Implementation and Application
of the Fundamental Theorem of Probability

by

Jeremy S. Cohen

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of Mechanical Engineering
and Bachelor of Science in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 22, 1998

© Copyright 1998 Jeremy S. Cohen. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant to others the right to do so.
Implementation and Application
of the Fundamental Theorem of Probability
by
Jeremy S. Cohen

Submitted to the Department of Electrical Engineering and Computer Science
May 22, 1998
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Engineering and Computer Science

ABSTRACT
The “RIK” (Reasoning with Incomplete Knowledge) algorithm, a mathematical
programming based algorithm for performing probabilistic inference on (possibly)
incompletely specified systems of discrete events is reviewed and implemented.
Developed by Myers, Freund, and Kaufman, it is a tractable reformulation of the
computational approach implicit to the Fundamental Theorem of Probability as stated by
De Finetti and extended by Lad, Dickey and Rahman. Enhancements to the original
algorithm are presented and several applications of the algorithm to real-world systems
including fault trees and belief networks are explored. The system is solved successfully
for moderately large problems, providing practical information for system designers
coping with uncertainty.

Thesis Supervisors:

Robert Freund
Director
Operations Research Center

Gordon Kaufman
Professor of Management
Alfred P. Sloan School of Management
Acknowledgements

I would like to thank my advisors Rob Freund and Gordon Kaufman for their guidance and encouragement during the course of this project. It has been a privilege to explore the intersection of mathematical programming and probability theory with men with such knowledge of and passion for their respective fields. I greatly enjoyed working with and learning from both of you. I would also like to thank Fernando Ordonez who has been a tremendous help with his keen insights and long hours of development work and debugging. Additionally, I would like to thank Tracy Myers for developing the algorithm to make my work possible and being an inexperienced enough C programmer to make my work necessary. Finally, I would like to thank my fiancée Fraya for her patience, support, and excellent taste in TV dinners.
Table of Contents

1 Problem Statement .................................................................................. 6

2 Introduction .......................................................................................... 8

3 Background ........................................................................................... 9

3.1 Fault Tree Analysis ............................................................................. 9

3.1.1 The Fault Tree .............................................................................. 10

3.1.2 Assumptions ................................................................................. 11

3.1.3 Analysis ......................................................................................... 11

3.1.4 Sample Problem: Elevator Safety ................................................. 13

3.1.5 Pitfalls ......................................................................................... 14

3.1.6 Further References ........................................................................ 15

3.2 Bayesian Inference Networks .............................................................. 15

3.2.1 Formal Definition .......................................................................... 15

3.2.2 Reasoning Bayesian Style ............................................................ 16

3.2.3 Sample Problem: Medical Diagnosis ............................................. 17

3.2.4 Further References ........................................................................ 19

4 The RIK Algorithm ............................................................................... 20

4.1 Simple Example ................................................................................. 20

4.2 Notation and Formal Problem Statement .......................................... 21

4.3 The Fundamental Theorem of Probability ...................................... 21

4.4 Overview of the RIK Algorithm ....................................................... 23

4.4.1 LP Notation .................................................................................. 23

4.4.2 The Related Linear Program (RLP) ............................................. 23

4.4.3 The Revised Simplex Algorithm .................................................. 24

4.4.4 A Compact Representation of the Outcomes and The Related Integer Program (RIP) .................................................. 25

4.4.5 The Related Integer Program (RIP) ............................................ 26

4.4.6 Flow of the RIK Algorithm ........................................................ 26

4.4.7 Extensions to the FTP .................................................................. 28

5 Implementation .................................................................................... 30

5.1 General Architecture ........................................................................ 30

5.1.1 RLPdata ...................................................................................... 30

5.1.2 RIPdata ....................................................................................... 32

5.1.3 FTPdata ...................................................................................... 32

6 Applications ....................................................................................... 39

6.1 The Elevator Fault Tree .................................................................... 40

6.1.1 Formulating the Fault Tree in RIK format ................................... 40

6.1.2 Solving the RIK Algorithm ........................................................ 40

6.1.3 General Fault Tree Analysis Conclusions ................................... 41

6.2 The Medical Diagnosis Problem ...................................................... 42

6.2.1 Problem Statement ....................................................................... 42

6.2.2 Direct Analysis with a Probability Table ..................................... 43

6.2.3 Formulating as an RIK Problem .................................................. 44

6.2.4 Other Considerations .................................................................... 46

6.2.5 Conclusions .................................................................................. 46

6.3 Reconfigurable Fault Detection Systems .......................................... 47
Chapter 1

1 Problem Statement

“If a man will begin with certainties, he shall end in doubts, but if he will be content to begin with doubts, he shall end in certainties.”

- Pliny

Man’s desire to understand the nature of uncertain systems is age-old. From the picnicking family checking the weather forecast, to the fantasizing Black-Jack player, to the team of highly-paid geologists prospecting for fossil fuels, our ability to make decisions hinges largely on our assessment and analysis of the uncertain.

Unfortunately, most uncertain systems are extremely complex. As a result, most analytical techniques require the user to make broad independence assumptions or to model uncertain components as conforming to standard, easily-analyzable, probability density functions. We are satisfied to approximate the outcome of the system by assuming away vague or tricky relationships and making a best guess of future value.

This thesis addresses the problem of system complexity by relaxing the need for broad assumptions. Instead, we utilize The Fundamental Theorem of Probability, as stated by DeFinetti and reformulated by Frank Lad [4], to infer exact bounds on the probability of the system event we care about. Rather than assume more information than we know, we exploit only the (possibly incomplete) knowledge we are given. The tradeoff is that we can only speak in terms of best and worst case scenarios for our event, rather than point estimates. Effectively, we trade a focused approximation for (possibly) loose truth.

More formally, the problem we solve is the following:
(1.1) *The Exact Probabilistic Inference Problem (EPI)*

Given a binary target event $x_{N+1}$, a set $x = \{x_1, \ldots, x_N\}$ of logically and/or probabilistically related events, and probability assessments for some events in $x$,

a) determine whether or not assessments for elements of $x$ are coherent and if so,

b) infer upper and lower bounds on the probability of the target event $x_{N+1}$, implied by probability assessments for elements of $x$. 
Chapter 2

2 Introduction
My thesis work was to implement an algorithm capable of solving the Exact Probabilistic Inference problem of (1.1) and to develop and compute some real-world examples. The algorithm I implemented was developed in concert by Robert Freund, Gordon Kaufman, and Tracy Myers and was the subject of Tracy’s Ph.D. thesis of 1995 [1]. The algorithm uses mathematical programming techniques to solve the EPI problem and is a tractable reformulation of DeFinneti’s Fundamental Theorem of Probability (FTP), which suggests an intractable linear programming solution.

To motivate this undertaking, I will first give some background information on other techniques available for solving similar problems. Then I will state the framework for the problem formally and present the RIK algorithm. Lastly, I will discuss implementation techniques and issues and explore some applications of the FTP.
Chapter 3

3 Background
Most techniques for solving probabilistic inference problems are based on special
dependence or independence assumptions and standard probability distributions. Two
well-developed solution techniques for dealing with complex real world inference
problems are fault tree analysis and Bayesian Networks. A discussion of the analytical
process and requisite assumptions for both of these approaches to inference follows. For
a solid grounding in the history and philosophy of logic and probability, see Operational
Subjective Statistical Methods by Frank Lad [4].

3.1 Fault Tree Analysis
Fault tree analysis (FTA) focuses on finding the probability of one (usually undesirable)
target event based on probabilistic assessments for related system events and an
understanding of the causality relationships among these events. Typically, each event
represents a failure of one or more of the system’s components. To analyze the fault
system, we construct a tree of faults, with assessments of failure probabilities assigned to
the leaf events. By propagating the faults up the tree along paths of causality, we are able
to estimate the reliability of the top-level event.

The technique grew out of safety concerns of the NEA Committee on the Safety of
Nuclear Installations (now the Nuclear Regulatory Commission) especially after the
nuclear plant accident at Three Mile Island in 1979:
“The Three Mile Island accident consisted of a sequence of plant failures and diagnostic errors and led to a succession of physical situations that had not been studied in sufficient depth, largely because safety assessments had been focused on design basis accidents and the single failure criterion. Most accidents of this kind happen very quickly, so attention had been focused on rapidly occurring events. By contrast the Three Mile Island accident, like the most previous reactor incidents (e.g. that at the Browns Ferry plant), evolved over a number of hours.” – The Response to the Three Mile Island Accident [9]

Fault trees are widely used for safety analysis and software reliability, and have even been used to model bridge deterioration [8]. In addition, they enable a system engineer to make informed decisions about low-level components of a system during the design phase.

3.1.1 The Fault Tree

The entire fault tree inference problem is represented by the tree itself. The top-level event, our target for inference, is connected to child events through logic gates. Each gate translates input events into a single output event. Typical gates include:

- AND: Output event occurs only if all input events occur.
- OR: Output event occurs if any input event occurs.
- Exclusive OR: Output event occurs if only one of two input events occurs.
- K out of M: Output event occurs if at least K of the M input events occur.
- Inhibit: An AND gate where one input is a lower fault event and the other a conditional qualifier.

Each child fault event, in turn, may or may not have children of its own. A sample fault tree, describing the likelihood of passenger injury in a simple, electronically-controlled elevator is shown in Figure 3.1.

The tree requires an assessment of the probability of occurrence of each basic (leaf) event. Usually, a simple exponential waiting time distribution is used to model the fault behavior of system components with assessments being determined as the probability of failure within a fixed time interval, say 10,000 hours. The process of estimating the mean waiting time model from data is often the most time-consuming aspect of the analysis.
3.1.2 Assumptions

Traditional methods of fault tree analysis are based on the critical assumption that all basic events are independent. This assumption allows us to combine event probabilities as we propagate the failures up the tree. For example, the probability of the output event of an AND gate is simply the product of the probabilities of the input events.

3.1.3 Analysis

Most FTA procedures follow a two-step process. The first is the “qualitative” process of determining a tree’s minimal cut-sets. A cut-set is defined as “any event at the basic level (or a combination of such events) which will result in the occurrence of the top event.” Each cut-set, in effect, determines a path (possibly forked) from the leaves to the root of the fault tree.

To analyze the tree, we determine a finite set of minimal cut-sets, where a minimal cut-set represents the shortest path between the basic event and the top event. The complete set of minimal cut-sets gives all possible event outcomes for which the top-event can occur.

Once we have the minimal cut-sets, all that remains is to calculate the probability of each cut-set occurring and aggregate these probabilities to deduce the probability of the top event. To do so, we exploit our independence assumption as follows (Let $R$ be a system’s reliability, and $F = 1 - R$ be its probability of failure):

- To find the probability of each minimal cut-set, we calculate the product of the probability of each element in the cut-set. This works because each cut-set is just the intersection of a group of basic events. We have $F_{\text{cut-set}} = F_1 F_2 \ldots F_N$, where $N$ is the number of basic events in the cut-set.

- To find the target event’s failure probability, we exploit the fact that the event of any failure occurring is just the complement of the event that no failures occur. The probability that no failures occur among the cut-sets is just the product of the reliabilities of the cut-sets. Thus, $F_{\text{top-level}} = 1 - (1-F_{\text{cut-set1}}) \ldots (1-F_{\text{cut-setM}})$, where $M$ is the size of the set of minimal cut-sets.
Passenger Injury Occurs

- Box Ice Fall
- Holding Brake Failure
- Cable off Policy
- Controller failure
- Lock failure
- Door close failure
- Door not at level

- No Holding Brake
- Motor Turns Free

- worn pulley
e13

- worn brake
  e14

- control unit
discovers brake
  e15

- no power to motor
  e16

- motor failure
  e17

- hardware failure
  e18

- control unit
  disconnects power
  e19

- loss of power
  e20

- software failure
  e21

- worn pulley
  e26

- worn cable
  e27

- cable slips
  e28

- controller failure
  e29

- worn pulley
  e30

- worn cable
  e31
3.1.4 Sample Problem: Elevator Safety

Consider the fault tree of Figure 3.1 which describes the fault behavior of a simple electronically controlled passenger elevator. The elevator consists of a passenger box suspended by a cable hanging on a pulley in a shaft. At the opposite end of the cable is a drive unit that consists of an electric motor and a braking device. The passenger box is counterweighted to ease the load on the drive box.

We break the system into a control assembly and a drive assembly. The control system contains a microprocessor that determines where to move the passenger box based on user requests and latches in the shaft that tell it the box location. The drive unit is a bi-directional motor that moves the passenger box in response to commands from the control unit. The drive unit also includes a brake which holds the box stationary when no power is applied and allows free movement otherwise.

