations was selected

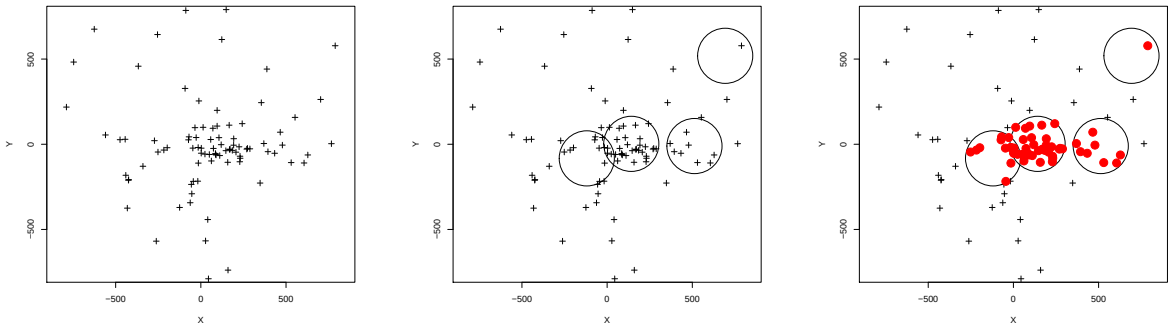

Figure 4.7: Set of figures showing the progression of the EA for a single instance of one fault tree; this example uses Fault Tree A-04, the same used in Figure 4.4 so the two methods can be compared.

by observation of preliminary experiments to ensure that convergence would occur. Specifically, the maximum number of generations was $15k + 10$.

My early work in this area [Hanes and Fay, 2015] focused on using genomes that consisted of strings of binary variables to represent whether components were damaged or not. This approach proved problematic because the fitness function was also binary; either the top level system was killed or it was fully functional. This situation negates one of the primary advantages of EAs, which is that they can gradually approach improved fitness values by making changes to the population of genomes. Thus, one of my original research goals was to find a fitness function that avoided the ”all or nothing” nature of the binary kill values. This leads to a key hypothesis related to the use of EAs:

Hypothesis H3: Using a fitness function based on radius of MCV will enable the EA to converge on MCV solutions for an arbitrary number of locations across a range of fault tree sizes.

Success criteria: EA with radius for fitness function shows convergence when used for fault tree

test set.

An important measure of how well a GA performs is the time required for it to converge to a solution. Convergence, in this case, is defined by the state in which the size of the best solution changes minimally from generation to generation.

I executed the EA representation described above for $k = \{2, 3, 4, 6, 8, 10\}$, by adding multiple genes to each genome. Rather than minimizing a radius of effect for each gene, the EA minimized a single radius for all locations in the genome. My goal was to define an EA that would exhibit tractable execution times as the size of fault trees increased. By "tractable," I mean that the times do not grow faster than polynomial time. This leads to my second hypothesis related to the use of EAs for this problem.

Hypothesis H4: An Evolutionary Algorithm with a radial fitness function will be able to converge on k-event MCV for complex fault trees with location constraints in polynomial time for different values of k as the size of the fault trees increases.

Success criteria: EA converges in polynomial time as fault trees increase component count.

4.5 Conclusion to Methodology

Historically, EAs have had difficulty solving fault trees, due primarily to the binary nature of the fitness function, which limits their ability to distinguish levels of quality for genomes that "almost" kill the top level system. However, the problem of finding MCV for components anchored in Euclidean space is distinctly different from finding MCS under the assumption of independence. Consequently, my expectation was that non-traditional solution methods would prove to be applicable when contrasted with the classic means of solving fault trees. In particular, for EAs, the

key to finding an effective representation was the ability to use a fitness function that has smooth gradient values rather than a binary output value. A key difference in this fault tree research is the focus on minimizing the effective volume of a cut set (the MCV) rather than searching only for MCS.

My intent for the cluster-based approach was that it would provide a fast-running method for sorting the components into useful groups from which combinations would generate cut sets. The k-means algorithm used in this approach also turned out to be an important element in creating the control method.

These considerations led to my overarching research goal, which was to develop a method capable of identifying vulnerable zones in a physical space created by the proximity of components to each other. Thus, my grand hypothesis was that I would find a means to do so.

Hypothesis H5: Evolutionary algorithms or clustering techniques, properly configured, can be developed that find smaller MCV in a complex system of systems when compared to traditional (*i.e.*, deterministic) methods of fault tree analysis augmented with heuristic clustering algorithms.

Success criteria: At least one of the research methods finds smaller MCV for medium and large fault trees than control methods.

This hypothesis was developed so that it would be challenging to achieve. I deliberately chose a software package based on a robust implementation of deterministic algorithms to form the basis of the control method in order to provide competitive, and therefore useful, metrics for the research methods. The reliability engineering community has developed these algorithms through decades of research and has confidence in the capabilities encapsulated in deterministic solution techniques. Therefore, an approach that does not take them into consideration and compare favorably with them would not be taken seriously in that community.

CHAPTER 5: RESULTS

The research proceeded in several stages. The results of the first stage, generating fault trees to create the test set, are described in Chapter 3 starting on page 45. The next stage was to develop a control method based on existing FTA software (AM) and apply it to the test set. Then, a method based on cluster analysis (CM) was developed and used to search for MCV in the component space. Finally, an evolutionary algorithm (EA) was implemented and used to identify MCV for the fault trees. Both the CM and the EA were compared to the AM to assess their viability for finding useful solutions to the problem.

5.1 Control Methods

The primary value derived from implementing the first control method, the grid approach, was that it provided a valuable means to visualize solutions to the problem (see Figure 4.2 on page 54). Although the method is a straightforward interpretation of the problem statement, it is by far the most time-consuming, with execution times in hours for $k = 1$. I did not attempt to run it for higher values of k due to the anticipated greater execution times for $k > 1$. Its primary value from a research perspective is to show clearly the need for more sophisticated algorithms. The computational time to execute the grid approach will grow as $O(n^k)$, where n is the number of grid locations and k is the number of locations in each combined event.

On the other hand, implementing the second control method, the AM, proved to be valuable for the overall direction of the research. First, the use of *xfta* provided a link with solution techniques widely used within the field of reliability engineering. Learning to use this tool and exploring the results that it produces provided insight into the process of calculating and interpreting MCS,

which are the classic approach for finding vulnerabilities in fault trees.

Second, the results described in this chapter will show that this approach can produce good solutions for the problem of finding MCV. Thus, it is a viable method for calculating MCV in some circumstances and should be considered by anyone applying this research to real world systems.

I found that the r_{MCV} of this method degraded as fault trees grew larger. Specifically, its performance dropped off as the component count approached 1,000. Some possible reasons for this will be discussed later. The performance of this method can be seen in Figure 5.1. Some thoughts regarding ways to improve its performance will be discussed in the Conclusion.

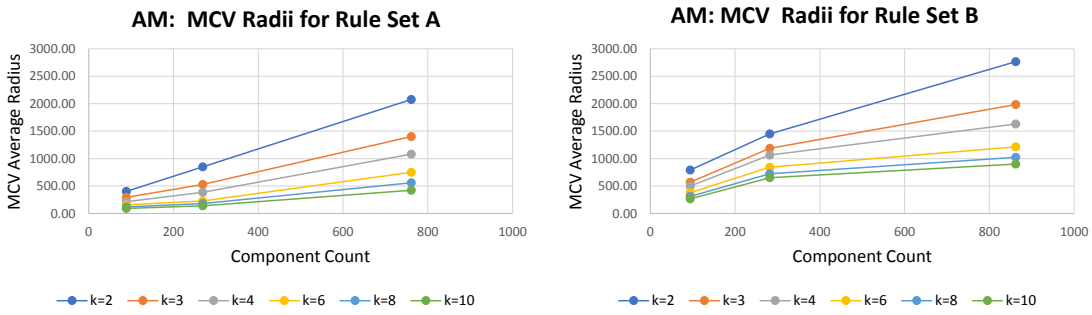

Figure 5.1: Plots showing the radii found for each fault tree (averaged across all instances) by AM at different values of k. Vertical scales are identical for purpose of comparison.

This figure shows the average radius found using the AM for all iterations of each fault tree at specified values of k, thus each marker represents 100 iterations of the indicated method. Rule Set A (depths 4, 5, and 6) is shown on the left and Rule Set B (depths 4, 5, and 6) is on the right.

Although there are differences in the progression and the shapes of the curves, it is clear that, under all conditions, the radius of the MCV increases as the size of the fault tree increases. As can be seen in these plots, the radii found by the AM consistently increase when analyzing larger fault trees and there are no examples where the radii decrease as the fault tree increases in size. These

data can be used to test hypothesis H1.

Repeat H1: the control method will perform less well for larger FTs than for smaller FTs.

To test this hypothesis, I construct a null hypothesis stating that the method produces equal or smaller radii for larger fault trees. In this case, $r_{n,k}$ denotes the radius found for a fault tree with n components for a given value of k. So the alternate hypothesis is that the method generates larger radii for larger fault trees.

null $H1_0$: $r_{n,k} \geq r_{m,k}$ where $n < m$, assuming similar structural metrics.

alternate $H1_a$: $r_{n,k} < r_{m,k}$ where $n < m$, assuming similar structural metrics.

With this definition, I calculated p-values for the data in the graphs in Figure 5.1. I compared the small (~ 100 components) to the medium (~ 300 components) fault trees and the medium to the large (> 700 components) fault trees. Since there are radii calculated for various fault trees at multiple values of k, I performed a paired t-test on these sets of values. Both comparisons showed p-values below a significance level of .01, which allows me to reject the null hypothesis.

