SELECTED PROBLEMS OF MINIMIZATION OF VARIABLE-VALUED LOGIC FORMULAS

by

Roland Phillipe Cuneo

May 1975
SELECTED PROBLEMS OF MINIMIZATION
OF VARIABLE-VALUED LOGIC FORMULAS

by

Roland Phillipe Cuneo

May 1975

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported in part by the Department of Computer Science and was submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, 1975.
ACKNOWLEDGMENT

I wish to express my thanks to my advisor, Professor R. S. Michalski, who suggested the thesis topic and who has provided invaluable counsel. I also wish to thank my friends, especially Mary Brownlee, for their support and understanding.

Special thanks go to Connie Slovak for her patience and efficient typing of this manuscript.
PREFACE

In recent years there has been a growing interest in the use of logic for pattern recognition, machine learning and other related fields. Such names as Bibel [6], Michalski [1]-[5], Stoffel [7], and others have been associated with research in the above-mentioned area.

This thesis is concerned with the implementation of algorithms for optimization of formulas of the variable-valued logic system $VL_1$. This thesis is also concerned with the use of an iterative process to obtain an optimal solution. By iterative process we mean a process by which a solution obtained at a given stage is used as the starting point to obtain a better solution at the next stage. Although this process might not produce an optimal solution, it is an efficient way to generate quasi-optimal or optimal solutions.

The first two chapters contain the basic background material to make this thesis self-contained. The third chapter describes the optimization algorithms, and Chapter 4 shows how we implemented these algorithms. It also contains a user's guide. Finally we have in Chapter 5 two examples of how this program may be used.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION TO VL&lt;sub&gt;1&lt;/sub&gt; AND GLD.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>SYNTHESIS OF COVERS.</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>MINIMIZATION ALGORITHMS.</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>3.1 Algorithm for Formula Minimization</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>3.2 Iterative Applications</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>IMPLEMENTATION OF THE ALGORITHMS.</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>4.1 Implementation.</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>4.2 User's Guide.</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>EXAMPLES.</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>LIST OF REFERENCES.</td>
<td>55</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Generalized Logical Diagram (GLD) representing ( E(d_1, d_2, d_3, d_4) )</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>The algorithm without generalization for VL mode</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>The algorithm without generalization for DC and IC mode</td>
<td>19</td>
</tr>
<tr>
<td>3.3</td>
<td>The algorithm with generalization for VL mode</td>
<td>20</td>
</tr>
<tr>
<td>3.4</td>
<td>The algorithm with generalization for DC mode</td>
<td>21</td>
</tr>
<tr>
<td>3.5</td>
<td>The algorithm with generalization for IC mode</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Data structure for the representation of a complex</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>Initial binary bit string obtained to later represent a complex in the event space ( E(3,4,5,2,4) )</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>Binary string and how variable and values are associated to it</td>
<td>26</td>
</tr>
<tr>
<td>4.4</td>
<td>GLD of Example I</td>
<td>27</td>
</tr>
<tr>
<td>4.5</td>
<td>GLD of Example II</td>
<td>28</td>
</tr>
<tr>
<td>4.6</td>
<td>The GLD diagram of Example I</td>
<td>38</td>
</tr>
<tr>
<td>4.7</td>
<td>The GLD diagram of Example II</td>
<td>43</td>
</tr>
<tr>
<td>5.1</td>
<td>The GLD representation of the input covers in Example I</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>The GLD representation of the resulting output covers in Example I</td>
<td>47</td>
</tr>
<tr>
<td>5.3</td>
<td>Coloring map of U.S.A</td>
<td>51</td>
</tr>
</tbody>
</table>
This first chapter is an introduction to the formal system used, which is called VL₁. The VL₁ system is a specific system which is a subset of a more general variable-valued logic (VL) system. The basic idea behind a VL system is that every VL formula and each of its variables may receive its own independent truth values, which are selected based on semantic and problem oriented considerations [1]. Truth values in a VL system differ as to the usual or standard binary value logic in that it doesn't only have true and false as its possible values. It may have more than two values if so desired, and each variable in a VL system doesn't have to have the same number of truth values since each variable in a VL system takes on its truth values from different domains.

The concept of the VL system and the definition of VL₁ were first introduced [2] at the IFIP Working Conference on Graphic Languages, Vancouver, Canada, May 1972.

The VL₁ system may be used in Pattern Recognition and Machine Learning since with its formulas one may describe an object or set of objects. Since the variables of the VL₁ system take on values from different domains one may have a variable taking on different values for different colors of an object and another variable may be taking different values for different predetermined sizes of objects, and so on, thus setting up a description of objects, people, and even concepts or conditions. Since one defines the domain from which
a variable takes on its values, one may make the domains consist of as many or as few truth values as one considers necessary. Thus making the variable which takes its values from that domain very precise or less precise. For example, say one has a domain of length, it may consist of three values (small, medium, large) or it may be more precise by containing seven values (one foot, two feet, ..., seven feet) and one may notice that one can make it even more precise depending on what you are trying to describe and how much precision is required.

From these domains one may set up a $V_{L_1}$ formula describing an object using the rules given later on. After having set up such a $V_{L_1}$ formula, we may use this formula to recognize the same or similar objects and reject dissimilar objects.

The algorithm discussed in the following chapters is mainly interested in minimizing these descriptions, thus giving the most compact and simplest rules that will enable the machine to recognize an object just by checking if it satisfies the $V_{L_1}$ formula describing it.

Following are the basic definitions and concepts of the $V_{L_1}$ system as described in papers [1] and [3].

BASIC DEFINITION

Definition of the $V_{L_1}$ System

The $V_{L_1}$ system is an ordered quintuple

$$(X, Y, S, R_P, R_I)$$

where

$X$ -- is a finite non-empty (f.n.) set of input or independent variables whose domains, denoted by $D_i$, $i=1,2,3,...$, are non-empty sets where
\[ D_i = (0, 1, 2, \ldots, d_i) \quad i = 1, 2, 3, \ldots \]

and \( d_i \) is a natural number.

\( Y \) -- is a f.n. set of output or independent variables, whose
domains, denoted \( jD, j = 1, 2, 3, \ldots \), are non-empty sets.

\( S \) -- is a f.n. set of symbols, called connecting symbols.

\( R_F \) -- is a set of the formation rules which define wff formulas
in the system \( (V \bot \text{ formulas}) \):

1. An element of \( D \) standing alone is a wff.
2. A form \([L \# R]\) is a wff, iff:
\[ \# \in \{=, \neq, \leq, \geq\}, L \in (X, V_1), R \in (c, V_2) \]
\( X \) is a sequence of symbols \( X_i \), \( i \in I, I \subseteq \{1, 2, \ldots, n\} \),
where \( X_i \) is a variable \( x_i \) or a form \( \neg x_i \) (also written
as \( \overline{x_i} \)), separated by symbol \( + \). Or a name (when \( x \)
denotes a variable whose values are such sequences)
of such a sequence.