The only two high-level system events that can lead to serious injury are that the box free falls while a passenger is present or that the external door on a floor opens while the passenger box is not present. These two events form the left and right branches of the fault tree, respectively.

Exploring further the “box free falls” branch we see that cable breakage, slippage off the pulley, and the combination of a brake failure with a free-turning motor can lead to free fall. Cable properties are summed up in single reliability estimates (possibly derived from further fault analysis). Brake failures, however, can be caused by combinations of worn friction material, solenoid failure, motor failure, and controller hardware and software problems.

The “door opens erroneously” failure can be caused by latch failure, controller error (software or hardware), or wear on the cable or pulley.

Performing the fault tree analysis outlined above yields the minimal cut-sets and corresponding probabilities of causing passenger injury as shown in Table 3.1.
The overall reliability of the elevator, the product of the cut-set reliabilities, is 0.8107 in 10,000 hours. Thus, the probability of passenger injury is 0.1892 under the assumption that all basic events are independent.

3.1.5 Pitfalls

If basic event independence is a bad assumption, the FTA procedure can lead to drastically wrong top-level inference. Consider the case where one unobservable event causes the failure of several basic components of the system. This sort of “common cause” failure may result from environmental intervention, human operator error, and power failures (among others) and will likely increase the probability of target event failure. In our elevator example, for instance, a series of high-humidity days could lead to wear of both the cable and the brake’s friction material. For this reason, we cannot view the two failure events as independent.

<table>
<thead>
<tr>
<th>Cut-set</th>
<th>Probability of Failure in 10,000 hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cable off drive pulley</td>
<td>0.0951</td>
</tr>
<tr>
<td>Cable broken</td>
<td>0.0951</td>
</tr>
<tr>
<td>Worn friction material, Motor failure</td>
<td>0.009351</td>
</tr>
<tr>
<td>Worn friction material, Loss of system power</td>
<td>0.0003097</td>
</tr>
<tr>
<td>Stuck solenoid, Motor failure</td>
<td>0.0000221</td>
</tr>
<tr>
<td>Stuck solenoid, Loss of system power</td>
<td>0.0000007</td>
</tr>
<tr>
<td>Control unit disengages brake, Motor failure</td>
<td>0.00000000074</td>
</tr>
<tr>
<td>Control unit disengages brake, Loss of system power</td>
<td>0.000000000245</td>
</tr>
<tr>
<td>Door latch fails open, Worn drive pulley</td>
<td>0.0000192</td>
</tr>
<tr>
<td>Door latch fails open, Worn cable</td>
<td>0.0002378</td>
</tr>
<tr>
<td>Control unit hardware failure</td>
<td>0.00000003</td>
</tr>
<tr>
<td>Control unit software failure</td>
<td>0.00000002</td>
</tr>
</tbody>
</table>

Table 3.1: Minimal cut-sets and probabilities for the elevator problem.
3.1.6 Further References

For a thorough discussion of Fault Trees and their applications to safety analysis and system design see the Fault Tree Analysis Application Guide published by the Reliability Analysis Center [2]. The most widely used software package for fault tree analysis is SAPHIRE (Systems Analysis Programs for Hands-On Integrated Reliability Evaluations, see http://saphire.inel.gov) distributed by the Idaho National Engineering and Environmental Laboratory in Idaho Falls, Idaho.

3.2 Bayesian Inference Networks

A Bayesian network (also called a belief network) is, as stated by David Heckerman “a graphical model that encodes probabilistic relationships among variables of interest.” The network enables us to represent the abstract concept of causality (more appropriately “relevance”) by directed edges between nodes representing events in a probabilistic system. With an assessment of our “degree of belief” in the likelihood for each of the node events, we can infer, using Bayesian reasoning, information about other nodes in the system.

Bayesian networks are ideal for seamlessly integrating observed data with prior conviction, even in the face of missing data. Over the last several years, sophisticated algorithms for learning causality from data have been developed though the application of Artificial Intelligence techniques to Bayesian nets [12]. Here we focus on Bayesian methods for probabilistic inference.

3.2.1 Formal Definition

A Bayesian network, as defined by Heckerman, is a set of variables $X = \{X_1, \ldots, X_n\}$ consisting of (1) a network structure $S$ (a directed acyclic graph) that encodes a set of conditional independence assertions about variables in $X$, and (2) a set $P$ of local probability distributions associated with each variable. Together these components define the joint probability distribution for $X$. The nodes are in one-to-one correspondence with the variables in $X$. In our case, we only consider systems where the node variables $X$ are binary (or $k$-nary for some integer $k$) events.
Edges in $S$ represent conditional dependence, and conversely, the lack of an edge in $S$ represents *conditional independence*. Thus, the probability of a node $X_i$ is wholly determined by the probability of its parents $\text{pa}_i$ and the joint probability distribution for a particular $x \in X$ is just the product of $P(x_i|\text{pa}_i)$ for all nodes $i$.

### 3.2.2 Reasoning Bayesian Style

Bayesian statistics differs from classical statistics in that probabilities denote “the measure of plausibility” of an event, rather than just the frequency of that event occurring. This is a subtle difference, most evident when attempting to describe an one-time-only event, for instance the 1997 World Series. To estimate the probability of the Florida Marlins winning four games out of seven, a frequentist can only speak after games have been played, when enough data has been gathered to state a statistical opinion. A Bayesian, on the other hand, can incorporate the prior opinions of an expert on the subject with the data from early games to infer a new “plausibility” for the Marlins winning.

Bayesian reasoning is called Bayesian because it exploits Bayes’ rule that for events $H$ and $D$, $P(H|D) = P(D|H)P(H)/P(D)$. If we view $H$ as a hypothetical event and $D$ as the past data for the event, we see that all we need to infer $H$ from $D$ is a model for $H$ from which we can determine the likelihood of the observed data $P(D|H)$ and a marginal *prior distribution* $P(H)$ of the probability of the hypothesis. The marginal *posterior* distribution $P(D)$ is the sum (integral) of $P(D|H)P(H)$ for all possible discrete (continuous) $H$.

Note that we can incorporate all other information $I$ by conditioning all quantities by $I$ as:

$$P(H|D,I) = P(D|H,I)P(H|I)/P(D|I)$$

To see this, expand $P(D,H,I)$, $P(H|I)$, and $P(D|I)$ to get:

$$P(H|D,I) = [P(H,D,I)/P(H,I)] [P(H,I)/P(I)] / [P(D,I)/P(I)]$$

$$= [P(D,H,I)]/ [P(D,I)]$$

16
Therefore, if \( I \) takes the form of more data, we can view the system as if all the data was taken together by simply merging \( D \) and \( I \) into the updated data event \( D' = (D \cup I) \).

### 3.2.3 Sample Problem: Medical Diagnosis

It is easiest to see the applicability of Bayesian network techniques through an example. Consider the following problem of medical diagnosis from symptoms as presented by Carlos A. de B. Pereira [3]:

In the search for a new indicant of a disease \( D \), doctors in a certain clinic selected \( m \) patients known to have the disease and \( n \) patients known not to have the disease. Here \( D^+ \) is the event that a patient has the disease, while \( D^- \) is the event that a patient does not have the disease. To each patient they applied a test obtaining a response \( E^+ \) for positive evidence or \( E^- \) for negative evidence, finding \( x \) of the \( m \) diseased patients \( E^+ \) and \( y \) of the \( n \) non-diseased patients \( E^- \).

A new patient comes to the clinic and is judged by the doctors, prior to taking the test, to have the disease with probability 0.1. The question now is, given data for \( x, n, y, \) and \( m \) along with the new patient’s test result, what can we infer about the probability the new patient has the disease?

Let us introduce some notation and solve the problem using the Bayesian approach. Let \( \delta \) be the state of the new patient, 1 if the patient is \( D^+ \), 0 otherwise. Similarly, let \( t \) be the result of the patient’s test, 1 if \( E^+ \), 0 if \( E^- \). To complete the picture, let the *true* sensitivity of the test, \( P(E^+ | D^+) \), be \( \pi \) and the *true* specificity of the test, \( P(E^- | D^-) \), be \( \theta \). Assuming that this is a new test for which we have no other information, we assume a priori that \( \pi \) and \( \theta \) are selected from the uniform \((0,1)\) distribution. As a result, all of our knowledge about the test comes from our sampling data \( x \) and \( y \).

The Bayesian network in Figure 2 illustrates the diagnosis process. We begin with edges representing causal conditional dependencies between parameter states and a marginal distribution for each parent-less node. Each distribution represents our entire state of knowledge about that node’s parameter prior to observing data.
By exploiting Bayes' rule we can reverse any edge we choose. We find P(A|B) where we previously had P(B|A) and P(A). By repeated “reversals” we can transform the graph from one of influence to inference, switching the direction of all arrows (conditional dependencies) that leave from a node we care about to leading to that node. Along the way, we can eliminate all unobservable nodes from the graph by finding the implied distributions at all observable nodes affected by them.

Now we can analyze the Bayesian network for a particular medical diagnosis problem. Say that we take sample sizes of 150 patients (n=m=150) and find 60 diseased patients exhibiting symptom E (x=60) and only 9 non-diseased patients exhibiting symptom E (y=141). Again, we assess the marginal probability of a new patient having the disease to be 0.1. Then, we find using Bayesian analysis that:

- If the test is positive (t=1), the probability that the new patient is D⁺ is 0.40.
- If the test is negative (t=0), the probability that the new patient is D⁻ is 0.93.

Thus, the data has large impact on our opinion if the test is positive, changing from 0.1 a priori to 0.4 a posteriori, and little if the test is negative, 0.90 a priori to 0.93 a posteriori.
When the next patient comes into the office, we will be able to incorporate this result into his or her analysis.

3.2.4 Further References

For an excellent discussion of Bayesian Networks, especially in regard to their ability learn relationships from data, see David Heckerman’s “Tutorial on Learning with Bayesian Networks” distributed by Microsoft’s Advanced Technology Division [12]. For a more detailed presentation of the medical diagnosis problem see “Influence Diagrams and Medical Diagnosis” by Carlos A. de B. Pereira [3].
Chapter 4

4 The RIK Algorithm

The crux of the Fundamental Theorem of Probability (and thus the RIK algorithm) is the observation that once we fully enumerate the sample space of a discrete probabilistic system, we can evaluate the probability of arbitrarily complex events within the system. With a finite set of outcome states, the probability of an arbitrary binary event occurring is just the sum of the outcome probabilities for those outcomes in which the event occurs.

4.1 Simple Example

Consider a geologist surveying three sites for an oil company. He determines the probability of finding oil at sites A, B, and C to be 50%, 80%, and 20% respectively. We would like to know the probability of finding oil at at least one site.

We can enumerate all of the possibilities as shown in Table 4.1.

<table>
<thead>
<tr>
<th>Event</th>
<th>Outcome e₁</th>
<th>Outcome e₂</th>
<th>Outcome e₃</th>
<th>Outcome e₄</th>
<th>Outcome e₅</th>
<th>Outcome e₆</th>
<th>Outcome e₇</th>
<th>Outcome e₈</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oil at A?</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Oil at B?</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Oil at C?</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Any Oil?</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.1: Realm matrix for geology problem

If we can find the probability of each possible outcome eₖ for k ∈ {1,...,8}, we can easily determine the probability of our event by summing over the outcome probabilities in which our event occurs. In this example, the probability of finding oil is just the sum of
the probabilities of $e_1, e_2, e_3, e_4, e_5, e_6,$ and $e_7$. If we can assess the outcome probabilities then we can examine the probability of any event.

Notice that if we define a $4 \times 8$ matrix $R_{oil}$ whose columns are those of the matrix in Table 4.1, a vector $x = \{x_1, x_2, x_3, x_4\}$ where $x_1$ is the event of finding oil at A, $x_2$ is the event of finding oil at B, $x_3$ is the event of finding oil at C, and $x_4$ is the event of finding any oil at all, then $x$ is representable as:

$$x = R_{oil} e,$$

with $e = \{e_1, \ldots, e_8\}^T$.

This algebraic representation of the logical relation between $x$ and $e$ has a probabilistic counterpart.