Status of H1: $H1_0$ is rejected by the data. Therefore, we can say that there is evidence supporting H1.

To provide a different view of the same data, Figure 5.2 shows bubble plots of the MCV radii as a function of component count and number of locations. One value of this plot is that it shows data for both rule sets in a single chart; the impact of differences in the characteristics of the fault trees can be seen in the radii of the bubbles in the plot.

It is important to note one caveat for this hypothesis. Because the fault tree component locations were assigned in such a way as to maintain fairly constant density, the 3-dimensional spaces containing the fault trees increase with the size of the fault trees. It is possible that if the greater

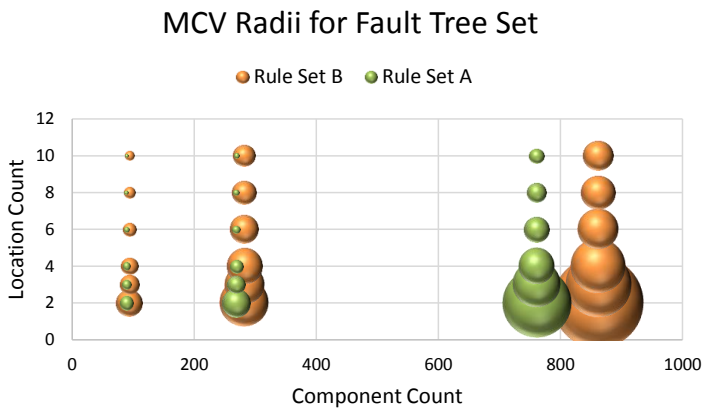

Figure 5.2: Plot showing the average radii for each fault tree rule set as circles to demonstrate scale. k is on the vertical axis and component count is on the horizontal axis.

quantity of components were compressed into the same search space used for the smaller fault trees, that the size of the radii would decrease or remain constant. This nuance was not explored in this research.

Obviously, there is some judgment required on this point. If the larger fault trees were contained within a search space that is smaller than the lowest radius found for the smaller fault trees, then this outcome would be reversed, but that would be entirely as a result of the layout of the fault trees locations. This is an example where a configuration decision made early in the research process has ramifications at this stage of the analysis — that decision was not necessarily wrong or right, but those who perform follow on investigations need to be aware that there is a need for more research on this point. One direction for future research would be finding a general metric for comparing MCV that is applicable to any situation — perhaps measuring MCV as a percentage of the overall search volume would be more useful in some situations.

One of the issues latent in this approach to finding MCV is that the proportion of MCS that can be evaluated drops very quickly to a miniscule fraction of the total known to exist as the number of

components increases. The total number of MCS that can exist for a fault tree increases very fast with the component count — plotting observed values indicate that growth may be exponential. Values for the number of MCS and the percentage of those that are explored to find MCV are shown in the table below and are plotted in Figure 3.3 on page 49.

Table 5.1: *Number of MCS used for finding MCV — showing overall fraction found and average radius of the MCV for two locations for all fault tree instances.*

System	comp count	#MCS calc	#MCS *xfta*	fraction	$\bar{r}_{MCV}$
A-4	89	382725	320517	.837	403.2
A-5	269	1.63E+13	1539	$9.4\text{x}10^{-11}$	848.5
A-6	761	1.44E+28	20000	$1.3\text{x}10^{-24}$	1938.4
B-4	94	9604	9604	1.0	791.2
B-5	282	8.31E+10	20000	$2.4\text{x}10^{-5}$	1400.9
B-6	862	2.04E+24	20000	$9.8\text{x}10^{-21}$	2766.3

Comparison with the EA (see Section 5.3.3) for larger fault trees may provide an indication of the extent to which this effect sabotaged the values found using the AM.

5.2 Cluster Method

The first experimental research method implemented in the course of my research was based on cluster analysis. I began by examining this approach on small and medium fault trees in the test set. Then, I used the fault trees from the full test set, which cover the full range from small to large sized defined for purposes of this research. The CM and AM were executed for $k = \{2, 3\}$ and the results are shown in Table 5.2.

This table shows the shortcomings of the CM for finding MCV. It is normal that multiple executions of stochastic algorithms are necessary; k-means is stochastic due to the random initial partitioning, therefore it can produce different answers each time it is executed. However, my implementation

Table 5.2: *Comparison of average radii found using CM ($\bar{r}_{CM}$) vs. AM ($\bar{r}_{AM}$) on test fault trees for $k = 2$ and $k = 3$. Smaller radii indicate better performance.*

rule	depth	*count*	p_{OR}	$\bar{r}_{CM}$ $(k = 2)$	$\bar{r}_{CM}$ $(k = 3)$	$\bar{r}_{AM}$ $(k = 2)$	$\bar{r}_{AM}$ $(k = 3)$
A	4	89	0.436	819.8	767.1	375.59	291.6
A	5	269	0.539	1436.5	1167.2	848.5	529.3
A	6	761	0.521	1890.9	1948.7	1938.4	1372.5
B	4	94	0.244	–	1236.5	791.2	570.3
B	5	282	0.24	–	1786.9	1400.9	1068.9
B	6	862	0.281	–	2635.8	2766.3	1984.6

of the cluster-based approach did not always find solutions; even when executed for 50 iterations. Therefore, it was necessary to execute the method many more times than the number of results I planned to compare with the AM, which I executed 10 times for each instance of a fault tree. When I used the CM on the fault trees based on rule set B, it was not able to find MCV for $k = 2$; that is, not one iteration produced a value. For $k = 3$, it found results for less than half of the instances, so the average values shown in Table 5.2 are based on incomplete data.

The AM proved to be more robust in comparison, providing MCV values for every iteration of the k-means algorithm. Since *xfta* is deterministic algorithm, it was only excuted once; when it failed, the Monte Carlo elimination technique was executed multiple times to generate MCS results. For fault trees with around 100 components, the AM is able to explore all possible fault trees in a time frame of a few hours. However, for the medium- and large-sized fault trees, this approach explored only a tiny fraction of all possible MCS, as can be seen in the "fraction" column in Table 5.1. Note that *xfta* was unable to find MCS for systems A-6, B-5 and B-6 due to the large size of the smallest MCS for these fault trees and Monte Carlo elimination was used to find MCS for these systems. This allowed the AM to function for all cases since there was always a set of MCS to use as a starting point for the k-means algorithm. The data in Table 5.2 provide sufficient evidence to test

hypothesis H2.

Repeat H2: Partitioning the graph into combinations of k clusters will find smaller cut volumes than those found by traditional methods.

To test this hypothesis, we construct a null hypothesis stating that the method produces equal or larger radii than those found by the control method, the AM. In this case, $\bar{r}_{m,k}$ denotes the average radius found for all instances of a fault tree by method m for a given value of k.

null H2$_0$: $\bar{r}_{CM,k} \geq \bar{r}_{AM,k}$ for $k = \{2, 3\}$

alternate H2$_a$: $\bar{r}_{CM,k} < \bar{r}_{AM,k}$ for fault trees with similar structural metrics.

A paired t-test is appropriate to test these hypotheses. There is a complication, however, since there are no results for fault trees B-04 through B-06 for $k = 2$. Testing only the results for the three "A" fault trees is an extremely small sample, and the calculated p-value is not significant at less than .05. Fortunately, there are results for all 6 fault trees at $k = 3$; the t-test for this provides a p-value below the .05 threshold for H2$_a$. That is, it rejects the alternative hypothesis. However, the power for this test is calculated at only 0.1626 due to the low sample size, so there is danger of falsely accepting the null hypothesis. Nevertheless, this result combined with the difficulty collecting sufficient iterations of the method for all instances of the fault trees and the fact that the method could not identify MCV at all for three of the fault trees at $k = 2$ provided a strong argument against H2, leading me to conclude that there is no support for the hypothesis in these data.

Status of H2: Data do not support H2.

The cluster-first approach was not explored further. While the method appears attractive at first glance, the lack of connection between the clusters of components and their role in defeating the fault tree means that the method is essentially looking blindly for combinations that might defeat

the top level system. In contrast, the AM uses components known to defeat the tree (as demonstrated by the MCS) ensures that the combination will be successful — from this perspective, the AM amounts to using cluster analysis *post facto* to search for the smallest of a set of known solutions.

This result confirms the value of developing a test set containing larger fault trees. Testing the method on these fault trees showed early in the process that there arre fundamental flaws in the approach that were not easily fixed. This allowed me to focus my research efforts on the more productive approaches.

5.3 Evolutionary Algorithms

The EA takes a radically different approach to solving fault trees from traditional FTA, so I approached it in a series of steps. The first step was to find a fitness function that enabled the process to converge gradually toward better answers. After that, I measured the execution speed of the EA to determine whether it is practical for regular analytical use. Finally, I compared the performance of the EA with the AM by directly comparing the MCV values found by both methods.