\( c \) is a sequence of different non-negative integers
ordered by the relation \(<\) (i.e. if \( c_i < c_j \), the \( c_i \)
will appear before \( c_j \) in the sequence) and separated
by ',' or ':'. \( c \) also may be a name of a sequence
such as defined above.

\( V_1, V_2 \) are wffs or names of wffs.

3. Forms \((V), \neg V, V_1 \land V_2 \) (also written \( V_1V_2 \)), \( V_1 \lor V_2 \),
\( V_1 \lor V_2 \) and \( V_1 | V_2 \) where \( V, V_1 \) and \( V_2 \) are wffs or names of
wffs, are wffs.
\( V \) is called the inverse of \( V \).

\( V_1V_2 \) is called the product (or conjunction) of \( V_1 \) and \( V_2 \).

\( V_1 \backslash V_2 \) is called the exception \( V_2 \) from \( V_1 \) (or \( V_1 \) except \( V_2 \)).

\( V_1 \lor V_2 \) is called the sum (or disjunction) of \( V_1 \) and \( V_2 \).

\( V_1 \mid V_2 \) is called the separation of \( V_1 \) and \( V_2 \).

\( R_1 \) -- is a set of interpretation rules which assign to any \( VL_1 \) formula \( V \) a value \( v(V) \in D \), depending on values of \( x_1, \ldots, x_n \):

1. The value \( v(c) \) of an element \( c \in D \), is \( c \), which is denoted:

\[
v(c) = c
\]

2. \( v([L \# R]) = \begin{cases} 1 & \text{if } v(L) \# v(R) \\ 0 & \text{otherwise} \end{cases} \)

where

\( v(L), L \in \{X, V_1\}, \) is \( v(V_1) \), i.e., value of wff \( V_1 \), if \( L \) is \( V_1 \), otherwise is \( v(X) \), i.e., value of the sequence \( X \).

Assuming that \( X \) is a sequence of literals, \( \hat{x}_i, i \in I \), separated by '+', \( v(x) \) is defined as:

\[
v(x) = \sum_{i \in I} v(\hat{x}_i),
\]

where \( \sum \) denotes arithmetic sum

\[
v(\hat{x}_i) = \begin{cases} x_i & \text{if } \hat{x}_i \text{ is } x_i \\ \bar{x}_i & \text{if } \hat{x}_i \text{ is } \bar{x}_i \end{cases}
\]

\( \hat{x}_i \) denotes a value of variable \( x_i, x_i \in D_i \).

\( v(R), R \in \{c, V_2\}, \) is a sequence \( c \), if \( R \) is \( c \); otherwise

\[
v(V_2) \ \ v(L) \# v(R), \ \ # \in \{=, \neq, \leq, \geq\}, \text{ is true if } v(L) \text{ is in relation } \# \text{ with } v(R). \ \ \text{If } R \text{ is } c, \text{ then:}
\]
\[ V(L) = c \quad (v(L) \neq c) \] is true if \( v(L) \) is (is not)
one of the integers in \( c \), or is (is not)
between any pair of integers in \( c \) separated
by ':'.

\[ V(L) \leq c \quad (v(L) \geq c) \] is true if \( V(L) \) is smaller than
or equal to (greater than or equal to) every
integer in \( c \) (in normal use of these relations
\( c \) will consist of just one element)

If \( v(L) \neq v(R) \), then the selector \([L \neq R]\) is said to be satisfied.

3. \( v((V)) = v(V) \)
\[ v(\neg V) = \overline{v(V)} \]
\[ v(V_1 \wedge V_2) = v(V_1 \cdot V_2) = \min\{v(V_1), v(V_2)\} \]
\[ v(V_1 \vee V_2) = v(V_1), \text{ if } v(V_2) = 0, \text{ otherwise } 0 \]
\[ v(V_1 \vee V_2) = \max\{v(V_1), v(V_2)\} \]
\[ v(V_1|V_2) = \begin{cases} v(V_1) \text{ if } v(V_2) = 0 \\ v(V_2) \text{ if } v(V_1) = 0 \\ 0 \text{ otherwise} \end{cases} \]

Parenthesis have their usual meaning, i.e. they denote a part
of the formula to be evaluated as a whole. Operations are ordered from
one with the highest to one with the lowest priority as follows:
\(- \wedge \vee \mid \).

If there is more than one operation \( \vee \), the priority goes from
the leftmost one to the rightmost one unless it is changed by parentheses.

The interpretation rules \( R_1 \) interpret \( VL_1 \) formulas as
expressions of a function \([1]\):
where $x$ is the cartesian product and $\rightarrow$ means 'mapping into'.

The universe of events or event space is all the possible sequence of values of input variables of the set $D_1 \times D_2 \times \ldots \times D_n$ where $D_i = \{0,1,\ldots, d_i\}$ for $i=1,2,\ldots,n$. Thus the event space is represented by $E(d_1,d_2,\ldots,d_n)$ where $d_i$ is the cardinality of $D_i$.

The elements of the event space $E$ are called events and represented by the vector $(x_1,x_2,\ldots,x_n)$, where $x_i$ is a value of the variable $x_i \in D_i$. Each event has a unique number which specifies it which is called the number of the event $e$ and denoted by $\gamma(e)$,

$$\gamma(e) = x_n + \sum_{k=n-1}^{1} x_k \prod_{i=n}^{k+1} d_i$$

to obtain the event given the number of the event one simply divides the number $\gamma(e)$ first by $d_n$ to obtain $x_n$ which is the remainder of the division, then divides the result by $d_{n-1}$ to obtain $x_{n-1}$ which is the remainder and applies this algorithm successively until one obtains $x_1$.

A geometrical model of the space $E$ is introduced in paper [4]. Following are the basic concepts and design of this planar geometric model of the event space $E$.

Given an event space $E$ such that $E = (d_1,d_2,\ldots,d_n)$. To make a visual representation of this event space one takes a rectangle and divide it into $d_1$ horizontal section, then one divides each of these sections into $d_2$ horizontal sections, and each of these sections one divide into $d_3$ horizontal sections and one does this until one reaches $d_k$, where $d_k$ is the last horizontal divisor and $d_k$ satisfies the condition that $d_1 \cdot d_2 \cdot \ldots \cdot d_k \neq d_{k+1} \cdot d_{k+2} \cdot \ldots \cdot d_n$. Also $k$ is the largest
number for which the above equation is satisfied. Then one takes this rectangle and divide it into $d_{k+1}$ vertical sections each of which are to be divided into $d_{k+2}$ vertical sections which in turn are to be divided into $d_{k+3}$ vertical sections and so on until one reaches $d_n$ which is the last vertical divisor. Then one assigns the vector $(x_1, x_2, \ldots, x_k), x_i \in \{0, 1, \ldots, d_i\}, i \leq k$ to the horizontal part of the rectangle or the rows of the rectangle, and assigns the vector $(x_{k+1}, x_{k+2}, \ldots, x_n), x_i \in \{0, 1, \ldots, d_i\}, v < i \leq n$ to the vertical part of the rectangle or the columns of the rectangle [4].

