A Mathematical Method to Optimize Fault Tree Displays

by Thomas C. Bingham II
Washington State, U.S.

Fault Tree Analysis (FTA) has undergone significant development since its inception in the early 1960s. It has been instrumental in modeling safety and reliability problems in which failure simply is not an option. Aerospace and nuclear industries, several universities, government agencies and software developers have played a large role in the development of FTA. As a result, practical and widely applicable mathematical tools have been established using reasonably common terminology and computer applications.

This paper focuses on the fault tree display itself, and on improvements that add significant benefits to it. Although there are multiple standards currently in use, fault tree displays tend to follow similar drawing and symbolic practices. This article will use *NASA's Fault Tree Handbook with Aerospace Applications* [Ref. 1] as a guiding standard. It is assumed that the reader is familiar with Fault Tree Analysis, and is therefore acquainted with the format, logic and purpose of the symbols that are typically used. Consequently, only a high-level summary of the fault tree is included. For a concise introduction to FTA, see Barlow and Lambert [Ref. 2]. If the reader desires a more complete background on the subject, see Ericson [Ref. 3].

A Brief Summary of Fault Tree Symbols and Terminology

A *fault tree* is composed of *events* and *failure paths*. The resulting structure appears as an inverted tree, with its trunk at the top. Failure is initiated through numerous *primary events* at all bottom levels of the tree. Those primary events that occur randomly are called basic events. Frequently they can be modeled as time-dependent random variables.

In the language of Operations Research, the fault tree is a network called a *directed graph*, and the events are called *nodes*. The fault tree display uses lines to visualize the failure pathways. A node's contents are displayed in a structure called a *cell*.

Failure flows upward as represented by connecting lines to *fault events* that are managed by logic nodes called *gates*. These gates usually receive failure signals from multiple failure paths. A given gate processes its input failure signals through OR, AND or other pre-specified logical rules. Based on those rules, failure propagates upward until it reaches the top gate, also called the *top event*. To help visualize the fault tree, consider the NASA example reproduced in Figure 1 [Ref. 1].

As seen in Figure 1, all nodes are given a description, a name and a type. The type is indicated through an icon specifying which kind of node is being defined along with its behavior. A high-level summary of node types used most often in fault trees can be found in Tables 1a and 1b.

A *fault event*, managed by a logic *gate*, is an abnormal state of the system, resulting from inputs taken from its immediate child nodes. Many types of gates

Table 1a — Fault Events Definition of Common Gate Types and Their Symbols.

Gate	Logic Definition	Data Code	Symbol
AND	True if and only if all of its direct child nodes are true.	*	
OR	True if at least one of its direct child nodes is true.	+	
Exclusive OR	True if an odd number of its direct child nodes is true.	X	
Combination*	True if of the "n" child nodes connecting to its parent node, "k" child nodes are true.	"k"	k
Sequential AND	True if all child nodes fail in the order they are defined.	<	
Inhibit	True if its input is true and the associated inhibit condition is on.	I	

* *Note that "k" is entered as a positive integer, generally a single numeric digit.*