### 4.2 Notation and Formal Problem Statement

Let us now introduce some notation and state this relationship formally. Let $x_N$ be a vector of binary events. Let $R_N$, the realm matrix of $x_N$, be a matrix with a row for every event in $x_N$ and a column for every possible outcome $e_j$ in the sample space. Let $q = [P(e_1), P(e_2), \ldots]^T$ be the vector of outcome probabilities. Then:

\begin{equation}
\text{(4.1) Fundamental Relationship #1}
\begin{align*}
P_N &= P(x_N) = R_N q \\
qu^T 1 &= 1 \\
q &\geq 0
\end{align*}
\end{equation}

This follows from the Law of Total Probability and from the fact that the outcomes in the sample space are disjoint and exhaustive.

### 4.3 The Fundamental Theorem of Probability

We see from Relationship (4.1) that if we know $q$, the outcome probability vector, then we can determine the marginal probabilities of the events. Recall the problem we would like to solve:

Given a binary event $x_{N+1}$, a set of logically or conditionally related events $x = \{x_1, \ldots, x_N\}$, and assessments $P_N$ for the probabilities of the events in $x$, infer the bounds on the probability that $x_{N+1}$ occurs such that the system is probabilistically coherent.
So, to solve for the extreme probabilities of the target \( x_{N+1} \), we can vary \( q \) until we find the feasible values that minimize and maximize \( P(x_{N+1}) \), exploiting Relationship (1.1) as follows:

I. Figure out how event \( x_{N+1} \) behaves for each outcome in \( R_N \), forming another row of the realm matrix \( r_{N+1} \). This requires an understanding of the logical and conditional relationships between \( x_{N+1} \) and the components of \( x \).

II. Find the values of \( q \) such that they:
- are consistent with the information given in \( P_N \)
- form a probabilistically coherent system (\( q^T 1 = 1 \) and \( q \geq 0 \))
- maximize/minimize \( (r_{N+1})q \)

This is the approach taken by Lad, Dickey and Rahman in their reformulation of De Finetti’s Fundamental Theorem of Probability [13]:

(4.2) *The Fundamental Theorem of Probability (FTP) and the MLP*:

Let \( x_N \) be a vector of events, \( p_N \) be a vector of probability assessments for \( x_N \), and \( x_{N+1} \) be any other event. The event vector \( x_{N+1} = [x_1, x_2, \ldots, x_N, x_{N+1}]^T \) generates a sample space \( \Omega \) of size \( \eta(N+1) \leq 2^{N+1} \). Let \( E_{N+1} \) be a vector of atomic events of \( \Omega \). Then there is a \((N+1) \times \eta(N+1)\) matrix \( R_{N+1} \) such that \( x_{N+1} = R_{N+1} E_{N+1} \). Denote the first \( N \) rows of \( R_{N+1} \) by \( R_N \) and the \((N+1)\)st row by \( r_{N+1} \). Then:

(a) An extended vector of probability assessments, \( p_{N+1} = [p_N, p_{N+1}]^T \), is coherent if and only if \( l \leq p_{N+1} \leq u \), where \( l \) and \( u \) are determined by the master linear programming (MLP) problems:

\[
\begin{align*}
    l &= \min_r r_{N+1} q \\
    \text{subject to: } &R_N q = p_N \\
    &q^T 1 = 1
\end{align*}
\]

\[
\begin{align*}
    u &= \max_r r_{N+1} q \\
    \text{subject to: } &R_N q = p_N \\
    &q^T 1 = 1
\end{align*}
\]

\[q \geq 0\]

(b) The probability vector \( p_N \) is coherent if and only if the feasible region for the MLP problems is non-empty.

Thus, we can find \( q^* \), the value of \( q \) that minimizes (or maximizes) \( P(x_{N+1}) \), by solving a linear program [14] with constraint matrix \( R_N \). Unfortunately, for problems with \( N \) sub-events, the size of the realm matrix is \( O(N) \times O(2^N) \). The exponential size of the realm matrix makes it intractable, even for moderate \( N \), for us to enumerate the entire sample.
space or even store $R_N$ within a computer program. We must find a way to reduce the size of the problem.

4.4 Overview of the RIK Algorithm

The RIK algorithm uses a clever integer programming approach to reduce the computation time and storage space required to solve the MLP. A brief overview of the algorithm follows. For a more detailed description see [17].

4.4.1 LP Notation

Let us examine the standard approach to solving linear programs in order to motivate the changes we can make to solve the MLP in reasonable time and space. The standard LP notation is:

$$\min/\max_q c^Tq$$

Subject to: $Aq = b \quad \text{dimension}(A) = n \times m$

$$q \geq 0$$

Translating the MLP into this notation gives:

$$c^T = r_{N+1} \quad A = \begin{bmatrix} R_N \\ 1 \end{bmatrix} \quad b = \begin{bmatrix} p_N \\ 1 \end{bmatrix} \quad q = q \quad n = N+1, m \equiv 2^{N+1}$$

where $[1]$ is a row vector of all ones $[1,1,\ldots,1]$ and $[0]$ a column vector of all zeros.

4.4.2 The Related Linear Program (RLP)

An important fact about linear programs is that feasible solutions utilize just a favored subset of the columns of $A$, a set of $n$ columns called the basis. For a particular feasible solution, all non-basic columns play no part in the solution. The Simplex Method allows us to solve LPs by simply iterating through feasible solutions until we find an optimal one. Thus, if we replace the constraint matrix $A$ by any $A'$ such that the columns of the optimal basis are a subset of $A'$, then the LP with $A'$ constraint matrix will have the same optimal objective value.
So, rather than attempting to store the entire Realm Matrix when solving the MLP, we can just store a small subset of outcomes, $R_{N+1}'$, called the column cache, and use it as our constraint matrix. The MLP is thus transformed into:

\[(4.3) \quad \text{The Related Linear Programs (RLP)}\]

\[
\begin{align*}
l &= \min_r r_{N+1}'q \\
\text{subject to:} & \quad R_N^Tq = p_N \\
q^T1 &= 1 \\
q &\ge 0
\end{align*}
\[
\begin{align*}
u &= \max_r r_{N+1}'q \\
\text{subject to:} & \quad R_N^Tq = p_N \\
q^T1 &= 1 \\
q &\ge 0
\end{align*}
\]

As long as the column cache contains the optimal basis columns at the end of our algorithm, we will have the same extrema $l$ and $u$. The caching analogy is a useful one and will be explored in detail later in this thesis.

4.4.3 The Revised Simplex Algorithm

The Revised Simplex algorithm is a time and space-efficient algorithm for solving linear programs [14]. Without loss of generality suppose we solve a minimization problem.

\[(4.4) \quad \text{The Revised Simplex Algorithm}\]

Let $B$ be a $n \times n$ sub-matrix consisting of columns of $A$. Call the $n$ corresponding column variables (in our case outcome probabilities) basic variables and call the remaining $m-n$ column variables non-basic variables. Call $B$ the basis. Let $c_B^T$ be the basic components of the cost vector $c$. Setting the non-basic elements of $q$ to zero determines a basic feasible solution if now $q \ge 0$. Our goal is to iterate through basic feasible solutions until we find the optimal one. We iterate by changing one column of $B$ at a time as follows:

1. Given $B$, calculate the inverse $B^{-1}$ and the vector of dual prices $\pi^T = c_B^T B^{-1}$
2. For each non-basic column $j$, calculate the reduced cost $r_j = c_j - \pi^T a^{(j)}$, where $a^{(j)}$ is the $j^{th}$ column of $A$ and $c_j$ the $j^{th}$ entry of $c$.
3. If $r_j \ge 0$ for all $j$, stop – the basic feasible solution corresponding to $B$ is optimal. Otherwise, select $j'$ corresponding to the smallest $r_j$.
4. Swap column $a^{(j')}$ into $B$, swapping out another column to create the new feasible basis $B'$. Adjust $c$ and $b$ accordingly. Go to step (1) using $B'$, $c'$, and $b'$ in place of $B$, $c$, and $b$.

The good news is that the revised simplex algorithm does not require us to store the entire constraint matrix, rather just a subset that includes the basis, which is of size $n \times n$. The
problem is that step (3) requires us to compute the reduced cost for all of the non-basic columns in $A$. For the MLP, this is $O(2^n)$ columns, obviously intractable. The solution here is to employ a simple, elegant, column generation approach. Rather than examine each non-basic column individually, we solve a related integer program (RIP) to generate the column with the minimum reduced cost. If this generated column has non-negative reduced cost then we have an optimal solution. Otherwise, we swap the generated column into the basis.

4.4.4 A Compact Representation of the Outcomes and The Related Integer Program (RIP)

For the column generation technique to work we need a compact way to represent the columns of $A$, the realm matrix for our event vector $x$. By understanding the logical structure of our events we can represent the realm matrix via a set of linear inequalities and integer constraints:

\begin{equation}
\text{(4.5) Realm Matrix Outcomes as a Solution to a Set of Linear Inequalities}
\end{equation}

If the event vector $x_N$ is such that for every $x_i \in x_N$, $x_i$ is either a basic event (independent of other events), a conjunction of other events, a disjunction of other events, a negation of another event, conditional upon other events, a linear combination of other events, or a normalization event ($x_i = 1$), then there exists some pair ($M$, $g$) such that an arbitrary vector $z$ is a column of $R_N$ (and thus $A$) if and only if $z \in \{ z | Mz \leq g, z_i \in D_j \}$. Here $D_j$ is defined to be the domain of possible values for the $j$th element of a generic $z$. That is, $z_j \in D_j, j=1,\ldots,N$ if and only if $z$ is a column vector of the realm matrix. $M$ has no more than $4(N+1)$ rows and $(N+1)$ columns.

$M$ and $g$ can be determined by a simple parsing through the logic of $x$. (Myers, 1995).

As an example, consider $x = (x_1, x_2, x_3)^T$ where $x_3 = x_1 \land x_2$, a simple conjunction. Only columns $z = (z_1, z_2, z_3)^T$ in $R_{N+1}$ satisfy the following set of linear constraints and every satisfying $z$ must be in $R_{N+1}$. Here, $D_j = \{0,1\}$ for all $j=1,\ldots,N$.

\begin{align*}
  z_1 + z_2 - 2z_3 &\geq 0 \\
  z_1 + z_2 - z_3 &\leq 1 \\
  z_i &\in \{0,1\} \text{ for all } i
\end{align*}

$\Leftrightarrow$ \hspace{1cm} $z \in \{(0,0,0), (0,1,0), (1,0,0), (1,1,1)\}$
4.4.5 The Related Integer Program (RIP)
We would like to find the column $z_N^*$ that has the minimum reduced cost. If the reduced cost is a linear function of $z$, then we can solve an integer program to find $z_N^*$. The reduced cost for the $j$th column of $A$ is $c_j - \pi^T a^{(j)}$, or in FTP notation, $r_{N+1j} - \pi^T r_{N}^{(j)}$. If we simply include $x_{N+1}$ in our generation of $M$ and $g$ so that $r_{N+1}$ is included in the represented realm matrix then we see that:

$$\text{Reduced cost of column } z = z_{N+1} - (\pi^T,0)^T z_N$$

Thus by solving the following integer program, using the well-established Branch-and-Bound algorithm, we can find the non-basic column of minimum reduced cost:

\begin{equation}
(4.6) \text{The Related Integer Program (RIP)}
\end{equation}

Let $z_{N+1}$ be the solution to the integer program:

\[
\begin{align*}
\min v &= z_{N+1} - (\pi^T,0)^T z_N \\
\text{subject to: } Mz &\leq g \\
& \quad z_j \in D_j, \quad j=1,2,\ldots,N
\end{align*}
\]

Then $v$ is the minimum reduced cost of the non-basic columns of $R_N$, $z_N^* = (z_{N+1} \setminus z_{N+1})$ is the optimal non-basic column, and $z_{N+1}$ is the corresponding component of $r_{N+1}$ in the RLP.

So by solving an integer program of size $O(N)$ we can, in turn, plug the results into the Revised Simplex Algorithm and solve the MLP in tractable time.

4.4.6 Flow of the RIK Algorithm

The RIK algorithm follows the method of the Revised Simplex Algorithm with a few subtleties. Rather than keep just the basis $B$ around, we instead keep the column cache $R_{N+1}'$. The reasoning behind this supports the caching analogy. Because the Simplex Algorithm iterates through potential solutions (column sets) in an unpredictable fashion, it is possible that a previously basic column that has turned non-basic may at some point re-enter the basis. Rather than spending the time and expense to re-generate the column by solving the RIP, we can keep it in the cache enabling the RLP to “turn” it basic at any iteration. The tradeoff is that with a larger cache the RLP takes slightly longer to solve. Negotiating this tradeoff is a source for optimization of the algorithm by the user.
The flow of the algorithm is as diagrammed in Figure 4.1. We initialize the column cache to a set of N artificial columns with extreme high costs so that they are soon removed from the cache. This column set forms the initial basis, and must be of basic form [14].