5.3.1 EA Convergence

The notion of using the radius of effect as the fitness value allows the EA to search directly for the smallest volume and demonstrate smooth convergence properties. Figure 5.3 illustrates an example of convergence for a set of EA executions selected at random; there is one chart per fault tree in the test set. In this case, all were from the execution set where $k = 6$. This value was selected to show the convergence behavior for a more complicated case since the EA tended to converge more slowly for higher values of k. In these plots, the lower curve shows the minimum fitness

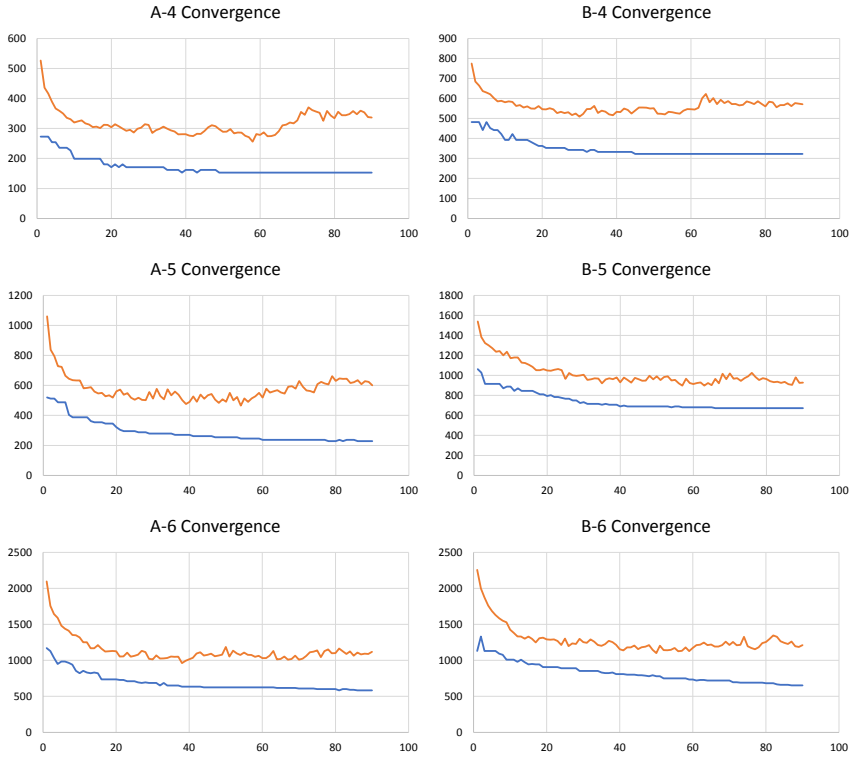

Figure 5.3: Convergence properties for a typical execution of the EA for each of the fault trees in the test set ($k = 6$).

at each generation, while the upper curve shows the average fitness of all genomes. Charting the maximum values was not illustrative of the performance; since my EA implementation added new random genomes each generation, the maximum remained large throughout the execution.

Here, by "converge" I mean that the minimum values reached a steady state. This was assessed by visual inspection of a sample of the output results. The consistent convergence behavior of the EA evident in these plots appears to support hypothesis H3.

Repeat H3: Using a fitness function based on radius of MCV will enable the EA to converge on MCV solutions for an arbitrary number of locations across a range of fault tree sizes.

To test this hypothesis statistically, I compared improvement in the minimum fitness values in the early and late generations of the EA for each fault tree at each value of k. Specifically, I averaged the minimum fitness values for each generation in every iteration of the EA for a given fault tree, then split the data into terciles based on the generations. In the first and last terciles, I compared the average minimum fitness ($\bar{f}$) for the first 5 generations with $\bar{f}$ for the last 5 generations in the same tercile. This notion is illustrated in Figure 5.4 in which the green bands indicate the generations at the beginning of the terciles and the yellow bands those at the end. If the hypothesis is true, then there should be a much smaller difference in the last tercile than in the first.

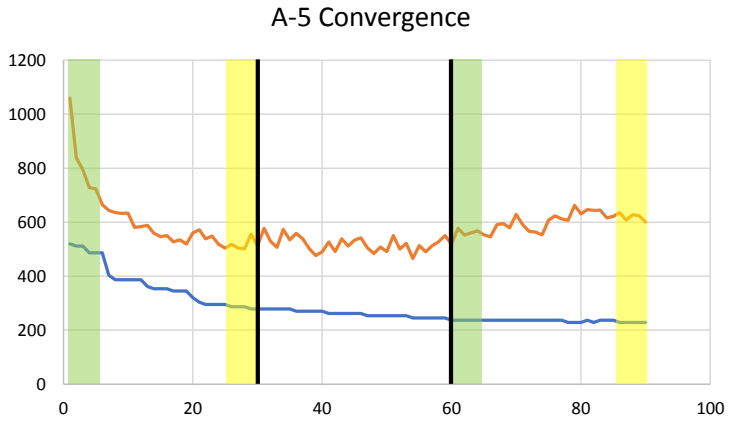

Figure 5.4: Process for testing convergence in minimum fitness values by terciles, shown for a data set where $k = 6$.

I translate this idea into the following alternate hypothesis, in which $\bar{f}_{t,b}$ indicates the average minimum fitness at the beginning of a tercile and $\bar{f}_{t,e}$ indicates the average minimum fitness at the end. I divide by the latter value to provide a measure of relative change in fitness for each tercile.

alternate H3$_a$: $\frac{\bar{f}_{1,b}-\bar{f}_{1,e}}{\bar{f}_{1,e}} \gg \frac{\bar{f}_{3,b}-\bar{f}_{3,e}}{\bar{f}_{3,e}}$ for all fault trees with similar structural metrics

To test for "much greater than," I construct the null hypothesis as follows:

null H3$_0$: $\frac{\bar{f}_{1,b}-\bar{f}_{1,e}}{\bar{f}_{1,e}} < 10\left(\frac{\bar{f}_{3,b}-\bar{f}_{3,e}}{\bar{f}_{3,e}}\right)$ for all fault trees with similar structural metrics

Extracting the minimum fitness values for each generation, then calculating the averages for the beginning and end of the terciles across all EA iterations for each fault tree instance produces the following table of data.

Table 5.3: *Percentage change in average minimum fitness value by generation terciles for all values of k used in the research.*

rule	depth	tercile	$k=2$	$k=3$	$k=4$	$k=6$	$k=8$	$k=10$
A	4	1^{st}	13.8%	25.5%	44.0%	58.1%	67.6%	71.2%
		3^{rd}	0.6%	1.7%	3.1%	3.8%	6.7%	7.2%
	5	1^{st}	17.6%	31.8%	53.9%	66.4%	81.6%	82.1%
		3^{rd}	0.5%	1.5%	1.8%	5.9%	5.8%	7.2%
	6	1^{st}	14.0%	26.3%	36.8%	56.3%	68.4%	74.1%
		3^{rd}	0.2%	1.2%	1.6%	4.2%	7.5%	7.7%
B	4	1^{st}	6.4%	16.1%	22.7%	30.6%	38.4%	43.5%
		3^{rd}	0.2%	0.8%	1.0%	2.4%	3.2%	4.5%
	5	1^{st}	7.5%	14.6%	22.2%	32.8%	40.4%	45.8%
		3^{rd}	0.1%	0.5%	0.8%	2.1%	2.8%	3.9%
	6	1^{st}	7.9%	19.0%	25.1%	36.3%	47.6%	49.7%
		3^{rd}	0.1%	0.8%	1.2%	2.4%	3.7%	4.3%

Since I am testing whether the generations in the third tercile will be much less than those in the first, I performed a paired t-test of the values in the first tercile compared with *10 times* the value in the third. A paired t-test is appropriate since the values are for the EA runs for a specific fault tree at a specific value of k. The resulting p-value is less than .01 for the entire data set, allowing me to reject the null hypothesis and say that these data show evidence in favor of H3.

Status of H3: Data show evidence supporting H3.

This result allowed me to conclude that the EA was indeed converging, so the next step was to examine the timing of the EA.

5.3.2 Execution Time

A key characteristic affecting the utility of any algorithm is its time to execute. This is particularly important as the size of the problem grows, since algorithms whose execution times grow too long become useless when problems grow larger, even though they may appear to be "good" for small problems.

In computational theory, problem domains that have instances in which the best known decision algorithm grows in execution time as a polynomial or faster function of the size are said to execute in *Non-deterministic Polynomial* (NP) time, and are therefore referred to as "NP Complete." These problems are are are often the subject of intense research to find faster solutions. Finding the smallest MCS for a fault tree as the number of components in the MCS increases is such a problem.

Many times, however, a heuristic algorithm is able to find reasonable answers in less than polynomial time. While heuristic solutions are not guaranteed to find solutions on any given execution, they are often the best known solution to a problem [Horoba, 2009] and thus can provide useful answers in practice while research focuses on finding deterministic algorithms that execute in polynomial time or faster. This research explores EAs to find such solutions for finding MCV.

Results from executing the EA across the test set of fault trees shows that the time to execute grows as the size of the fault tree increases. Specifically, it appears to grow in a roughly linear fashion, as can be seen in Figure 5.5. I ran the EA for all 6 fault trees derived from Rule Sets A & B and for values of $k = \{2, 3, 4, 6, 8, 10\}$ and was able to gather timing numbers for 36 distinct cases. The times shown are averaged from 10 executions of the EA for 10 iterations of each fault tree at a specific value of k, thus the points in the figure represent 100 executions of the EA for each fault tree. These results show visually that, for fault trees approaching 1000 components, the EA execution times grow in roughly linear fashion and provide data to test Hypothesis H4.

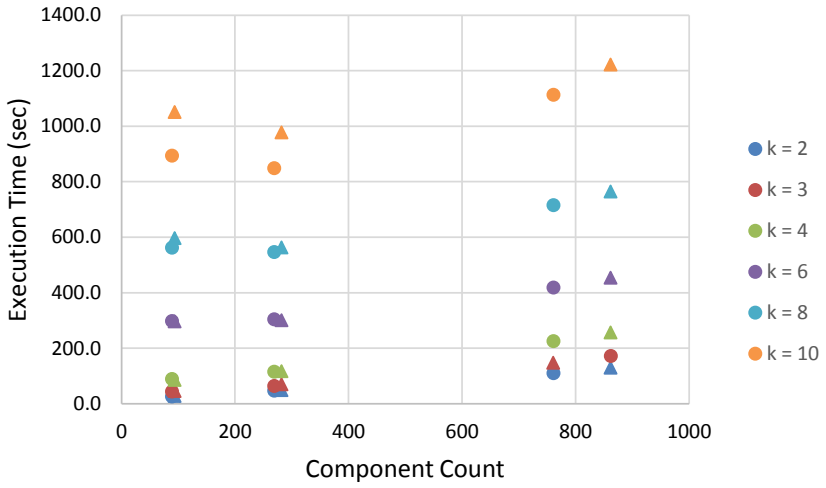

Figure 5.5: Plot of execution time (Y axis) vs. component count for fault tree test set. Circles show fault trees from Rule Set A, while triangles show fault trees from Rule Set B.