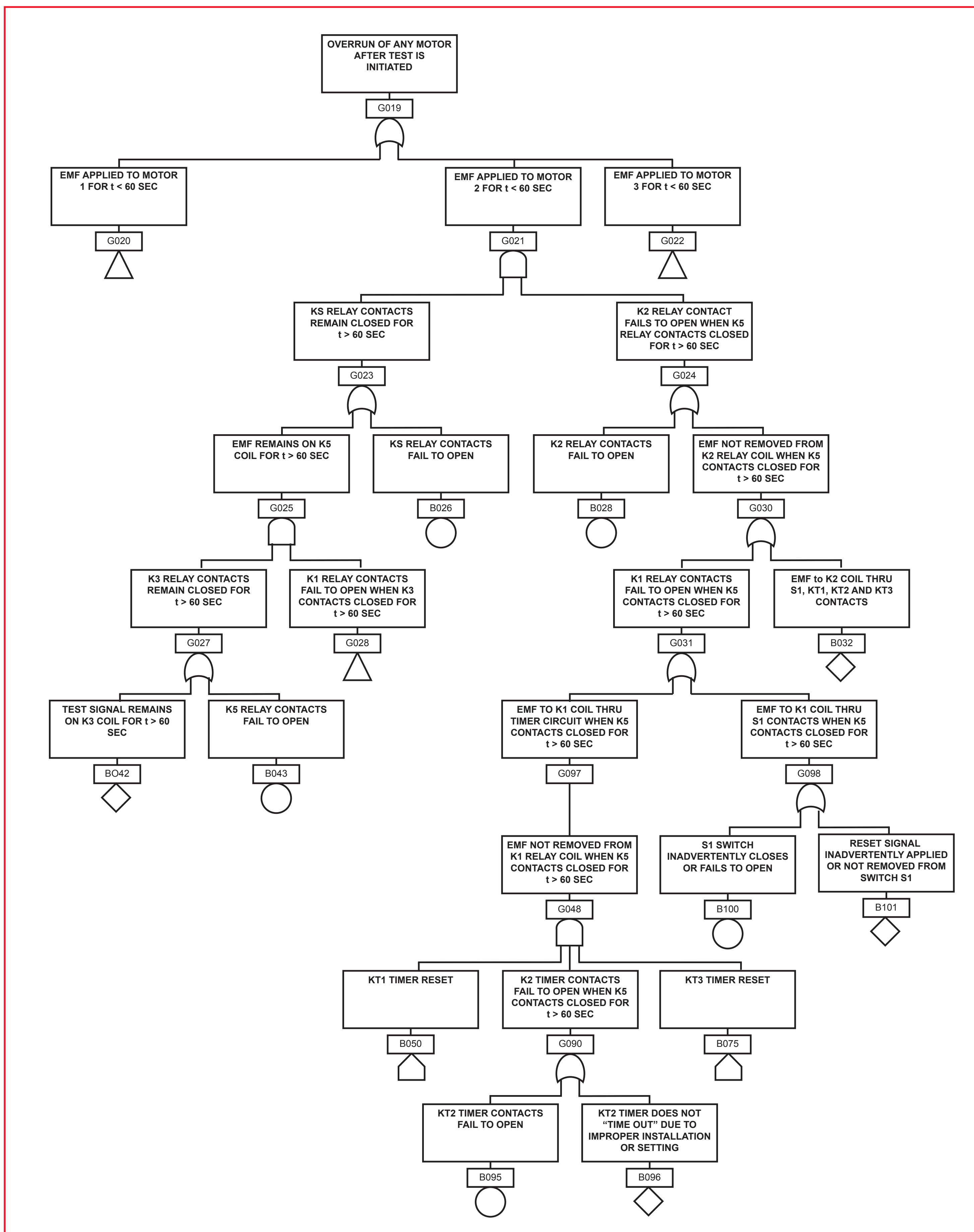


Figure 1 — A Typical Aerospace Fault Tree Using Fixed-sized Description Boxes [Ref. 1].

Table 1b — Primary Events Definition of Primary Event Types and Their Symbols.

Primary Event Type	Definition	Data Code	Symbol
Basic Event	A primary, inherent failure of a system element when operating within its design specifications.	#	○
Undeveloped Event	A basic or other event that has yet to be fully defined, or a secondary event.	U	◇
House	An event that is normally expected to occur based on system design and operation.	^	⌂
Inhibit Condition, also called Conditioning Event	A probability event tied to an inhibit gate that is applied as a multiplier to the failure probability associated with that gate. If the inhibit condition is attached to another type of gate, the inhibit gate is implied, operating after the indicated gate has completed its function.	&	⬭

exist. Each of these types is assigned a unique symbol as listed in Table 1a. When this abnormal state (failure) occurs, the gate is set to "true."

Later in this article the *transfer gate* will be introduced using the data code "T." It will be depicted as a triangle. Three instances of transfer gates are shown in Figure 1.

A *primary event* serves as a failure initiator. *Basic, undeveloped, house* and *conditioning events* are called primary events and are defined in Table 1b. These are part of a larger class called *terminating events,* a term which will be applied to any node that has no descendants in its display.

A basic event is typically parameterized using its *time to failure* and is modeled as a random variable. The failure time often employs a simple exponential distribution, but other more general probability distributions such as a Weibull distribution or piecewise exponential

Table 2a — Fault Events Supporting Figure 1.

Gate	Description	Type	Child Nodes
G019	OVERRUN OF ANY MOTOR AFTER EST IS INITIATED	+	G020 G021 G022
G021	EMF APPLIED TO MOTOR-2 FOR t>60 SEC	*	G023 G024
G023	KS RELAY CONTACTS REMAIN CLOSED FOR T>60 SEC	+	G025 B026
G025	EMF REMAINS ON K5 COIL FOR T>60 SEC	*	G027 G028
G027	K3 RELAY CONTACTS REMAIN CLOSED FOR T>60 SEC	+	B042 B043
G024	K2 RELAY CONTACT FAILS TO OPEN WHEN K5 RELAY CONTACTS CLOSED FOR T>60 SEC	+	B028 G030
G030	EMF NOT REMOVED FROM K2 RELAY COIL WHEN K5 CONTACTS CLOSED FOR t>60 SEC	+	G031 B032
G031	K1 RELAY CONTACTS FAIL TO OPEN WHEN K5 CONTACTS CLOSED FOR t>60 SEC	+	G097 G098
G097	EMF TO K1 COIL THRU TIMER CIRCUIT WHEN K5 CONTACTS CLOSED FOR t>60 SEC	+	G048
G048	EMF NOT REMOVED FROM K1 RELAY COIL WHEN K5 CONTACTS CLOSED FOR t>60 SEC	*	WB050 G090 WB075
G090	KT2 TIMER CONTACTS FAIL TO OPEN WHEN K5 CONTACTS CLOSED FOR t>60 SEC	+	B095 B096
G098	EMF TO K1 COIL THRU S1 CONTACTS WHEN K5 CONTACTS CLOSED FOR t>60 SEC	*	B100 B101

distribution may fit better assuming the distribution's parameters are known or can be estimated. Some applications use the basic event's failure probability directly rather than its failure rate.

Preparing Fault Tree Data

The names in the *Gate* column of Table 2a form an exhaustive list of all the fault events used in Figure 1. Their names are unique. Each gate's *description* and *type* follow the gate name in its row. These same nodes are also listed in the *Child Nodes* column, except for the top event (G019 in this example). Any child node that is not also found in the gate column is a primary event. Each row forms the immediate cause and effect relationship between the parent and child nodes. The order in which the child nodes are listed is maintained in the fault tree graphic, and potentially is used in subsequent analysis.

The data in Table 2a is sufficient to build the structure of the fault tree shown in Figure 1, although it is missing the primary events' descriptions and types. Table 2b completes the data for each of these primary events.

In addition to the data needed to support the fault tree graphic, failure rates (or probabilities in some cases) are also included. The presence of several values implies the existence of multiple time phases, which may be needed in subsequent probabilistic modeling. Such modeling is not covered in this article.

Building the Fault Tree Display

The tree in Figure 1 has good visual appeal having the following attributes:

- Each displayed cell consists of its description, name and type.
- Centering each parent cell between its first and last child cells, and connecting child to parent with lines creates balance and clearly illustrates the failure paths.
- Each description allocates space for up to four text segments. A fixed-sized box surrounds this space.
- This description box is followed vertically by the node name in its own small box, which is then followed by its symbol.
- The horizontal distance between adjacent cells is minimized, given a predefined minimum gap between them.
- Additional space is added between cells where a wider spread is needed to accommodate its branch substructure.