We then alternate between solutions of the RLP and RIP until we achieve an optimal solution. We solve the RLP to find the basis among the columns in our current cache and the corresponding lower bound \( l \). Using the dual from the RLP in the objective function of the RIP we solve to find the best column of \( \mathbf{R}_{N+1} \) not currently in our basis or cache. If this column has non-negative reduced costs, then all non-basic columns must have non-
negative reduced costs, and by the revised simplex algorithm \( l \) is the optimal solution to the MLP. If the generated column has negative reduced costs, its introduction into the basis will result in a lower \( l \), so we put the column into the cache, swapping out the oldest non-basic column, and continue to iterate.

4.4.7 Extensions to the FTP

Two crucial extensions to the Fundamental Theorem of Probability involve the description and probabilistic analysis of conditional quantities. By reformulating the RLP and RIP appropriately we can handle both conditional assessments where we have a bound estimate for the probability of one event conditional upon another and conditional targets, where we bound the probability of a conditional event.

4.4.7.1 Conditional Assessments

Suppose we have the conditional assessment \( P(x|y) = \phi \). By the product rule for coherent prevision we may write \( P(x|y) = P(xy)/P(y) = \phi \) [4]. Because \( P(xy) = r_{xy}q \) and \( P(y) = r_yq \) in our framework:

\[
P(x|y) = \frac{r_{xy}q}{r_yq} = \phi \quad \Rightarrow \quad [r_{xy} - \phi r_y]q = 0
\]

So by appending the row \( r(x|y) = r_{xy} - \phi r_y \) to the realm matrix with \( p(x|y) = 0 \), we can cohere with our assessments of conditional quantities. We may use assessed bounds of the form \( P(x|y) \geq \phi \) and \( P(x|y) \leq \phi \) by changing the sense of the RLP constraint \( r(x|y)q = 0 \) to \( r(x|y)q \geq 0 \) and \( r(x|y)q \leq 0 \) respectively. Note, however, that to deal with an assessment of the form \( \phi_1 \leq P(x|y) \leq \phi_2 \) we must create two separate realm matrix rows for \( \phi_1 \leq P(x|y) \) and \( P(x|y) \leq \phi_2 \) individually.

In addition, each conditional assessment requires the addition of an auxiliary continuous variable in the RIP constraint matrix \( M \), as described later. Unfortunately, abstract independence assessments of the form \( P(\text{a|b}) = P(\text{a}) \) require more complex expressions and mathematical programming techniques [11].
4.4.7.2 Conditional Target Variables

Suppose we wish to bound the probability of the conditional quantity \( P(x|y) \). We may again exploit Bayes rule to rewrite \( P(x|y) \) as \( P(xy)/P(y) \) and solve the *Linear Fractional Program*:

\[
\begin{align*}
\text{(4.6) Linear Fractional Program (LFP) Formulation of Target Conditional} \\
& l = \min (r_{xy}/r_{y})'q \\
& \text{subject to: } R_N'q = P_N \\
& q^T1 = 1 \\
& q \geq 0 \\
& u = \max (r_{xy}/r_{y})'q \\
& \text{subject to: } R_N'q = P_N \\
& q^T1 = 1 \\
& q \geq 0
\end{align*}
\]

Fortunately, LFP’s can be transformed into linear programs with the introduction of one more row of constraints. For details see [4].
Chapter 5

5 Implementation

My primary goal for this research project was to create a clean, efficient, object-oriented system that could solve arbitrary EPI problems as specified by the user. To do this, I constructed a C++ program that interfaced with CPLEX, a commercial optimization software package. The result is a fast implementation that is sufficiently modular to allow for incremental future improvement.

5.1 General Architecture

I used a limited object-oriented structure to cleanly separate the different aspects of the algorithm in order to emulate the flow diagram of Figure 4.1. Each object contains data structures and the relevant procedural calls that can be enacted on the data. The implementation uses three types of data object types, RLPdata, RIPdata, and FTPdata which are explained below and in Figure 5.1.

5.1.1 RLPdata

An RLPdata object contains all the data required to solve an iteration of the Related Linear Program. It stores the cache of columns ($R_N'$), the objective function (row $r_{N+1}'$ of the partial realm matrix, also called the “cost”), the right hand side vector ($p_N$ or $b$), and a vector age, which is the number of iterations the column has been in the cache. It is worth noting that the cache of columns is stored in CPLEX’s preferred matrix notation (see Appendix A) which allows CPLEX to traverse columns much faster at the expense of slow row traversal.

An RLPdata object has four associated procedures. Insert_column_into_cache takes a new column and cost coefficient as arguments and adds the new column to the cache, aging the cache by one when done. If the cache is already full, it removes the oldest
**FTPdata**

Data:
- **rlp** (of type RLPdata)
- **rip** (of type RIPdata)
- CPLEX environment information
- **problem parameters**: target variable, bound assessments, etc.

Important Procedures:
- **readfromfile**: Read problem file for problem parameters and parse logic to create M, g
- **doPhaseI**: Initiate column cache in rip
- **doPhaseII**: Run the RIK algorithm iteration by iteration until optimality.
- **createprobs**: Create the RIP and RLP in CPLEX
- **closeprobs**: Close CPLEX upon completion

---

**RLPdata**

Data:
- **CACHE**: the constraint matrix (in CPLEX format)
- **b**: the right hand side
- **c**: the objective function
- **age**: number of iterations since generation for each column

Important Procedures:
- **insert_column_into_CACHE**: inserts new column into CACHE removing oldest one if necessary
- **createRLP**: creates RLP problem in CPLEX
- **doRLP**: solves RLP in cplex and generates dual price vector $\pi$

---

**RIPdata**

Data:
- **M**: the logical constraint matrix (in CPLEX format)
- **g**: the right hand side
- **ripobj**: the objective function
- **sense**: {<, =, >} for each constraint

Important Procedures:
- **put_M_in_matrix_form**: converts 2D M into CPLEX matrix format
- **createRIP**: creates RIP problem in CPLEX
- **doRIP**: solves RIP in CPLEX and generates new column $z^*$ and $v^*$
- **put_ripobj**: Updates RIP objective function with $\pi$

---

*Figure 5.1: Data Objects. Main object "ftp" is of type FTPdata*
non-basic column to make room. \textit{CreateRLP} initializes the linear program in CPLEX, and \textit{doRLP} solves the problem in CPLEX and returns the calculated best-to-date bound (the optimal objective value of the RLP) and the associated dual prices for passage to the RIP.

\subsection*{5.1.2 RIPdata}
An RIPdata object is similar to an RLPdata object in that it stores all the relevant data for the Related Integer Program. Specifically, it stores \( M \) (the logic constraint matrix), \( g \) (the right hand side), \textit{sense} (the sense \( \{=,\geq, \leq \} \) of each row) and the objective function. \textit{createRIP} initializes the RIP in CPLEX. At every iteration, \textit{put_ripobj} updates the objective function of the RIP with the dual prices of the RLP and \textit{doRIP} solves the RIP and generates the new column \( z^* \) with reduced cost \( v^* \). \textit{put_M_in_matrix_form} is a utility function that converts a standard two dimensional array version of the \( M \) matrix into the more efficient CPLEX matrix format [See Appendix A].

\subsection*{5.1.3 FTPdata}
An FTPdata object contains everything needed to solve a particular instance of the EPI problem. It reads the problem description from a file, opens CPLEX, initializes the RLP and RIP problems, and iterates between them until a solution is reached. The associated functions are described in more detail below.

\subsubsection*{5.1.3.1 Input File Format}
The top-level FTPdata object, \texttt{ftp}, is responsible for parsing the problem parameters and logic in from the input file to its data structures. The input file must be a text file of the following line-by-line format (top to bottom):

- Five ignored lines. May be used for comments describing the problem.
- “CONDTARGET” followed by a 1 if the target variable is a conditional (min/max \( P(x|y) \)), or a 0 otherwise.
- “ASSESSED” followed by the number of assessed variables. These are the basic variables for which we have pre-assessed bounds.
• “RELATED” followed by the number of related variables. These are variables defined by their logical or conditional relationship to basic or other related variables.

• “RELATIONSHIPS” followed by the number of relationships. These are expressions describing a relationship among/between variables without incurring the overhead of creating a new variable. A subtle difference from related that will be clarified later.

• The line “VarName VarNum LOW HIGH” which the parser ignores.

• One line per assessed variable in the format:
  • “<Variable Name> <Variable #> <Low Bound> <High Bound>”, where <variable #> increases one per line, starting with zero. This number is the unique identifier describing the variable. The variable name may be up to twenty letters and cannot include spaces or tabs. [e.g. “OilAtA 0 0.5 0.6”]

• One line per related variable in the format: (note that if we have no prior information about a variable or relationship set the bounds to (0,1))
  • “<Variable Name> <Variable #> <Low Bound> <High Bound> <Relation> <Related List>” Where relations can be of the following types:
    • AND – the union of the listed variables.
    • OR – the intersection of the listed variables.
    • NOT {or NEG} – the compliment of the (one) listed variable.
    • COND – the conditional quantity. The listed variables are in the order x, y, where we assess P(xly). Note that only unidirectional bounds are allowed. To assess \( \phi_1 \leq P(xly) \leq \phi_2 \), one must enter two COND variables: \( 0.0 \leq P(xly) \leq 1.0 \).
    • TCOND – the conditional quantity (xly) used as the target variable. Must be specified as the last related variable listed.
    • ATLEAST – true if at least k (first argument) of the listed variables are true. [e.g. “y 4 0.00 1.00 ATLEAST 2 0 1 2 3” => variable 4, named y, is true if at least 2 of variables 0, 1, 2, and 3 are true]
    • LINEAR – a linear combination of the listed variables, with associated coefficients and RHS given. LINEARG, LINEARE, and LINEARL denote ‘>‘, ‘=’, and ‘≤’ respectively. [e.g. “y 4 0.00 1.00 LINEARE 7.3 2.2 1 1.5 3” => (variable 4) + 2.2(variable 1) + 1.5(variable 3) = 7.3]

• One line per relationship in the format:
  • “<RELATIONSHIP> <Variable List>”, where relationships can be of the following types (no end-of-line comments allowed):
- MUTEX – the listed variables are mutually exclusive and exhaustive, in other words exactly one from the list occurs for every possible outcome.
- NAND – the listed variables do not all occur together.
- ATLEAST – at least k (the first argument) of the listed variables occur.
- NOMORETHAN – no more than k (the first argument) of the listed variables occur.
- LINEAR – Listed variables occur such that the (in)equality specified is true. As above LINEARG, LINEARE, and LINEARL denote ‘≥’, ‘=’, and ‘≤’ respectively. \{ e.g. “LINEARG 2.3 8.6 1 7.4 0” => 8.6(variable 1) + 7.4(variable 0) ≥ 2.3 \}
- “TARGET” followed by the number of the target variable.
- “SENSE” followed by “MIN” or “MAX” to minimize/maximize P(x_{target})
- “MAXCACHESIZE” followed by the desired limit on the cache size. After this number of columns in the cache, we begin replacing columns.
- “VERBOSE” followed by a one if a detailed output is desired, zero otherwise.

A typical input file is shown in Figure 5.2.
5.1.3.2 Constructing the RIP Constraints from the Input File

The FTPdata procedure `readfromfile` is responsible for parsing the input file to determine the problem parameters such as the target variable and cache size as well as constructing the RIP constraints. The RIP constraints are stored in the data object `rip`, which is of type RIPdata. To do this, `readfromfile` makes one pass through the input file, constructing a two-dimensional matrix which is then loaded into `rip` via `put_M_in_matrix_form`.

The `(M,g,sense)` trio is constructed according to Table 5.1 and Table 5.2. Recall that the goal is to describe all valid outcomes of the system in a set of linear constraints.