Repeat H4: An Evolutionary Algorithm with a radial fitness function will be able to converge on k-event MCV for complex fault trees with location constraints in polynomial time for different values of k as the size of the fault trees increases.

To demonstrate this, I fit a linear model to the average time values for each fault tree at each value of k. The calculated R^2 values and significance levels for the linear models are shown in Table 5.4.

Table 5.4: *R^2 values and level of significance for linear fit models of EA average execution times at each value of k.*

k	R^2	significance
2	0.9966	$< .01$
3	0.9913	$< .01$
4	0.9906	$< .01$
6	0.9367	$< .01$
8	0.8152	$< .01$
10	0.521	$< .1$

These values show that one can be confident that the times grow in a linear fashion for most values of k — the only exception being $k = 10$. However, even in this case, R^2 shows that the timing is close to linear. The oddity is that the EA executes more slowly for small fault trees than for medium. A similar effect is noticeable when $k = 8$, though not as pronounced. Nevertheless, these data support the hypothesis that the EA executes in linear time, which is better than polynomial time.

Status of H4: Timing data support H4.

As a side note, these results also show that the run times do not grow alarmingly as k increases. In part, this is simply a result of the number of generations, since that was set manually for each value of k. However, the number of generations was chosen after observing the convergence characteristics during the course of numerous trial runs. Thus, it is a reflection of the observed behavior of the EA over a variety of conditions.

5.3.3 Comparison of EA with AM

Having established that the EA showed good convergence properties and acceptable run times while generating MCV for the test set fault trees, I compared the results of the EA to the AM, my control method. For each fault tree defined in the trade space, I used 10 instances with different physical locations for every component, creating 60 distinct fault tree instances to evaluate the methods. Additionally, I evaluated each fault tree instance for 6 different values of k (*i.e.*, location count) using both the AM and the EA.

The averaged results are shown in Figure 5.6. These can be compared with Figure 5.1 on page 76 to get a visual indication of the differences in the two methods. A more detailed analysis follows.

This defined the number of clusters used in the k-means algorithm for the AM and the number of

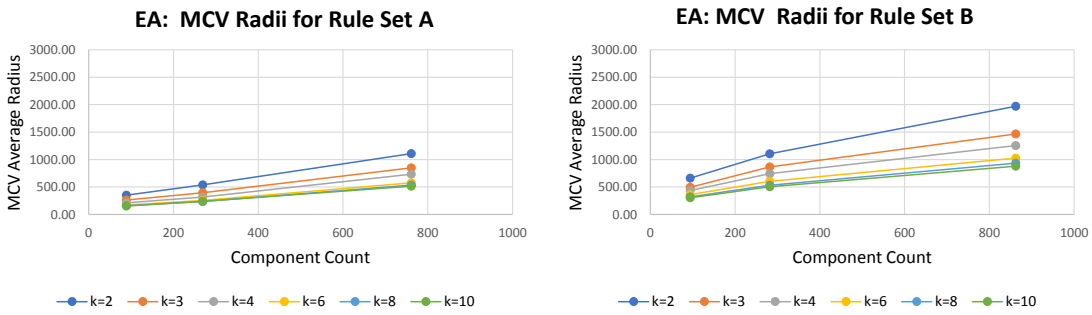

Figure 5.6: Plots showing the radii found for each fault tree (averaged across all instances) by EA at different values of k. Vertical scales are identical for purpose of comparison.

locations in each genome for the EA. Thus, there were a total of 360 cases evaluated using each method. To provide statistical variation, the methods were executed 10 times for each fault tree instance and value of k. In the case of the AM, the k-means algorithm was executed 10 times. *xfta* was only run once for the fault tree since it is a deterministic solution and will produce identical results since it does not use the component locations. However, if *xfta* failed, the Monte Carlo elimination technique was executed multiple times to generate the desired number of MCS; the resulting set of MCS was used for all executions of k-means.

Figure 5.7 shows the radii found by both methods for all instances of a single fault tree. The fault tree selected for this example is A-5 to show performance for a "typical fault tree". Each figure shows the radii found for a different value of k, including all executions for each instance. In some cases, the results fall so close to each other that individual markers cannot be distinguished, thus it will sometimes appear that there are fewer than 10 iterations for a given fault tree instance.

Examining the plots in Figure 5.7 reveals that for $k = 2$, $k = 3$ and $k = 4$, the radii found by the AM are larger than those found by the EA. This indicates that, for this fault tree, the EA performed better. In contrast, for $k = 8$ and $k = 10$, the opposite is true; the AM performed better. For $k = 6$, it is difficult to tell visually whether there is any difference between the two methods.

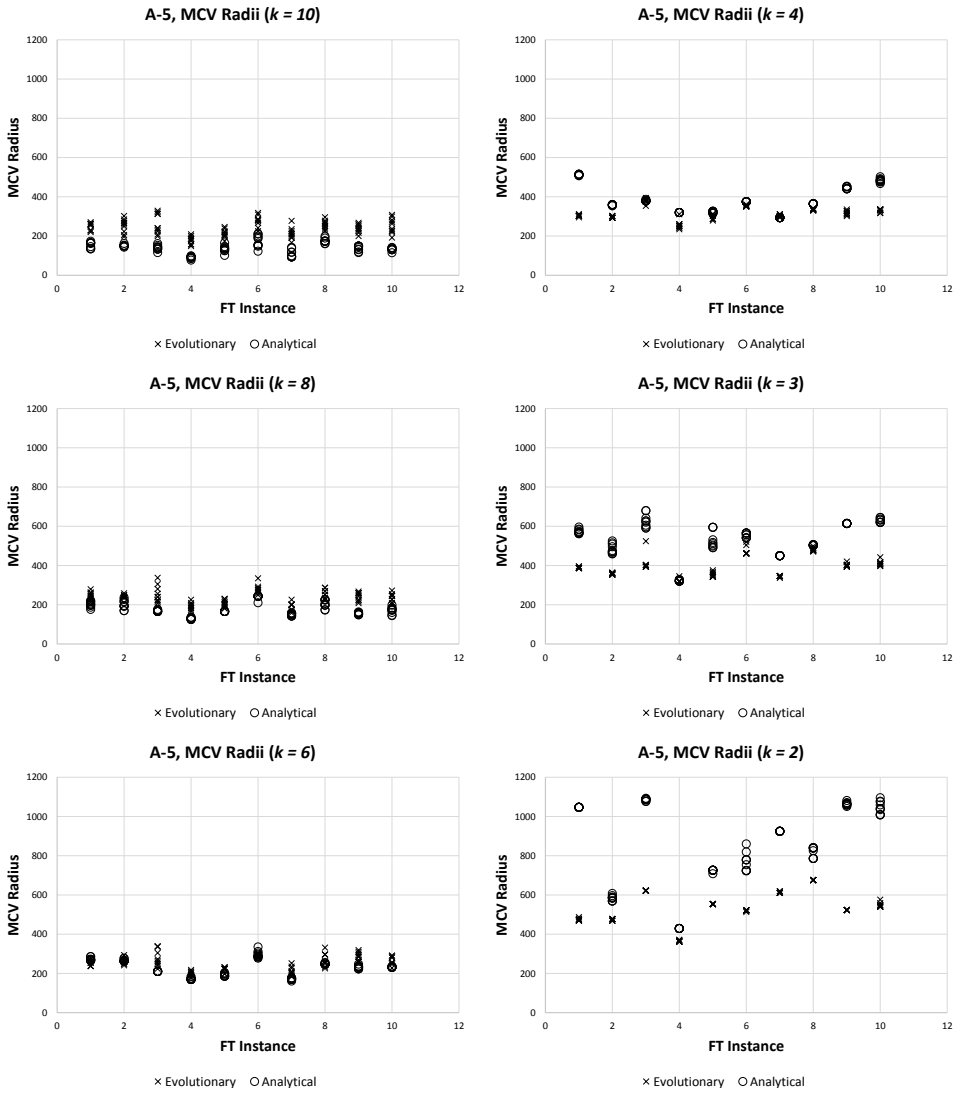

Figure 5.7: Scatter plots showing the radii found for $k = \{2, 3, 4, 6, 8, 10\}$ by the EA and AM for all instances of fault tree A-05. Vertical scale on all plots is identical for purpose of comparison.

Given that another 30 such plots can be created from the data generated, it is not helpful to attempt to draw inferences from these data alone, nor is it instructive to show figures for more than one rule set. The plots shown are examples intended to show the basis for the more aggregated plots shown in Figure 5.8 and Figure 5.9.

Inspired by this view, I defined a metric for comparing the performance of the methods by averaging the radii found for each instance. I label the average radius found for and instance of a fault tree using the AM as $\bar{r}_{AM}$ and that found using the EA as $\bar{r}_{EA}$. I then defined the difference in methods, d_m, for a given instance as $d_m = \bar{r}_{AM} - \bar{r}_{EA}$. This difference in averages was computed for each value of k for each instance of the fault trees in the test set.