Although this format appears optimal, its rigidity causes inefficiency in the display. Rather than allocating a fixed size for all text boxes, a better use of space is possible by permitting each box to have a width that is tailored to its content. This allows improvements that can produce the following benefits:

Table 2b — Primary Event Definitions Supporting Figure 1

Event	Title	Type	Rates or probabilities
G020	EMF APPLIED TO MOTOR 1 FOR t>60 SEC	T	0.015 0.010
G022	EMF APPLIED TO MOTOR 3 FOR T>60 SEC	T	0.025 0.035
G028	K1 RELAY CONTACTS FAIL TO OPEN WHEN K3 CONTACTS CLOSED FOR t>60 SEC	T	0.005 0.015
B032	EMF TO K2 COIL THRU S1, KT1, KT2 AND KT3 CONTACTS	#	0.020 0.050
B096	KT2 TIMER DOES NOT "TIME OUT" DUE TO IMPROPER INSTALLATION OR SETTING	#	0.015 0.015
B101	RESET SIGNAL INADVERTENTLY APPLIED OR NOT REMOVED FROM SWITCH S1	U	0.045 0.075
B026	KS RELAY CONTACTS FAIL TO OPEN	#	0.01 0.02
B042	TEST SIGNAL REMAINS ON K3 COIL FOR t>60 SEC	U	0.02 0.01
B043	KS RELAY CONTACTS FAIL TO OPEN	#	0.02 0.02
B028	KS RELAY CONTACTS FAIL TO OPEN	#	0.01 0.01
B095	TKT2 TIMER CONTACTS FAIL TO OPEN	#	0.005 0.005
B100	S1 SWITCH INADVERTENTLY CLOSES OR FAILS TO OPEN	#	0.020 0.015
WB050	KT1 TIMER RESET	^	1.0 0.0
WB075	KT3 TIMER RESET	^	0.0 1.0

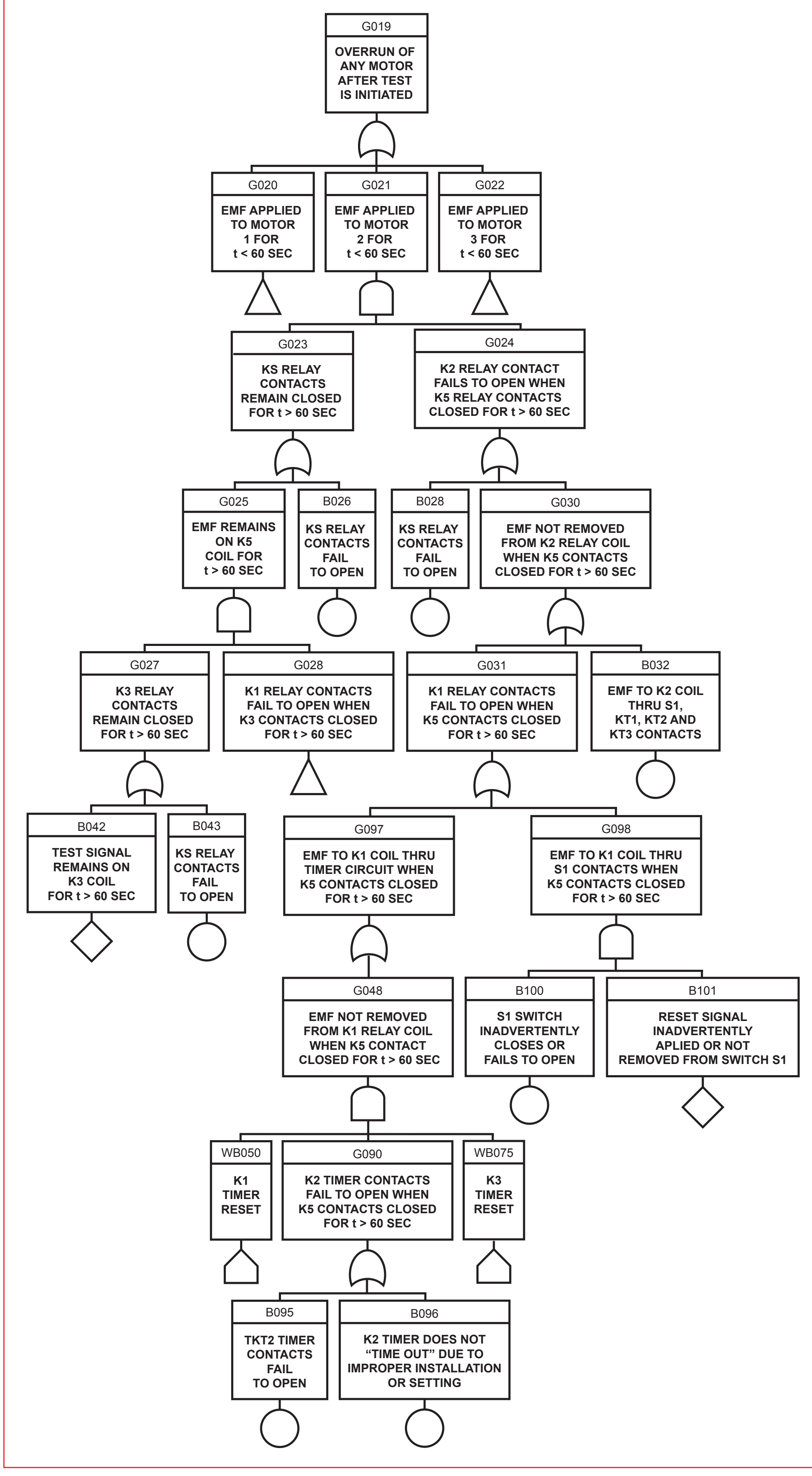


*Figure 2 — Re-plot of Figure 1 Using Variable Cell Widths.**

** Figure 2, and all subsequent figures are created algorithmically using a Python program designed to make use of the new constraints and algorithms referenced in the text.*

- Reduced layout size of the tree
- Enlarged font size and, consequently, improved print clarity
- Cell-specific display attributes, tailored only to that which is required by each node

To develop an optimal display, the fault tree should adhere to a balanced design, use as little space as possible, and eliminate any risk of a cell colliding with another. A horizontal *collision* occurs when any cell intended to be displayed to the right of another cell fails to achieve a minimum gap between them. Such a collision can cause the right cell to overlap the left, or even precede it. Particularly when variable width cells are allowed, care must be taken to avoid horizontal collisions. Vertical collisions are easy to avoid since the requirement that cells are of equal depth will be maintained.