A drawing of the above described Generalized Logic Diagram representing an event space is shown in Figure 1.1. As one can see, for a large event space this gets hard to do.
Figure 1.1  Generalized Logical Diagram (GLD) representing $E(d_1, d_2, d_3, d_4)$. 

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$d_2^{-1}$</th>
<th>$d_3^{-1}$</th>
<th>$d_4^{-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$X_3$</td>
<td>$X_4$</td>
<td>$d_1^{-1}$</td>
<td>$d_2^{-1}$</td>
<td>$d_3^{-1}$</td>
</tr>
</tbody>
</table>
2. SYNTHESIS OF COVERS

In this chapter we will introduce the concept of a cover of one event set against another event set. The concept of a cover is fundamental for the synthesis and minimization of $\mathit{VL}_1$ formulas.

One may ask, "Why minimize a $\mathit{VL}_1$ formula?" There are several reasons. $\mathit{VL}_1$ formulas can be applied as descriptors of object classes. Therefore, the shorter and less expensive these $\mathit{VL}_1$ formulas are the faster and cheaper it will be to recognize an object from an object class as described by the $\mathit{VL}_1$ formula. For example, suppose one could find out if a person had a certain disease by either checking their blood chemistry or by performing an operation on that person. Obviously it would be less expensive and much quicker to check the person's blood chemistry and we would want this method of checking if the person had the certain disease as opposed to having him operated. Therefore we not only want to minimize a formula, we also want to specify the manner in which we would like it minimized or synthesized. That is why we have the optimal formula $\mathit{DVL}_1$ under functional $A = \langle a\text{-list}, t\text{-list} \rangle$ [5] of formula $V_3$ where:

- **a-list**: called attribute list is a vector $a = (a_1, a_2, ..., a_t)$, where the $a_i$ denote single or many-valued attributes used to characterize $\mathit{DVL}_1$ formula (e.g., number of terms, of selectors, total number of
variables involved, the weight of
each variable, etc.)

\( \tau \)-list: called tolerance list, is a vector

\[ \tau = (\tau_1, \tau_2, \ldots, \tau_l) \] where \( 0 \leq \tau_i \leq 1 \),
i=1,2,...,l and \( \tau_i \) is called tolerance
for attribute \( a_i \).

The optimal formulas are constructed by constructing a set
of optimal covers of one event set against another event set. Thus
in order to describe a process of optimizing \( \text{VL}_1 \) formulas we have to
describe the concept of a cover \( \text{CV}(E_1|E_2) \). Complexes are the building
blocks of a cover. A complex is a \( \text{VL}_1 \) formula which describes a set
of events.

A set \( \mathcal{L} \subseteq \mathcal{A} \) is called universe of complexes, if it satisfies

(i) Coverability criterion

\[ \forall e \in \mathcal{N} \quad \exists C \in \mathcal{L}, \quad e \in C \]

(ii) Separability criterion

\[ \forall e_1, e_2 \in \mathcal{N}, \quad \exists C_1, C_2 \in \mathcal{L}, \quad e_1 \in C_1 \text{ and } e_2 \in C_2 \]

Definition 1: A cover \( \text{CV}(E_1|E_2) \), (where \( E_1 \) and \( E_2 \) are any two disjoint
subsets of \( \mathcal{N} \)), of sets \( E_1 \) against \( E_2 \) is defined as a set of complexes,
\( \{C_i\}_{i \in I} \) such that

\[ E_1 \subseteq \bigcup_{i \in I} C_i \subseteq \mathcal{N} \setminus E_2 \]

A complex \( C \) is formed by a sequence of selectors which describes
the literal \( X \). There are two types of literals. They are:
1. Cartesian literal \( [X^a_i] \) is a set of all events \( e = (x_1, x_2, \ldots, x_i, \ldots, x_n) \in E \) such that the value of \( x_i \) is an element of \( \alpha_i \), \( \alpha_i \in \mathcal{D}_i \).

2. Interval complex \( [a^b_i] \) is a set of all events \( e \) such that the value of \( X_i \) is between \( a \) and \( b \) inclusively.

Following are a set of rules for obtaining the product of two literals \( x_i \) and \( x_j \) if \( i \neq j \) then we obtain \( x_i x_j \). Otherwise, if \( i = j \) then we have the following possibilities:

1. \( a^b_i = c^d_i \) which means \( a = c \) and \( b = d \) and the product is either one.

2. \( a^b_i \subseteq c^d_i \) which implies that either \( d > a > c \) and \( b \leq d \) or \( a \geq c \) and \( c < b < d \). The product is \( a^b_i \).

3. \( a^b_i \) is not completely contained in \( c^d_i \) which means either \( d > a > c \) and \( b > d \) or \( a < c \) and \( d < b < c \) the result is \( a^d_i \) and \( c^b_i \) respectively.

Let \( C \) be the universe of complexes.

**Definition 2:** The root, \( \sqrt{E} \) (\( \overline{\sqrt{E}} \)), of \( E \) is the set of all maximal complexes included in \( E \)

\[
\sqrt{E} = \{ C \in C \mid C \subseteq E \text{ and } \exists \ C' \subseteq E, \ C \subseteq C' \}
\]

**Definition 3:** The stars \( G(E_1 \mid E_2) \) of \( E_1 \) against \( E_2 \) is defined as

\[
G(E_1 \mid E_2) = \{ C \mid C \in \overline{\sqrt{E_2}} \text{ and } C \cap E_1 \neq \emptyset \}
\]

\( \overline{E}_2 \) is the negation of \( E_2 \).

**Definition 4:** The extension \( E_1 \rightarrow E_2 \) of the event set \( E_1 \) against the event set \( E_2 \) is defined as
E_1 \rightarrow E_2 = U(C | C \in Q(E_1 \setminus E_2))

MS -- is maxstar, which is the maximum number of complexes we want generated before trimming.

CS -- is cutstar, which, when we reach MS number of complexes, we trim down to CS number of best complexes according to \( A = \langle a\text{-list}, \tau\text{-list} \rangle \).

Given a cover \( CV_1 \) defined as \( CV_1 = \bigcup_i C_i \) where \( C_i \) is a complex, one can reduce the number of complexes in \( CV_1 \) as follows if \( C_k \subseteq C_j \), which means every literal of \( C_k \) is equal to or contained in the literals of \( C_j \). Then we can omit \( C_k \) since it is completely contained in \( C_j \).