---

1 Table 5.1 is based on Definition 3.17 of Tracy Myers’ Doctoral Thesis [1]. Changes include a more efficient representation of conditional variables (from 4 RIP constraints to 3) and a reformulation of them to handle bound assessments. Extensions include the `ATLEAST` and `LINEAR` variables. Table 5.2 is new work.
<table>
<thead>
<tr>
<th>RELATED VARIABLE</th>
<th>LINEAR (IN)EQUALITIES in (M,g)</th>
</tr>
</thead>
</table>
| \( Y = \text{AND} (X_1 \ldots X_n) \) | \( X_1 + \ldots + X_n - nY \geq 0 \)  
  \( X_1 + \ldots + X_n - Y \leq n - 1 \) |
| \( Y = \text{OR} (X_1 \ldots X_n) \) | \( X_1 + \ldots + X_n - Y \geq 0 \)  
  \( X_1 + \ldots + X_n - nY \leq 0 \) |
| \( Y = \text{NOT} (X) \) | \( X + Y = 1 \) |
| \( Y = \text{COND} (XIZ), \ P(XIZ) = \phi \) | \( \omega - \phi Z - Y = 0 \)  
  \( \omega \in \{0,1\} \)  
  \( X + Z - 2\omega \geq 0 \) (auxiliary  
  \( X + Z - \omega \leq 1 \) AND variable)  
  (Note: RLP sense depends on \( P(XIZ) \) sense) |
| \( Y = \text{ATLEAST}(K, X_1 \ldots X_n) \) | \( \omega_1 + \omega_2 - Y \geq 0 \)  
  \( \omega_1, \omega_2 \in \{0,1\} \)  
  \( \omega_1 + \omega_2 - 2Y \leq 0 \)  
  \( X_1 + \ldots + X_n + (n+1) \omega_1 \geq n+k+1 \)  
  \( X_1 + \ldots + X_n + (k+1) \omega_2 \leq k-1 \) |
| \( Y = \text{LINEARG} (k, a_1, X_1, \ldots, a_n, X_n) \) | \( a_1X_1 + \ldots a_nX_n + Y \geq k \) |
| \( Y = \text{LINEARE} (k, a_1, X_1, \ldots, a_n, X_n) \) | \( a_1X_1 + \ldots a_nX_n + Y = k \) |
| \( Y = \text{LINEARL} (k, a_1, X_1, \ldots, a_n, X_n) \) | \( a_1X_1 + \ldots a_nX_n + Y \leq k \) |

**Table 5.1:** Related variable logic as linear inequalities
## Table 5.2: Logical relationships as linear inequalities.

<table>
<thead>
<tr>
<th>RELATIONSHIP</th>
<th>LINEAR (IN)EQUALITIES in ((M,g))</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUTEX ((X_1 \ldots X_n))</td>
<td>(X_1 + \ldots + X_n = 1)</td>
</tr>
<tr>
<td>NAND ((X_1 \ldots X_n))</td>
<td>(X_1 + \ldots + X_n \leq n - 1)</td>
</tr>
<tr>
<td>OR ((X_1 \ldots X_n))</td>
<td>(X_1 + \ldots + X_n \geq 1)</td>
</tr>
<tr>
<td>ATLEAST ((k, X_1 \ldots X_n))</td>
<td>(X_1 + \ldots + X_n \geq k)</td>
</tr>
<tr>
<td>ABSORB ((X_1, X_2))</td>
<td>(X_1 \geq X_2)  (for modeling Markov Chains)</td>
</tr>
<tr>
<td>NOMORETHAN ((k, X_1 \ldots X_n))</td>
<td>(X_1 + \ldots + X_n \leq k)</td>
</tr>
<tr>
<td>LINEARG ((k, a_1, X_1, \ldots, a_n, X_n))</td>
<td>(a_1 X_1 + \ldots + a_n X_n \geq k)</td>
</tr>
<tr>
<td>LINEARE ((k, a_1, X_1, \ldots, a_n, X_n))</td>
<td>(a_1 X_1 + \ldots + a_n X_n = k)</td>
</tr>
<tr>
<td>LINEARL ((k, a_1, X_1, \ldots, a_n, X_n))</td>
<td>(a_1 X_1 + \ldots + a_n X_n \leq k)</td>
</tr>
</tbody>
</table>

### 5.1.3.3 Phase I: Initializing the Cache

The FTPdata object ftp also contains the procedures for iterating the RIK algorithm until an optimal bound is found. Before running the iterations, we must initialize the column cache, rlp.cache to contain an artificial basis. This is called Phase I. The original cache is comprised of “artificial columns” forming the identity matrix, each with extreme cost; positive if the sense is minimization, negative for maximization. With a feasible system, these artificial columns will be replaced in the basis by valid columns with lower cost. In an infeasible system, they will remain.

### 5.1.3.4 Phase II: Iterating the RLP and RIP

FTPdata also contains the procedure doPhaseII for performing the iterations until the optimal bound is found. The process is as diagrammed in Figure 4.1. We solve the RLP using rlp.doRLP, recording the bound and the dual price vector \(p_i\), and allow CPLEX to designate the new basic columns in the cache so that they will not be overwritten during the next Insert_Column_into_Cache. We then construct rip.ripobj by copying negative \(p_i\) and inserting a 1 in the column corresponding to the target variable. Next, we solve the RIP using rip.doRIP, check for optimality by comparing \(v^*\) with zero, and either report the final exact bound or insert the new column \(z^*\) into the rlp.cache and repeat depending on the result.
As an implementation note, it is important to observe that the RIP uses full realm columns with the target included while the RLP uses only the non-target realm columns as constraints, using the target realm column as its objective function. Thus, after we solve the RIP using rip.doRIP we must be careful to isolate and remove the target component from \( z^* \) before inserting the column. Similarly, converting the pi vector into the RIP objective function must be done with care.

5.1.3.5 Target Conditionals

Transforming the Linear Fractional Program into a straightforward linear program requires a few extra considerations. First, we must introduce a new column (call it \( b_\theta \)) into the cache and make sure not to swap it out in Insert_Column_into_Cache [see Freund on LFP]. Secondly, we are forced to split those rows of the realm matrix with unequal bounds into two rows, one “less-than-or-equal-to” inequality and one “greater-than-or-equal-to”. (Normally we can use CPLEX’s support of “ranged rows”). This requires extra consideration when solving the RIP because the two rows of the realm matrix correspond to only one event in \( z \). To handle this, we must sum the two dual price vector components that correspond to one row of \( z \). Also we are careful to duplicate the necessary columns of \( z^* \) when inserting back into the cache.
Chapter 6

6 Applications

A primary goal of this research was to study real world problems to which the RIK algorithm could be applied. Fault tree analysis allows straightforward application of the RIK algorithm. [The independence assumptions adopted in FTA are worth questioning especially in the presence of common-cause failures.] The RIK’s ability to handle arbitrary logic and conditional relationships is beyond the capabilities of most fault tree software.

Belief networks fit naturally into the RIK framework, more for the RIK’s ability to perform inference given incomplete or partial information than for checking the impact of independence assumptions. The RIK algorithm enables us to relax conditional and marginal assessments much more easily than standard do Bayesian approaches. Because the RIK provides probabilistically coherent bounds, Bayes’ rule is always satisfied [4].

A secondary goal was to compare the RIK with other methods for calculating probabilities. A third goal is to improve the algorithm’s time performance. To accomplish these, we found two large problems. First, we acquired a large fault tree (about 1500 logic gates) describing the seismic and fire susceptibility of the Surry 50 nuclear plant [16] from Curtis Smith of the Idaho National Laboratory. Second, we designed a scalable system for analyzing a reconfigurable fault-detection system with transient and permanent errors based on analysis by Jon Sjogren [5].

Unfortunately, the algorithm, before further enhancement, cannot process problems of larger than 250 variables in less than three hours on a Sun Sparc 20 so the nuclear plant
fault tree is left out of this thesis. The reconfigurable fault detection system is presented here in detail but analyzed to less precision than desired.

A description of several applications of the RIK algorithm and comparisons with results from standard analytical techniques follows.

6.1 The Elevator Fault Tree

Analyzing the elevator passenger safety fault tree of section 3.1.4 with the RIK algorithm allows us to bound the probability of passenger injury occurring without assuming independence of the basic events.

6.1.1 Formulating the Fault Tree in RIK format

To set the fault tree problem up for the RIK algorithm to solve, we state the basic events with assessments for their marginal probabilities and also the logical relationships that define higher level events. The target for probabilistic bounding is the top-level event of the tree, the output of the AND gate “G1.” In this case we have point estimates for the fragility (the probability of failure in 10,000 hours) of each basic component. The input file specification for the problem is shown in Figure 6.1.

6.1.2 Solving the RIK Algorithm

Solving the RIK algorithm for the elevator fault tree yield bounds of [0.0951, 0.20585] for the probability of passenger injury in 10,000 hours of use. Recall that this probability was calculated as 0.1892 with the independence assumption. Thus, by removing the independence assumption, we admit the possibility of an increase of 1.665% in the probability of passenger injury. This is a significant increase and suggests that it is worthwhile for us to reevaluate the independence assumption by looking further for common cause failures. We can also use the RIK to analyze the top-level failure’s sensitivity to the individual component failure rates. If, for instance, we must guarantee by law that the probability of passenger injury in 10,000 hours is less than 0.20, then we can use the RIK to decide which combination of parts we should improve (or perhaps introduce redundancy for) to meet the safety mandate.
// Elevator Safety Problem: Fault tree w/o independence
//
// Ignore E2 and E3, artifacts of conversion from SAPHIRE
//
//
CONDTARGET 0
ASSESSED 16
RELATED 13
RELATIONSHIPS 0

<table>
<thead>
<tr>
<th>Var Name</th>
<th>Num</th>
<th>Low bound</th>
<th>Hi bound</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>E11</td>
<td>0</td>
<td>0.000250</td>
<td>0.000250</td>
<td>BASIC</td>
</tr>
<tr>
<td>E13</td>
<td>1</td>
<td>0.632200</td>
<td>0.632200</td>
<td>BASIC</td>
</tr>
<tr>
<td>E14</td>
<td>2</td>
<td>0.001500</td>
<td>0.001500</td>
<td>BASIC</td>
</tr>
<tr>
<td>E15</td>
<td>3</td>
<td>0.000000049</td>
<td>0.000000049</td>
<td>BASIC</td>
</tr>
<tr>
<td>E17</td>
<td>4</td>
<td>0.014900</td>
<td>0.014900</td>
<td>BASIC</td>
</tr>
<tr>
<td>E18</td>
<td>5</td>
<td>0.0005</td>
<td>0.0005</td>
<td>BASIC</td>
</tr>
<tr>
<td>E2</td>
<td>6</td>
<td>1.000000</td>
<td>1.000000</td>
<td>BASIC</td>
</tr>
<tr>
<td>E20</td>
<td>7</td>
<td>0.0000003</td>
<td>0.0000003</td>
<td>BASIC</td>
</tr>
<tr>
<td>E21</td>
<td>8</td>
<td>0.0000002</td>
<td>0.0000002</td>
<td>BASIC</td>
</tr>
<tr>
<td>E24</td>
<td>9</td>
<td>0.0000003</td>
<td>0.0000003</td>
<td>BASIC</td>
</tr>
<tr>
<td>E25</td>
<td>10</td>
<td>0.0000002</td>
<td>0.0000002</td>
<td>BASIC</td>
</tr>
<tr>
<td>E26</td>
<td>11</td>
<td>0.076900</td>
<td>0.076900</td>
<td>BASIC</td>
</tr>
<tr>
<td>E27</td>
<td>12</td>
<td>0.095100</td>
<td>0.095100</td>
<td>BASIC</td>
</tr>
<tr>
<td>E3</td>
<td>13</td>
<td>1.000000</td>
<td>1.000000</td>
<td>BASIC</td>
</tr>
<tr>
<td>E4</td>
<td>14</td>
<td>0.095100</td>
<td>0.095100</td>
<td>BASIC</td>
</tr>
<tr>
<td>E6</td>
<td>15</td>
<td>0.095100</td>
<td>0.095100</td>
<td>BASIC</td>
</tr>
<tr>
<td>G1</td>
<td>16</td>
<td>0.000000</td>
<td>1.000000 OR 17 18</td>
<td></td>
</tr>
<tr>
<td>G2</td>
<td>17</td>
<td>0.000000</td>
<td>1.000000 OR 19 14 15</td>
<td></td>
</tr>
<tr>
<td>G8</td>
<td>18</td>
<td>0.000000</td>
<td>1.000000 AND 24 25</td>
<td></td>
</tr>
<tr>
<td>G3</td>
<td>19</td>
<td>0.000000</td>
<td>1.000000 AND 20 21</td>
<td></td>
</tr>
<tr>
<td>G4</td>
<td>20</td>
<td>0.000000</td>
<td>1.000000 OR 1 2 3</td>
<td></td>
</tr>
<tr>
<td>G5</td>
<td>21</td>
<td>0.000000</td>
<td>1.000000 OR 22 4</td>
<td></td>
</tr>
<tr>
<td>G6</td>
<td>22</td>
<td>0.000000</td>
<td>1.000000 OR 23 5</td>
<td></td>
</tr>
<tr>
<td>G7</td>
<td>23</td>
<td>0.000000</td>
<td>1.000000 OR 7 8</td>
<td></td>
</tr>
<tr>
<td>G9</td>
<td>24</td>
<td>0.000000</td>
<td>1.000000 OR 28 0</td>
<td></td>
</tr>
<tr>
<td>G10</td>
<td>25</td>
<td>0.000000</td>
<td>1.000000 OR 26 27</td>
<td></td>
</tr>
<tr>
<td>G13</td>
<td>26</td>
<td>0.000000</td>
<td>1.000000 OR 27 28</td>
<td></td>
</tr>
<tr>
<td>G12</td>
<td>27</td>
<td>0.000000</td>
<td>1.000000 OR 11 12</td>
<td></td>
</tr>
<tr>
<td>G11</td>
<td>28</td>
<td>0.000000</td>
<td>1.000000 OR 9 10</td>
<td></td>
</tr>
</tbody>
</table>

TARGET 16
SENSE MIN
MAXCACHESIZE 75
VERBOSE 0

Figure 6.1: Input file for elevator fault tree.