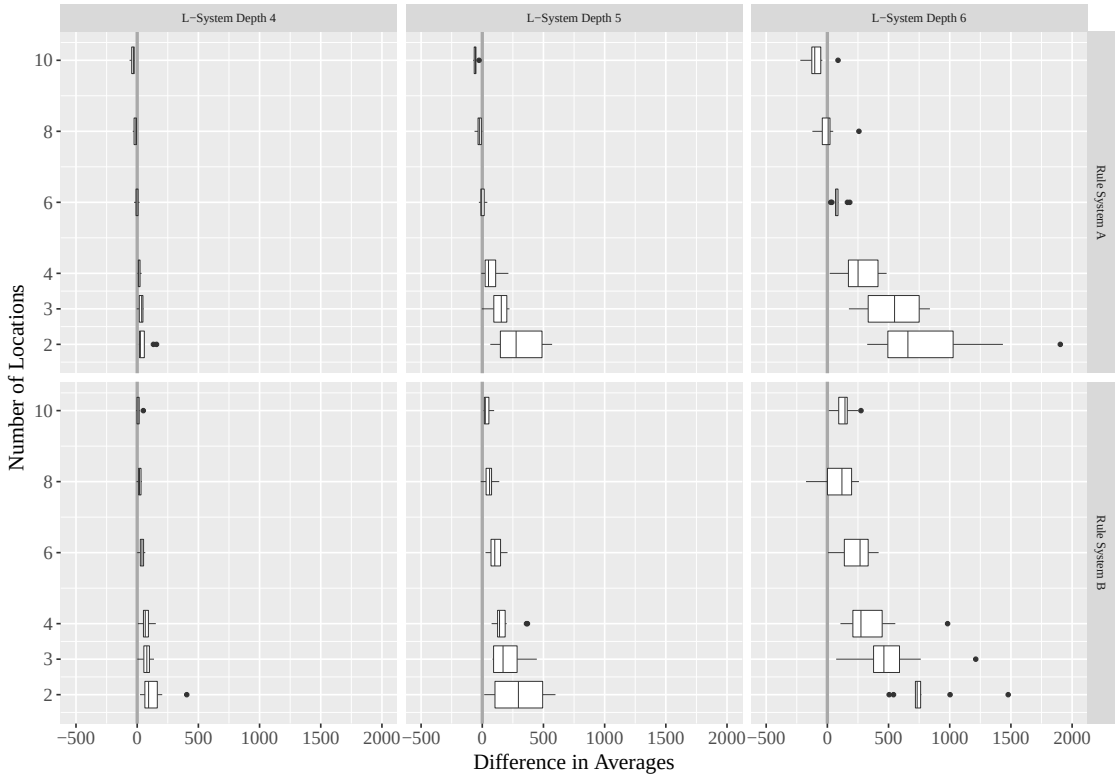

Figure 5.8: Box plots showing differences in averages produced by the EA vs. AM for the test matrix.

With this definition, positive values of d_m indicate that the EA produced smaller radii on average and thus, performed better. Negative values indicate that the AM performed better. d_m was calculated for each instance of every fault tree used in the study. The values of d_m can be plotted using a box and whisker approach to compare the performance of both methods at different values of k. The results of the analysis can then be shown in a single box and whisker graph for each fault tree;

each box represents the range of differences across all instances for one value of k, which is shown on the Y axis. The box plots for all fault trees are shown in Figure 5.8. In each graph, positive values of d_m are right of the vertical grey line while negative are to the left.

Figure 5.8 reveals two consistent trends about the nature of the solutions to the problem. One, as the location count increases, the differences between the methods grow smaller. Two, the EA provides better solutions for lower values of k while the AM sometimes provides better solutions for higher values of k. Closer examination shows that the crossover point where the AM performs better shifts to the right as the fault trees become larger and more complex. That is, it does not work as well for smaller values of k. In particular, it will be noted that for fault trees generated from Rule Set B, the EA performs better for almost all location counts examined, although the AM's relative performance improves as k increases.

Another way to compare the performance of the methods is to create a scatterplot of the radii found by each method. This is analogous to the "predicted" vs. "actual" plots used for showing performance of simulations. However, for this research, the actual value is unknown, so the two methods are compared to each other. Plots were generated for each fault tree instance, plotting the average radius found by the AM on the X axis and the radius found by the EA on the Y axis. The result can be seen in Figure 5.9.

The line for $x = y$ is superimposed as a gray diagonal on each scatterplot to show the boundary where the methods perform equally well. In this configuration, points above the line show where the AM performs better, while points below the line show where the EA performs better. A separate scatterplot is shown for each fault tree in the collection of plots in Figure 5.9.

These plots show clearly that the lower the value of k, the better the EA performs. The previous plots show that the radius is always larger for the smaller location count. Intuitively, this makes sense — as the number of locations decreases, each must contain more components to generate the

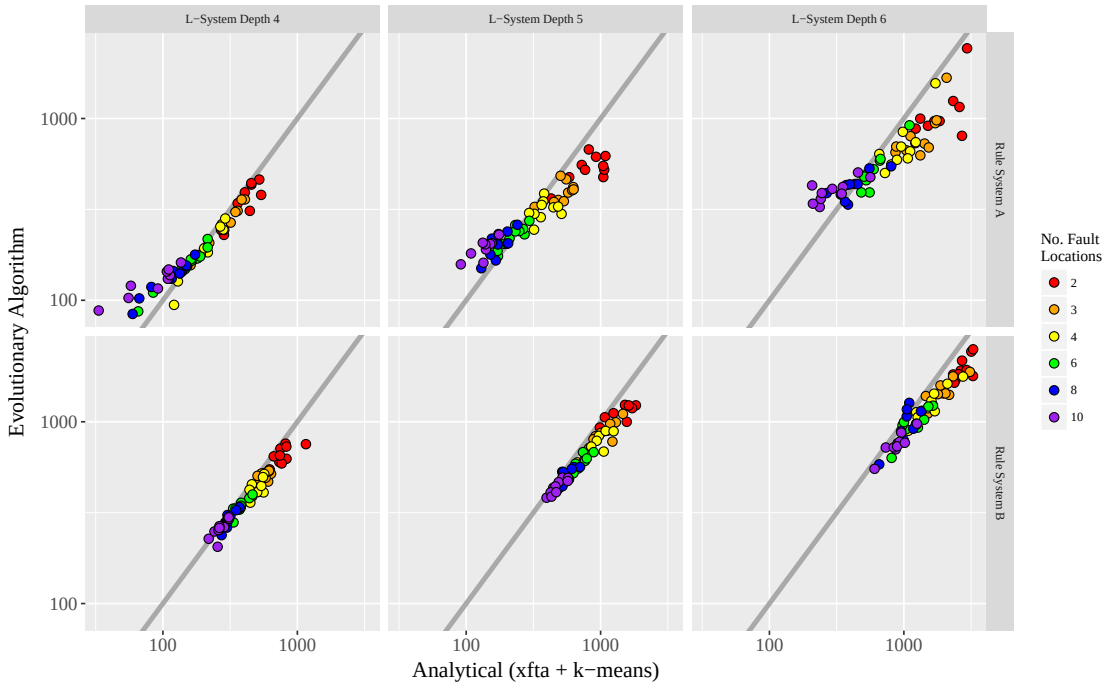

Figure 5.9: Scatterplots comparing results of Evolutionary Algorithm vs. Analytical Method.

cut set, thus they have to be larger to encircle the necessary components. Therefore, the scatterplot shows in a different way that the EA dominates when the location count is small and the AM performs equally or better for larger location counts on some fault trees. Viewing the scatterplots also shows that larger fault trees (those expanded to depth 5 or 6 in this case, so greater than 250 components) tend to have more points below the line than the smaller fault trees. These plots show the data which enable the statistical evaluation of the final hypothesis.

Repeat H5: Evolutionary algorithms or clustering techniques, properly configured, can be developed that find smaller MCV in a complex system of systems when compared to traditional (*i.e.* deterministic) methods of fault tree analysis augmented with heuristic clustering algorithms.

A third way to view the data is to check the statistical significance of the differences seen in the plots above. Since there is a pair of average radii representing the performance of the two methods for every distinct configuration, one can use a paired t-test to compare these values. Since either method could prove to be better than the other, I performed two-tailed tests in this case. I calculated p-values based on the 10 pairs of points for each instance of a fault tree and value of k.

Thus, for each fault tree evaluation at a particular value of k, I test the following null and alternate hypotheses and show the results graphically in Figure 5.10.

null H5$_0$: $\bar{r}_{EA} \approx \bar{r}_{AM}$ for the fault tree

alternate H5$_a$: $\bar{r}_{EA} \neq \bar{r}_{AM}$ for the fault tree

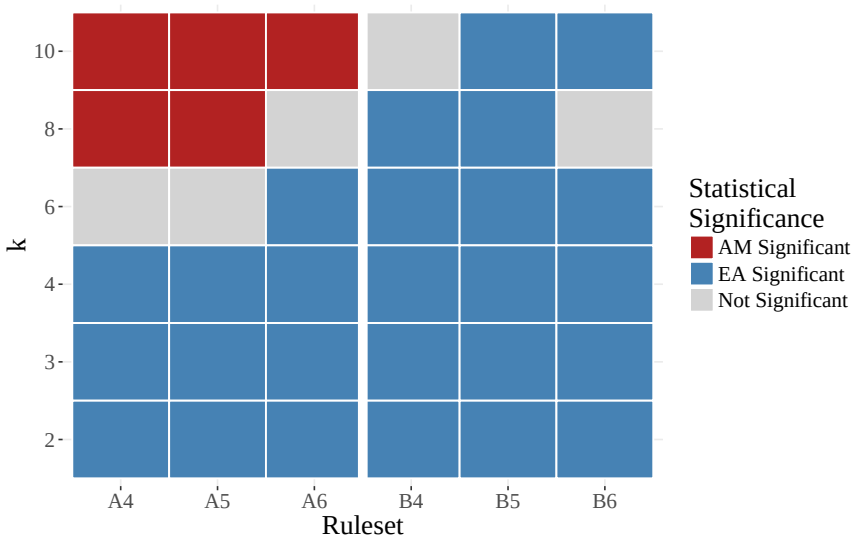

Figure 5.10: Results of pairwise t-tests comparing performance of EA vs. AM for each fault tree and value of k.