To synthesize a cover \( CV_1 \), we use here a slightly modified version of the algorithm \( A^q \) described in article [5]. Minimal and quasi-minimal covers are best and approximately best cover.

\[
\begin{align*}
\text{Input } & E, E_1, CV_1^U, MS, CS, A \\
M^q & = \emptyset \quad \text{Last value of } M^q \text{ is the quasi-minimal cover.} \\
\text{Select a complex } C \text{ from } CV_1^U \quad \text{is } CV_1 = \emptyset \text{ END} \\
\text{Generate } & G(C | E \setminus CV_1^U) \\
\text{with restriction } & \langle MS, CS \rangle \\
\text{Select } L^q & \text{ from } G(C | E \setminus CV_1^U) \text{ according to } A \\
M^q & = M^q \cup \{L^q\}, CV = CV_1 \setminus L^q \\
E \setminus CV & = E \setminus CV \cup \{L^q\}
\end{align*}
\]
The description of the algorithm is as follows. Given a cover \( CV_1 \) and given the event space \( E \), one takes a complex from the cover \( CV_1 \) and extend it into \( E \setminus CV^U \), i.e., \( CV_1 \rightarrow E \setminus CV^U \), thus generating a star for the complex chosen \( C_i \). The star is of course generated with the restrictions maxstar and cutstar. Then when one finishes generating the star, one picks the best complex that covers \( C_i \) completely and does so according to the attribute \( A \). Thus one has obtained the first complex \( L^q \) of the quasi-minimal cover \( M^q \) of \( CV_1 \). Now removing \( C_i \) from \( CV_1 \) and any other complexes in \( CV_1 \) completely covered by \( L^q \), thus places \( L^q \) as part of \( E \setminus CV^U \) since one no longer cares to consider that complex. Then one repeats the above procedure until one has eliminated all the complexes of \( CV_1 \) and thus have \( M^q \) as the quasi-minimal cover for \( CV \).
3. MINIMIZATION ALGORITHMS

The first section of this chapter discusses the algorithm used in the program to minimize \( \text{VL}_1 \) formulas. The second section discusses iterative minimization.

3.1 Algorithm for Formula Minimization

There are several different types and modes for minimal covers. When asking for a minimal cover one has to specify the kind of minimal cover one wants. For different problems one may require different ways of covering one's classes. That is why there are two ways of minimizing a cover. One is without generalization, which means minimizing the cover but without extending its coverage or try to enlarge its range. Thus given a cover that covers \( n \) events one minimizes the cover without including any other event or events except for the \( n \) original events covered. The second method is with generalization, which means that if one can better minimize the cover given by including other events not included in the original \( n \) cover one should do so.

Now for both of these types of covers there are three different modes of coverage. These are \( \text{VL} \), \( \text{DC} \), and \( \text{IC} \). For the covers without generalization, \( \text{DC} \) and \( \text{IC} \) turn out to be the same thing. Following are the definitions of the different modes of coverage.

\( \text{VL} \): the formula is to be a \( \text{DVL}_1 \) formula without any restriction (\( \text{DVL}_1 \) are simple disjunctive \( \text{VL}_1 \) formulas).
DC: the formula is to be a DVL₁ formula with the property that the product of any two terms with different constants is zero ('disjoint covers')

IC: the formula is to be a DVL₁ formula with the property that the product of any two terms with different constants may be nonempty, but must not be satisfied by any event from the sets \( p^k \) (\( k=0,1,\ldots,\phi \)) which are specified in the input data ('intersecting covers').

Now that the description on the type and modes of the minimal covers has been made, there are also two different types of complexes within the minimal cover. They are inclusive, i.e., the intersection of any two complexes may be non-empty, and exclusive, i.e., the intersection of any two complexes is empty.

The algorithm \( A^q [1] \) shown in the previous chapter is for exclusive complexes, since we set \( E \setminus CV^u = E \setminus CV^u \cup L^q \). For inclusive complexes we simply leave \( E \) unchanged throughout the star generating process.

Following are the algorithms used for obtaining the minimal cover with and without generalization and using the three different modes VL, DC, and IC. These algorithms were derived from a more general algorithm constructed by Professor R. S. Michalski.

Before actually showing the algorithm used in the program, there are a few notations to be defined.

\( CV_i \) -- is the \( i \)th class or cover

\( CV = \{C_1, C_2, \ldots, C_n\} \) where \( C_i \) is the \( i \)th complex. So \( CV^u = C_1 \cup C_2 \cup C_3 \cup \ldots \cup C_n \) and \( CV^u = C_1 \cup C_2 \cup \ldots \cup C_n \)
\( C_1 \cap C_2 \cap C_3 \cap \ldots \cap C_n \) which after multiplication gives the
following result \( CV = C'_1 \cup C'_2 \cup \ldots \cup C'_m \) so one gets
\( CV = \{C'_1, C'_2, \ldots, C'_m\} \) where \( C'_n \) is the \( n^{th} \) complex.

\( \overline{CV} \) - is the negation of the \( i^{th} \) class or cover

\( D_i \) - represents the cover against which we are covering \( CV_i \)

\( Q_i \) - is the quasi-minimal cover obtained for \( CV_i \)

\( Q(CV_i/D_i) \) - is the algorithm \( A^q \) by which one obtains the quasi-
minimal cover \( CV_i \) against \( D_i \)

\( X \cap Y \) - is the intersection of \( X \) and \( Y \)

\( X \cup Y \) - is the union of \( X \) and \( Y \)

\( X \setminus Y \) - is all of \( X \) excluding \( Y \).

Given \( n \) classes one has the algorithm shown in Figure 3.1
for obtaining the minimal cover without generalization and in VL mode.

The last class is not covered since in VL mode that last
class is the default class.

To familiarize the reader with the notation and flowchart
of the algorithm, the following is a verbal description of the first
algorithm.

First one negates the \( n^{th} \) class (which is the first class of the
input classes) to obtain \( D_n \). Then one uses the algorithm \( A^q[1] \) to
obtain the quasi-minimal cover \( Q_n \) which is the cover of \( CV_n \) against \( D_n \).
Next one intersects \( D_n \) with the negation of \( CV_{n-1} \) (i.e., the \( n-1^{st} \)
class) to obtain \( D_{n-1} \). Now one gets \( Q_{n-1} \) as the quasi-minimal cover of
\( CV_{n-1} \) against \( D_{n-1} \). One follows this general pattern until one reaches
class two, i.e. \( CV_2 \), which is the last class one obtains a quasi-minimal
form for. That leaves \( CV_1 \) to be the default class as required by the VL
mode.
Figure 3.1 The algorithm without generalization for VL mode.
Figure 3.2 shows the algorithm to obtain the minimal cover of n classes without generalization and in DC mode which is the same for IC when there are no generalizations.

If one wants a minimal cover with generalization and in VL mode for n classes, one has the algorithm shown in Figure 3.3.

The last class is not covered since in VL mode that last class is the default class.

Figure 3.4 shows the algorithm that will give the quasi-minimal cover of n classes with generalization and in DC mode.

Figure 3.5 shows the last algorithm which is to obtain quasi-minimal covers for n classes in IC mode and with generalization.

3.2 Iterative Applications

Iterative minimization of $\text{VL}_1$ formulas is a procedure by which after obtaining the quasi-minimal covers $E'_1$, $i=1,2,\ldots,n$ from the n input classes $E_i$, $i=1,2,\ldots,n$ one uses these quasi-minimal covers as one's input covers and goes through the minimization algorithm again to see if one obtains a better result. The program developed for this thesis has this above feature.