6.1.3 General Fault Tree Analysis Conclusions

We see that the RIK is naturally adapted to fault trees and provides useful information, especially in finding worst-case scenarios. The RIK is more flexible than most FTA techniques not only because we can eliminate independence assumptions but also
because we can incorporate incomplete assessments, assessments on joint events, and additional types of logical and conditional gates.

Once the algorithm is enhanced to handle hundreds of events in reasonable time, we hope to be able to analyze the fault tree for the Surry-50 nuclear plant.

6.2 The Medical Diagnosis Problem

The medical diagnosis problem of section 3.2.3 fits nicely into the RIK framework. To illustrate the ease with which the RIK allows one to reason about such systems, let us consider a more complicated diagnosis problem where we do not have complete information. We shall see that, in such a case, a Bayesian approach leads to a closed-form but non-trivial optimization problem, whereas the RIK provides an exact solution.

6.2.1 Problem Statement

Recall that the medical diagnosis problem of section 3.2.3 was to infer the probability that a new patient has a disease $D$ given the result of a test $E$, sample data for the sensitivity and specificity of that test, and a doctor’s a priori assessment of the marginal probability that the patient is diseased. Consider the following reformulation of the problem:

Suppose the disease manifests itself in two forms: a malignant form $D^{++}$ and a benign form $D^-$. Let $D$ represent the absence of disease. The doctor assesses a priori that the probability the new patient is $D^{++}$ is between 0.1 and 0.2 and that the probability he is $D^+$ is between 0.2 and 0.3. The doctor has at his disposal tests $E^{++}$, $E^+$, and $E^-$, which test for malignancy, benignancy, and absence respectively. Due to budgetary constraints, the doctor can only collect sample data for four properties of these tests, $P(E^{++}|D^{++})$, $P(E^+|D^{++})$, $P(E^+|D^+)$, and $P(E^-|D^-)$, finding sample averages of 0.4, 0.5, 0.8, and 0.9 respectively. Suppose that we would like to bound the probability that the new patient has the malignant form of the disease given that he tests positive for it, $P(D^{++}|E^{++})$, and the probability that he has the benign form given he tests positive for the malignant form, $P(D^+|E^{++})$. 

42
6.2.2 Direct Analysis with a Probability Table

We can construct a probability table to solve the above diagnosis problem directly, using Bayes’ rule to infer \(P(D^{++}|E^{++})\) and \(P(D^+|E^{++})\). Let us first illustrate this technique with a simplified problem.

6.2.2.1 A Simpler Problem

Suppose now that instead of a priori bounded estimates for \(P(D^{++})\) and \(P(D^+)\), we have the exact marginal a priori probabilities \(P(D^{++}) = 0.1\) and \(P(D^+) = 0.2\). We can infer that \(P(D^-)\) is 0.7. Next, we can construct the probability table shown in Table 6.1 describing our state of information (given probabilities are in bold). The variables \(p\) and \(q\), where \(0.0 \leq p \leq 0.2\) and \(0.0 \leq q \leq 0.1\), represent the unknown probabilities resulting from our inability to sample an appropriate population for this data.

<table>
<thead>
<tr>
<th>Test Result:</th>
<th>E^{++}</th>
<th>E^{+}</th>
<th>E^{-}</th>
<th>D^+ Marginals</th>
</tr>
</thead>
<tbody>
<tr>
<td>With Patient:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D^{++}</td>
<td>(0.4)(0.1)</td>
<td>(0.5)(0.1)</td>
<td>(0.1)(0.1)</td>
<td>0.1</td>
</tr>
<tr>
<td>D^{+}</td>
<td>(p)(0.2)</td>
<td>(0.8)(0.2)</td>
<td>(0.2-p)(0.2)</td>
<td>0.2</td>
</tr>
<tr>
<td>D^{-}</td>
<td>(q)(0.7)</td>
<td>(0.1-q)(0.7)</td>
<td>(0.9)(0.7)</td>
<td>0.7</td>
</tr>
<tr>
<td>E^{+} Marginals</td>
<td>(0.04) + (0.2)p + (0.7)q</td>
<td>(.28) - (0.7)q</td>
<td>(0.68)- (.2)p</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.1: Probability table for simplified medical diagnosis problem. Cells represent joint event occurrences. Marginal probabilities are the sum of all joint probabilities involving a selected event.

We can use this table to infer:

\[
P(D^{++}|E^{++}) = \frac{P(E^{++},D^{++})}{P(E^{++})} = \frac{(0.04)}{[(0.04) + (0.2)p + (0.7)q]}
\]

and:

\[
P(D^{+}|E^{++}) = \frac{P(E^{++},D^{+})}{P(E^{++})} = \frac{(0.2)(p)}{[(0.04) + (0.2)p + (0.7)q]}
\]

It is easy to see that \(P(D^{++}|E^{++})\) is minimized when \(p\) and \(q\) are at their maximum and minimized when \(p\) and \(q\) are at their minimum. However, it is less obvious that \(P(D^{+}|E^{++})\) reaches its minimum when \(p=0\) and reaches it maximum when \(q=0\) and \(p=0.2\). We infer that \(0.266 \leq P(D^{++}|E^{++}) \leq 1.0\) and that \(0.0 \leq P(D^{+}|E^{++}) \leq 0.5\).
6.2.2.2 The Actual Case

Now we can consider the case where the doctor bounds $P(D^{++})$ and $P(D^+)$ Let $P(D^{++}) = \alpha$ be in the region $[0.1, 0.2]$ and $P(D^+) = \beta$ be in $[0.2, 0.3]$. We can introduce $\alpha$ and $\beta$ into the probability table as follows:

<table>
<thead>
<tr>
<th>Test Result:</th>
<th>$E^{++}$</th>
<th>$E^{+}$</th>
<th>$E^{-}$</th>
<th>$\text{Marginals}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>With Patient:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D^{++}$</td>
<td>$(0.4)(\alpha)$</td>
<td>$(0.5)(\alpha)$</td>
<td>$(0.1)(\alpha)$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>$D^{+}$</td>
<td>$(p)(\beta)$</td>
<td>$(0.8)(\beta)$</td>
<td>$(0.2-p)(\beta)$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>$D^{-}$</td>
<td>$(q)(1-\alpha-\beta)$</td>
<td>$(0.1-q)(1-\alpha-\beta)$</td>
<td>$(0.9)(1-\alpha-\beta)$</td>
<td>$1-\alpha-\beta$</td>
</tr>
</tbody>
</table>

| $E^+$ Marginals |          |        |        |                  |
| $(0.4)(\alpha)$  | $(0.5)(\alpha)$  | $(0.1)(\alpha)$  | $(0.9)(1-\alpha-\beta)$ | $1$              |
| $(p)(\beta)$     | $(0.8)(\beta)$   | $(0.2-p)(\beta)$ | $(0.9)(1-\alpha-\beta)$ |                  |
| $(q)(1-\alpha-\beta)$ | $(0.1-q)(1-\alpha-\beta)$ | $(0.1-q)(1-\alpha-\beta)$ |                  |                  |

Table 6.2: Probability table for fully specified medical diagnosis problem.

We use this table to infer:

$$P(D^{++}|E^{++}) = P(E^{++}, D^{++})/P(E^{++}) = (0.4)(\alpha)/[(0.4)(\alpha) + (p\beta) + (q)(1-\alpha-\beta)]$$

and:

$$P(D^{+}|E^{++}) = P(E^{++}, D^{+})/P(E^{++}) = (p\beta)/[(0.4)(\alpha) + (p\beta) + (q)(1-\alpha-\beta)]$$

subject to:

$$0.1 \leq \alpha \leq 0.2, \quad 0.1 \leq \beta \leq 0.2, \quad 0.0 \leq p \leq 0.1, \quad 0.0 \leq q \leq 0.1$$

which are not trivial to bound with respect to $p, q, \alpha,$ and $\beta$ by inspection.

6.2.3 Formulating as an RIK Problem

We can formulate the problem for the RIK algorithm as follows:

- Define $\{0,1\}$ basic events $D^{++}, D^+, D^-$ describing whether the patient has the malignant form of the disease, the benign form, or doesn’t have the disease. Similarly, define basic events $E^{++}, E^+$, and $E^-$ to represent testing “positive” for each form of the disease.
• Specify that events $D^{++}$, $D^+$, and $D^-$ are mutually exclusive and exhaustive. One and only one of the three cases can be true for the new patient. Do the same for $E^{++}$, $E^+$, and $E^-$ as the patient can have only one test result.

• Assess $P(D^{++})$ to be between 0.10 and 0.20 and $P(D^+)$ to be between 0.2 and 0.3. All other basic events have a priori probability bounds between 0.0 and 1.0 denoting no further information is available.

• Define conditional events $(E^{++}|D^{++})$, $(E^{++}|D^+)$, $(E^+|D^+)$, and $(E^-|D^-)$ with point probability assessments of 0.4, 0.5, 0.8, and 0.9 respectively.

• Define the target conditional event $(D^{++}|E^+)$ and solve for bounds on the event. Repeat for $(D^+|E^{++})$.

The corresponding input file for this problem with target $(D^+|E^{++})$ is shown in Figure 6.2.

```
// Medical Diagnosis Problem:
// - include malignancy, benignancy, health
// - wider a priori assessments
//
// CONDTARGET 1
ASSESSED 6
RELATED 5
RELATIONSHIPS 2
VarName VarNum LOW HIGH
D++ 0 0.1 0.2
D+ 1 0.2 0.3
D- 2 0.0 1.0
E++ 3 0.0 1.0
E+ 4 0.0 1.0
E- 5 0.0 1.0
(E++|D++) 6 0.4 0.4 COND 3 0
(E|D++) 7 0.5 0.5 COND 4 0
(E|D+) 8 0.8 0.8 COND 4 1
(E|D-) 9 0.9 0.9 COND 5 2
(D+|E++) 10 0.0 1.0 TCOND 1 3
MUTEX 0 1 2
MUTEX 3 4 5
TARGET 10
SENSE MAX
MAXCACHESIZE 30
VERBOSE 0
```

Figure 6.2: Input file specification for medical diagnosis problem.

Solving this RIK problem, we infer that $0.25 \leq P(D^{++}|E^{++}) \leq 1.0$ and $0.0 \leq P(D^+|E^{++}) \leq 0.6$. Clearly, as $0 \leq P(D^{++}|E^{++}) + P(D^+|E^{++}) \leq 1.0$, we can jointly constrain these
probabilities further. We see the exact effect of widening the a priori assessment on the target bounds.

6.2.4 Other Considerations

The analysis of the medical diagnosis problem raises issues. First the number of people tested has a direct effect on the tightness of the implied bounds. Although the above problem used exact estimates for test sensitivities, a more precise approach is to incorporate uncertainty about sensitivities and to take into account the size of the sample population.

An analytical problem worth exploring is to allocate our test resources optimally given that we wish to bound $P(D^{++}|E^{++})$ as tightly as possible. By incorporating a cost model for the sampling process, we can begin to decide which tests to offer to which people.