An improved fault tree display can be achieved by managing several constraints among all neighboring cells. It is required that all cells have the same depth, which usually mandates breaking node descriptions into a predefined number of line segments. An algorithm can be devised to break text into these segments, making them all of roughly equal length. This effectively populates the cell depth, but requires flexibility regarding cell width. The spaces between words can be used as candidates for the segment breaks. Each cell width is therefore determined only by its own content and is not influenced by that of any other cell.

A second requirement involves the length and use of the node name. The tree in Figure 1 separates the node name from the node description and uses a small fixed-size box to house its name. Without customizing the length of this box, part of its name could be displayed outside its box. In Figure 2, the node name is

combined with its description as a separate text segment, and all five segments are managed jointly. This eliminates the risk of a collision between the name and the box surrounding it, thereby avoiding a separately managed plot requirement.

As a third requirement, failure paths merge as they approach their common parent. The resulting structure requires less layout space, but with no loss of failure path visibility.

The fault tree in Figure 2 has the same content as Figure 1, but Figure 2 has several important features, noted here:

- Each cell divides its node description into four comparably sized text segments, except where there are not enough words available to create all four segments.
- Text divisions are allowed only between words.
- The node name appears within the node cell, occupying its own dedicated space that is separated from the node description using a horizontal line.
- The width of each cell box is determined only by the lengths of its text segments, plus a consistent margin. This avoids the risk of any box colliding with its associated text.
- Connecting lines that join child with parent cells are simplified into a single failure flow structure.
- The center of each parent cell is consistently positioned halfway between the leftmost and rightmost child cell box centers.
- Adjacent cells all have a common gap, except where larger gaps are needed to avoid plot collisions among descendent substructures.
- No cell collides with its neighboring cell, even though the cell widths vary.

Creating Horizontal Coordinates Using a System of Linear Equations

Achieving all these features listed here can be a nightmare — especially given the last bullet — unless one can simultaneously manage the horizontal plot coordinates for all cells as a system of linear equations. To understand the specifics of the algorithm it is useful to demonstrate the approach using a very small fault tree. Table 3 provides the data necessary to create the simple seven-node tree shown in Figure 3. For simplicity, the cell widths will be constant. However, the algorithm does not require this limitation, as illustrated by Figure 2.

Each horizontal coordinate, referenced by its node name, is measured as the number of pixels from its horizontal center to the "TOP" node's horizontal center. This coordinate system permits both positive and negative entries, which will later need shifting to develop usable (positive) pixel locations. To create the system of equations that produces these coordinates, the following criteria will be applied:

Within Branch Criteria

1. Parent/child centering
   a. GA1 must be in the middle of XA1 and XA2.
   b. GA2 must be in the middle of XA3 and XA4.
   c. TOP must be in the middle of GA1 and GA2.

2. Adjacent node separation
   a. XA2 must exceed XA1 by the number of pixels required to form the desired node spacing (or minimum gap).
   b. The same requirement exists for XA3 and XA4 as in 2a.

Adjacent Branch Criteria

3. The substructures of GA1 and GA2 must not collide at any level.
   a. GA2 must exceed GA1 by at least the minimum gap.
   b. XA3 must exceed XA2 by at least the minimum gap.

Plot width optimization

4. The substructures of GA1 and GA2 must have a minimum separation at some level, choosing the one criterion below that does not result in collisions.

Table 3 — Input for Figure 3

TOP	+	GA1	GA2
GA1	*	XA1	XA2
GA2	*	XA3	XA4

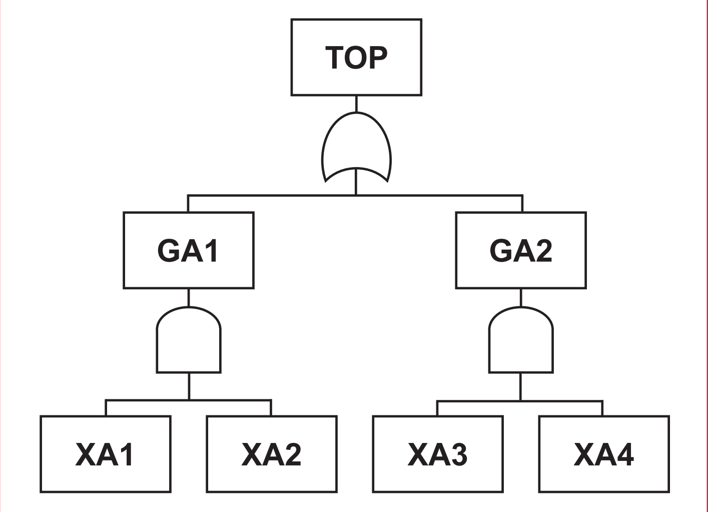


Figure 3 — A Very Small Fault Tree.

a. GA2 must exceed GA1 by the minimum gap with no specification for the spacing between XA2 and XA3.

– OR –

b. XA3 must exceed XA2 by the minimum gap with no specification for the spacing between GA1 and GA2.

Linear equations can be formed using criteria 1, 2 and 3 and either 4a or 4b (but not both). The selected system will serve as the plot width minimization requirement. For the purpose of developing the algorithm, the notion of a branch must be defined.