The reason for having the iteration feature is to see if one can obtain an even more minimal cover than the first one obtained. The iteration feature was beyond the scope of this thesis. That is why only the simplest form of the iterative feature has been applied, which means that with a little research one might find a better way to use this iterative feature. For example, one may collect data as one first obtains the minimal covers. Such data is how many stars one generated to obtain each complex in the minimal cover will enable one
Figure 3.2 The algorithm without generalization for DC and IC mode.
Figure 3.3 The algorithm with generalization for VL mode.
Figure 3.4 The algorithm with generalization for DC mode.
Figure 3.5 The algorithm with generalization for IC mode.
to start with the complex that generated the fewest stars so as to speed up the process of picking the best one. This type of information and other information may be used to improve the iteration feature.
4. IMPLEMENTATION OF THE ALGORITHMS

In the first section of this chapter discusses the method in which the algorithm described in the previous chapter for $VL_1$ formula minimization was implemented, the data structure for the program and the way different $VL_1$ operations were performed. The second section is a guide on how to use the program, how to specify the input and how to interpret the output from the program.

4.1 Implementation

Each complex is stored as a sequence of binary bits, i.e., as a binary string, whose length is the sum of the levels of all the variables in the event space, thus having L.B.S. (length of binary string). The identical representation for complexes as was used in the program AQVAL/l (AQ7) is used here.

$$\text{L.B.S.} = \sum_{i=1}^{n} d_i$$

where

$$E = D_1 \times D_2 \times \ldots \times D_n$$

and

$$d_i = c(D_i)$$

Thus obtaining the following data structure where BPTR is the back pointer (i.e., the pointer which points to the following complex) and FPTR is the front pointer (i.e., the pointer which points to the previous complex) and their value is NULL if there are no complexes to point to.
To represent a cover consisting of more than one complex one simply links several of the structures shown in Figure 4.1 using FPTR and BPTR to know which complexes belong to a certain cover or class.

<table>
<thead>
<tr>
<th>BINARY STRING REPRESENTING THE COMPLEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPTR</td>
</tr>
</tbody>
</table>

Figure 4.1 Data structure for the representation of a complex.

Following is an example of exactly how a complex is represented with a binary string. Given the event space \( E = (3,4,5,2,4) \) we get L.B.S. = 18 so we obtain the following binary string of eighteen bits in length.

The first variable is represented by the first three bits in Figure 4.2 which represents the values zero, one and two respectively.

\[ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \]

Figure 4.2 Initial binary bit string obtained to later represent a complex in the event space \( E(3,4,5,2,4) \).
So if the first variable is equal to one (i.e., \([x_1=1]\)) one places a one for the bit that represents the value one (i.e., \(1\)). The second variable is represented by the next four bits, the next five bits represent the third variable, the two bits after that represent the fourth variable and the fifth variable is represented by the last four bits. Figure 4.3 shows which bits are assigned to which variable and the value each bit represents for its variable.

![Figure 4.3](image)

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 4.3 Binary bit string and how variable and values are associated to it.

A '1' is placed wherever that value is associated with the variable. '1's are placed for all the values of a variable if that variable is not in a complex. Thus obtaining the following complex

\([x_1=0,2][x_3=1,2,3][x_4=1][x_5=0]\)

represented by the following binary string

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>1111</td>
<td>01110</td>
<td>01</td>
<td>1000</td>
</tr>
</tbody>
</table>
where one has all ones for $x_2$ since $x_2$ is not in the complex which means that the complex is true for all values of $x_2$ given the above conditions for the other four variables.

There are really only three essential operations used. They are multiplication of complexes (also known as the intersection of complexes), the negation of complexes, and the containment of a complex (i.e., the operation to check whether a complex is the proper subset of another complex).

The operation multiplication or intersection of two complexes is simply done by a bit by bit AND of the binary bit strings representing these complexes.

**Example I:**

Consider the event space $E = (2,3,2)$ and the complexes $L_1 = \{x_1=0\} \{x_3=1\}$ and $L_2 = \{x_1=0\} \{x_2=2\}$. Multiplying them one obtains the complex $L_3 = \{x_1=0\} \{x_2=2\} \{x_3=1\}$ as shown in Figure 4.4.

![Figure 4.4 GLD of Example I](image-url)
The binary bit string representation of $L_1$ and $L_2$ are 1011101 and 1000111 respectively. When by ANDing these two binary strings, one obtains 1000101 which is the binary bit string representation complex $L_3$.

Since multiplication is the intersection of complexes, it might happen that the intersection is empty. When this happens, one finds that all the values of one or more variables have a zero in the binary string. Example II shows this case.

Example II:

Given the same event space as in the last example and our two complexes being $L_1 = \{x_1=1\} \cap \{x_3=0\}$ and $L_2 = \{x_1=0\} \cap \{x_2=2\}$ one notices that the intersection is the empty set as shown in Figure 4.5.

![Figure 4.5 GLD of Example II](image)

The binary string for $L_1$ and $L_2$ are 0111100 and 1000111 respectively and if one bit by bit AND's these two strings one gets 0000110. As can be seen, there are no values for $x_1$ so one gets the empty set as one's answer.
When given two complexes $L_1$ and $L_2$, if one wants to check if $L_1$ is a proper subset of $L_2$ one simply applies a bit by bit AND of the two binary strings representing $L_1$ and $L_2$, and check if the binary string representing $L_1$ is the same as the binary string of the result of $L_1 \text{ AND } L_2$. If it is, then $L_1$ is a proper subset of $L_2$; otherwise it is not.

Last but not least is the manner in which a complex is negated. Suppose the event space is $E = (2,4,3,3)$ and let the complex to be negated be $L_1 = \{x_1=0\} \{x_2=1,3\} \{x_4=2\}$ the negation of $L_1$ is the set of three complexes $L_1 \lor L_2 \lor L_3$ which are $\{x_1=1\}, \{x_2=0,2\}$ and $\{x_4=0,1\}$ respectively.

When negating a complex one creates a set of binary strings, one for each variable in the complex to be negated, that does not have all ones in the binary string of the complex to be negated. These binary strings contain all ones except where the variables they are matched up with have ones in the binary string of the complex being negated.

**Example III:**

The binary string of $L_1$ is 100101111001. So, for the negation of $L_1$ one gets three binary strings since $x_3$ has all ones in the binary string representing $L_1$. The binary string matched up or associated with $x_1$ is 01111111111, the binary string for $x_2$ is 11101011111, and for $x_4$ we have 11111111110. As you see, each variable in its binary string has a zero where there was a one for it in the binary string of the complex to be negated.