Of the 36 p-values calculated, all but 5 show that the differences seen are significant to the .05 level, indicating that we can reject with 95% confidence the null hypothesis that the methods produce the same results. Of the values that show significant differences, the AM performs better for 5

cases. These can be seen in Figure 5.10; here, the red cells represent cases in which the AM was statistically superior, and the blue cells represent cases in which the EA was statistically superior. The gray cells show cases where there is no statistically significant difference.

The statistical power was calculated for a sample size of 10 and .05 level of significance. The data show that the differences in the radii of the MCV, when they appear, are large relative to the standard deviations. When examining differences in the mean that are the same size as the standard deviation, the power is computed to be $.803$. Therefore, since the null hypothesis was rejected in 5 cases, there is a likelihood that one of the methods might actually be better for one of those cases. However, even if that is the case, the overall trend remains clear.

All cases where the AM performs better are for fault trees derived from Rule Set A (which has a roughly even proportion of OR and AND operators) and for values of k greater than 6. In the remaining 26 cases, the EA performs better at finding smaller radii for the MCV. Given those results, it is reasonable to conclude that the EA performs better at finding smaller MCV for low values of k for the medium and large fault trees evaluated in this research. Therefore, we can say that the null hypothesis is rejected more often for the fault trees explored in this research and that the results show evidence to support hypothesis H5 that the EA generally performs better than the control method for finding MCV for medium and large fault trees.

Status of H5: Data provide support for H5.

The caveat is that there are circumstances in which an analyst should consider using the AM or both methods together. The AM provides valuable insight to the problem and should not be overlooked by analysts working with real world problems.

5.4 Status of Hypotheses

In this research, I explored a series of Hypotheses related to the performance of various heuristic methods for finding MCV for fault trees augmented with component location data. The results of these hypotheses show the merits of each method and indicate which are more likely to help analysts tasked with identifying vulnerabilities in large, complex systems. These results are summarized in Table 5.5 for convenient review.

Table 5.5: *Summary of Results for hypotheses defined in Chapter 3.*

Hypothesis	Synopsis	Defined	Status
H0:	L-systems can maintain structural metrics	3.3	results demonstrate $H0$
H1:	AM shows smaller radii for smaller fault trees	4.2.4	data support $H1_a$
H2:	CM finds smaller radii than AM	4.3	data do not support $H2_a$
H3:	EA convergence	4.4	data support $H3_a$
H4:	EA execution times are linear	4.4	data support $H4$
H5:	EA or CM will find smaller MCV than AM	4.5	data support $H5$

CHAPTER 6: CONCLUSION

The work presented in this dissertation has many potential applications that could be of tremendous benefit in urban and/or industrial settings. Ideally, I will have the opportunity to apply it in real world industrial settings such as oil refineries or nuclear power plants. Facilities such as these often have hidden vulnerabilities of the type addressed in this research and could provide an opportunity to demonstrate the analytical power of these methods. Anecdotally, I have been told that some facilities in the power generation industry go through a manual process to identify locations where a fire could cause the shutdown of a facility; this is exactly the type of analysis that my methods could help to automate.

6.1 Generating Fault Trees

The initial focus on creating a set of benchmark fault trees for testing proved to be of great benefit at various stages throughout the process. It enabled me to generate fault trees for my own use, and to pioneer a compact method for practitioners to share fault trees in the future. L-systems have great potential for improving the level of testing across the reliability industry by allowing researchers to create and share benchmark problems for evaluating methods as was done in this research.

L-system research has also produced many nuanced variants designed to enhance the developer's control over the outcome of the rule expansion. These include such techniques as context-sensitive and parametric L-systems, which provide the ability to modify rules in response to the setting of each node or to respond to specific variables created by the developer, respectively. For creating my benchmark fault trees, these abilities did not prove to be necessary since my focus was on

creating a set of fault trees for testing. However, these techniques could provide useful capabilities for tailoring fault trees, which is a good topic for future research.

One of the innovations developed in the course of arriving at the benchmark set was the creation of metrics for comparing fault trees — apart from the component count, I have not seen any such metrics used or discussed in literature related to fault trees. Nevertheless, a collection of standard metrics could greatly improve the dialog on FTA methods by enabling analysts to communicate their problems to each other clearly and consistently. Such metrics are valuable in themselves as they provide a means to compare methods and to measure performance in an unbiased way.

A potential follow on research idea is to develop a "taxonomy of fault trees" that seeks to define the major characteristics that differentiate fault trees and that affect the difficulty of solving them. An effort of this nature would require access to as many real world fault trees as possible to build up a data set that shows the normal range of parameters encountered in industrial problems and to identify the metrics that matter most. This would be valuable for method developers to understand the conditions in which FTA is used.

6.2 Control Method

The AM proved to be more than just a control for the research methods — it is a viable method of finding MCV in its own right and preferable in some cases. The primary weakness of this approach is that the number of MCS grows very fast with the size of the fault tree. The reader is reminded of Table 5.1 on page 79, particularly the values for fault trees A-6 and B-6. Thus, when the AM finds a solution for medium or large fault trees, it leaves open the question of whether an MCS exists that has more components but occupies a smaller volume than the MCV found in the calculation. Without fully evaluating all MCS, this remains unknown; however, since finding all MCS is #P

Complete, this problem is unlikely to be solved in the near future.

In addition, when the size of the MCS grows too large, this method can fail due to memory limitations and provide no answers at all. Although Monte Carlo elimination is able to provide MCS when deterministic methods fail, it cannot solve the performance problem since it is a slow-running heuristic and is not guaranteed to find all possible MCS.

However, the fact that the AM starts from MCS that are known to defeat the system is a tremendous benefit and allows it to outperform my first research method and to provide a worthwhile comparison for the second.

6.3 Cluster Analysis

Although searching for clusters in the components seems initially to be closely aligned with the problem statement, it suffers from the fact that it ignores information about the influence of the components on the system, namely their membership in MCS. The approach of finding clusters and then evaluating the fault tree can only succeed by happenstance — the binary nature of the fault tree hinders this method the same way that it did for my original genetic algorithm approach. This leaves open the possibility that a more refined cluster technique could be developed to find MCV. Such a technique would need to use a gradient of some sort to group components by more than just their physical location. However, there has been significant research in multi-dimensional clustering which could provide means to incorporate the logical structure of the fault tree in the clustering process.

The sequence of steps required to generate the MCV radii beginning with cluster analysis highlights one of the unforeseen weaknesses of this approach as implemented in my research. That is, one must generate clusters of a specific size, then stop to determine whether any combination of

clusters results in a cut set. Once a successful size is found, attempting to find a smaller radius entails refining clusters, then trying all combinations again to determine whether any of them kill the system. Consequently, this approach can be a very labor intensive means of seeking MCV. As the clusters get smaller there are more of them, so each round of radius computations requires more combinations, since the number of combinations increases exponentially (*e.g.*, double the number of clusters creates four times as many combinations of 2 and eight times as many combinations of 3). All of this can dramatically increase run times as k increases and r decreases. I did not see this effect because the cluster approach did not readily identify MCV, so there was no need to seek more refined clusters.

This behavior contrasts poorly with the AM, which starts with a set of MCS known to defeat the top level system, then creates clusters from those to find the radius (or radii) required to encompass each MCS in a MCV. Thus, the smallest radius is found in a single pass, rather than requiring multiple iterations of an intermittently successful algorithm.

Despite the poor showing of the cluster-first analysis approach, cluster analysis in general was an integral part of the more successful AM, which leveraged knowledge about the fault tree in the form of MCS to bound the search. Consequently, clustering proved to be an important technique for finding MCV for small to medium fault trees, but not in the manner originally envisioned.

6.4 Evolutionary Algorithms

This research once again demonstrates the power and flexibility of EAs at finding solutions to complex problems in relatively short time scales. The focus on using spheres of effect for the fitness proved to be crucial to the success of this approach. Other efforts (mine included) that focus only on the logical structure of the fault tree encountered a combinatorial explosion of possibilities

that proved impossible to solve in reasonable time. Deterministic Boolean solvers are much faster at finding MCS, as long as the size of the problem is not too great.

Once the focus of the problem changes to seeking volumes of effect, it makes sense to develop an EA that uses the desired outcome as the fitness value. This avoids the problem confronted by the CM (*i.e.*, blindly searching for an effect) because it implicitly uses information derived in each generation to improve the performance of the following generation. Thus although it does not explore the structure of the fault tree directly, it uses the knowledge from evaluating the fault tree in each generation to continually refine the cut volumes.

One direction for future research will be to allow the method to use different effect sizes at each location so that it finds a true minimum value for the MCV. Also, blast and fire are not always spherical in their effects, especially when confined in a building or blocked by a structure. Therefore, an important extension to this research will be to vary the shape of the volume of effect as well as the size. This will enhance the fidelity of the final result because it can be tuned to emulate data from experimentation or modeling.

6.5 Expanding the Problem Definition

In addition to refining the methods, there are some areas in which changing the nature of the problem could provide more insight into the real world situations addressed by this analysis. Most of these add complexity to the analysis, so they were deemed to be distractions from the initial focus on the existence of viable methods to identify MCV.