Definition: *A* branch *is a group of nodes that includes a specified top node (through which the branch is named) and selected descendants of that top node. These nodes are found in a top/down, left/right recursive order starting with the top. A given branch terminates when a descendent row is found to contain solely terminating events (primary events, transfer gates or repeated uses of multiple occurring events, or MOEs)* All terminating events in a given row must belong to the same branch as that of their parent. Any gate found following the termination condition must belong to a subsequent branch.*

In this example there are two branches. The branch starting at TOP contains nodes TOP, GA1, XA1 and XA2. The row containing XA1 and XA2 terminates this branch. The branch starting at GA2 contains nodes GA2, XA3 and XA4, with the row XA3 and XA4 acting as its terminating condition.

In this simple example, a width of 36 horizontal pixels nicely accommodates the three character node names, while leaving an aesthetically appealing edge margin within each box. Adding a gap of eight pixels creates adequate space between adjacent boxes. This means that the minimum horizontal separation between the centers of adjacent cells is 44 pixels. Using this spacing, linear constraints for the horizontal coordinates can be constructed as follows:

Tree Reference Point

0) TOP = 0

Within Branch Parent / Child Centering

1a) $GA1 = \frac{XA1+XA2}{2} \rightarrow 2 * GA1 - XA1 - XA2 = 0$

1b) $GA2 = \frac{XA3+XA4}{2} \rightarrow 2 * GA2 - XA3 - XA4 = 0$

1c) $TOP = \frac{GA1+GA2}{2} \rightarrow 2 * TOP - GA1 - GA2 = 0$

Within Branch Adjacent Node Criteria

2a) $XA2 = XA1 + 44 \rightarrow XA2 - XA1 = 44$

2b) $XA4 = XA3 + 44 \rightarrow XA4 - XA3 = 44$

Adjacent Branch Criteria

A different kind of problem exists here, since this requirement produces inequalities. A tool that is developed in the Simplex algorithm [Ref. 7], namely the *slack variable*, will prove helpful. In this case, two inequalities must be maintained, creating the need for two non-negative slack variables S1 (the excess distance beyond the minimum spacing of GA2 and GA1) and S2 (the excess distance beyond the minimum spacing of XA3 and XA2). These inequalities may now be written as equations.

3a) $GA2 - GA1 > 44 \rightarrow GA2 - GA1 = S1 + 44 \rightarrow GA2 - GA1 - S1 = 44$

3b) $XA3 - XA2 > 44 \rightarrow XA3 - XA2 = S2 + 44 \rightarrow XA3 - XA2 - S2 = 44$

There are now eight equations with nine variables, including the two slack variables. Upon review of the criteria it is clear that only one slack variable should be used. Inspecting figure Figure 1 shows that if *GA1* and *GA2* are set to the minimum gap then XA2 and XA3 will collide. It is tempting therefore to change equation 3b to: $XA3\text{-}XA2=44$, resulting in an equal number of equations and unknowns. However, this knowledge depends on having seen the finished fault tree which does not yet exist. Consequently, both slack variables must be carried by the algorithm until one can be eliminated without any prior knowledge of the completed tree display.

Table 4 shows the final system of equations using eight equations and nine unknowns.

By adding either the constraint S1 = 0 or S2 = 0 (but not both), the above system can be solved. One may use MATLAB (https://www.mathworks.com/), a linear algebra package, or apply an LU (lower-upper) or other matrix decomposition directly in a programming language such as C++ or Python [Refs. 5, 6 & 8]. The author's preferred approach is to program a tableau much like that employed by the Simplex algorithm using Python. This allows an LU decomposition to be strategically managed by selecting a best guess of which slack variable

** This branch termination criterion disallows overlapping branches, making all branches graphically disjoint.*

Table 4 — Summary of Constraints Governing Cell Plot Locations.

Constraint		Purpose
0:	TOP = 0	TOP is anchored to a specific horizontal pixel position
1:	2*TOP – GA1 – GA2 = 0	TOP is centered between its child nodes, GA1 & GA2
2:	2*GA1 – XA1 – XA2 = 0	GA1 is centered between its child nodes, XA1 & XA2
3:	2*GA2 – XA3 – XA4 = 0	GA2 is centered between its child nodes, XA3 & XA4
4:	XA2 – XA1 = 44	XA1 and XA2 are separated by the minimum gap
5:	XA4 – XA3 = 44	XA3 and XA4 are separated by the minimum gap
6:	GA2 – GA1 – S1 = 44	GA1 and GA2 are separated by the minimum gap + S1
7:	XA3 – XA2 – S2 = 44	XA2 and XA3 are separated by the minimum gap + S2

Table 5 — Tableau Associated with Table 4 Equations

row	TOP	GA1	XA1	XA2	GA2	XA3	XA4	S1	S2	b
0	1	0	0	0	0	0	0	0	0	0
1	2	-1	0	0	-1	0	0	0	0	0
2	0	2	-1	-1	0	0	0	0	0	0
3	0	0	0	0	2	-1	-1	0	0	0
4	0	0	-1	1	0	0	0	0	0	44
5	0	-1	0	0	1	0	0	-1	0	44
6	0	0	0	-1	0	1	0	0	-1	44
7	0	0	0	0	0	-1	1	0	0	44

to eliminate, solving the resulting system, and if any collisions occur, using the solution to identify a better choice for the eliminated slack variable. The final solution is achieved iteratively quite quickly even in far more complex trees than the one used here. If S1 is eliminated, a collision will occur between *XA*2 and *XA*3 revealed by S2 calculated to be negative. This provides an algebraic reason to eliminate S2 rather than S1 in the next iteration.

Table 5 shows the tableau applicable to the system identified in Table 4, after identifying S2 for elimination. The header row, less S2, is interpreted as vector x^T. The un-shaded portion of the tableau under x^T is matrix A. The outcome vector b is as shown. Now the matrix equation Ax = b can be solved to determine x.