Now for a simplified overview of how one obtains the minimal cover.
Take the first algorithm given in Chapter 3 of this thesis. Given a cover \( CV = L_1 \lor L_2 \lor \ldots \lor L_n \) where \( L_i \), \( i=1,2,\ldots,n \) are complexes, we negate it to obtain \( \overline{CV} \) which is the same as \( \overline{L_1} \cdot \overline{L_2} \cdot \overline{L_3} \cdot \ldots \cdot \overline{L_n} \) using a routine that will negate any complex using the method explained above. Then multiplying these complexes using the method shown for multiplication of complexes, and absorbing them using the containment operation given earlier so as not to have any redundant complexes. After these operations, one comes out with \( \overline{CV} = L'_1 \lor L'_2 \lor \ldots \lor L'_m \) where \( L'_i \), \( i=1,2,\ldots,m \) are complexes. Now by simply using the star generation for \( CV \) against \( \overline{CV} \) the quasi-minimal form for the cover \( CV \) is obtained.

4.2 User's Guide

AQ9 is a PL/l implementation of an algorithm for the minimization of \( VL_1 \) formulas. This program will synthesize quasi-optimal formulas given in \( DVL_1 \) form. One may specify for the quasi-minimal complexes to be inclusive or exclusive of each other. One may also ask for dependent or independent covers. There are three cover modes which are \( VL \), \( DC \) and \( IC \). They are described below.

Program Input

The format for the input to AQ9 is as follows:

1. The variables LTRACE, QTRACE, GNNC, ITERA, GONG, INOREX, MODE, MAXSTR, and CUTSTR are read in PL/l DATA format. Default values are given within parentheses.

LTRACE('O'B): if 'l'B is specified a full trace of the synthesis of covers will be printed. This feature is for debugging purposes. (Users should not use it.)
QTRACE('O'B): if '1'B is specified a quick (short) trace of
the synthesis of covers will be printed. Users
should not use this unless necessary.

GNNC(200): this is an array parameter which should be big
if the event space is large.

ITERA(1): this specifies the number of iterations wanted.
Where after the first iteration one takes the output
and uses it as their input for the subsequent iteration
in an attempt to improve our result.

GONG(GN): this is the type of cover to be generated. GN
means a cover without generalization, and GE is
a cover with generalization.

INOREX(EX): specifies type of complex one wants EX for exclusive
(i.e., any two complexes in the same class when
intersected give the empty set) and IN for inclusive
(i.e., any two complexes in the same class when
intersected may be non-empty).

MODE(VL): there are three modes. They are:
VL the formula is to be a $DVL_1^*$ formula without
any restrictions (i.e., any two complexes
in the same class when intersected gives the
empty set).

DC the formula is to be a $DVL_1$ formula with the
property that the product of any two terms
with different constants is 0 ('disjoint covers').

$*$where $DVL_1$ formulas are disjunctive simple $VL_1$ formulas.
IC the formula is to be a DVL₁ formula with the property that the product of any two terms with different constants may be non-empty, but must not be satisfied by any event from sets $F^k$ (k=0,1,...,$d$) which are specified in the input data ('intersecting covers').

MAXSTR, CUTSTR(50000 each): if an intermediate star contains more than MAXSTR complexes, it is cut to CUTSTR best complexes (according to the attributes specified).

2. The number of variables and the number of classes, which are read in PL/1 LIST format.

3. The variables NCRIT, CLIST, and TLIST are read in PL/1 LIST format.

   NCRIT: specifies the number of attributes.

   CLIST: is the list of attributes.

   TLIST: is the list of tolerance of each attribute.

   So far 4 attributes are available to select from.

   ATTRIBUTE#1. The number of complexes completely covered by the newly selected complex (the more the better).

   ATTRIBUTE#2. The number of variables in the newly selected complex (the less the better).

   ATTRIBUTE#3. The number of states (i.e., levels) for each variable in the newly selected complex (the more the better).

   ATTRIBUTE#4. The sum of the weight of all the variables in the newly selected complex (the less the better).

   For attribute#4 one has to specify the weight for each variable. This is read in PL/1 LIST format.
4. Specifies the number of states (i.e., levels) of the variables, (the event space). This is read in PL/l LIST format.

5. The variable TYPE is read in PL/l DATA format. Default value is given within parentheses.

\[ \text{TYPE(EVENTS)}: \text{this specifies the type of data to be given.} \]
\[ \text{TYPE} = \text{'BOTH'}; \text{implies events and complexes in that order. \ TYPE} = \text{'COMPLEXES', \ implies complexes only and \ TYPE} = \text{'EVENTS', \ implies events only.} \]

This is done for each class.

6a. If TYPE is specified as BOTH, then the next piece of data is the number of events and the number of events to be converted to complexes. These are read in PL/l LIST format. Since this program is mainly oriented to minimize complexes, number of events to be converted to complexes, allows one to specify how many events are to be actually considered as complexes in this cover. The program will check if any of the events are already included in the complexes one has given. Then it will check how many events remain, if that is larger than the number of events to be converted to complexes, it will not convert any of the events into complexes, otherwise it will convert them all.

After one enters the events, which are read in PL/l LIST format, then one inputs the number of complexes followed by the number of selectors for each complex. This is also read in PL/l LIST format, and then one inputs one's complexes.

6b. If TYPE is specified as COMPLEXES, then one inputs the number of complexes followed by the number of selectors for each complex. This is read in PL/l LIST format. All this is then followed by your complexes.
6c. If TYPE is unspecified or specified as EVENTS, then one just
number of events read in PL/1 LIST format, followed
by the events also read in PL/1 LIST format.

JCL for running AQ9
/*ID (id number and name)
/*ID (codeword)
/*ID (time, lines, ioreq, region, etc.)
//JOBLIB DD DSN = USER.P2123.COMEVE, DISP = SHR
// EXEC PROGPLIX, PROG = AQ9, REGION = 250K
//GO.SYSIN DD *
      (Program input)
/* (last symbol)

Following are two examples of the format in which the input
data should be and the type of output you obtain from it.

Restrictions
The main restriction is the size of the event space in that
the larger the event space the longer it takes. There are two things
that make the event space large:

1. the number of variables
2. the cardinality of the variables.
Due to lack of symbols, [ ] are replaced by ( ).

**Sample Input**

**Actual Input**

<table>
<thead>
<tr>
<th>(1)</th>
<th>LTRACE='O'B QTRACE='O'B GNEC=300</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ITERA=1 GONG='GE' INOREX='IN'</td>
</tr>
<tr>
<td></td>
<td>MODE='VL' MAXSTR=3 CUTSTR=1;</td>
</tr>
</tbody>
</table>

| (2) | 2 2 |
|     |     |

| (3) | 1 1 |
|     |     |

| (4) | 6 6 |
|     |     |

| (5) | TYPE='COMPLEX'; |
|     |                 |

| (6b) | 6 2 2 2 2 2 2   |
|      |                 |

**Information about Input**

(number of variables and number of classes)

(number of attributes and attribute number)

(tolerance of above attribute number)

(number of states of the above variables)

(number of complexes followed by the number of selectors for each complex)

(first complex with two selectors)

(second complex with two selectors)

(third complex with two selectors)

(fourth complex with two selectors)

(fifth complex with two selectors)
(X1=5)  
(X2=5)  

(5) TYPE='COMPLEX';

(6b) 6 2 2 2 2 2 2

(X1=0)  
(X2=1,2)  

(X1=1)  
(X2=0,2,3,4,5)  