One way to add realism to the problem definition is to add a probability of effect that varies with the radius of the volume. The current formulation is a sphere inside which the probability of effect is 1.0 and outside which the probability is 0.0. However, real damage from disasters is generally not

discontinuous. Therefore, adding a probability of damage that is a function of the distance from the center would increase the realism of the overall result. Of course, both the AM and the EA would need to be modified to incorporate this information. Probability risk analysis tools would need to be combined with the current MCV methods to define probabilities of system failure instead of a simple binary failure condition as used in this research.

Another area of interest would be to combine the spatial approach used here with the temporal analysis performed using dynamic fault trees (DFT). Combining these two techniques would add significantly to the complexity of the methods, but it would also allow research into progressive failures, in which certain events can cause delayed failures or in which specific sequences of events might cause disasters to be more or less severe than they would otherwise be. It could also allow the DFT to inject response time and access routes of emergency personnel such as firefighters to determine the degree to which they mitigate vulnerabilities found in MCV analysis. Adding temporal effects would enable modeling of almost any kind of disaster and response, providing a high level picture of the vulnerability of a facility and its preparedness to meet with a wide variety of disaster situations.

Also, expanding the performance metrics could be of benefit. During the course of the research, as results were generated using both methods, it became clear that the most direct metric for comparing the methods was the size of the smallest radius found. The originally proposed metric, finding the largest number of "hot spots" (*e.g.*, the number of combinations with a radius that fell below some threshold) was less useful for the initial research on the topic. However, in the long term, it could also prove to be a valuable metric and should be explored in future research. Conceptually, the smallest radius represents the worst pathological case; that is, the one in which it is easiest for a disaster to cause the top level system to fail. On the other hand, a different metric constructed from the number of radii below a specific threshold would represent all combinations of concern. In theory, one could conceive of a general metric that combines both the smallest radius and the

number of combinations that fall within some range close to it.

6.6 Future Work

I have discussed several potential directions in which this research could be expanded in the future. Because these are scattered throughout different sections of the dissertation as they naturally arise, I collate and summarize these ideas here for ease of reference.

- Develop a "taxonomy" of fault trees based on metrics defined in this dissertation to characterize them for communication in research.
- Expand L-system grammar techniques to tailor fault tree characteristics to match more closely those encountered in industrial applications.
- Use EAs to create a fully automated approach to creating L-system rule sets that produce fault trees with desired characteristics.
- Apply the methods to one or more real world, complex systems to validate the degree to which the methods developed in this research truly provide the desired insight.
- Analyze fault trees with a wider range of characteristics to discover the conditions under which each solution method is better at identifying MCV.
- Explore multi-dimensional clustering methods to help the AM make better use of the information in the MCS.
- Explore different EA structures to create EAs that find better MCV in fewer generations.
- Enhance the EA to use different effect sizes at each location so that it finds a true minimum value for the MCV.

- Vary the shape of the volume of effect to emulate more accurately the effects of fire or blast.
- Add a probability of effect that varies with the radius of the volume enclosed by the EA or the AM.
- Combine MCV approaches with dynamic fault trees.
- Develop a metric for comparison based on the number of cut volumes that fall below a given threshold.

This list demonstrates that there are many ways in which this research could be extended to improve the fidelity of the analysis. All of these topics will elaborate the ideas initiated in this dissertation — collectively, they could grow into a process that provides deep insight into the vulnerabilities of systems at risk from disaster. Clearly, the most important next step is to use these techniques on a few complex systems from industrial settings to assess the level of understanding they afford in their current state.

6.7 Final Thoughts

Fault trees were developed to find weaknesses in designs without spending significant resources building systems or facilities that could be seriously flawed. In this vein, using fault trees to find weaknesses caused by physical proximity of components in three dimensional space can be considered a natural expansion of their utility. The methods demonstrated in this research are suitable for identifying such weaknesses and for evaluating potential solutions for mitigating them.

Underlying this discussion is the assumption that there exists a true smallest MCV for each system. The methods explored in this research seek to estimate that value through approximations of failure volumes. At this time, closed-form solutions do not exist to find such a volume, so there are no

means to determine how well the research methods conform to the "true value". Thus, they are compared to each other as a first order figure of merit — it remains to be seen how far the results of this research differ from the "true values."

The methods used here are all developed from stochastic techniques — in fact, only one of them contains deterministic elements. Thus, they are subject to the usual caveats that apply when using stochastic methods:

- Execute the method multiple times to generate statistical variation — examine the results to understand the variance,
- Adjust the dynamic parameters of the execution to provide the best possible answer for the problem at hand.
- Understand that the methods cannot provide a provable "best" answer, but they can be good enough to reveal design problems that must be addressed.

The EA performs better (*i.e.*, finds smaller MCV radii) for more complex fault trees and lower values of k. On the other hand, the AM showed good results for the less complex fault trees and higher values of k.

The real limitation for the AM is simply the small fraction of the total number of possible MCS that can be explored in a useful period of time. Given that the number of possible MCS was as high as 10^{28} among the benchmark fault trees examined in this research, it will not be possible using current hardware to evaluate more than a tiny fraction of the overall space. Therefore, analysts tasked with evaluating larger fault trees will always face the likelihood that the "true" best solution is likely to be found in the vast number of unexplored MCS.

Another important research challenge is to find a closed-form deterministic method to find optimal

MCV in the solution space. However, given that finding all MCS for a fault tree is #P-complete, an algorithm to find the true MCV is likely to execute in polynomial or exponential time. Therefore, heuristics will still be needed to find MCV for medium to large sized fault trees for the foreseeable future.

Until there is a viable deterministic solution, analysts and engineers faced with the challenge of eliminating vulnerabilities in large systems could use the methods described herein as a means to identify MCV and evaluate mitigation schemes. My recommendation for such analysis is to use both approaches in parallel, since they appear to have complementary strengths. This "toolbox" approach would enable designers to find the most useful solution for their particular situation. There may also be other means to approach the problem which could be incorporated into designers' "toolboxes." The most time-consuming aspect of FTA is still the process of creating the fault trees representing the system being studied. Creating a good fault tree that represents a large, complex system accurately and in appropriate detail takes a significant amount of effort from a team of experienced engineers. Once the fault tree has been created, analyzing it with tools such as those presented in this article is relatively fast. Therefore, it makes sense to use as many analytical methods as possible to gain the necessary insight into the possible failure modes of the system.

As mentioned earlier, it would be useful to explore the characteristics of fault trees besides just size and operator proportion that affect the performance of the methods. There are many possible fault tree metrics that could make them easier or harder to solve — understanding how these affect the process of solving for MCV can help designers find ways to reduce the vulnerability of real world systems and thus, save lives in the face of future disasters.

APPENDIX A: EXAMPLE L-SYSTEM RULES

Below are four L-system rule sets designed to emulate fault trees found in the literature as described in Section 3.3. Metrics calculated for comparison are shown in that Section.

Example 1: L-System to match Song et al. [2009]

start: A
A: (OR) BE
B: (OR) FCF
C: (AND) FE
E: (OR) AF
F: (T)

Example 2: L-System to match Lacey [2011]

start: A
A: (OR) BE
B: (AND) ACE
C: (OR) FE
E: (OR) CZ
F: (AND) BF
Z: (T)

Example 3: L-System to match [Sui and Pan, 2011]

start: A
A: (OR) *BCD
B: (OR) ACE
C: (AND) *DRE
D: (OR) CDZ
E: (OR) ZZ
F: (AND) RE
R: (RS)
Z: (T)

Example 4: L-System to match [Shahriar et al., 2012]

start: A
A: (OR) BCD
B: (OR) *ACZZ
C: (AND) *DER
D: (OR) FRZ
E: (OR) GZZ
F: (AND) RE
G: (OR) RD
H: (AND) ACE
R: (RS)
Z: (T)

LIST OF REFERENCES

S. S. Agarwal and M. L. Kansal. Fuzzy fault tree analysis of a power transformer. In *2012 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering*, pages 1000–1004, June 2012. doi: 10.1109/ICQR2MSE.2012.6246393.

Farhan Ahammed and Pablo Moscato. Evolving l-systems as an intelligent design approach to find classes of difficult-to-solve traveling salesman problem instances. In *Proceedings of the 2011 International Conference on Applications of Evolutionary Computation - Volume Part I*, pages 1–11. Springer, 2011.

A. Baklouti, N. Nguyen, J. Y. Choley, F. Mhenni, and A. Mlika. Free and open source fault tree analysis tools survey. In *2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–8, April 2017. doi: 10.1109/SYSCON.2017.7934794.

M. O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Transactions on Reliability*, 35(3):230–239, Aug 1986. ISSN 0018-9529. doi: 10.1109/TR.1986.4335422.

Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. A compositional semantics for dynamic fault trees in terms of interactive markov chains. In *International Symposium on Automated Technology for Verification and Analysis*, pages 441–456. Springer Berlin Heidelberg, 2007.

Gary Chartrand. *Introductory Graph Theory*. Dover, New York, NY, 1977.

Kenneth Alan De Jong. *Evolutionary Computation: A Unified Approach*. MIT press, 2006. ISBN 0262041944. URL `http://mitpress.mit.edu/0262041944`.

Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1 – 38, 1987. ISSN 0004-3702. doi: https://doi.org/

10.1016/0004-3702(87)90002-6. URL http://www.sciencedirect.com/science/article/pii/0004370287900026.

Paul H. Deitz, Harry L. Reed Jr., J. Terrence Klopcic, and James N. Walbert. *Fundamentals of Ground Combat System Ballistic Vulnerability/Lethality*. Progress in Astronautics and Aeronautics. AIAA, Reston, VA, 2009.