The solution to this system is:

$$x^T = [TOP, GA1, XA1, XA2, GA2, XA3, XA4, S1] = [0, -44, -66, -22, 44, 22, 66, 44]$$

This solution must be shifted to the right by 67 (one pixel above the minimum value) to achieve positive horizontal pixel locations.

Generalizing the Approach

The method used in the previous example can be generalized to accommodate an arbitrary tree. This generalization is too tedious to be fully described in this paper, but the results applied to the NASA aerospace example introduced using Figure 1 are shown in Table 6.

Vector b is no longer set to only two distinct values as it was in Table 5, but rather to values tied to the length of the largest text segment for each node. This allows the cell widths to vary. Such tailoring of the b vector is the only impact on the linear system when allowing each cell's width to be tailored to its content. Care must be taken to assure that elements in b reflect the cell-center to cell-center horizontal distance (in pixels).

One should observe that the matrices shown in Tables 5 and 6 have sparse structures, and contain many zeros. This sparsity increases as the fault tree gets larger. There will never be more than three non-zero entries per row, but the row and column dimensions of the matrix increase somewhat proportionally to the number of nodes in the tree. Consequently, a sparse matrix manager should be used to solve these systems.

For large trees, the iterative approach used here requires repeatedly solving very similar linear systems. This may require excessive memory and computation time. Using a sparse matrix method helps significantly. A matrix modification method can be used to quickly re-solve systems that have minor column modifications. Sparse matrix and matrix modification methods are discussed by Duff, Erisman and Reid, who give a complete treatment regarding rapidly solving small perturbations of a system when the original solution and its LU decomposition exist [Ref. 9].

Multi-page Fault Trees

Many industrial safety and reliability models involve hundreds or thousands of nodes. The resulting fault trees become quite large and often require conference room walls to display them. Even when large print devices are available, the resulting fault trees and their probabilistic analyses must find their way to a report that needs to fit on standard-sized sheets of paper. This section adapts the system of equations model repeatedly in order to logically divide sub-trees of a large fault tree, making each such sub-tree a size commensurate with a pre-specified output dimension.

Choosing nodes that will serve as the tops of page breaks in the graph can effectively divide a large tree into a series of interrelated smaller ones. The choice of where to use such breaks should be influenced by intrinsic subsystems, different data sources or even visual aesthetics. *Transfer gates* are a way to organize multiple pages, each with a single multi-page graph, or reference a different source altogether. These gates need to provide a page number (or file name) where its associated plot can be found. The triangle is the legacy symbol for a transfer. When a triangular transfer symbol appears to the left of a cell, failure is being transferred out of the given page and into the page indicated inside the triangle. When the transfer symbol appears below a cell, failure is being flowed from the indicated page into that cell.

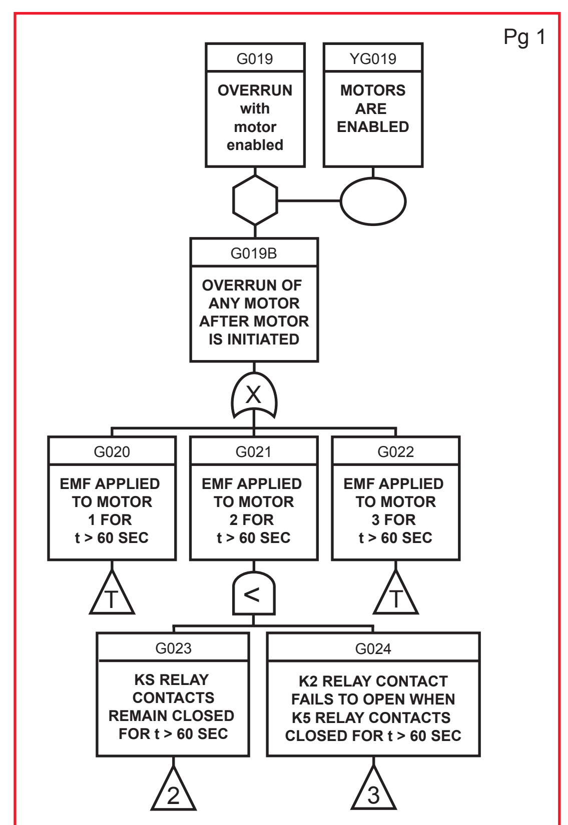
To illustrate managing large displays, the fault tree in Figure 2 will be broken into four pieces through the use of transfer gates. The result is shown in Figure 4 as a series of small plots, each on its own page. Generally, multiple pages would only be used on much larger trees, but this example will serve in showing how pages interact with each other. To display some additional possible improvements, minor changes made from the original fault tree serve to present alternate display symbols for consideration.

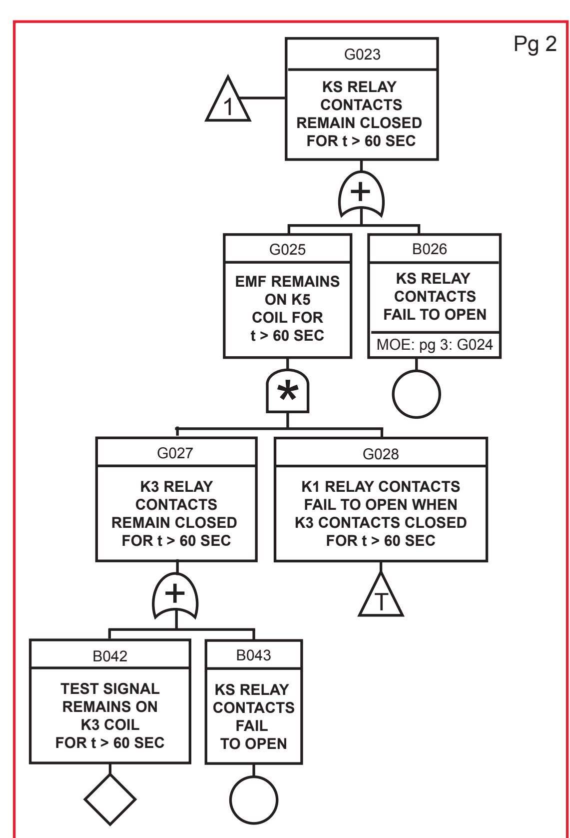
Table 6 — Sparse System of Linear Equations Tableau to solve for horizontal plot coordinates of the tree in Figure 2.*