6.2.5 Conclusions

Although we can analyze the medical diagnosis problem in closed form using probability tables, incomplete assessments result in a non-trivial nonlinear minimization or maximization problems. The RIK framework incorporates incomplete assessments easily and leads to a quick solution.

Generally speaking, the more incomplete our knowledge, the harder the problem is to solve by Bayesian methods. For larger Bayesian inference problems where our knowledge is incomplete, the RIK algorithm will provide a useful and flexible means for computing exact probabilistic bounds.
6.3 Reconfigurable Fault Detection Systems

In “Closed-Form Solution of Decomposable Stochastic Models,” Jon Sjogran studies a reconfigurable fault detection problem [5]. We compare the RIK analysis with Sjogren’s in order to understand the impact of the assumption that some components of the error system and some components of the fault detection system shown in Figure 6.3 are probabilistically independent. We chose this problem in part for its scalability -- the RIK problem size grows exponentially with our analytical precision -- enabling us to figure out the biggest problem size the algorithm can solve in reasonable time and to benchmark the impact of future enhancements to the algorithm.

6.3.1 Problem Statement

Consider a fault-tolerant fighter jet control system with a transient-fault detection mechanism as illustrated in Figure 6.3. In state A, the mechanism is functioning normally – we are able to detect arbitrary faults in the system. In state B, transient faults (for instance a bird temporarily interfering with radar) are incorrectly diagnosed as being permanent. In state C, a rare kind of error causes spurious signals to be sent to external...
parts of the system, causing an overall crash (in our case the ejection of the plane’s pilot!).

We model the underlying fault process as a Markov with transitions from state $Y$, the absence (disappearance) of transient faults, to and from state $X$, the presence (emergence) of transient faults.

We consider the system as being “up” at time $t$, whenever we are in the joint states, \{A(t), X(t)\}, \{A(t), Y(t)\}, or \{B(t), Y(t)\}. However whenever the system enters the joint state \{B(t), X(t)\}, we incorrectly diagnose a transient fault as permanent and the system crashes (pilot ejects). Similarly, in state \{C(t), X(t) or Y(t)\} we crash due to catastrophic detector error.

Transitions between states are governed by the exponential waiting time density function:

$$f_{\text{waiting time}}(T) = \eta \exp(-\eta T)$$

where $\eta$ is our arrival rate. The associated cumulative distribution function is:

$$F_{\text{waiting time}}(T) = 1 - \exp(-\eta T).$$

An important property of the exponential distribution is that it is memoryless. Given that we have waited until time $\tau$ without a transition, the waiting time after time $\tau$ is an exponential random variable with the same rate. Thus, knowledge of past behavior tells us nothing about future behavior.

Let the arrival rates of the transitions be as shown in Figure 6.3. The arrival rate for transitions from A to B is $\alpha$, from B to A is $\beta$, and from B to C is $\lambda$. The arrival rates for the X to Y and Y to X transitions are $r$ and $s$ respectively. Given numerical values for parameters $\alpha$, $\beta$, and $\lambda$, we can completely characterize the probability law of the detection system. Given numerical values for $r$ and $s$, we can completely characterize the detection system and underlying error process.

We wish to know the probability that the process enters the absorbing state BX by a time $T$ given that we start in state AX at time zero.
6.3.2 Analysis

6.3.2.1 Assuming Independence

Let us examine the system under the standard assumption that the fault detection system and error process are probabilistically independent. One approach we can take is to simply discretize our time scale and analyze each system for each time step $\Delta t$ as an independent increment of a Markov Chain. As we let $\Delta t \to 0$, we get a closer approximation to the true continuous time behavior.

The probability that a particular transition governed by an exponential process with parameter $\eta$ occurs in a time step $\Delta t$ is just the CDF, $P(\text{Time to Transition} < \Delta t) = 1 - \exp(-\eta \Delta t)$. Similarly, the probability of a no transition is just $\exp(-\eta \Delta t)$. By exploiting the memoryless property we see that the transition probabilities are time invariant; knowing whether or not we transitioned in the last $\Delta t$ segment tells us nothing about the next $\Delta t$ segment. Our independence assumption enables us to analyze the joint behavior of the fault detection and error systems by simply multiplying the transition probabilities, constrained by the logic of the absorbing states. A diagram of the joint process under independence is shown in Figure 6.4.

By iterating the joint system Markov Chain $k$ times, we can analyze the probability of reaching the absorbing states at time $k\Delta t$ given we start in states $A$ and $X$. To do this, we simply raise $P_{\Delta t}$, the transition matrix, to the $k^{th}$ power.

The results from this approach are shown in Figure 6.5, where $\Delta t$ is taken to be 0.01, $\alpha = 0.3$, $\beta = 0.5$, $\lambda = 0.1$, $r = 2$ and $s = 3$. 

49
Sjogren solves the continuous time version of the problem and calculates the closed form result that the CDF for the waiting time to absorption into state BX when the process starts in AX. That is, he shows that:

\[
P(BX \text{ by time } t \mid AX \text{ at time } 0) = 0.98884551031790 - 0.05848988319612 \times \exp(-5.35172855471732 \times t) + 0.08378848442687 \times \exp(-3.56645190997164 \times t) - 1.01414411154865 \times \exp(-0.28181953531104 \times t)
\]

The graph of this equation is visually indistinguishable from our discrete approximation to it.
6.3.2.2 Without Probabilistic Independence

The assumptions that the systems are probabilistically independent may not be a good assumption in this case, especially if transient faults may have some intermittent causal effect on the detector’s ability to function or vice versa. If we do away with independence, then we must consider the joint state probabilities for the Fault Detection System and Error Process as we may no longer say that, for example, 

\[ P(AX \rightarrow BX \text{ in } (0,\Delta t)) = P(A \rightarrow B \text{ in } (0,\Delta t)) P(X \rightarrow X \text{ in } (0,\Delta t)). \]

In essence, the only information directly available is marginal information about the error process acting alone and the fault detection system acting alone. To do this, we once again discretize, using a small enough \( \Delta t \) such that the probability of two transitions in the same system is negligible.

Now, consider the case where we are in states A and X at time 0. We know that the probability of going to state B from A in the Fault Detection System in the next time step is 

\[ 1 - \exp(-\alpha \Delta t), \]

assuming the error process does not transition first to a demobilizing state. What effect could the error process have on this transition probability? In this case, none.
We see that a transition in the Error Process simply means a transition to Joint State \(AY\), from which we transition to \(BY\) with the same probability as \(AX\) to \(BX\). The order of the transitions \(A \rightarrow B\) and \(X \rightarrow Y\) does not effect our probability of winding up in state \(B\). Another way to see this is to consider the behavior in time – the exponential arrival process from \(A\) to \(B\) will not be affected by an arrival from \(X\) to \(Y\) at some time \(\tau \in (0, \Delta t)\).

However, not all transitions are unaffected by a transition in the other system. Consider transitions from state \(B \rightarrow A\) when in joint state \(BY\) at the beginning of a time step. In the best case, no transition \(Y \rightarrow X\) occurs, and the probability of a \(B \rightarrow A\) transition is just \((1 - \exp(-\beta \Delta t))\). However, if a \(Y \rightarrow X\) transition does occur, we reach absorbing state \(BY\), and cannot make another transition. The worst case for our \(B \rightarrow A\) transition is an instantaneous transition from \(Y \rightarrow X\) at the start of the time step, resulting in a zero probability of transitioning \(B \rightarrow A\).

Thus, without assuming independence, we can make the following assessments on the transition probabilities for \(k \in \{0, 1, \ldots, N\}\) where \(N\) is the maximum number of time steps:

- \(P(A_{k\Delta t} \rightarrow B_{k\Delta t+1} \mid AX_{k\Delta t}) = 1 - \exp(-\alpha \Delta t)\)
- \(0 \leq P(B_{k\Delta t} \rightarrow A_{k\Delta t+1} \mid BY_{k\Delta t}) \leq 1 - \exp(-\beta \Delta t)\)

Continuing along this line of reasoning we can finalize our list of assessments:

- \(P(A_{\Delta t} \rightarrow A_{\Delta t+1} \mid AX_{\Delta t}) = \exp(-\alpha \Delta t)\)
- \(P(A_{\Delta t} \rightarrow C_{\Delta t+1} \mid AX_{\Delta t}) = 0\)
- \(P(A_{\Delta t} \rightarrow A_{\Delta t+1} \mid AY_{\Delta t}) = \exp(-\alpha \Delta t)\)
- \(P(A_{k\Delta t} \rightarrow B_{k\Delta t+1} \mid AY_{k\Delta t}) = 1 - \exp(-\alpha \Delta t)\)
- \(P(A_{k\Delta t} \rightarrow C_{k\Delta t+1} \mid AY_{k\Delta t}) = 0\)
- \(\exp(-(\beta + \lambda) \Delta t) \leq P(B_{k\Delta t} \rightarrow B_{k\Delta t+1} \mid BY_{k\Delta t}) \leq 1\)
- \(0 \leq P(B_{k\Delta t} \rightarrow C_{k\Delta t+1} \mid BY_{k\Delta t}) \leq 1 - \exp(-\lambda \Delta t)\)
- \(\exp(-r \Delta t) \leq P(X_{k\Delta t} \rightarrow X_{k\Delta t+1} \mid AX_{k\Delta t}) \leq 1\)
- \(0 \leq P(X_{k\Delta t} \rightarrow Y_{k\Delta t+1} \mid AX_{k\Delta t}) \leq 1 - \exp(-r \Delta t)\)
- \(P(Y_{k\Delta t} \rightarrow X_{k\Delta t+1} \mid AY_{k\Delta t}) = 1 - \exp(-\lambda \Delta t)\)
- \(P(Y_{k\Delta t} \rightarrow Y_{k\Delta t+1} \mid AY_{k\Delta t}) = \exp(-s \Delta t)\)
\[ 0 \leq P(Y_{k\Delta t} \rightarrow X_{k\Delta t+1} | BY_{k\Delta t}) \leq 1 - \exp(-s\Delta t) \]
\[ \exp(-s\Delta t) \leq P(Y_{k\Delta t} \rightarrow Y_{k\Delta t+1} | BY_{k\Delta t}) \leq 1 \]

All transitions from BX lead to BX and all transitions from CY lead to CY as they are absorbing states.

**6.3.2.3 Specifying the Non-Independence Case in the RIK framework**

We can solve the non-independence case for bounds on \( P(BX_{k\Delta t} | AX_0) \) using the RIK algorithm, although the size of the problem grows linearly with our analytical precision.

Specifying the system in the RIK format is simple. It is just a specification of a discrete time stationary Markov chain with bounded transition probabilities. We create a variable for each of the possible joint states at each time period (e.g. \( \{AX_0, AX_1, AY_1, BY_1, \ldots, CY_{N\Delta t}\} \)), and represent individual states of the fault detection system and error process by using related “OR” variables. For instance, the variable representing the state that the fault detector is in A at timestep one, \( A_1 \), is defined as \( \text{OR}(AX_1, AY_1) \).

The related conditional assessments are as stated above. These assessments represent fully our limited knowledge of the system. All that remains is to use MUTEX relationships to ensure we are only in one joint state at each time step and ABSORB relationships to ensure that once we are in states BX and CY, we cannot leave.

We considered an alternative representation whereby the individual states were denoted by basic events and the joint states by “AND” relations, but chose the above representation for its smaller size.

**6.3.2.4 Solving a Looser Version of the Non-Independence Case with the RIK**

Unfortunately, the RIK algorithm requires over five hours to solve the fully-specified version of the fault detection problem on a Sun Sparc 20. In order to achieve preliminary results, we decided to remove all inequality constraints from our specification of the problem, reducing its size dramatically. This means we infer (potentially) looser bounds on the probability of being in state BX at the time in question. We chose this course of action, in part, because we observed that removing the inequality constraints had no
effect on the bounds in the few cases we analyzed. However, we did not attempt to prove that the inequality constraints provide no extra information.

With the inequality constraints removed, the RIK algorithm can solve a problem of ten time steps in about 30 minutes. Solving the fault detection problem for upper and lower bounds with N fixed at 10 and \( \Delta t \) ranging from 0.05 to 0.50 gives the results shown in Figure 6.6.

![Graph showing Upper & Lower Bounds with N=10](image)

**Figure 6.6:** Upper and lower bounds on the probability of transition to BX by time \( t \) without assuming independence of detector system and error process. Uses reduced constraint set.