Morris R. Driels. *Weaponeering: Conventional Weapon System Effectiveness*. AIAA education series. American Institute of Aeronautics & Astronautics, 2013. ISBN 9781600869259. URL https://books.google.com/books?id=nBq6MAEACAAJ.

J. B. Dugan, S. J. Bavuso, and M. A. Boyd. Fault trees and sequence dependencies. In *Annual Proceedings on Reliability and Maintainability Symposium*, pages 286–293, Jan 1990. doi: 10.1109/ARMS.1990.67971.

J. B. Dugan, S. J. Bavuso, and M. A. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–377, Sep 1992. ISSN 0018-9529. doi: 10.1109/24.159800.

Clifton Ericson. Fault tree analysis – a history. In *Proceings of the 17th International System Safety Conference*, 1999.

M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP Completeness*. Freeman & Company, 1979.

J. Gauthier, X. Leduc, and A. Rauzy. Assessment of large automatically generated fault trees by means of binary decision diagrams. *Journal of Risk and Reliability, Professional Engineering Publishing*, 221, 2007.

Youngjung Geum, Hyeonju Seol, Sungjoo Lee, and Yongtae Park. Application of fault tree analysis to the service process: service tree analysis approach. *Journal of Service Management*, 20(4):

433–454, 2009. doi: 10.1108/09564230910978520. URL https://doi.org/10.1108/09564230910978520.

Vibhav Gogate and Rina Dechter. Approximate inference algorithms for hybrid bayesian networks with discrete constraints. 2005.

Danan Gu, Patrick Gerland, Franois Pelletier, and Barney Cohen. Risks of exposure and vulnerability to natural disasters at the city level: A global overview. *United Nations Population Division Technical Paper*, (2015/2), 2015. URL https://esa.un.org/unpd/wup/publications/Files/WUP2014-TechnicalPaper-NaturalDisaster.pdf.

Jiawei Han, M Kamber, and Anthony Tung. Spatial clustering methods in data mining: a survey. 01 2001.

Jeff Hanes and John F. Fay. An exploration of fault tree analysis methods for military simulations. In *Proceedings of the 48th Annual Simulation Symposium*, ANSS '15, pages 41–48, San Diego, CA, USA, 2015. Society for Computer Simulation International. ISBN 978-1-5108-0099-1.

Jeff Hanes and R. Paul Wiegand. Using l-systems to generate fault trees for benchmarking and testing. In *Proceedings from the* 29^{th} *Florida Artificial Intelligence Research Society Conference*, 2016.

Brent Hecht and Emily Moxley. *Terabytes of Tobler: Evaluating the First Law in a Massive, Domain-Neutral Representation of World Knowledge*, pages 88–105. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-03832-7. doi: 10.1007/978-3-642-03832-7_6. URL https://doi.org/10.1007/978-3-642-03832-7_6.

E.J. Henley and H. Kumamoto. *Reliability engineering and risk assessment*. Prentice-Hall, 1981.

Christian Horoba. Analysis of a simple evolutionary algorithm for the multiobjective shortest path problem. In *Proceedings of the Tenth ACM SIGEVO Workshop on Foundations of Genetic Algo-*

rithms, FOGA '09, pages 113–120, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-414-0. doi: 10.1145/1527125.1527140. URL http://doi.acm.org/10.1145/1527125.1527140.

Naveed Imran and Ronald F. DeMara. A self-configuring tmr scheme utilizing discrepancy resolution. pages 398–403, 2011.

Chris Isidore. Gm's total recall cost: $4.1 billion. *CNN Money*, February 2015. URL http://money.cnn.com/2015/02/04/news/companies/gm-earnings-recall-costs/index.html.

Peter Lacey. An application of fault tree analysis to the identification and management of risks in government funded human service delivery. In *Proceedings of the 2nd International Conference on Public Policy and Social Sciences held in Kuching*, 2011.

S. Manousakis. Musical l-systems. Master's thesis, Institute of Sonology, 2006.

Glenn A. Martin, Charles E. Hughes, Sae Schatz, and Denise Nicholson. The use of functional l-systems for scenario generation in serious games. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pages 6:1–6:5. ACM, 2010.

Christoph Meinel and Anna Slobodová. *On the complexity of constructing optimal ordered binary decision diagrams*, pages 515–524. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994. ISBN 978-3-540-48663-3. doi: 10.1007/3-540-58338-6_98. URL https://doi.org/10.1007/3-540-58338-6_98.

Guillaume Merle, Jean-Marc Roussel, Jean-Jacques Lesage, and Andrea Bobbio. Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events. *IEEE Transactions on Reliability*, 59(1):250–261, 2010.

Guillaume Merle, Jean-Marc Roussel, and Jean-Jacques Lesage. Dynamic fault tree analysis based on the structure function. In *2010 Annual Reliability and Maintainability Symposium*, pages 462–467, Jan 2011.

Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998. ISBN 0262631857.

Hongxia Pan, Jinying Huang, and Guangmin Liu. Fault diagnosis of circuit board based on fault tree. In *2008 10th International Conference on Control, Automation, Robotics and Vision*, pages 1666–1671, Dec 2008. doi: 10.1109/ICARCV.2008.4795777.

P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, NY, 1996.

Steven D. Pyle, Vignesh Thangavel, Stephen M. Williams, and Ronald F. DeMara. Self-scaling evolution of analog computation circuits with digital accuracy refinement. pages 1–8, 2015.

Antoine Rauzy. New algorithms for fault tree analysis. *Reliability Engineering and System Safety*, 40:203–211, 1993.

Antoine Rauzy. Mathematical foundation of minimal cutsets. *IEEE Transactions on Reliability*, 50(4), 2001.

Antoine Rauzy. Xfta: An open-psa fault tree engine. 2012a.

Antoine Rauzy. Anatomy of an efficient fault tree assessment engine. In R. Virolainen, editor, *Proceedings of International Joint Conference PSAM'11/ESREL'12*, June 2012b. ISBN 978-162276436-5.

Enno Ruijters and Marille Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15-16(Supplement C):29 – 62,

2015. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2015.03.001. URL http://www.sciencedirect.com/science/article/pii/S1574013715000027.

Heiko Ruppert and Bernd Bertsche. Cad-integrated reliability evaluation and calculation for automotive systems. In *Annual Reliability and Maintainability Symposium. 2001 Proceedings. International Symposium on Product Quality and Integrity (Cat. No.01CH37179)*, pages 264–271, 2001.

M. S. Saglimbene. Reliability analysis techniques: How they relate to aircraft certification. In *2009 Annual Reliability and Maintainability Symposium*, pages 218–222, Jan 2009.

Anjuman Shahriar, Rehan Sadiq, and Solomon Tesfamariam. Risk analysis for oil & gas pipelines: A sustainability assessment approach using fuzzy based bow-tie analysis. *Journal of Loss Prevention in the Process Industries*, 25(3):505 – 523, 2012. ISSN 0950-4230. doi: http://dx.doi.org/10.1016/j.jlp.2011.12.007. URL http://www.sciencedirect.com/science/article/pii/S0950423011002154.

Wenhua Song, Huifang Shi, and Qinggong Li. Application of fault tree knowledge in reasoning of safety risk assessment expert system in petrochemical industry. In *Knowledge Engineering and Software Engineering, 2009. KESE '09. Pacific-Asia Conference on*, pages 167–170, Dec 2009. doi: 10.1109/KESE.2009.51.

M. Stamatelatos, W.E. Vesely, J. Dugan, J. Fragola, J. Minarick, and J. Railsback. *Fault Tree Handbook with Aerospace Applications*. NASA Office of Safety and Mission Assurance, Washington, DC, 2002.

Yongqin Sui and Xiaodong Pan. Reliability assessment of urban anti-disasters system based on fuzzy fault tree analysis. In *Emergency Management and Management Sciences (ICEMMS), 2011 2nd IEEE International Conference on*, pages 159–162, Aug 2011. doi: 10.1109/ICEMMS.2011.6015644.

Lukas Sundermann, Oliver Schelske, and Peter Hausmann. *Mind the Risk A global ranking of cities under threat from natural disasters*. SwissRe, Zurich, 2014.

J. A. E. ten Veldhuis, F. H. L. R. Clemens, and P. H. A. J. M. van Gelder. Fault tree analysis for urban flooding. *Water Science and Technology*, 59(8):1621–1629, 2009. ISSN 0273-1223. doi: 10.2166/wst.2009.171. URL `http://wst.iwaponline.com/content/59/8/1621`.

P. Tian, J. Wang, W. Zhang, and J. Liu. A fault tree analysis based software system reliability allocation using genetic algorithm optimization. In *2009 WRI World Congress on Software Engineering*, volume 2, pages 194–198, May 2009. doi: 10.1109/WCSE.2009.227.

J.K. Vaurio. Common cause failure probabilities in standby safety system fault tree analysis with testing - scheme and timing dependencies. 79:43–57, 01 2003.

W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl. *Fault tree handbook, NUREG-0492*. United States Nuclear Regulatory Commission, Washington, DC, 1981.

B. Watson, P. Mller, O. Veryovka, A. Fuller, P. Wonka, and C. Sexton. Procedural urban modeling in practice. *IEEE Computer Graphics and Applications*, 28(3):18–26, May 2008. ISSN 0272-1716. doi: 10.1109/MCG.2008.58.

J. Xiang and K. Yanoo. Formal static fault tree analysis. In *2010 International Conference on Computer Engineering & Systems*. IEEE, 2010.

Ling Zhu, Kalyanmoy Deb, and Sandeep Kulkarni. *Multi-scenario optimization using multi-criterion methods: A case study on Byzantine agreement problem*, pages 2601–2608. Institute of Electrical and Electronics Engineers Inc., 9 2014. ISBN 9781479914883. doi: 10.1109/CEC.2014.6900637.