(X1=2)  
(X2=0,1,3,4)  

(X1=3)  
(X2=1,2,4)  

(X1=4)  
(X2=1,2,3,5)  

(X1=5)  
(X2=1,4)  

(sixth complex with two selectors)

(number of complexes followed by the number of selectors for each complex)

(first complex with two selectors)

(second complex with two selectors)

(third complex with two selectors)

(fourth complex with two selectors)

(fifth complex with two selectors)

(sixth complex with two selectors)

With this input you obtain the output shown on the following page.
Sample Output

EXAMPLE #1

Actual Output

NUMBER OF VARIABLES= 2
NUMBER OF CLASSES 2
MODE IS VL
MAXSTAR IS 3 CUTSTAR IS 1
CLIST= 1
TLIST= 0.00
NUMBER OF LEVELS FOR EACH VARIABLE: 6 6

THE NUMBER OF COMPLEXES FOR CLASS 0 ARE 6
THE COMPLEXES OF E(1) ARE:

(XL=0)(X2=0) V (XL=1)(X2=1) V (XL=2)(X2=2) V (XL=3)(X2=3) V (XL=4)(X2=4) V (XL=5)(X2=5)

THE NUMBER OF COMPLEXES FOR CLASS 1 ARE 6
THE COMPLEXES OF E(1) ARE:

(XL=0)(X2=1,2) V (XL=1)(X2=0,2,3,4,5) V (XL=2)(X2=0,1,3,4) V (XL=3)(X2=1,2,4) V (XL=4)(X2=1,2,3,5) V (XL=5)(X2=1,4)

***ITERATION #1***

THE COMPLEXES OF Q(0) ARE:

(XL=0,3,5)(X2=0,3,5) V (XL=1)(X2=1) V (XL=2,5)(X2=2,5) V (XL=0,4)(X2=0,4) V

****NORMAL TERMINATION****

EXECUTION TIME = 157 centiseconds
EXAMPLE #1

Figure 4.6 The GLD diagram of Example I.
Sample Input

Example #2

Actual Input
(1) LTRACE='O'B QTRACE='O'B GNNC=300
ITERA=1 GONG='GN' INOREX='EX'
MODE='DC' MAXSTR=3 CUTSTR=1;

(2) 2 2 (number of variables and number of classes)

(3) 1 1 (number of attributes and attribute number)
0.00 (tolerance of above attribute number)

(4) 6 6 (number of states of the above variables)

(5) TYPE='BOTH';

(6a) 8 5 (number of events and number of events to be converted to complexes)
0 4
5 0
2 5
0 3
5 2
4 0
5 3
0 5
2 2 2 (the events)

(number of complexes followed by the number of selectors for each complex)
(X1=3) \} \quad \text{(first complex with two selectors)}
(X2=0,5) \}
(X1=5) \}
(X2=0,2,3) \}

(5) \text{TYPE='EVENTS';}

(6a) \text{6} \quad \text{(number of events)}
0 0
1 1
2 2
3 3
4 4
5 5

(\text{the events})

The following pages contain the output to the above input.
Sample Output

EXAMPLE #2

Actual Output

NUMBER OF VARIABLES = 2
NUMBER OF CLASSES = 2
MODE IS DC
MAXSTAR IS 3 CUTSTAR IS 1
CLIST = 1
TLIST = 0.00
NUMBER OF LEVELS FOR EACH VARIABLE: 6 6

THERE ARE 8 EVENTS FOR CLASS 1
THE NUMBER OF COMPLEXES FOR CLASS 1 ARE 2
THE NUMBER OF EVENTS IS 5
THE COMPLEXES OF E(1) ARE:

(X1=3)(X2=0,5) \lor (X1=5)(X2=0,2,3) \lor
(X1=0)(X2=4) \lor (X1=2)(X2=5) \lor
(X1=0)(X2=3) \lor (X1=4)(X2=0) \lor
(X1=0)(X2=5)

THERE ARE 6 EVENTS FOR CLASS 0
THE COMPLEXES OF E(0) ARE:

(X1=0)(X2=0) \lor (X1=1)(X2=1) \lor (X1=2)(X2=2) \lor
(X1=3)(X2=3) \lor (X1=4)(X2=4) \lor
(X1=5)(X2=5)

***ITERATION #1***

THE COMPLEXES OF Q(1) ARE:

(X1=3)(X2=0,5) \lor (X1=5)(X2=0,2,3) \lor
(X1=0)(X2=3,4,5) \lor
(X1=2)(X2=5) \lor (X1=4)(X2=0)

(this is the output giving the quasi-minimal cover of the first class after one iteration in DC mode)
THE COMPLEXES OF D(1) ARE:

\[(X_l=1) \lor (X_l=2)(X_2=0,1,2,3,4) \lor (X_l=4)(X_2=1,2,3,4,5) \lor (X_l=3)(X_2=1,2,3,4) \lor (X_l=5)(X_2=1,4,5) \lor (X_l=0)(X_2=0,1,2)\]

(this is the output giving the quasi-minimal cover of the negation of the first class)

THE COMPLEXES OF Q(0) ARE:

\[(X_l=1)(X_2=1) \lor (X_l=2)(X_2=2) \lor (X_l=4)(X_2=4) \lor (X_l=3)(X_2=3) \lor (X_l=5)(X_2=5) \lor (X_l=0)(X_2=0)\]

(this is the output giving the quasi-minimal cover of the second class after one iteration in DC mode)

THE COMPLEXES OF D(0) ARE:

\[(X_l=1)(X_2=0,2,3,4,5) \lor (X_l=2)(X_2=0,1,3,4,5) \lor (X_l=4)(X_2=0,1,2,3,5) \lor (X_l=3)(X_2=0,1,2,4,5) \lor (X_l=5)(X_2=0,1,2,3,4) \lor (X_l=0)(X_2=1,2,3,4,5)\]

(this is the output giving the quasi-minimal cover of the negation of the second class)

*****NORMAL TERMINATION*****

EXECUTION TIME = 1,304 centiseconds
EXAMPLE #2

Figure 4.7 The GLD diagram of Example II.

1 -- class one
0 -- class zero
This chapter consists of two examples to demonstrate the ability and use of the program discussed in the previous chapters.

**EXAMPLE #1**

Here are three classes and their descriptions are as follows. These classes are from the event space: \( E = (3,4,4,3) \).