We see that the probability of being in BX at time \( t \) under independence is very close to our calculated upper bound. The lower bound stays at zero for all time because we allow for the possibility that all transitions to state B are to joint state BY. We conclude that the worst-case behavior, which maximizes our likelihood of winding up in the absorbing state (ejecting the pilot because of a detector error), is not much worse than the behavior under the independence assumption. We state this with confidence only for the tested time region, \( t \leq 5 \).
Removing the inequality constraints appears to have little effect on the upper bound. The largest the effect could be is the small difference between the generated bounds and the independence-generated probability curve.

### 6.3.2.5 Accuracy Issues

To analyze the fault detection system with reasonable accuracy, $\Delta t$ must be small enough that the probability of two or more transitions in the same process is negligible. The tradeoff is that for smaller $\Delta t$, we require more time steps $N$ to infer the bounds at the same time $t$. Unfortunately, for us to solve the RIK algorithm in a few hours $N$ cannot exceed sixteen. Until the algorithm is made significantly faster, negotiating this tradeoff will remain an unpleasant task.

To examine the ramifications of increasing the discrete time step, let us examine the probability of two transitions occurring from the same exponential arrival process within time $\Delta t$. Let $\eta$ be the arrival rate:

\[
P(\geq 2 \text{ transitions in } [0, \Delta t]) = 1 - P(1 \text{ transition in } [0, \Delta t]) - P(0 \text{ transitions in } [0, \Delta t])
\]

\[
= 1 - \left[ \int_{0}^{\Delta t} P(\text{transition at } \tau)P(\text{no transition in } [\tau, \Delta t])d\tau \right] - \exp(-\eta\Delta t)
\]

\[
= 1 - \left[ \int_{0}^{\Delta t} (\tau)\exp(-\eta\tau)(1 - \exp(-\eta(\Delta t-\tau)))d\tau \right] - \exp(-\eta\Delta t)
\]

\[
= 1 - (\eta\Delta t)\exp(-\eta\Delta t) - \exp(-\eta\Delta t)
\]

For $\eta=1.18$ (the average among our process arrival rates), the probability of two or more transitions occurring in the interval $\Delta t$ as a function of $\Delta t$ is shown in Figure 6.7:

If we are willing to suffer a 10% chance on average of two transitions from the same process (a pretty severe concession), then we can allow $\Delta t$ to be as high as 0.5. Abiding
by this rule yields the graph of Figure 6.6. Note that the probability of two transitions in

Figure 6.7: Graph of the probability of two transitions from an exponential
process with rate 1.18 within a time step vs. the size of the time step.

Δt is higher for the fault detection system than for the error process, because it has
smaller arrival rates. This means than by using Δt=0.5 we actually suffer higher than a
10% chance of invalidating our model in each time step of the fault detection system.
Until the algorithm is faster, however, we must incur this invalidation to get bound
estimates for times greater than five.

6.3.2.6 Conclusions
We conclude, with caveats about our precision, that the independence assumption does
not cost us much in terms of understanding worst-case behavior.
Chapter 7

7 Performance

The bottleneck for solving almost all EPI problems with the RIK algorithm is the Related (Mixed) Integer Program. The number of RIP constraints grows in proportion to the number of related variables (3 constraints for each COND, 2 for each AND/OR, etc.) and relationships (1 constraint each) in the system. The time to solve an iteration of the RIP, in turn increases almost exponentially.

The RLP solves quickly relative to the RIP. This is due in part to the fact that CPLEX can “re-use” much of the work from past iterations to optimize. At each iteration the RLP basis only changes slightly. Each iteration of the RIP, on the other hand, requires a completely new branch-and-bound solution as the objective function changes every time.

Note that the RLP performance is proportional to the maximum cache size. If we fill the cache before we reach a solution, the size of the RLP will not change and the time to solve the RLP will remain relatively fixed. By making the cache size large, we make for a slower RLP but reduce the number of times the RIP has to regenerate a previously generated column. Because each RIP solution is so time-expensive relative to each RLP, a large cache size makes sense. Deciding the optimal tradeoff between time to solve each RLP and the time spent regenerating old columns in the RIP is an area for future research exploration.

In order to see how well our algorithm performs, we ran the fault detection problem with $\Delta t = 0.5$ for $N$ ranging from 1 to 10. The results are shown in Table 7.1 (the asterisked columns denote the time it takes to reach the correct answer within $10^{-6}$ precision). There are two dynamics at work here; (1) larger problems have more RLP constraints and thus
require more RIK iterations to eliminate the artificial columns and fill the basis with legitimate columns, and (2) larger problems (typically) have more RIP constraints, meaning each iteration takes longer to solve. The combination of these two leads to the fast growth in the overall solution time.

Figure 7.1 displays the number of iterations required to solve the problem (within $10^{-6}$ precision) as a function of N. Figure 7.2 shows the average time to solve each iteration as function of the number of RIP constraints and Figure 7.2 shows the total time required versus N.

<table>
<thead>
<tr>
<th>N</th>
<th># Basic Variables</th>
<th># Related Variables</th>
<th># Relationships</th>
<th># RLP Constraints</th>
<th># RIP Constraints</th>
<th>RIK Time (sec)</th>
<th>RIK Iterations</th>
<th>RIK* Time (sec)</th>
<th>RIK* Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>10</td>
<td>9</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>8</td>
<td>6</td>
<td>20</td>
<td>26</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>13</td>
<td>10</td>
<td>30</td>
<td>43</td>
<td>3</td>
<td>30</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>18</td>
<td>14</td>
<td>40</td>
<td>60</td>
<td>9</td>
<td>40</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>26</td>
<td>23</td>
<td>18</td>
<td>50</td>
<td>77</td>
<td>32</td>
<td>50</td>
<td>9</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
<td>28</td>
<td>22</td>
<td>60</td>
<td>94</td>
<td>79</td>
<td>60</td>
<td>30</td>
<td>35</td>
</tr>
<tr>
<td>7</td>
<td>36</td>
<td>33</td>
<td>26</td>
<td>70</td>
<td>111</td>
<td>101</td>
<td>70</td>
<td>54</td>
<td>35</td>
</tr>
<tr>
<td>8</td>
<td>41</td>
<td>38</td>
<td>30</td>
<td>80</td>
<td>128</td>
<td>492</td>
<td>80</td>
<td>277</td>
<td>49</td>
</tr>
<tr>
<td>9</td>
<td>46</td>
<td>43</td>
<td>34</td>
<td>90</td>
<td>145</td>
<td>1667</td>
<td>90</td>
<td>947</td>
<td>67</td>
</tr>
<tr>
<td>10</td>
<td>51</td>
<td>48</td>
<td>38</td>
<td>100</td>
<td>162</td>
<td>5545</td>
<td>100</td>
<td>3204</td>
<td>77</td>
</tr>
</tbody>
</table>

Table 7.1: Time and iterations of RIK algorithm required to solve the fault detection problem with N time steps.
Figure 7.1 Number of iterations as a function of time steps for the fault detection example. Solution stopped when within $10^{-6}$ of optimality.

Figure 7.2 Iteration time as a function of RIP constraints for the fault detection example. Solution stopped when within $10^{-6}$ of optimality.

Figure 7.3 Overall solution time as a function of the number of time steps for the fault detection example. Solution stopped when within $10^{-6}$ of optimality.
8 Conclusions

The primary goal of this project was to develop and test an efficient implementation of the RIK algorithm that would allow for incremental improvements with little structural change. I believe we have achieved this. The C++ program is able to parse the input file and set up the problem quickly, leaving the solution of the RIP in CPLEX as the speed-determining factor.

The second goal was to explore some applications of the RIK algorithm to understand the value it can add to inference problems and motivate future algorithm enhancements. The RIK algorithm adds a great deal of value to fault tree analysis, giving a system designer an exact understanding of the worst-case probabilistic behavior of his system by removing the assumption of component independence. The RIK algorithm can support a wider variety of logic gates than traditional fault tree analysis programs and easily integrates incomplete assessments of component reliability. One disadvantage of the RIK approach to fault tree analysis is that assessments must be in the form of bounds. Thus if we have assessed a components failure process in terms of an approximate distribution (e.g. log-normal), we lose information in the conversion to the FTP specification.

The RIK also adds value to a probabilistic inference system where incomplete information makes traditional analysis difficult. The medical diagnosis problem presented here illustrates this. For larger Bayesian networks with incomplete information, the direct Bayesian approach becomes even more difficult.

The fault detection example reveals the biggest disadvantage of the RIK approach. The time needed to solve an RIK problem grows nearly exponentially with the size of the problem. With the current implementation, we must limit the problem size to less than 250 RIP constraints to solve the problem in reasonable time.
8.1 Advantages

Summing up the main advantages of the RIK algorithm over traditional techniques we see that:

- The RIK model provides a universal framework for probabilistic inference on discrete events.
- The RIK model seamlessly handles incomplete information.
- The RIK model yields an exact solution, with no need for overarching independence assumptions.

8.2 Disadvantages

Conversely, the chief disadvantages versus traditional means of inference are:

- The RIK model problem is slow for large problems.
- The RIK cannot handle “abstract” independence information such as $P(A|B) = P(A) \ [11]$.
- The RIK model cannot incorporate distribution assessments for the probability of an event, only point and bound assessments.

8.3 Suggestions for Further Research

There is still much work that can be done to enhance the RIK algorithm. To make it run faster, we can optimize the RIP, perhaps stopping short of generating optimal columns in early iterations. We are content to generate good columns quickly if some remain in the final basis. To do this, we can consider limiting the time that we give CPLEX to solve the RIP or explicitly cut off the branch-and-bound routine when we get sufficiently close to the best column that the iteration can attain. We can also consider optimizations to the RLP and experiment with cache sizes.

Another useful addition that would speed up the algorithm is a more efficient logic parsing mechanism. If we could define related variables with more than one logical or conditional statement we could reduce drastically the number of related variables needed to solve a problem. Consider the following case. We wish to bound the probability that $Y = \text{NOT}(\text{OR}(\text{AND}(X,Z),W))$ occurs. To specify $Y$ in the current implementation we must specify each relation individually (i.e. $Y = \text{NOT} \ A$, $A = \text{OR}(B,W)$, and
B=AND(X,Z)) introducing supplementary related variables. By parsing the logic in a more sophisticated manner we can reduce the number of RLP constraints and also the number of RIP constraints.

There are enhancements to be made as far as the types of relationships the RIK algorithm can handle. For example, we can implement support for abstract constraints of the form \( P(A) = P(B) \) or more generally \( p_1 \leq P(A) \pm P(B) \leq p_2 \) by defining a new related variable “E” such that \( r_E = r_A \pm r_B \) by adding one constraint to the RIP and setting \( p_1 \leq r_{Eq} \leq p_2 \) in the RLP. Currently, this can be accomplished by use of a “LINEAR” relationship, but the more explicit relationships we can specify the easier the program will be for the user.

Finally, an interesting question is how to best formalize and capitalize upon assessment data. Should we first model a system with a standard stochastic process or feed actual data in the system with estimated bounds adjusted according for sample size? How should we account for the opinions of experts? Developing a general approach to this “assessment problem” would be a worthwhile exercise. An interesting attempt at formalizing such an approach for belief networks is made by Mark Druzdel and Linda C van der Gaag in their 1995 paper “Elicitation of Probabilities for Belief Networks: Combining Qualitative and Quantitative Information [10].”
Chapter 9

9 Appendices

A) CPLEX matrix storage structure:

CPLEX uses a unique representation for its constraint matrices. To reduce the storage space needed for sparse matrices and to speed up column manipulation it stores the constraint matrix in four separate structures:

\textit{matval}: The non-zero matrix entries listed column by column.

\textit{matind}: The corresponding matrix row for each entry in \textit{matval}.

\textit{matbeg}: Entry \( i \) contains the index of the beginning of column \( i \) of the matrix.

\textit{matcnt}: Entry \( i \) contains the number of non-zero elements in column \( i \).

Note that row and column indices are numbered starting at zero.

As an example, the 3 by 5 matrix:

\[
\mathbf{H} = \begin{bmatrix}
0 & 2 & 1 & 0 & 9 \\
3 & 0 & 4 & 0 & 7 \\
0 & 0 & 0 & 5 & 3
\end{bmatrix}
\]

has CPLEX representation:

\textit{matval}_\mathbf{H} = [3,2,1,4,5,9,7,3]

\textit{matind}_\mathbf{H} = [1,0,0,1,2,0,1,2]

\textit{matbeg}_\mathbf{H} = [0,1,2,4,5]

\textit{matcnt}_\mathbf{H} = [1,1,2,1,3]

See the CPLEX manual [14] for further information.


**Bibliography**