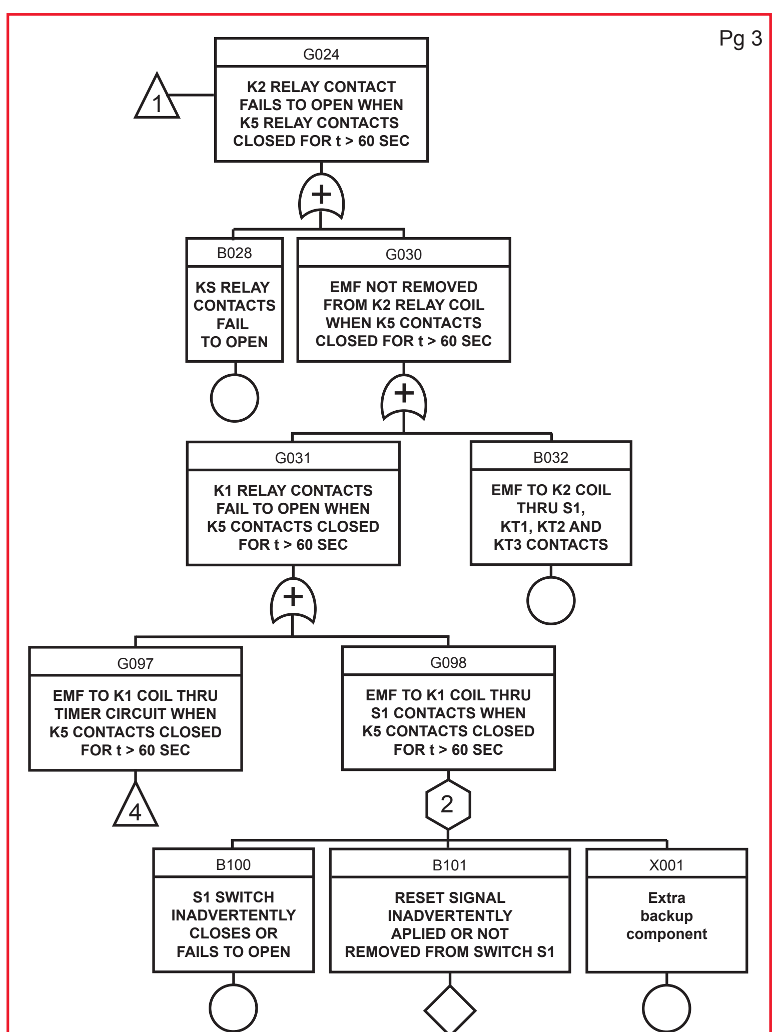
Fault Tree Display Optimization Tableau

	Plot																										Slack						
	G019. 00	G020. 01	G021. 02	G023. 03	G025. 04	G027. 05	B042. 06	B043. 07	G028. 08	B026. 09	G024. 10	B048. 11	G030. 12	G031. 13	G097. 14	G048. 15	WB050 16	G090. 17	B095. 18	B096. 19	WB075 20	G098. 21	B100. 22	B101. 23	B032. 24	G022. 25	S6... 31	S2... 27	S3... 28	S4... 29	S5... 30	S1... 26	b
0	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0
1	2.0	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1.0	.	.	.	.	.	.	-8
2	.	.	2.0	-1.0	.	.	.	.	.	.	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	40
3	.	.	.	2.0	-1.0	.	.	.	.	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-28
4	.	.	.	.	2.0	-1.0	.	.	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	36
5	.	.	.	.	.	2.0	-1.0	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-24
6	.	.	.	.	.	.	.	.	.	.	2.0	-1.0	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-44
7	.	.	.	.	.	.	.	.	.	.	.	.	2.0	-1.0	.	.	.	.	.	.	.	.	.	.	-1.0	.	.	.	.	.	.	.	-24
8	.	.	.	.	.	.	.	.	.	.	.	.	.	2.0	-1.0	.	.	.	.	.	.	-1.0	.	.	.	.	.	.	.	.	.	.	8
9	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.0	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0
10	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2.0	-1.0	.	.	.	-1.0	.	.	.	.	.	.	.	.	.	.	.	-112
11	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2.0	-1.0	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	-20
12	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2.0	-1.0	-1.0	.	.	.	.	.	.	.	.	-12
13	.	-1.0	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	108
14	.	.	.	.	.	.	-1.0	1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	116
15	.	.	.	.	.	-1.0	.	.	1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	124
16	.	.	.	.	-1.0	.	.	.	.	1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	108
17	.	.	.	-1.0	.	.	.	.	.	.	1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1.0	124
18	.	.	.	.	.	.	.	.	.	-1.0	.	1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1.0	.	.	.	.	84
19	.	.	.	.	.	.	.	.	.	.	.	-1.0	1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	84
20	.	.	.	.	.	.	.	.	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	164
21	.	.	.	.	.	.	.	-1.0	.	.	.	.	.	.	1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1.0	.	.	84
22	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1.0	1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	60
23	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1.0	1.0	.	.	.	.	.	.	.	.	.	.	.	.	100
24	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1.0	.	.	1.0	.	.	.	.	.	.	.	.	.	.	.	164
25	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1.0	.	.	.	.	.	.	1.0	.	.	.	.	.	.	.	.	-1.0	.	172
26	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1.5	.	.	.	.	.	.	1.0	.	.	.	-1.0	.	.	.	.	.	172
27	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1.0	1.0	.	.	.	.	.	.	.	.	124
28	.	.	.	.	.	.	.	.	.	.	.	.	.	-1.0	.	.	.	.	.	.	.	.	.	.	1.0	.	.	.	.	.	.	.	164
29	.	.	-1.0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.0	.	.	.	.	.	.	108