Class one = \( L_{11} \cup L_{12} \cup L_{13} \)

where

\[
L_{11} = \{x_1=1, 2\} \{x_3=2\} \{x_4=0, 1\}
\]

\[
L_{12} = \{x_1=1, 2\} \{x_2=0, 1, 2\} \{x_3=2\} \{x_4=1, 2\}
\]

\[
L_{13} = \{x_1=1, 2\} \{x_2=2, 3\} \{x_3=2\}
\]

Class two = \( L_{21} \cup L_{22} \cup L_{23} \)

where

\[
L_{21} = \{x_1=0\} \{x_2=0, 1\} \{x_3=1, 3\}
\]

\[
L_{22} = \{x_1=0\} \{x_3=0, 2\}
\]

\[
L_{23} = \{x_1=0\} \{x_2=1, 2, 3\}
\]

Class three = \( L_{31} \cup L_{32} \cup L_{33} \cup L_{34} \cup L_{35} \)

where

\[
L_{31} = \{x_1=2\} \{x_2=0\} \{x_3=0\}
\]

\[
L_{32} = \{x_1=2\} \{x_2=1\} \{x_3=0\}
\]

\[
L_{33} = \{x_1=2\} \{x_2=2\} \{x_3=0\}
\]

\[
L_{34} = \{x_1=2\} \{x_2=3\} \{x_3=0\}
\]

\[
L_{35} = \{x_1=2\} \{x_3=0\} \{x_4=1\}
\]
Figure 5.1 gives a GLD representation of these above three classes. As one can see, the complexes within a class are redundant in that they almost all intersect each other. This data was run through the minimization program. The type of cover was with generalizations and the mode was disjoint covers (DC). The result obtained was as follows:

Class one = \( L'_{11} = \{x_1=1,2\} \{x_3=1,2,3\} \)
Class two = \( L'_{21} = \{x_1=0\} \)
Class three = \( L'_{31} = \{x_1=1,2\} \{x_3=0\} \)

The result is shown in the form of a GLD in Figure 5.2.

As one may have noticed, class one was expanded and simplified in that only two variables were sufficient to describe or cover it. Class two was simplified so as to only require one variable to describe it. It was not expanded since if it had been its descriptor, it would have been more complicated. Class three was simplified and expanded.

**EXAMPLE #2**

The problem described in this example was suggested by R. S. Michalski. The problem is to color the map of the United States in such a manner so as not to have bordering states of the same color (i.e., two states next to each other are not supposed to be of the same color). States that were kitty-cornered were allowed to be the same color. The whole map was not colored all at once since the event space would be rather large and thus take a lot of computer time and money. Therefore, the map was colored in parts of about ten to eleven states at a time, which took anywhere from ten to fifty seconds to execute. The type of cover asked for was with generalization and the mode was VL. The maxstar was three and cutstar was one. There
Figure 5.1 The GLD representation of the input covers in Example 1.
Figure 5.2  The GLD representation of the resulting output covers in Example I.
are two classes, the first one has the states to be colored and the second one has all the states that border on each state of the first class and it does not include any state that is not in the first class. The results obtained are several groups (between three or four) of states that can be the same color.

It took seven runs to color all of the map of the U.S.A. The reason for this was that one could not just take five sets of ten states and color them since one could get two states that border on each other being the same color because they were in separate sets. So, one had to color one set at a time. Therefore, the next set would contain some states from the previous sets. It would contain all the states that were already colored that bordered on the states to be colored. Figure 5.3 (a) through (g) show this process of coloring the map of the U.S.A., with Figure 5.3(g) being the end result.

In this example, one has two variables $x_1$ and $x_2$, denoting the state and the bordering states that are not supposed to be the same color. For the first run one has

0 -- Washington 5 -- Arizona
1 -- Oregon 6 -- Montana
2 -- California 7 -- Wyoming
3 -- Idaho 8 -- Utah
4 -- Nevada 9 -- Colorado

The two classes were

$[x_1=0][x_2=0] \lor [x_1=1][x_2=1] \lor \ldots \lor [x_1=9][x_2=9]$

and

$[x_1=0][x_2=1,3] \lor [x_1=1][x_2=0,2,3,4] \lor [x_1=2][x_2=1,4,5] \lor [x_1=3][x_2=0,1,4,6,7,8] \lor [x_1=4][x_2=1,2,3,5,8] \lor [x_1=5][x_2=2,4,8] \lor [x_1=6][x_2=3,7] \lor [x_1=7][x_2=3,6,8,9] \lor [x_1=8][x_2=3,4,5,7,9] \lor [x_1=9][x_2=7,8]$
The first class specifies the states to be colored and the second class specifies which of these states to be colored cannot be the same color. The program was run in VL mode with generalization. The result obtained was

\[(x_1=0, 2, 6, 8) \lor (x_2=0, 2, 6, 8) \lor (x_1=1, 5, 7) \lor (x_2=1, 5, 7) \lor (x_1=3, 9) \lor (x_2=2, 3, 9) \lor (x_1=4) \lor (x_2=0, 4, 6, 7, 9)\]

This means that states 0, 2, 6, and 8 can be the same color, say color one. States 1, 5, and 7 are color two. Color three is for states 3 and 9, and state four will be color four (as shown in Figure 5.3(a)).

The next set of states was as follows

0 — Montana 5 — Nebraska
1 — Wyoming 6 — Kansas
2 — Colorado 7 — Minnesota
3 — North Dakota 8 — Iowa
4 — South Dakota 9 — Missouri

The two classes are as follows and they specify the same thing as the last time

\[(x_1=0) \lor (x_2=0) \lor (x_1=1) \lor (x_2=1) \lor \ldots \lor (x_1=9) \lor (x_2=9)\]

and

\[(x_1=0) \lor (x_2=1, 2, 3, 4) \lor (x_1=1) \lor (x_2=0, 2, 4, 5) \lor (x_1=2) \lor (x_2=0, 1, 5, 6) \lor (x_1=3) \lor (x_2=0, 4, 7) \lor (x_1=4) \lor (x_2=0, 1, 3, 5, 7, 8) \lor (x_1=5) \lor (x_2=1, 2, 4, 6, 8, 9) \lor (x_1=6) \lor (x_2=2, 5, 9) \lor (x_1=7) \lor (x_2=3, 4, 8) \lor (x_1=8) \lor (x_2=4, 5, 7, 9) \lor (x_1=9) \lor (x_2=5, 6, 8)\]

It was run the same way as the last one and the result was as follows

\[(x_1=0, 5, 7) \lor (x_2=0, 5, 7) \lor (x_1=1, 3, 6, 8) \lor (x_2=1, 3, 6, 8) \lor (x_1=2, 4, 9) \lor (x_2=2, 4, 9)\]
As one can see, three states were used that were already colored so as to know which of the four colors used this set of states should be colored. One gets that states 0, 5, and 7 get color one since Montana was color one in the first run. States 1, 3, 6, and 8 get color two since Wyoming was color two and states 2, 4, and 9 get color three since Colorado was color three. So the result obtained is as shown in Figure 5.3(b). One follows this strategy until one has colored all the states. The progress is shown on Figure 5.3 (a)-(g), as each set was colored until the whole map was colored.
Figure 5.3(a) Coloring map of U.S.A.

Figure 5.3(b) (continued) Coloring map of U.S.A.
Figure 5.3(c) Coloring map of U.S.A.

Figure 5.3(d) (continued) Coloring map of U.S.A.
Figure 5.3(e) Coloring map of U.S.A.

Figure 5.3(f) (continued) Coloring map of U.S.A.
Figure 5.3(g)  Coloring map of U.S.A.
LIST OF REFERENCES