** A period in the table represents a logical zero. It occupies no computer memory if a sparse matrix manager is used.*







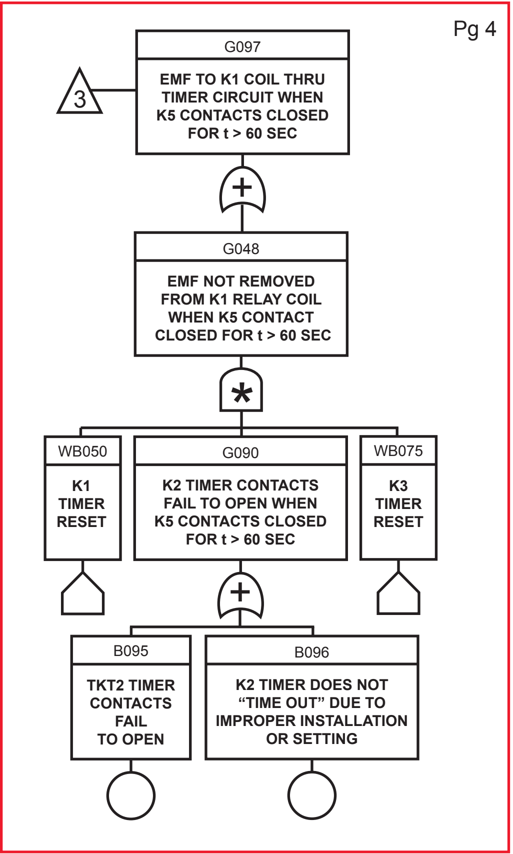


Figure 4 — Revised Typical Aerospace Fault Tree using Multiple Pages

These features, particularly when MOEs are involved, show how specific information that is not used everywhere throughout the tree can be added easily to cells as needed. In such cases, expanding the cell width makes the extra space needed for an additional text segment. This is another direct benefit of having variable sized cells.

By making use of the transfer gate symbols, one can identify where the transfers occur, thus permitting large fault trees with complex logic to be traceable in a document.

Computational Considerations

In addition to providing significant organization to large fault trees, multiple pages also have the advantage of breaking fault trees into a group of smaller, less complex sub-trees. The group can be computationally managed with much more efficiency than combining the entire tree into a single plot. The reason for this is that the underlying LU or other matrix solver is an order of n^3 algorithm where n is the number of nodes; or perhaps at best, order of n^2 if sparse matrix methods are used. Therefore, a series of smaller plots greatly reduces processing time in comparison to maintaining a single-tree display.

The methods described in this paper tend to be highly recursive and use much dynamic memory allocation. This logic is nicely accommodated through recursive programming techniques using any modern object-oriented programming language. However, because of its memory management system, modern programming constructs and its compact but not cryptic syntax, the Python interpreter is viewed in this paper as the programming tool of choice. That said, a native compiler would produce faster-running code.

Conclusion

This article has sought to make useful improvements to fault tree displays. Fault tree technology has been a safety and reliability tool that has stood the test of time. By keeping the structure, symbols and intent of the fault tree intact, improvements in the display present an opportunity to add clarity and improved space utilization to its graphical presentation. It also provides an opportunity to make the fault tree structure tailored from cell to cell to support specific information that may not need to be uniformly included throughout the entire tree.

The author invites any questions regarding the methods used and ideas regarding their application.

Acknowledgements

Special thanks are due to Dr. Albert M. Erisman for his consultation regarding the matrix modification method. This method can be applied to the paper's iterative optimization algorithm, significantly reducing its computational needs.

Special thanks are also due to Clifton A. Ericson II for his review and guidance in current fault tree practices used in industry and major reliability software developers.

About the Author

Mr. Thomas C. Bingham is a career statistician retired from 38 years at The Boeing Company. He received bachelor degrees in mathematical statistics and physics, and a master of science in applied mathematics from the University of Washington. His professional experience includes industrial statistics applied to engineering, manufacturing and quality assurance. During his service at Boeing Research and Technology he helped design and program internal fault tree software. In retirement Mr. Bingham and his wife share their time between Washington State and Costa Rica.

References

1. Stamatelatos, Michael, William Vesely, Joanne Dugan, Joseph Fragola, Joseph Minarick III and Jan Railsback. "Fault Tree Handbook with Aerospace Applications," National Aeronautics and Space Administration (NASA), pages 33-45, retetrieved from http://www.barringer1.com/mil_files/NASA-FTA-1.1.pdf.
2. Barlow, R.E and H.E. Lambert. "Introduction to Fault Tree Analysis," *Reliability and Fault Tree Analysis*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1975.
3. Ericson, Clifton A. *Fault Tree Analysis Primer,* pp 1-18, CreateSpace Independent Publishing Platform, 2011.
4. Vesely, W.E, F.F. Goldberg, N.H. Roberts and D.F. Haasl. "Fault Tree Handbook, (NUREG-0492), U.S. Nuclear Regulatory Commission, 1981.
5. Trefethen, Lloyd N. and David Bau, III, *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, pp 69-76, 1997.
6. Stewart, G.W. *Introduction to Matrix Computations*, Academic Press, Inc., Cambridge, Massachusetts, pp 144-147, 1973.
7. Spivey, W. Allen and Robert M. Thrall. *Linear Optimization*, Holt, Rinehart and Winston, Inc., New York, New York, p 61ff, 1970.
8. Cheney, E. Ward and David R. Kincaid. *Numerical Mathematics and Computing, 7th Edition*, Cengage Learning, Boston, Massachusetts, pp 70-78, 82-89, 2012
9. Duff, I.S, A.M. Erisman and J.K. Reid. *Direct Methods for Sparse Matrices, 2nd Edition,* Oxford Science Publication, Oxford, U.K., 2017.